An R Companion for Introduction to Data Mining

Michael Hahsler

2024-09-26

An R Companion for Introduction to Data Mining

by Michael Hahsler

Contents

Preface **9**

1 Introduction **11**
1.1 Used Software . . . 12
1.2 Base-R . . . 12
1.2.1 Vectors . . . 12
1.2.2 Vectorized Operations . . . 13
1.2.3 Subsetting Vectors . . . 13
1.2.4 Functions . . . 14
1.2.5 Strings . . . 14
1.2.6 Plotting . . . 15
1.2.7 Objects . . . 15
1.3 R Markdown . . . 16
1.4 Tidyverse . . . 16
1.4.1 Tibbles . . . 17
1.4.2 Transformations . . . 18
1.4.3 ggplot2 . . . 19

2 Data **23**
Packages Used in this Chapter . . . 23
2.1 Types of Data . . . 24
2.1.1 Attributes and Measurement . . . 24
2.1.2 The Iris Dataset . . . 25
2.2 Data Quality . . . 26
2.3 Data Preprocessing . . . 29
2.3.1 Aggregation . . . 29
2.3.2 Sampling . . . 29
2.3.3 Dimensionality Reduction . . . 32
2.3.4 Feature Subset Selection . . . 40
2.3.5 Discretization . . . 40
2.3.6 Variable Transformation: Standardization . . . 45
2.4 Measures of Similarity and Dissimilarity . . . 46
2.4.1 Minkowsky Distances . . . 47

2.4.2 Distances for Binary Data . . . 48
2.4.3 Distances for Mixed Data . . . 49
2.4.4 More Proximity Measures . . . 51
2.5 Data Exploration* . . . 52
2.5.1 Basic statistics . . . 53
2.5.2 Grouped Operations and Calculations . . . 54
2.5.3 Tabulate data . . . 56
2.5.4 Percentiles (Quantiles) . . . 58
2.5.5 Correlation . . . 59
2.5.6 Density . . . 64
2.6 Visualization . . . 69
2.6.1 Histogram . . . 69
2.6.2 Boxplot . . . 71
2.6.3 Scatter plot . . . 74
2.6.4 Scatter Plot Matrix . . . 77
2.6.5 Matrix Visualization . . . 79
2.6.6 Correlation Matrix . . . 82
2.6.7 Parallel Coordinates Plot . . . 85
2.6.8 Star Plot . . . 87
2.6.9 More Visualizations . . . 88
2.7 Exercises* . . . 88

3 Classification: Basic Concepts 91
Packages Used in this Chapter . . . 91
3.1 Basic Concepts . . . 92
3.2 General Framework for Classification . . . 93
3.2.1 The Zoo Dataset . . . 93
3.3 Decision Tree Classifiers . . . 95
3.3.1 Create Tree . . . 95
3.3.2 Make Predictions for New Data . . . 97
3.3.3 Manual Calculation of the Resubstitution Error . . . 98
3.3.4 Confusion Matrix using Caret . . . 99
3.4 Model Overfitting . . . 101
3.5 Model Selection . . . 103
3.6 Model Evaluation . . . 104
3.6.1 Holdout Method . . . 104
3.6.2 Cross-Validation . . . 104
3.7 Hyperparameter Tuning . . . 105
3.8 Pitfalls of Model Selection and Evaluation . . . 111
3.9 Model Comparison . . . 111
3.10 Feature Selection* . . . 114
3.10.1 Univariate Feature Importance Score . . . 114
3.10.2 Feature Subset Selection . . . 118
3.10.3 Using Dummy Variables for Factors . . . 119
3.11 Exercises* . . . 122

4 Classification: Alternative Techniques 125
Packages Used in this Chapter . . . 125
4.1 Types of Classifiers . . . 126
4.1.1 Set up the Training and Test Data . . . 126
4.2 Rule-based classifier: PART . . . 128
4.3 Nearest Neighbor Classifier . . . 129
4.4 Naive Bayes Classifier . . . 130
4.5 Bayesian Network . . . 132
4.6 Logistic regression . . . 132
4.7 Artificial Neural Network (ANN) . . . 134
4.8 Support Vector Machines . . . 136
4.9 Ensemble Methods . . . 137
4.9.1 Random Forest . . . 137
4.9.2 Gradient Boosted Decision Trees (xgboost) . . . 139
4.10 Class Imbalance . . . 140
4.10.1 Option 1: Use the Data As Is and Hope For The Best . . . 143
4.10.2 Option 2: Balance Data With Resampling . . . 145
4.10.3 Option 3: Build A Larger Tree and use Predicted Probabilities . . . 149
4.10.4 Option 4: Use a Cost-Sensitive Classifier . . . 153
4.11 Model Comparison . . . 156
4.12 Comparing Decision Boundaries of Popular Classification Techniques* . . . 161
4.12.1 Iris Dataset . . . 163
4.12.2 Circle Dataset . . . 179
4.13 More Information on Classification with R . . . 193
4.14 Exercises* . . . 193

5 Association Analysis: Basic Concepts 195
Packages Used in this Chapter . . . 195
5.1 Preliminaries . . . 196
5.1.1 The arules Package . . . 196
5.1.2 Transactions . . . 197
5.2 Frequent Itemset Generation . . . 210
5.3 Rule Generation . . . 215
5.3.1 Calculate Additional Interest Measures . . . 219
5.3.2 Mine Using Templates . . . 220
5.4 Compact Representation of Frequent Itemsets . . . 222
5.5 Association Rule Visualization* . . . 224
5.5.1 Static Visualizations . . . 224
5.5.2 Interactive Visualizations . . . 228
5.6 Exercises* . . . 231

6 Association Analysis: Advanced Concepts 233
Packages Used in this Chapter . . . 233
6.1 Handling Categorical Attributes . . . 233

6.2 Handling Continuous Attributes . . . 234
6.3 Handling Concept Hierarchies . . . 236
6.3.1 Aggregation . . . 236
6.3.2 Multi-level Analysis . . . 238
6.4 Sequential Patterns . . . 240

7 Cluster Analysis 245
Packages Used in this Chapter . . . 245
7.1 Overview . . . 246
7.1.1 Data Preparation . . . 246
7.1.2 Data cleaning . . . 247
7.1.3 Scale data . . . 248
7.2 K-means . . . 248
7.3 Agglomerative Hierarchical Clustering . . . 255
7.4 DBSCAN . . . 260
7.5 Cluster Evaluation . . . 263
7.5.1 Unsupervized Cluster Evaluation . . . 263
7.5.2 Unsupervised Cluster Evaluation using the Proximity Matrix . . . 268
7.5.3 Determining the Correct Number of Clusters . . . 274
7.5.4 Clustering Tendency . . . 278
7.5.5 Supervised Measures of Cluster Validity . . . 285
7.6 More Clustering Algorithms* . . . 293
7.6.1 Partitioning Around Medoids (PAM) . . . 293
7.6.2 Gaussian Mixture Models . . . 296
7.6.3 Spectral clustering . . . 298
7.6.4 Fuzzy C-Means Clustering . . . 300
7.7 Outliers in Clustering* . . . 303
7.7.1 Visual inspection of the data . . . 304
7.7.2 Local Outlier Factor (LOF) . . . 305
7.8 Exercises* . . . 308

8 Regression* 311
Packages Used in this Chapter . . . 311
8.1 Introduction . . . 311
8.2 A First Linear Regression Model . . . 312
8.3 Comparing Nested Models . . . 316
8.4 Stepwise Variable Selection . . . 319
8.5 Modeling with Interaction Terms . . . 320
8.6 Prediction . . . 322
8.7 Using Nominal Variables . . . 323
8.8 Alternative Regression Models . . . 326
8.8.1 Regression Trees . . . 326
8.8.2 Regularized Regression . . . 328
8.8.3 Other Types of Regression . . . 334
8.9 Exercises . . . 334

9 Logistic Regression* 337
Packages Used in this Chapter 337
9.1 Introduction 337
9.2 Data Preparation 338
9.3 Create a Logistic Regression Model 339
9.4 Stepwise Variable Selection 340
9.5 Calculate the Response 342
9.6 Check Classification Performance 343
9.7 Regularized Logistic Regression 345
9.8 Exercises 346

References 347

Preface

This companion book contains documented R examples to accompany several chapters of the popular data mining textbook *Introduction to Data Mining*[1] by Pang-Ning Tan, Michael Steinbach, Anuj Karpatne and Vipin Kumar. It is not intended as a replacement for the textbook since it does not cover the theory, but as a guide accompanying the textbook. The companion book can be used with either edition: 1st edition (Tan, Steinbach, and Kumar 2005) or 2nd edition (Tan et al. 2017). The sections are numbered to match the 2nd edition. Sections marked with an asterisk are additional content that is not covered in the textbook.

The code examples collected in this book were developed for the course *CS 5/7331 Data Mining* taught at the advanced undergraduate and graduate level at the Computer Science Department[2] at Southern Methodist University (SMU) since Spring 2013 and will be regularly updated and improved. The learning method used in this book is learning-by-doing. The code examples throughout this book are written in a self-contained manner so you can copy and paste a portion of the code, try it out on the provided dataset and then apply it directly to your own data. Instructors can use this companion as a component to create an introduction to data mining course for advanced undergraduates and graduate students who are proficient in programming and have basic statistics knowledge. The latest version of this book (html and PDF) with a complete set of lecture slides (PDF and PowerPoint) is provided on the book's GitHub page.[3]

The latest update includes the use of the popular packages in the meta-package `tidyverse` (Wickham 2023c) including `ggplot2` (Wickham et al. 2024) for data wrangling and visualization, along with `caret` (Kuhn 2023) for model building and evaluation. Please use the edit function within this book or visit the book's GitHub project page[4] to submit corrections or suggest improvements.

[1] https://www-users.cs.umn.edu/~kumar001/dmbook/
[2] https://www.smu.edu/lyle/departments/cs
[3] https://github.com/mhahsler/Introduction_to_Data_Mining_R_Examples
[4] https://github.com/mhahsler/Introduction_to_Data_Mining_R_Examples

To cite this book, use:

> Michael Hahsler (2024). *An R Companion for Introduction to Data Mining.* figshare. DOI: 10.6084/m9.figshare.26750404 URL: https://mhahsler.github.io/Introduction_to_Data_Mining_R_Examples/book/

I hope this book helps you to learn to use R more efficiently for your data mining projects.

Michael Hahsler

This book was built on Thu Sep 26 09:53:32 2024 (latest GitHub tag: 1.0.2)

[5] http://creativecommons.org/licenses/by-sa/4.0/
[6] https://www.flickr.com/photos/69017177@N00/5063131410
[7] https://www.flickr.com/photos/69017177@N00
[8] https://creativecommons.org/licenses/by/2.0/?ref=ccsearch&atype=rich

Chapter 1

Introduction

Data mining[1] has the goal of finding patterns in large data sets. The popular data mining textbook *Introduction to Data Mining*[2] (Tan et al. 2017) covers many important aspects of data mining. This companion contains annotated R code examples to complement the textbook. To make following along easier, we follow the chapters in data mining textbook which is organized by the main data mining tasks:

2. Data covers types of data and also includes data preparation and exploratory data analysis in the chapter.

3. Classification: Basic Concepts introduces the purpose of classification, basic classifiers using decision trees, and model training and evaluation.

4. Classification: Alternative Techniques introduces and compares methods including rule-based classifiers, nearest neighbor classifiers, naive Bayes classifier, logistic regression and artificial neural networks.

5. Association Analysis: Basic Concepts covers algorithms for frequent itemset and association rule generation and analysis including visualization.

6. Association Analysis: Advanced Concepts covers categorical and continuous attributes, concept hierarchies, and frequent sequence pattern mining.

7. Cluster Analysis discusses clustering approaches including k-means, hierarchical clustering, DBSCAN and how to evaluate clustering results.

For completeness, we have added chapters on Regression* and on Logistic Regression*. Sections in this book followed by an asterisk contain code examples for methods that are not described in the data mining textbook.

[1] https://en.wikipedia.org/wiki/Data_mining
[2] https://www-users.cs.umn.edu/~kumar001/dmbook/

This book assumes that you are familiar with the basics of R, how to run R code, and install packages. The rest of this chapter will provide an overview and point you to where you can learn more about R and the used packages.

1.1 Used Software

To use this book you need to have R[3] and RStudio Desktop[4] installed.

Each book chapter will use a set of packages that must be installed. The installation code can be found at the beginning of each chapter. Here is the code to install the packages used in this chapter:

```
pkgs <- c('tidyverse')

pkgs_install <- pkgs[!(pkgs %in% installed.packages()[,"Package"])]
if(length(pkgs_install)) install.packages(pkgs_install)
```

The code examples in this book use the R package collection `tidyverse` (Wickham 2023c) to manipulate data. Tidyverse also includes the package `ggplot2` (Wickham et al. 2024) for visualization. Tidyverse packages make working with data in R very convenient. Data analysis and data mining reports are typically done by creating R Markdown documents. Everything in R is built on top of the core R programming language and the packages that are automatically installed with R. This is referred to as Base-R.

1.2 Base-R

Base-R is covered in detail in An Introduction to R[5]. The most important differences between R and many other programming languages like Python are:

- R is a functional programming language (with some extensions).
- R only uses vectors and operations are vectorized. You rarely will see loops.
- R starts indexing into vectors with 1 not 0.
- R uses `<-` for assignment. Do not use `=` for assignment.

1.2.1 Vectors

The basic data structure in R is a vector of real numbers. Scalars do not exist, they are just vectors of length 1.

We can combine values into a vector using the combine function `c()`. Special values for infinity (`Inf`) and missing values (`NA`) can be used.

[3] https://cran.r-project.org/
[4] https://www.rstudio.com/products/rstudio/
[5] https://cran.r-project.org/doc/manuals/r-release/R-intro.html

```
x <- c(10.4, 5.6, Inf, NA, 21.7)
x
## [1] 10.4  5.6  Inf   NA 21.7
```

We often use sequences. A simple sequence of integers can be produced using the colon operator in the form `from:to`.

```
3:10
## [1]  3  4  5  6  7  8  9 10
```

More complicated sequences can be created using `seq()` or one of the related functions starting with `seq_`.

```
y <- seq(from = 0, to = 1, length.out = 5)
y
## [1] 0.00 0.25 0.50 0.75 1.00
```

1.2.2 Vectorized Operations

Operations are vectorized and applied to each element, so loops are typically not necessary.

```
x + 1
## [1] 11.4  6.6  Inf   NA 22.7
```

Comparisons are also vectorized are performed element-wise. They return a `logical` vector (R's name for the datatype Boolean).

```
x > y
## [1] TRUE TRUE TRUE   NA TRUE
```

1.2.3 Subsetting Vectors

We can select vector elements using the `[` operator like in other programming languages but the index always starts at 1.

We can select elements 1 through 3 using an index sequence.

```
x[1:3]
## [1] 10.4  5.6  Inf
```

Negative indices remove elements. We can select al but the first element.

```
x[-1]
## [1]  5.6  Inf   NA 21.7
```

We can use any function that creates a `logical` vector for subsetting. Here we select all non-missing values using the function `is.na()`.

```
is.na(x)
## [1] FALSE FALSE FALSE  TRUE FALSE
```

```
x[!is.na(x)] # select all non-missing values
## [1] 10.4  5.6  Inf 21.7
```

We can also assign values to a selection. For example, this code gets rid of infinite values.

```
x[!is.finite(x)] <- NA
x
## [1] 10.4  5.6   NA   NA 21.7
```

1.2.4 Functions

Functions work like in many other languages. However, they also operate vectorized and arguments can be specified positional or by named. An increment function can be defined as:

```
inc <- function(x, by = 1) {
    x + by
  }
```

The value of the last evaluated expression in the function body is automatically returned by the function. The function can also explicitly use `return(return_value)`.

Calling the increment function on vector `x`.

```
inc(x, by = 2)
## [1] 12.4  7.6   NA   NA 23.7
```

Before you implement your own function, check if R does not already provide an implementation. R has many built-in functions like `min()`, `max()`, `mean()`, and `sum()`.

1.2.5 Strings

R uses `character` vectors where the datatype `character` represents a string and not like in other programming language for a single character. R accepts double or single quotation marks to delimit strings.

```
string <- c("Hello", "Goodbye")
string
## [1] "Hello"   "Goodbye"
```

Strings can be combined using `paste()`.

```
paste(string, "World!")
## [1] "Hello World!"   "Goodbye World!"
```

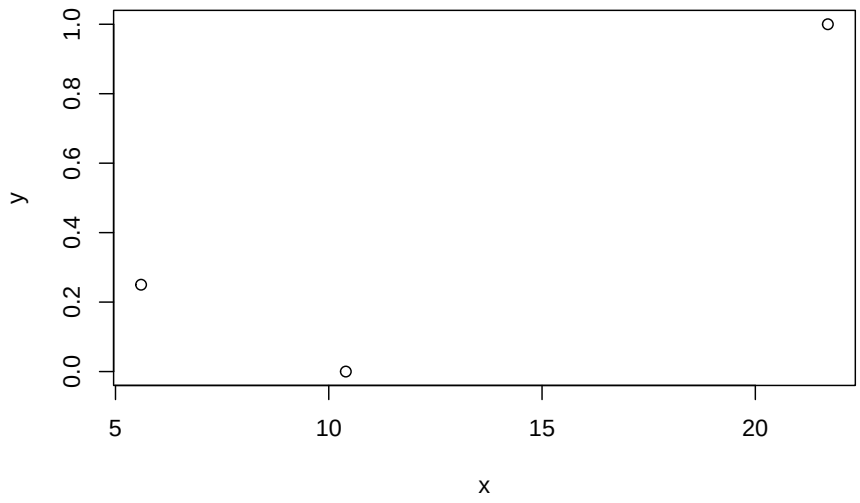
Note that paste is vectorized and the string `"World!"` is used twice to match the length of `string`. This behavior is called in R *recycling* and works if one

vector's length is an exact multiple of the other vector's length. The special case where one vector has one element is particularly often used. We have used it already above in the expression `x + 1` where one is recycled for each element in `x`.

1.2.6 Plotting

Basic plotting in Base-R is done by calling `plot()`.

```
plot(x, y)
```



Other plot functions are `pairs()`, `hist()`, `barplot()`. In this book, we will focus on plots created with the ggplot2 package.

1.2.7 Objects

Other often used data structures include `list`, `data.frame`, `matrix`, and `factor`. In R, everything is an object. Objects are printed by either just using the object's name or put it explicitly into the `print()` function.

```
x
## [1] 10.4  5.6   NA   NA 21.7
```

For many objects, a summary can be created.

```
summary(x)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##     5.6     8.0    10.4    12.6    16.1    21.7       2
```

Objects have a class.

```
class(x)
## [1] "numeric"
```

Some functions are generic, meaning that they do something else depending on the class of the first argument. For example, `plot()` has many methods implemented to visualize data of different classes. The specific function is specified with a period after the method name. There also exists a default method. For plot, this method is `plot.default()`. Different methods may have their own manual page, so specifying the class with the dot can help finding the documentation.

It is often useful to know what information it stored in an object. `str()` returns a humanly readable string representation of the object.

```
str(x)
##  num [1:5] 10.4 5.6 NA NA 21.7
```

There is much to learn about R. It is highly recommended to go through the official An Introduction to R[6] manual. There is also a good Base R Cheat Sheet[7] available.

1.3 R Markdown

R Markdown is a simple method to include R code inside a text document written using markdown syntax. Using R markdown is especially convention to analyze data and compose a data mining report. RStudio makes creating and translating R Markdown document easy. Just choose `File -> New File -> R Markdown...` RStudio will create a small demo markdown document that already includes examples for code and how to include plots. You can switch to visual mode if you prefer a What You See Is What You Get editor.

To convert the R markdown document to HTML, PDF or Word, just click the Knit button. All the code will be executed and combined with the text into a complete document.

Examples and detailed documentation can be found on RStudio's R Markdown website[8] and in the R Markdown Cheatsheet[9].

1.4 Tidyverse

`tidyverse` (Wickham 2023c) is a collection of many useful packages that work well together by sharing design principles and data structures. `tidyverse` also

[6] https://cran.r-project.org/doc/manuals/r-release/R-intro.html
[7] https://github.com/rstudio/cheatsheets/blob/main/base-r.pdf
[8] https://rmarkdown.rstudio.com/
[9] https://rstudio.github.io/cheatsheets/html/rmarkdown.html

includes `ggplot2` (Wickham et al. 2024) for visualization.

In this book, we will use

- often tidyverse tibbles to replace R's built-in data.frames,
- the pipe operator `|>` to chain functions together, and
- data transformation functions like `filter()`, `arrange()`, `select()`, `group_by()`, and `mutate()` provided by the tidyverse package `dplyr` (Wickham et al. 2023).

A good introduction can be found in the Section on Data Transformation[10] (Wickham, Çetinkaya-Rundel, and Grolemund 2023).

Load the tidyverse packages.

```
library(tidyverse)
## -- Attaching core tidyverse packages ---- tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ---------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

1.4.1 Tibbles

Here is a short example That analyses the vitamin C content of different fruits. that will get you familiar with the basic syntax. Data tables in R are called data.frames. Tidyverse introduces its own version called tibbles. We create a tibble with the price in dollars per pound and the vitamin C content in milligrams (mg) per pound for five different fruits.

```
fruit <- tibble(
  name = c("apple", "banana", "mango", "orange", "lime"),
  price = c(2.5, 2.0, 4.0, 3.5, 2.5),
  vitamin_c = c(20, 45, 130, 250, 132),
  type = c("pome", "tropical", "tropical", "citrus", "citrus"))
fruit
## # A tibble: 5 x 4
##   name   price vitamin_c type
##   <chr>  <dbl>     <dbl> <chr>
## 1 apple    2.5        20 pome
## 2 banana   2          45 tropical
## 3 mango    4         130 tropical
```

[10] https://r4ds.hadley.nz/data-transform

```
## 4 orange   3.5       250 citrus
## 5 lime     2.5       132 citrus
```

1.4.2 Transformations

We can modify the table by adding a column with the vitamin C (in mg) that a dollar buys using `mutate()`. Then we filter only rows with fruit that provides more than 20 mg, and finally we arrange the data rows by the vitamin C per dollar from largest to smallest.

```
affordable_vitamin_c_sources <- fruit |>
  mutate(vitamin_c_per_dollar = vitamin_c / price) |>
  filter(vitamin_c_per_dollar > 20) |>
  arrange(desc(vitamin_c_per_dollar))

affordable_vitamin_c_sources
## # A tibble: 4 x 5
##   name   price vitamin_c type     vitamin_c_per_dollar
##   <chr>  <dbl>     <dbl> <chr>                   <dbl>
## 1 orange   3.5       250 citrus                   71.4
## 2 lime     2.5       132 citrus                   52.8
## 3 mango    4         130 tropical                 32.5
## 4 banana   2          45 tropical                 22.5
```

The pipes operator `|>` lets you pass the value to the left (often the result of a function) as the first argument to the function on the right. This makes composing a sequence of function calls to transform data much easier to write and read. You will often see `%>%` as the pipe operator, especially in examples using tidyverse. Both operators work similarly where `|>` is a native R operator while `%>%` is provided in the extension package `magrittr`.

The code above starts with the fruit data and pipes it through three transformation functions. The final result is assigned with `<-` to a variable.

We can create summary statistics of price using `summarize()`.

```
affordable_vitamin_c_sources |>
  summarize(min = min(price),
            mean = mean(price),
            max = max(price))
## # A tibble: 1 x 3
##     min  mean   max
##   <dbl> <dbl> <dbl>
## 1     2     3     4
```

We can also calculate statistics for groups by first grouping the data wirh `group_by()`. Here we produce statistics for fruit types.

```
affordable_vitamin_c_sources |>
  group_by(type) |>
  summarize(min = min(price),
            mean = mean(price),
            max = max(price))
## # A tibble: 2 x 4
##   type       min  mean   max
##   <chr>    <dbl> <dbl> <dbl>
## 1 citrus     2.5     3   3.5
## 2 tropical   2       3   4
```

Often, we want to apply the same function to multiple columns. This can be achieved using `across(columns, function)`.

```
affordable_vitamin_c_sources |>
  summarize(across(c(price, vitamin_c), mean))
## # A tibble: 1 x 2
##   price vitamin_c
##   <dbl>     <dbl>
## 1     3      139.
```

`dplyr` syntax and evaluation is slightly different from standard R which may lead to confusion. One example is that column names can be references without quotation marks. A very useful reference resource when working with `dplyr` is the RStudio Data Transformation Cheatsheet[11] which covers on two pages almost everything you will need and also contains contains simple example code that you can take and modify for your use case.

1.4.3 ggplot2

For visualization, we will use mainly `ggplot2`. The *gg* in `ggplot2` stands for **Grammar of Graphics** introduced by Wilkinson (2005). The main idea is that every graph is built from the same basic components:

- the data,
- a coordinate system, and
- visual marks representing the data (geoms).

In `ggplot2`, the components are combined using the `+` operator.

```
ggplot(data, mapping = aes(x = ..., y = ..., color = ...)) +
  geom_point()
```

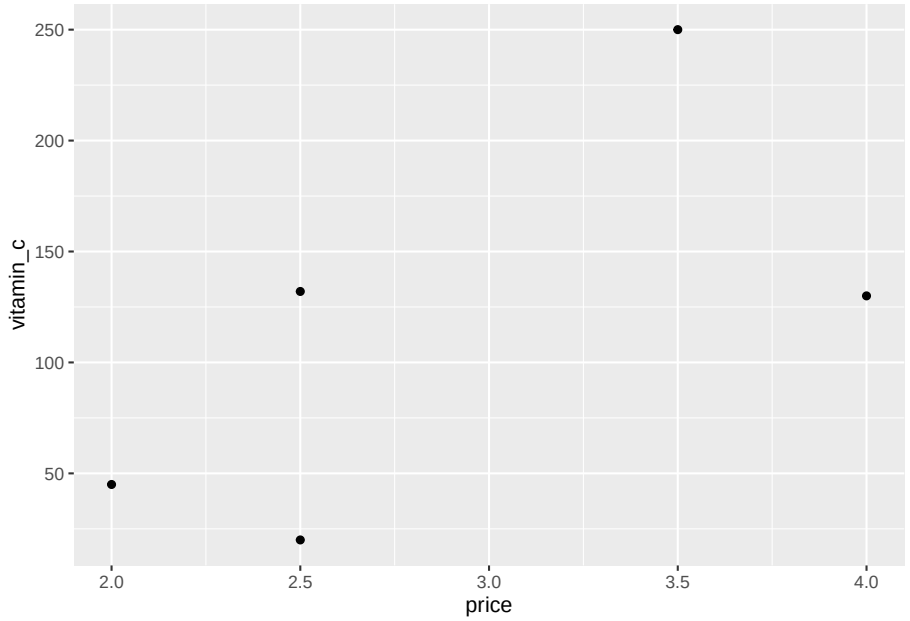
Since we typically use a Cartesian coordinate system, `ggplot` uses that by default. Each `geom_` function uses a `stat_` function to calculate what is visualizes. For example, `geom_bar` uses `stat_count` to create a bar chart by counting how often each value appears in the data (see `? geom_bar`). `geom_point` just uses

[11] https://rstudio.github.io/cheatsheets/html/data-transformation.html

the stat `"identity"` to display the points using the coordinates as they are. A great introduction can be found in the Section on Data Visualization[12] (Wickham, Çetinkaya-Rundel, and Grolemund 2023), and very useful is RStudio's Data Visualization Cheatsheet[13].

We can visualize our fruit data as a scatter plot.

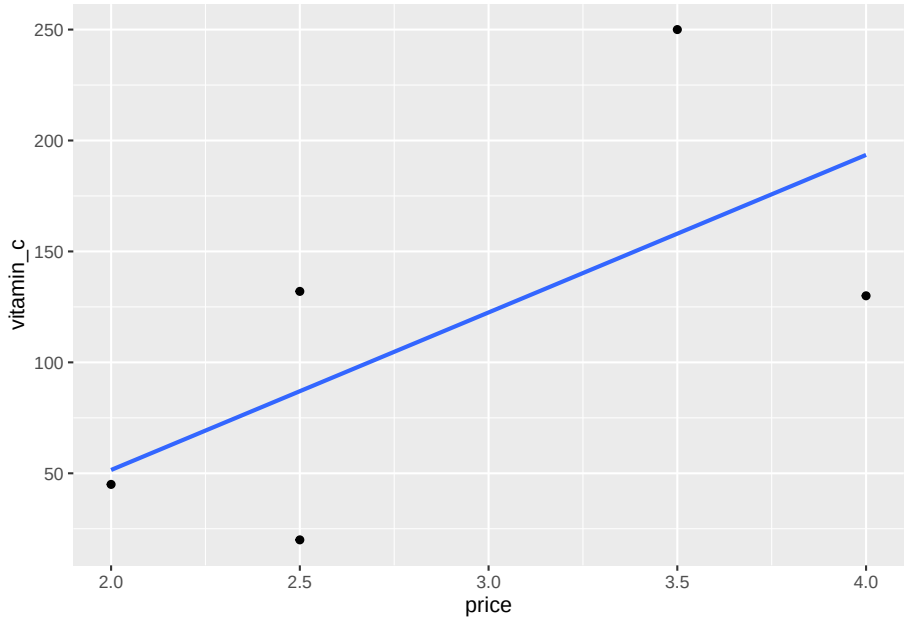
```
ggplot(fruit, aes(x = price, y = vitamin_c)) +
  geom_point()
```



It is easy to add more geoms. For example, we can add a regression line using `geom_smooth` with the method `"lm"` (linear model). We suppress the confidence interval since we only have 3 data points.

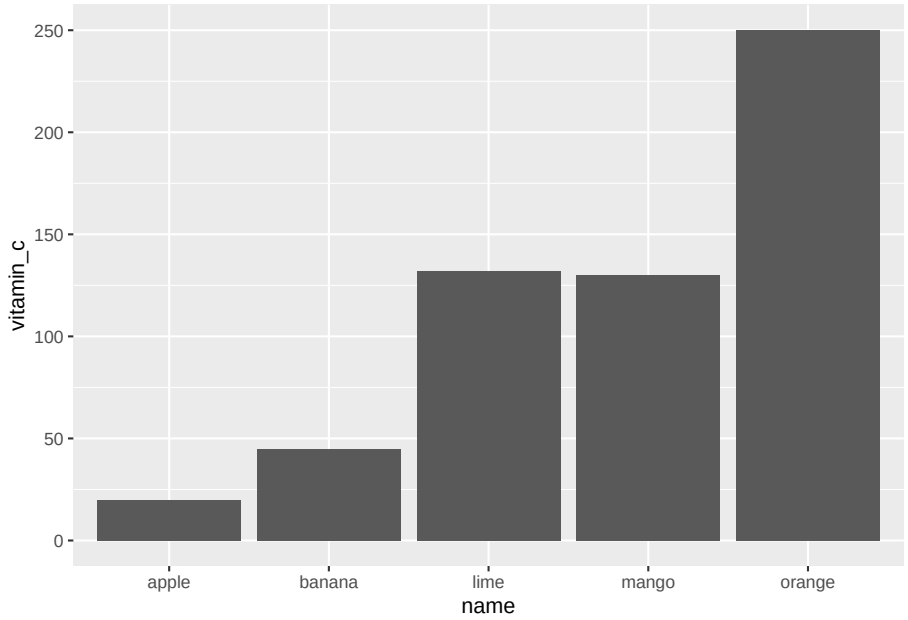
```
ggplot(fruit, aes(x = price, y = vitamin_c)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
## `geom_smooth()` using formula = 'y ~ x'
```

[12] https://r4ds.hadley.nz/data-visualize
[13] https://rstudio.github.io/cheatsheets/html/data-visualization.html



Alternatively, we can visualize each fruit's vitamin C content per dollar using a bar chart.

```
ggplot(fruit, aes(x = name, y = vitamin_c)) +
  geom_bar(stat = "identity")
```



Note that `geom_bar` by default uses the `stat_count` function to aggregate data by counting, but we just want to visualize the value in the tibble, so we specify the identity function instead.

Chapter 2

Data

Data for data mining is typically organized in tabular form, with rows containing the objects of interest and columns representing attributes describing the objects. We will discuss topics like data quality, sampling, feature selection, and how to measure similarities between objects and features. The second part of this chapter deals with data exploration and visualization.

Packages Used in this Chapter

```
pkgs <- c("arules", "caret", "factoextra", "GGally",
          "ggcorrplot", "hexbin", "palmerpenguins", "plotly",
          "proxy", "seriation", "tidyverse")

pkgs_install <- pkgs[!(pkgs %in% installed.packages()[,"Package"])]
if(length(pkgs_install)) install.packages(pkgs_install)
```

The packages used for this chapter are:

- *arules* (Hahsler et al. 2024)
- *caret* (Kuhn 2023)
- *factoextra* (Kassambara and Mundt 2020)
- *GGally* (Schloerke et al. 2024)
- *ggcorrplot* (Kassambara 2023)
- *hexbin* (Carr, Lewin-Koh, and Maechler 2024)
- *palmerpenguins* (Horst, Hill, and Gorman 2022)
- *plotly* (Sievert et al. 2024)
- *proxy* (Meyer and Buchta 2022)
- *seriation* (Hahsler, Buchta, and Hornik 2024)
- *tidyverse* (Wickham 2023c)

2.1 Types of Data

2.1.1 Attributes and Measurement

The values of features can be measured on several scales[1] ranging from simple labels all the way to numbers. The scales come in four levels.

Scale Name	Description	Operations	Statistics	R
Nominal	just a label (e.g., red, green)	$==, !=$	counts	`factor`
Ordinal	label with order (e.g., small, med., large)	$<, >$	median	`ordered factor`
Interval	difference between two values is meaningful (regular number)	$+, -$	mean, sd	`numeric`
Ratio	has a natural zero (e.g., count, distance)	$/, *$	percent	`numeric`

The scales build on each other meaning that an ordinal variable also has the characteristics of a nominal variable with the added order information. We often do not differentiate between interval and ratio scale because we rarely not need to calculate percentages or other statistics that require a meaningful zero value.

Nominal data is created using `factor()`. If the factor levels are not specified, then they are created in alphabetical order.

```
factor(c("red", "green", "green", "blue"))
## [1] red   green green blue
## Levels: blue green red
```

Ordinal data is created using `ordered()`. The levels specify the order.

```
ordered(c("S", "L", "M", "S"),
        levels = c("S", "M", "L"))
## [1] S L M S
## Levels: S < M < L
```

Ratio/interval data is created as a simple vector.

```
c(1, 2, 3, 4, 3, 3)
## [1] 1 2 3 4 3 3
```

[1] https://en.wikipedia.org/wiki/Level_of_measurement

2.1.2 The Iris Dataset

We will use a toy dataset that comes with R. Fisher's iris dataset[2] gives the measurements in centimeters of the variables sepal length, sepal width petal length, and petal width representing the features for 150 flowers (the objects). The dataset contains 50 flowers from each of 3 species of iris. The species are Iris Setosa, Iris Versicolor, and Iris Virginica. For more details see `? iris`.

We load the iris data set. Datasets that come with R or R packages can be loaded with `data()`. The standard format for data in R is a data.frame. We convert the data.frame into a tidyverse tibble.

```
library(tidyverse)
data(iris)
iris <- as_tibble(iris)
iris
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>
##  1          5.1         3.5          1.4         0.2 setosa
##  2          4.9         3            1.4         0.2 setosa
##  3          4.7         3.2          1.3         0.2 setosa
##  4          4.6         3.1          1.5         0.2 setosa
##  5          5           3.6          1.4         0.2 setosa
##  6          5.4         3.9          1.7         0.4 setosa
##  7          4.6         3.4          1.4         0.3 setosa
##  8          5           3.4          1.5         0.2 setosa
##  9          4.4         2.9          1.4         0.2 setosa
## 10          4.9         3.1          1.5         0.1 setosa
## # i 140 more rows
```

We see that the data contains 150 rows (flowers) and 5 features. tibbles only show the first few rows and do not show all features, if they do not fit the screen width. We can call `print` and define how many rows to show using parameter `n` and force print to show all features by changing the `width` to infinity.

```
print(iris, n = 3, width = Inf)
## # A tibble: 150 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>
## 1          5.1         3.5          1.4         0.2 setosa
## 2          4.9         3            1.4         0.2 setosa
## 3          4.7         3.2          1.3         0.2 setosa
## # i 147 more rows
```

[2] https://en.wikipedia.org/wiki/Iris_flower_data_set

2.2 Data Quality

Assessing the quality of the available data is crucial before we start using the data. Start with summary statistics for each column to identify outliers and missing values. The easiest way is to use the base R function `summary()`.

```
summary(iris)
##   Sepal.Length   Sepal.Width    Petal.Length   Petal.Width
##  Min.   :4.30   Min.   :2.00   Min.   :1.00   Min.   :0.1
##  1st Qu.:5.10   1st Qu.:2.80   1st Qu.:1.60   1st Qu.:0.3
##  Median :5.80   Median :3.00   Median :4.35   Median :1.3
##  Mean   :5.84   Mean   :3.06   Mean   :3.76   Mean   :1.2
##  3rd Qu.:6.40   3rd Qu.:3.30   3rd Qu.:5.10   3rd Qu.:1.8
##  Max.   :7.90   Max.   :4.40   Max.   :6.90   Max.   :2.5
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

You can also summarize individual columns using tidyverse's dplyr functions.

```
iris |>
  summarize(mean = mean(Sepal.Length))
## # A tibble: 1 x 1
##    mean
##   <dbl>
## 1  5.84
```

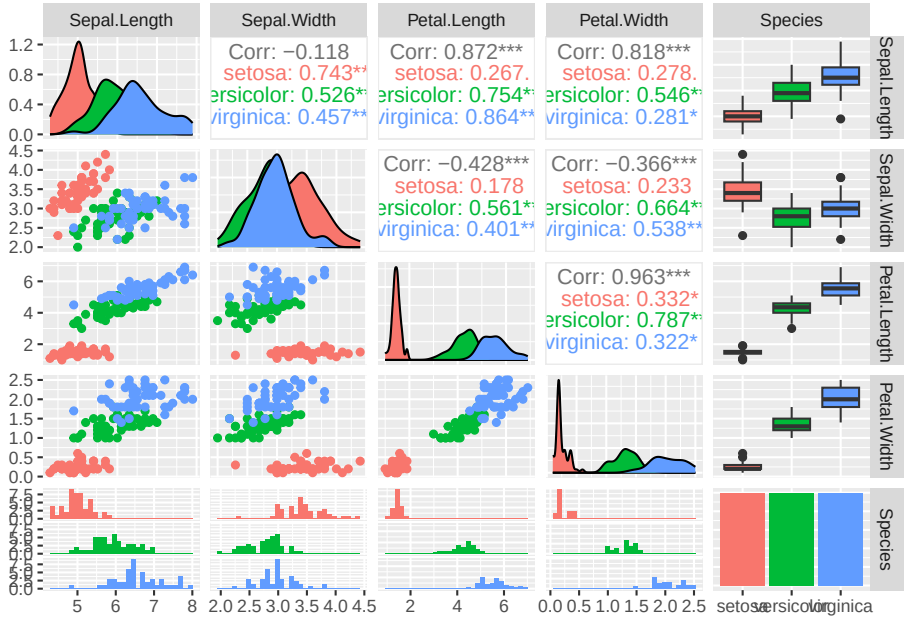
Using `across()`, multiple columns can be summarized. Un the following, we calculate all numeric columns using the `mean` function.

```
iris |>
  summarize(across(where(is.numeric), mean))
## # A tibble: 1 x 4
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
##          <dbl>       <dbl>        <dbl>       <dbl>
## 1         5.84        3.06         3.76        1.20
```

To find outliers or data problems, you need to look for very small values (often a suspicious large number of zeros) using min and for extremely large values using max. Comparing median and mean tells us if the distribution is symmetric.

A visual method to inspect the data is to use a scatterplot matrix (we use here `ggpairs()` from package `GGally`). In this plot, we can visually identify noise data points and outliers (points that are far from the majority of other points).

```
library(GGally)
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
ggpairs(iris, aes(color = Species), progress = FALSE)
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
```



This useful visualization combines many visualizations used to understand the data and check for quality issues. Rows and columns are the features in the data. We have also specified the aesthetic that we want to group each species using a different color.

- The visualizations in the diagonal panels show the smoothed histograms with the distribution for each feature. The plot tries to pick a good number of bins for the histogram (see messages above). The distribution can be checked if it is close to normal, unimodal or highly skewed. Also, we can see if the different groups are overlapping or separable for each feature. For example, the three distributions for `Sepal.Width` are almost identical

meaning that it is hard to distinguish between the different species using this feature alone. `Petal.Lenght` and `Petal.Width` are much better.

- The lower-left triangle panels contain scatterplots for all pairs features. These are useful to see if there if features are correlated (the pearson correlation coefficient if printed in the upper-right triangle). For example, `Petal.Length` and `Petal.Width` are highly correlated overall This makes sense since larger plants will have both longer and wider petals. Inside the Setosa group this correlation it is a lot weaker. We can also see if groups are well separated using projections on two variables. Almost all panels show that Setosa forms a point cloud well separated from the other two classes while Versicolor and Virginica overlap. We can also see outliers that are far from the other data points in its group. See if you can spot the one red dot that is far away from all others.

- The last row/column represents in this data set the class label Species. It is a nominal variable so the plots are different. The bottom row panels show (regular) histograms. The last column shows boxplots to represent the distribution of the different features by group. Dots represent outliers. Finally, the bottom-right panel contains the counts for the different groups as a barplot. In this data set, each group has the same number of observations.

Many data mining methods require complete data, that is the data cannot contain missing values (`NA`). To remove missing values and duplicates (identical data points which might be a mistake in the data), we often do this:

```
clean.data <- iris |>
  drop_na() |>
  unique()

summary(clean.data)
##   Sepal.Length   Sepal.Width    Petal.Length
##  Min.   :4.30   Min.   :2.00   Min.   :1.00
##  1st Qu.:5.10   1st Qu.:2.80   1st Qu.:1.60
##  Median :5.80   Median :3.00   Median :4.30
##  Mean   :5.84   Mean   :3.06   Mean   :3.75
##  3rd Qu.:6.40   3rd Qu.:3.30   3rd Qu.:5.10
##  Max.   :7.90   Max.   :4.40   Max.   :6.90
##   Petal.Width          Species
##  Min.   :0.10   setosa    :50
##  1st Qu.:0.30   versicolor:50
##  Median :1.30   virginica :49
##  Mean   :1.19
##  3rd Qu.:1.80
##  Max.   :2.50
```

Note that one non-unique case is gone leaving only 149 flowers. The data did

not contain missing values, but if it did, they would also have been dropped. Typically, you should spend a lot more time on data cleaning.

2.3 Data Preprocessing

2.3.1 Aggregation

Data often contains groups and we want to compare these groups. We group the iris dataset by species and then calculate a summary statistic for each group.

```
iris |>
  group_by(Species) |>
  summarize(across(everything(), mean))
## # A tibble: 3 x 5
##   Species  Sepal.Length Sepal.Width Petal.Length Petal.Width
##   <fct>           <dbl>       <dbl>        <dbl>       <dbl>
## 1 setosa           5.01        3.43         1.46       0.246
## 2 versico~         5.94        2.77         4.26       1.33
## 3 virgini~         6.59        2.97         5.55       2.03
iris |>
  group_by(Species) |>
  summarize(across(everything(), median))
## # A tibble: 3 x 5
##   Species  Sepal.Length Sepal.Width Petal.Length Petal.Width
##   <fct>           <dbl>       <dbl>        <dbl>       <dbl>
## 1 setosa            5           3.4          1.5         0.2
## 2 versico~          5.9         2.8          4.35        1.3
## 3 virgini~          6.5         3            5.55        2
```

Using this information, we can compare how features differ between groups.

2.3.2 Sampling

Sampling[3] is often used in data mining to reduce the dataset size before modeling or visualization.

2.3.2.1 Random Sampling

The built-in sample function can sample from a vector. Here we sample with replacement.

```
sample(c("A", "B", "C"), size = 10, replace = TRUE)
##  [1] "C" "A" "C" "C" "B" "C" "C" "C" "B" "A"
```

We often want to sample rows from a dataset. This can be done by sampling without replacement from a vector with row indices (using the functions `seq()`

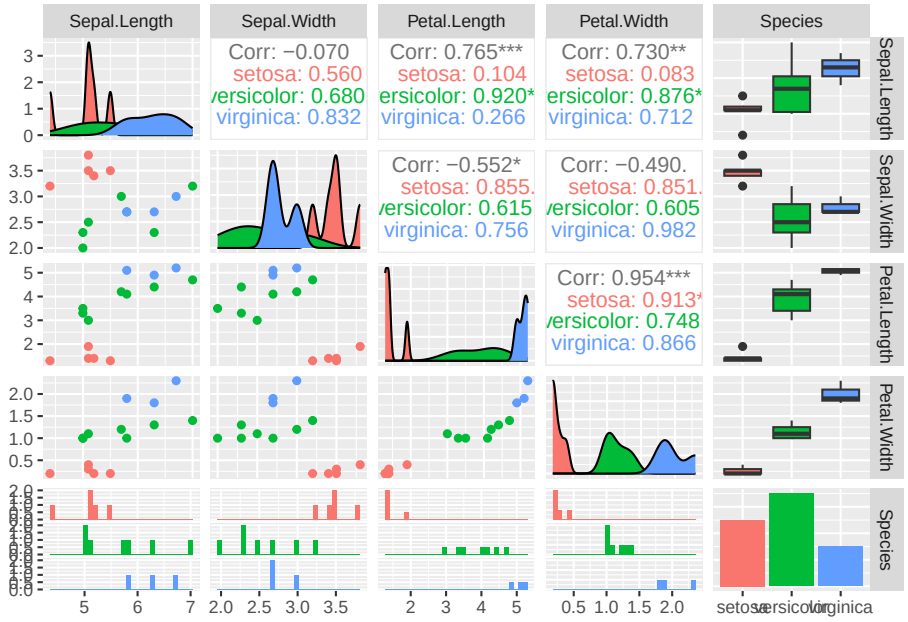
[3] https://en.wikipedia.org/wiki/Sampling_(statistics)

and `nrow()`). The sample vector is then used to subset the rows of the dataset.

```
take <- sample(seq(nrow(iris)), size = 15)
take
##  [1]  94  92 110 122 145 105  50  87  43  91  63  15 134  99
## [15] 100
iris[take, ]
## # A tibble: 15 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>
##  1          5           2.3          3.3         1   versic~
##  2          6.1         3            4.6         1.4 versic~
##  3          7.2         3.6          6.1         2.5 virgin~
##  4          5.6         2.8          4.9         2   virgin~
##  5          6.7         3.3          5.7         2.5 virgin~
##  6          6.5         3            5.8         2.2 virgin~
##  7          5           3.3          1.4         0.2 setosa
##  8          6.7         3.1          4.7         1.5 versic~
##  9          4.4         3.2          1.3         0.2 setosa
## 10          5.5         2.6          4.4         1.2 versic~
## 11          6           2.2          4           1   versic~
## 12          5.8         4            1.2         0.2 setosa
## 13          6.3         2.8          5.1         1.5 virgin~
## 14          5.1         2.5          3           1.1 versic~
## 15          5.7         2.8          4.1         1.3 versic~
```

`dplyr` from tidyverse lets us sample rows from tibbles directly using `slice_sample()`. I set the random number generator seed to make the results reproducible.

```
set.seed(1000)
s <- iris |>
  slice_sample(n = 15)

library(GGally)
ggpairs(s, aes(color = Species), progress = FALSE)
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
```



Instead of `n` you can also specify the proportion of rows to select using `prob`.

2.3.2.2 Stratified Sampling

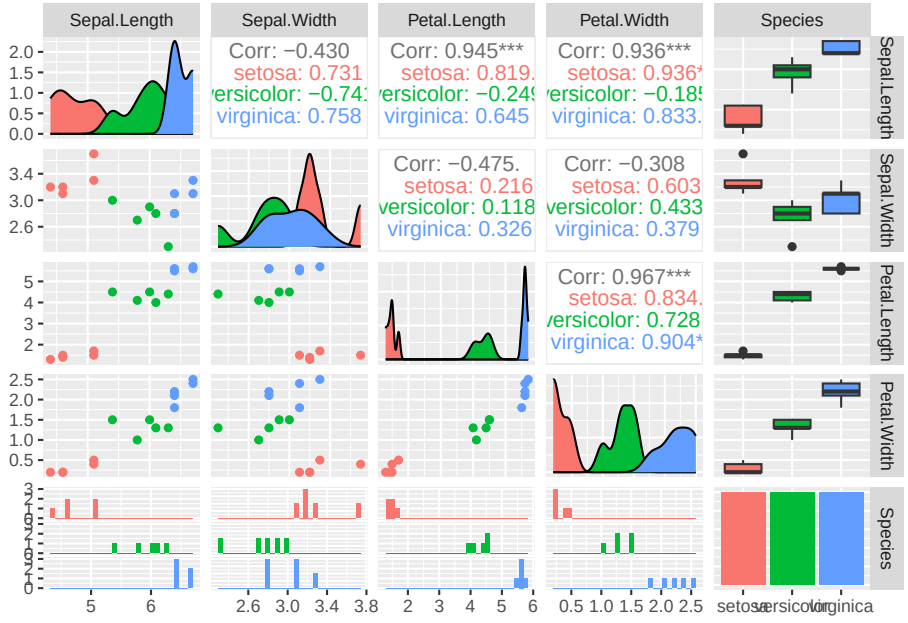
Stratified sampling[4] is a method of sampling from a population which can be partitioned into subpopulations, while controlling the proportions of the subpopulation in the resulting sample.

In the following, the subpopulations are the different types of species and we want to make sure to sample the same number (5) flowers from each. This can be achieved by first grouping the data by species and then sampling a number of flowers from each group.

```
set.seed(1000)

s2 <- iris |>
  group_by(Species) |>
  slice_sample(n = 5) |>
  ungroup()

library(GGally)
ggpairs(s2, aes(color = Species), progress = FALSE)
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with
```

[4] https://en.wikipedia.org/wiki/Stratified_sampling

```
## `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
```



More sophisticated sampling procedures are implemented in the package `sampling`.

2.3.3 Dimensionality Reduction

The number of features is often called the dimensional of the data following the idea that each feature (at least the numeric features) can be seen as an axis of the data. High-dimensional data is harder to analyze by the user (e.g., visualize). It also is problematic for many data mining algorithms since it requires more memory and computational resources.

Dimensionality reduction[5] tries to represent high-dimensional data in a low-dimensional space so that the low-dimensional representation retains some meaningful properties (e.g., information about similarity or distances) of the original data. Dimensionality reduction is used for visualization and as a prepossessing technique before using data mining models.

A special case of dimensionality reduction is feature selection (see Feature Selection* in Chapter 3).

[5] https://en.wikipedia.org/wiki/Dimensionality_reduction

2.3.3.1 Principal Components Analysis (PCA)

PCA[6] calculates principal components (a set of new orthonormal basis vectors in the data space) from data points such that the first principal component explains the most variability in the data, the second the next most and so on. In data analysis, PCA is used to project high-dimensional data points onto the first few (typically two) principal components for visualization as a scatter plot and as preprocessing for modeling (e.g., before k-means clustering). Points that are closer together in the high-dimensional original space, tend also be closer together when projected into the lower-dimensional space,

We can use an interactive 3-d plot (from package `plotly`) to look at three of the four dimensions of the iris dataset. Note that it is hard to visualize more than 3 dimensions.

```
plotly::plot_ly(iris,
                x = ~Sepal.Length,
                y = ~Petal.Length,
                z = ~Sepal.Width,
      color = ~Species, size = 1) |>
  plotly::add_markers()
```

The resulting interactive plot can be seen in the online version of this book.[7]

The principal components can be calculated from a matrix using the function `prcomp()`. We select all numeric columns (by unselecting the species column) and convert the tibble into a matrix before the calculation.

```
pc <- iris |>
  select(-Species) |>
  as.matrix() |>
  prcomp()
summary(pc)
## Importance of components:
##                           PC1    PC2    PC3     PC4
## Standard deviation     2.056 0.4926 0.2797 0.15439
## Proportion of Variance 0.925 0.0531 0.0171 0.00521
## Cumulative Proportion  0.925 0.9777 0.9948 1.00000
```

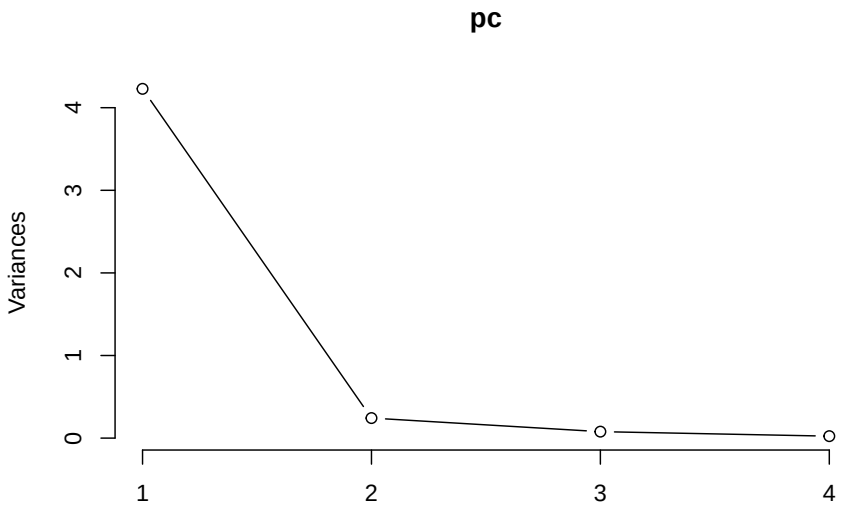
How important is each principal component can also be seen using a scree plot[8]. The plot function for the result of the `prcomp` function visualizes how much variability in the data is explained by each additional principal component.

```
plot(pc, type = "line")
```

[6] https://en.wikipedia.org/wiki/Principal_component_analysis

[7] https://mhahsler.github.io/Introduction_to_Data_Mining_R_Examples/book/data.html#dimensionality-reduction

[8] https://en.wikipedia.org/wiki/Scree_plot



Note that the first principal component (PC1) explains most of the variability in the iris dataset.

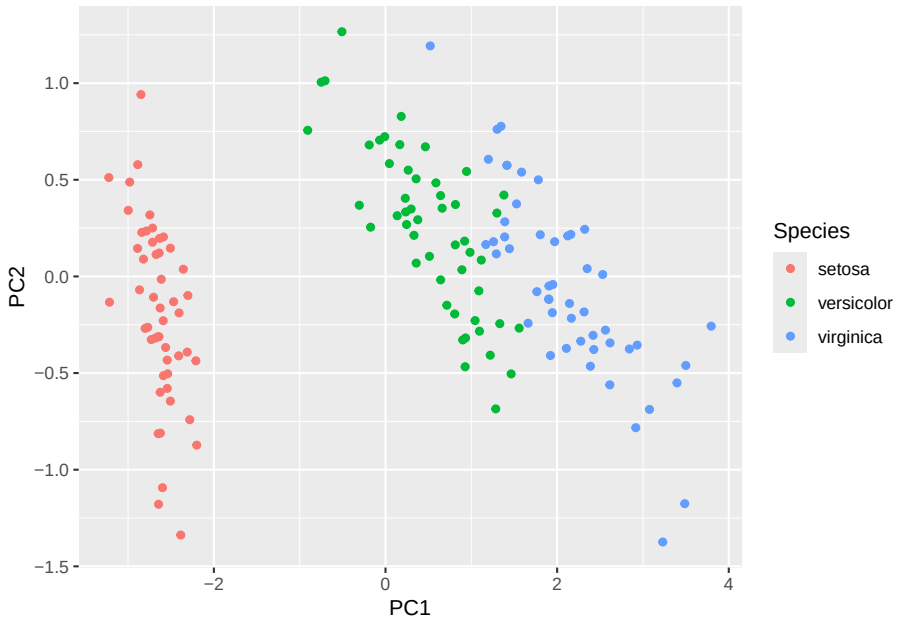
To find out what information is stored in the object `pc`, we can inspect the raw object (display *str*ucture).

```
str(pc)
## List of 5
##  $ sdev    : num [1:4] 2.056 0.493 0.28 0.154
##  $ rotation: num [1:4, 1:4] 0.3614 -0.0845 0.8567 0.3583 -0.6566 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
##   .. ..$ : chr [1:4] "PC1" "PC2" "PC3" "PC4"
##  $ center  : Named num [1:4] 5.84 3.06 3.76 1.2
##   ..- attr(*, "names")= chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Peta
##  $ scale   : logi FALSE
##  $ x       : num [1:150, 1:4] -2.68 -2.71 -2.89 -2.75 -2.73 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:4] "PC1" "PC2" "PC3" "PC4"
##  - attr(*, "class")= chr "prcomp"
```

The object `pc` (like most objects in R) is a list with a class attribute. The list element `x` contains the data points projected on the principal components. We can convert the matrix into a tibble and add the species column from the original dataset back (since the rows are in the same order), and then display

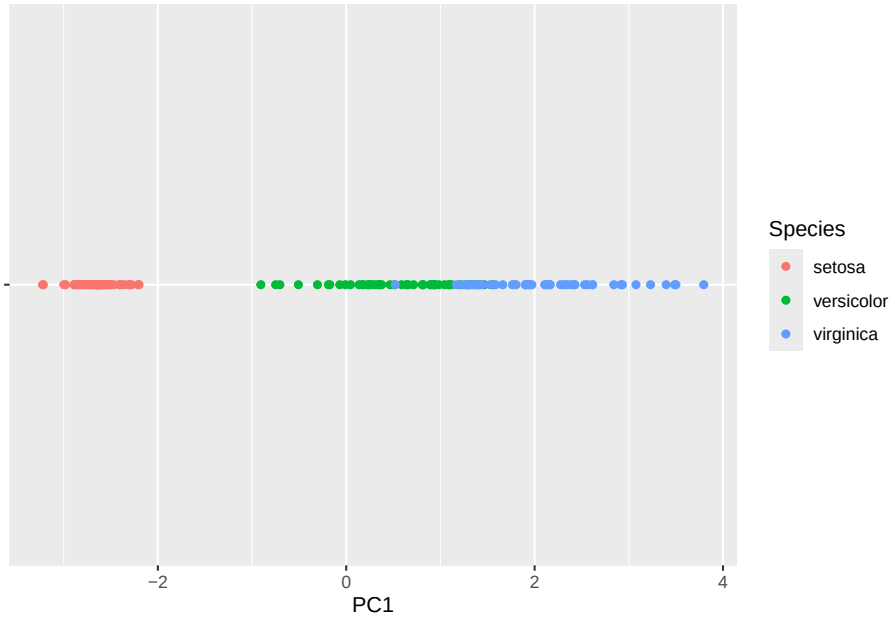
the data projected on the first two principal components.

```
iris_projected <- as_tibble(pc$x) |>
  add_column(Species = iris$Species)

ggplot(iris_projected, aes(x = PC1, y = PC2, color = Species)) +
  geom_point()
```



Flowers that are displayed close together in this projection are also close together in the original 4-dimensional space. Since the first principal component represents most of the variability, we can also show the data projected only on PC1.

```
ggplot(iris_projected,
  aes(x = PC1, y = 0, color = Species)) +
  geom_point() +
  scale_y_continuous(expand=c(0,0)) +
  theme(axis.text.y = element_blank(),
      axis.title.y = element_blank()
  )
```

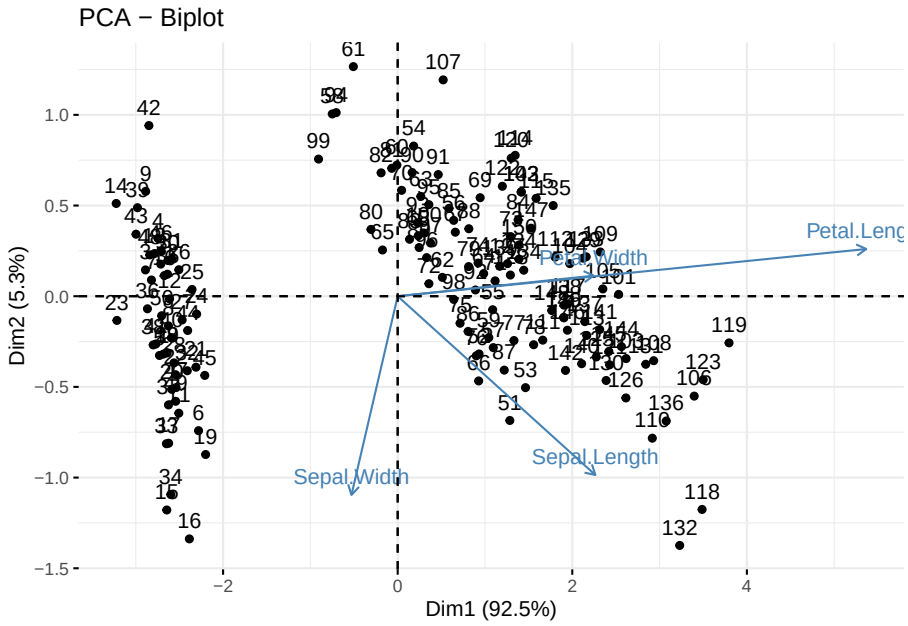


We see that we can perfectly separate the species Setosa using just the first principal component. The other two species are harder to separate.

A plot of the projected data with the original axes added as arrows is called a biplot[9]. If the arrows (original axes) align roughly with the axes of the projection, then they are correlated (linearly dependent).

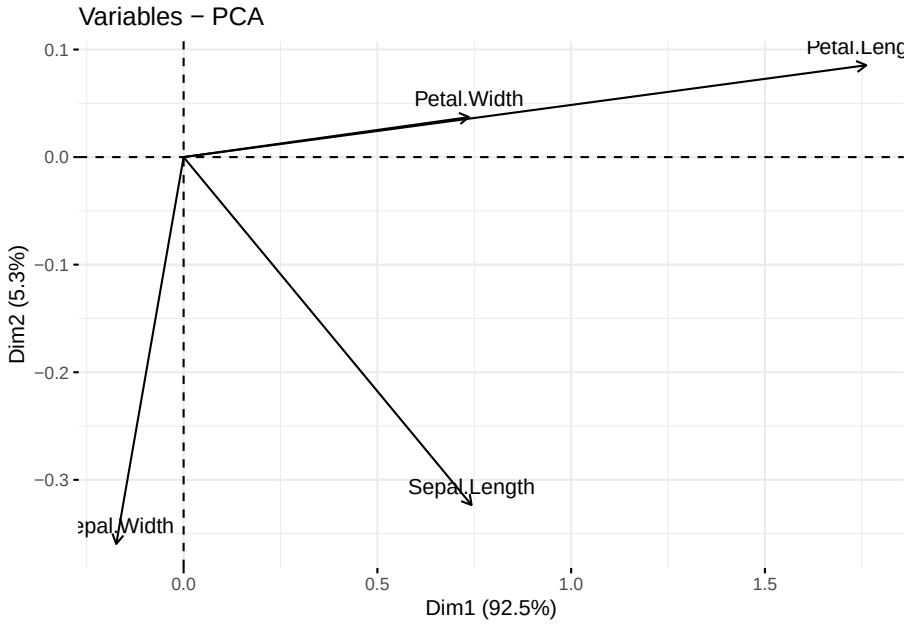
```
library(factoextra)
fviz_pca(pc)
```

[9] https://en.wikipedia.org/wiki/Biplot



We can also display only the old and new axes.

```
fviz_pca_var(pc)
```



We see Petal.Width and Petal.Length point in the same direction which indi-

cates that they are highly correlated. They are also roughly aligned with PC1 (called Dim1 in the plot) which means that PC1 represents most of the variability of these two variables. Sepal.Width is almost aligned with the y-axis and therefore it is represented by PC2 (Dim2). Petal.Width/Petal.Length and Sepal.Width are almost at 90 degrees, indicating that they are close to uncorrelated. Sepal.Length is correlated to all other variables and represented by both, PC1 and PC2 in the projection.

There exist other methods to embed data from higher dimensions into a lower-dimensional space. A popular method to project data into lower dimensions for visualization is **t-distributed stochastic neighbor embedding (t-SNE)** available in package `Rtsne`.

2.3.3.2 Multi-Dimensional Scaling (MDS)

MDS[10] is similar to PCA. Instead of data points, it starts with pairwise distances (i.e., a distance matrix) and produces a space where points are placed to represent these distances as well as possible. The axes in this space are called components and are similar to the principal components in PCA.

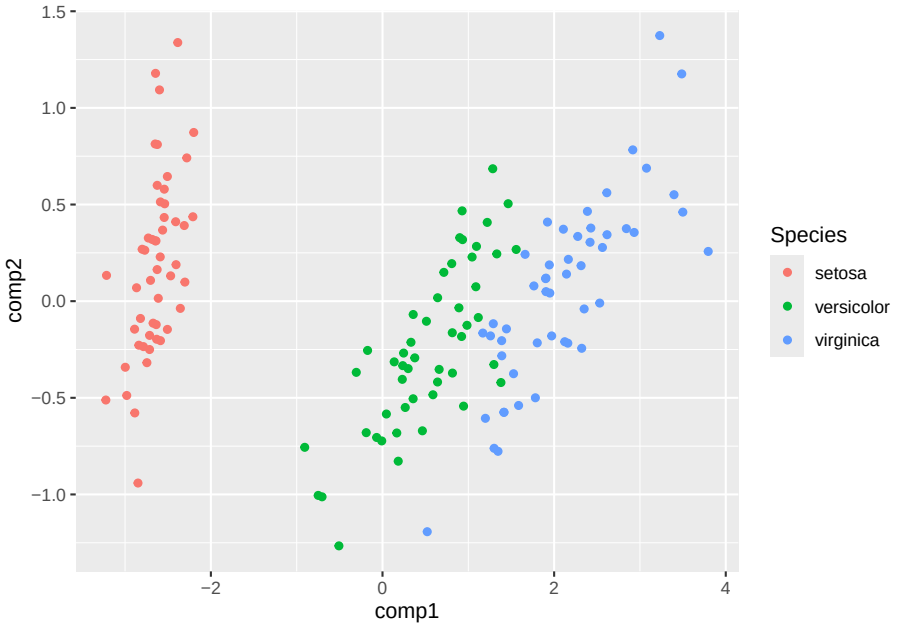
First, we calculate a distance matrix (Euclidean distances) from the 4-d space of the iris dataset.

```
d <- iris |>
  select(-Species) |>
  dist()
```

Metric (classic) MDS tries to construct a space where points with lower distances are placed closer together. We project the data represented by a distance matrix on `k = 2` dimensions.

```
fit <- cmdscale(d, k = 2)
colnames(fit) <- c("comp1", "comp2")
fit <- as_tibble(fit) |>
  add_column(Species = iris$Species)

ggplot(fit, aes(x = comp1, y = comp2, color = Species)) +
  geom_point()
```

[10] https://en.wikipedia.org/wiki/Multidimensional_scaling



The resulting projection is similar (except for rotation and reflection) to the result of the projection using PCA.

2.3.3.3 Non-Parametric Multidimensional Scaling

Non-parametric multidimensional scaling performs MDS while relaxing the need of linear relationships. Methods are available in package `MASS` as functions `isoMDS()` (implements isoMAP[11]) and `sammon()`.

2.3.3.4 Other Nonlinear Dimensionality Reduction Methods

Nonlinear dimensionality reduction[12] is also called manifold learning or creating a low-Dimensional embedding. Many methods are available with varying properties exist. Some popular methods are:

- t-distributed stochastic neighbor embedding[13] (`Rtsne()` in package `Rtsne`) and uniform manifold approximation and projection (`umap()` in package `umap`) are used for projecting data to 2 dimensions for visualization.
- Word2vec[14] (`word2vec` in package `word2vec`) is used in natural language processing[15] to convert words to numeric vectors for similarity calculation.

[11] https://en.wikipedia.org/wiki/Isomap
[12] https://en.wikipedia.org/wiki/Nonlinear_dimensionality_reduction
[13] https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding
[14] https://en.wikipedia.org/wiki/Word2vec
[15] https://cran.r-project.org/web/views/NaturalLanguageProcessing.html

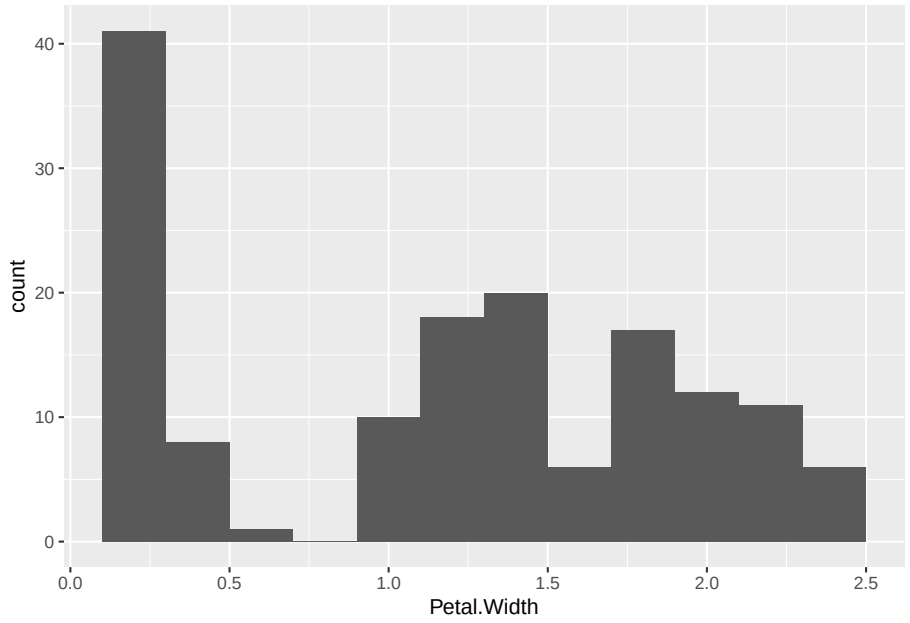
2.3.4 Feature Subset Selection

Feature selection is the process of identifying the features that are used to create a model. We will talk about feature selection when we discuss classification models in Chapter 3 in Feature Selection*.

2.3.5 Discretization

Some data mining methods require discrete data. Discretization converts continuous features into discrete features. As an example, we will discretize the continuous feature Petal.Width. Before we perform discretization, we should look at the distribution and see if it gives us an idea how we should group the continuous values into a set of discrete values. A histogram visualizes the distribution of a single continuous feature.

```
ggplot(iris, aes(x = Petal.Width)) +
  geom_histogram(binwidth = .2)
```



The bins in the histogram represent a discretization using a fixed bin width. The R function `cut()` performs equal interval width discretization creating a vector of type `factor` where each level represents an interval.

```
iris |>
  pull(Sepal.Width) |>
  cut(breaks = 3)
##   [1] (2.8,3.6] (2.8,3.6] (2.8,3.6] (2.8,3.6] (2.8,3.6]
##   [6] (3.6,4.4] (2.8,3.6] (2.8,3.6] (2.8,3.6] (2.8,3.6]
```

```
##  [11] (3.6,4.4] (2.8,3.6] (2.8,3.6] (2.8,3.6] (3.6,4.4]
##  [16] (3.6,4.4] (3.6,4.4] (2.8,3.6] (3.6,4.4] (3.6,4.4]
##  [21] (2.8,3.6] (3.6,4.4] (2.8,3.6] (2.8,3.6] (2.8,3.6]
##  [26] (2.8,3.6] (2.8,3.6] (2.8,3.6] (2.8,3.6] (2.8,3.6]
##  [31] (2.8,3.6] (2.8,3.6] (3.6,4.4] (3.6,4.4] (2.8,3.6]
##  [36] (2.8,3.6] (2.8,3.6] (2.8,3.6] (2.8,3.6] (2.8,3.6]
##  [41] (2.8,3.6] (2,2.8]   (2.8,3.6] (2.8,3.6] (3.6,4.4]
##  [46] (2.8,3.6] (3.6,4.4] (2.8,3.6] (3.6,4.4] (2.8,3.6]
##  [51] (2.8,3.6] (2.8,3.6] (2.8,3.6] (2,2.8]   (2,2.8]
##  [56] (2,2.8]   (2.8,3.6] (2,2.8]   (2.8,3.6] (2,2.8]
##  [61] (2,2.8]   (2.8,3.6] (2,2.8]   (2.8,3.6] (2.8,3.6]
##  [66] (2.8,3.6] (2.8,3.6] (2,2.8]   (2,2.8]   (2,2.8]
##  [71] (2.8,3.6] (2,2.8]   (2,2.8]   (2,2.8]   (2.8,3.6]
##  [76] (2.8,3.6] (2,2.8]   (2.8,3.6] (2.8,3.6] (2,2.8]
##  [81] (2,2.8]   (2,2.8]   (2,2.8]   (2,2.8]   (2.8,3.6]
##  [86] (2.8,3.6] (2.8,3.6] (2,2.8]   (2.8,3.6] (2,2.8]
##  [91] (2,2.8]   (2.8,3.6] (2,2.8]   (2,2.8]   (2,2.8]
##  [96] (2.8,3.6] (2.8,3.6] (2.8,3.6] (2,2.8]   (2,2.8]
## [101] (2.8,3.6] (2,2.8]   (2.8,3.6] (2.8,3.6] (2.8,3.6]
## [106] (2.8,3.6] (2,2.8]   (2.8,3.6] (2,2.8]   (2.8,3.6]
## [111] (2.8,3.6] (2,2.8]   (2.8,3.6] (2,2.8]   (2,2.8]
## [116] (2.8,3.6] (2.8,3.6] (3.6,4.4] (2,2.8]   (2,2.8]
## [121] (2.8,3.6] (2,2.8]   (2,2.8]   (2,2.8]   (2.8,3.6]
## [126] (2.8,3.6] (2,2.8]   (2.8,3.6] (2,2.8]   (2.8,3.6]
## [131] (2,2.8]   (3.6,4.4] (2,2.8]   (2,2.8]   (2,2.8]
## [136] (2.8,3.6] (2.8,3.6] (2.8,3.6] (2.8,3.6] (2.8,3.6]
## [141] (2.8,3.6] (2.8,3.6] (2,2.8]   (2.8,3.6] (2.8,3.6]
## [146] (2.8,3.6] (2,2.8]   (2.8,3.6] (2.8,3.6] (2.8,3.6]
## Levels: (2,2.8] (2.8,3.6] (3.6,4.4]
```

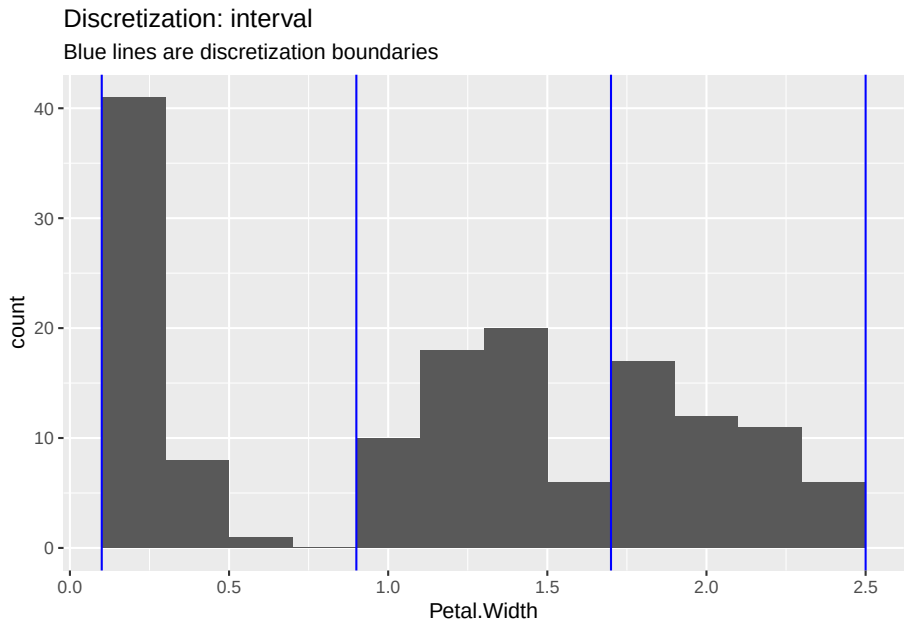
Other discretization methods include equal frequency discretization or using k-means clustering. These methods are implemented by several R packages. We use here the implementation in package `arules` and visualize the results as histograms with blue lines to separate intervals assigned to each discrete value.

```
library(arules)
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
## Attaching package: 'arules'
## The following object is masked from 'package:dplyr':
##
```

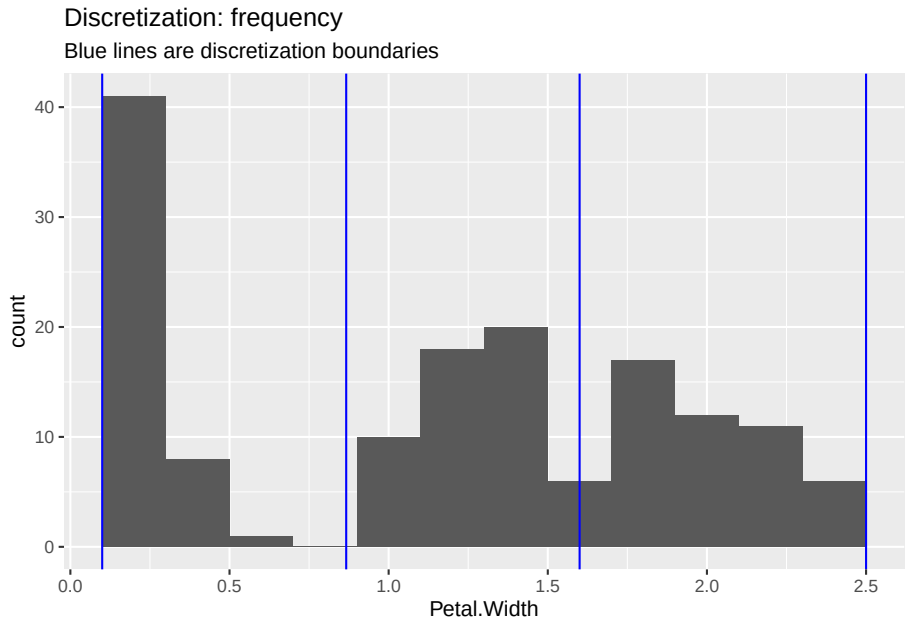
```
##     recode
## The following objects are masked from 'package:base':
##
##     abbreviate, write
iris |> pull(Petal.Width) |>
  discretize(method = "interval", breaks = 3)
##   [1] [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9)
##   [6] [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9)
##  [11] [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9)
##  [16] [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9)
##  [21] [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9)
##  [26] [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9)
##  [31] [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9)
##  [36] [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9)
##  [41] [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9)
##  [46] [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9) [0.1,0.9)
##  [51] [0.9,1.7) [0.9,1.7) [0.9,1.7) [0.9,1.7) [0.9,1.7)
##  [56] [0.9,1.7) [0.9,1.7) [0.9,1.7) [0.9,1.7) [0.9,1.7)
##  [61] [0.9,1.7) [0.9,1.7) [0.9,1.7) [0.9,1.7) [0.9,1.7)
##  [66] [0.9,1.7) [0.9,1.7) [0.9,1.7) [0.9,1.7) [0.9,1.7)
##  [71] [1.7,2.5] [0.9,1.7) [0.9,1.7) [0.9,1.7) [0.9,1.7)
##  [76] [0.9,1.7) [0.9,1.7) [1.7,2.5] [0.9,1.7) [0.9,1.7)
##  [81] [0.9,1.7) [0.9,1.7) [0.9,1.7) [0.9,1.7) [0.9,1.7)
##  [86] [0.9,1.7) [0.9,1.7) [0.9,1.7) [0.9,1.7) [0.9,1.7)
##  [91] [0.9,1.7) [0.9,1.7) [0.9,1.7) [0.9,1.7) [0.9,1.7)
##  [96] [0.9,1.7) [0.9,1.7) [0.9,1.7) [0.9,1.7) [0.9,1.7)
## [101] [1.7,2.5] [1.7,2.5] [1.7,2.5] [1.7,2.5] [1.7,2.5]
## [106] [1.7,2.5] [1.7,2.5] [1.7,2.5] [1.7,2.5] [1.7,2.5]
## [111] [1.7,2.5] [1.7,2.5] [1.7,2.5] [1.7,2.5] [1.7,2.5]
## [116] [1.7,2.5] [1.7,2.5] [1.7,2.5] [1.7,2.5] [0.9,1.7)
## [121] [1.7,2.5] [1.7,2.5] [1.7,2.5] [1.7,2.5] [1.7,2.5]
## [126] [1.7,2.5] [1.7,2.5] [1.7,2.5] [1.7,2.5] [0.9,1.7)
## [131] [1.7,2.5] [1.7,2.5] [1.7,2.5] [0.9,1.7) [0.9,1.7)
## [136] [1.7,2.5] [1.7,2.5] [1.7,2.5] [1.7,2.5] [1.7,2.5]
## [141] [1.7,2.5] [1.7,2.5] [1.7,2.5] [1.7,2.5] [1.7,2.5]
## [146] [1.7,2.5] [1.7,2.5] [1.7,2.5] [1.7,2.5] [1.7,2.5]
## attr(,"discretized:breaks")
## [1] 0.1 0.9 1.7 2.5
## attr(,"discretized:method")
## [1] interval
## Levels: [0.1,0.9) [0.9,1.7) [1.7,2.5]
```

To show the differences between the methods, we use the three discretization methods and draw blue lines in the histogram to show how they cut the data.

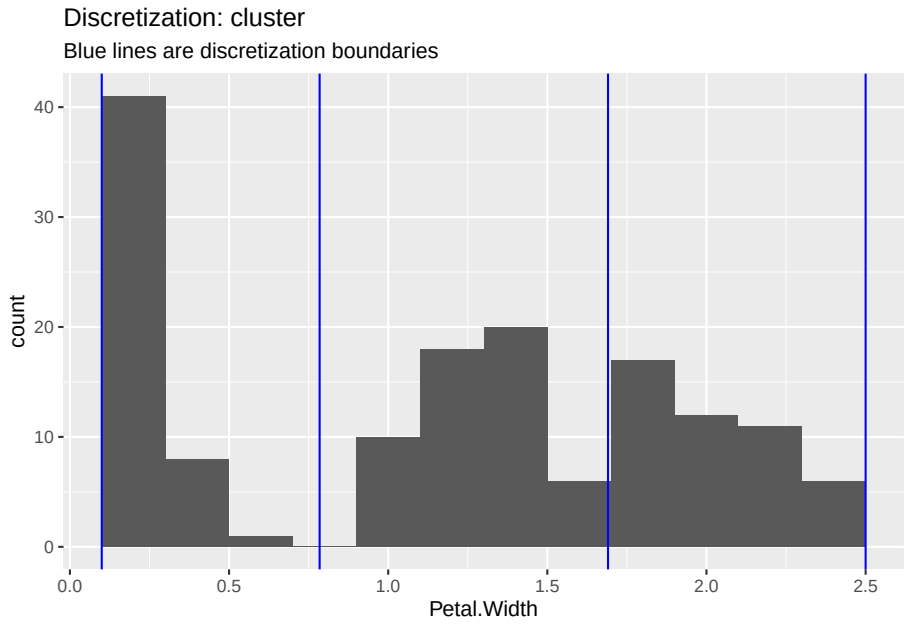
```
ggplot(iris, aes(Petal.Width)) + geom_histogram(binwidth = .2) +
  geom_vline(
      xintercept = iris |>
      pull(Petal.Width) |>
      discretize(method = "interval", breaks = 3, onlycuts = TRUE),
    color = "blue"
  ) +
  labs(title = "Discretization: interval",
       subtitle = "Blue lines are discretization boundaries")
```




```
ggplot(iris, aes(Petal.Width)) + geom_histogram(binwidth = .2) +
  geom_vline(
    xintercept = iris |>
    pull(Petal.Width) |>
    discretize(method = "frequency", breaks = 3, onlycuts = TRUE),
   color = "blue"
  ) +
  labs(title = "Discretization: frequency",
       subtitle = "Blue lines are discretization boundaries")
```




```
ggplot(iris, aes(Petal.Width)) + geom_histogram(binwidth = .2) +
  geom_vline(
    xintercept = iris |>
    pull(Petal.Width) |>
    discretize(method = "cluster", breaks = 3, onlycuts = TRUE),
   color = "blue"
  ) +
  labs(title = "Discretization: cluster",
       subtitle = "Blue lines are discretization boundaries")
```



The user needs to decide on the number of intervals and the used method.

2.3.6 Variable Transformation: Standardization

Standardizing (scaling, normalizing) the range of features values is important to make them comparable. The most popular method is to convert the values of each feature to z-scores[16]. by subtracting the mean (centering) and dividing by the standard deviation (scaling). The standardized feature will have a mean of zero and are measured in standard deviations from the mean. Positive values indicate how many standard deviation the original feature value was above the average. Negative standardized values indicate below-average values.

R-base provides the function `scale()` to standardize the columns in a data.frame. Tidyverse currently does not have a simple scale function, so we make one It mutates all numeric columns using an anonymous function that calculates the z-score.

```
scale_numeric <- function(x)
  x |>
  mutate(across(where(is.numeric),
                function(y) (y - mean(y, na.rm = TRUE)) / sd(y, na.rm = TRUE)))
```

```
iris.scaled <- iris |>
  scale_numeric()
iris.scaled
```

[16] https://en.wikipedia.org/wiki/Standard_score

```
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>
##  1       -0.898      1.02          -1.34       -1.31 setosa
##  2       -1.14      -0.132         -1.34       -1.31 setosa
##  3       -1.38       0.327         -1.39       -1.31 setosa
##  4       -1.50       0.0979        -1.28       -1.31 setosa
##  5       -1.02       1.25          -1.34       -1.31 setosa
##  6       -0.535      1.93          -1.17       -1.05 setosa
##  7       -1.50       0.786         -1.34       -1.18 setosa
##  8       -1.02       0.786         -1.28       -1.31 setosa
##  9       -1.74      -0.361         -1.34       -1.31 setosa
## 10       -1.14       0.0979        -1.28       -1.44 setosa
## # i 140 more rows
summary(iris.scaled)
##   Sepal.Length      Sepal.Width      Petal.Length
##  Min.   :-1.8638   Min.   :-2.426   Min.   :-1.562
##  1st Qu.:-0.8977   1st Qu.:-0.590   1st Qu.:-1.222
##  Median :-0.0523   Median :-0.132   Median : 0.335
##  Mean   : 0.0000   Mean   : 0.000   Mean   : 0.000
##  3rd Qu.: 0.6722   3rd Qu.: 0.557   3rd Qu.: 0.760
##  Max.   : 2.4837   Max.   : 3.080   Max.   : 1.780
##   Petal.Width          Species
##  Min.   :-1.442   setosa    :50
##  1st Qu.:-1.180   versicolor:50
##  Median : 0.132   virginica :50
##  Mean   : 0.000
##  3rd Qu.: 0.788
##  Max.   : 1.706
```

The standardized feature has a mean of zero and most "normal" values will fall in the range $[-3, 3]$ and is measured in standard deviations from the average. Negative values mean smaller than the average and positive values mean larger than the average.

2.4 Measures of Similarity and Dissimilarity

Proximities help with quantifying how similar two objects are. Similariy[17] is a concept from geometry. The best-known way to define similarity is Euclidean distance, but proximities can be measured in different ways depending on the information we have about the objects.

R stores proximity as dissimilarities/distances matrices. Similarities are first converted to dissimilarities. Distances are symmetric, i.e., the distance from A

[17] https://en.wikipedia.org/wiki/Similarity_(geometry)

to B is the same as the distance from B to A. R therefore stores only a triangle (typically the lower triangle) of the distance matrix.

2.4.1 Minkowsky Distances

The Minkowsky distance[18] is a family of metric distances including Euclidean and Manhattan distance. To avoid one feature to dominate the distance calculation, scaled data is typically used. We select the first 5 flowers for this example.

```
iris_sample <- iris.scaled |>
  select(-Species) |>
  slice(1:5)
iris_sample
## # A tibble: 5 x 4
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
##          <dbl>       <dbl>        <dbl>       <dbl>
## 1       -0.898      1.02         -1.34       -1.31
## 2       -1.14      -0.132        -1.34       -1.31
## 3       -1.38       0.327        -1.39       -1.31
## 4       -1.50       0.0979       -1.28       -1.31
## 5       -1.02       1.25         -1.34       -1.31
```

Different types of Minkowsky distance matrices between the first 5 flowers can be calculated using `dist()`.

```
dist(iris_sample, method = "euclidean")
##        1      2      3      4
## 2 1.1723
## 3 0.8428 0.5216
## 4 1.1000 0.4326 0.2829
## 5 0.2593 1.3819 0.9883 1.2460
dist(iris_sample, method = "manhattan")
##        1      2      3      4
## 2 1.3887
## 3 1.2280 0.7570
## 4 1.5782 0.6484 0.4635
## 5 0.3502 1.4973 1.3367 1.6868
dist(iris_sample, method = "maximum")
##        1      2      3      4
## 2 1.1471
## 3 0.6883 0.4589
## 4 0.9177 0.3623 0.2294
## 5 0.2294 1.3766 0.9177 1.1471
```

We see that only the lower triangle of the distance matrices are stored (note

[18] https://en.wikipedia.org/wiki/Minkowski_distance

that rows start with row 2).

2.4.2 Distances for Binary Data

Binary data can be encodes as `0` and `1` (numeric) or `TRUE` and `FALSE` (logical).

```
b <- rbind(
  c(0,0,0,1,1,1,1,0,0,1),
  c(0,0,1,1,1,0,0,1,0,0)
  )
b
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    0    0    1    1    1    1    0    0     1
## [2,]    0    0    1    1    1    0    0    1    0     0
b_logical <- apply(b, MARGIN = 2, as.logical)
b_logical
##       [,1]  [,2]  [,3] [,4] [,5]  [,6]  [,7]  [,8]  [,9]
## [1,] FALSE FALSE FALSE TRUE TRUE  TRUE  TRUE FALSE FALSE
## [2,] FALSE FALSE  TRUE TRUE TRUE FALSE FALSE  TRUE FALSE
##      [,10]
## [1,]  TRUE
## [2,] FALSE
```

2.4.2.1 Hamming Distance

The Hamming distance[19] is the number of mismatches between two binary vectors. For 0-1 data this is equivalent to the Manhattan distance and also the squared Euclidean distance.

```
dist(b, method = "manhattan")
##   1
## 2 5
dist(b, method = "euclidean")^2
##   1
## 2 5
```

2.4.2.2 Jaccard Index

The Jaccard index[20] is a similarity measure that focuses on matching 1s. R converts the similarity into a dissimilarity using $d_J = 1 - s_J$.

```
dist(b, method = "binary")
##        1
## 2 0.7143
```

[19] https://en.wikipedia.org/wiki/Hamming_distance
[20] https://en.wikipedia.org/wiki/Jaccard_index

2.4.3 Distances for Mixed Data

Most distance measures work only on numeric data. Often, we have a mixture of numbers and nominal or ordinal features like this data:

```
people <- tibble(
  height = c(      160,    185,    170),
  weight = c(       52,     90,     75),
  sex    = c( "female", "male", "male")
)
people
## # A tibble: 3 x 3
##   height weight sex
##    <dbl>  <dbl> <chr>
## 1    160     52 female
## 2    185     90 male
## 3    170     75 male
```

It is important that nominal features are stored as factors and not character (`<chr>`).

```
people <- people |>
  mutate(across(where(is.character), factor))
people
## # A tibble: 3 x 3
##   height weight sex
##    <dbl>  <dbl> <fct>
## 1    160     52 female
## 2    185     90 male
## 3    170     75 male
```

2.4.3.1 Gower's Coefficient

The Gower's coefficient of similarity works with mixed data by calculating the appropriate similarity for each feature and then aggregating them into a single measure. The package `proxy` implements Gower's coefficient converted into a distance.

```
library(proxy)
##
## Attaching package: 'proxy'
## The following object is masked from 'package:Matrix':
##
##     as.matrix
## The following objects are masked from 'package:stats':
##
##     as.dist, dist
## The following object is masked from 'package:base':
```

```
##
##     as.matrix
d_Gower <- dist(people, method = "Gower")
d_Gower
##        1      2
## 2 1.0000
## 3 0.6684 0.3316
```

Gower's coefficient calculation implicitly scales the data because it calculates distances on each feature individually, so there is no need to scale the data first.

2.4.3.2 Using Euclidean Distance with Mixed Data

Sometimes methods (e.g., k-means) can only use Euclidean distance. In this case, nominal features can be converted into 0-1 dummy variables. After scaling, Euclidean distance will result in a usable distance measure.

We use package `caret` to create dummy variables.

```
library(caret)
## Loading required package: lattice
##
## Attaching package: 'caret'
## The following object is masked from 'package:purrr':
##
##     lift
data_dummy <- dummyVars(~., people) |>
  predict(people)
data_dummy
##   height weight sex.female sex.male
## 1    160     52          1        0
## 2    185     90          0        1
## 3    170     75          0        1
```

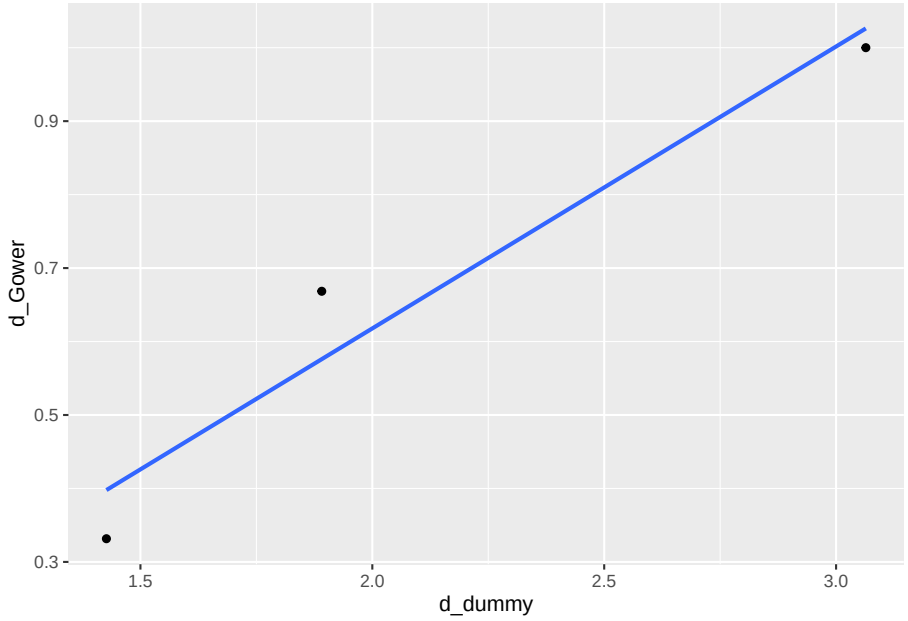
Note that feature sex has now two columns. If we want that height, weight and sex have the same influence on the distance measure, then we need to weight the sex columns by 1/2 after scaling.

```
weight_matrix <- matrix(c(1, 1, 1/2, 1/2),
                        ncol = 4,
                        nrow = nrow(data_dummy),
                        byrow = TRUE)
data_dummy_scaled <- scale(data_dummy) * weight_matrix

d_dummy <- dist(data_dummy_scaled)
d_dummy
##       1     2
## 2 3.064
```

```
## 3 1.891 1.427
```

The distance using dummy variables is consistent with Gower's distance. However, note that Gower's distance is scaled between 0 and 1 while the Euclidean distance is not.

```
ggplot(tibble(d_dummy, d_Gower), aes(x = d_dummy, y = d_Gower)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
## Don't know how to automatically pick scale for object of
## type <dist>. Defaulting to continuous.
## Don't know how to automatically pick scale for object of
## type <dist>. Defaulting to continuous.
## `geom_smooth()` using formula = 'y ~ x'
```



2.4.4 More Proximity Measures

The package `proxy` implements a wide array of proximity measures (similarity measures are converted into distances).

```
library(proxy)
pr_DB$get_entry_names()
##  [1] "Jaccard"          "Kulczynski1"      "Kulczynski2"
##  [4] "Mountford"        "Fager"            "Russel"
##  [7] "simple matching" "Hamman"           "Faith"
## [10] "Tanimoto"         "Dice"             "Phi"
```

```
## [13] "Stiles"          "Michael"          "Mozley"
## [16] "Yule"            "Yule2"            "Ochiai"
## [19] "Simpson"         "Braun-Blanquet"   "cosine"
## [22] "angular"         "eJaccard"         "eDice"
## [25] "correlation"     "Chi-squared"      "Phi-squared"
## [28] "Tschuprow"       "Cramer"           "Pearson"
## [31] "Gower"           "Euclidean"        "Mahalanobis"
## [34] "Bhjattacharyya"  "Manhattan"        "supremum"
## [37] "Minkowski"       "Canberra"         "Wave"
## [40] "divergence"      "Kullback"         "Bray"
## [43] "Soergel"         "Levenshtein"      "Podani"
## [46] "Chord"           "Geodesic"         "Whittaker"
## [49] "Hellinger"       "fJaccard"
```

Note that loading the package `proxy` overwrites the default `dist()` function in R. You can specify which dist function to use by specifying the package in the call. For example `stats::dist()` calls the default function in R (the package `stats` is part of R) while `proxy::dist()` calls the version in the package `proxy`.

2.5 Data Exploration*

The following code covers the important part of data exploration. For space reasons, this chapter was moved from the printed textbook to this Data Exploration Web Chapter.[21]

We will use again the iris dataset.

```
library(tidyverse)
data(iris)
iris <- as_tibble(iris)
iris
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>
##  1          5.1         3.5          1.4         0.2 setosa
##  2          4.9         3            1.4         0.2 setosa
##  3          4.7         3.2          1.3         0.2 setosa
##  4          4.6         3.1          1.5         0.2 setosa
##  5          5           3.6          1.4         0.2 setosa
##  6          5.4         3.9          1.7         0.4 setosa
##  7          4.6         3.4          1.4         0.3 setosa
##  8          5           3.4          1.5         0.2 setosa
##  9          4.4         2.9          1.4         0.2 setosa
## 10          4.9         3.1          1.5         0.1 setosa
```

[21] https://www-users.cse.umn.edu/~kumar001/dmbook/data_exploration_1st_edition.pdf

```
## # i 140 more rows
```

2.5.1 Basic statistics

Get summary statistics (using base R)

```
summary(iris)
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.30   Min.   :2.00   Min.   :1.00   Min.   :0.1
##  1st Qu.:5.10   1st Qu.:2.80   1st Qu.:1.60   1st Qu.:0.3
##  Median :5.80   Median :3.00   Median :4.35   Median :1.3
##  Mean   :5.84   Mean   :3.06   Mean   :3.76   Mean   :1.2
##  3rd Qu.:6.40   3rd Qu.:3.30   3rd Qu.:5.10   3rd Qu.:1.8
##  Max.   :7.90   Max.   :4.40   Max.   :6.90   Max.   :2.5
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

Get mean and standard deviation for sepal length.

```
iris |>
  summarize(avg_Sepal.Length = mean(Sepal.Length),
            sd_Sepal.Length = sd(Sepal.Length))
## # A tibble: 1 x 2
##   avg_Sepal.Length sd_Sepal.Length
##              <dbl>           <dbl>
## 1             5.84           0.828
```

Data with missing values will result in statistics of `NA`. Adding the parameter `na.rm = TRUE` can be used in most statistics functions to ignore missing values.

```
mean(c(1, 2, NA, 3, 4, 5))
## [1] NA
mean(c(1, 2, NA, 3, 4, 5),  na.rm = TRUE)
## [1] 3
```

Outliers are typically the smallest or the largest values of a feature. To make the mean more robust against outliers, we can trim 10% of observations from each end of the distribution.

```
iris |>
  summarize(
    avg_Sepal.Length = mean(Sepal.Length),
    trimmed_avg_Sepal.Length = mean(Sepal.Length, trim = .1)
```

```
  )
## # A tibble: 1 x 2
##   avg_Sepal.Length trimmed_avg_Sepal.Length
##              <dbl>                    <dbl>
## 1             5.84                     5.81
```

Sepal length does not have outliers, so the trimmed mean is almost identical.

To calculate a summary for a set of features (e.g., all numeric features), tidyverse provides `across(where(is.numeric), fun)`.

```
iris |> summarize(across(where(is.numeric), mean))
## # A tibble: 1 x 4
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
##          <dbl>       <dbl>        <dbl>       <dbl>
## 1         5.84        3.06         3.76        1.20
iris |> summarize(across(where(is.numeric), sd))
## # A tibble: 1 x 4
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
##          <dbl>       <dbl>        <dbl>       <dbl>
## 1        0.828       0.436         1.77       0.762
iris |> summarize(across(where(is.numeric),
            list(min = min,
                 median = median,
                 max = max)))
## # A tibble: 1 x 12
##   Sepal.Length_min Sepal.Length_median Sepal.Length_max
##              <dbl>               <dbl>            <dbl>
## 1              4.3                 5.8              7.9
## # i 9 more variables: Sepal.Width_min <dbl>,
## #   Sepal.Width_median <dbl>, Sepal.Width_max <dbl>,
## #   Petal.Length_min <dbl>, Petal.Length_median <dbl>,
## #   Petal.Length_max <dbl>, Petal.Width_min <dbl>,
## #   Petal.Width_median <dbl>, Petal.Width_max <dbl>
```

The median absolute deviation (MAD) is another measure of dispersion.

```
iris |> summarize(across(where(is.numeric), mad))
## # A tibble: 1 x 4
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
##          <dbl>       <dbl>        <dbl>       <dbl>
## 1         1.04       0.445         1.85        1.04
```

2.5.2 Grouped Operations and Calculations

We can use the nominal feature to form groups and then calculate group-wise statistics for the continuous features. We often use group-wise averages to see

if they differ between groups.

```
iris |>
  group_by(Species) |>
  summarize(across(Sepal.Length, mean))
## # A tibble: 3 x 2
##   Species    Sepal.Length
##   <fct>             <dbl>
## 1 setosa             5.01
## 2 versicolor         5.94
## 3 virginica          6.59
iris |>
  group_by(Species) |>
  summarize(across(where(is.numeric), mean))
## # A tibble: 3 x 5
##   Species  Sepal.Length Sepal.Width Petal.Length Petal.Width
##   <fct>           <dbl>       <dbl>        <dbl>       <dbl>
## 1 setosa           5.01        3.43         1.46       0.246
## 2 versico~         5.94        2.77         4.26       1.33
## 3 virgini~         6.59        2.97         5.55       2.03
```

We see that the species Virginica has the highest average for all, but Sepal.Width.

The statistical difference between the groups can be tested using ANOVA (analysis of variance)[22].

```
res.aov <- aov(Sepal.Length ~ Species, data = iris)
summary(res.aov)
##              Df Sum Sq Mean Sq F value Pr(>F)
## Species       2   63.2   31.61     119 <2e-16 ***
## Residuals   147   39.0    0.27
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
TukeyHSD(res.aov)
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = Sepal.Length ~ Species, data = iris)
##
## $Species
##                       diff    lwr    upr p adj
## versicolor-setosa    0.930 0.6862 1.1738     0
## virginica-setosa     1.582 1.3382 1.8258     0
## virginica-versicolor 0.652 0.4082 0.8958     0
```

[22] http://www.sthda.com/english/wiki/one-way-anova-test-in-r

The summary shows that there is a significant difference for Sepal.Length between the groups. `TukeyHDS` evaluates differences between pairs of groups. In this case, all are significantly different. If the data only contains two groups, the `t.test` can be used.

2.5.3 Tabulate data

We can count the number of flowers for each species.

```
iris |>
  group_by(Species) |>
  summarize(n())
## # A tibble: 3 x 2
##   Species    `n()`
##   <fct>      <int>
## 1 setosa        50
## 2 versicolor    50
## 3 virginica     50
```

In base R, this can be also done using `count(iris$Species)`.

For the following examples, we discretize the data using cut.

```
iris_ord <- iris |>
  mutate(across(where(is.numeric),
    function(x) cut(x, 3, labels = c("short", "medium", "long"),
                    ordered = TRUE)))

iris_ord
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##    <ord>        <ord>       <ord>        <ord>       <fct>
##  1 short        medium      short        short       setosa
##  2 short        medium      short        short       setosa
##  3 short        medium      short        short       setosa
##  4 short        medium      short        short       setosa
##  5 short        medium      short        short       setosa
##  6 short        long        short        short       setosa
##  7 short        medium      short        short       setosa
##  8 short        medium      short        short       setosa
##  9 short        medium      short        short       setosa
## 10 short        medium      short        short       setosa
## # i 140 more rows
summary(iris_ord)
##  Sepal.Length Sepal.Width Petal.Length Petal.Width
##  short :59    short :47   short :50    short :50
##  medium:71    medium:88   medium:54    medium:54
```

```
##  long  :20    long  :15   long  :46    long  :46
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
```

Cross tabulation is used to find out if two discrete features are related.

```
tbl <- iris_ord |>
  select(Sepal.Length, Species) |>
  table()
tbl
##              Species
## Sepal.Length setosa versicolor virginica
##       short      47         11         1
##       medium      3         36        32
##       long        0          3        17
```

The table contains the number of rows that contain the combination of values (e.g., the number of flowers with a short Sepal.Length and species Setosa is 47). If a few cells have very large counts and most others have very low counts, then there might be a relationship. For the iris data, we see that species Setosa has mostly a short Sepal.Length, while Versicolor and Virginica have longer sepals.

Creating a cross table with tidyverse is a little more involved and uses pivot operations and grouping.

```
iris_ord |>
  select(Species, Sepal.Length) |>
### Relationship Between Nominal and Ordinal Features
  pivot_longer(cols = Sepal.Length) |>
  group_by(Species, value) |>
  count() |>
  ungroup() |>
  pivot_wider(names_from = Species, values_from = n)
## # A tibble: 3 x 4
##   value  setosa versicolor virginica
##   <ord>   <int>      <int>     <int>
## 1 short      47         11         1
## 2 medium      3         36        32
## 3 long       NA          3        17
```

We can use a statistical test to determine if there is a significant relationship between the two features. Pearson's chi-squared test[23] for independence is performed with the null hypothesis that the joint distribution of the cell counts in a 2-dimensional contingency table is the product of the row and column marginals.

[23] https://en.wikipedia.org/wiki/Chi-squared_test

The null hypothesis h0 is independence between rows and columns.

```
tbl |>
  chisq.test()
##
##  Pearson's Chi-squared test
##
## data:  tbl
## X-squared = 112, df = 4, p-value <2e-16
```

The small p-value indicates that the null hypothesis of independence needs to be rejected. For small counts (cells with counts <5), Fisher's exact test[24] is better.

```
fisher.test(tbl)
##
##  Fisher's Exact Test for Count Data
##
## data:  tbl
## p-value <2e-16
## alternative hypothesis: two.sided
```

2.5.4 Percentiles (Quantiles)

Quantiles[25] are cutting points dividing the range of a probability distribution into continuous intervals with equal probability. For example, the median is the empirical 50% quantile dividing the observations into 50% of the observations being smaller than the median and the other 50% being larger than the median.

By default quartiles are calculated. 25% is typically called Q1, 50% is called Q2 or the median and 75% is called Q3.

```
iris |>
  pull(Petal.Length) |>
  quantile()
##   0%  25%  50%  75% 100%
## 1.00 1.60 4.35 5.10 6.90
```

The interquartile range is a measure for variability that is robust against outliers. It is defined the length Q3 - Q2 which covers the 50% of the data in the middle.

```
iris |>
  summarize(IQR =
  quantile(Petal.Length, probs = 0.75) -
    quantile(Petal.Length, probs = 0.25))
## # A tibble: 1 x 1
```

[24] https://en.wikipedia.org/wiki/Fisher%27s_exact_test
[25] https://en.wikipedia.org/wiki/Quantile

```
##     IQR
##   <dbl>
## 1   3.5
```

2.5.5 Correlation

2.5.5.1 Pearson Correlation

Correlation can be used for ratio/interval scaled features. We typically think of the Pearson correlation coefficient[26] between features (columns).

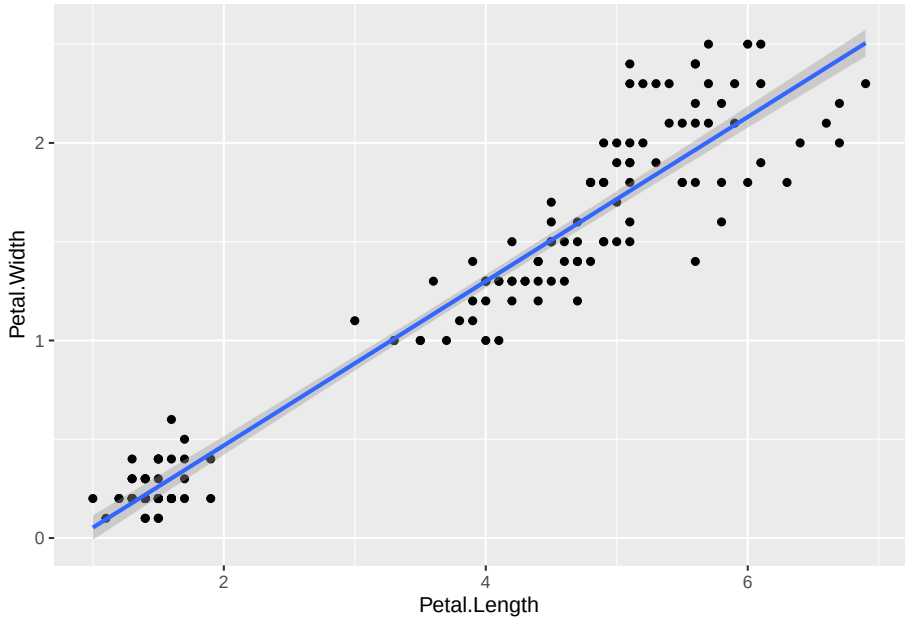
```
cc <- iris |>
  select(-Species) |>
  cor()
cc
##              Sepal.Length Sepal.Width Petal.Length
## Sepal.Length       1.0000     -0.1176       0.8718
## Sepal.Width       -0.1176      1.0000      -0.4284
## Petal.Length       0.8718     -0.4284       1.0000
## Petal.Width        0.8179     -0.3661       0.9629
##              Petal.Width
## Sepal.Length      0.8179
## Sepal.Width      -0.3661
## Petal.Length      0.9629
## Petal.Width       1.0000
```

`cor` calculates a correlation matrix with pairwise correlations between features. Correlation matrices are symmetric, but different to distances, the whole matrix is stored.

The correlation between Petal.Length and Petal.Width can be visualized using a scatter plot.

```
ggplot(iris, aes(Petal.Length, Petal.Width)) +
  geom_point() +
  geom_smooth(method = "lm")
## `geom_smooth()` using formula = 'y ~ x'
```

[26] https://en.wikipedia.org/wiki/Pearson_correlation_coefficient



`geom_smooth` adds a regression line by fitting a linear model (`lm`). Most points are close to this line indicating strong linear dependence (i.e., high correlation).

We can calculate individual correlations by specifying two vectors.

```
with(iris, cor(Petal.Length, Petal.Width))
## [1] 0.9629
```

Note: `with` lets you use columns using just their names and `with(iris, cor(Petal.Length, Petal.Width))` is the same as `cor(iris$Petal.Length, iris$Petal.Width)`.

Finally, we can test if a correlation is significantly different from zero.

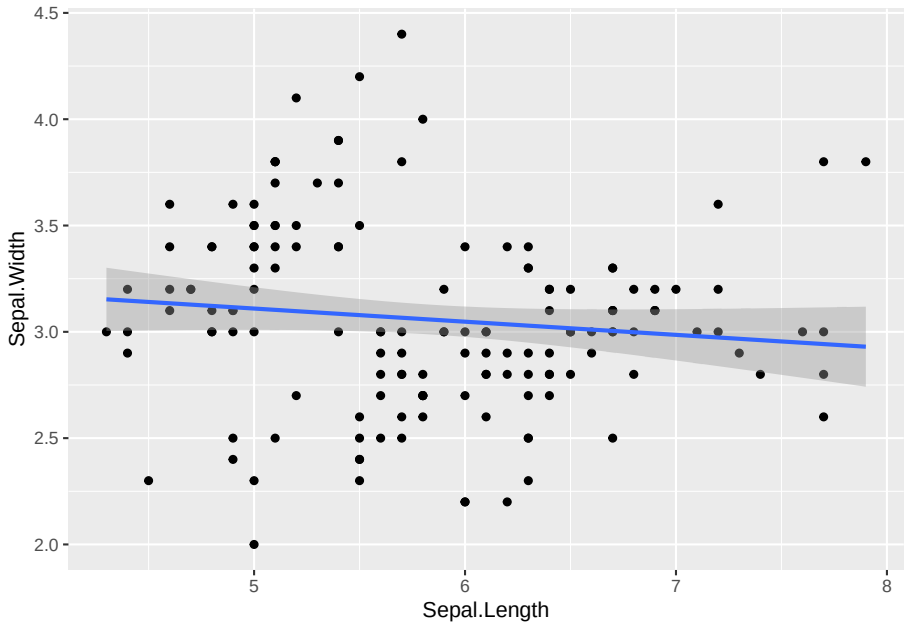
```
with(iris, cor.test(Petal.Length, Petal.Width))
##
##  Pearson's product-moment correlation
##
## data:  Petal.Length and Petal.Width
## t = 43, df = 148, p-value <2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.9491 0.9730
## sample estimates:
##    cor
## 0.9629
```

A small p-value (less than 0.05) indicates that the observed correlation is sig-

nificantly different from zero. This can also be seen by the fact that the 95% confidence interval does not span zero.

Sepal.Length and Sepal.Width show little correlation:

```
ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_point() +
  geom_smooth(method = "lm")
## `geom_smooth()` using formula = 'y ~ x'
```




```
with(iris, cor(Sepal.Length, Sepal.Width))
## [1] -0.1176
with(iris, cor.test(Sepal.Length, Sepal.Width))
##
##  Pearson's product-moment correlation
##
## data:  Sepal.Length and Sepal.Width
## t = -1.4, df = 148, p-value = 0.2
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.27269  0.04351
## sample estimates:
##     cor
## -0.1176
```

2.5.5.2 Rank Correlation

Rank correlation is used for ordinal features or if the correlation is not linear. To show this, we first convert the continuous features in the Iris dataset into ordered factors (ordinal) with three levels using the function `cut`.

```
iris_ord <- iris |>
  mutate(across(where(is.numeric),
    function(x) cut(x, 3,
                    labels = c("short", "medium", "long"),
                    ordered = TRUE)))

iris_ord
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##    <ord>        <ord>       <ord>        <ord>       <fct>
##  1 short        medium      short        short       setosa
##  2 short        medium      short        short       setosa
##  3 short        medium      short        short       setosa
##  4 short        medium      short        short       setosa
##  5 short        medium      short        short       setosa
##  6 short        long        short        short       setosa
##  7 short        medium      short        short       setosa
##  8 short        medium      short        short       setosa
##  9 short        medium      short        short       setosa
## 10 short        medium      short        short       setosa
## # i 140 more rows
summary(iris_ord)
##  Sepal.Length Sepal.Width Petal.Length Petal.Width
##  short :59    short :47   short :50    short :50
##  medium:71    medium:88   medium:54    medium:54
##  long  :20    long  :15   long  :46    long  :46
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
iris_ord |>
  pull(Sepal.Length)
##   [1] short  short  short  short  short  short  short
##   [8] short  short  short  short  short  short  short
##  [15] medium medium short  short  medium short  short
##  [22] short  short  short  short  short  short  short
##  [29] short  short  short  short  short  short  short
##  [36] short  short  short  short  short  short  short
##  [43] short  short  short  short  short  short  short
##  [50] short  long   medium long   short  medium medium
```

```
##  [57] medium short  medium short  short  medium medium
##  [64] medium medium medium medium medium medium medium
##  [71] medium medium medium medium medium medium long
##  [78] medium medium medium short  short  medium medium
##  [85] short  medium medium medium medium short  short
##  [92] medium medium short  medium medium medium medium
##  [99] short  medium medium medium long   medium medium
## [106] long   short  long   medium long   medium medium
## [113] long   medium medium medium medium long   long
## [120] medium long   medium long   medium medium long
## [127] medium medium medium long   long   long   medium
## [134] medium medium long   medium medium medium long
## [141] medium long   medium long   medium medium medium
## [148] medium medium medium
## Levels: short < medium < long
```

Two measures for rank correlation are Kendall's Tau and Spearman's Rho.

Kendall's Tau Rank Correlation Coefficient[27] measures the agreement between two rankings (i.e., ordinal features).

```
iris_ord |>
  select(-Species) |>
  sapply(xtfrm) |>
  cor(method = "kendall")
##              Sepal.Length Sepal.Width Petal.Length
## Sepal.Length       1.0000     -0.1438       0.7419
## Sepal.Width       -0.1438      1.0000      -0.3299
## Petal.Length       0.7419     -0.3299       1.0000
## Petal.Width        0.7295     -0.3154       0.9198
##              Petal.Width
## Sepal.Length      0.7295
## Sepal.Width      -0.3154
## Petal.Length      0.9198
## Petal.Width       1.0000
```

Note: We have to use `xtfrm` to transform the ordered factors into ranks, i.e., numbers representing the order.

Spearman's Rho[28] is equal to the Pearson correlation between the rank values of those two features.

```
iris_ord |>
  select(-Species) |>
  sapply(xtfrm) |>
```

[27] https://en.wikipedia.org/wiki/Kendall_rank_correlation_coefficient
[28] https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient

```
  cor(method = "spearman")
##              Sepal.Length Sepal.Width Petal.Length
## Sepal.Length       1.0000     -0.1570       0.7938
## Sepal.Width       -0.1570      1.0000      -0.3663
## Petal.Length       0.7938     -0.3663       1.0000
## Petal.Width        0.7843     -0.3517       0.9399
##              Petal.Width
## Sepal.Length      0.7843
## Sepal.Width      -0.3517
## Petal.Length      0.9399
## Petal.Width       1.0000
```

Spearman's Rho is much faster to compute on large datasets then Kendall's Tau.

Comparing the rank correlation results with the Pearson correlation on the original data shows that they are very similar. This indicates that discretizing data does not result in the loss of too much information.

```
iris |>
  select(-Species) |>
  cor()
##              Sepal.Length Sepal.Width Petal.Length
## Sepal.Length       1.0000     -0.1176       0.8718
## Sepal.Width       -0.1176      1.0000      -0.4284
## Petal.Length       0.8718     -0.4284       1.0000
## Petal.Width        0.8179     -0.3661       0.9629
##              Petal.Width
## Sepal.Length      0.8179
## Sepal.Width      -0.3661
## Petal.Length      0.9629
## Petal.Width       1.0000
```

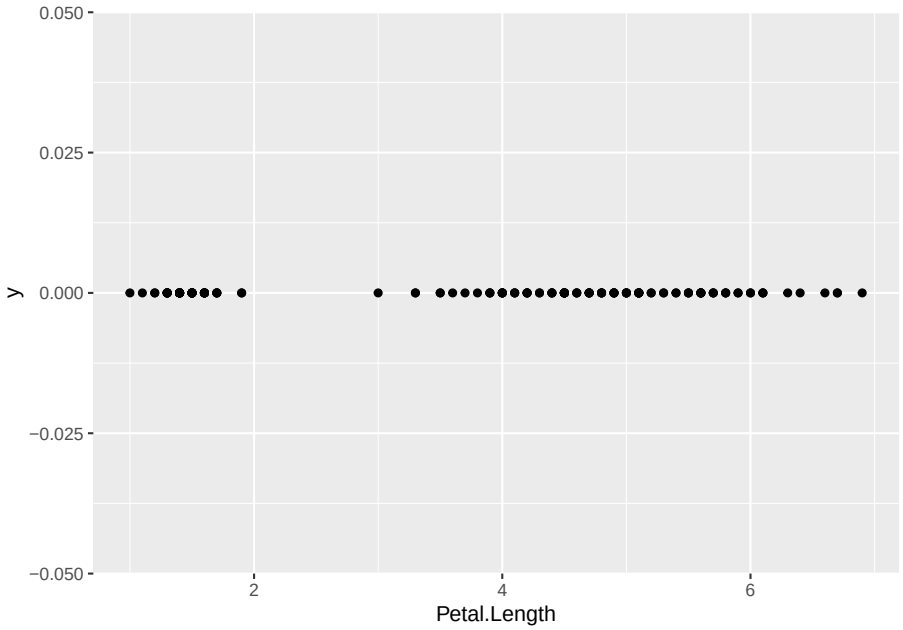
2.5.6 Density

Density estimation[29] estimate the probability density function (distribution) of a continuous variable from observed data.

Just plotting the data using points is not very helpful for a single feature.

[29] https://en.wikipedia.org/wiki/Density_estimation

```
ggplot(iris, aes(x = Petal.Length, y = 0)) + geom_point()
```

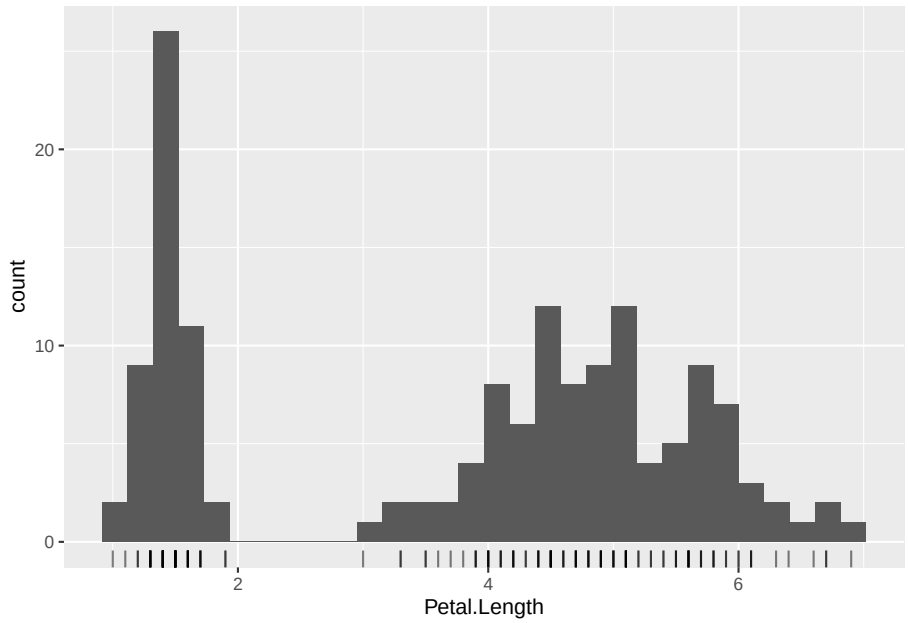


2.5.6.1 Histograms

A histograms[30] shows more about the distribution by counting how many values fall within a bin and visualizing the counts as a bar chart. We use `geom_rug` to place marks for the original data points at the bottom of the histogram.

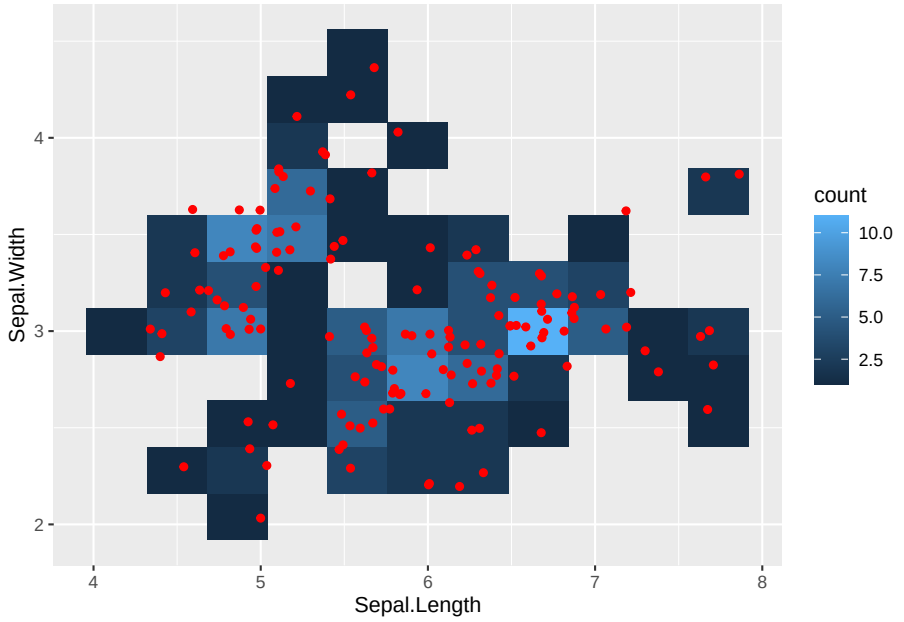
```
ggplot(iris, aes(x = Petal.Length)) +
  geom_histogram() +
  geom_rug(alpha = 1/2)
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
```

[30] https://en.wikipedia.org/wiki/Histogram

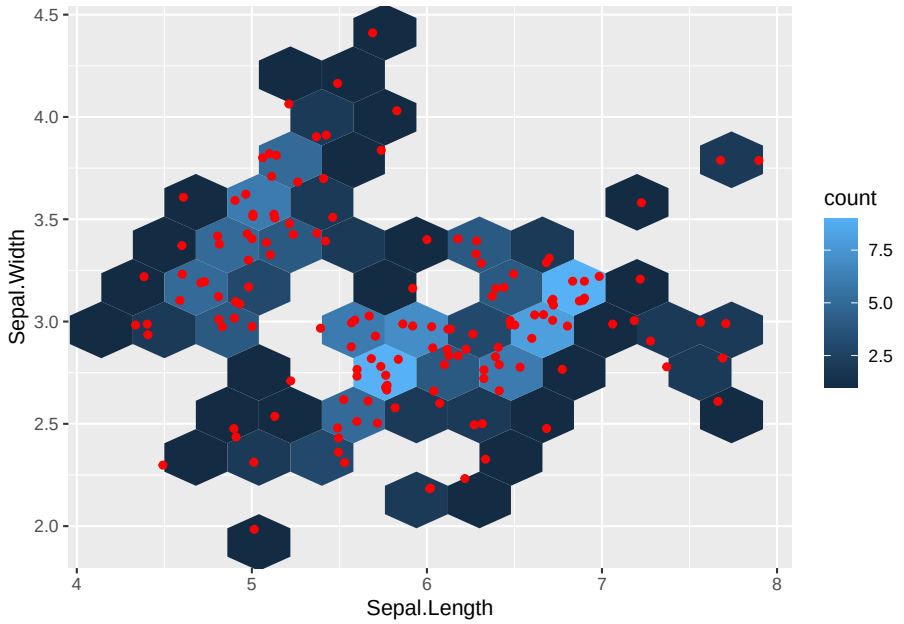


Two-dimensional distributions can be visualized using 2-d binning or hexagonal bins.

```
ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_bin2d(bins = 10) +
  geom_jitter(color = "red")
```



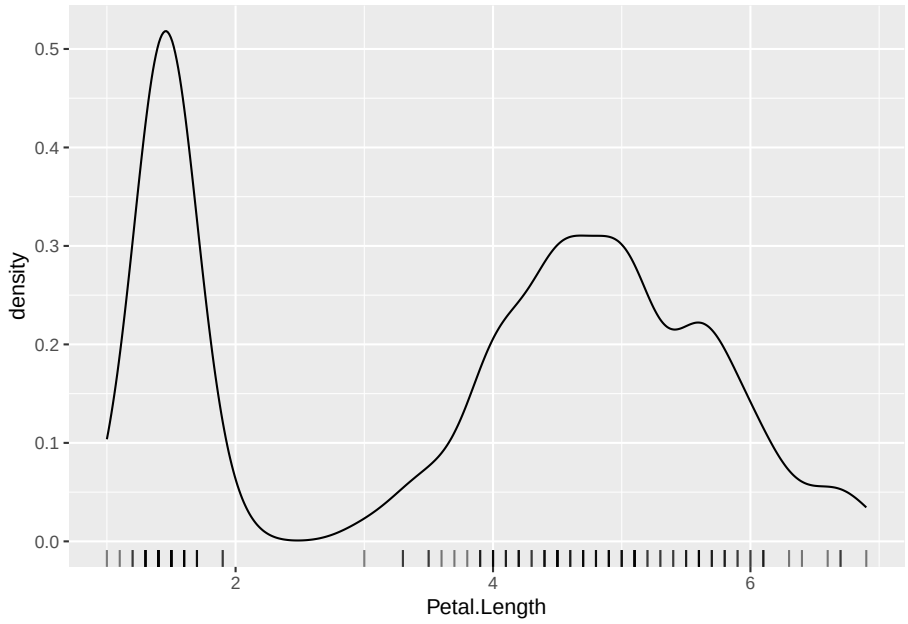

```
ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_hex(bins = 10) +
  geom_jitter(color = "red")
```



2.5.6.2 Kernel Density Estimate (KDE)

Kernel density estimation[31] is used to estimate the probability density function (distribution) of a feature. It works by replacing each value with a kernel function (often a Gaussian) and then adding them up. The result is an estimated probability density function that looks like a smoothed version of the histogram. The bandwidth (`bw`) of the kernel controls the amount of smoothing.

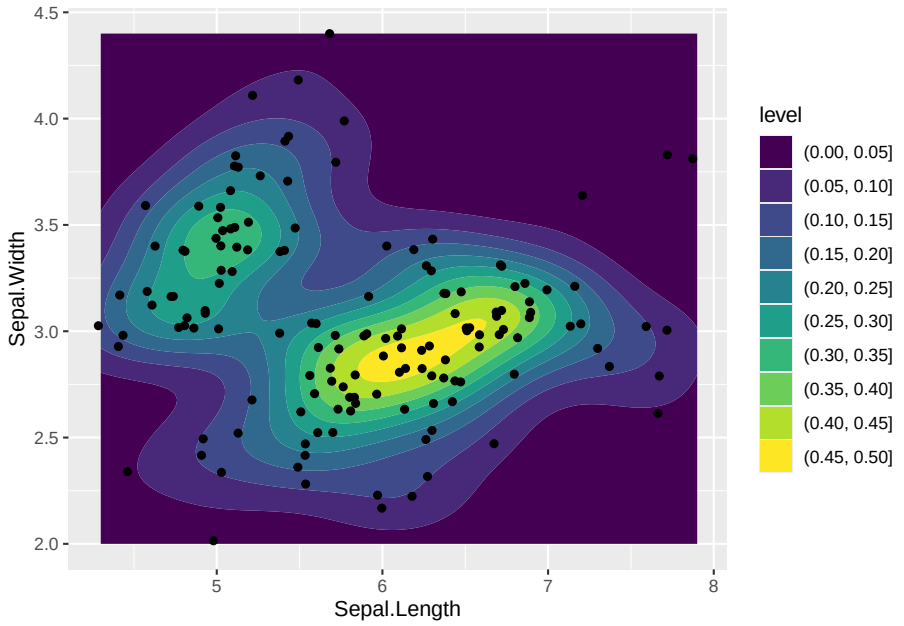
```
ggplot(iris, aes(Petal.Length)) +
  geom_density(bw = .2) +
  geom_rug(alpha = 1/2)
```



Kernel density estimates can also be done in two dimensions.

```
ggplot(iris, aes(Sepal.Length, Sepal.Width)) +
  geom_density_2d_filled() +
  geom_jitter()
```

[31] https://en.wikipedia.org/wiki/Kernel_density_estimation



2.6 Visualization

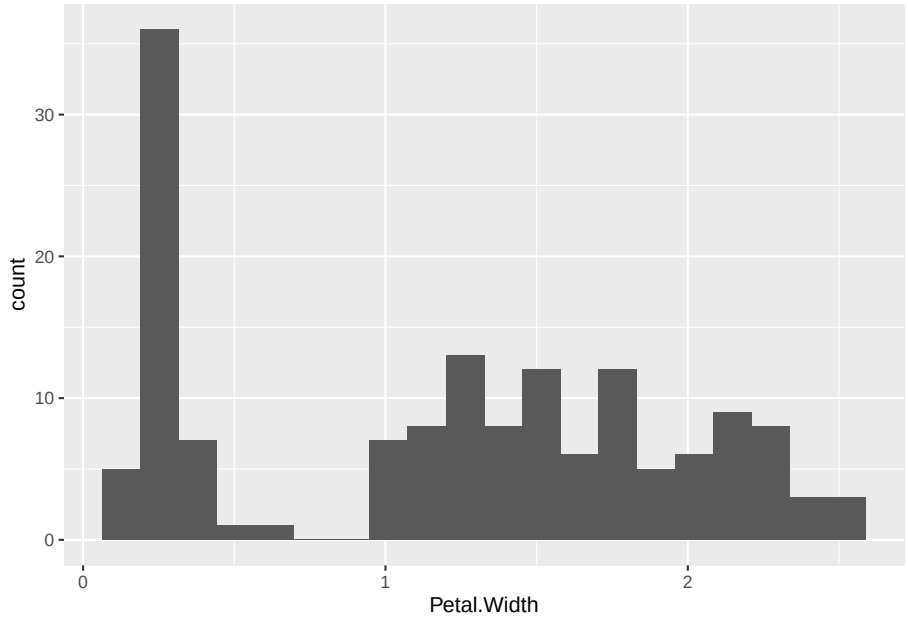
We go through the basic visualizations used when working with data. For space reasons, this chapter was moved from the printed textbook to this Data Exploration Web Chapter.[32]

2.6.1 Histogram

Histograms show the distribution of a single continuous feature.

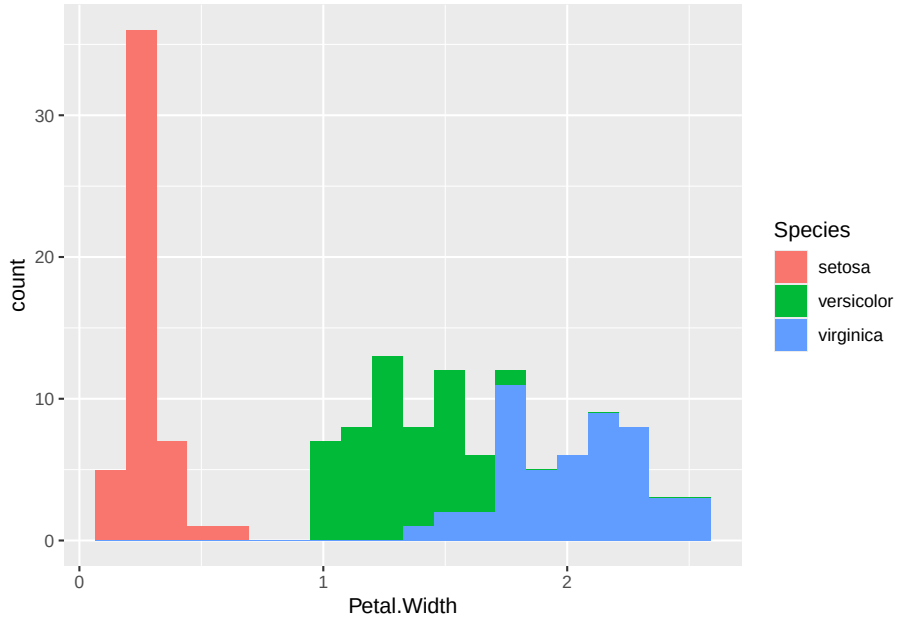
```
ggplot(iris, aes(Petal.Width)) + geom_histogram(bins = 20)
```

[32] https://www-users.cse.umn.edu/~kumar001/dmbook/data_exploration_1st_edition.pdf



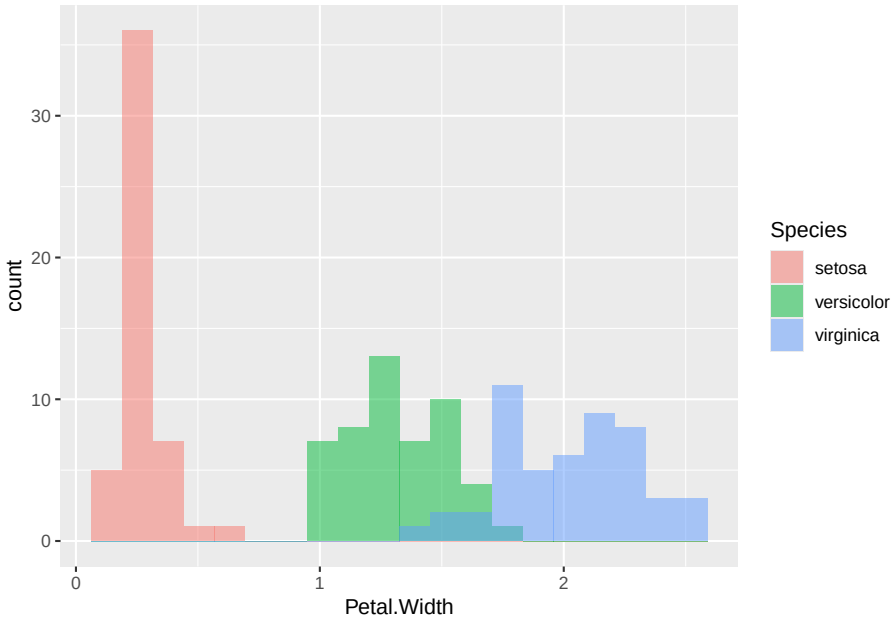
If the data contains groups, then the group information can be easily added as an aesthetic to the histogram.

```
ggplot(iris, aes(Petal.Width)) +
        geom_histogram(bins = 20, aes(fill = Species))
```



The bars appear stacked with a green blocks on top pf blue blocks. To display the three distributions behind each other, we change `position` for placement and make the bars slightly translucent using `alpha`.

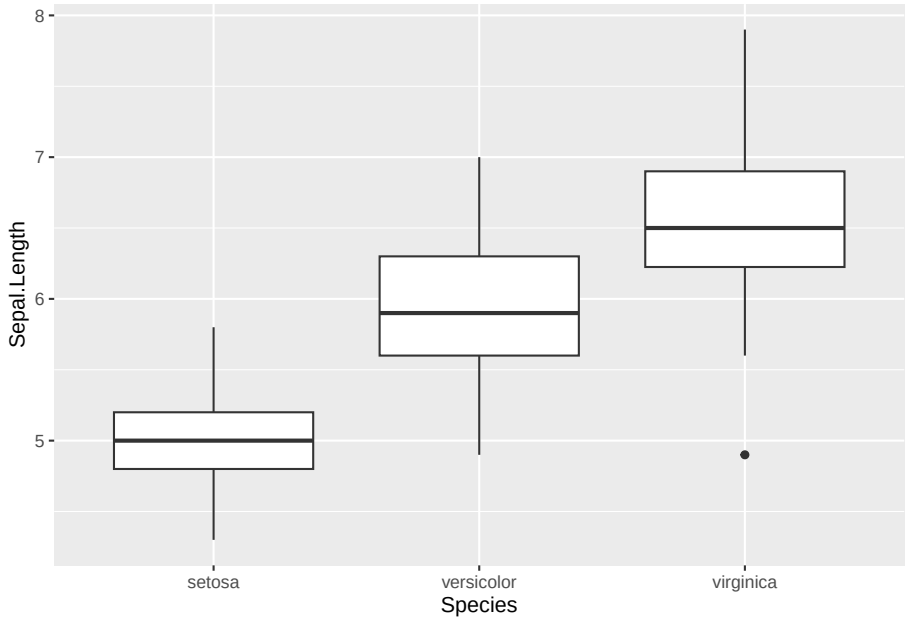
```
ggplot(iris, aes(Petal.Width)) +
        geom_histogram(bins = 20, aes(fill = Species), alpha = .5, position = 'identity')
```



2.6.2 Boxplot

Boxplots are used to compare the distribution of a feature between different groups. The horizontal line in the middle of the boxes are the group-wise medians, the boxes span the interquartile range. The whiskers (vertical lines) span typically 1.4 times the interquartile range. Points that fall outside that range are typically outliers shown as dots.

```
ggplot(iris, aes(Species, Sepal.Length)) +
  geom_boxplot()
```

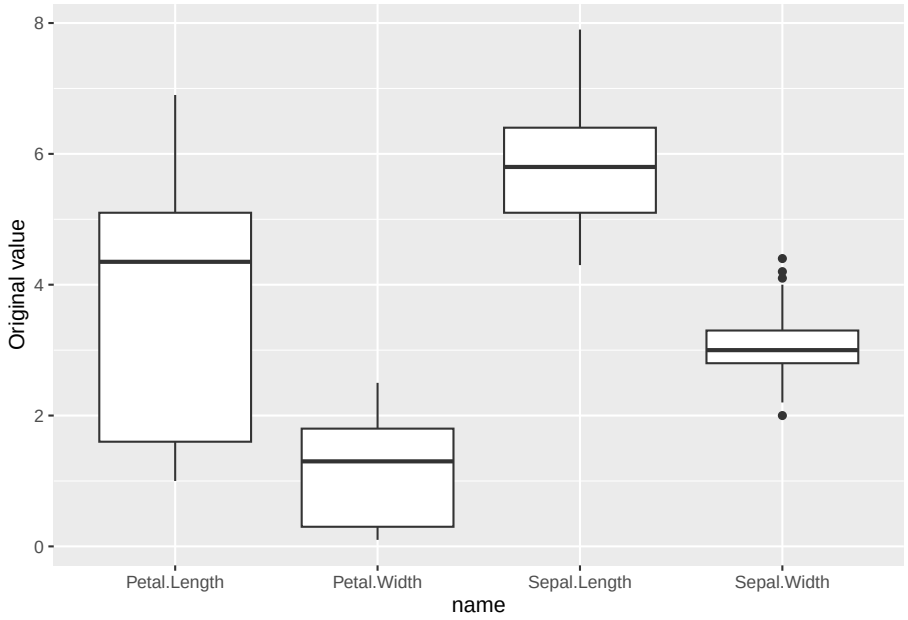


The group-wise medians can also be calculated directly.

```
iris |> group_by(Species) |>
  summarize(across(where(is.numeric), median))
## # A tibble: 3 x 5
##   Species  Sepal.Length Sepal.Width Petal.Length Petal.Width
##   <fct>           <dbl>       <dbl>        <dbl>       <dbl>
## 1 setosa            5           3.4          1.5         0.2
## 2 versico~          5.9         2.8          4.35        1.3
## 3 virgini~          6.5         3            5.55        2
```

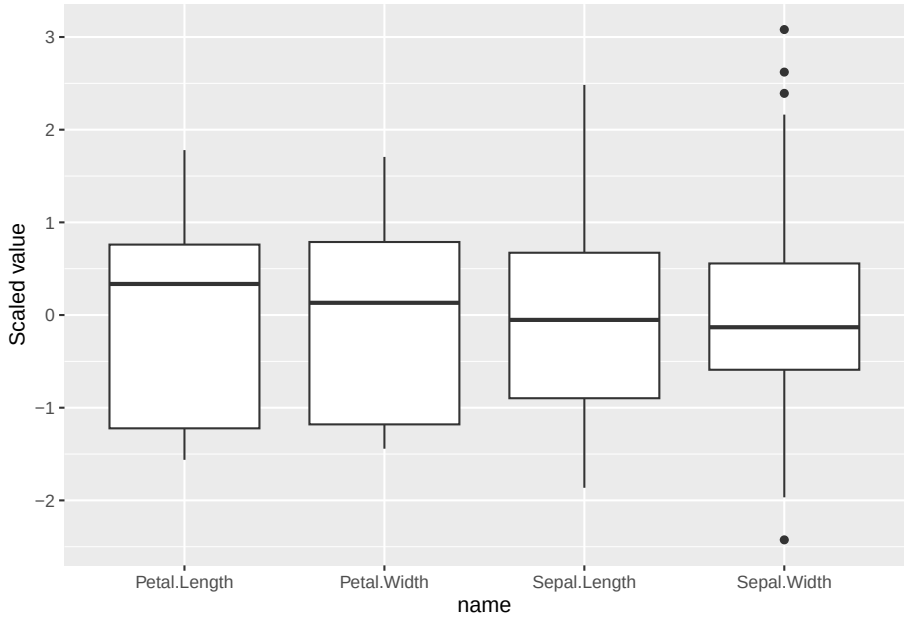
To compare the distribution of the four features using a ggplot boxplot, we first have to transform the data into long format (i.e., all feature values are combined into a single column).

```
library(tidyr)
iris_long <- iris |>
  mutate(id = row_number()) |>
  pivot_longer(1:4)

ggplot(iris_long, aes(name, value)) +
  geom_boxplot() +
  labs(y = "Original value")
```



This visualization is only useful if all features have roughly the same range. The data can be scaled first to compare the distributions.

```
library(tidyr)
iris_long_scaled <- iris |>
  scale_numeric() |>
  mutate(id = row_number()) |> pivot_longer(1:4)

ggplot(iris_long_scaled, aes(name, value)) +
  geom_boxplot() +
  labs(y = "Scaled value")
```



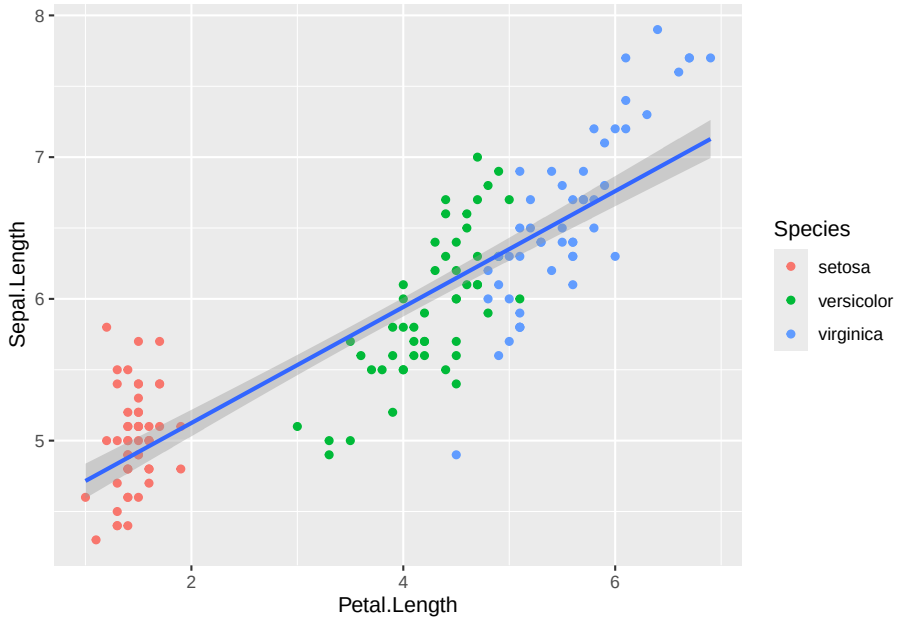
2.6.3 Scatter plot

Scatter plots show the relationship between two continuous features.

```
ggplot(iris, aes(x = Petal.Length, y = Sepal.Length)) +
  geom_point(aes(color = Species))
```

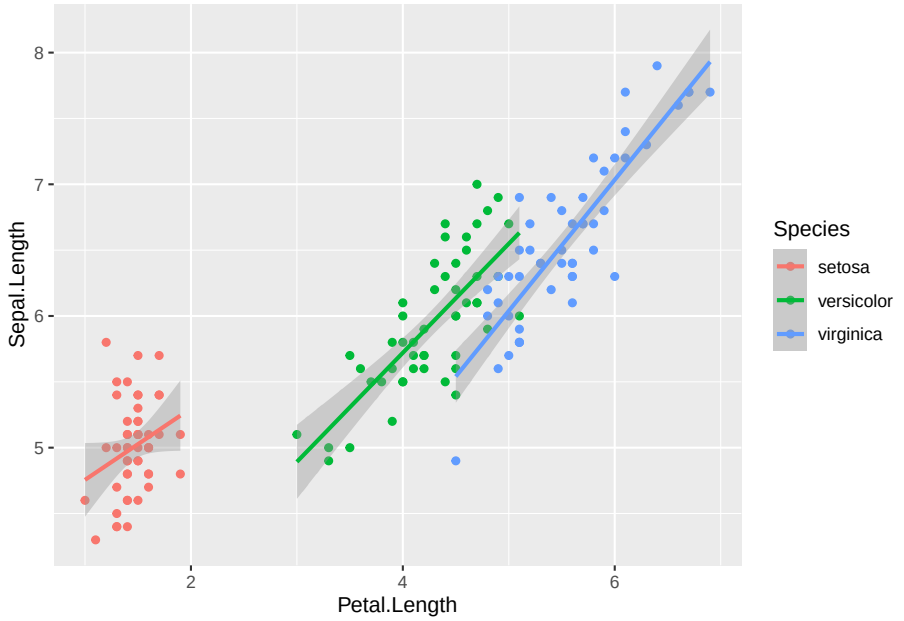
We can add a regression using `geom_smooth` with the linear model method to show that there is a linear relationship between the two variables. A confidence interval for the regression is also shown. This can be suppressed using `se = FALSE`.

```
ggplot(iris, aes(x = Petal.Length, y = Sepal.Length)) +
  geom_point(aes(color = Species)) +
  geom_smooth(method = "lm")
## `geom_smooth()` using formula = 'y ~ x'
```



We can also perform group-wise linear regression by adding the color aesthetic also to `geom_smooth`.

```
ggplot(iris, aes(x = Petal.Length, y = Sepal.Length)) +
  geom_point(aes(color = Species)) +
  geom_smooth(method = "lm", aes(color = Species))
## `geom_smooth()` using formula = 'y ~ x'
```

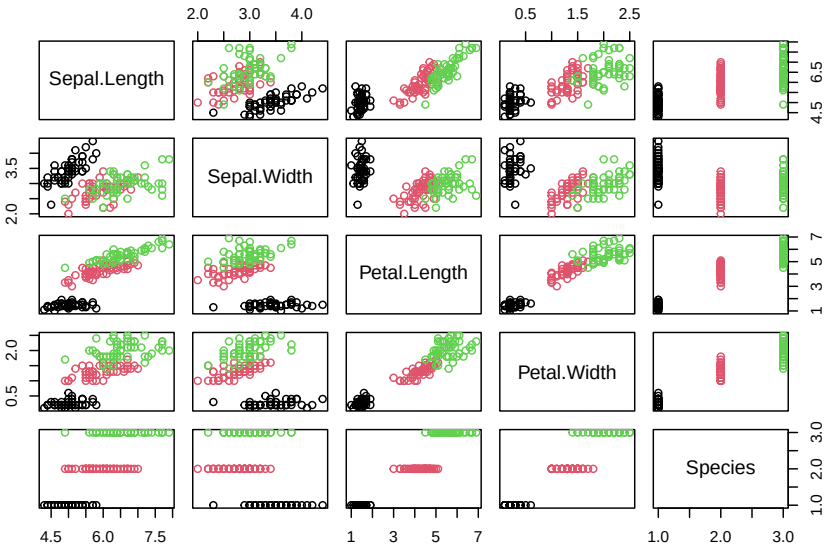


The same can be achieved by using the color aesthetic in the `qqplot` call, then it applies to all geoms.

2.6.4 Scatter Plot Matrix

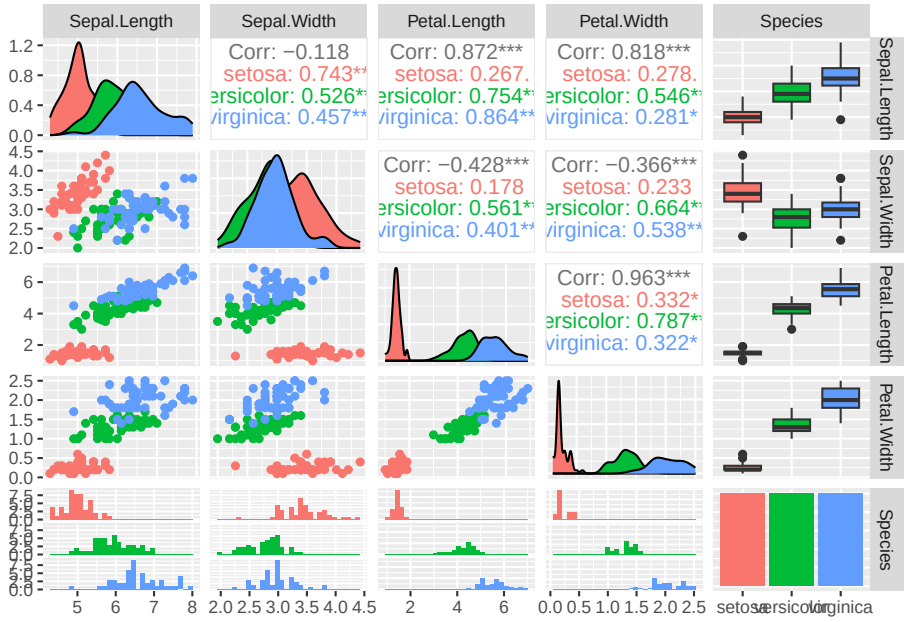
A scatter plot matrix show the relationship between all pairs of features by arranging panels in a matrix. First, lets look at a regular R-base plot.

```
pairs(iris, col = iris$Species)
```



The package `GGally` provides a way more sophisticated visualization.

```
library("GGally")
ggpairs(iris,  aes(color = Species), progress = FALSE)
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
```



Additional plots (histograms, density estimates and box plots) and correlation coefficients are shown in different panels. See the Data Quality section for a description of how to interpret the different panels.

2.6.5 Matrix Visualization

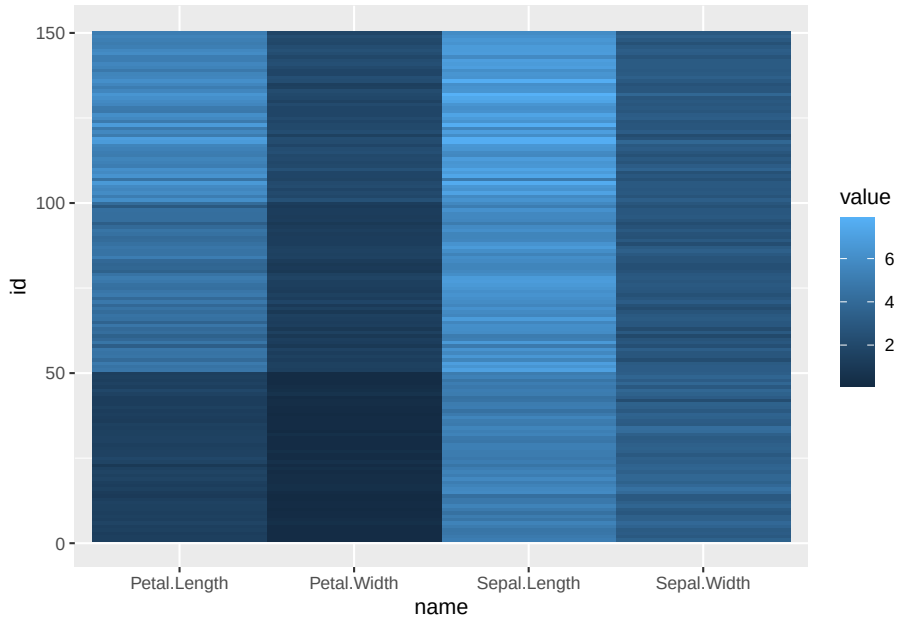
Matrix visualization shows the values in the matrix using a color scale.

```
iris_matrix <- iris |> select(-Species) |> as.matrix()
```

We need the long format for tidyverse.

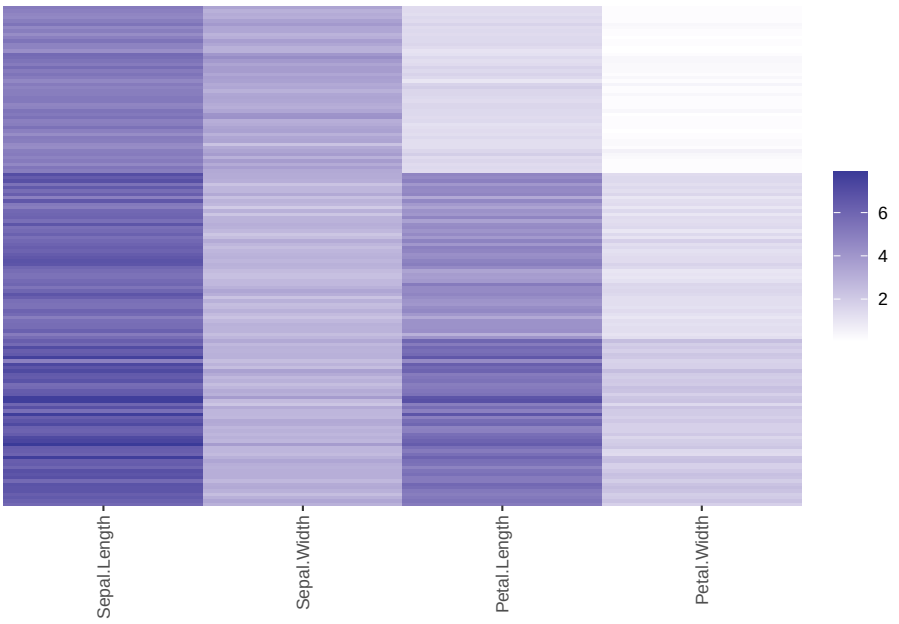
```
iris_long <- as_tibble(iris_matrix) |>
  mutate(id = row_number()) |>
  pivot_longer(1:4)

head(iris_long)
## # A tibble: 6 x 3
##      id name         value
##   <int> <chr>        <dbl>
## 1     1 Sepal.Length   5.1
## 2     1 Sepal.Width    3.5
## 3     1 Petal.Length   1.4
## 4     1 Petal.Width    0.2
## 5     2 Sepal.Length   4.9
## 6     2 Sepal.Width    3
```

```
ggplot(iris_long, aes(x = name, y = id)) +
  geom_tile(aes(fill = value))
```



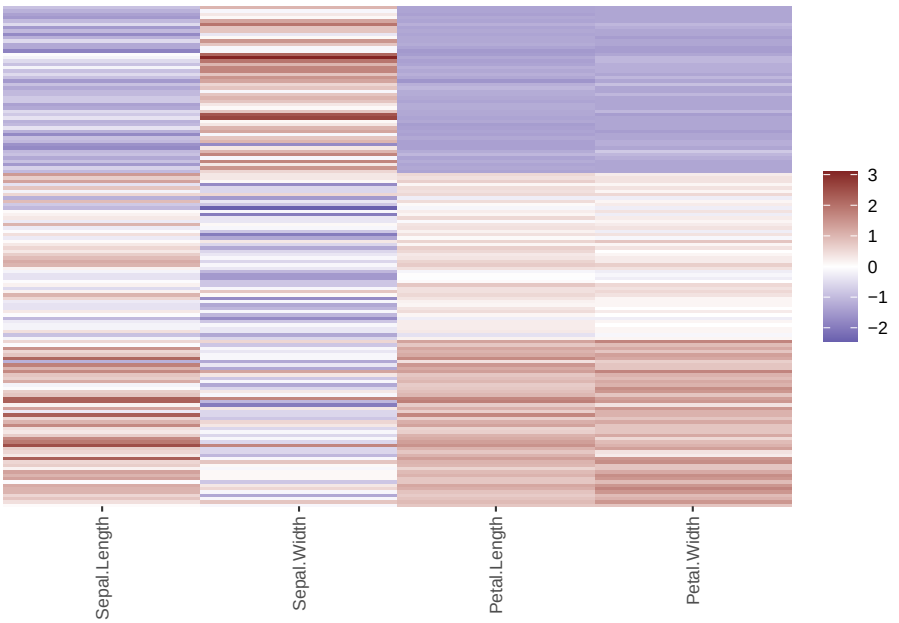
Smaller values are darker. Package `seriation` provides a simpler plotting function.

```
library(seriation)
## Registered S3 methods overwritten by 'registry':
##   method               from
##   print.registry_field proxy
##   print.registry_entry proxy
##
## Attaching package: 'seriation'
## The following object is masked from 'package:lattice':
##
##     panel.lines
ggpimage(iris_matrix, prop = FALSE)
```



We can scale the features to z-scores to make them better comparable.

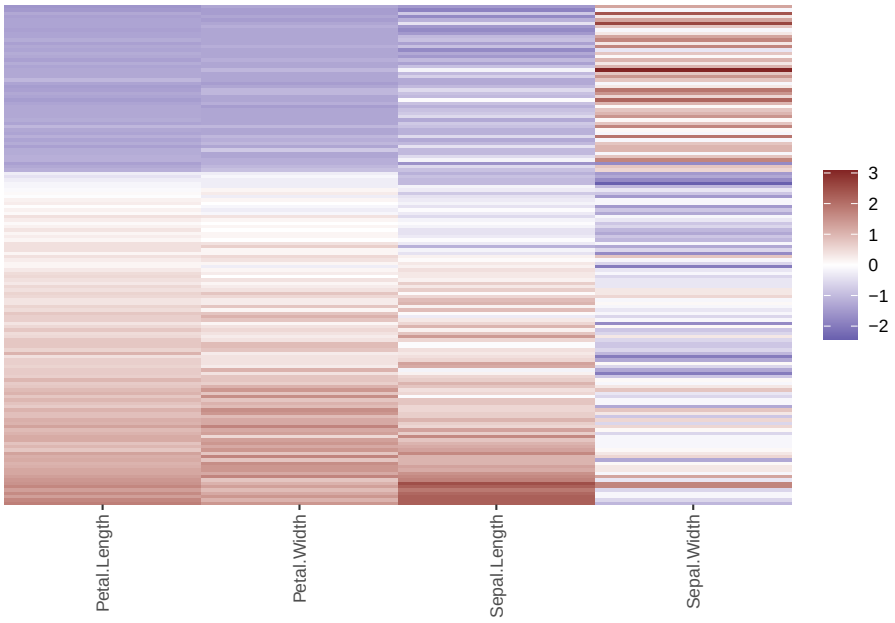
```
iris_scaled <- scale(iris_matrix)
ggpimage(iris_scaled, prop = FALSE)
```



This reveals red and blue blocks. Each row is a flower and the flowers in the Iris dataset are sorted by species. The blue blocks for the top 50 flowers show that these flowers are smaller than average for all but Sepal.Width and the red blocks show that the bottom 50 flowers are larger for most features.

Often, reordering data matrices help with visualization. A reordering technique is called seriation. Ir reorders rows and columns to place more similar points closer together.

```
ggpimage(iris_scaled, order = seriate(iris_scaled), prop = FALSE)
```



We see that the rows (flowers) are organized from very blue to very red and the features are reordered to move Sepal.Width all the way to the right because it is very different from the other features.

2.6.6 Correlation Matrix

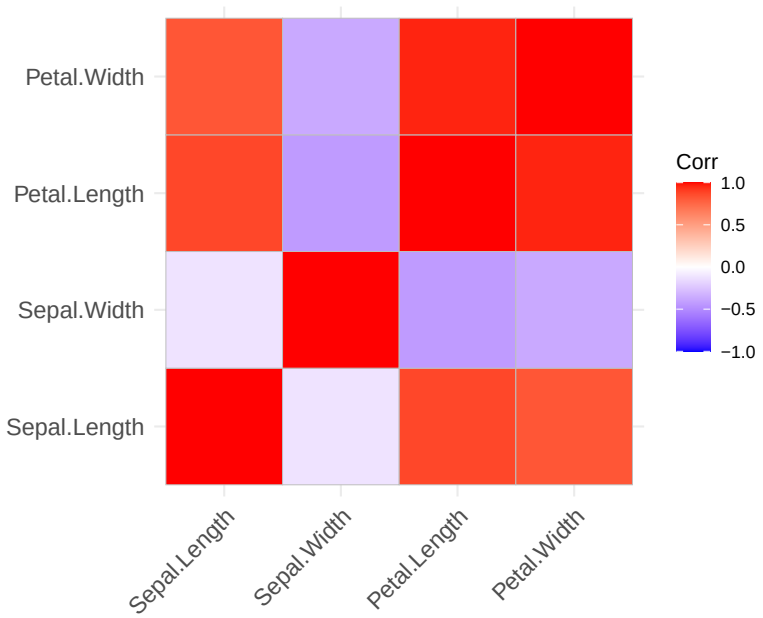
A correlation matrix contains the correlation between features.

```
cm1 <- iris |>
  select(-Species) |>
  as.matrix() |>
  cor()
cm1
##              Sepal.Length Sepal.Width Petal.Length
## Sepal.Length       1.0000     -0.1176       0.8718
## Sepal.Width       -0.1176      1.0000      -0.4284
```

```
## Petal.Length       0.8718      -0.4284       1.0000
## Petal.Width        0.8179      -0.3661       0.9629
##              Petal.Width
## Sepal.Length      0.8179
## Sepal.Width      -0.3661
## Petal.Length      0.9629
## Petal.Width       1.0000
```

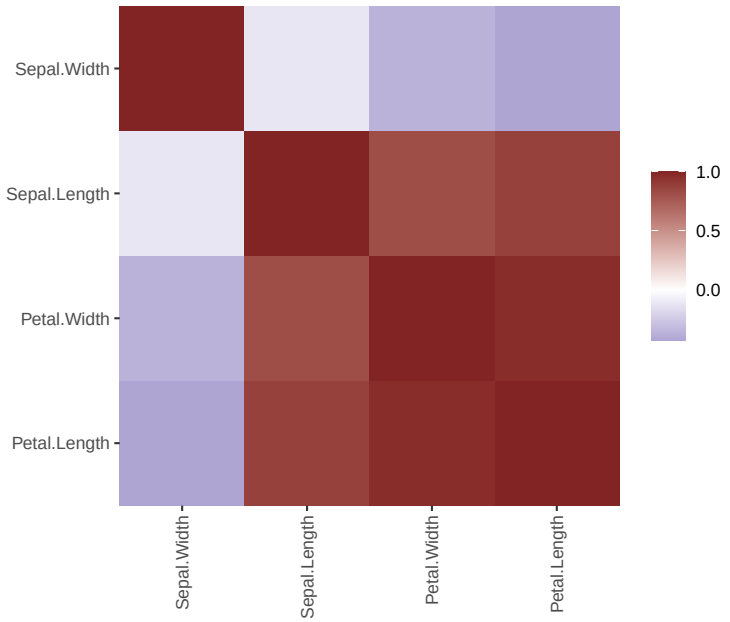
Package `ggcorrplot` provides a visualization for correlation matrices.

```
library(ggcorrplot)
ggcorrplot(cm1)
```



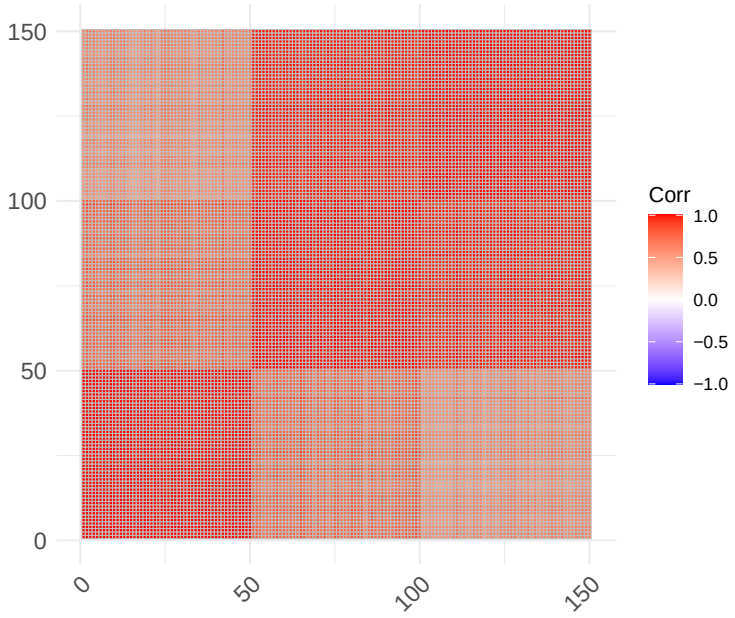
Package `seriation` provides a reordered version for this plot using a heatmap.

```
gghmap(cm1, prop = TRUE)
```



Correlations can also be calculates between objects by transposing the data matrix.

```
cm2 <- iris |>
  select(-Species) |>
  as.matrix() |>
  t() |>
  cor()

ggcorrplot(cm2)
```



Object-to-object correlations can be used as a measure of similarity. The dark red blocks indicate different species.

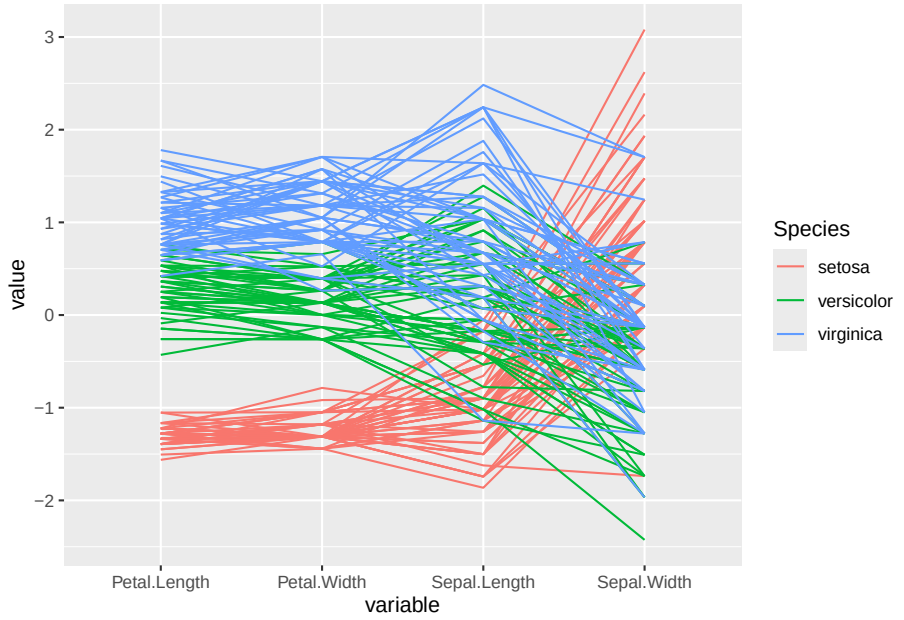
2.6.7 Parallel Coordinates Plot

Parallel coordinate plots can visualize several features in a single plot. Lines connect the values for each object (flower).

```
library(GGally)
ggparcoord(iris, columns = 1:4, groupColumn = 5)
```

The plot can be improved by reordering the variables to place correlated features next to each other.

```
o <- seriate(as.dist(1-cor(iris[,1:4])), method = "BBURCG")
get_order(o)
## Petal.Length  Petal.Width Sepal.Length  Sepal.Width
##            3            4            1            2
ggparcoord(iris,
           columns = as.integer(get_order(o)),
           groupColumn = 5)
```



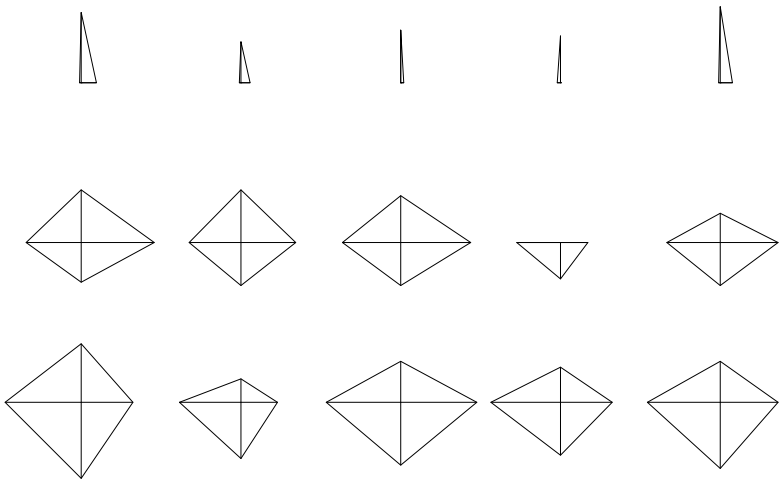
2.6.8 Star Plot

Star plots are a type of radar chart[33] to visualize three or more quantitative variables represented on axes starting from the plot's origin. R-base offers a simple star plot. We plot the first 5 flowers of each species.

```
flowers_5 <- iris[c(1:5, 51:55, 101:105), ]
flowers_5
## # A tibble: 15 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>
##  1          5.1         3.5          1.4         0.2 setosa
##  2          4.9         3            1.4         0.2 setosa
##  3          4.7         3.2          1.3         0.2 setosa
##  4          4.6         3.1          1.5         0.2 setosa
##  5          5           3.6          1.4         0.2 setosa
##  6          7           3.2          4.7         1.4 versic~
##  7          6.4         3.2          4.5         1.5 versic~
##  8          6.9         3.1          4.9         1.5 versic~
##  9          5.5         2.3          4           1.3 versic~
## 10          6.5         2.8          4.6         1.5 versic~
## 11          6.3         3.3          6           2.5 virgin~
## 12          5.8         2.7          5.1         1.9 virgin~
## 13          7.1         3            5.9         2.1 virgin~
```

[33] https://en.wikipedia.org/wiki/Radar_chart

```
## 14          6.3         2.9          5.6         1.8 virgin~
## 15          6.5         3            5.8         2.2 virgin~
stars(flowers_5[, 1:4], ncol = 5)
```



2.6.9 More Visualizations

A well organized collection of visualizations with code can be found at The R Graph Gallery[34].

2.7 Exercises*

The R package **palmerpenguins** contains measurements for penguin of different species from the Palmer Archipelago, Antarctica. Install the package. It provides a CSV file which can be read in the following way:

```
library("palmerpenguins")
penguins <- read_csv(path_to_file("penguins.csv"))
## Rows: 344 Columns: 8
## -- Column specification --------------------------------------
## Delimiter: ","
## chr (3): species, island, sex
## dbl (5): bill_length_mm, bill_depth_mm, flipper_length_m...
##
```

[34] https://www.r-graph-gallery.com/

```
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
head(penguins)
## # A tibble: 6 x 8
##   species island    bill_length_mm bill_depth_mm
##   <chr>   <chr>              <dbl>         <dbl>
## 1 Adelie  Torgersen           39.1          18.7
## 2 Adelie  Torgersen           39.5          17.4
## 3 Adelie  Torgersen           40.3          18
## 4 Adelie  Torgersen           NA            NA
## 5 Adelie  Torgersen           36.7          19.3
## 6 Adelie  Torgersen           39.3          20.6
## # i 4 more variables: flipper_length_mm <dbl>,
## #   body_mass_g <dbl>, sex <chr>, year <dbl>
```

Create in RStudio a new R Markdown document. Apply the code in the sections of this chapter to the data set to answer the following questions.

1. What is the scale of measurement for each column?
2. Are there missing values in the data? How much data is missing?
3. Compute and discuss basic statistics.
4. Group the penguins by species, island or sex. What can you find out?
5. Can you identify correlations?
6. Apply histograms, boxplots, scatterplots and correlation matrix visualization. What do the visualizations show.

Make sure your markdown document contains now a well formatted report. Use the `Knit` button to create a HTML document.

Chapter 3

Classification: Basic Concepts

This chapter introduces decision trees for classification and discusses how models are built and evaluated.

The corresponding chapter of the data mining textbook is available online: Chapter 3: Classification: Basic Concepts and Techniques.[1]

Packages Used in this Chapter

```
pkgs <- c("caret", "FSelector", "lattice", "mlbench",
          "palmerpenguins", "party", "pROC", "rpart",
          "rpart.plot", "tidyverse")

pkgs_install <- pkgs[!(pkgs %in% installed.packages()[,"Package"])]
if(length(pkgs_install)) install.packages(pkgs_install)
```

The packages used for this chapter are:

- *caret* (Kuhn 2023)
- *FSelector* (Romanski, Kotthoff, and Schratz 2023)
- *lattice* (Sarkar 2023)
- *mlbench* (Leisch and Dimitriadou 2024)
- *palmerpenguins* (Horst, Hill, and Gorman 2022)
- *party* (Hothorn et al. 2024)
- *pROC* (Robin et al. 2023)
- *rpart* (Therneau and Atkinson 2023)
- *rpart.plot* (Milborrow 2024)

[1] https://www-users.cs.umn.edu/~kumar001/dmbook/ch3_classification.pdf

- *tidyverse* (Wickham 2023c)

In the examples in this book, we use the popular machine learning R package `caret`[2]. It makes preparing training sets, building classification (and regression) models and evaluation easier. A great cheat sheet can be found here[3].

A newer R framework for machine learning is `tidymodels`[4], a set of packages that integrate more naturally with tidyverse. Using tidymodels, or any other framework (e.g., Python's scikit-learn[5]) should be relatively easy after learning the concepts using caret.

3.1 Basic Concepts

Classification is a machine learning task with the goal to learn a predictive function of the form

$$y = f(\mathbf{x}),$$

where $\mathbf{x}$ is called the attribute set and y the class label. The attribute set consists of feature which describe an object. These features can be measured using any scale (i.e., nominal, interval, ...). The class label is a nominal attribute. It it is a binary attribute, then the problem is called a binary classification problem.

Classification learns the classification model from training data where both the features and the correct class label are available. This is why it is called a supervised learning problem[6].

A related supervised learning problem is regression[7], where y is a number instead of a label. Linear regression is a very popular supervised learning model which is taught in almost any introductory statistics course. Code examples for regression are available in the extra Chapter Regression*.

This chapter will introduce decision trees, model evaluation and comparison, feature selection, and then explore methods to handle the class imbalance problem.

You can read the free sample chapter from the textbook (Tan, Steinbach, and Kumar 2005): Chapter 3. Classification: Basic Concepts and Techniques[8]

[2] https://topepo.github.io/caret/
[3] https://ugoproto.github.io/ugo_r_doc/pdf/caret.pdf
[4] https://www.tidymodels.org/
[5] https://scikit-learn.org/
[6] https://en.wikipedia.org/wiki/Supervised_learning
[7] https://en.wikipedia.org/wiki/Linear_regression
[8] https://www-users.cs.umn.edu/~kumar001/dmbook/ch3_classification.pdf

3.2 General Framework for Classification

Supervised learning has two steps:

1. Induction: Training a model on **training data** with known class labels.
2. Deduction: Predicting class labels for new data.

We often test model by predicting the class for data where we know the correct label. We test the model on **test data** with known labels and can then calculate the error by comparing the prediction with the known correct label. It is tempting to measure how well the model has learned the training data, by testing it on the training data. The error on the training data is called **resubstitution error.** It does not help us to find out if the model generalizes well to new data that was not part of the training.

We typically want to evaluate how well the model generalizes new data, so it is important that the test data and the training data do not overlap. We call the error on proper test data the **generalization error.**

This chapter builds up the needed concepts. A complete example of how to perform model selection and estimate the generalization error is in the section Hyperparameter Tuning.

3.2.1 The Zoo Dataset

To demonstrate classification, we will use the Zoo dataset which is included in the R package **mlbench** (you may have to install it). The Zoo dataset containing 17 (mostly logical) variables for 101 animals as a data frame with 17 columns (hair, feathers, eggs, milk, airborne, aquatic, predator, toothed, backbone, breathes, venomous, fins, legs, tail, domestic, catsize, type). The first 16 columns represent the feature vector $\mathbf{x}$ and the last column called type is the class label y. We convert the data frame into a tidyverse tibble (optional).

```
data(Zoo, package="mlbench")
head(Zoo)
##           hair feathers  eggs  milk airborne aquatic
## aardvark  TRUE    FALSE FALSE  TRUE    FALSE   FALSE
## antelope  TRUE    FALSE FALSE  TRUE    FALSE   FALSE
## bass     FALSE    FALSE  TRUE FALSE    FALSE    TRUE
## bear      TRUE    FALSE FALSE  TRUE    FALSE   FALSE
## boar      TRUE    FALSE FALSE  TRUE    FALSE   FALSE
## buffalo   TRUE    FALSE FALSE  TRUE    FALSE   FALSE
##          predator toothed backbone breathes venomous  fins
## aardvark     TRUE    TRUE     TRUE     TRUE    FALSE FALSE
## antelope    FALSE    TRUE     TRUE     TRUE    FALSE FALSE
## bass         TRUE    TRUE     TRUE    FALSE    FALSE  TRUE
## bear         TRUE    TRUE     TRUE     TRUE    FALSE FALSE
## boar         TRUE    TRUE     TRUE     TRUE    FALSE FALSE
```

```
## buffalo     FALSE    TRUE     TRUE     TRUE    FALSE FALSE
##          legs  tail domestic catsize   type
## aardvark    4 FALSE    FALSE    TRUE mammal
## antelope    4  TRUE    FALSE    TRUE mammal
## bass        0  TRUE    FALSE   FALSE   fish
## bear        4 FALSE    FALSE    TRUE mammal
## boar        4  TRUE    FALSE    TRUE mammal
## buffalo     4  TRUE    FALSE    TRUE mammal
```

Note: data.frames in R can have row names. The Zoo data set uses the animal name as the row names. tibbles from `tidyverse` do not support row names. To keep the animal name you can add a column with the animal name.

```
library(tidyverse)
as_tibble(Zoo, rownames = "animal")
## # A tibble: 101 x 18
##    animal   hair  feathers eggs  milk  airborne aquatic
##    <chr>    <lgl> <lgl>    <lgl> <lgl> <lgl>    <lgl>
##  1 aardvark TRUE  FALSE    FALSE TRUE  FALSE    FALSE
##  2 antelope TRUE  FALSE    FALSE TRUE  FALSE    FALSE
##  3 bass     FALSE FALSE    TRUE  FALSE FALSE    TRUE
##  4 bear     TRUE  FALSE    FALSE TRUE  FALSE    FALSE
##  5 boar     TRUE  FALSE    FALSE TRUE  FALSE    FALSE
##  6 buffalo  TRUE  FALSE    FALSE TRUE  FALSE    FALSE
##  7 calf     TRUE  FALSE    FALSE TRUE  FALSE    FALSE
##  8 carp     FALSE FALSE    TRUE  FALSE FALSE    TRUE
##  9 catfish  FALSE FALSE    TRUE  FALSE FALSE    TRUE
## 10 cavy     TRUE  FALSE    FALSE TRUE  FALSE    FALSE
## # i 91 more rows
## # i 11 more variables: predator <lgl>, toothed <lgl>,
## #   backbone <lgl>, breathes <lgl>, venomous <lgl>,
## #   fins <lgl>, legs <int>, tail <lgl>, domestic <lgl>,
## #   catsize <lgl>, type <fct>
```

You will have to remove the animal column before learning a model! In the following I use the data.frame.

I translate all the TRUE/FALSE values into factors (nominal). This is often needed for building models. Always check `summary()` to make sure the data is ready for model learning.

```
Zoo <- Zoo |>
  mutate(across(where(is.logical),
         function (x) factor(x, levels = c(TRUE, FALSE)))) |>
  mutate(across(where(is.character), factor))

summary(Zoo)
```

```
##     hair     feathers     eggs       milk     airborne
##  TRUE :43   TRUE :20   TRUE :59   TRUE :41   TRUE :24
##  FALSE:58   FALSE:81   FALSE:42   FALSE:60   FALSE:77
##
##
##
##
##
##   aquatic    predator   toothed    backbone   breathes
##  TRUE :36   TRUE :56   TRUE :61   TRUE :83   TRUE :80
##  FALSE:65   FALSE:45   FALSE:40   FALSE:18   FALSE:21
##
##
##
##
##
##   venomous     fins          legs         tail     domestic
##  TRUE : 8   TRUE :17   Min.   :0.00   TRUE :75   TRUE :13
##  FALSE:93   FALSE:84   1st Qu.:2.00   FALSE:26   FALSE:88
##                        Median :4.00
##                        Mean   :2.84
##                        3rd Qu.:4.00
##                        Max.   :8.00
##
##   catsize              type
##  TRUE :44   mammal       :41
##  FALSE:57   bird         :20
##             reptile      : 5
##             fish         :13
##             amphibian    : 4
##             insect       : 8
##             mollusc.et.al:10
```

3.3 Decision Tree Classifiers

We use here the recursive partitioning implementation which follows largely CART and uses the Gini index to make splitting decisions and early stopping (also called pre-pruning).

```
library(rpart)
```

3.3.1 Create Tree

We create first a tree with the default settings (see `? rpart.control`).

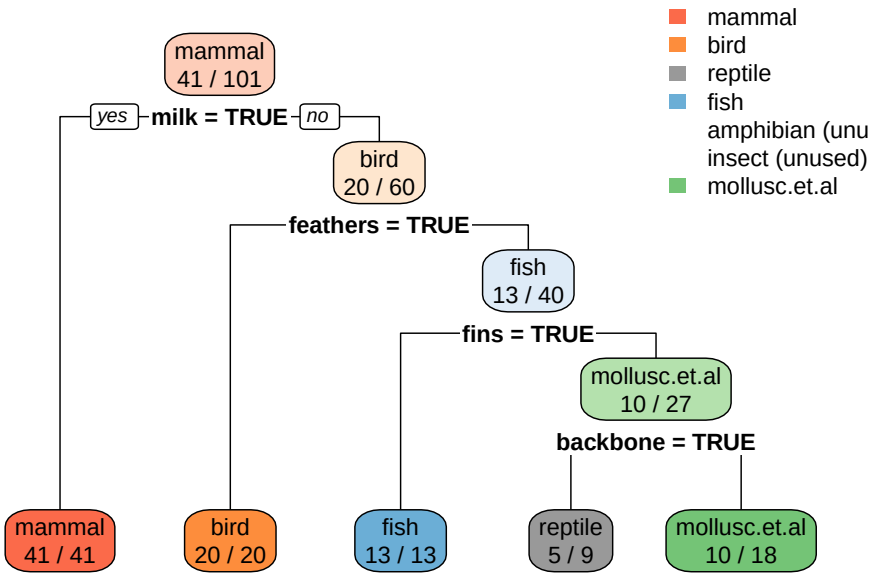
```
tree_default <- Zoo |>
  rpart(type ~ ., data = _)
tree_default
## n= 101
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 101 60 mammal (0.41 0.2 0.05 0.13 0.04 0.079 0.099)
##    2) milk=TRUE 41  0 mammal (1 0 0 0 0 0 0) *
##    3) milk=FALSE 60 40 bird (0 0.33 0.083 0.22 0.067 0.13 0.17)
##      6) feathers=TRUE 20  0 bird (0 1 0 0 0 0 0) *
##      7) feathers=FALSE 40 27 fish (0 0 0.12 0.33 0.1 0.2 0.25)
##       14) fins=TRUE 13  0 fish (0 0 0 1 0 0 0) *
##       15) fins=FALSE 27 17 mollusc.et.al (0 0 0.19 0 0.15 0.3 0.37)
##         30) backbone=TRUE 9  4 reptile (0 0 0.56 0 0.44 0 0) *
##         31) backbone=FALSE 18  8 mollusc.et.al (0 0 0 0 0 0.44 0.56) *
```

Notes:

- `|>` supplies the data for `rpart`. Since `data` is not the first argument of `rpart`, the syntax `data = _` is used to specify where the data in `Zoo` goes. The call is equivalent to `tree_default <- rpart(type ~ ., data = Zoo)`.
- The formula models the `type` variable by all other features represented by a single period (`.`).
- The class variable needs to be a factor to be recognized as nominal or rpart will create a regression tree instead of a decision tree. Use `as.factor()` on the column with the class label first, if necessary.

We can plot the resulting decision tree.

```
library(rpart.plot)
rpart.plot(tree_default, extra = 2)
```



Note: `extra=2` prints for each leaf node the number of correctly classified objects from data and the total number of objects from the training data falling into that node (correct/total).

3.3.2 Make Predictions for New Data

I will make up my own animal: A lion with feathered wings.

```
my_animal <- tibble(hair = TRUE, feathers = TRUE, eggs = FALSE,
  milk = TRUE, airborne = TRUE, aquatic = FALSE, predator = TRUE,
  toothed = TRUE, backbone = TRUE, breathes = TRUE,
  venomous = FALSE, fins = FALSE, legs = 4, tail = TRUE,
  domestic = FALSE, catsize = FALSE, type = NA)
```

The data types need to match the original data so we change the columns to be factors like in the training set.

```
my_animal <- my_animal |>
  mutate(across(where(is.logical),
                function(x) factor(x, levels = c(TRUE, FALSE))))
my_animal
## # A tibble: 1 x 17
##   hair  feathers eggs  milk  airborne aquatic predator
##   <fct> <fct>    <fct> <fct> <fct>    <fct>   <fct>
## 1 TRUE  TRUE     FALSE TRUE  TRUE     FALSE   TRUE
## # i 10 more variables: toothed <fct>, backbone <fct>,
```

```
## #   breathes <fct>, venomous <fct>, fins <fct>, legs <dbl>,
## #   tail <fct>, domestic <fct>, catsize <fct>, type <fct>
```

Next, we make a prediction using the default tree

```
predict(tree_default , my_animal, type = "class")
##      1
## mammal
## 7 Levels: mammal bird reptile fish amphibian ... mollusc.et.al
```

3.3.3 Manual Calculation of the Resubstitution Error

We will calculate error of the model on the training data manually first, so we see how it is calculated.

```
predict(tree_default, Zoo) |> head ()
##          mammal bird reptile fish amphibian insect
## aardvark      1    0       0    0         0      0
## antelope      1    0       0    0         0      0
## bass          0    0       0    1         0      0
## bear          1    0       0    0         0      0
## boar          1    0       0    0         0      0
## buffalo       1    0       0    0         0      0
##          mollusc.et.al
## aardvark             0
## antelope             0
## bass                 0
## bear                 0
## boar                 0
## buffalo              0
pred <- predict(tree_default, Zoo, type="class")
head(pred)
## aardvark antelope     bass     bear     boar  buffalo
##   mammal   mammal     fish   mammal   mammal   mammal
## 7 Levels: mammal bird reptile fish amphibian ... mollusc.et.al
```

We can easily tabulate the true and predicted labels to create a confusion matrix.

```
confusion_table <- with(Zoo, table(type, pred))
confusion_table
##                pred
## type            mammal bird reptile fish amphibian insect
##   mammal            41    0       0    0         0      0
##   bird               0   20       0    0         0      0
##   reptile            0    0       5    0         0      0
##   fish               0    0       0   13         0      0
##   amphibian          0    0       4    0         0      0
```

```
##   insect              0    0       0    0         0      0
##   mollusc.et.al       0    0       0    0         0      0
##                 pred
## type             mollusc.et.al
##   mammal                     0
##   bird                       0
##   reptile                    0
##   fish                       0
##   amphibian                  0
##   insect                     8
##   mollusc.et.al             10
```

The counts in the diagonal are correct predictions. Off-diagonal counts represent errors (i.e., confusions).

We can summarize the confusion matrix using the accuracy measure.

```
correct <- confusion_table |> diag() |> sum()
correct
## [1] 89
error <- confusion_table |> sum() - correct
error
## [1] 12
accuracy <- correct / (correct + error)
accuracy
## [1] 0.8812
```

Here is the accuracy calculation as a simple function.

```
accuracy <- function(truth, prediction) {
    tbl <- table(truth, prediction)
    sum(diag(tbl))/sum(tbl)
}

accuracy(Zoo |> pull(type), pred)
## [1] 0.8812
```

3.3.4 Confusion Matrix using Caret

The caret package provides a convenient way to create and analyze a confusion table including many useful statistics.

```
library(caret)
confusionMatrix(data = pred,
                reference = Zoo |> pull(type))
## Confusion Matrix and Statistics
##
##                Reference
```

```
## Prediction      mammal bird reptile fish amphibian insect
##   mammal            41    0       0    0         0      0
##   bird               0   20       0    0         0      0
##   reptile            0    0       5    0         4      0
##   fish               0    0       0   13         0      0
##   amphibian          0    0       0    0         0      0
##   insect             0    0       0    0         0      0
##   mollusc.et.al      0    0       0    0         0      8
##                Reference
## Prediction      mollusc.et.al
##   mammal                    0
##   bird                      0
##   reptile                   0
##   fish                      0
##   amphibian                 0
##   insect                    0
##   mollusc.et.al            10
##
## Overall Statistics
##
##                Accuracy : 0.881
##                  95% CI : (0.802, 0.937)
##     No Information Rate : 0.406
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.843
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: mammal Class: bird
## Sensitivity                  1.000       1.000
## Specificity                  1.000       1.000
## Pos Pred Value               1.000       1.000
## Neg Pred Value               1.000       1.000
## Prevalence                   0.406       0.198
## Detection Rate               0.406       0.198
## Detection Prevalence         0.406       0.198
## Balanced Accuracy            1.000       1.000
##                      Class: reptile Class: fish
## Sensitivity                  1.0000       1.000
## Specificity                  0.9583       1.000
## Pos Pred Value               0.5556       1.000
## Neg Pred Value               1.0000       1.000
```

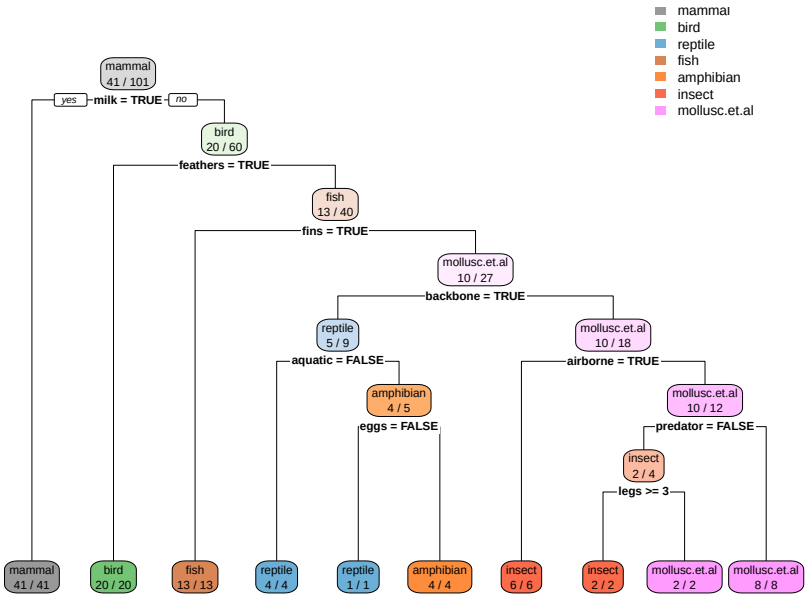
```
## Prevalence                     0.0495       0.129
## Detection Rate                 0.0495       0.129
## Detection Prevalence           0.0891       0.129
## Balanced Accuracy              0.9792       1.000
##                       Class: amphibian Class: insect
## Sensitivity                     0.0000        0.0000
## Specificity                     1.0000        1.0000
## Pos Pred Value                     NaN           NaN
## Neg Pred Value                  0.9604        0.9208
## Prevalence                      0.0396        0.0792
## Detection Rate                  0.0000        0.0000
## Detection Prevalence            0.0000        0.0000
## Balanced Accuracy               0.5000        0.5000
##                       Class: mollusc.et.al
## Sensitivity                          1.000
## Specificity                          0.912
## Pos Pred Value                       0.556
## Neg Pred Value                       1.000
## Prevalence                           0.099
## Detection Rate                       0.099
## Detection Prevalence                 0.178
## Balanced Accuracy                    0.956
```

Note: Calculating accuracy on the training data is not a good idea. Keep on reading!

3.4 Model Overfitting

We are tempted to create the largest possible tree to get the most accurate model. This can be achieved by changing the algorithms hyperparameter (parameters that change how the algorithm works). We set the complexity parameter `cp` to 0 (split even if it does not improve the tree) and we set the minimum number of observations in a node needed to split to the smallest value of 2 (see: `?rpart.control`). *Note:* This is not a good idea! As we will see later, full trees overfit the training data!

```
tree_full <- Zoo |>
  rpart(type ~ . , data = _,
        control = rpart.control(minsplit = 2, cp = 0))
rpart.plot(tree_full, extra = 2,
           roundint=FALSE,
           box.palette = list("Gy", "Gn", "Bu", "Bn",
                              "Or", "Rd", "Pu"))
```




```
tree_full
## n= 101
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 101 60 mammal (0.41 0.2 0.05 0.13 0.04 0.079 0.099)
##     2) milk=TRUE 41  0 mammal (1 0 0 0 0 0 0) *
##     3) milk=FALSE 60 40 bird (0 0.33 0.083 0.22 0.067 0.13 0.17)
##       6) feathers=TRUE 20  0 bird (0 1 0 0 0 0 0) *
##       7) feathers=FALSE 40 27 fish (0 0 0.12 0.33 0.1 0.2 0.25)
##        14) fins=TRUE 13  0 fish (0 0 0 1 0 0 0) *
##        15) fins=FALSE 27 17 mollusc.et.al (0 0 0.19 0 0.15 0.3 0.37)
##          30) backbone=TRUE 9  4 reptile (0 0 0.56 0 0.44 0 0)
##            60) aquatic=FALSE 4  0 reptile (0 0 1 0 0 0 0) *
##            61) aquatic=TRUE 5  1 amphibian (0 0 0.2 0 0.8 0 0)
##             122) eggs=FALSE 1  0 reptile (0 0 1 0 0 0 0) *
##             123) eggs=TRUE 4  0 amphibian (0 0 0 0 1 0 0) *
##          31) backbone=FALSE 18  8 mollusc.et.al (0 0 0 0 0 0.44 0.56)
##            62) airborne=TRUE 6  0 insect (0 0 0 0 0 1 0) *
##            63) airborne=FALSE 12  2 mollusc.et.al (0 0 0 0 0 0.17 0.83)
##             126) predator=FALSE 4  2 insect (0 0 0 0 0 0.5 0.5)
##               252) legs>=3 2  0 insect (0 0 0 0 0 1 0) *
##               253) legs< 3 2  0 mollusc.et.al (0 0 0 0 0 0 1) *
##             127) predator=TRUE 8  0 mollusc.et.al (0 0 0 0 0 0 1) *
```

Error on the training set of the full tree

```
pred_full <- predict(tree_full, Zoo, type = "class")

accuracy(Zoo |> pull(type), pred_full)
## [1] 1
```

We see that the error is smaller then for the pruned tree. This, however, does not mean that the model is better. It actually is overfitting the training data (it just memorizes it) and has worse generalization performance on new data. This effect is called overfitting the training data and needs to be avoided.

3.5 Model Selection

We often can create many different models for a classification problem. Above, we have created a decision tree using the default settings and also a full tree. The question is: Which one should we use. This problem is called model selection.

In order to select the model we need to split the training data into a **validation set** and the training set that is actually used to train model. The error rate on the validation set can then be used to choose between several models.

Caret has model selection build into the `train()` function. We will select between the default complexity `cp = 0.01` and a full tree `cp = 0` (see `tuneGrid`). `trControl` specified how the validation set is obtained. `LGOCV` picks randomly the proportion `p` of data to train and uses the rest as the validation set. To get a better estimate of the error, this process is repeated `number` times and the errors are averaged.

```
fit <- Zoo |>
  train(type ~ .,
    data = _ ,
    method = "rpart",
    control = rpart.control(minsplit = 2), # we have little data
    tuneGrid = data.frame(cp = c(0.01, 0)),
    trControl = trainControl(method = "LGOCV",
                             p = 0.8,
                             number = 10),
    tuneLength = 5)

fit
## CART
##
## 101 samples
##  16 predictor
##   7 classes: 'mammal', 'bird', 'reptile', 'fish', 'amphibian', 'insect', 'mollusc.et.al'
##
```

```
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (10 reps, 80%)
## Summary of sample sizes: 83, 83, 83, 83, 83, 83, ...
## Resampling results across tuning parameters:
##
##   cp    Accuracy  Kappa
##   0.00  0.9444    0.9232
##   0.01  0.9389    0.9155
##
## Accuracy was used to select the optimal model using
##  the largest value.
## The final value used for the model was cp = 0.
```

We see that in this case, the full tree model performs slightly better. This is a result of the fact that we only have a very small dataset.

3.6 Model Evaluation

Models should be evaluated on a test set that has no overlap with the training set. We typically split the data using random sampling. To get reproducible results, we set random number generator seed.

```
set.seed(2000)
```

3.6.1 Holdout Method

Test data is not used in the model building process and set aside purely for testing the model. Here, we partition data the 80% training and 20% testing.

```
inTrain <- createDataPartition(y = Zoo$type, p = .8)[[1]]
Zoo_train <- Zoo |> slice(inTrain)
Zoo_test <- Zoo |> slice(-inTrain)
```

Now we can train on the test set and get the generalization error on the test set.

3.6.2 Cross-Validation

k-fold cross-validation splits the data randomly into k folds. It then holds one fold back for testing and trains on the other $k-1$ folds. This is done with each fold and the resulting statistic (e.g., accuracy) is averaged. This method uses the data more efficiently then the holdout method.

Cross validation can be directly used in `train()` using `trControl = trainControl(method = "cv", number = 10)`. If no model selection is necessary then this will give the generalization error.

Cross-validation runs are independent and can be done faster in parallel. To enable multi-core support, `caret` uses the package `foreach` and you need to load a `do` backend. For Linux, you can use `doMC` with 4 cores. Windows needs different backend like `doParallel` (see `caret` cheat sheet above).

```
## Linux backend
# library(doMC)
# registerDoMC(cores = 4)
# getDoParWorkers()

## Windows backend
# library(doParallel)
# cl <- makeCluster(4, type="SOCK")
# registerDoParallel(cl)
```

3.7 Hyperparameter Tuning

Hyperparameters are parameters that change how a training algorithm works. An example is the complexity parameter `cp` for rpart decision trees. Tuning the hyperparameter means that we want to perform model selection to pick the best setting.

We typically use the holdout method to create a test set and then use cross validation using the training data for model selection. Let us use 80% for training and hold out 20% for testing.

```
inTrain <- createDataPartition(y = Zoo$type, p = .8)[[1]]
Zoo_train <- Zoo |> slice(inTrain)
Zoo_test <- Zoo |> slice(-inTrain)
```

The package `caret` combines training and validation for hyperparameter tuning into the `train()` function. It internally splits the data into training and validation sets and thus will provide you with error estimates for different hyperparameter settings. `trainControl` is used to choose how testing is performed.

For rpart, train tries to tune the `cp` parameter (tree complexity) using accuracy to chose the best model. I set `minsplit` to 2 since we have not much data. **Note:** Parameters used for tuning (in this case `cp`) need to be set using a data.frame in the argument `tuneGrid`! Setting it in control will be ignored.

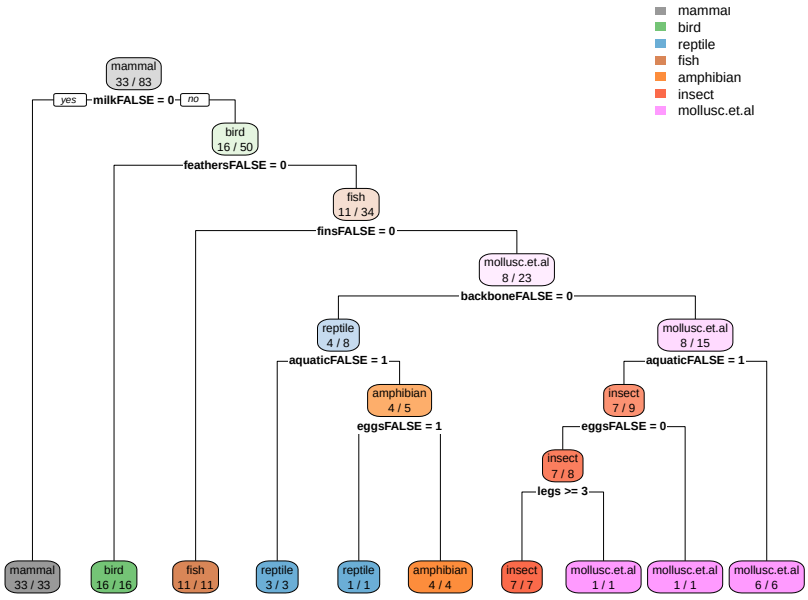
```
fit <- Zoo_train |>
  train(type ~ .,
    data = _ ,
    method = "rpart",
    control = rpart.control(minsplit = 2), # we have little data
    trControl = trainControl(method = "cv", number = 10),
    tuneLength = 5)
```

```
fit
## CART
##
## 83 samples
## 16 predictors
##  7 classes: 'mammal', 'bird', 'reptile', 'fish', 'amphibian', 'insect', 'mollusc.et
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 74, 73, 77, 75, 73, 75, ...
## Resampling results across tuning parameters:
##
##   cp    Accuracy  Kappa
##   0.00  0.9289    0.9058
##   0.08  0.8603    0.8179
##   0.16  0.7296    0.6422
##   0.22  0.6644    0.5448
##   0.32  0.4383    0.1136
##
## Accuracy was used to select the optimal model using
##  the largest value.
## The final value used for the model was cp = 0.
```

Note: Train has built 10 trees using the training folds for each value of `cp` and the reported values for accuracy and Kappa are the averages on the validation folds.

A model using the best tuning parameters and using all the data supplied to `train()` is available as `fit$finalModel`.

```
library(rpart.plot)
rpart.plot(fit$finalModel, extra = 2,
  box.palette = list("Gy", "Gn", "Bu", "Bn", "Or", "Rd", "Pu"))
```

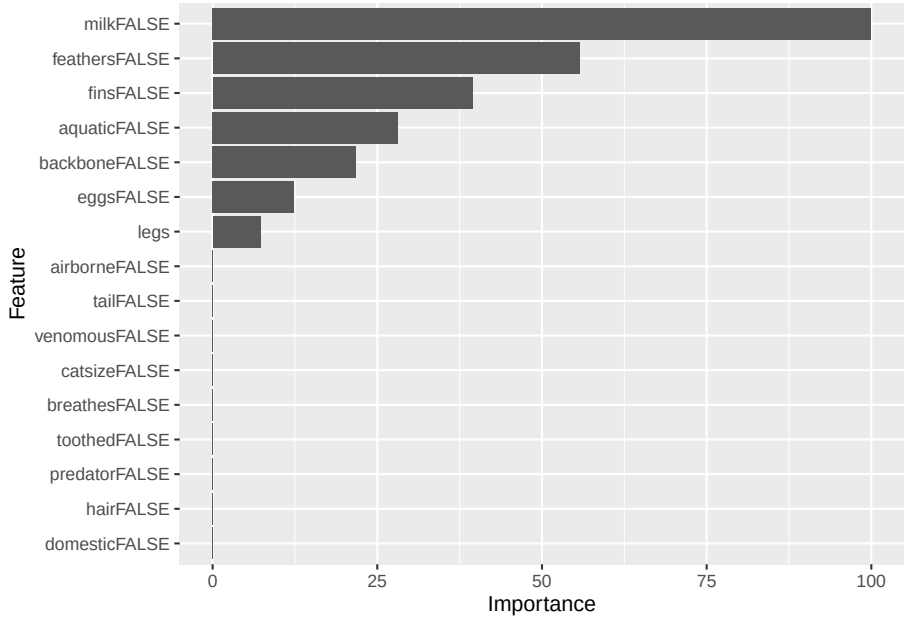


caret also computes variable importance. By default it uses competing splits (splits which would be runners up, but do not get chosen by the tree) for rpart models (see `? varImp`). Toothed is the runner up for many splits, but it never gets chosen!

```
varImp(fit)
## rpart variable importance
##
##                Overall
## toothedFALSE    100.0
## feathersFALSE    79.5
## eggsFALSE        67.7
## milkFALSE        63.3
## backboneFALSE    57.3
## finsFALSE        53.5
## hairFALSE        52.1
## breathesFALSE    48.9
## legs             41.4
## tailFALSE        29.0
## aquaticFALSE     27.5
## airborneFALSE    26.5
## predatorFALSE    10.6
## venomousFALSE     1.8
## catsizeFALSE      0.0
## domesticFALSE     0.0
```

Here is the variable importance without competing splits.

```
imp <- varImp(fit, compete = FALSE)
imp
## rpart variable importance
##
##                Overall
## milkFALSE       100.00
## feathersFALSE    55.69
## finsFALSE        39.45
## aquaticFALSE     28.11
## backboneFALSE    21.76
## eggsFALSE        12.32
## legs              7.28
## tailFALSE         0.00
## domesticFALSE     0.00
## airborneFALSE     0.00
## catsizeFALSE      0.00
## toothedFALSE      0.00
## venomousFALSE     0.00
## hairFALSE         0.00
## breathesFALSE     0.00
## predatorFALSE     0.00
ggplot(imp)
```



Note: Not all models provide a variable importance function. In this case caret

might calculate the variable importance by itself and ignore the model (see `?varImp`)!

Now, we can estimate the generalization error of the best model on the held out test data.

```
pred <- predict(fit, newdata = Zoo_test)
pred
##  [1] mammal        bird          mollusc.et.al bird
##  [5] mammal        mammal        insect        bird
##  [9] mammal        mammal        mammal        mammal
## [13] bird          fish          fish          reptile
## [17] mammal        mollusc.et.al
## 7 Levels: mammal bird reptile fish amphibian ... mollusc.et.al
```

Caret's `confusionMatrix()` function calculates accuracy, confidence intervals, kappa and many more evaluation metrics. You need to use separate test data to create a confusion matrix based on the generalization error.

```
confusionMatrix(data = pred,
                ref = Zoo_test |> pull(type))
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      mammal bird reptile fish amphibian insect
##   mammal             8    0       0    0         0      0
##   bird               0    4       0    0         0      0
##   reptile            0    0       1    0         0      0
##   fish               0    0       0    2         0      0
##   amphibian          0    0       0    0         0      0
##   insect             0    0       0    0         0      1
##   mollusc.et.al      0    0       0    0         0      0
##                Reference
## Prediction      mollusc.et.al
##   mammal                    0
##   bird                      0
##   reptile                   0
##   fish                      0
##   amphibian                 0
##   insect                    0
##   mollusc.et.al             2
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.815, 1)
##     No Information Rate : 0.444
```

```
##     P-Value [Acc > NIR] : 4.58e-07
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: mammal Class: bird
## Sensitivity                  1.000       1.000
## Specificity                  1.000       1.000
## Pos Pred Value               1.000       1.000
## Neg Pred Value               1.000       1.000
## Prevalence                   0.444       0.222
## Detection Rate               0.444       0.222
## Detection Prevalence         0.444       0.222
## Balanced Accuracy            1.000       1.000
##                      Class: reptile Class: fish
## Sensitivity                  1.0000       1.000
## Specificity                  1.0000       1.000
## Pos Pred Value               1.0000       1.000
## Neg Pred Value               1.0000       1.000
## Prevalence                   0.0556       0.111
## Detection Rate               0.0556       0.111
## Detection Prevalence         0.0556       0.111
## Balanced Accuracy            1.0000       1.000
##                      Class: amphibian Class: insect
## Sensitivity                        NA        1.0000
## Specificity                         1        1.0000
## Pos Pred Value                     NA        1.0000
## Neg Pred Value                     NA        1.0000
## Prevalence                          0        0.0556
## Detection Rate                      0        0.0556
## Detection Prevalence                0        0.0556
## Balanced Accuracy                  NA        1.0000
##                      Class: mollusc.et.al
## Sensitivity                         1.000
## Specificity                         1.000
## Pos Pred Value                      1.000
## Neg Pred Value                      1.000
## Prevalence                          0.111
## Detection Rate                      0.111
## Detection Prevalence                0.111
## Balanced Accuracy                   1.000
```

Some notes

- Many classification algorithms and `train` in caret do not deal well with missing values. If your classification model can deal with missing values (e.g., `rpart`) then use `na.action = na.pass` when you call `train` and `predict`. Otherwise, you need to remove observations with missing values with `na.omit` or use imputation to replace the missing values before you train the model. Make sure that you still have enough observations left.
- Make sure that nominal variables (this includes logical variables) are coded as factors.
- The class variable for train in caret cannot have level names that are keywords in R (e.g., `TRUE` and `FALSE`). Rename them to, for example, "yes" and "no."
- Make sure that nominal variables (factors) have examples for all possible values. Some methods might have problems with variable values without examples. You can drop empty levels using `droplevels` or `factor`.
- Sampling in train might create a sample that does not contain examples for all values in a nominal (factor) variable. You will get an error message. This most likely happens for variables which have one very rare value. You may have to remove the variable.

3.8 Pitfalls of Model Selection and Evaluation

- The training data and the test sets cannot overlap or we will not evaluate the generalization performance. The training set can be come contaminated by things like preprocessing the all the data together.
- Do not measure the error on the training set or use the validation error as a generalization error estimate. Always use the generalization error on a test set!

3.9 Model Comparison

We will compare decision trees with a k-nearest neighbors (kNN) classifier. We will create fixed sampling scheme (10-folds) so we compare the different models using exactly the same folds. It is specified as `trControl` during training.

```
train_index <- createFolds(Zoo_train$type,
                           k = 10,
                           returnTrain = TRUE)
```

Build models

```
rpartFit <- Zoo_train |>
  train(type ~ .,
        data = _,
        method = "rpart",
```

```
        tuneLength = 10,
        trControl = trainControl(method = "cv",
                                 index = train_index)
  )
```

Note: for kNN we ask `train` to scale the data using `preProcess = "scale"`. Logicals will be used as 0-1 variables in Euclidean distance calculation.

```
knnFit <- Zoo_train |>
  train(type ~ .,
        data = _,
        method = "knn",
        preProcess = "scale",
          tuneLength = 10,
          trControl = trainControl(method = "cv",
                                   index = train_index)
  )
```

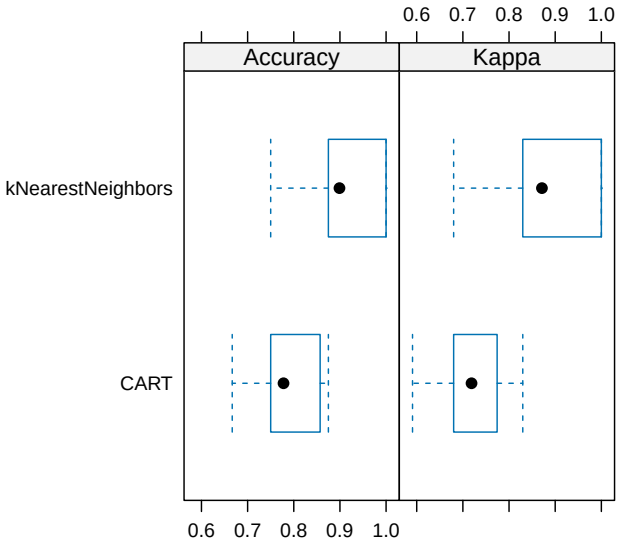
Compare accuracy over all folds.

```
resamps <- resamples(list(
        CART = rpartFit,
        kNearestNeighbors = knnFit
        ))

summary(resamps)
##
## Call:
## summary.resamples(object = resamps)
##
## Models: CART, kNearestNeighbors
## Number of resamples: 10
##
## Accuracy
##                      Min. 1st Qu. Median   Mean 3rd Qu.
## CART               0.6667   0.750 0.7778 0.7890  0.8512
## kNearestNeighbors 0.7500   0.875 0.8990 0.9076  1.0000
##                    Max. NA's
## CART               0.875    0
## kNearestNeighbors 1.000    0
##
## Kappa
##                      Min. 1st Qu. Median   Mean 3rd Qu.
## CART               0.5909  0.6800 0.7188 0.7227  0.7706
## kNearestNeighbors 0.6800  0.8315 0.8713 0.8816  1.0000
##                      Max. NA's
## CART               0.8298    0
```

```
## kNearestNeighbors 1.0000    0
```

`caret` provides some visualizations. For example, a boxplot to compare the accuracy and kappa distribution (over the 10 folds).

```
bwplot(resamps, layout = c(3, 1))
```



We see that kNN is performing consistently better on the folds than CART (except for some outlier folds).

Find out if one models is statistically better than the other (is the difference in accuracy is not zero).

```
difs <- diff(resamps)
difs
##
## Call:
## diff.resamples(x = resamps)
##
## Models: CART, kNearestNeighbors
## Metrics: Accuracy, Kappa
## Number of differences: 1
## p-value adjustment: bonferroni
summary(difs)
##
## Call:
```

```
## summary.diff.resamples(object = difs)
##
## p-value adjustment: bonferroni
## Upper diagonal: estimates of the difference
## Lower diagonal: p-value for H0: difference = 0
##
## Accuracy
##                    CART    kNearestNeighbors
## CART                       -0.119
## kNearestNeighbors 0.00768
##
## Kappa
##                    CART    kNearestNeighbors
## CART                       -0.159
## kNearestNeighbors 0.00538
```

p-values tells you the probability of seeing an even more extreme value (difference between accuracy) given that the null hypothesis (difference = 0) is true. For a better classifier, the p-value should be less than .05 or 0.01. `diff` automatically applies Bonferroni correction for multiple comparisons. In this case, kNN seems better but the classifiers do not perform statistically differently.

3.10 Feature Selection*

Decision trees implicitly select features for splitting, but we can also select features before we apply any learning algorithm. Since different features lead to different models, choosing the best set of features is also a type of model selection.

Many feature selection methods are implemented in the FSelector package.

```
library(FSelector)
```

3.10.1 Univariate Feature Importance Score

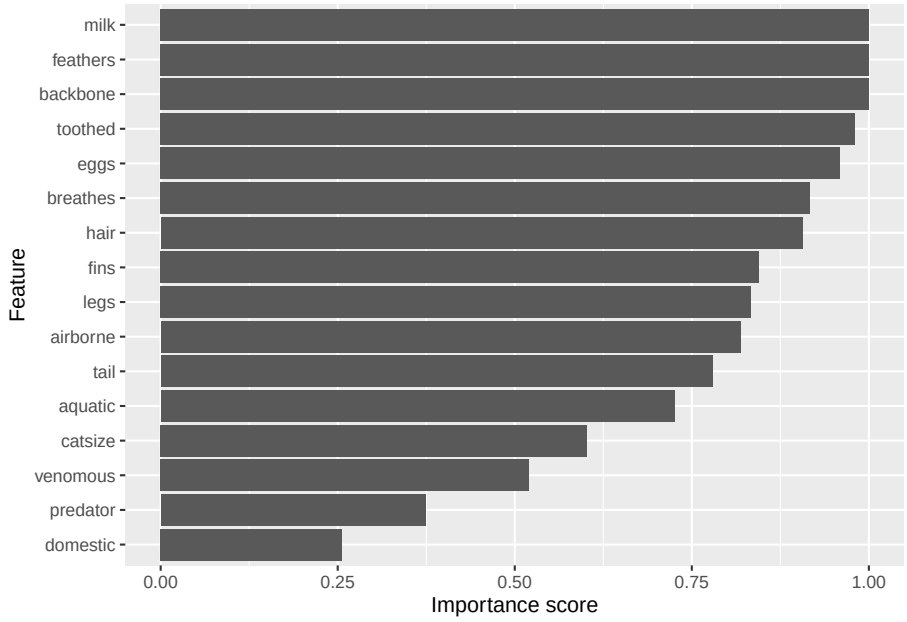
These scores measure how related each feature is to the class variable. For discrete features (as in our case), the chi-square statistic can be used to derive a score.

```
weights <- Zoo_train |>
  chi.squared(type ~ ., data = _) |>
  as_tibble(rownames = "feature") |>
  arrange(desc(attr_importance))

weights
## # A tibble: 16 x 2
```

```
##    feature  attr_importance
##    <chr>              <dbl>
##  1 feathers           1
##  2 milk               1
##  3 backbone           1
##  4 toothed            0.981
##  5 eggs               0.959
##  6 breathes           0.917
##  7 hair               0.906
##  8 fins               0.845
##  9 legs               0.834
## 10 airborne           0.818
## 11 tail               0.779
## 12 aquatic            0.725
## 13 catsize            0.602
## 14 venomous           0.520
## 15 predator           0.374
## 16 domestic           0.256
```

We can plot the importance in descending order (using `reorder` to order factor levels used by `ggplot`).

```
ggplot(weights,
  aes(x = attr_importance,
      y = reorder(feature, attr_importance))) +
  geom_bar(stat = "identity") +
  xlab("Importance score") +
  ylab("Feature")
```

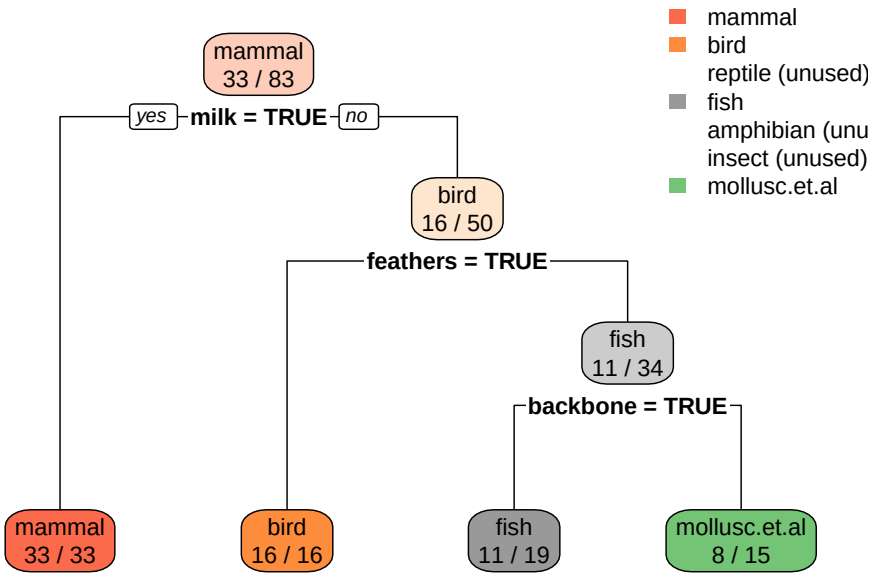


Picking the best features is called the feature ranking approach. Here we pick the 5 highest-ranked features.

```
subset <- cutoff.k(weights |>
                   column_to_rownames("feature"),
                   5)
subset
## [1] "feathers" "milk"     "backbone" "toothed"  "eggs"
```

Use only the selected features to build a model (`Fselector` provides `as.simple.formula`).

```
f <- as.simple.formula(subset, "type")
f
## type ~ feathers + milk + backbone + toothed + eggs
## <environment: 0x55a28baa6c18>
m <- Zoo_train |> rpart(f, data = _)
rpart.plot(m, extra = 2, roundint = FALSE)
```



There are many alternative ways to calculate univariate importance scores (see package FSelector). Some of them (also) work for continuous features. One example is the information gain ratio based on entropy as used in decision tree induction.

```
Zoo_train |>
  gain.ratio(type ~ ., data = _) |>
  as_tibble(rownames = "feature") |>
  arrange(desc(attr_importance))
## # A tibble: 16 x 2
##    feature  attr_importance
##    <chr>              <dbl>
##  1 milk               1
##  2 backbone           1
##  3 feathers           1
##  4 toothed            0.959
##  5 eggs               0.907
##  6 breathes           0.845
##  7 hair               0.781
##  8 fins               0.689
##  9 legs               0.689
## 10 airborne           0.633
## 11 tail               0.573
## 12 aquatic            0.474
## 13 venomous           0.429
```

```
## 14 catsize            0.310
## 15 domestic           0.115
## 16 predator           0.110
```

3.10.2 Feature Subset Selection

Often, features are related and calculating importance for each feature independently is not optimal. We can use greedy search heuristics. For example `cfs` uses correlation/entropy with best first search.

```
Zoo_train |>
  cfs(type ~ ., data = _)
##  [1] "hair"     "feathers" "eggs"     "milk"     "toothed"
##  [6] "backbone" "breathes" "fins"     "legs"     "tail"
```

Black-box feature selection uses an evaluator function (the black box) to calculate a score to be maximized. This is typically the best method, since it can use the actual model for selection. First, we define an evaluation function that builds a model given a subset of features and calculates a quality score. We use here the average for 5 bootstrap samples (`method = "cv"` can also be used instead), no tuning (to be faster), and the average accuracy as the score.

```
evaluator <- function(subset) {
  model <- Zoo_train |>
    train(as.simple.formula(subset, "type"),
          data = _,
          method = "rpart",
          trControl = trainControl(method = "boot", number = 5),
          tuneLength = 0)

  results <- model$resample$Accuracy

  cat("Trying features:", paste(subset, collapse = " + "), "\n")

  m <- mean(results)
  cat("Accuracy:", round(m, 2), "\n\n")
  m
}
```

Start with all features (but not the class variable `type`)

```
features <- Zoo_train |>
  colnames() |>
  setdiff("type")
```

There are several (greedy) search strategies available. These run for a while so they commented out below. Remove the comment for one at a time to try these

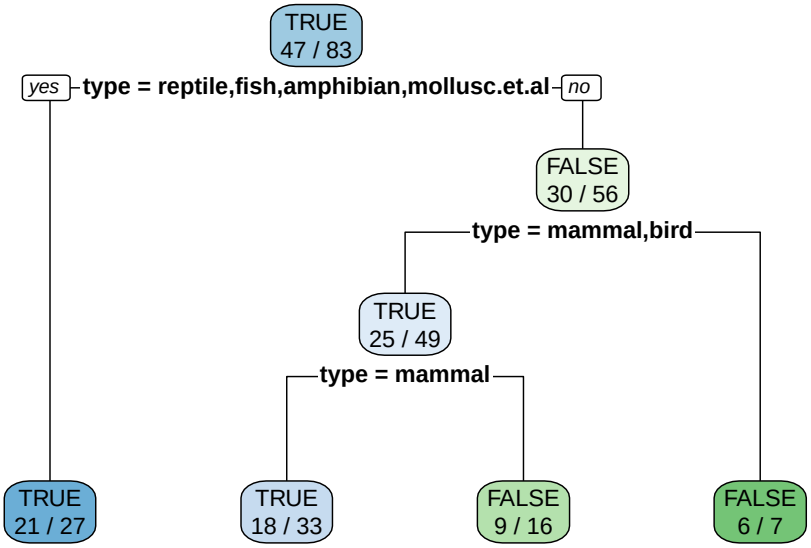
types of feature selection.

```
#subset <- backward.search(features, evaluator)
#subset <- forward.search(features, evaluator)
#subset <- best.first.search(features, evaluator)
#subset <- hill.climbing.search(features, evaluator)
#subset
```

3.10.3 Using Dummy Variables for Factors

Nominal features (factors) are often encoded as a series of 0-1 dummy variables. For example, let us try to predict if an animal is a predator given the type. First we use the original encoding of type as a factor with several values.

```
tree_predator <- Zoo_train |>
  rpart(predator ~ type, data = _)
rpart.plot(tree_predator, extra = 2, roundint = FALSE)
```

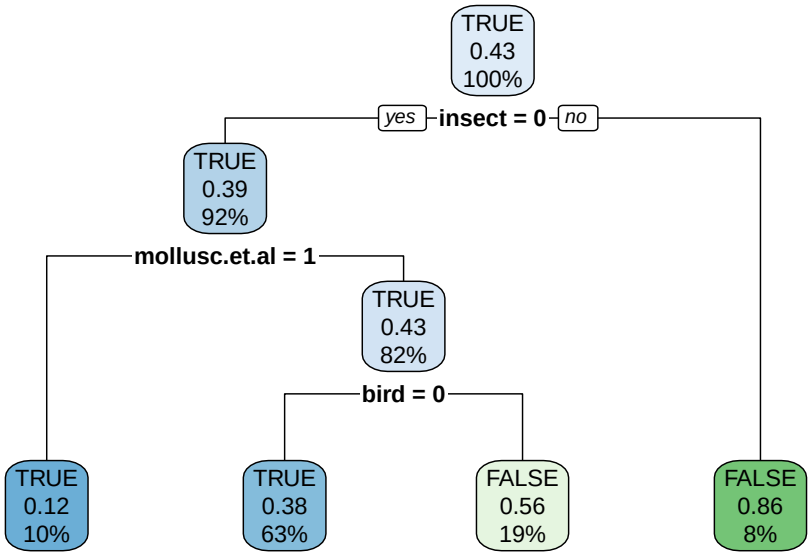


Note: Some splits use multiple values. Building the tree will become extremely slow if a factor has many levels (different values) since the tree has to check all possible splits into two subsets. This situation should be avoided.

Convert type into a set of 0-1 dummy variables using `class2ind`. See also `?dummyVars` in package `caret`.

```
Zoo_train_dummy <- as_tibble(class2ind(Zoo_train$type)) |>
  mutate(across(everything(), as.factor)) |>
```

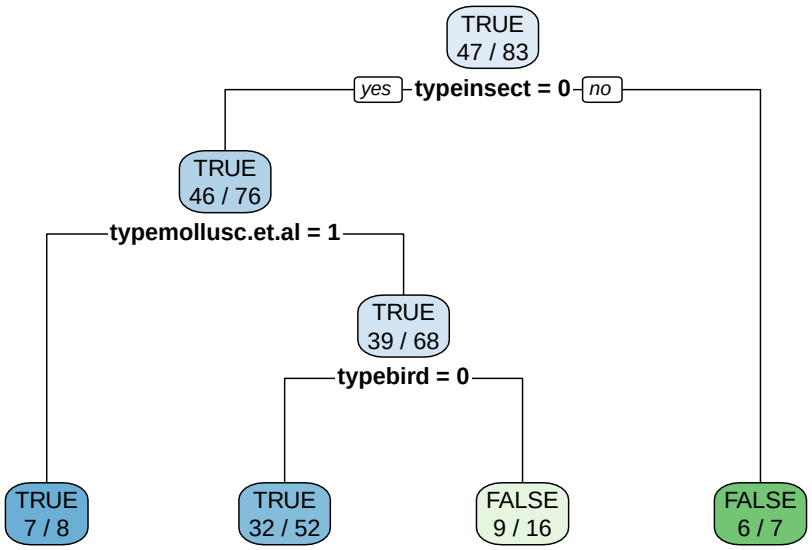
```
  add_column(predator = Zoo_train$predator)
Zoo_train_dummy
## # A tibble: 83 x 8
##    mammal bird  reptile fish  amphibian insect mollusc.et.al
##    <fct>  <fct> <fct>   <fct> <fct>     <fct>  <fct>
##  1 1      0     0       0     0         0      0
##  2 0      0     0       1     0         0      0
##  3 1      0     0       0     0         0      0
##  4 1      0     0       0     0         0      0
##  5 1      0     0       0     0         0      0
##  6 1      0     0       0     0         0      0
##  7 0      0     0       1     0         0      0
##  8 0      0     0       1     0         0      0
##  9 1      0     0       0     0         0      0
## 10 1      0     0       0     0         0      0
## # i 73 more rows
## # i 1 more variable: predator <fct>
tree_predator <- Zoo_train_dummy |>
  rpart(predator ~ .,
        data = _,
        control = rpart.control(minsplit = 2, cp = 0.01))
rpart.plot(tree_predator, roundint = FALSE)
```



Using `caret` on the original factor encoding automatically translates factors (here type) into 0-1 dummy variables (e.g., `typeinsect = 0`). The reason is

that some models cannot directly use factors and `caret` tries to consistently work with all of them.

```
fit <- Zoo_train |>
  train(predator ~ type,
        data = _,
        method = "rpart",
        control = rpart.control(minsplit = 2),
        tuneGrid = data.frame(cp = 0.01))
fit
## CART
##
## 83 samples
##  1 predictor
##  2 classes: 'TRUE', 'FALSE'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 83, 83, 83, 83, 83, 83, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.5635    0.1166
##
## Tuning parameter 'cp' was held constant at a value of 0.01
rpart.plot(fit$finalModel, extra = 2)
```



Note: To use a fixed value for the tuning parameter `cp`, we have to create a tuning grid that only contains that value.

3.11 Exercises*

We will use again the Palmer penguin data for the exercises.

```
library(palmerpenguins)
head(penguins)
## # A tibble: 6 x 8
##   species island    bill_length_mm bill_depth_mm
##   <chr>   <chr>              <dbl>         <dbl>
## 1 Adelie  Torgersen           39.1          18.7
## 2 Adelie  Torgersen           39.5          17.4
## 3 Adelie  Torgersen           40.3          18
## 4 Adelie  Torgersen           NA            NA
## 5 Adelie  Torgersen           36.7          19.3
## 6 Adelie  Torgersen           39.3          20.6
## # i 4 more variables: flipper_length_mm <dbl>,
## #   body_mass_g <dbl>, sex <chr>, year <dbl>
```

Create a R markdown file with the code and discussion for the following below. Remember, the complete approach is described in section Hyperparameter Tuning.

1. Split the data into a training and test set.

2. Create an rpart decision tree to predict the species. You will have to deal with missing values.
3. Experiment with setting `minsplit` for rpart and make sure `tuneLength` is at least 5. Discuss the model selection process (hyperparameter tuning) and what final model was chosen.
4. Visualize the tree and discuss what the splits mean.
5. Calculate the variable importance from the fitted model. What variables are the most important? What variables do not matter?
6. Use the test set to evaluate the generalization error and accuracy.

Chapter 4

Classification: Alternative Techniques

This chapter introduces different types of classifiers. It also discusses the important problem of class imbalance in data and options to deal with it. In addition, this chapter compares visually the decision boundaries used by different algorithms. This will provide a better understanding of the model bias that different algorithms have.

Packages Used in this Chapter

```
pkgs <- c("basemodels", "C50", "caret", "e1071", "keras3", "klaR",
          "lattice", "MASS", "mlbench", "nnet", "palmerpenguins",
          "randomForest", "rpart", "RWeka", "scales", "tidyverse",
          "xgboost")

pkgs_install <- pkgs[!(pkgs %in% installed.packages()[,"Package"])]
if(length(pkgs_install)) install.packages(pkgs_install)
```

The packages used for this chapter are:

- *basemodels* (Y.-J. Chen et al. 2023)
- *C50* (Kuhn and Quinlan 2023)
- *caret* (Kuhn 2023)
- *e1071* (Meyer et al. 2024)
- *keras3* (Kalinowski, Allaire, and Chollet 2024)
- *klaR* (Roever et al. 2023)
- *lattice* (Sarkar 2023)
- *MASS* (Ripley and Venables 2024)
- *mlbench* (Leisch and Dimitriadou 2024)

- *nnet* (Ripley 2023)
- *palmerpenguins* (Horst, Hill, and Gorman 2022)
- *randomForest* (Breiman et al. 2024)
- *rpart* (Therneau and Atkinson 2023)
- *RWeka* (Hornik 2023)
- *scales* (Wickham, Pedersen, and Seidel 2023)
- *tidyverse* (Wickham 2023c)
- *xgboost* (T. Chen et al. 2024)

4.1 Types of Classifiers

Many different classification algorithms[1] have been proposed in the literature. In this chapter, we will apply some of the more popular methods.

4.1.1 Set up the Training and Test Data

We will use the Zoo dataset which is included in the R package `mlbench` (you may have to install it). The Zoo dataset containing 17 (mostly logical) variables on different 101 animals as a data frame with 17 columns (hair, feathers, eggs, milk, airborne, aquatic, predator, toothed, backbone, breathes, venomous, fins, legs, tail, domestic, catsize, type). We convert the data frame into a tidyverse tibble (optional).

```
library(tidyverse)
data(Zoo, package = "mlbench")

Zoo <- as_tibble(Zoo)
Zoo
## # A tibble: 101 x 17
##    hair  feathers eggs  milk  airborne aquatic predator
##    <lgl> <lgl>    <lgl> <lgl> <lgl>    <lgl>   <lgl>
##  1 TRUE  FALSE    FALSE TRUE  FALSE    FALSE   TRUE
##  2 TRUE  FALSE    FALSE TRUE  FALSE    FALSE   FALSE
##  3 FALSE FALSE    TRUE  FALSE FALSE    TRUE    TRUE
##  4 TRUE  FALSE    FALSE TRUE  FALSE    FALSE   TRUE
##  5 TRUE  FALSE    FALSE TRUE  FALSE    FALSE   TRUE
##  6 TRUE  FALSE    FALSE TRUE  FALSE    FALSE   FALSE
##  7 TRUE  FALSE    FALSE TRUE  FALSE    FALSE   FALSE
##  8 FALSE FALSE    TRUE  FALSE FALSE    TRUE    FALSE
##  9 FALSE FALSE    TRUE  FALSE FALSE    TRUE    TRUE
## 10 TRUE  FALSE    FALSE TRUE  FALSE    FALSE   FALSE
## # i 91 more rows
## # i 10 more variables: toothed <lgl>, backbone <lgl>,
## #   breathes <lgl>, venomous <lgl>, fins <lgl>, legs <int>,
```

[1] https://en.wikipedia.org/wiki/Supervised_learning

```
## #   tail <lgl>, domestic <lgl>, catsize <lgl>, type <fct>
```

We will use the package **caret**[2] to make preparing training sets and building classification (and regression) models easier. A great cheat sheet can be found here[3].

```
library(caret)
```

Multi-core support can be used for cross-validation. **Note:** It is commented out here because it does not work with rJava used by the RWeka-based classifiers below.

```
##library(doMC, quietly = TRUE)
##registerDoMC(cores = 4)
##getDoParWorkers()
```

Test data is not used in the model building process and needs to be set aside purely for testing the model after it is completely built. Here I use 80% for training.

```
inTrain <- createDataPartition(y = Zoo$type, p = .8)[[1]]
Zoo_train <- Zoo |> slice(inTrain)
Zoo_test <- Zoo |> slice(-inTrain)
```

`train()` is aware of the tunable hyperparameters of each method and automatically performs model selection using a validation set. We will use for all models the same validation sets by creating a fixed sampling scheme (10-folds). This will help when we can compare the fitted models later.

```
train_index <- createFolds(Zoo_train$type,
                           k = 10,
                           returnTrain = TRUE)
```

The fixed folds are used in `train()` with the argument `trControl = trainControl(method = "cv", index = train_index))`. If you don't need fixed folds, then remove `index = train_index` in the code below.

For help with building models in caret see: `? train`

Note: Be careful if you have many `NA` values in your data. `train()` and cross-validation many fail in some cases. If that is the case then you can remove features (columns) which have many `NA`s, omit `NA`s using `na.omit()` or use imputation to replace them with reasonable values (e.g., by the feature mean or via kNN). Highly imbalanced datasets are also problematic since there is a chance that a fold does not contain examples of each class leading to a hard to understand error message.

[2] https://topepo.github.io/caret/
[3] https://ugoproto.github.io/ugo_r_doc/pdf/caret.pdf

4.2 Rule-based classifier: PART

```
rulesFit <- Zoo_train |> train(type ~ .,
  method = "PART",
  data = _,
  tuneLength = 5,
  trControl = trainControl(method = "cv",
                           index = train_index))
rulesFit
## Rule-Based Classifier
##
## 83 samples
## 16 predictors
##  7 classes: 'mammal', 'bird', 'reptile', 'fish', 'amphibian', 'insect', 'mollusc.et
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 74, 75, 75, 75, 75, 74, ...
## Resampling results across tuning parameters:
##
##   threshold  pruned  Accuracy  Kappa
##   0.0100     yes     0.9567    0.9439
##   0.0100     no      0.9467    0.9309
##   0.1325     yes     0.9567    0.9439
##   0.1325     no      0.9467    0.9309
##   0.2550     yes     0.9567    0.9439
##   0.2550     no      0.9467    0.9309
##   0.3775     yes     0.9567    0.9439
##   0.3775     no      0.9467    0.9309
##   0.5000     yes     0.9567    0.9439
##   0.5000     no      0.9467    0.9309
##
## Accuracy was used to select the optimal model using
##  the largest value.
## The final values used for the model were threshold =
##  0.5 and pruned = yes.
```

The model selection results are shown in the table. This is the selected model:

```
rulesFit$finalModel
## PART decision list
## ------------------
##
## feathersTRUE <= 0 AND
## milkTRUE > 0: mammal (33.0)
##
```

```
## feathersTRUE > 0: bird (16.0)
##
## backboneTRUE <= 0 AND
## airborneTRUE <= 0: mollusc.et.al (9.0/1.0)
##
## airborneTRUE <= 0 AND
## finsTRUE > 0: fish (11.0)
##
## airborneTRUE > 0: insect (6.0)
##
## aquaticTRUE > 0: amphibian (5.0/1.0)
##
## : reptile (3.0)
##
## Number of Rules  :   7
```

PART returns a decision list, i.e., an ordered rule set. For example, the first rule shows that an animal with no feathers but milk is a mammal. I ordered rule sets, the decision of the first matching rule is used.

4.3 Nearest Neighbor Classifier

K-Nearest neighbor classifiers classify a new data point by looking at the majority class labels of its k nearest neighbors in the training data set. The used kNN implementation uses Euclidean distance to determine what data points are near by, so data needs be standardized (scaled) first. Here legs are measured between 0 and 6 while all other variables are between 0 and 1. Scaling to z-scores can be directly performed as preprocessing in `train` using the parameter `preProcess = "scale"`.

The k value is typically choose as an odd number so we get a clear majority.

```
knnFit <- Zoo_train |> train(type ~ .,
  method = "knn",
  data = _,
  preProcess = "scale",
  tuneGrid = data.frame(k = c(1, 3, 5, 7, 9)),
    trControl = trainControl(method = "cv",
                             index = train_index))
knnFit
## k-Nearest Neighbors
##
## 83 samples
## 16 predictors
##  7 classes: 'mammal', 'bird', 'reptile', 'fish', 'amphibian', 'insect', 'mollusc.et.al'
##
```

```
## Pre-processing: scaled (16)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 74, 75, 75, 75, 75, 74, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy  Kappa
##   1  0.9342    0.9158
##   3  0.9331    0.9145
##   5  0.9231    0.9007
##   7  0.8981    0.8708
##   9  0.8994    0.8676
##
## Accuracy was used to select the optimal model using
##  the largest value.
## The final value used for the model was k = 1.
```

```
knnFit$finalModel
## 1-nearest neighbor model
## Training set outcome distribution:
##
##        mammal          bird       reptile          fish
##            33            16             4            11
##     amphibian        insect mollusc.et.al
##             4             7             8
```

kNN classifiers are lazy models meaning that instead of learning, they just keep the complete dataset. This is why the final model just gives us a summary statistic for the class labels in the training data.

4.4 Naive Bayes Classifier

Caret's train formula interface translates logicals and factors into dummy variables which the classifier interprets as numbers so it would used a Gaussian naive Bayes[4] estimation. To avoid this, I directly specify `x` and `y`.

```
NBFit <- train(x = as.data.frame(Zoo_train[, -ncol(Zoo_train)]),
               y = pull(Zoo_train, "type"),
               method = "nb",
               tuneGrid = data.frame(fL = c(.2, .5, 1, 5),
                                     usekernel = TRUE, adjust = 1),
               trControl = trainControl(method = "cv",
                                        index = train_index))
NBFit
## Naive Bayes
```

[4] https://en.wikipedia.org/wiki/Naive_Bayes_classifier#Gaussian_naive_Bayes

```
##
## 83 samples
## 16 predictors
##  7 classes: 'mammal', 'bird', 'reptile', 'fish', 'amphibian', 'insect', 'mollusc.et.al'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 74, 75, 75, 75, 75, 74, ...
## Resampling results across tuning parameters:
##
##   fL   Accuracy  Kappa
##   0.2  0.9453    0.9294
##   0.5  0.9453    0.9294
##   1.0  0.9342    0.9157
##   5.0  0.8981    0.8697
##
## Tuning parameter 'usekernel' was held constant at a
##  value of TRUE
## Tuning parameter 'adjust' was held
##  constant at a value of 1
## Accuracy was used to select the optimal model using
##  the largest value.
## The final values used for the model were fL =
##  0.2, usekernel = TRUE and adjust = 1.
```

The final model contains the prior probabilities for each class.

```
NBFit$finalModel$apriori
## grouping
##        mammal          bird       reptile          fish
##       0.39759       0.19277       0.04819       0.13253
##     amphibian        insect mollusc.et.al
##       0.04819       0.08434       0.09639
```

And the conditional probabilities as a table for each feature. For brevity, we only show the tables for the first three features. For example, the condition probability $P(\text{hair} = \text{TRUE}|\text{class} = \text{mammal})$ is 0.9641.

```
NBFit$finalModel$tables[1:3]
## $hair
##                var
## grouping          FALSE    TRUE
##   mammal        0.03593 0.96407
##   bird          0.98780 0.01220
##   reptile       0.95455 0.04545
##   fish          0.98246 0.01754
##   amphibian     0.95455 0.04545
```

```
##   insect        0.43243 0.56757
##   mollusc.et.al 0.97619 0.02381
##
## $feathers
##                var
## grouping          FALSE     TRUE
##   mammal        0.994012 0.005988
##   bird          0.012195 0.987805
##   reptile       0.954545 0.045455
##   fish          0.982456 0.017544
##   amphibian     0.954545 0.045455
##   insect        0.972973 0.027027
##   mollusc.et.al 0.976190 0.023810
##
## $eggs
##                var
## grouping          FALSE    TRUE
##   mammal        0.96407 0.03593
##   bird          0.01220 0.98780
##   reptile       0.27273 0.72727
##   fish          0.01754 0.98246
##   amphibian     0.04545 0.95455
##   insect        0.02703 0.97297
##   mollusc.et.al 0.14286 0.85714
```

4.5 Bayesian Network

Bayesian networks are not covered here. R has very good support for modeling with Bayesian Networks. An example is the package bnlearn[5].

4.6 Logistic regression

Logistic regression is a very powerful classification method and should always be tried as one of the first models. A detailed discussion with more code is available in Section Logistic Regression*.

Regular logistic regression predicts only one outcome coded as a binary variable. Since we have data with several classes, we use multinomial logistic regression[6], also called a log-linear model which is an extension of logistic regresses for multi-class problems. Caret can use method `"multinom"` which uses an implementation in package `nnet` for penalized multinomial regression.

[5] https://www.bnlearn.com/

[6] https://en.wikipedia.org/wiki/Multinomial_logistic_regression

```
logRegFit <- Zoo_train |> train(type ~ .,
  method = "multinom",
  data = _,
  trace = FALSE, # suppress some output
    trControl = trainControl(method = "cv",
                             index = train_index))
logRegFit
## Penalized Multinomial Regression
##
## 83 samples
## 16 predictors
##  7 classes: 'mammal', 'bird', 'reptile', 'fish', 'amphibian', 'insect', 'mollusc.et.al'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 74, 75, 75, 75, 75, 74, ...
## Resampling results across tuning parameters:
##
##   decay  Accuracy  Kappa
##   0e+00  0.9064    0.8768
##   1e-04  0.8842    0.8492
##   1e-01  0.9231    0.9006
##
## Accuracy was used to select the optimal model using
##  the largest value.
## The final value used for the model was decay = 0.1.
```

```
logRegFit$finalModel
## Call:
## nnet::multinom(formula = .outcome ~ ., data = dat, decay = param$decay,
##     trace = FALSE)
##
## Coefficients:
##               (Intercept) hairTRUE featherTRUE eggsTRUE
## bird              -0.33330  -1.0472       2.9696   0.8278
## reptile            0.01303  -2.0808      -1.0891   0.6731
## fish              -0.17462  -0.2762      -0.1135   1.8817
## amphibian         -1.28295  -1.5165      -0.2698   0.6801
## insect            -0.75300  -0.3903      -0.1445   0.8980
## mollusc.et.al      1.52104  -1.2287      -0.2492   0.9320
##               milkTRUE airborneTRUE aquaticTRUE
## bird           -1.2523      1.17310     -0.1594
## reptile        -2.1800     -0.51796     -1.0890
## fish           -1.3571     -0.09009      0.5093
## amphibian      -1.6014     -0.36649      1.6271
```

```
## insect          -1.0130      1.37404     -1.0752
## mollusc.et.al   -0.9035     -1.17882      0.7160
##                predatorTRUE toothedTRUE backboneTRUE
## bird                0.22312     -1.7846       0.4736
## reptile             0.04172     -0.2003       0.8968
## fish               -0.33094      0.4118       0.2768
## amphibian          -0.13993      0.7399       0.2557
## insect             -1.11743     -1.1852      -1.5725
## mollusc.et.al       0.83070     -1.7390      -2.6045
##                breathesTRUE venomousTRUE  finsTRUE     legs
## bird                 0.1337      -0.3278 -0.545979 -0.59910
## reptile             -0.5039       1.2776 -1.192197 -0.24200
## fish                -1.9709      -0.4204  1.472416 -1.15775
## amphibian            0.4594       0.1611 -0.628746  0.09302
## insect               0.1341      -0.2567 -0.002527  0.59118
## mollusc.et.al       -0.6287       0.8411 -0.206104  0.12091
##                tailTRUE domesticTRUE catsizeTRUE
## bird             0.5947      0.14176     -0.1182
## reptile          1.1863     -0.40893     -0.4305
## fish             0.3226      0.08636     -0.3132
## amphibian       -1.3529     -0.40545     -1.3581
## insect          -1.6908     -0.24924     -1.0416
## mollusc.et.al   -0.7353     -0.22601     -0.7079
##
## Residual Deviance: 35.46
## AIC: 239.5
```

The coefficients are log-odds ratios. A negative log-odds ratio means that the odds go down with an increase in the value of the predictor. A predictor with a positive log-odds ratio increases the odds. For example, in the model above, `hair=TRUE` has a negative coefficient for bird but `feathers=TRUE` has a large positive coefficient.

4.7 Artificial Neural Network (ANN)

Standard networks have an input layer, an output layer and in between a single hidden layer.

```
nnetFit <- Zoo_train |> train(type ~ .,
  method = "nnet",
  data = _,
    tuneLength = 5,
    trControl = trainControl(method = "cv",
                             index = train_index),
  trace = FALSE # no progress output
```

```
  )
nnetFit
## Neural Network
##
## 83 samples
## 16 predictors
##  7 classes: 'mammal', 'bird', 'reptile', 'fish', 'amphibian', 'insect', 'mollusc.et.al'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 74, 75, 75, 75, 75, 74, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy  Kappa
##   1     0e+00  0.7475    0.6538
##   1     1e-04  0.7125    0.5839
##   1     1e-03  0.8758    0.8362
##   1     1e-02  0.8036    0.7462
##   1     1e-01  0.7339    0.6439
##   3     0e+00  0.8483    0.7997
##   3     1e-04  0.8383    0.7876
##   3     1e-03  0.9106    0.8856
##   3     1e-02  0.9231    0.8982
##   3     1e-01  0.8758    0.8402
##   5     0e+00  0.8969    0.8636
##   5     1e-04  0.9108    0.8850
##   5     1e-03  0.9331    0.9142
##   5     1e-02  0.9456    0.9295
##   5     1e-01  0.9106    0.8857
##   7     0e+00  0.9108    0.8847
##   7     1e-04  0.8953    0.8641
##   7     1e-03  0.9119    0.8875
##   7     1e-02  0.9331    0.9142
##   7     1e-01  0.9331    0.9141
##   9     0e+00  0.9175    0.8910
##   9     1e-04  0.9053    0.8729
##   9     1e-03  0.9219    0.9006
##   9     1e-02  0.9231    0.9011
##   9     1e-01  0.9231    0.9011
##
## Accuracy was used to select the optimal model using
##  the largest value.
## The final values used for the model were size = 5 and
##  decay = 0.01.
```

The input layer has a size of 16, one for each input feature and the output layer has a size of 7 representing the 7 classes. Model selection chose a network architecture with a hidden layer with 5 units resulting in 127 learned weights. Since the model is considered a black-box model only the network architecture and the used variables are shown as a summary.

```
nnetFit$finalModel
## a 16-5-7 network with 127 weights
## inputs: hairTRUE feathersTRUE eggsTRUE milkTRUE airborneTRUE aquaticTRUE predatorTR
## output(s): .outcome
## options were - softmax modelling  decay=0.01
```

For deep Learning, R offers packages for using tensorflow[7] and Keras[8].

4.8 Support Vector Machines

```
svmFit <- Zoo_train |> train(type ~.,
  method = "svmLinear",
  data = _,
    tuneLength = 5,
    trControl = trainControl(method = "cv",
                             index = train_index))
svmFit
## Support Vector Machines with Linear Kernel
##
## 83 samples
## 16 predictors
##  7 classes: 'mammal', 'bird', 'reptile', 'fish', 'amphibian', 'insect', 'mollusc.et
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 74, 75, 75, 75, 75, 74, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.9356    0.9173
##
## Tuning parameter 'C' was held constant at a value of 1
```

We use a linear support vector machine. Support vector machines can use kernels to create non-linear decision boundaries. `method` above can be changed to `"svmPoly"` or `"svmRadial"` to use kernels. The choice of kernel is typically make by experimentation.

[7] https://tensorflow.rstudio.com/
[8] https://keras3.posit.co/

The support vectors determining the decision boundary are stored in the model.

```
svmFit$finalModel
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 42
##
## Objective Function Value : -0.1432 -0.22 -0.1501 -0.1756 -0.0943 -0.1047 -0.2804 -0.0808 -0.15
## Training error : 0
```

4.9 Ensemble Methods

Many ensemble methods are available in R. We only cover here code for two popular methods.

4.9.1 Random Forest

```
randomForestFit <- Zoo_train |> train(type ~ .,
  method = "rf",
  data = _,
    tuneLength = 5,
    trControl = trainControl(method = "cv",
                             index = train_index))
randomForestFit
## Random Forest
##
## 83 samples
## 16 predictors
##  7 classes: 'mammal', 'bird', 'reptile', 'fish', 'amphibian', 'insect', 'mollusc.et.al'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 74, 75, 75, 75, 75, 74, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa
##    2    0.9331    0.9136
##    5    0.9356    0.9168
##    9    0.9467    0.9313
##   12    0.9467    0.9316
```

```
##   16    0.9467    0.9311
##
## Accuracy was used to select the optimal model using
##  the largest value.
## The final value used for the model was mtry = 9.
```

The default number of trees is 500 and `mtry` determines the number of variables randomly sampled as candidates at each split. This number is a tradeoff where a larger number allows each tree to pick better splits, but a smaller number increases the independence between trees.

```
randomForestFit$finalModel
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 9
##
##         OOB estimate of  error rate: 7.23%
## Confusion matrix:
##               mammal bird reptile fish amphibian insect
## mammal            33    0       0    0         0      0
## bird               0   16       0    0         0      0
## reptile            0    0       2    1         1      0
## fish               0    0       0   11         0      0
## amphibian          0    0       1    0         3      0
## insect             0    0       0    0         0      6
## mollusc.et.al      0    0       1    0         0      1
##               mollusc.et.al class.error
## mammal                    0      0.0000
## bird                      0      0.0000
## reptile                   0      0.5000
## fish                      0      0.0000
## amphibian                 0      0.2500
## insect                    1      0.1429
## mollusc.et.al             6      0.2500
```

The model is a set of 500 trees and the prediction is made by applying all trees and then using the majority vote.

Since random forests use bagging (bootstrap sampling to train trees), the remaining data can be used like a test set. The resulting error is called out-of-bag (OOB) error and gives an estimate for the generalization error. The model above also shows the confusion matrix based on the OOB error.

4.9.2 Gradient Boosted Decision Trees (xgboost)

The idea of gradient boosting is to learn a base model and then to learn successive models to predict and correct the error of all previous models. Typically, tree models are used.

```
xgboostFit <- Zoo_train |> train(type ~ .,
  method = "xgbTree",
  data = _,
  tuneLength = 5,
  trControl = trainControl(method = "cv",
                           index = train_index),
  tuneGrid = expand.grid(
    nrounds = 20,
    max_depth = 3,
    colsample_bytree = .6,
    eta = 0.1,
    gamma=0,
    min_child_weight = 1,
    subsample = .5
  ))
xgboostFit
## eXtreme Gradient Boosting
##
## 83 samples
## 16 predictors
##  7 classes: 'mammal', 'bird', 'reptile', 'fish', 'amphibian', 'insect', 'mollusc.et.al'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 74, 75, 75, 75, 75, 74, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.9442    0.9267
##
## Tuning parameter 'nrounds' was held constant at a value
##  held constant at a value of 1
## Tuning parameter
##  'subsample' was held constant at a value of 0.5
```

The final model is a complicated set of trees, so only some summary information is shown.

```
xgboostFit$finalModel
## ##### xgb.Booster
## raw: 111.6 Kb
```

```
## call:
##   xgboost::xgb.train(params = list(eta = param$eta, max_depth = param$max_depth,
##     gamma = param$gamma, colsample_bytree = param$colsample_bytree,
##     min_child_weight = param$min_child_weight, subsample = param$subsample),
##     data = x, nrounds = param$nrounds, num_class = length(lev),
##     objective = "multi:softprob")
## params (as set within xgb.train):
##   eta = "0.1", max_depth = "3", gamma = "0", colsample_bytree = "0.6", min_child_we
## xgb.attributes:
##   niter
## callbacks:
##   cb.print.evaluation(period = print_every_n)
## # of features: 16
## niter: 20
## nfeatures : 16
## xNames : hairTRUE feathersTRUE eggsTRUE milkTRUE airborneTRUE aquaticTRUE predatorT
## problemType : Classification
## tuneValue :
##    nrounds max_depth eta gamma colsample_bytree
## 1      20         3 0.1     0              0.6
##   min_child_weight subsample
## 1                1       0.5
## obsLevels : mammal bird reptile fish amphibian insect mollusc.et.al
## param :
##  list()
```

4.10 Class Imbalance

Classifiers have a hard time to learn from data where we have much more observations for one class (called the majority class). This is called the class imbalance problem.

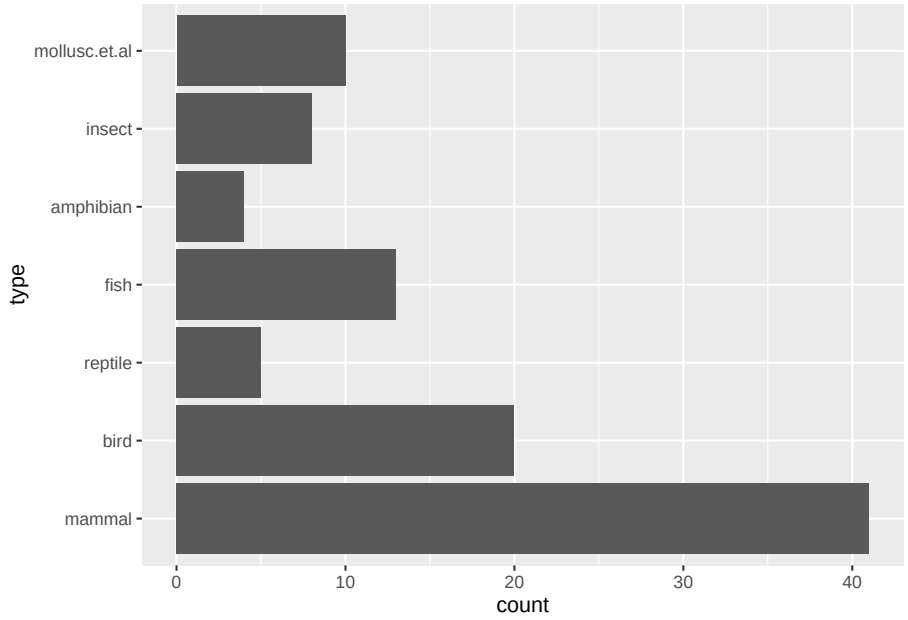
Here is a very good article about the problem and solutions.[9]

```
library(rpart)
library(rpart.plot)
data(Zoo, package = "mlbench")
```

Class distribution

```
ggplot(Zoo, aes(y = type)) + geom_bar()
```

[9] http://www.kdnuggets.com/2016/08/learning-from-imbalanced-classes.html



To create an imbalanced problem, we want to decide if an animal is an reptile. First, we change the class variable to make it into a binary reptile/no reptile classification problem. **Note:** We use here the training data for testing. You should use a separate testing data set!

```
Zoo_reptile <- Zoo |>
  mutate(type = factor(Zoo$type == "reptile",
                       levels = c(FALSE, TRUE),
                       labels = c("nonreptile", "reptile")))
```

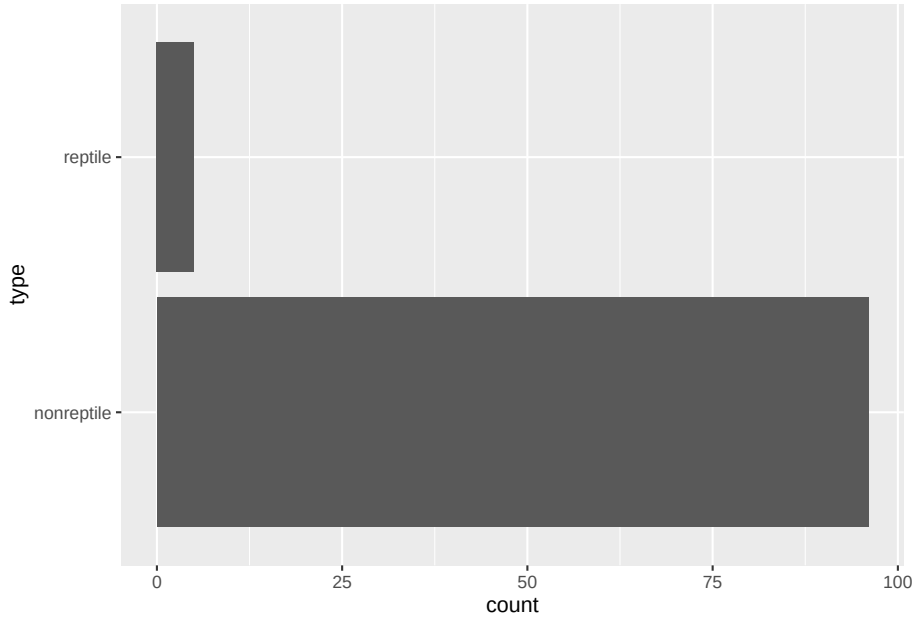
Do not forget to make the class variable a factor (a nominal variable) or you will get a regression tree instead of a classification tree.

```
summary(Zoo_reptile)
##     hair          feathers          eggs
##  Mode :logical   Mode :logical   Mode :logical
##  FALSE:58        FALSE:81        FALSE:42
##  TRUE :43        TRUE :20        TRUE :59
##
##
##
##     milk          airborne        aquatic
##  Mode :logical   Mode :logical   Mode :logical
##  FALSE:60        FALSE:77        FALSE:65
##  TRUE :41        TRUE :24        TRUE :36
##
```

```
##
##
##   predator         toothed          backbone
##  Mode :logical   Mode :logical   Mode :logical
##  FALSE:45        FALSE:40        FALSE:18
##  TRUE :56        TRUE :61        TRUE :83
##
##
##
##   breathes         venomous           fins
##  Mode :logical   Mode :logical   Mode :logical
##  FALSE:21        FALSE:93        FALSE:84
##  TRUE :80        TRUE :8         TRUE :17
##
##
##
##       legs          tail           domestic
##  Min.   :0.00   Mode :logical   Mode :logical
##  1st Qu.:2.00   FALSE:26        FALSE:88
##  Median :4.00   TRUE :75        TRUE :13
##  Mean   :2.84
##  3rd Qu.:4.00
##  Max.   :8.00
##   catsize                type
##  Mode :logical   nonreptile:96
##  FALSE:57        reptile   : 5
##  TRUE :44
##
##
##
```

See if we have a class imbalance problem.

```
ggplot(Zoo_reptile, aes(y = type)) + geom_bar()
```



Create test and training data. I use here a 50/50 split to make sure that the test set has some samples of the rare reptile class.

```
set.seed(1234)

inTrain <- createDataPartition(y = Zoo_reptile$type, p = .5)[[1]]
training_reptile <- Zoo_reptile |> slice(inTrain)
testing_reptile <- Zoo_reptile |> slice(-inTrain)
```

the new class variable is clearly not balanced. This is a problem for building a tree!

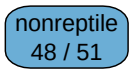
4.10.1 Option 1: Use the Data As Is and Hope For The Best

```
fit <- training_reptile |>
  train(type ~ .,
        data = _,
        method = "rpart",
        trControl = trainControl(method = "cv"))
## Warning in nominalTrainWorkflow(x = x, y = y, wts =
## weights, info = trainInfo, : There were missing values in
## resampled performance measures.
```

Warnings: "There were missing values in resampled performance measures."

means that some test folds did not contain examples of both classes. This is very likely with class imbalance and small datasets.

```
fit
## CART
##
## 51 samples
## 16 predictors
##  2 classes: 'nonreptile', 'reptile'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 46, 47, 46, 46, 45, 46, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.9467    0
##
## Tuning parameter 'cp' was held constant at a value of 0
rpart.plot(fit$finalModel, extra = 2)
```



the tree predicts everything as non-reptile. Have a look at the error on the test set.

```
confusionMatrix(data = predict(fit, testing_reptile),
                ref = testing_reptile$type, positive = "reptile")
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   nonreptile reptile
##   nonreptile         48       2
##   reptile             0       0
##
##                Accuracy : 0.96
##                  95% CI : (0.863, 0.995)
##     No Information Rate : 0.96
##     P-Value [Acc > NIR] : 0.677
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 0.480
##
##             Sensitivity : 0.00
##             Specificity : 1.00
##          Pos Pred Value :  NaN
##          Neg Pred Value : 0.96
##              Prevalence : 0.04
##          Detection Rate : 0.00
##    Detection Prevalence : 0.00
##       Balanced Accuracy : 0.50
##
##        'Positive' Class : reptile
##
```

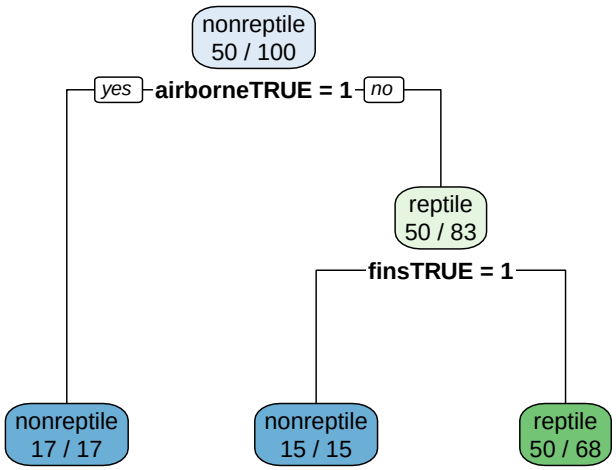
Accuracy is high, but it is exactly the same as the no-information rate and kappa is zero. Sensitivity is also zero, meaning that we do not identify any positive (reptile). If the cost of missing a positive is much larger than the cost associated with misclassifying a negative, then accuracy is not a good measure! By dealing with imbalance, we are **not** concerned with accuracy, but we want to increase the sensitivity, i.e., the chance to identify positive examples.

Note: The positive class value (the one that you want to detect) is set manually to reptile using `positive = "reptile"`. Otherwise sensitivity/specificity will not be correctly calculated.

4.10.2 Option 2: Balance Data With Resampling

We use stratified sampling with replacement (to oversample the minority/positive class). You could also use SMOTE (in package **DMwR**) or other sampling strategies (e.g., from package **unbalanced**). We use 50+50 observations here (**Note:** many samples will be chosen several times).

```
library(sampling)
##
## Attaching package: 'sampling'
## The following object is masked from 'package:caret':
##
##     cluster
set.seed(1000) # for repeatability

id <- strata(training_reptile, stratanames = "type",
             size = c(50, 50), method = "srswr")
training_reptile_balanced <- training_reptile |>
  slice(id$ID_unit)
table(training_reptile_balanced$type)
##
## nonreptile    reptile
##         50         50
fit <- training_reptile_balanced |>
  train(type ~ .,
        data = _,
        method = "rpart",
        trControl = trainControl(method = "cv"),
        control = rpart.control(minsplit = 5))

fit
## CART
##
## 100 samples
##  16 predictor
##   2 classes: 'nonreptile', 'reptile'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 90, 90, 90, 90, 90, 90, ...
## Resampling results across tuning parameters:
##
##   cp    Accuracy  Kappa
##   0.18  0.81      0.62
##   0.30  0.63      0.26
##   0.34  0.53      0.06
##
## Accuracy was used to select the optimal model using
##  the largest value.
## The final value used for the model was cp = 0.18.
rpart.plot(fit$finalModel, extra = 2)
```



Check on the unbalanced testing data.

```
confusionMatrix(data = predict(fit, testing_reptile),
                ref = testing_reptile$type, positive = "reptile")
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   nonreptile reptile
##   nonreptile         19       0
##   reptile            29       2
##
##                Accuracy : 0.42
##                  95% CI : (0.282, 0.568)
##     No Information Rate : 0.96
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.05
##
##  Mcnemar's Test P-Value : 2e-07
##
##             Sensitivity : 1.0000
##             Specificity : 0.3958
##          Pos Pred Value : 0.0645
##          Neg Pred Value : 1.0000
##              Prevalence : 0.0400
```

```
##          Detection Rate : 0.0400
##    Detection Prevalence : 0.6200
##       Balanced Accuracy : 0.6979
##
##        'Positive' Class : reptile
##
```

Note that the accuracy is below the no information rate! However, kappa (improvement of accuracy over randomness) and sensitivity (the ability to identify reptiles) have increased.

There is a tradeoff between sensitivity and specificity (how many of the identified animals are really reptiles) The tradeoff can be controlled using the sample proportions. We can sample more reptiles to increase sensitivity at the cost of lower specificity (this effect cannot be seen in the data since the test set has only a few reptiles).

```
id <- strata(training_reptile, stratanames = "type",
             size = c(50, 100), method = "srswr")
training_reptile_balanced <- training_reptile |>
  slice(id$ID_unit)
table(training_reptile_balanced$type)
##
## nonreptile    reptile
##         50        100
fit <- training_reptile_balanced |>
  train(type ~ .,
        data = _,
        method = "rpart",
        trControl = trainControl(method = "cv"),
        control = rpart.control(minsplit = 5))

confusionMatrix(data = predict(fit, testing_reptile),
                ref = testing_reptile$type, positive = "reptile")
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   nonreptile reptile
##   nonreptile         33       0
##   reptile            15       2
##
##                Accuracy : 0.7
##                  95% CI : (0.554, 0.821)
##     No Information Rate : 0.96
##     P-Value [Acc > NIR] : 1.000000
##
```

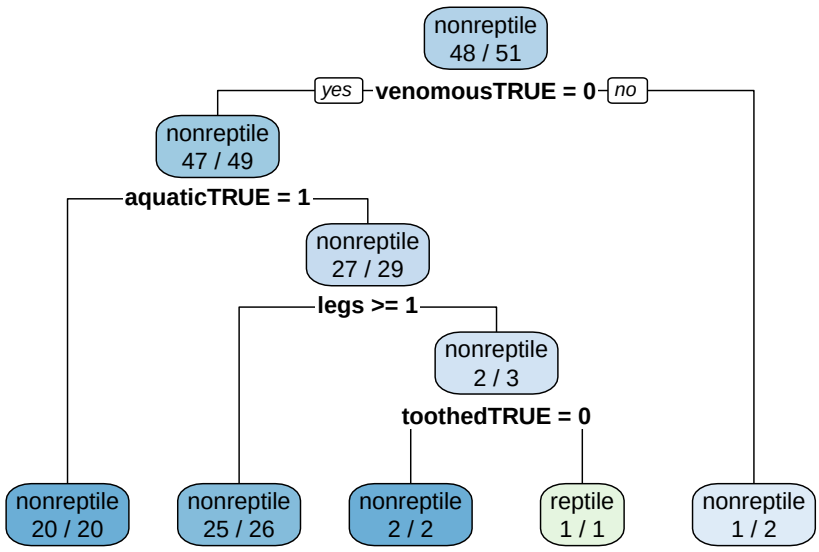
```
##                  Kappa : 0.15
##
##  Mcnemar's Test P-Value : 0.000301
##
##             Sensitivity : 1.000
##             Specificity : 0.688
##          Pos Pred Value : 0.118
##          Neg Pred Value : 1.000
##              Prevalence : 0.040
##          Detection Rate : 0.040
##    Detection Prevalence : 0.340
##       Balanced Accuracy : 0.844
##
##        'Positive' Class : reptile
##
```

4.10.3 Option 3: Build A Larger Tree and use Predicted Probabilities

Increase complexity and require less data for splitting a node. Here I also use AUC (area under the ROC) as the tuning metric. You need to specify the two class summary function. Note that the tree still trying to improve accuracy on the data and not AUC! I also enable class probabilities since I want to predict probabilities later.

```
fit <- training_reptile |>
  train(type ~ .,
        data = _,
        method = "rpart",
        tuneLength = 10,
        trControl = trainControl(method = "cv",
        classProbs = TRUE,  ## for predict with type="prob"
        summaryFunction=twoClassSummary),  ## for ROC
        metric = "ROC",
        control = rpart.control(minsplit = 3))
## Warning in nominalTrainWorkflow(x = x, y = y, wts =
## weights, info = trainInfo, : There were missing values in
## resampled performance measures.
fit
## CART
##
## 51 samples
## 16 predictors
##  2 classes: 'nonreptile', 'reptile'
##
```

```
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 46, 47, 46, 46, 46, 45, ...
## Resampling results:
##
##   ROC     Sens   Spec
##   0.3583  0.975  0
##
## Tuning parameter 'cp' was held constant at a value of 0
rpart.plot(fit$finalModel, extra = 2)
```




```
confusionMatrix(data = predict(fit, testing_reptile),
                ref = testing_reptile$type, positive = "reptile")
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   nonreptile reptile
##   nonreptile         48       2
##   reptile             0       0
##
##                Accuracy : 0.96
##                  95% CI : (0.863, 0.995)
##     No Information Rate : 0.96
##     P-Value [Acc > NIR] : 0.677
##
```

```
##                  Kappa : 0
##
##  Mcnemar's Test P-Value : 0.480
##
##             Sensitivity : 0.00
##             Specificity : 1.00
##          Pos Pred Value :  NaN
##          Neg Pred Value : 0.96
##              Prevalence : 0.04
##          Detection Rate : 0.00
##    Detection Prevalence : 0.00
##       Balanced Accuracy : 0.50
##
##        'Positive' Class : reptile
##
```

Note: Accuracy is high, but it is close or below to the no-information rate!

4.10.3.1 Create A Biased Classifier

We can create a classifier which will detect more reptiles at the expense of misclassifying non-reptiles. This is equivalent to increasing the cost of misclassifying a reptile as a non-reptile. The usual rule is to predict in each node the majority class from the test data in the node. For a binary classification problem that means a probability of >50%. In the following, we reduce this threshold to 1% or more. This means that if the new observation ends up in a leaf node with 1% or more reptiles from training then the observation will be classified as a reptile. The data set is small and this works better with more data.

```
prob <- predict(fit, testing_reptile, type = "prob")
tail(prob)
##      nonreptile reptile
## tuna     1.0000 0.00000
## vole     0.9615 0.03846
## wasp     0.5000 0.50000
## wolf     0.9615 0.03846
## worm     1.0000 0.00000
## wren     0.9615 0.03846
pred <- ifelse(prob[,"reptile"]>=0.01, "reptile", "nonreptile") |>
  as.factor()

confusionMatrix(data = pred,
                ref = testing_reptile$type, positive = "reptile")
## Confusion Matrix and Statistics
##
##             Reference
```

```
## Prediction   nonreptile reptile
##   nonreptile         13       0
##   reptile            35       2
##
##                Accuracy : 0.3
##                  95% CI : (0.179, 0.446)
##     No Information Rate : 0.96
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.029
##
##  Mcnemar's Test P-Value : 9.08e-09
##
##             Sensitivity : 1.0000
##             Specificity : 0.2708
##          Pos Pred Value : 0.0541
##          Neg Pred Value : 1.0000
##              Prevalence : 0.0400
##          Detection Rate : 0.0400
##    Detection Prevalence : 0.7400
##       Balanced Accuracy : 0.6354
##
##        'Positive' Class : reptile
##
```

Note that accuracy goes down and is below the no information rate. However, both measures are based on the idea that all errors have the same cost. What is important is that we are now able to find more reptiles.

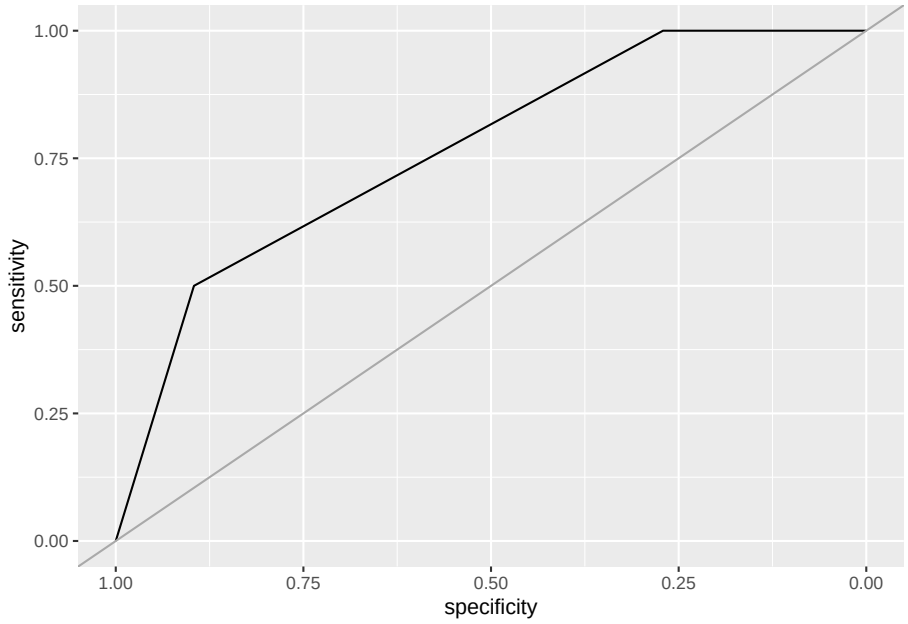
4.10.3.2 Plot the ROC Curve

Since we have a binary classification problem and a classifier that predicts a probability for an observation to be a reptile, we can also use a receiver operating characteristic (ROC)[10] curve. For the ROC curve all different cutoff thresholds for the probability are used and then connected with a line. The area under the curve represents a single number for how well the classifier works (the closer to one, the better).

```
library("pROC")
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

[10] https://en.wikipedia.org/wiki/Receiver_operating_characteristic

```
r <- roc(testing_reptile$type == "reptile", prob[,"reptile"])
## Setting levels: control = FALSE, case = TRUE
## Setting direction: controls < cases
r
##
## Call:
## roc.default(response = testing_reptile$type == "reptile", predictor = prob[,     "reptile"])
##
## Data: prob[, "reptile"] in 48 controls (testing_reptile$type == "reptile" FALSE) < 2 cases (te
## Area under the curve: 0.766
ggroc(r) + geom_abline(intercept = 1, slope = 1, color = "darkgrey")
```



4.10.4 Option 4: Use a Cost-Sensitive Classifier

The implementation of CART in `rpart` can use a cost matrix for making splitting decisions (as parameter `loss`). The matrix has the form

TP FP FN TN

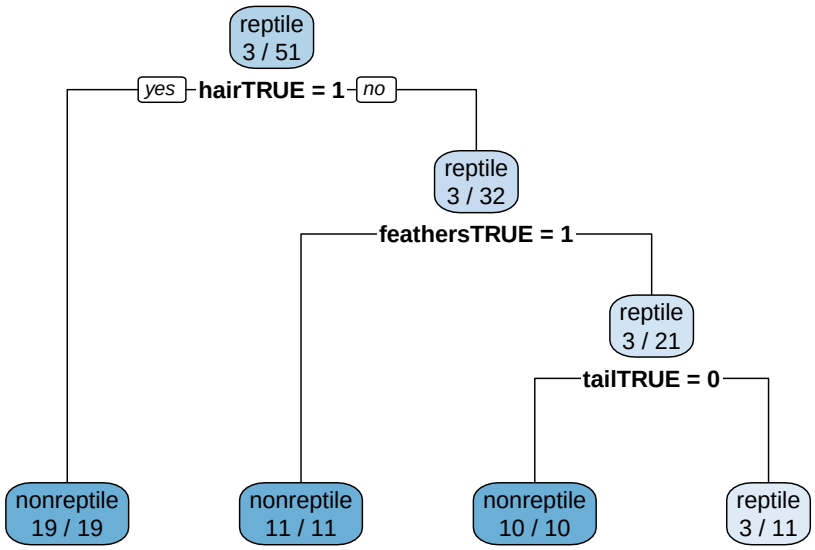
TP and TN have to be 0. We make FN very expensive (100).

```
cost <- matrix(c(
  0,   1,
  100, 0
), byrow = TRUE, nrow = 2)
cost
```

```
##      [,1] [,2]
## [1,]    0    1
## [2,]  100    0
fit <- training_reptile |>
  train(type ~ .,
        data = _,
        method = "rpart",
        parms = list(loss = cost),
        trControl = trainControl(method = "cv"))
```

The warning "There were missing values in resampled performance measures" means that some folds did not contain any reptiles (because of the class imbalance) and thus the performance measures could not be calculates.

```
fit
## CART
##
## 51 samples
## 16 predictors
##  2 classes: 'nonreptile', 'reptile'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 46, 46, 46, 45, 46, 45, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.4767    -0.03039
##
## Tuning parameter 'cp' was held constant at a value of 0
rpart.plot(fit$finalModel, extra = 2)
```




```
confusionMatrix(data = predict(fit, testing_reptile),
                ref = testing_reptile$type, positive = "reptile")
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   nonreptile reptile
##   nonreptile         39       0
##   reptile             9       2
##
##                Accuracy : 0.82
##                  95% CI : (0.686, 0.914)
##     No Information Rate : 0.96
##     P-Value [Acc > NIR] : 0.99998
##
##                   Kappa : 0.257
##
##  Mcnemar's Test P-Value : 0.00766
##
##             Sensitivity : 1.000
##             Specificity : 0.812
##          Pos Pred Value : 0.182
##          Neg Pred Value : 1.000
##              Prevalence : 0.040
##          Detection Rate : 0.040
##    Detection Prevalence : 0.220
```

```
##       Balanced Accuracy : 0.906
##
##        'Positive' Class : reptile
##
```

The high cost for false negatives results in a classifier that does not miss any reptile.

Note: Using a cost-sensitive classifier is often the best option. Unfortunately, the most classification algorithms (or their implementation) do not have the ability to consider misclassification cost.

4.11 Model Comparison

We first create a weak baseline model that always predicts the the majority class mammal.

```
baselineFit <- Zoo_train |> train(type ~ .,
  method = basemodels::dummyClassifier,
  data = _,
  strategy = "constant",
  constant = "mammal",
  trControl = trainControl(method = "cv",
                           index = train_index))
baselineFit
## dummyClassifier
##
## 83 samples
## 16 predictors
##  7 classes: 'mammal', 'bird', 'reptile', 'fish', 'amphibian', 'insect', 'mollusc.et
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 74, 75, 75, 75, 75, 74, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.4011    0
```

The kappa of 0 clearly indicates that the baseline model has no power.

We collect the performance metrics from the models trained on the same data.

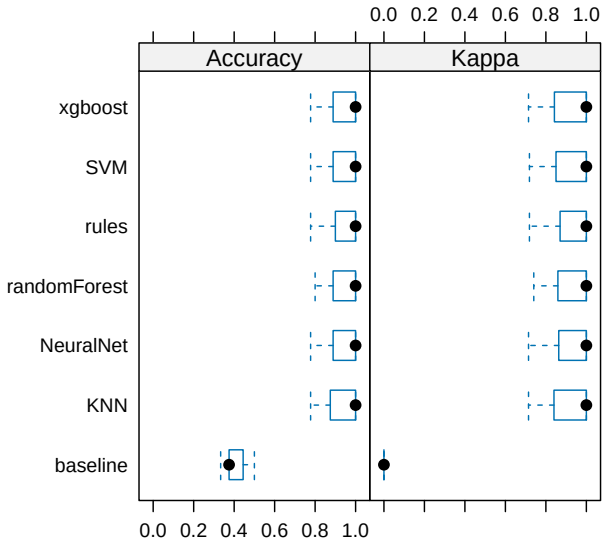
```
resamps <- resamples(list(
  baseline = baselineFit,
  SVM = svmFit,
  KNN = knnFit,
```

```
  rules = rulesFit,
  randomForest = randomForestFit,
  xgboost = xgboostFit,
  NeuralNet = nnetFit
    ))
resamps
##
## Call:
## resamples.default(x = list(baseline = baselineFit, SVM
##  = randomForestFit, xgboost = xgboostFit, NeuralNet
##  = nnetFit))
##
## Models: baseline, SVM, KNN, rules, randomForest, xgboost, NeuralNet
## Number of resamples: 10
## Performance metrics: Accuracy, Kappa
## Time estimates for: everything, final model fit
```

The summary statistics shows the performance. We see that all methods do well on this easy data set, only the baseline model performs as expected poorly.

```
summary(resamps)
##
## Call:
## summary.resamples(object = resamps)
##
## Models: baseline, SVM, KNN, rules, randomForest, xgboost, NeuralNet
## Number of resamples: 10
##
## Accuracy
##                 Min. 1st Qu. Median   Mean 3rd Qu. Max. NA's
## baseline     0.3333  0.3750  0.375 0.4011  0.4333  0.5    0
## SVM          0.7778  0.8889  1.000 0.9356  1.0000  1.0    0
## KNN          0.7778  0.8785  1.000 0.9342  1.0000  1.0    0
## rules        0.7778  0.9250  1.000 0.9567  1.0000  1.0    0
## randomForest 0.8000  0.8889  1.000 0.9467  1.0000  1.0    0
## xgboost      0.7778  0.8917  1.000 0.9442  1.0000  1.0    0
## NeuralNet    0.7778  0.8917  1.000 0.9456  1.0000  1.0    0
##
## Kappa
##                 Min. 1st Qu. Median   Mean 3rd Qu. Max. NA's
## baseline     0.0000  0.0000      0 0.0000       0    0    0
## SVM          0.7188  0.8534      1 0.9173       1    1    0
## KNN          0.7143  0.8459      1 0.9158       1    1    0
## rules        0.7188  0.9026      1 0.9439       1    1    0
## randomForest 0.7403  0.8604      1 0.9313       1    1    0
## xgboost      0.7143  0.8491      1 0.9267       1    1    0
```

```
## NeuralNet    0.7143  0.8644      1 0.9295       1    1    0
library(lattice)
bwplot(resamps, layout = c(3, 1))
```



Perform inference about differences between models. For each metric, all pairwise differences are computed and tested to assess if the difference is equal to zero. By default Bonferroni correction for multiple comparison is used. Differences are shown in the upper triangle and p-values are in the lower triangle.

```
difs <- diff(resamps)
summary(difs)
##
## Call:
## summary.diff.resamples(object = difs)
##
## p-value adjustment: bonferroni
## Upper diagonal: estimates of the difference
## Lower diagonal: p-value for H0: difference = 0
##
## Accuracy
##              baseline SVM      KNN      rules
## baseline              -0.53444 -0.53306 -0.55556
## SVM          2.76e-07           0.00139 -0.02111
## KNN          9.97e-08 1                 -0.02250
## rules        1.48e-07 1        1
```

```
## randomForest 1.26e-07 1        1        1
## xgboost      1.44e-07 1        1        1
## NeuralNet    6.35e-08 1        1        1
##              randomForest xgboost  NeuralNet
## baseline     -0.54556     -0.54306 -0.54444
## SVM          -0.01111     -0.00861 -0.01000
## KNN          -0.01250     -0.01000 -0.01139
## rules         0.01000      0.01250  0.01111
## randomForest               0.00250  0.00111
## xgboost      1                     -0.00139
## NeuralNet    1            1
##
## Kappa
##              baseline SVM      KNN      rules
## baseline              -0.91726 -0.91582 -0.94389
## SVM          2.46e-08           0.00145 -0.02662
## KNN          2.81e-08 1                 -0.02807
## rules        4.69e-09 1        1
## randomForest 3.88e-09 1        1        1
## xgboost      8.27e-09 1        1        1
## NeuralNet    6.49e-09 1        1        1
##              randomForest xgboost  NeuralNet
## baseline     -0.93133     -0.92665 -0.92946
## SVM          -0.01406     -0.00939 -0.01219
## KNN          -0.01551     -0.01083 -0.01364
## rules         0.01256      0.01724  0.01443
## randomForest               0.00468  0.00187
## xgboost      1                     -0.00281
## NeuralNet    1            1
```

All perform similarly well except the baseline model (differences in the first row are negative and the p-values in the first column are $<.05$ indicating that the null-hypothesis of a difference of 0 can be rejected).

Most models do similarly well on the data. We choose here the random forest model and evaluate its generalization performance on the held-out test set.

```
pr <- predict(randomForestFit, Zoo_test)
pr
##  [1] mammal        fish          mollusc.et.al fish
##  [5] mammal        insect        mammal        mammal
##  [9] mammal        mammal        bird          mammal
## [13] mammal        bird          reptile       bird
## [17] mollusc.et.al bird
## 7 Levels: mammal bird reptile fish amphibian ... mollusc.et.al
```

Calculate the confusion matrix for the held-out test data.

```
confusionMatrix(pr, reference = Zoo_test$type)
## Confusion Matrix and Statistics
##
##                Reference
## Prediction      mammal bird reptile fish amphibian insect
##   mammal             8    0       0    0         0      0
##   bird               0    4       0    0         0      0
##   reptile            0    0       1    0         0      0
##   fish               0    0       0    2         0      0
##   amphibian          0    0       0    0         0      0
##   insect             0    0       0    0         0      1
##   mollusc.et.al      0    0       0    0         0      0
##                Reference
## Prediction      mollusc.et.al
##   mammal                    0
##   bird                      0
##   reptile                   0
##   fish                      0
##   amphibian                 0
##   insect                    0
##   mollusc.et.al             2
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.815, 1)
##     No Information Rate : 0.444
##     P-Value [Acc > NIR] : 4.58e-07
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: mammal Class: bird
## Sensitivity                  1.000       1.000
## Specificity                  1.000       1.000
## Pos Pred Value               1.000       1.000
## Neg Pred Value               1.000       1.000
## Prevalence                   0.444       0.222
## Detection Rate               0.444       0.222
## Detection Prevalence         0.444       0.222
## Balanced Accuracy            1.000       1.000
##                      Class: reptile Class: fish
```

```
## Sensitivity                 1.0000       1.000
## Specificity                 1.0000       1.000
## Pos Pred Value              1.0000       1.000
## Neg Pred Value              1.0000       1.000
## Prevalence                  0.0556       0.111
## Detection Rate              0.0556       0.111
## Detection Prevalence        0.0556       0.111
## Balanced Accuracy           1.0000       1.000
##                      Class: amphibian Class: insect
## Sensitivity                        NA        1.0000
## Specificity                         1        1.0000
## Pos Pred Value                     NA        1.0000
## Neg Pred Value                     NA        1.0000
## Prevalence                          0        0.0556
## Detection Rate                      0        0.0556
## Detection Prevalence                0        0.0556
## Balanced Accuracy                  NA        1.0000
##                      Class: mollusc.et.al
## Sensitivity                         1.000
## Specificity                         1.000
## Pos Pred Value                      1.000
## Neg Pred Value                      1.000
## Prevalence                          0.111
## Detection Rate                      0.111
## Detection Prevalence                0.111
## Balanced Accuracy                   1.000
```

4.12 Comparing Decision Boundaries of Popular Classification Techniques*

Classifiers create decision boundaries to discriminate between classes. Different classifiers are able to create different shapes of decision boundaries (e.g., some are strictly linear) and thus some classifiers may perform better for certain datasets. In this section, we visualize the decision boundaries found by several popular classification methods.

The following function defines a plot that adds the decision boundary (black lines) and classification confidence (color intensity) by evaluating the classifier at evenly spaced grid points. Note that low resolution will make evaluation faster but it also will make the decision boundary look like it has small steps even if it is a straight line.

```
library(tidyverse)
library(ggplot2)
library(scales)
```

```
library(caret)
```

```
decisionplot <- function(model, data, class_var,
  predict_type = c("class", "prob"), resolution = 3 * 72) {
  # resolution is set to 72 dpi for 3 inches wide images.

  y <- data |> pull(class_var)
  x <- data |> dplyr::select(-all_of(class_var))

  # resubstitution accuracy
  prediction <- predict(model, x, type = predict_type[1])

  # LDA returns a list
  if(is.list(prediction)) prediction <- prediction$class

  prediction <- factor(prediction, levels = levels(y))
  cm <- confusionMatrix(data = prediction,
                        reference = y)
  acc <- cm$overall["Accuracy"]

  # evaluate model on a grid
  r <- sapply(x[, 1:2], range, na.rm = TRUE)
  xs <- seq(r[1,1], r[2,1], length.out = resolution)
  ys <- seq(r[1,2], r[2,2], length.out = resolution)
  g <- cbind(rep(xs, each = resolution), rep(ys,
                                             time = resolution))
  colnames(g) <- colnames(r)
  g <- as_tibble(g)

  # guess how to get class labels from predict
  # (unfortunately not very consistent between models)
  cl <- predict(model, g, type = predict_type[1])

  prob <- NULL
  if(is.list(cl)) { # LDA returns a list
    prob <- cl$posterior
    cl <- cl$class
  } else
    if(!is.na(predict_type[2]))
      try(prob <- predict(model, g, type = predict_type[2]))

  # We visualize the difference in probability/score between
  # the winning class and the second best class. We only use
  # probability if the classifier's predict function supports it.
  max_prob <- 1
```

```
  if(!is.null(prob))
    try({
      max_prob <- t(apply(prob, MARGIN = 1, sort, decreasing = TRUE))
      max_prob <- max_prob[,1] - max_prob[,2]
    }, silent = TRUE)

  cl <- factor(cl, levels = levels(y))

  g <- g |> add_column(prediction = cl,
                       probability = max_prob)

  ggplot(g, mapping = aes(
    x = .data[[colnames(g)[1]]],
    y = .data[[colnames(g)[2]]])) +
    geom_raster(mapping = aes(fill = prediction,
                              alpha = probability)) +
    geom_contour(mapping = aes(z = as.numeric(prediction)),
      bins = length(levels(cl)),
      linewidth = .5,
      color = "black") +
    geom_point(data = data, mapping =  aes(
      x = .data[[colnames(data)[1]]],
      y = .data[[colnames(data)[2]]],
      shape = .data[[class_var]]),
      alpha = .7) +
    scale_alpha_continuous(range = c(0,1),
                           limits = c(0,1),
                           guide = "none") +
    labs(subtitle = paste("Training accuracy:", round(acc, 2)))
}
```

4.12.1 Iris Dataset

For easier visualization, we use two dimensions of the Iris dataset.

```
set.seed(1000)
data(iris)
iris <- as_tibble(iris)

x <- iris |> dplyr::select(Sepal.Length, Sepal.Width, Species)
# Note: package MASS overwrites the select function.

x
## # A tibble: 150 x 3
##    Sepal.Length Sepal.Width Species
```

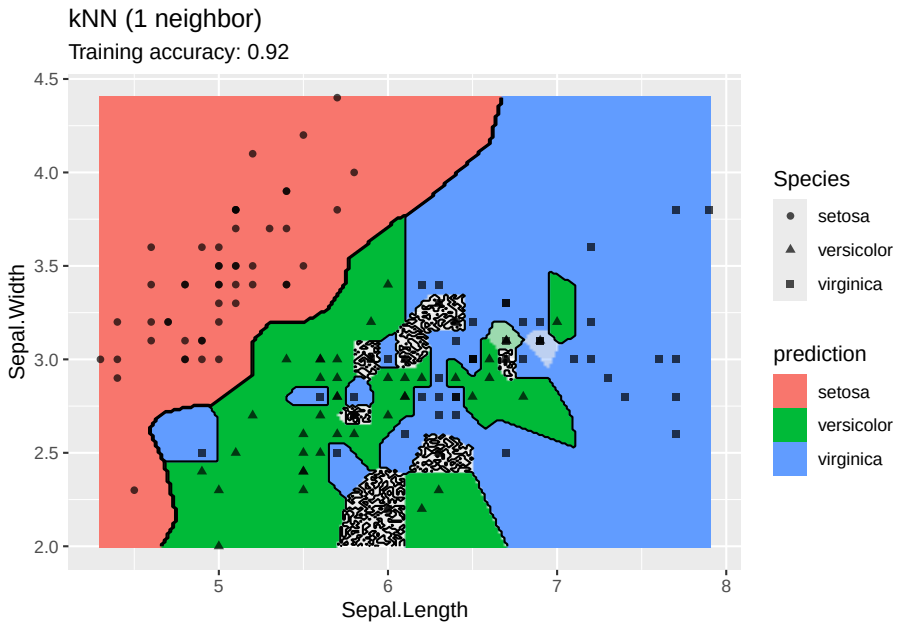
```
##            <dbl>       <dbl> <fct>
##  1          5.1         3.5 setosa
##  2          4.9         3   setosa
##  3          4.7         3.2 setosa
##  4          4.6         3.1 setosa
##  5          5           3.6 setosa
##  6          5.4         3.9 setosa
##  7          4.6         3.4 setosa
##  8          5           3.4 setosa
##  9          4.4         2.9 setosa
## 10          4.9         3.1 setosa
## # i 140 more rows
ggplot(x, aes(x = Sepal.Length,
              y = Sepal.Width,
              fill = Species)) +
  stat_density_2d(geom = "polygon",
                  aes(alpha = after_stat(level))) +
  geom_point()
```

This is the original data. Color is used to show the density. Note that there is some overplotting with several points in the same position. You could use `geom_jitter()` instead of `geom_point()`.

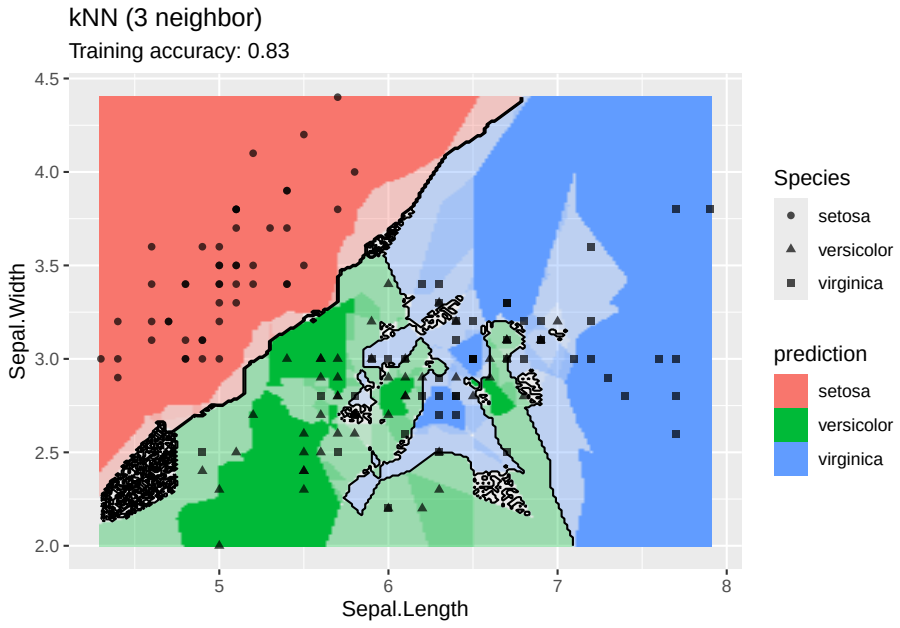
4.12.1.1 Nearest Neighbor Classifier

We try several values for k.

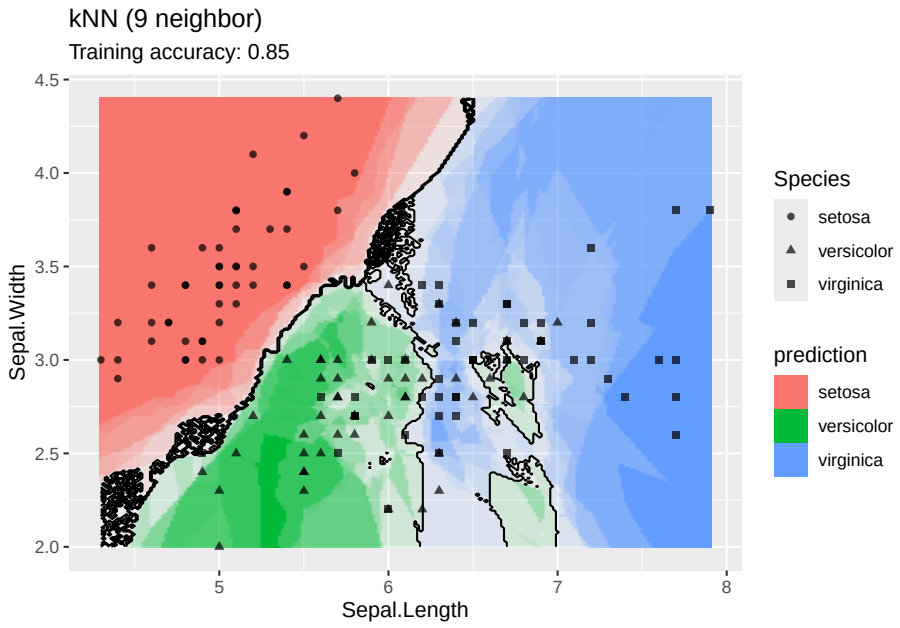
```
model <- x |> caret::knn3(Species ~ ., data = _, k = 1)
decisionplot(model, x, class_var = "Species") +
  labs(title = "kNN (1 neighbor)")
```




```
model <- x |> caret::knn3(Species ~ ., data = _, k = 3)
decisionplot(model, x, class_var = "Species") +
  labs(title = "kNN (3 neighbor)")
```




```
model <- x |> caret::knn3(Species ~ ., data = _, k = 9)
decisionplot(model, x, class_var = "Species") +
  labs(title = "kNN (9 neighbor)")
```



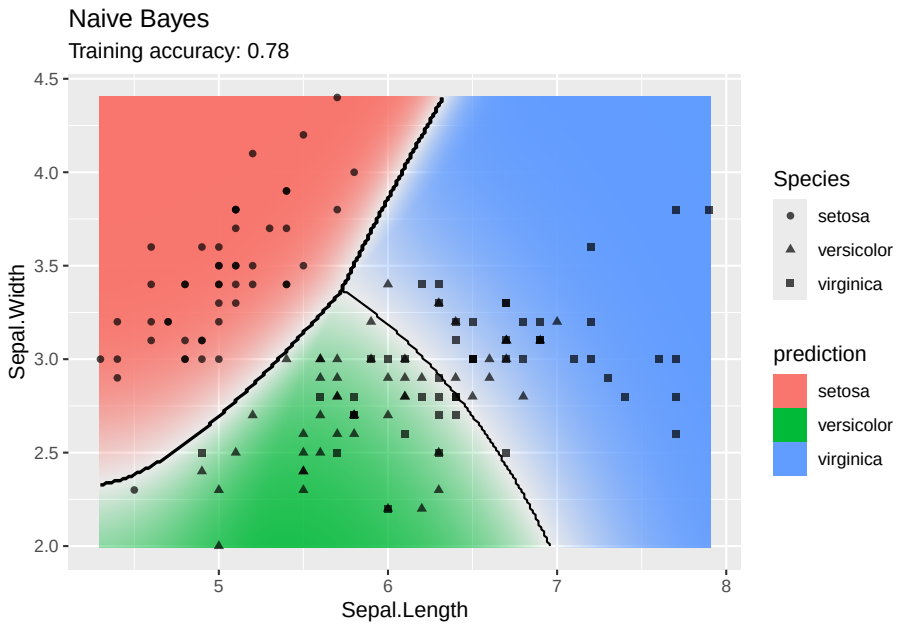
Increasing k smooths the decision boundary. At $k = 1$, we see white areas around

points where flowers of two classes are in the same spot. Here, the algorithm randomly chooses a class during prediction resulting in the meandering decision boundary. The predictions in that area are not stable and every time we ask for a class, we may get a different class.

4.12.1.2 Naive Bayes Classifier

Use a Gaussian naive Bayes classifier.

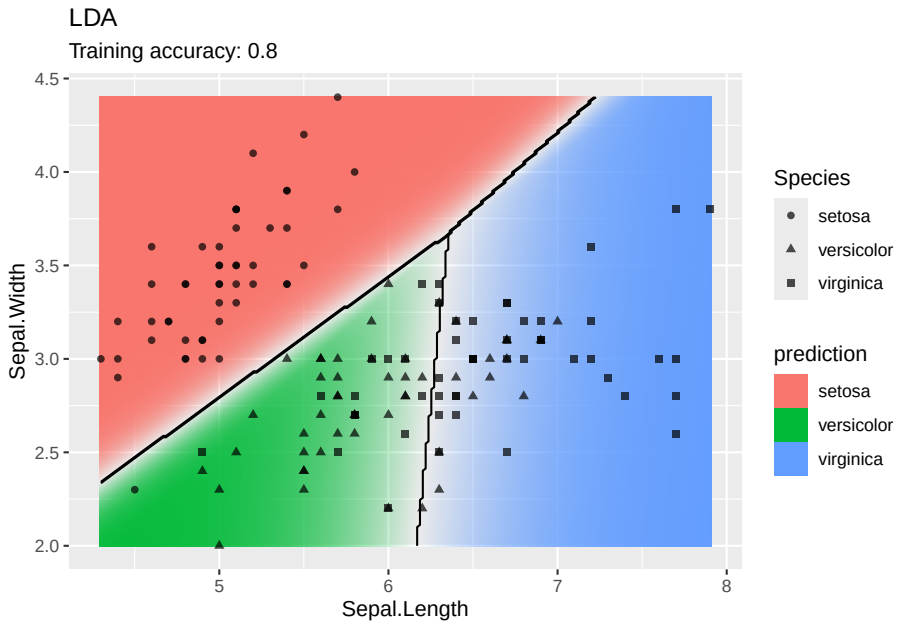
```
model <- x |> e1071::naiveBayes(Species ~ ., data = _)
decisionplot(model, x, class_var = "Species",
             predict_type = c("class", "raw")) +
  labs(title = "Naive Bayes")
```



4.12.1.3 Linear Discriminant Analysis

LDA finds linear decision boundaries.

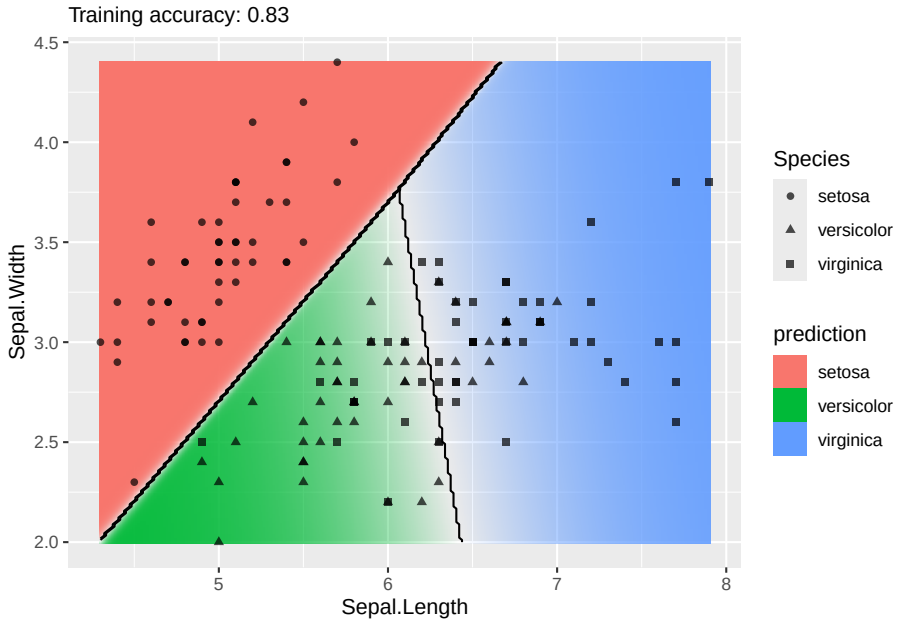
```
model <- x |> MASS::lda(Species ~ ., data = _)
decisionplot(model, x, class_var = "Species") +
  labs(title = "LDA")
```



4.12.1.4 Multinomial Logistic Regression

Multinomial logistic regression is an extension of logistic regression to problems with more than two classes.

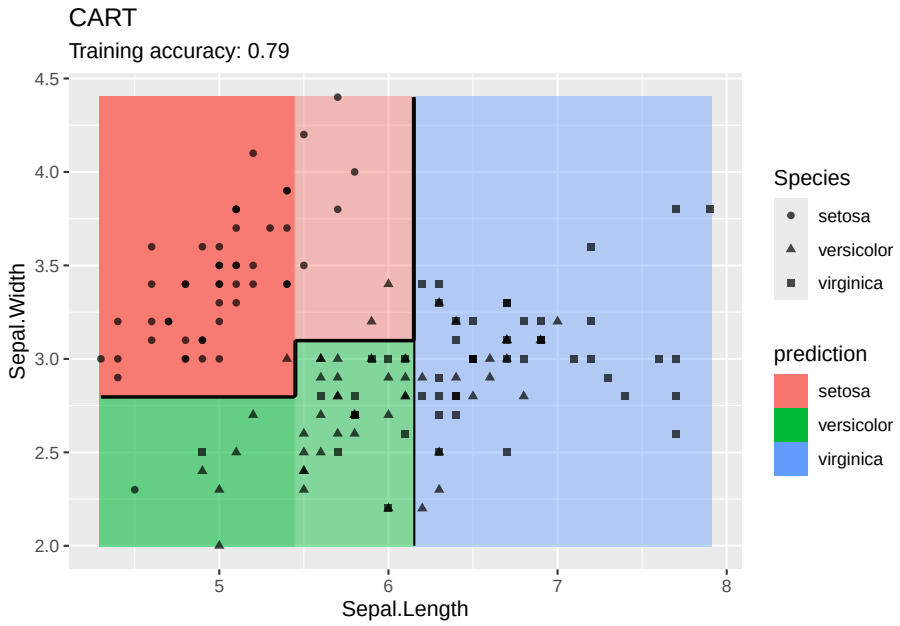
```
model <- x |> nnet::multinom(Species ~., data = _)
## # weights:  12 (6 variable)
## initial  value 164.791843
## iter  10 value 62.715967
## iter  20 value 59.808291
## iter  30 value 55.445984
## iter  40 value 55.375704
## iter  50 value 55.346472
## iter  60 value 55.301707
## iter  70 value 55.253532
## iter  80 value 55.243230
## iter  90 value 55.230241
## iter 100 value 55.212479
## final  value 55.212479
## stopped after 100 iterations
decisionplot(model, x, class_var = "Species") +
  labs(titel = "Multinomial Logistic Regression")
```



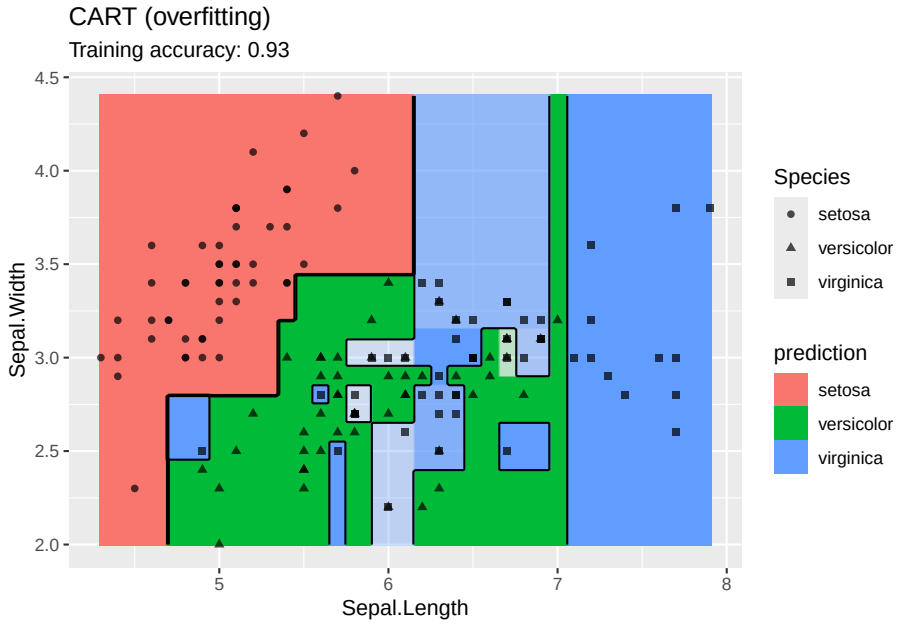
4.12.1.5 Decision Trees

Compare different types of decision trees.

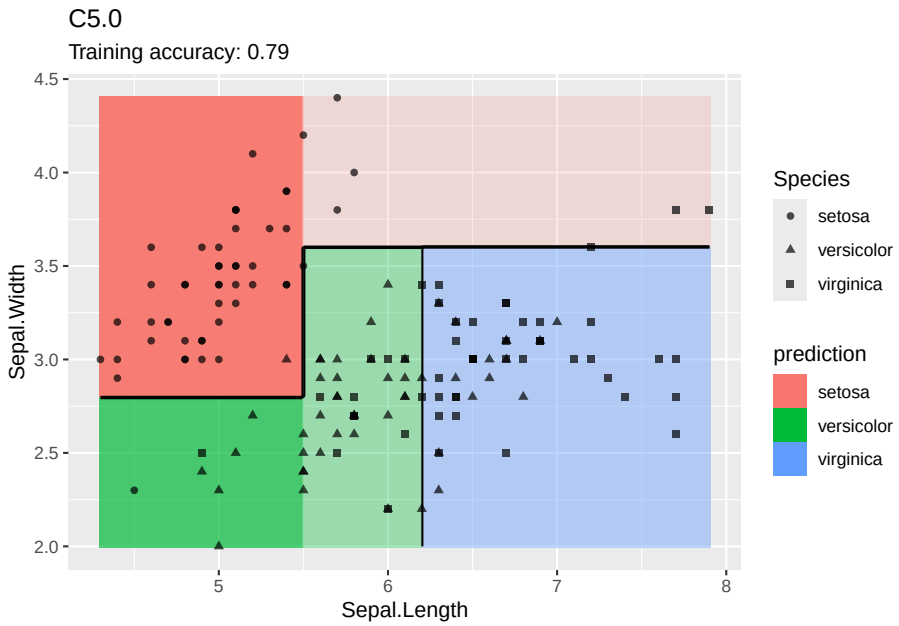
```
model <- x |> rpart::rpart(Species ~ ., data = _)
decisionplot(model, x, class_var = "Species") +
  labs(title = "CART")
```




```
model <- x |> rpart::rpart(Species ~ ., data = _,
  control = rpart::rpart.control(cp = 0.001, minsplit = 1))
decisionplot(model, x, class_var = "Species") +
  labs(title = "CART (overfitting)")
```



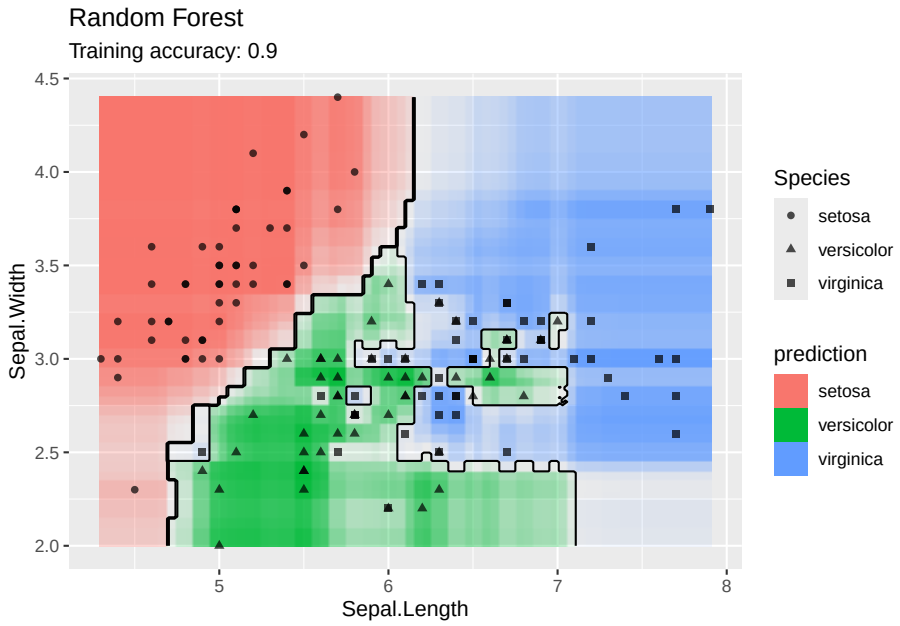
```
model <- x |> C50::C5.0(Species ~ ., data = _)
decisionplot(model, x, class_var = "Species") +
  labs(title = "C5.0")
```



4.12.1.6 Ensemble: Random Forest

Use an ensemble method.

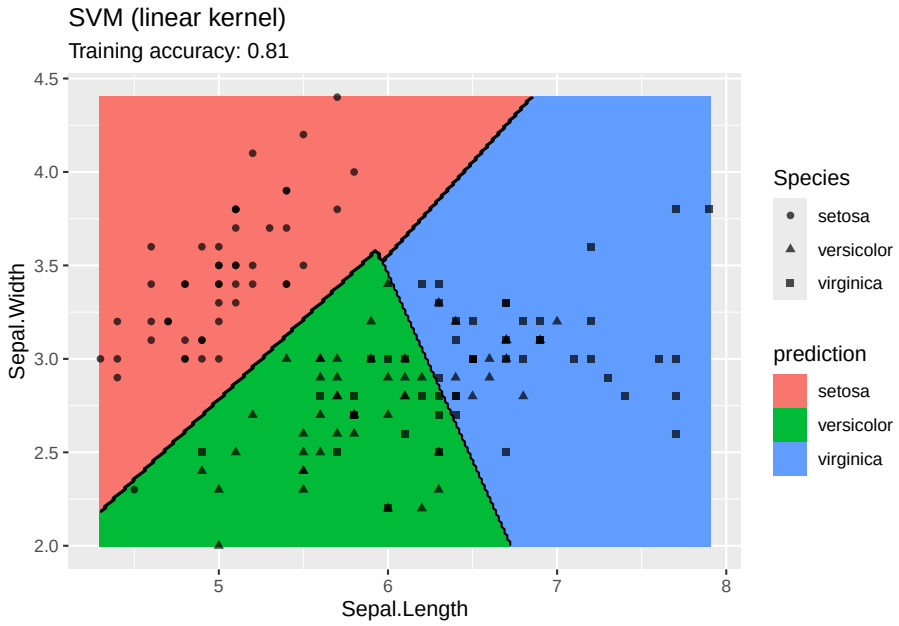
```
model <- x |> randomForest::randomForest(Species ~ ., data = _)
decisionplot(model, x, class_var = "Species") +
  labs(title = "Random Forest")
```



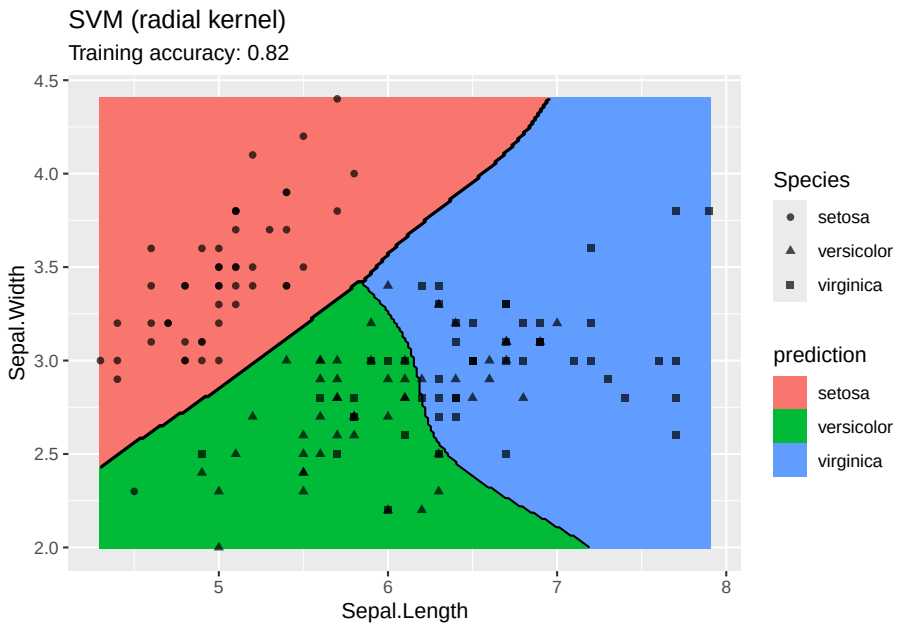
4.12.1.7 Support Vector Machine

Compare SVMs with different kernel functions.

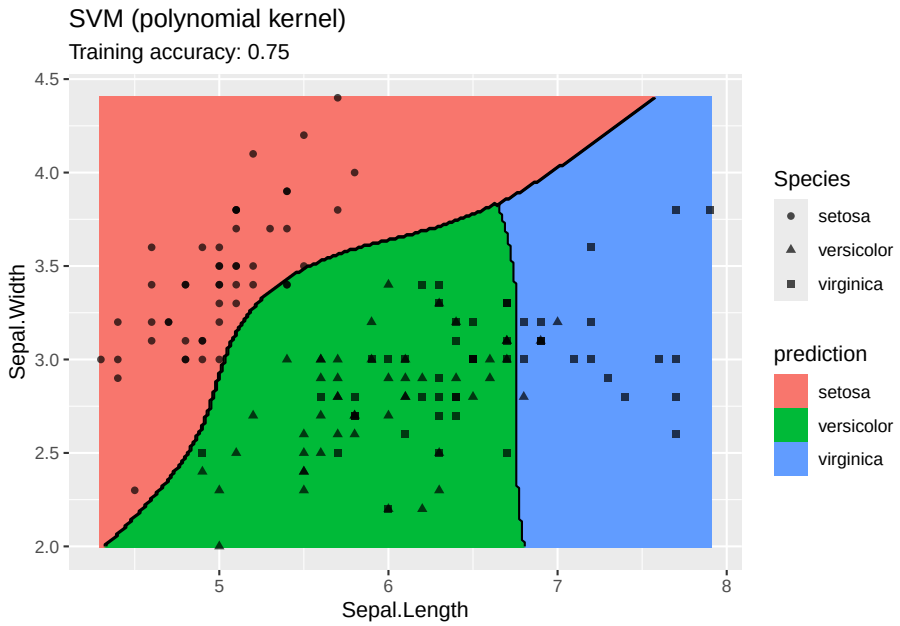
```
model <- x |> e1071::svm(Species ~ ., data = _,
                         kernel = "linear")
decisionplot(model, x, class_var = "Species") +
  labs(title = "SVM (linear kernel)")
```



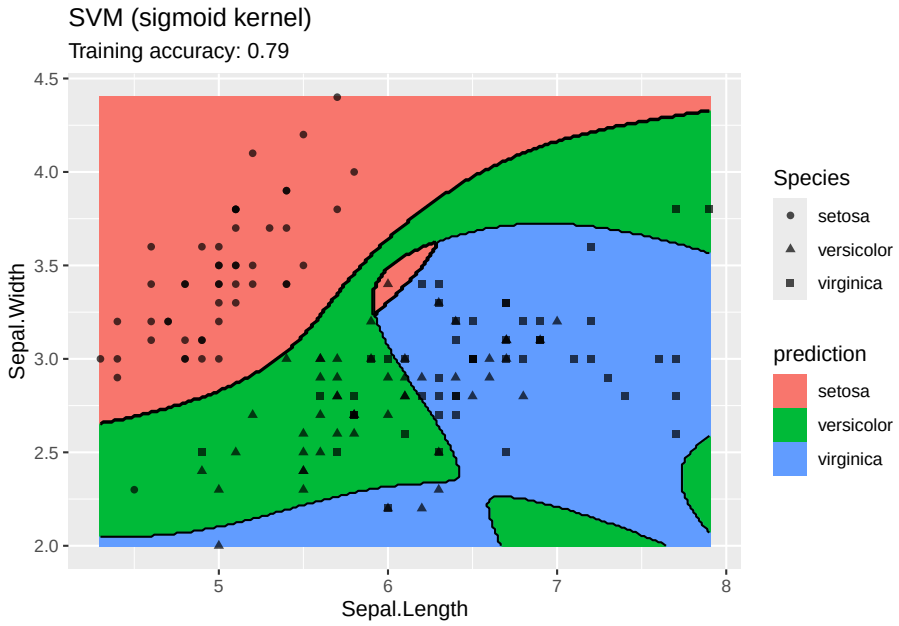

```
model <- x |> e1071::svm(Species ~ ., data = _,
                         kernel = "radial")
decisionplot(model, x, class_var = "Species") +
  labs(title = "SVM (radial kernel)")
```



```
model <- x |> e1071::svm(Species ~ ., data = _,
                         kernel = "polynomial")
decisionplot(model, x, class_var = "Species") +
  labs(title = "SVM (polynomial kernel)")
```




```
model <- x |> e1071::svm(Species ~ ., data = _,
                         kernel = "sigmoid")
decisionplot(model, x, class_var = "Species") +
  labs(title = "SVM (sigmoid kernel)")
```

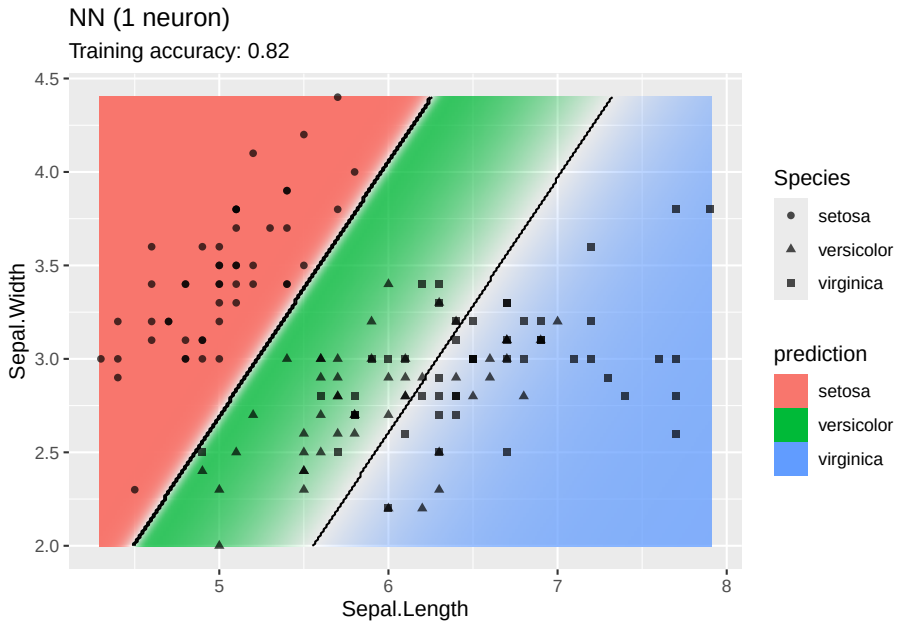


Most kernels do well only the sigmoid kernel seems to find a very strange decision boundary.

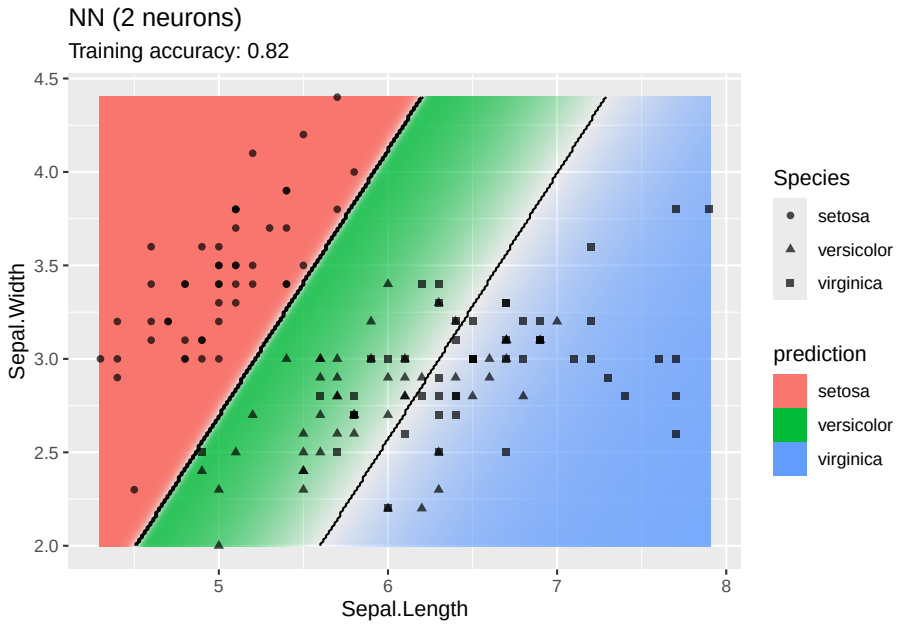
4.12.1.8 Single Layer Feed-forward Neural Networks

Use a simple network with one hidden layer. We will try a different number of neurons for the hidden layer.

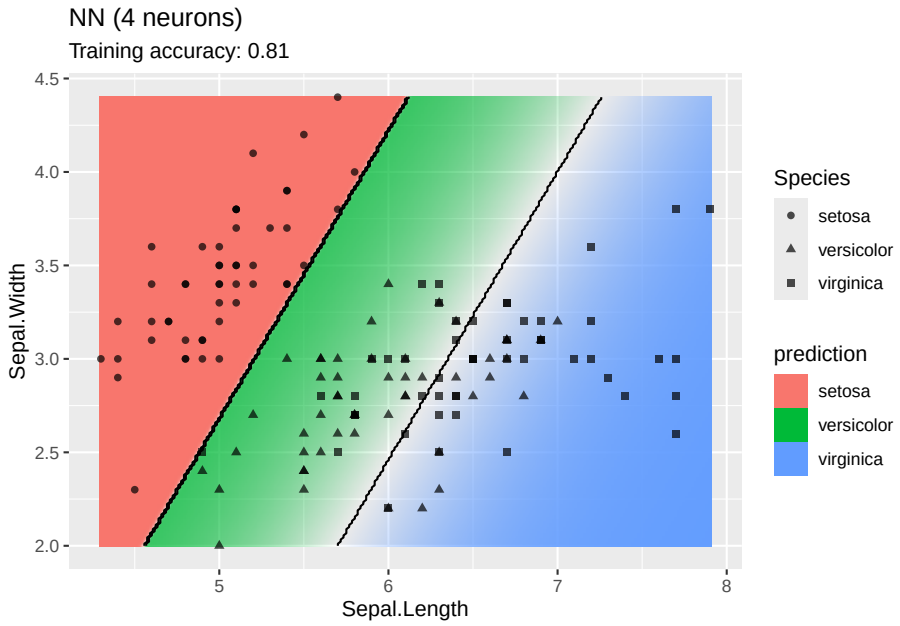
```
model <-x |> nnet::nnet(Species ~ ., data = _,
                        size = 1, trace = FALSE)
decisionplot(model, x, class_var  = "Species",
             predict_type = c("class", "raw")) +
  labs(title = "NN (1 neuron)")
```



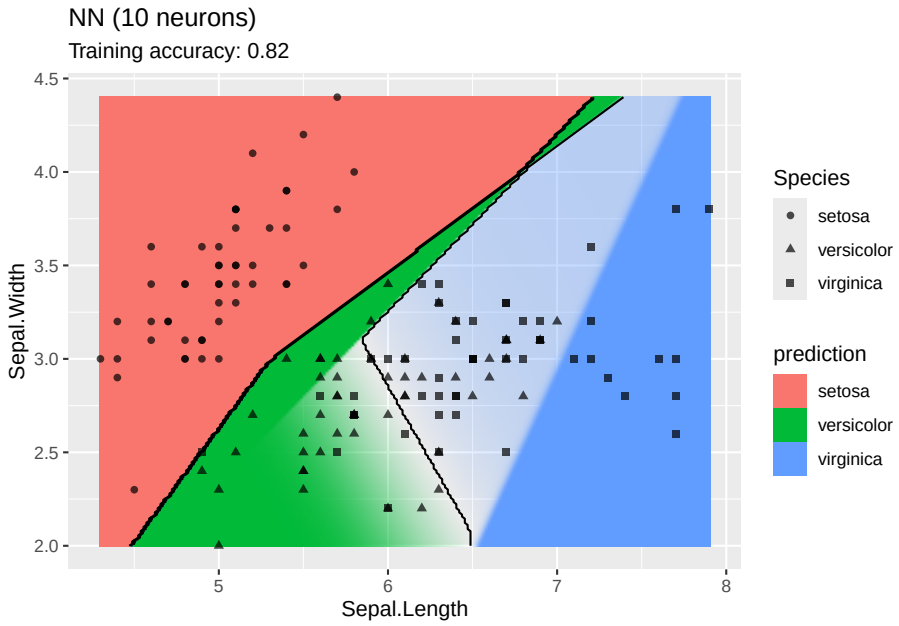

```
model <-x |> nnet::nnet(Species ~ ., data = _,
                        size = 2, trace = FALSE)
decisionplot(model, x, class_var  = "Species",
             predict_type = c("class", "raw")) +
  labs(title = "NN (2 neurons)")
```




```
model <-x |> nnet::nnet(Species ~ ., data = _,
                        size = 4, trace = FALSE)
decisionplot(model, x, class_var  = "Species",
             predict_type = c("class", "raw")) +
  labs(title = "NN (4 neurons)")
```



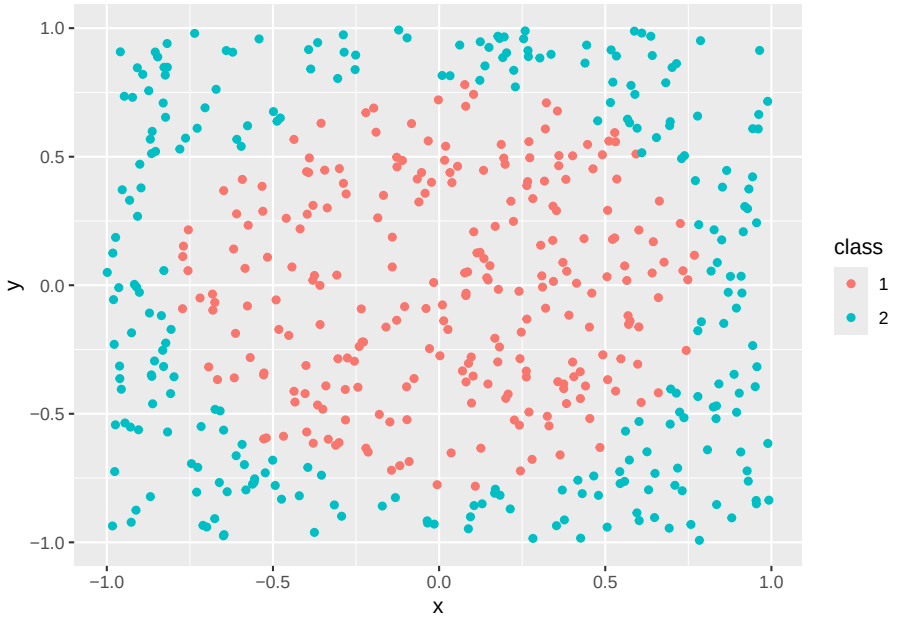

```
model <-x |> nnet::nnet(Species ~ ., data = _,
                        size = 10, trace = FALSE)
decisionplot(model, x, class_var  = "Species",
             predict_type = c("class", "raw")) +
  labs(title = "NN (10 neurons)")
```



4.12.2 Circle Dataset

```
set.seed(1000)

x <- mlbench::mlbench.circle(500)

# You can also experiment with the following datasets.
#x <- mlbench::mlbench.cassini(500)
#x <- mlbench::mlbench.spirals(500, sd = .1)
#x <- mlbench::mlbench.smiley(500)

x <- cbind(as.data.frame(x$x), factor(x$classes))
colnames(x) <- c("x", "y", "class")
x <- as_tibble(x)
x
## # A tibble: 500 x 3
##          x       y class
##      <dbl>   <dbl> <fct>
##  1 -0.344   0.448  1
##  2  0.518   0.915  2
##  3 -0.772  -0.0913 1
##  4  0.382   0.412  1
##  5  0.0328  0.438  1
##  6 -0.865  -0.354  2
```

```
##  7  0.477   0.640  2
##  8  0.167  -0.809  2
##  9 -0.568  -0.281  1
## 10 -0.488   0.638  2
## # i 490 more rows
ggplot(x, aes(x = x, y = y, color = class)) +
  geom_point()
```

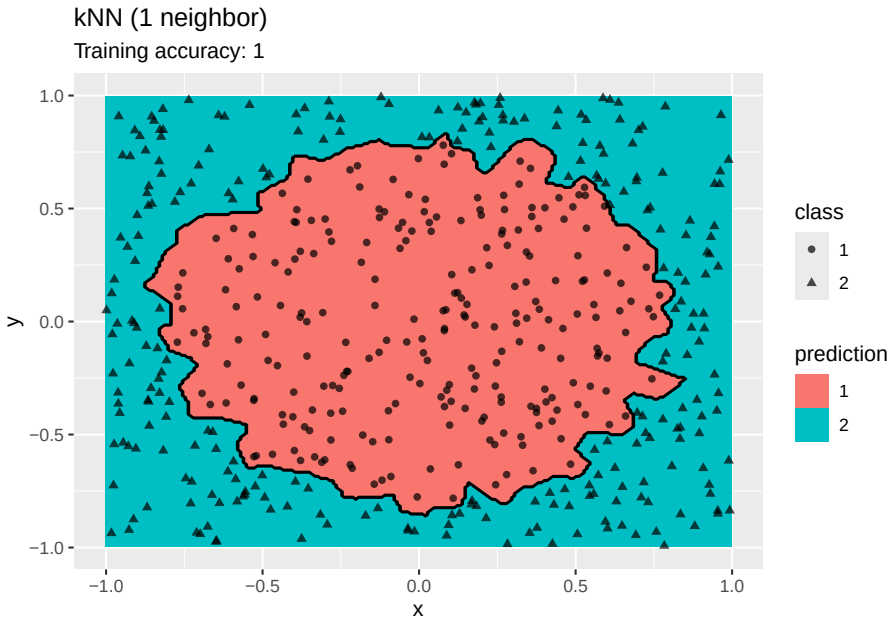


This dataset is challenging for some classification algorithms since the optimal decision boundary is a circle around the class in the center.

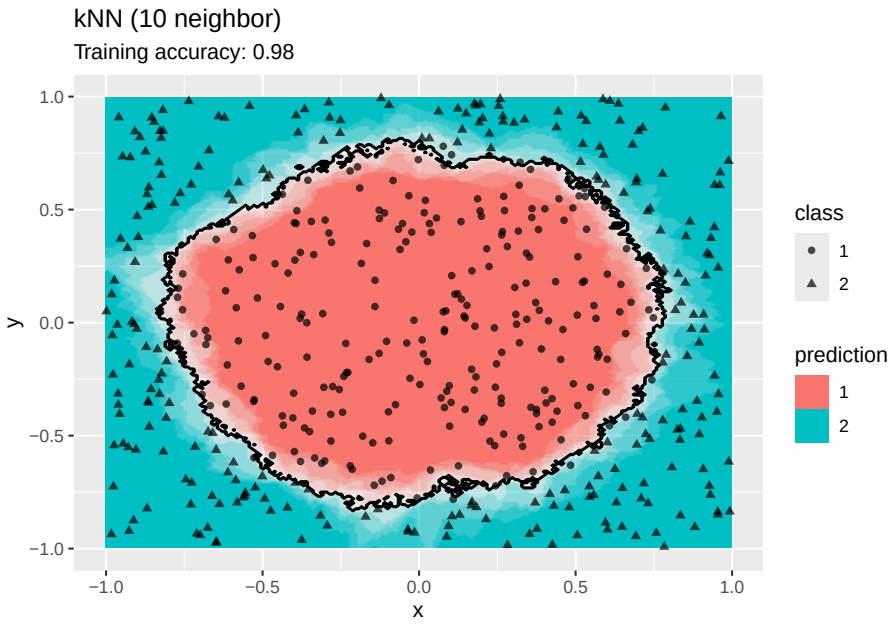
4.12.2.1 Nearest Neighbor Classifier

Compare kNN classifiers with different values for k.

```
model <- x |> caret::knn3(class ~ ., data = _, k = 1)
decisionplot(model, x, class_var = "class") +
  labs(title = "kNN (1 neighbor)")
```




```
model <- x |> caret::knn3(class ~ ., data = _, k = 10)
decisionplot(model, x, class_var = "class") +
  labs(title = "kNN (10 neighbor)")
```



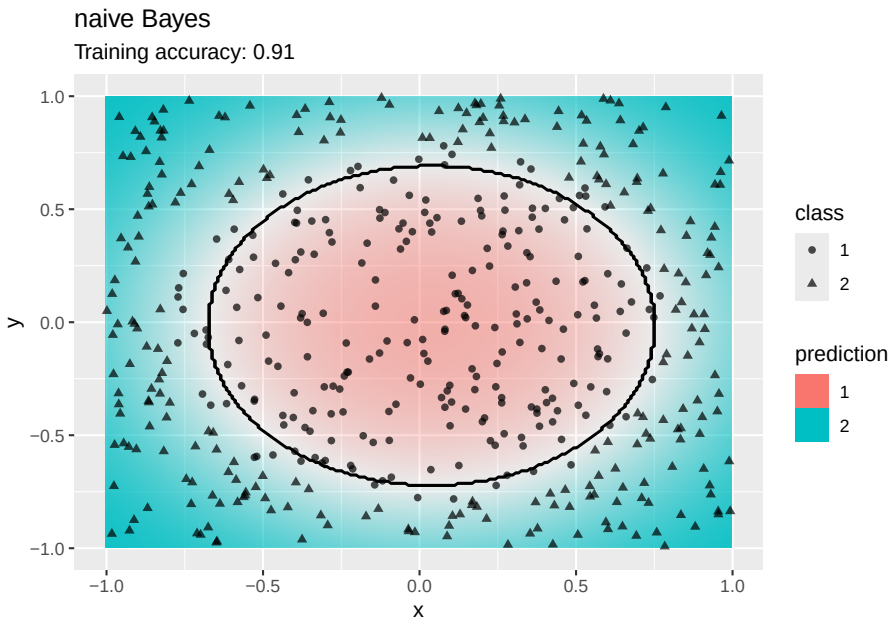
k-Nearest does not find a smooth decision boundary, but tends to overfit the

training data at low values for k.

4.12.2.2 Naive Bayes Classifier

The Gaussian naive Bayes classifier works very well on the data.

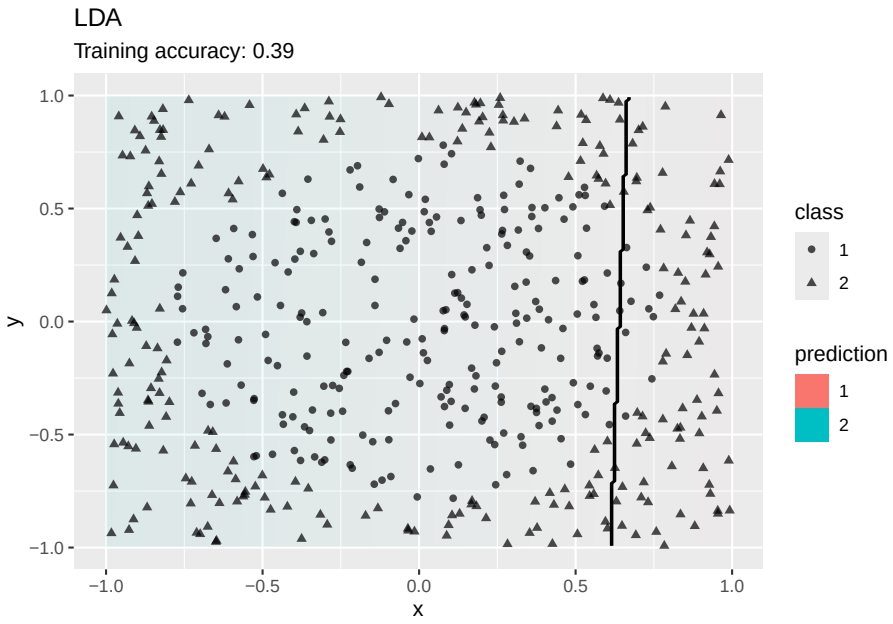
```
model <- x |> e1071::naiveBayes(class ~ ., data = _)
decisionplot(model, x, class_var = "class",
  predict_type = c("class", "raw")) +
  labs(title = "naive Bayes")
```



4.12.2.3 Linear Discriminant Analysis

LDA cannot find a good model since the true decision boundary is not linear.

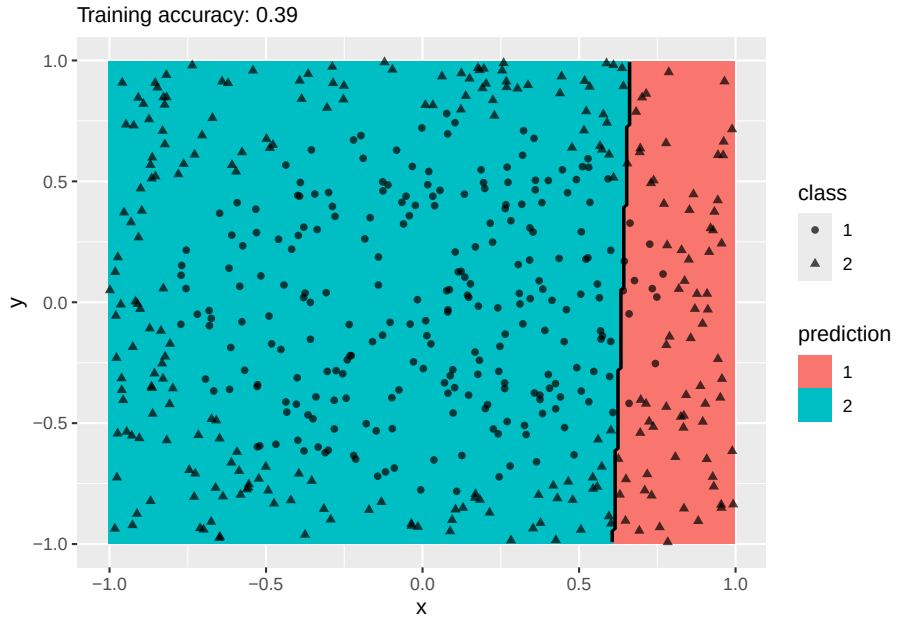
```
model <- x |> MASS::lda(class ~ ., data = _)
decisionplot(model, x, class_var = "class") + labs(title = "LDA")
```



4.12.2.4 Multinomial Logistic Regression

Multinomial logistic regression is an extension of logistic regression to problems with more than two classes.

```
model <- x |> nnet::multinom(class ~., data = _)
## # weights:  4 (3 variable)
## initial  value 346.573590
## final  value 346.308371
## converged
decisionplot(model, x, class_var = "class") +
  labs(titel = "Multinomial Logistic Regression")
```

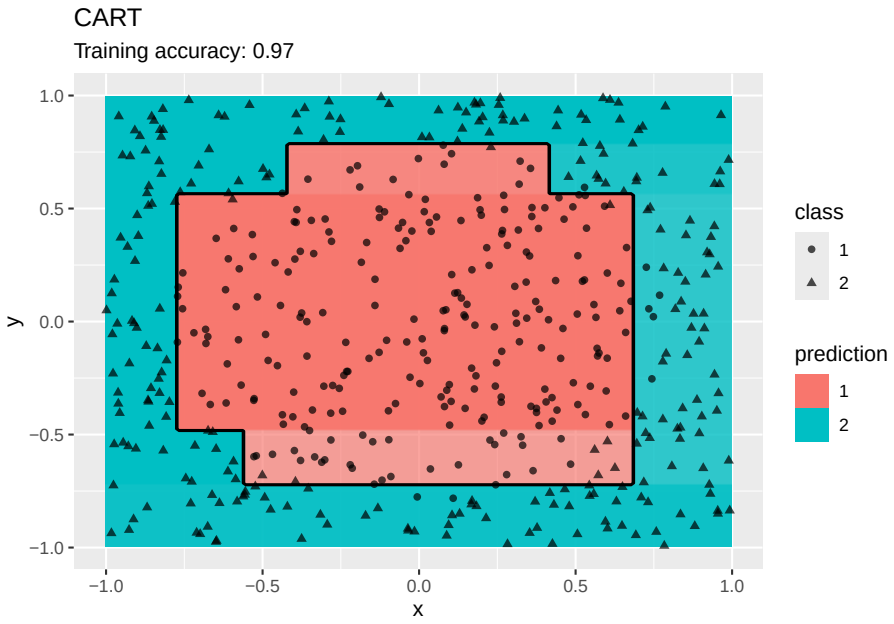


Logistic regression also tries to find a linear decision boundary and fails.

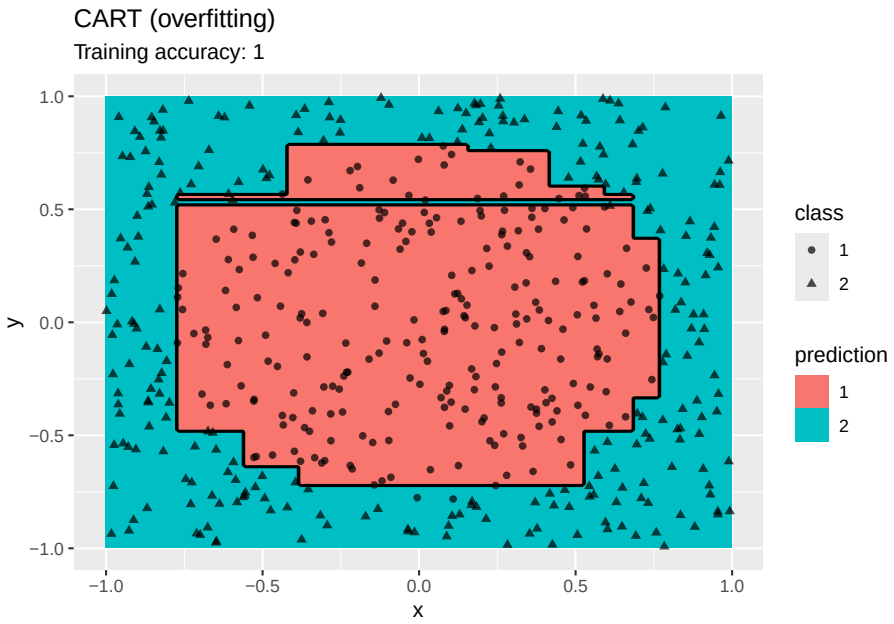
4.12.2.5 Decision Trees

Compare different decision tree algorithms.

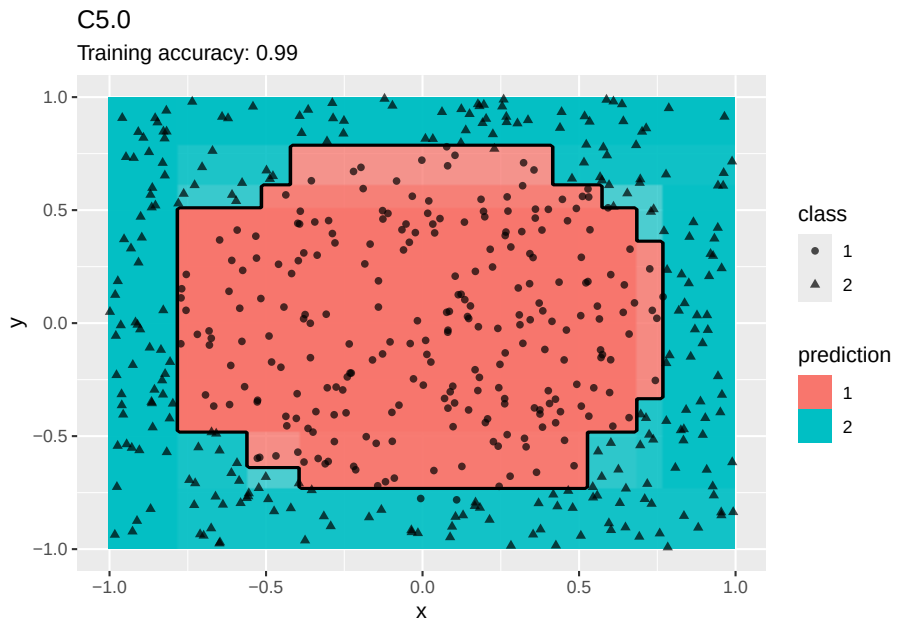
```
model <- x |> rpart::rpart(class ~ ., data = _)
decisionplot(model, x, class_var = "class") +
  labs(title = "CART")
```




```
model <- x |> rpart::rpart(class ~ ., data = _,
  control = rpart::rpart.control(cp = 0.001, minsplit = 1))
decisionplot(model, x, class_var = "class") +
  labs(title = "CART (overfitting)")
```



```
model <- x |> C50::C5.0(class ~ ., data = _)
decisionplot(model, x, class_var = "class") +
  labs(title = "C5.0")
```

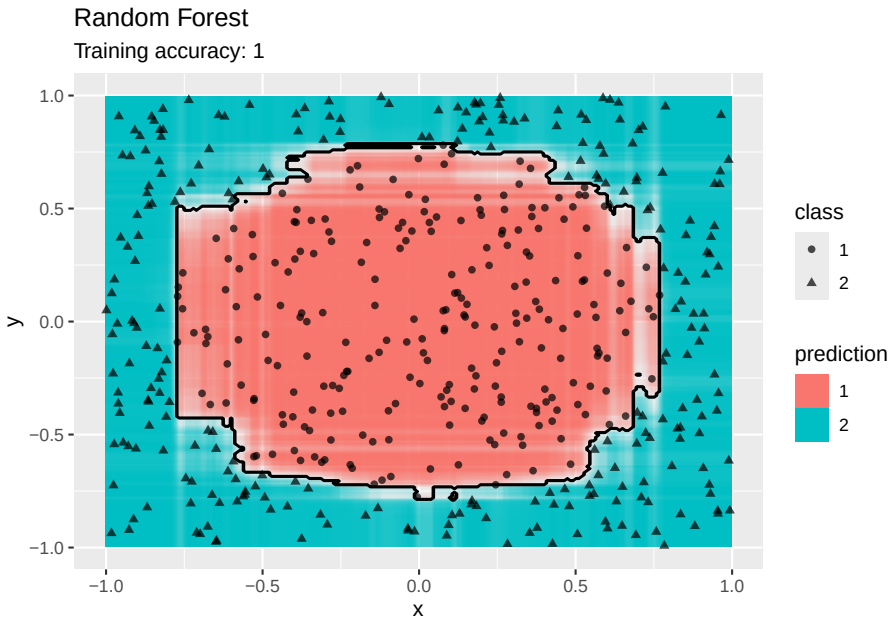


Decision trees do well with the restriction that they can only create cuts parallel to the axes.

4.12.2.6 Ensemble: Random Forest

Try random forest on the dataset.

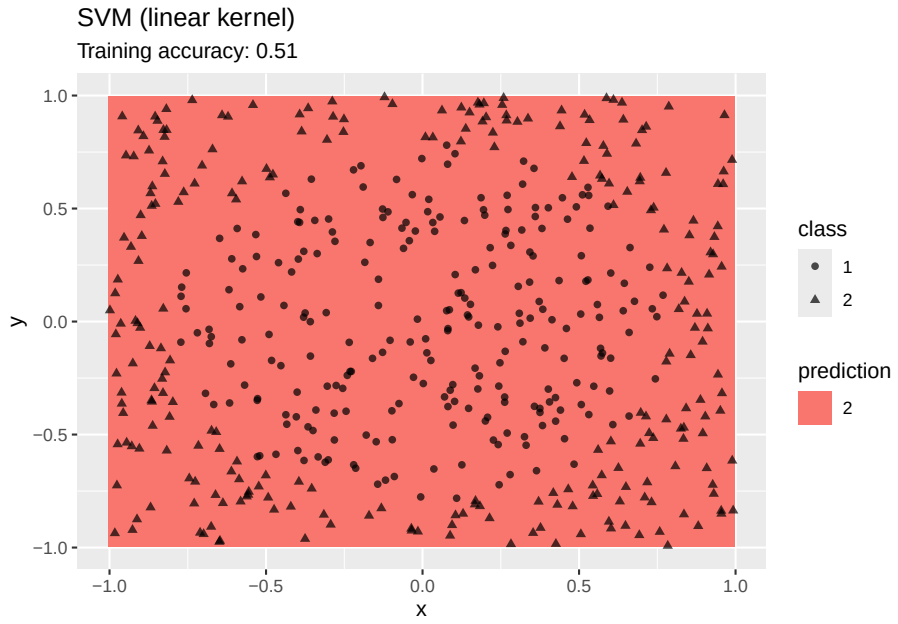
```
library(randomForest)
## randomForest 4.7-1.2
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:dplyr':
##
##     combine
## The following object is masked from 'package:ggplot2':
##
##     margin
model <- x |> randomForest(class ~ ., data = _)
decisionplot(model, x, class_var = "class") +
  labs(title = "Random Forest")
```



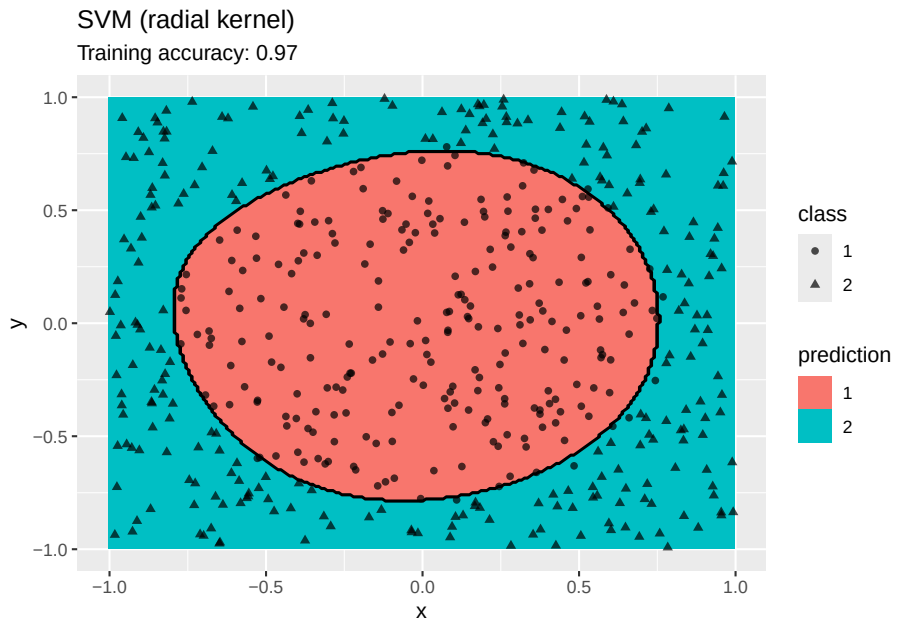
4.12.2.7 Support Vector Machine

Compare SVMs with different kernels.

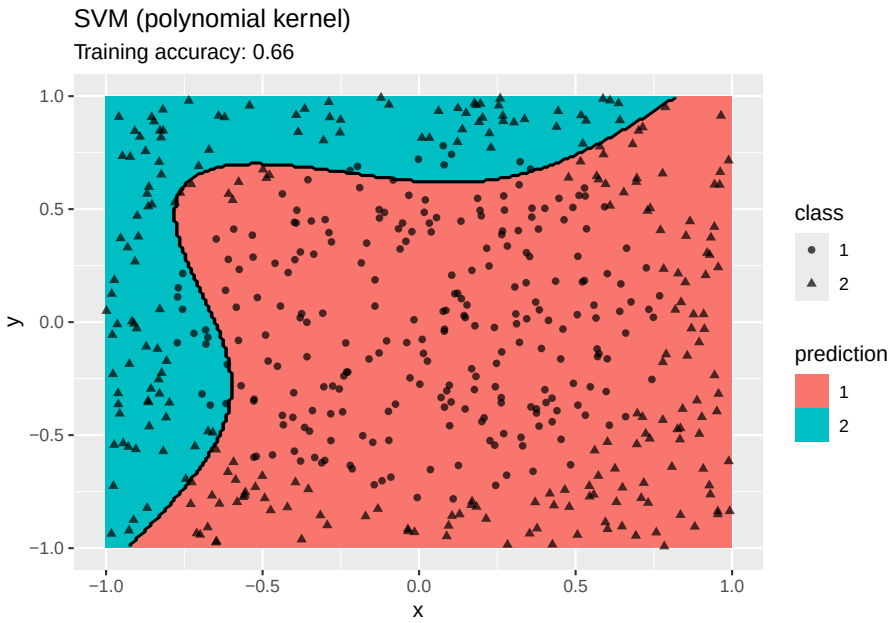
```
model <- x |> e1071::svm(class ~ ., data = _,
                         kernel = "linear")
decisionplot(model, x, class_var = "class") +
  labs(title = "SVM (linear kernel)")
## Warning: Computation failed in `stat_contour()`.
## Caused by error in `if (zero_range(range)) ...`:
## ! missing value where TRUE/FALSE needed
```



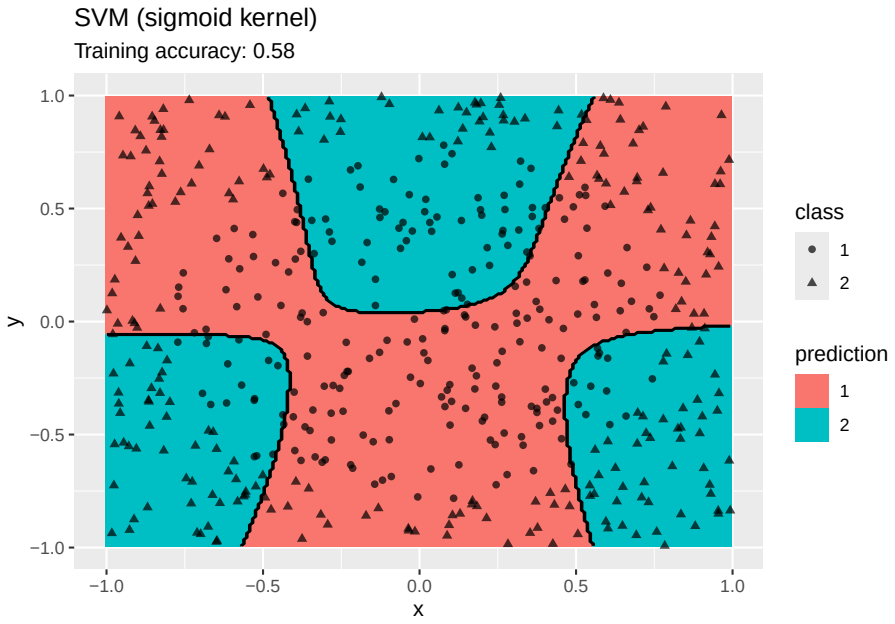

```
model <- x |> e1071::svm(class ~ ., data = _,
                         kernel = "radial")
decisionplot(model, x, class_var = "class") +
  labs(title = "SVM (radial kernel)")
```



```
model <- x |> e1071::svm(class ~ ., data = _,
                         kernel = "polynomial")
decisionplot(model, x, class_var = "class") +
  labs(title = "SVM (polynomial kernel)")
```




```
model <- x |> e1071::svm(class ~ ., data = _,
                         kernel = "sigmoid")
decisionplot(model, x, class_var = "class") +
  labs(title = "SVM (sigmoid kernel)")
```

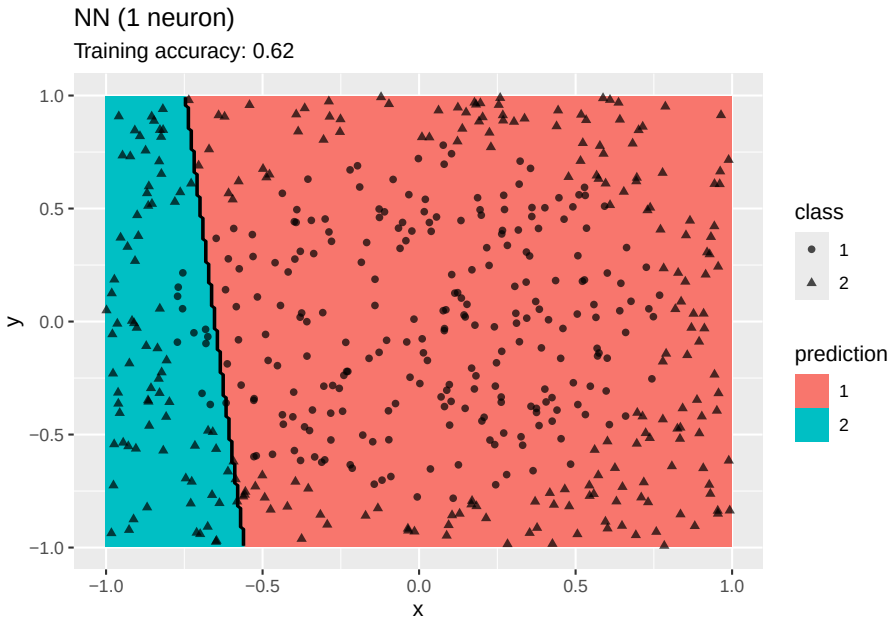


SMV with a radial kernel performs well, the other kernels have issues with the data.

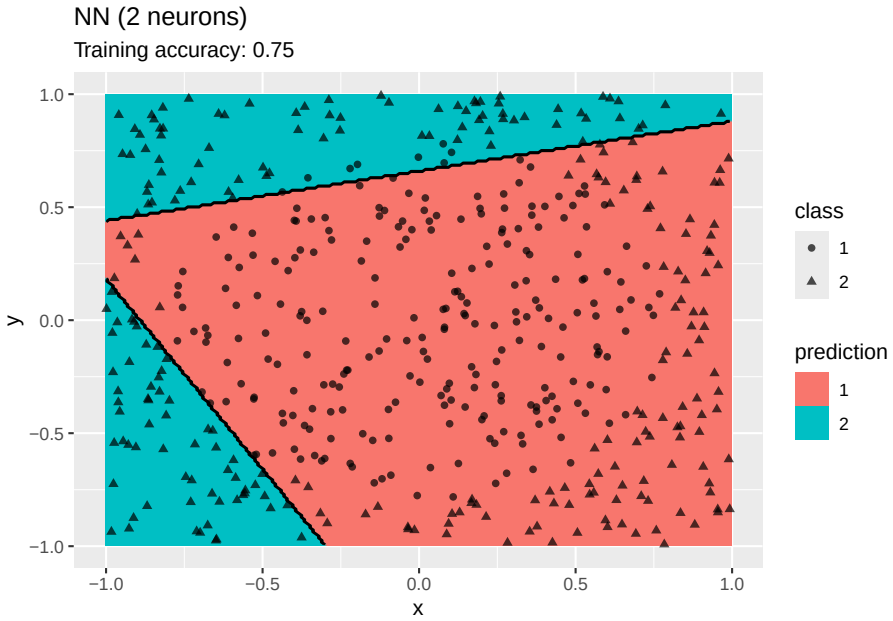
4.12.2.8 Single Layer Feed-forward Neural Networks

Use a simple network with one hidden layer. We will try a different number of neurons for the hidden layer.

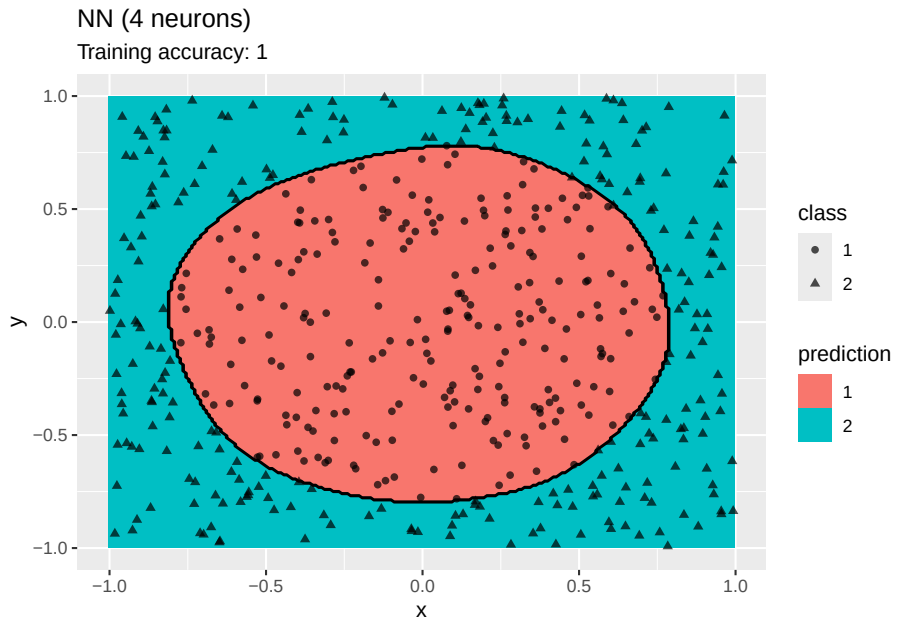
```
model <-x |> nnet::nnet(class ~ ., data = _, size = 1, trace = FALSE)
decisionplot(model, x, class_var = "class",
  predict_type = c("class")) + labs(title = "NN (1 neuron)")
```



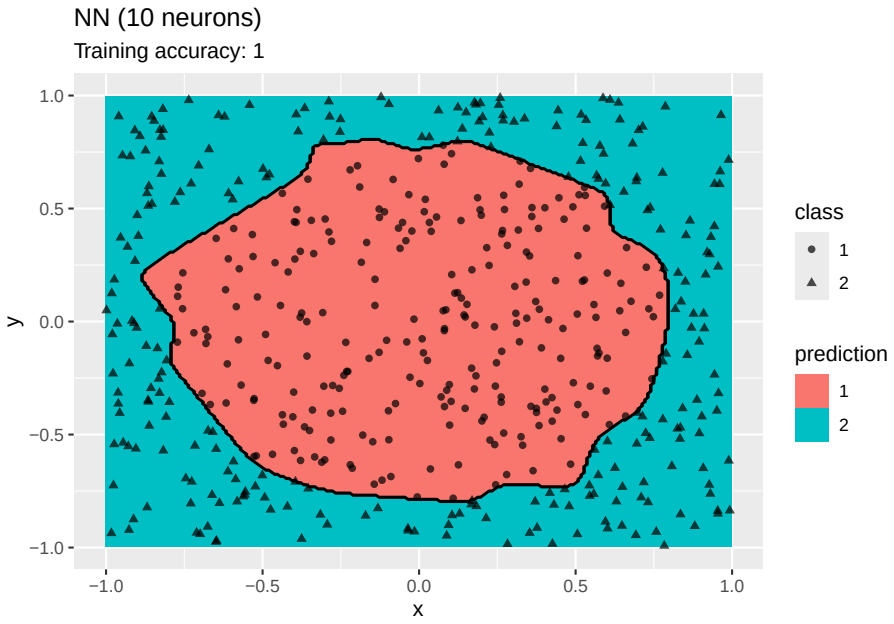

```
model <-x |> nnet::nnet(class ~ ., data = _, size = 2, trace = FALSE)
decisionplot(model, x, class_var = "class",
  predict_type = c("class")) + labs(title = "NN (2 neurons)")
```



```
model <-x |> nnet::nnet(class ~ ., data = _, size = 4, trace = FALSE)
decisionplot(model, x, class_var = "class",
  predict_type = c("class")) + labs(title = "NN (4 neurons)")
```




```
model <-x |> nnet::nnet(class ~ ., data = _, size = 10, trace = FALSE)
decisionplot(model, x, class_var = "class",
  predict_type = c("class")) + labs(title = "NN (10 neurons)")
```



The plots show that a network with 4 neurons performs well, while a larger number of neurons leads to overfitting the training data.

4.13 More Information on Classification with R

- Package caret: http://topepo.github.io/caret/index.html
- Tidymodels (machine learning with tidyverse): https://www.tidymodels.org/
- R taskview on machine learning: http://cran.r-project.org/web/views/MachineLearning.html

4.14 Exercises*

We will again use the Palmer penguin data for the exercises.

```
library(palmerpenguins)
head(penguins)
## # A tibble: 6 x 8
##   species island    bill_length_mm bill_depth_mm
##   <chr>   <chr>              <dbl>         <dbl>
## 1 Adelie  Torgersen           39.1          18.7
## 2 Adelie  Torgersen           39.5          17.4
## 3 Adelie  Torgersen           40.3          18
## 4 Adelie  Torgersen           NA            NA
```

```
## 5 Adelie  Torgersen           36.7          19.3
## 6 Adelie  Torgersen           39.3          20.6
## # i 4 more variables: flipper_length_mm <dbl>,
## #   body_mass_g <dbl>, sex <chr>, year <dbl>
```

Create a R markdown file with the code and do the following below.

1. Apply at least 3 different classification models to the data.
2. Compare the models and a simple baseline model. Which model performs the best? Does it perform significantly better than the other models?

Chapter 5

Association Analysis: Basic Concepts

This chapter introduces association rules mining using the APRIORI algorithm. In addition, analyzing sets of association rules using visualization techniques is demonstrated.

The corresponding chapter of the data mining textbook is available online: Chapter 5: Association Analysis: Basic Concepts and Algorithms.[1]

Packages Used in this Chapter

```
pkgs <- c("arules", "arulesViz", "mlbench",
          "palmerpenguins", "tidyverse")

pkgs_install <- pkgs[!(pkgs %in% installed.packages()[,"Package"])]
if(length(pkgs_install)) install.packages(pkgs_install)
```

The packages used for this chapter are:

- *arules* (Hahsler et al. 2024)
- *arulesViz* (Hahsler 2024)
- *mlbench* (Leisch and Dimitriadou 2024)
- *palmerpenguins* (Horst, Hill, and Gorman 2022)
- *tidyverse* (Wickham 2023c)

[1] https://www-users.cs.umn.edu/~kumar001/dmbook/ch5_association_analysis.pdf

5.1 Preliminaries

Association rule mining[2] plays a vital role in discovering hidden patterns and relationships within large transactional datasets. Applications range from exploratory data analysis in marketing to building rule-based classifiers. Agrawal, Imielinski, and Swami (1993) introduced the problem of mining association rules from transaction data as follows (the definition is taken from Hahsler, Grün, and Hornik (2005)):

Let $I = \{i_1, i_2, ..., i_n\}$ be a set of n binary attributes called items. Let $D = \{t_1, t_2, ..., t_m\}$ be a set of transactions called the database. Each transaction in D has a unique transaction ID and contains a subset of the items in I. A rule is defined as an implication of the form $X \Rightarrow Y$ where $X, Y \subseteq I$ and $X \cap Y = \emptyset$ are called itemsets. On itemsets and rules several quality measures can be defined. The most important measures are support and confidence. The support $supp(X)$ of an itemset X is defined as the proportion of transactions in the data set which contain the itemset. Itemsets with a support which surpasses a user-defined threshold σ are called frequent itemsets. The confidence of a rule is defined as $conf(X \Rightarrow Y) = supp(X \cup Y)/supp(X)$. Association rules are rules with $supp(X \cup Y) \geq \sigma$ and $conf(X) \geq \delta$ where σ and δ are user-defined thresholds. The found set of association rules is then used reason about the data.

You can read the free sample chapter from the textbook (Tan, Steinbach, and Kumar 2005): Chapter 5. Association Analysis: Basic Concepts and Algorithms[3]

5.1.1 The arules Package

Association rule mining in R is implemented in the package `arules`.

```
library(tidyverse)
library(arules)
library(arulesViz)
```

For information about the `arules` package try: `help(package="arules")` and `vignette("arules")` (also available at CRAN[4])

arules uses the S4 object system to implement classes and methods. Standard R objects use the S3 object system[5] which do not use formal class definitions and are usually implemented as a list with a class attribute. `arules` and many other R packages use the S4 object system[6] which is based on formal class definitions with member variables and methods (similar to object-oriented programming

[2] https://en.wikipedia.org/wiki/Association_rule_learning
[3] https://www-users.cs.umn.edu/~kumar001/dmbook/ch5_association_analysis.pdf
[4] http://cran.r-project.org/web/packages/arules/vignettes/arules.pdf
[5] http://adv-r.had.co.nz/S3.html
[6] http://adv-r.had.co.nz/S4.html

languages like Java and C++). Some important differences of using S4 objects compared to the usual S3 objects are:

- coercion (casting): `as(from, "class_name")`
- help for classes: `class? class_name`

5.1.2 Transactions

5.1.2.1 Create Transactions

We will use the Zoo dataset from `mlbench`.

```
data(Zoo, package = "mlbench")
head(Zoo)
##           hair feathers  eggs  milk airborne aquatic
## aardvark  TRUE    FALSE FALSE  TRUE    FALSE   FALSE
## antelope  TRUE    FALSE FALSE  TRUE    FALSE   FALSE
## bass     FALSE    FALSE  TRUE FALSE    FALSE    TRUE
## bear      TRUE    FALSE FALSE  TRUE    FALSE   FALSE
## boar      TRUE    FALSE FALSE  TRUE    FALSE   FALSE
## buffalo   TRUE    FALSE FALSE  TRUE    FALSE   FALSE
##          predator toothed backbone breathes venomous  fins
## aardvark     TRUE    TRUE     TRUE     TRUE    FALSE FALSE
## antelope    FALSE    TRUE     TRUE     TRUE    FALSE FALSE
## bass         TRUE    TRUE     TRUE    FALSE    FALSE  TRUE
## bear         TRUE    TRUE     TRUE     TRUE    FALSE FALSE
## boar         TRUE    TRUE     TRUE     TRUE    FALSE FALSE
## buffalo     FALSE    TRUE     TRUE     TRUE    FALSE FALSE
##          legs  tail domestic catsize   type
## aardvark    4 FALSE    FALSE    TRUE mammal
## antelope    4  TRUE    FALSE    TRUE mammal
## bass        0  TRUE    FALSE   FALSE   fish
## bear        4 FALSE    FALSE    TRUE mammal
## boar        4  TRUE    FALSE    TRUE mammal
## buffalo     4  TRUE    FALSE    TRUE mammal
```

The data in the data.frame need to be converted into a set of transactions where each row represents a transaction and each column is translated into items. This is done using the constructor `transactions()`. For the Zoo data set this means that we consider animals as transactions and the different traits (features) will become items that each animal has. For example the animal *antelope* has the item *hair* in its transaction.

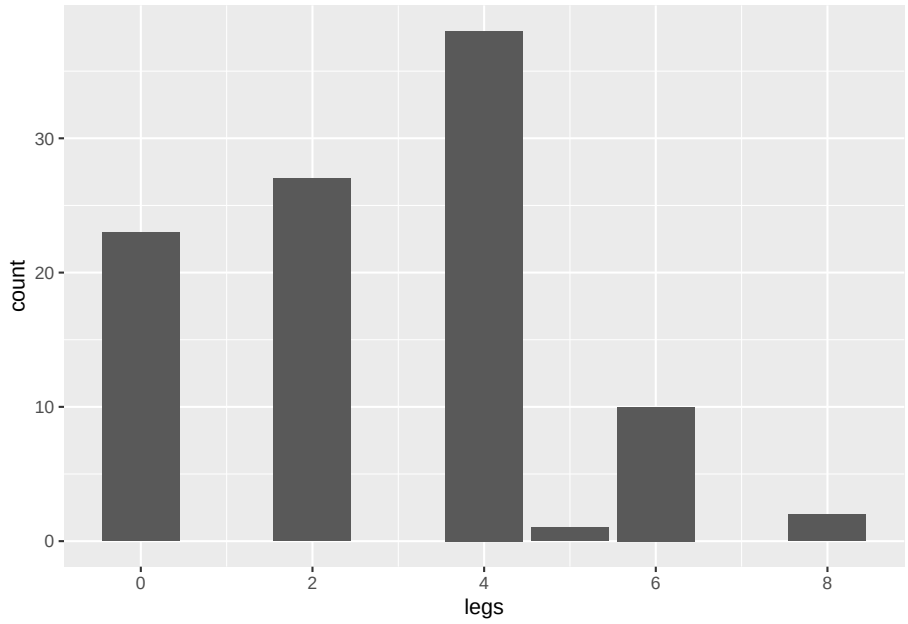
```
trans <- transactions(Zoo)
## Warning: Column(s) 13 not logical or factor. Applying
## default discretization (see '? discretizeDF').
```

The conversion gives a warning because only discrete features (`factor` and

`logical`) can be directly translated into items. Continuous features need to be discretized first.

What is column 13?

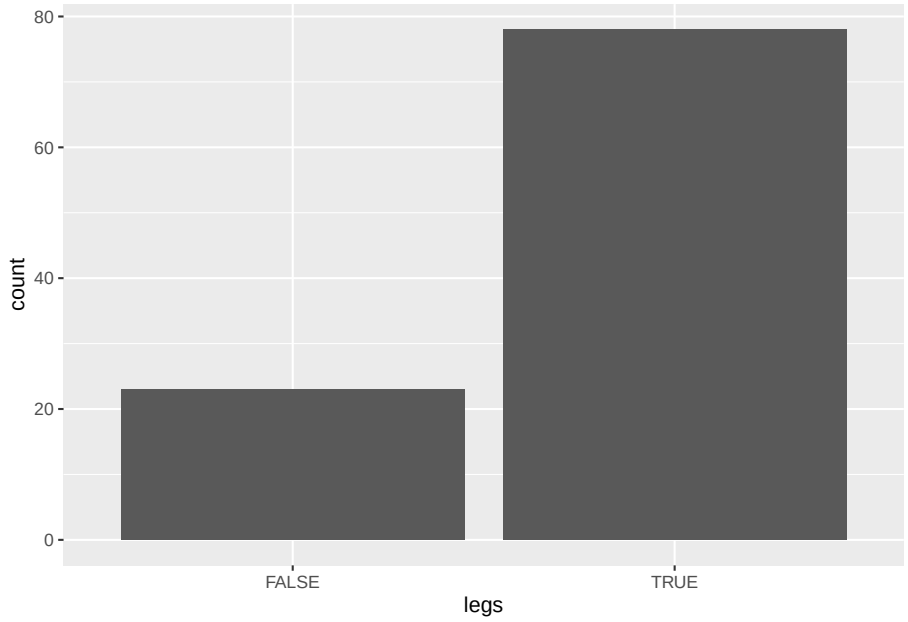
```
summary(Zoo[13])
##       legs
##  Min.   :0.00
##  1st Qu.:2.00
##  Median :4.00
##  Mean   :2.84
##  3rd Qu.:4.00
##  Max.   :8.00
ggplot(Zoo, aes(legs)) + geom_bar()
```




```
Zoo$legs |> table()
##
##  0  2  4  5  6  8
## 23 27 38  1 10  2
```

Possible solution: Make legs into has/does not have legs

```
Zoo_has_legs <- Zoo |> mutate(legs = legs > 0)
ggplot(Zoo_has_legs, aes(legs)) + geom_bar()
```




```
Zoo_has_legs$legs |> table()
##
## FALSE  TRUE
##    23    78
```

Alternatives:

Use each unique value as an item:

```
Zoo_unique_leg_values <- Zoo |> mutate(legs = factor(legs))
Zoo_unique_leg_values$legs |> head()
## [1] 4 4 0 4 4 4
## Levels: 0 2 4 5 6 8
```

Use the discretize function (see `? discretize`[7] and discretization in the code for Chapter 2[8]):

```
Zoo_discretized_legs <- Zoo |> mutate(
  legs = discretize(legs, breaks = 2, method="interval")
)
table(Zoo_discretized_legs$legs)
##
## [0,4) [4,8]
##    50    51
```

[7] https://www.rdocumentation.org/packages/arules/topics/discretize
[8] chap2.html#discretize-features

Convert data into a set of transactions

```
trans <- transactions(Zoo_has_legs)
trans
## transactions in sparse format with
##  101 transactions (rows) and
##  23 items (columns)
```

5.1.2.2 Inspect Transactions

```
summary(trans)
## transactions as itemMatrix in sparse format with
##  101 rows (elements/itemsets/transactions) and
##  23 columns (items) and a density of 0.3612
##
## most frequent items:
## backbone breathes     legs     tail  toothed  (Other)
##       83       80       78       75       61      462
##
## element (itemset/transaction) length distribution:
## sizes
##  3  4  5  6  7  8  9 10 11 12
##  3  2  6  5  8 21 27 25  3  1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    3.00    8.00    9.00    8.31   10.00   12.00
##
## includes extended item information - examples:
##     labels variables levels
## 1     hair      hair   TRUE
## 2 feathers  feathers   TRUE
## 3     eggs      eggs   TRUE
##
## includes extended transaction information - examples:
##   transactionID
## 1      aardvark
## 2      antelope
## 3          bass
```

Look at created items. They are still called column names since the transactions are actually stored as a large sparse logical matrix (see below).

```
colnames(trans)
##  [1] "hair"              "feathers"
##  [3] "eggs"              "milk"
##  [5] "airborne"          "aquatic"
##  [7] "predator"          "toothed"
```

```
##  [9] "backbone"           "breathes"
## [11] "venomous"           "fins"
## [13] "legs"               "tail"
## [15] "domestic"           "catsize"
## [17] "type=mammal"        "type=bird"
## [19] "type=reptile"       "type=fish"
## [21] "type=amphibian"     "type=insect"
## [23] "type=mollusc.et.al"
```

Compare with the original features (column names) from Zoo

```
colnames(Zoo)
##  [1] "hair"     "feathers" "eggs"     "milk"     "airborne"
##  [6] "aquatic"  "predator" "toothed"  "backbone" "breathes"
## [11] "venomous" "fins"     "legs"     "tail"     "domestic"
## [16] "catsize"  "type"
```

Look at a (first) few transactions as a matrix. 1 indicates the presence of an item.

```
as(trans, "matrix")[1:3,]
##           hair feathers  eggs  milk airborne aquatic
## aardvark  TRUE    FALSE FALSE  TRUE    FALSE   FALSE
## antelope  TRUE    FALSE FALSE  TRUE    FALSE   FALSE
## bass     FALSE    FALSE  TRUE FALSE    FALSE    TRUE
##          predator toothed backbone breathes venomous  fins
## aardvark     TRUE    TRUE     TRUE     TRUE    FALSE FALSE
## antelope    FALSE    TRUE     TRUE     TRUE    FALSE FALSE
## bass         TRUE    TRUE     TRUE    FALSE    FALSE  TRUE
##           legs  tail domestic catsize type=mammal type=bird
## aardvark  TRUE FALSE    FALSE    TRUE        TRUE     FALSE
## antelope  TRUE  TRUE    FALSE    TRUE        TRUE     FALSE
## bass     FALSE  TRUE    FALSE   FALSE       FALSE     FALSE
##          type=reptile type=fish type=amphibian type=insect
## aardvark        FALSE     FALSE          FALSE       FALSE
## antelope        FALSE     FALSE          FALSE       FALSE
## bass            FALSE      TRUE          FALSE       FALSE
##          type=mollusc.et.al
## aardvark              FALSE
## antelope              FALSE
## bass                  FALSE
```

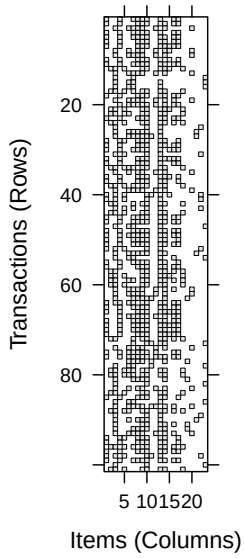
Look at the transactions as sets of items

```
inspect(trans[1:3])
##     items         transactionID
## [1] {hair,
```

```
##      milk,
##      predator,
##      toothed,
##      backbone,
##      breathes,
##      legs,
##      catsize,
##      type=mammal}      aardvark
## [2] {hair,
##      milk,
##      toothed,
##      backbone,
##      breathes,
##      legs,
##      tail,
##      catsize,
##      type=mammal}      antelope
## [3] {eggs,
##      aquatic,
##      predator,
##      toothed,
##      backbone,
##      fins,
##      tail,
##      type=fish}        bass
```

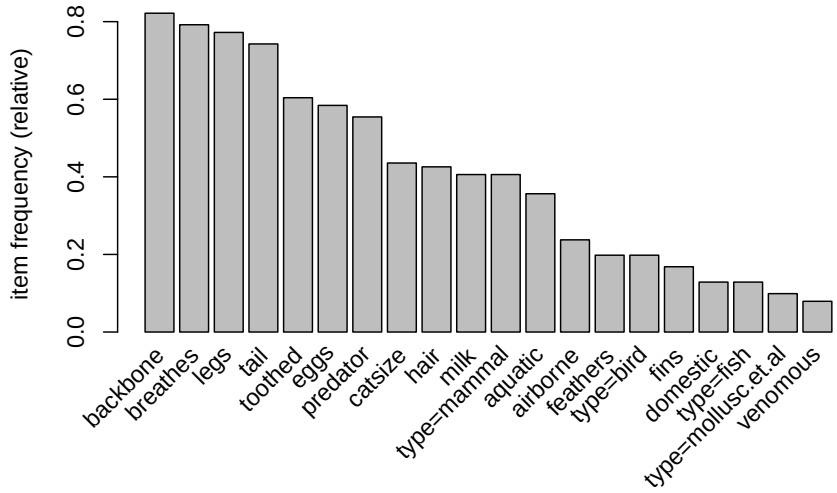
Plot the binary matrix. Dark dots represent 1s.

```
image(trans)
```

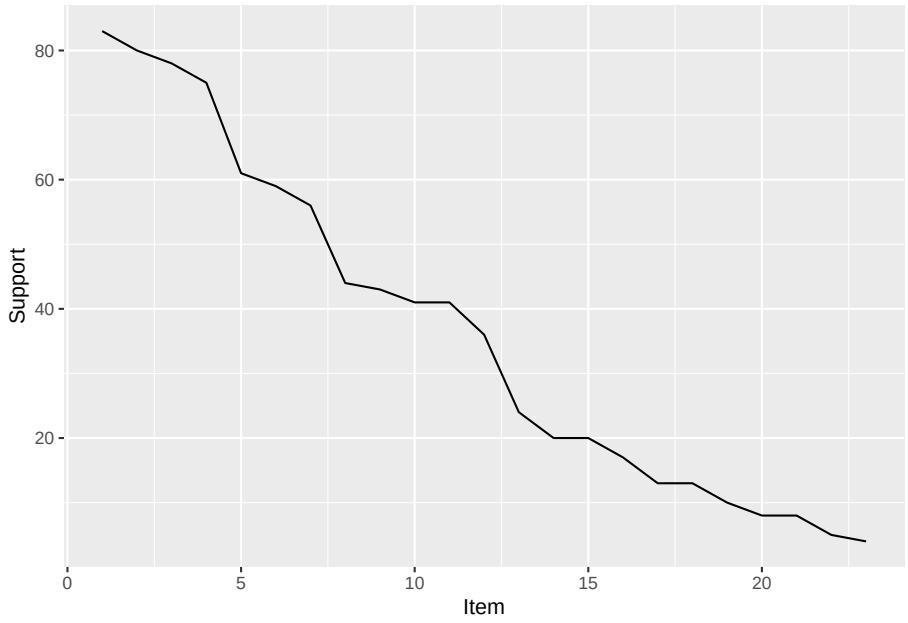


Look at the relative frequency (=support) of items in the data set. Here we look at the 10 most frequent items.

```
itemFrequencyPlot(trans,topN = 20)
```



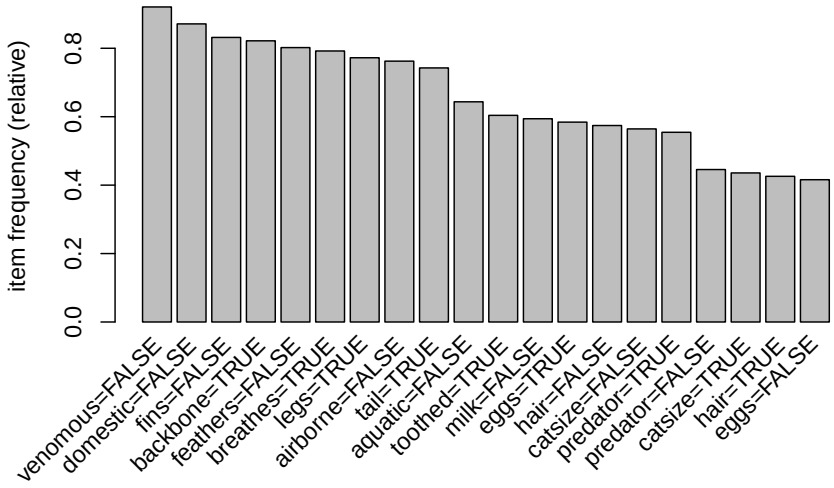
```
ggplot(
  tibble(
    Support = sort(itemFrequency(trans, type = "absolute"),
                   decreasing = TRUE),
    Item = seq_len(ncol(trans))
  ), aes(x = Item, y = Support)) +
  geom_line()
```



Alternative encoding: Also create items for FALSE (use factor)

```
sapply(Zoo_has_legs, class)
##      hair  feathers      eggs      milk  airborne   aquatic
## "logical" "logical" "logical" "logical" "logical" "logical"
##  predator   toothed  backbone  breathes  venomous      fins
## "logical" "logical" "logical" "logical" "logical" "logical"
##      legs      tail  domestic   catsize      type
## "logical" "logical" "logical" "logical"  "factor"
Zoo_factors <- Zoo_has_legs |>
  mutate(across(where(is.logical), factor))
sapply(Zoo_factors, class)
##     hair feathers     eggs     milk airborne  aquatic
## "factor" "factor" "factor" "factor" "factor" "factor"
## predator  toothed backbone breathes venomous     fins
## "factor" "factor" "factor" "factor" "factor" "factor"
##     legs     tail domestic  catsize     type
```

```
## "factor" "factor" "factor" "factor" "factor"
summary(Zoo_factors)
##     hair     feathers     eggs       milk     airborne
##  FALSE:58   FALSE:81   FALSE:42   FALSE:60   FALSE:77
##  TRUE :43   TRUE :20   TRUE :59   TRUE :41   TRUE :24
##
##
##
##
##
##   aquatic    predator   toothed    backbone   breathes
##  FALSE:65   FALSE:45   FALSE:40   FALSE:18   FALSE:21
##  TRUE :36   TRUE :56   TRUE :61   TRUE :83   TRUE :80
##
##
##
##
##
##   venomous     fins       legs       tail     domestic
##  FALSE:93   FALSE:84   FALSE:23   FALSE:26   FALSE:88
##  TRUE : 8   TRUE :17   TRUE :78   TRUE :75   TRUE :13
##
##
##
##
##
##   catsize              type
##  FALSE:57   mammal       :41
##  TRUE :44   bird         :20
##             reptile      : 5
##             fish         :13
##             amphibian    : 4
##             insect       : 8
##             mollusc.et.al:10
trans_factors <- transactions(Zoo_factors)
trans_factors
## transactions in sparse format with
##  101 transactions (rows) and
##  39 items (columns)
itemFrequencyPlot(trans_factors, topN = 20)
```




```
## Select transactions that contain a certain item
trans_insects <- trans_factors[trans %in% "type=insect"]
trans_insects
## transactions in sparse format with
##  8 transactions (rows) and
##  39 items (columns)
inspect(trans_insects)
##     items              transactionID
## [1] {hair=FALSE,
##      feathers=FALSE,
##      eggs=TRUE,
##      milk=FALSE,
##      airborne=FALSE,
##      aquatic=FALSE,
##      predator=FALSE,
##      toothed=FALSE,
##      backbone=FALSE,
##      breathes=TRUE,
##      venomous=FALSE,
##      fins=FALSE,
##      legs=TRUE,
##      tail=FALSE,
##      domestic=FALSE,
##      catsize=FALSE,
##      type=insect}          flea
```

```
## [2] {hair=FALSE,
##      feathers=FALSE,
##      eggs=TRUE,
##      milk=FALSE,
##      airborne=TRUE,
##      aquatic=FALSE,
##      predator=FALSE,
##      toothed=FALSE,
##      backbone=FALSE,
##      breathes=TRUE,
##      venomous=FALSE,
##      fins=FALSE,
##      legs=TRUE,
##      tail=FALSE,
##      domestic=FALSE,
##      catsize=FALSE,
##      type=insect}          gnat
## [3] {hair=TRUE,
##      feathers=FALSE,
##      eggs=TRUE,
##      milk=FALSE,
##      airborne=TRUE,
##      aquatic=FALSE,
##      predator=FALSE,
##      toothed=FALSE,
##      backbone=FALSE,
##      breathes=TRUE,
##      venomous=TRUE,
##      fins=FALSE,
##      legs=TRUE,
##      tail=FALSE,
##      domestic=TRUE,
##      catsize=FALSE,
##      type=insect}          honeybee
## [4] {hair=TRUE,
##      feathers=FALSE,
##      eggs=TRUE,
##      milk=FALSE,
##      airborne=TRUE,
##      aquatic=FALSE,
##      predator=FALSE,
##      toothed=FALSE,
##      backbone=FALSE,
##      breathes=TRUE,
##      venomous=FALSE,
```

```
##      fins=FALSE,
##      legs=TRUE,
##      tail=FALSE,
##      domestic=FALSE,
##      catsize=FALSE,
##      type=insect}          housefly
## [5] {hair=FALSE,
##      feathers=FALSE,
##      eggs=TRUE,
##      milk=FALSE,
##      airborne=TRUE,
##      aquatic=FALSE,
##      predator=TRUE,
##      toothed=FALSE,
##      backbone=FALSE,
##      breathes=TRUE,
##      venomous=FALSE,
##      fins=FALSE,
##      legs=TRUE,
##      tail=FALSE,
##      domestic=FALSE,
##      catsize=FALSE,
##      type=insect}          ladybird
## [6] {hair=TRUE,
##      feathers=FALSE,
##      eggs=TRUE,
##      milk=FALSE,
##      airborne=TRUE,
##      aquatic=FALSE,
##      predator=FALSE,
##      toothed=FALSE,
##      backbone=FALSE,
##      breathes=TRUE,
##      venomous=FALSE,
##      fins=FALSE,
##      legs=TRUE,
##      tail=FALSE,
##      domestic=FALSE,
##      catsize=FALSE,
##      type=insect}          moth
## [7] {hair=FALSE,
##      feathers=FALSE,
##      eggs=TRUE,
##      milk=FALSE,
##      airborne=FALSE,
```

```
##      aquatic=FALSE,
##      predator=FALSE,
##      toothed=FALSE,
##      backbone=FALSE,
##      breathes=TRUE,
##      venomous=FALSE,
##      fins=FALSE,
##      legs=TRUE,
##      tail=FALSE,
##      domestic=FALSE,
##      catsize=FALSE,
##      type=insect}          termite
## [8] {hair=TRUE,
##      feathers=FALSE,
##      eggs=TRUE,
##      milk=FALSE,
##      airborne=TRUE,
##      aquatic=FALSE,
##      predator=FALSE,
##      toothed=FALSE,
##      backbone=FALSE,
##      breathes=TRUE,
##      venomous=TRUE,
##      fins=FALSE,
##      legs=TRUE,
##      tail=FALSE,
##      domestic=FALSE,
##      catsize=FALSE,
##      type=insect}          wasp
```

5.1.2.3 Vertical Layout (Transaction ID Lists)

The default layout for transactions is horizontal layout (i.e. each transaction is a row). The vertical layout represents transaction data as a list of transaction IDs for each item (= transaction ID lists).

```
vertical <- as(trans, "tidLists")
as(vertical, "matrix")[1:10, 1:5]
##          aardvark antelope  bass  bear  boar
## hair         TRUE     TRUE FALSE  TRUE  TRUE
## feathers    FALSE    FALSE FALSE FALSE FALSE
## eggs        FALSE    FALSE  TRUE FALSE FALSE
## milk         TRUE     TRUE FALSE  TRUE  TRUE
## airborne    FALSE    FALSE FALSE FALSE FALSE
## aquatic     FALSE    FALSE  TRUE FALSE FALSE
## predator     TRUE    FALSE  TRUE  TRUE  TRUE
```

```
## toothed      TRUE     TRUE  TRUE  TRUE  TRUE
## backbone     TRUE     TRUE  TRUE  TRUE  TRUE
## breathes     TRUE     TRUE FALSE  TRUE  TRUE
```

5.2 Frequent Itemset Generation

For this dataset we have already a huge number of possible itemsets

```
2^ncol(trans)
## [1] 8388608
```

Find frequent itemsets (target="frequent") with the default settings.

```
its <- apriori(trans, parameter=list(target = "frequent"))
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime
##          NA    0.1    1 none FALSE            TRUE       5
##  support minlen maxlen            target  ext
##      0.1      1     10 frequent itemsets TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 10
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[23 item(s), 101 transaction(s)] done [0.00s].
## sorting and recoding items ... [18 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 7 8 9 10
## Warning in apriori(trans, parameter = list(target =
## "frequent")): Mining stopped (maxlen reached). Only
## patterns up to a length of 10 returned!
##  done [0.00s].
## sorting transactions ... done [0.00s].
## writing ... [1465 set(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
its
## set of 1465 itemsets
```

Default minimum support is .1 (10%). **Note:** We use here a very small data set. For larger datasets the default minimum support might be to low and you may run out of memory. You probably want to start out with a higher minimum

support like .5 (50%) and then work your way down.

```
5/nrow(trans)
## [1] 0.0495
```

In order to find itemsets that effect 5 animals I need to go down to a support of about 5%.

```
its <- apriori(trans, parameter=list(target = "frequent",
                                     support = 0.05))
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime
##          NA    0.1    1 none FALSE            TRUE       5
##  support minlen maxlen            target  ext
##     0.05      1     10 frequent itemsets TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 5
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[23 item(s), 101 transaction(s)] done [0.00s].
## sorting and recoding items ... [21 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 7 8 9 10
## Warning in apriori(trans, parameter = list(target =
## "frequent", support = 0.05)): Mining stopped (maxlen
## reached). Only patterns up to a length of 10 returned!
##  done [0.00s].
## sorting transactions ... done [0.00s].
## writing ... [2537 set(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
its
## set of 2537 itemsets
```

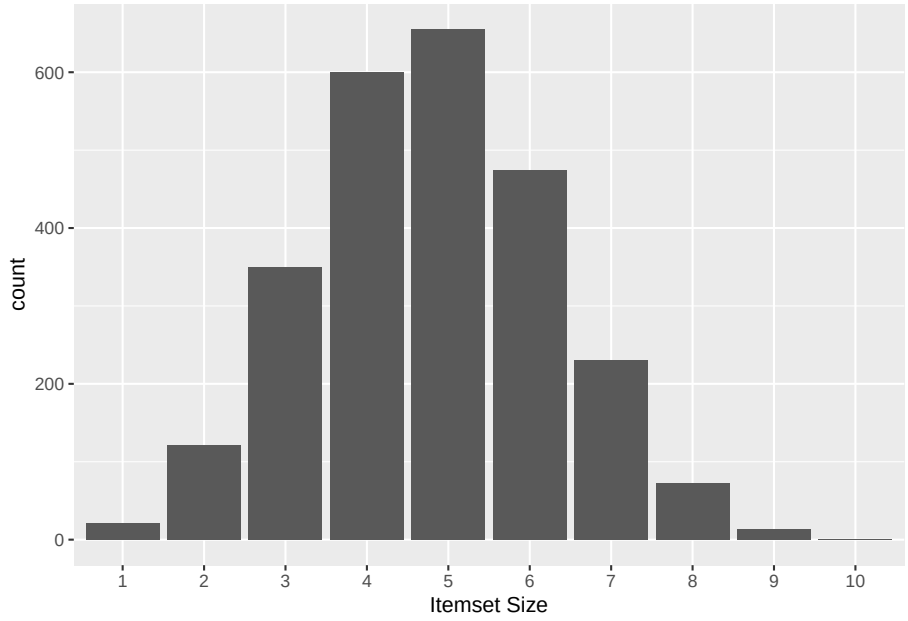
Sort by support

```
its <- sort(its, by = "support")
its |> head(n = 10) |> inspect()
##      items                      support count
## [1]  {backbone}                 0.8218  83
## [2]  {breathes}                 0.7921  80
## [3]  {legs}                     0.7723  78
## [4]  {tail}                     0.7426  75
```

```
## [5]  {backbone, tail}           0.7327  74
## [6]  {breathes, legs}           0.7228  73
## [7]  {backbone, breathes}       0.6832  69
## [8]  {backbone, legs}           0.6337  64
## [9]  {backbone, breathes, legs} 0.6337  64
## [10] {toothed}                  0.6040  61
```

Look at frequent itemsets with many items (set breaks manually since Automatically chosen breaks look bad)

```
ggplot(tibble(`Itemset Size` = factor(size(its))),
       aes(`Itemset Size`)) +
  geom_bar()
```




```
its[size(its) > 8] |> inspect()
##      items          support count
## [1]  {hair,
##       milk,
##       toothed,
##       backbone,
##       breathes,
##       legs,
##       tail,
##       catsize,
##       type=mammal} 0.23762    24
## [2]  {hair,
```

```
##       milk,
##       predator,
##       toothed,
##       backbone,
##       breathes,
##       legs,
##       catsize,
##       type=mammal} 0.15842    16
## [3]  {hair,
##       milk,
##       predator,
##       toothed,
##       backbone,
##       breathes,
##       legs,
##       tail,
##       type=mammal} 0.14851    15
## [4]  {hair,
##       milk,
##       predator,
##       backbone,
##       breathes,
##       legs,
##       tail,
##       catsize,
##       type=mammal} 0.13861    14
## [5]  {hair,
##       milk,
##       predator,
##       toothed,
##       breathes,
##       legs,
##       tail,
##       catsize,
##       type=mammal} 0.12871    13
## [6]  {hair,
##       milk,
##       predator,
##       toothed,
##       backbone,
##       legs,
##       tail,
##       catsize,
##       type=mammal} 0.12871    13
## [7]  {hair,
```

```
##       milk,
##       predator,
##       toothed,
##       backbone,
##       breathes,
##       tail,
##       catsize,
##       type=mammal} 0.12871    13
## [8]  {milk,
##       predator,
##       toothed,
##       backbone,
##       breathes,
##       legs,
##       tail,
##       catsize,
##       type=mammal} 0.12871    13
## [9]  {hair,
##       milk,
##       predator,
##       toothed,
##       backbone,
##       breathes,
##       legs,
##       tail,
##       catsize}     0.12871    13
## [10] {hair,
##       predator,
##       toothed,
##       backbone,
##       breathes,
##       legs,
##       tail,
##       catsize,
##       type=mammal} 0.12871    13
## [11] {hair,
##       milk,
##       predator,
##       toothed,
##       backbone,
##       breathes,
##       legs,
##       tail,
##       catsize,
##       type=mammal} 0.12871    13
```

```
## [12] {hair,
##       milk,
##       toothed,
##       backbone,
##       breathes,
##       legs,
##       domestic,
##       catsize,
##       type=mammal} 0.05941     6
## [13] {hair,
##       milk,
##       toothed,
##       backbone,
##       breathes,
##       legs,
##       tail,
##       domestic,
##       type=mammal} 0.05941     6
## [14] {feathers,
##       eggs,
##       airborne,
##       predator,
##       backbone,
##       breathes,
##       legs,
##       tail,
##       type=bird}   0.05941     6
```

5.3 Rule Generation

We use the APRIORI algorithm (see `? apriori`[9])

```
rules <- apriori(trans,
                 parameter = list(support = 0.05,
                                  confidence = 0.9))
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime
##         0.9    0.1    1 none FALSE            TRUE       5
##  support minlen maxlen target  ext
##     0.05      1     10  rules TRUE
##
```

[9] https://www.rdocumentation.org/packages/arules/topics/apriori

```
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 5
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[23 item(s), 101 transaction(s)] done [0.00s].
## sorting and recoding items ... [21 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 7 8 9 10
## Warning in apriori(trans, parameter = list(support = 0.05,
## confidence = 0.9)): Mining stopped (maxlen reached). Only
## patterns up to a length of 10 returned!
##  done [0.00s].
## writing ... [7174 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
length(rules)
## [1] 7174
rules |> head() |> inspect()
##     lhs                     rhs        support confidence
## [1] {type=insect}        => {eggs}     0.07921 1.0
## [2] {type=insect}        => {legs}     0.07921 1.0
## [3] {type=insect}        => {breathes} 0.07921 1.0
## [4] {type=mollusc.et.al} => {eggs}     0.08911 0.9
## [5] {type=fish}          => {fins}     0.12871 1.0
## [6] {type=fish}          => {aquatic}  0.12871 1.0
##     coverage lift  count
## [1] 0.07921  1.712  8
## [2] 0.07921  1.295  8
## [3] 0.07921  1.262  8
## [4] 0.09901  1.541  9
## [5] 0.12871  5.941 13
## [6] 0.12871  2.806 13
rules |> head() |> quality()
##   support confidence coverage  lift count
## 1 0.07921        1.0  0.07921 1.712     8
## 2 0.07921        1.0  0.07921 1.295     8
## 3 0.07921        1.0  0.07921 1.262     8
## 4 0.08911        0.9  0.09901 1.541     9
## 5 0.12871        1.0  0.12871 5.941    13
## 6 0.12871        1.0  0.12871 2.806    13
```

Look at rules with highest lift

```
rules <- sort(rules, by = "lift")
rules |> head(n = 10) |> inspect()
##      lhs             rhs         support confidence coverage  lift count
## [1]  {eggs,
##       fins}      => {type=fish} 0.12871          1  0.12871 7.769    13
## [2]  {eggs,
##       aquatic,
##       fins}      => {type=fish} 0.12871          1  0.12871 7.769    13
## [3]  {eggs,
##       predator,
##       fins}      => {type=fish} 0.08911          1  0.08911 7.769     9
## [4]  {eggs,
##       toothed,
##       fins}      => {type=fish} 0.12871          1  0.12871 7.769    13
## [5]  {eggs,
##       fins,
##       tail}      => {type=fish} 0.12871          1  0.12871 7.769    13
## [6]  {eggs,
##       backbone,
##       fins}      => {type=fish} 0.12871          1  0.12871 7.769    13
## [7]  {eggs,
##       aquatic,
##       predator,
##       fins}      => {type=fish} 0.08911          1  0.08911 7.769     9
## [8]  {eggs,
##       aquatic,
##       toothed,
##       fins}      => {type=fish} 0.12871          1  0.12871 7.769    13
## [9]  {eggs,
##       aquatic,
##       fins,
##       tail}      => {type=fish} 0.12871          1  0.12871 7.769    13
## [10] {eggs,
##       aquatic,
##       backbone,
##       fins}      => {type=fish} 0.12871          1  0.12871 7.769    13
```

Create rules using the alternative encoding (with "FALSE" item)

```
r <- apriori(trans_factors)
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime
##         0.8    0.1    1 none FALSE            TRUE       5
##  support minlen maxlen target  ext
```

```
##       0.1      1     10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 10
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[39 item(s), 101 transaction(s)] done [0.00s].
## sorting and recoding items ... [34 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 7 8 9 10
## Warning in apriori(trans_factors): Mining stopped (maxlen
## reached). Only patterns up to a length of 10 returned!
##  done [0.08s].
## writing ... [1517191 rule(s)] done [0.18s].
## creating S4 object  ... done [0.89s].
r
## set of 1517191 rules
print(object.size(r), unit = "Mb")
## 110.2 Mb
inspect(r[1:10])
##      lhs                 rhs                 support confidence
## [1]  {}                  => {feathers=FALSE} 0.8020  0.8020
## [2]  {}                  => {backbone=TRUE}  0.8218  0.8218
## [3]  {}                  => {fins=FALSE}     0.8317  0.8317
## [4]  {}                  => {domestic=FALSE} 0.8713  0.8713
## [5]  {}                  => {venomous=FALSE} 0.9208  0.9208
## [6]  {domestic=TRUE} => {predator=FALSE} 0.1089  0.8462
## [7]  {domestic=TRUE} => {aquatic=FALSE}  0.1188  0.9231
## [8]  {domestic=TRUE} => {legs=TRUE}      0.1188  0.9231
## [9]  {domestic=TRUE} => {breathes=TRUE}  0.1188  0.9231
## [10] {domestic=TRUE} => {backbone=TRUE}  0.1188  0.9231
##      coverage lift  count
## [1]  1.0000   1.000 81
## [2]  1.0000   1.000 83
## [3]  1.0000   1.000 84
## [4]  1.0000   1.000 88
## [5]  1.0000   1.000 93
## [6]  0.1287   1.899 11
## [7]  0.1287   1.434 12
## [8]  0.1287   1.195 12
## [9]  0.1287   1.165 12
## [10] 0.1287   1.123 12
```

```
r |> head(n = 10, by = "lift") |> inspect()
##      lhs                    rhs          support confidence coverage  lift count
## [1]  {breathes=FALSE,
##       fins=TRUE}         => {type=fish}  0.1287          1   0.1287 7.769    13
## [2]  {eggs=TRUE,
##       fins=TRUE}         => {type=fish}  0.1287          1   0.1287 7.769    13
## [3]  {milk=FALSE,
##       fins=TRUE}         => {type=fish}  0.1287          1   0.1287 7.769    13
## [4]  {breathes=FALSE,
##       fins=TRUE,
##       legs=FALSE}        => {type=fish}  0.1287          1   0.1287 7.769    13
## [5]  {aquatic=TRUE,
##       breathes=FALSE,
##       fins=TRUE}         => {type=fish}  0.1287          1   0.1287 7.769    13
## [6]  {hair=FALSE,
##       breathes=FALSE,
##       fins=TRUE}         => {type=fish}  0.1287          1   0.1287 7.769    13
## [7]  {eggs=TRUE,
##       breathes=FALSE,
##       fins=TRUE}         => {type=fish}  0.1287          1   0.1287 7.769    13
## [8]  {milk=FALSE,
##       breathes=FALSE,
##       fins=TRUE}         => {type=fish}  0.1287          1   0.1287 7.769    13
## [9]  {toothed=TRUE,
##       breathes=FALSE,
##       fins=TRUE}         => {type=fish}  0.1287          1   0.1287 7.769    13
## [10] {breathes=FALSE,
##       fins=TRUE,
##       tail=TRUE}         => {type=fish}  0.1287          1   0.1287 7.769    13
```

5.3.1 Calculate Additional Interest Measures

```
interestMeasure(rules[1:10], measure = c("phi", "gini"),
  trans = trans)
##       phi   gini
## 1  1.0000 0.2243
## 2  1.0000 0.2243
## 3  0.8138 0.1485
## 4  1.0000 0.2243
## 5  1.0000 0.2243
## 6  1.0000 0.2243
## 7  0.8138 0.1485
## 8  1.0000 0.2243
## 9  1.0000 0.2243
```

```
## 10 1.0000 0.2243
```

Add measures to the rules

```
quality(rules) <- cbind(quality(rules),
  interestMeasure(rules, measure = c("phi", "gini"),
    trans = trans))
```

Find rules which score high for Phi correlation

```
rules |> head(by = "phi") |> inspect()
##     lhs             rhs          support confidence coverage  lift count phi   gini
## [1] {eggs,
##      fins}      => {type=fish}  0.1287          1   0.1287 7.769    13   1 0.2243
## [2] {eggs,
##      aquatic,
##      fins}      => {type=fish}  0.1287          1   0.1287 7.769    13   1 0.2243
## [3] {eggs,
##      toothed,
##      fins}      => {type=fish}  0.1287          1   0.1287 7.769    13   1 0.2243
## [4] {eggs,
##      fins,
##      tail}      => {type=fish}  0.1287          1   0.1287 7.769    13   1 0.2243
## [5] {eggs,
##      backbone,
##      fins}      => {type=fish}  0.1287          1   0.1287 7.769    13   1 0.2243
## [6] {eggs,
##      aquatic,
##      toothed,
##      fins}      => {type=fish}  0.1287          1   0.1287 7.769    13   1 0.2243
```

5.3.2 Mine Using Templates

Sometimes it is beneficial to specify what items should be where in the rule. For apriori we can use the parameter appearance to specify this (see `? APappearance`[10]). In the following we restrict rules to an animal `type` in the RHS and any item in the LHS.

```
type <- grep("type=", itemLabels(trans), value = TRUE)
type
## [1] "type=mammal"        "type=bird"
## [3] "type=reptile"       "type=fish"
## [5] "type=amphibian"     "type=insect"
## [7] "type=mollusc.et.al"
rules_type <- apriori(trans, appearance= list(rhs = type))
## Apriori
```

[10] https://www.rdocumentation.org/packages/arules/topics/APappearance

```
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime
##         0.8    0.1    1 none FALSE            TRUE       5
##  support minlen maxlen target  ext
##      0.1      1     10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 10
##
## set item appearances ...[7 item(s)] done [0.00s].
## set transactions ...[23 item(s), 101 transaction(s)] done [0.00s].
## sorting and recoding items ... [18 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 7 8 9 10
## Warning in apriori(trans, appearance = list(rhs = type)):
## Mining stopped (maxlen reached). Only patterns up to a
## length of 10 returned!
##  done [0.00s].
## writing ... [571 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
rules_type |> sort(by = "lift") |> head() |> inspect()
##     lhs             rhs         support confidence coverage  lift count
## [1] {eggs,
##      fins}      => {type=fish}  0.1287          1   0.1287 7.769    13
## [2] {eggs,
##      aquatic,
##      fins}      => {type=fish}  0.1287          1   0.1287 7.769    13
## [3] {eggs,
##      toothed,
##      fins}      => {type=fish}  0.1287          1   0.1287 7.769    13
## [4] {eggs,
##      fins,
##      tail}      => {type=fish}  0.1287          1   0.1287 7.769    13
## [5] {eggs,
##      backbone,
##      fins}      => {type=fish}  0.1287          1   0.1287 7.769    13
## [6] {eggs,
##      aquatic,
##      toothed,
##      fins}      => {type=fish}  0.1287          1   0.1287 7.769    13
```

Saving rules as a CSV-file to be opened with Excel or other tools.

```
write(rules, file = "rules.csv", quote = TRUE)
```

5.4 Compact Representation of Frequent Itemsets

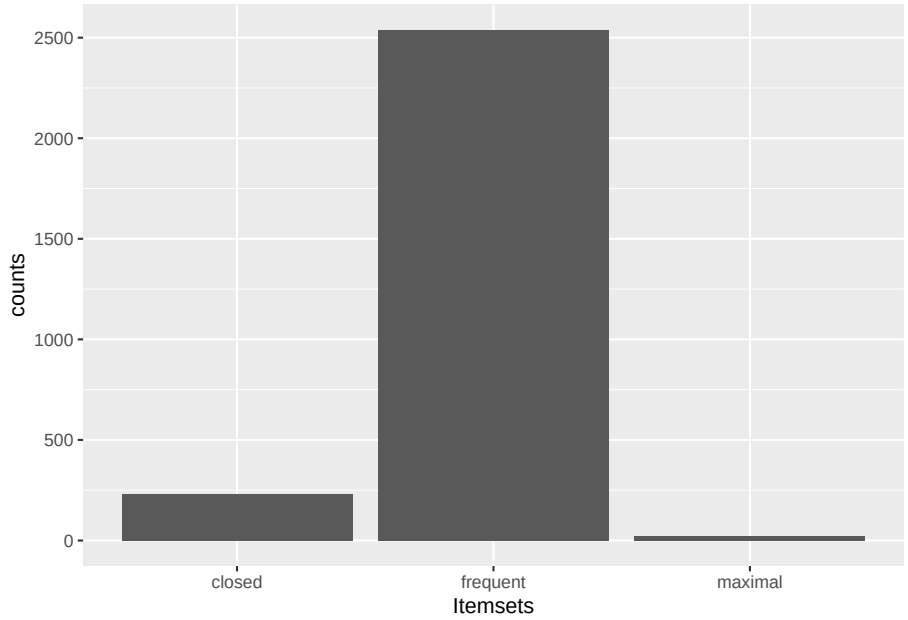
Find maximal frequent itemsets (no superset if frequent)

```
its_max <- its[is.maximal(its)]
its_max
## set of 22 itemsets
its_max |> head(by = "support") |> inspect()
##     items          support count
## [1] {hair,
##      milk,
##      predator,
##      toothed,
##      backbone,
##      breathes,
##      legs,
##      tail,
##      catsize,
##      type=mammal} 0.12871    13
## [2] {eggs,
##      aquatic,
##      predator,
##      toothed,
##      backbone,
##      fins,
##      tail,
##      type=fish}   0.08911     9
## [3] {aquatic,
##      predator,
##      toothed,
##      backbone,
##      breathes}    0.07921     8
## [4] {aquatic,
##      predator,
##      toothed,
##      backbone,
##      fins,
##      tail,
##      catsize}     0.06931     7
## [5] {eggs,
```

```
##       venomous}    0.05941     6
## [6] {predator,
##       venomous}    0.05941     6
```

Find closed frequent itemsets (no superset if frequent)

```
its_closed <- its[is.closed(its)]
its_closed
## set of 230 itemsets
its_closed |> head(by = "support") |> inspect()
##     items             support count
## [1] {backbone}        0.8218  83
## [2] {breathes}        0.7921  80
## [3] {legs}            0.7723  78
## [4] {tail}            0.7426  75
## [5] {backbone, tail}  0.7327  74
## [6] {breathes, legs}  0.7228  73
counts <- c(
  frequent=length(its),
  closed=length(its_closed),
  maximal=length(its_max)
)

ggplot(as_tibble(counts, rownames = "Itemsets"),
  aes(Itemsets, counts)) + geom_bar(stat = "identity")
```



5.5 Association Rule Visualization*

Visualization is a very powerful approach to analyse large sets of mined association rules and frequent itemsets. We present here some options to create static visualizations and inspect rule sets interactively.

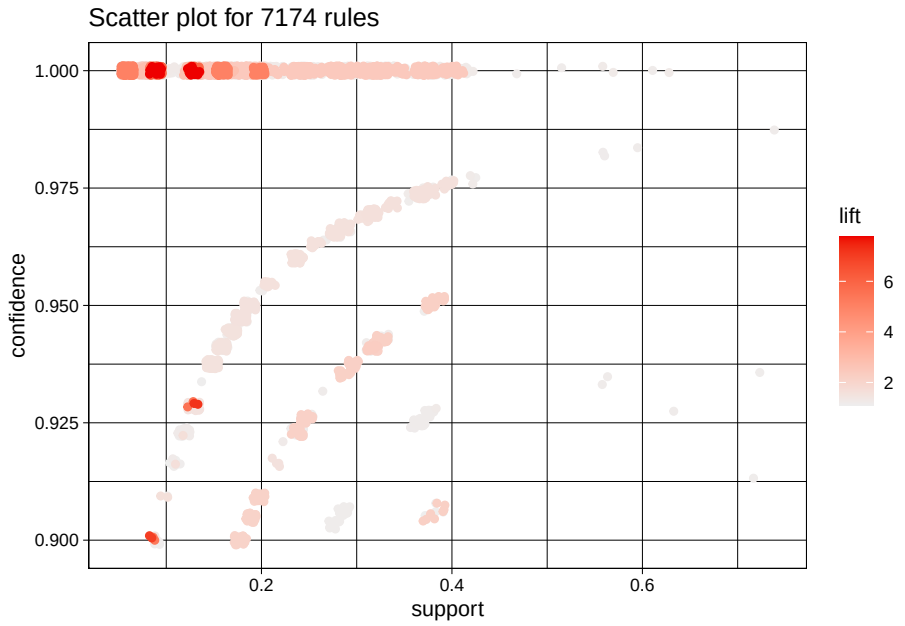
5.5.1 Static Visualizations

Load the `arulesViz` library.

```
library(arulesViz)
```

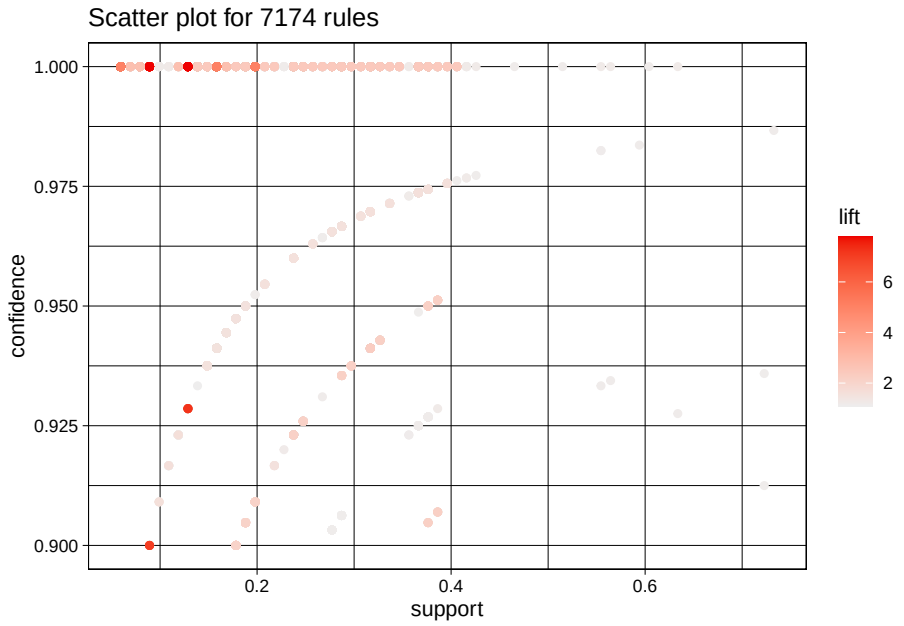
Default scatterplot

```
plot(rules)
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```

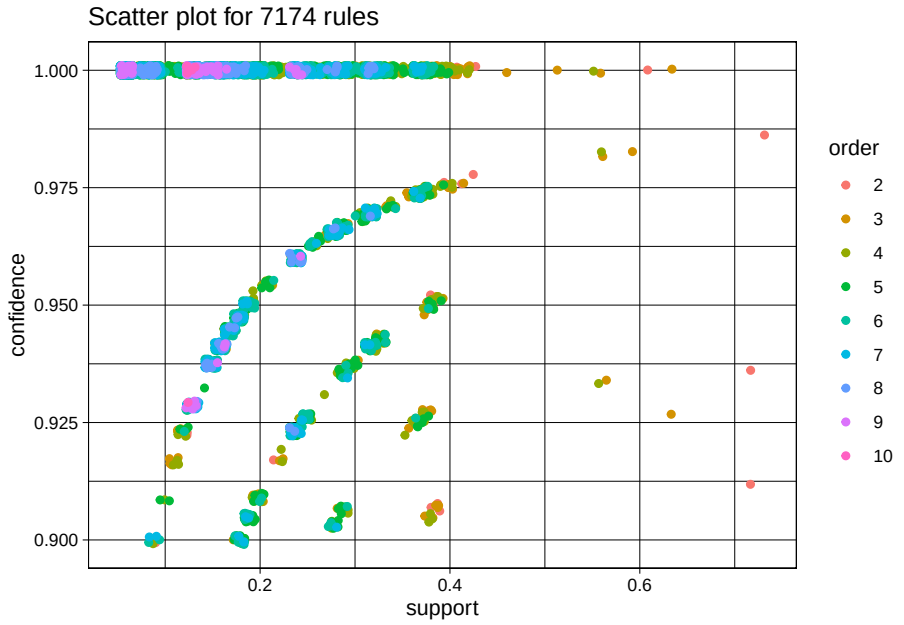


Note that some jitter (randomly move points) was added to show how many rules have the same confidence and support value. Without jitter:

```
plot(rules, control = list(jitter = 0))
```

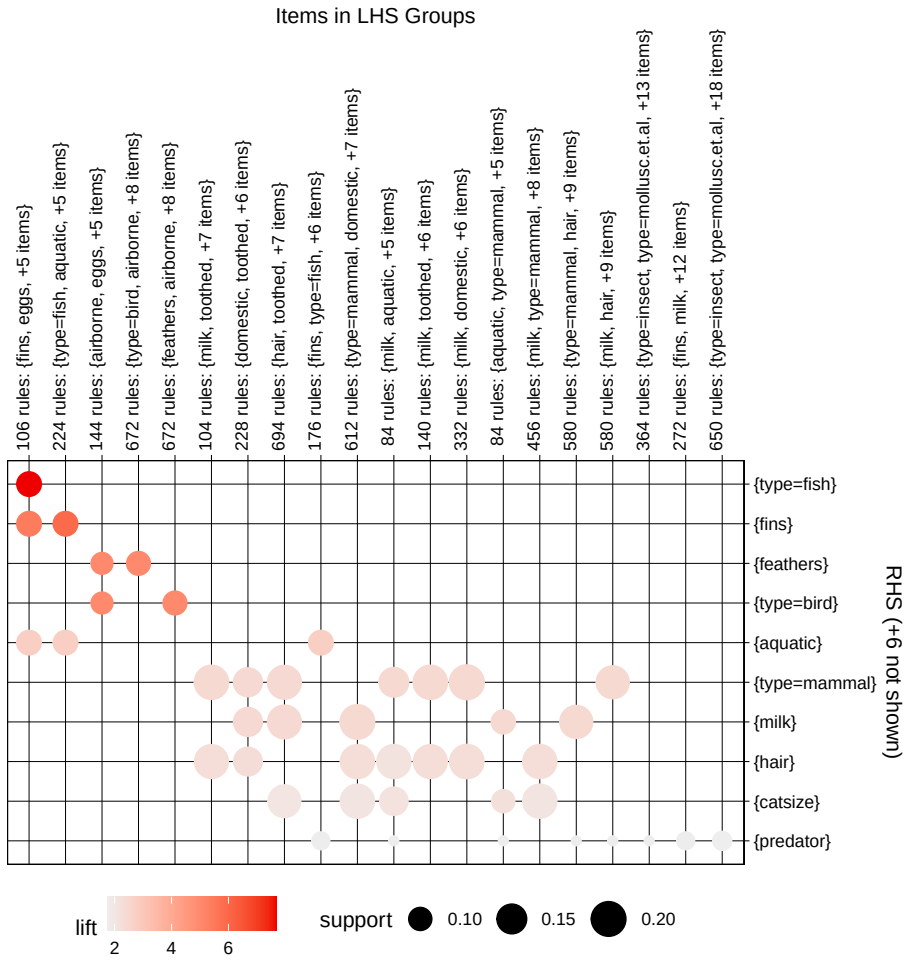



```
plot(rules, shading = "order")
## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.
```



Grouped plot

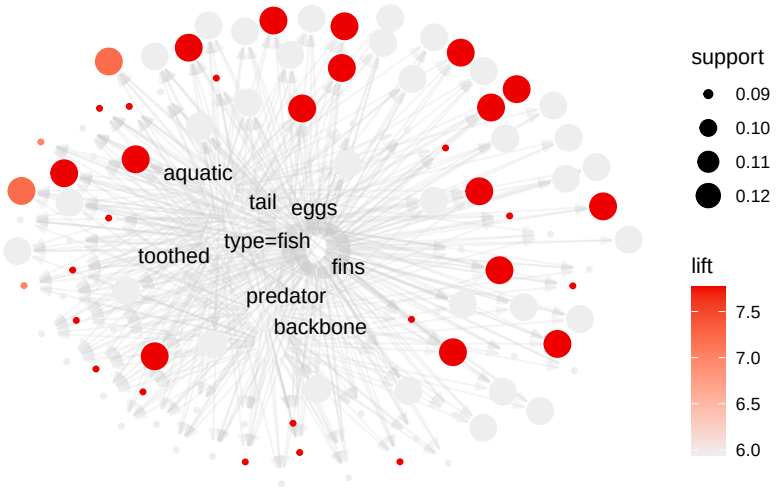
```
plot(rules, method = "grouped")
```



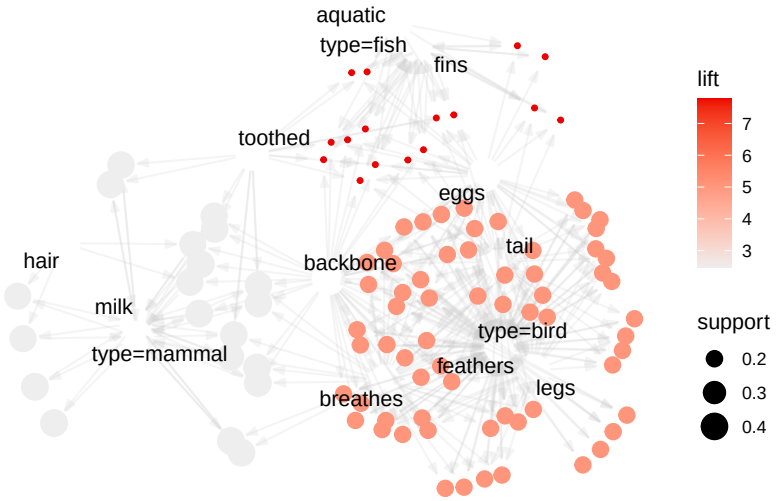
This plot can also be used interactively using the parameter `engine = "interactive"`.

As a graph

```
plot(rules, method = "graph")
## Warning: Too many rules supplied. Only plotting the best
## 100 using 'lift' (change control parameter max if needed).
```




```
plot(rules |> head(by = "phi", n = 100), method = "graph")
```



5.5.2 Interactive Visualizations

We will use the association rules mined from the Iris dataset for the following examples.

```
data(iris)
summary(iris)
##   Sepal.Length   Sepal.Width    Petal.Length   Petal.Width
##  Min.   :4.30   Min.   :2.00   Min.   :1.00   Min.   :0.1
##  1st Qu.:5.10   1st Qu.:2.80   1st Qu.:1.60   1st Qu.:0.3
##  Median :5.80   Median :3.00   Median :4.35   Median :1.3
##  Mean   :5.84   Mean   :3.06   Mean   :3.76   Mean   :1.2
##  3rd Qu.:6.40   3rd Qu.:3.30   3rd Qu.:5.10   3rd Qu.:1.8
##  Max.   :7.90   Max.   :4.40   Max.   :6.90   Max.   :2.5
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

Convert the data to transactions.

```
iris_trans <- transactions(iris)
## Warning: Column(s) 1, 2, 3, 4 not logical or factor.
## Applying default discretization (see '? discretizeDF').
```

Note that this conversion gives a warning to indicate that some potentially unwanted conversion happens. Some features are numeric and need to be discretized. The conversion automatically applies frequency-based discretization with 3 classes to each numeric feature, however, the use may want to use a different discretization strategy.

```
iris_trans |> head() |> inspect()
##     items                       transactionID
## [1] {Sepal.Length=[4.3,5.4),
##      Sepal.Width=[3.2,4.4],
##      Petal.Length=[1,2.63),
##      Petal.Width=[0.1,0.867),
##      Species=setosa}                        1
## [2] {Sepal.Length=[4.3,5.4),
##      Sepal.Width=[2.9,3.2),
##      Petal.Length=[1,2.63),
##      Petal.Width=[0.1,0.867),
##      Species=setosa}                        2
## [3] {Sepal.Length=[4.3,5.4),
##      Sepal.Width=[3.2,4.4],
```

```
##      Petal.Length=[1,2.63),
##      Petal.Width=[0.1,0.867),
##      Species=setosa}                        3
## [4] {Sepal.Length=[4.3,5.4),
##      Sepal.Width=[2.9,3.2),
##      Petal.Length=[1,2.63),
##      Petal.Width=[0.1,0.867),
##      Species=setosa}                        4
## [5] {Sepal.Length=[4.3,5.4),
##      Sepal.Width=[3.2,4.4],
##      Petal.Length=[1,2.63),
##      Petal.Width=[0.1,0.867),
##      Species=setosa}                        5
## [6] {Sepal.Length=[5.4,6.3),
##      Sepal.Width=[3.2,4.4],
##      Petal.Length=[1,2.63),
##      Petal.Width=[0.1,0.867),
##      Species=setosa}                        6
```

Next, we mine association rules.

```
rules <- apriori(iris_trans, parameter = list(support = 0.1,
                                              confidence = 0.8))
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime
##         0.8    0.1    1 none FALSE            TRUE       5
##  support minlen maxlen target  ext
##      0.1      1     10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 15
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[15 item(s), 150 transaction(s)] done [0.00s].
## sorting and recoding items ... [15 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 done [0.00s].
## writing ... [144 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
rules
## set of 144 rules
```

5.5.2.1 Interactive Inspect With Sorting, Filtering and Paging

```
inspectDT(rules,options = list(scrollX = TRUE))
```

The resulting interactive table can be seen in the online version of this book.[11]

5.5.2.2 Scatter Plot

Plot rules as a scatter plot using an interactive html widget. To avoid overplotting, jitter is added automatically. Set `jitter = 0` to disable jitter. Hovering over rules shows rule information. *Note:* plotly/javascript does not do well with too many points, so plot selects the top 1000 rules with a warning if more rules are supplied.

```
plot(rules, engine = "html")
```

The resulting interactive plot can be seen in the online version of this book.[12]

5.5.2.3 Matrix Visualization

Plot rules as a matrix using an interactive html widget.

```
plot(rules, method = "matrix", engine = "html")
```

The resulting interactive plot can be seen in the online version of this book.[13]

5.5.2.4 Visualization as Graph

Plot rules as a graph using an interactive html widget. *Note:* the used javascript library does not do well with too many graph nodes, so plot selects the top 100 rules only (with a warning).

```
plot(rules, method = "graph", engine = "html")
```

The resulting interactive plot can be seen in the online version of this book.[14]

5.5.2.5 Interactive Rule Explorer

You can specify a rule set or a dataset. To explore rules that can be mined from iris, use: `ruleExplorer(iris)`

[11] https://mhahsler.github.io/Introduction_to_Data_Mining_R_Examples/book/association-analysis-basic-concepts-and-algorithms.html#interactive-visualizations

[12] https://mhahsler.github.io/Introduction_to_Data_Mining_R_Examples/book/association-analysis-basic-concepts-and-algorithms.html#interactive-visualizations

[13] https://mhahsler.github.io/Introduction_to_Data_Mining_R_Examples/book/association-analysis-basic-concepts-and-algorithms.html#interactive-visualizations

[14] https://mhahsler.github.io/Introduction_to_Data_Mining_R_Examples/book/association-analysis-basic-concepts-and-algorithms.html#interactive-visualizations

The rule explorer creates an interactive Shiny application that can be used locally or deployed on a server for sharing. A deployed version of the ruleExplorer is available here[15] (using shinyapps.io[16]).

5.6 Exercises*

We will again use the Palmer penguin data for the exercises.

```
library(palmerpenguins)
head(penguins)
## # A tibble: 6 x 8
##   species island    bill_length_mm bill_depth_mm
##   <chr>   <chr>              <dbl>         <dbl>
## 1 Adelie  Torgersen           39.1          18.7
## 2 Adelie  Torgersen           39.5          17.4
## 3 Adelie  Torgersen           40.3          18
## 4 Adelie  Torgersen           NA            NA
## 5 Adelie  Torgersen           36.7          19.3
## 6 Adelie  Torgersen           39.3          20.6
## # i 4 more variables: flipper_length_mm <dbl>,
## #   body_mass_g <dbl>, sex <chr>, year <dbl>
```

1. Translate the penguin data into transaction data with:

```
trans <- transactions(penguins)
## Warning: Column(s) 1, 2, 3, 4, 5, 6, 7, 8 not logical or
## factor. Applying default discretization (see '?
## discretizeDF').
## Warning in discretize(x = c(2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, : The calculated b
##   Only unique breaks are used reducing the number of intervals. Look at ? discretize for detai
trans
## transactions in sparse format with
##  344 transactions (rows) and
##  22 items (columns)
```

Why does the conversion report warnings?

2. What do the following first three transactions mean?

```
inspect(trans[1:3])
##     items                              transactionID
## [1] {species=Adelie,
##      island=Torgersen,
##      bill_length_mm=[32.1,40.8),
##      bill_depth_mm=[18.3,21.5],
```

[15] https://mhahsler-apps.shinyapps.io/ruleExplorer_demo/
[16] https://www.shinyapps.io/

```
##       flipper_length_mm=[172,192),
##       body_mass_g=[3.7e+03,4.55e+03),
##       sex=male,
##       year=[2007,2008)}                          1
## [2] {species=Adelie,
##       island=Torgersen,
##       bill_length_mm=[32.1,40.8),
##       bill_depth_mm=[16.2,18.3),
##       flipper_length_mm=[172,192),
##       body_mass_g=[3.7e+03,4.55e+03),
##       sex=female,
##       year=[2007,2008)}                          2
## [3] {species=Adelie,
##       island=Torgersen,
##       bill_length_mm=[32.1,40.8),
##       bill_depth_mm=[16.2,18.3),
##       flipper_length_mm=[192,209),
##       body_mass_g=[2.7e+03,3.7e+03),
##       sex=female,
##       year=[2007,2008)}                          3
```

Next, use the `ruleExplorer()` function to analyze association rules created for the transaction data set.

1. Use the default settings for the parameters. Using the *Data Table*, what is the association rule with the highest lift. What does its LHS, RHS, support, confidence and lift mean?

2. Use the *Graph* visualization. Use select by id to highlight different species and different islands and then hover over some of the rules. What do you see?

Chapter 6

Association Analysis: Advanced Concepts

This chapter discusses a few advanced concepts of association analysis. First, we look at how categorical and continuous attributes are converted into items. The we look at integrating item hierarchies into the analysis. Finally, sequence pattern mining is introduced.

Packages Used in this Chapter

```
pkgs <- c("arules", "arulesSequences", "tidyverse")

pkgs_install <- pkgs[!(pkgs %in% installed.packages()[,"Package"])]
if(length(pkgs_install)) install.packages(pkgs_install)
```

The packages used for this chapter are:

- *arules* (Hahsler et al. 2024)
- *arulesSequences* (Buchta and Hahsler 2024)
- *tidyverse* (Wickham 2023c)

6.1 Handling Categorical Attributes

Categorical attributes are nominal or ordinal variables. In R they are `factors` or `ordinal`. They are translated into a series of binary items (one for each level constructed as variable `name = level`). Items cannot represent order and this ordered factors lose the order information. Note that nominal variables need to be encoded as factors (and not characters or numbers) before converting them into transactions.

For the special case of Boolean variables (`logical`), the `TRUE` value is converted into an item with the name of the variable and for the `FALSE` values no item is created.

We will give an example in the next section.

6.2 Handling Continuous Attributes

Continuous variables cannot directly be represented as items and need to be discretized first (see Discretization in Chapter 2). An item resulting from discretization might be `age>18` and the column contains only `TRUE` or `FALSE`. Alternatively, it can be a factor with levels `age<=18`, `50=>age>18` and `age>50`. These will be automatically converted into 3 items, one for each level. Discretization is described in functions `discretize()` and `discretizeDF()` to discretize all columns in a data.frame.

We give a short example using the iris dataset. We add an extra `logical` column to show how Boolean attributes are converted in to items.

```
data(iris)

## add a Boolean attribute
iris$Versicolor <- iris$Species == "versicolor"
head(iris)
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
##   Versicolor
## 1      FALSE
## 2      FALSE
## 3      FALSE
## 4      FALSE
## 5      FALSE
## 6      FALSE
```

The first step is to discretize continuous attributes (marked as `<dbl>` in the table above). We discretize the two Petal features.

```
library(tidyverse)
library(arules)

iris_disc <- iris %>%
  mutate(Petal.Length = discretize(Petal.Length,
```

```
                          method = "frequency",
                          breaks = 3,
                          labels = c("short", "medium", "long")),
         Petal.Width = discretize(Petal.Width,
                          method = "frequency",
                          breaks = 2,
                          labels = c("narrow", "wide"))
         )


head(iris_disc)
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5        short      narrow  setosa
## 2          4.9         3.0        short      narrow  setosa
## 3          4.7         3.2        short      narrow  setosa
## 4          4.6         3.1        short      narrow  setosa
## 5          5.0         3.6        short      narrow  setosa
## 6          5.4         3.9        short      narrow  setosa
##   Versicolor
## 1      FALSE
## 2      FALSE
## 3      FALSE
## 4      FALSE
## 5      FALSE
## 6      FALSE
```

Next, we convert the dataset into transactions.

```
trans <- transactions(iris_disc)
## Warning: Column(s) 1, 2 not logical or factor. Applying
## default discretization (see '? discretizeDF').
trans
## transactions in sparse format with
##  150 transactions (rows) and
##  15 items (columns)
```

The conversion creates a warning because there are still two undiscretized columns in the data. The warning indicates that the default discretization is used automatically.

```
itemLabels(trans)
##  [1] "Sepal.Length=[4.3,5.4)" "Sepal.Length=[5.4,6.3)"
##  [3] "Sepal.Length=[6.3,7.9]" "Sepal.Width=[2,2.9)"
##  [5] "Sepal.Width=[2.9,3.2)"  "Sepal.Width=[3.2,4.4]"
##  [7] "Petal.Length=short"     "Petal.Length=medium"
##  [9] "Petal.Length=long"      "Petal.Width=narrow"
## [11] "Petal.Width=wide"       "Species=setosa"
```

```
## [13] "Species=versicolor"     "Species=virginica"
## [15] "Versicolor"
```

We see that all continuous variables are discretized and the different ranges create an item. For example `Petal.Width` has the two items `Petal.Width=narrow` and `Petal.Width=wide`. The automatically discretized variables show intervals. `Sepal.Length=[4.3,5.4)` means that this item used for flowers with a sepal length between 4.3 and 5.4 cm.

The species is converted into three items, one for each class. The logical variable `Versicolor` created only a single item that is used when the variable is `TRUE`.

6.3 Handling Concept Hierarchies

Often an item hierarchy is available for transactions used for association rule mining. For example in a supermarket dataset items like "bread" and "beagle" might belong to the item group (category) "baked goods." Transactions can store item hierarchies as additional columns in the itemInfo data.frame.

6.3.1 Aggregation

To perform analysis at a group level of the item hierarchy, `aggregate()` produces a new object with items aggregated to a given group level. A group-level item is present if one or more of the items in the group are present in the original object. If rules are aggregated, and the aggregation would lead to the same aggregated group item in the lhs and in the rhs, then that group item is removed from the lhs. Rules or itemsets, which are not unique after the aggregation, are also removed. Note also that the quality measures are not applicable to the new rules and thus are removed. If these measures are required, then aggregate the transactions before mining rules.

We use the Groceries data set in this example. It contains 1 month (30 days) of real-world point-of-sale transaction data from a typical local grocery outlet. The items are 169 products categories.

```
data("Groceries")
Groceries
## transactions in sparse format with
##  9835 transactions (rows) and
##  169 items (columns)
```

The dataset also contains two aggregation levels.

```
head(itemInfo(Groceries))
##              labels  level2           level1
## 1       frankfurter sausage meat and sausage
## 2           sausage sausage meat and sausage
```

```
## 3        liver loaf sausage meat and sausage
## 4               ham sausage meat and sausage
## 5             meat sausage meat and sausage
## 6 finished products sausage meat and sausage
```

We aggregate to level1 stored in Groceries. All items with the same level2 label will become a single item with that name. This reduces the number of items to the 55 level2 categories

```
Groceries_level2 <- aggregate(Groceries, by = "level2")
Groceries_level2
## transactions in sparse format with
##  9835 transactions (rows) and
##  55 items (columns)
head(itemInfo(Groceries_level2)) ## labels are alphabetically sorted!
##             labels           level2           level1
## 1        baby food        baby food      canned food
## 2             bags             bags         non-food
## 3  bakery improver  bakery improver   processed food
## 4 bathroom cleaner bathroom cleaner        detergent
## 5             beef             beef meat and sausage
## 6             beer             beer           drinks
```

We can now compare an original transaction with the aggregated transaction.

```
inspect(head(Groceries, 3))
##     items
## [1] {citrus fruit,
##      semi-finished bread,
##      margarine,
##      ready soups}
## [2] {tropical fruit,
##      yogurt,
##      coffee}
## [3] {whole milk}
inspect(head(Groceries_level2, 3))
##     items
## [1] {bread and backed goods,
##      fruit,
##      soups/sauces,
##      vinegar/oils}
## [2] {coffee,
##      dairy produce,
##      fruit}
## [3] {dairy produce}
```

For example, citrus fruit in the first transaction was translated to the category

fruit. Note that the order of items in a transaction is not important, so it might change during aggregation.

It is now easy to mine rules on the aggregated data.

```
rules <- apriori(Groceries_level2, support = 0.005)
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime
##         0.8    0.1    1 none FALSE            TRUE       5
##  support minlen maxlen target  ext
##    0.005      1     10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 49
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[55 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [47 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 done [0.00s].
## writing ... [243 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
rules |> head(3, by = "support") |> inspect()
##     lhs                          rhs             support confidence coverage  lift
## [1] {bread and backed goods,
##      cheese,
##      fruit}                   => {dairy produce} 0.02481     0.8385  0.02959 1.893
## [2] {bread and backed goods,
##      cheese,
##      vegetables}              => {dairy produce} 0.02379     0.8239  0.02888 1.860
## [3] {cheese,
##      fruit,
##      vegetables}              => {dairy produce} 0.02267     0.8479  0.02674 1.914
```

You can add your own aggregation to an existing dataset by constructing the and iteminfo data.frame and adding it to the transactions. See `? hierarchy` for details.

6.3.2 Multi-level Analysis

To analyze relationships between individual items and item groups at the same time, `addAggregate()` can be used to create a new transactions object which

contains both, the original items and group-level items.

```
Groceries_multilevel <- addAggregate(Groceries, "level2")
Groceries_multilevel |> head(n=3) |> inspect()
##     items
## [1] {citrus fruit,
##      semi-finished bread,
##      margarine,
##      ready soups,
##      bread and backed goods*,
##      fruit*,
##      soups/sauces*,
##      vinegar/oils*}
## [2] {tropical fruit,
##      yogurt,
##      coffee,
##      coffee*,
##      dairy produce*,
##      fruit*}
## [3] {whole milk,
##      dairy produce*}
```

The added group-level items are marked with an * after the name. Now we can mine rules including items from multiple levels.

```
rules <- apriori(Groceries_multilevel,
  parameter = list(support = 0.005))
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime
##         0.8    0.1    1 none FALSE            TRUE       5
##  support minlen maxlen target  ext
##    0.005      1     10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 49
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[224 item(s), 9835 transaction(s)] done [0.01s].
## sorting and recoding items ... [167 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 7 8 done [0.05s].
## writing ... [21200 rule(s)] done [0.00s].
```

```
## creating S4 object  ... done [0.01s].
rules
## set of 21200 rules
```

Mining rules with group-level items added will create many spurious rules of the type

```
item A => group of item A
```

with a confidence of 1. This will also happen if you mine itemsets. `filterAggregate()` can be used to filter these spurious rules or itemsets.

```
rules <- filterAggregate(rules)
rules
## set of 838 rules
rules |> head(n = 3, by = "lift") |> inspect()
##     lhs                            rhs                          support confidence cov
## [1] {whole milk,
##      whipped/sour cream,
##      bread and backed goods*,
##      cheese*}                   => {vegetables*}              0.005186     0.8095 0.0
## [2] {sausage,
##      poultry*}                  => {vegetables*}              0.005084     0.8065 0.0
## [3] {other vegetables,
##      soda,
##      fruit*,
##      sausage*}                  => {bread and backed goods*} 0.005287     0.8525 0.0
```

Using multi-level mining can reduce the number of rules and help to analyze if customers differentiate between products in a group.

6.4 Sequential Patterns

The frequent sequential pattern mining algorithm cSPADE (Zaki 2000) is implemented in the `arules` extension package `arulesSequences`.

Sequential pattern mining starts with sequences of events. Each sequence is identified by a sequence ID and each event is a set of items that happen together. The order of events is specified using event IDs. The goal is to find subsequences of items in events that follow each other frequently. These are called frequent sequential pattern.

We will look at a small example dataset that comes with the package arulesSequences.

```
library(arulesSequences)
##
## Attaching package: 'arulesSequences'
```

```
## The following object is masked from 'package:arules':
##
##     itemsets
data(zaki)

inspect(zaki)
##      items        sequenceID eventID SIZE
## [1]  {C, D}       1          10      2
## [2]  {A, B, C}    1          15      3
## [3]  {A, B, F}    1          20      3
## [4]  {A, C, D, F} 1          25      4
## [5]  {A, B, F}    2          15      3
## [6]  {E}          2          20      1
## [7]  {A, B, F}    3          10      3
## [8]  {D, G, H}    4          10      3
## [9]  {B, F}       4          20      2
## [10] {A, G, H}    4          25      3
```

The dataset contains four sequences (see `sequenceID`) and the event IDs are integer numbers to provide the order events in a sequence. In arulesSequences, this set of sequences is implemented as a regular transaction set, where each transaction is an event. The temporal information is added as extra columns to the transaction's `transactionInfo()` data.frame.

Mine frequent sequence patterns using cspade is very similar to using apriori. Here we set support so we will find patterns that occur in 50% of the sequences.

```
fsp <- cspade(zaki, parameter = list(support = .5))
fsp |> inspect()
##     items support
##   1 <{A}>    1.00
##   2 <{B}>    1.00
##   3 <{D}>    0.50
##   4 <{F}>    1.00
##   5 <{A,
##      F}>    0.75
##   6 <{B,
##      F}>    1.00
##   7 <{D},
##     {F}>    0.50
##   8 <{D},
##     {B,
##      F}>    0.50
##   9 <{A,
##      B,
##      F}>    0.75
```

```
##  10 <{A,
##      B}>    0.75
##  11 <{D},
##      {B}>    0.50
##  12 <{B},
##      {A}>    0.50
##  13 <{D},
##      {A}>    0.50
##  14 <{F},
##      {A}>    0.50
##  15 <{D},
##      {F},
##      {A}>    0.50
##  16 <{B,
##      F},
##      {A}>    0.50
##  17 <{D},
##      {B,
##      F},
##      {A}>    0.50
##  18 <{D},
##      {B},
##      {A}>    0.50
##
```

For example, pattern 17 shows that D in an event, it is often followed by an event by containing B and F which in turn is followed by an event containing A.

The cspade algorithm supports many additional parameters to control gaps and windows. Details can be found in the manual page for `cspade`.

Rules, similar to regular association rules can be generated from frequent sequence patterns using `ruleInduction()`.

```
rules <- ruleInduction(fsp, confidence = .8)
rules |> inspect()
##    lhs      rhs   support confidence lift
##  1 <{D}> => <{F}>     0.5          1    1
##  2 <{D}> => <{B,      0.5          1    1
##               F}>
##  3 <{D}> => <{B}>     0.5          1    1
##  4 <{D}> => <{A}>     0.5          1    1
##  5 <{D},
##     {F}> => <{A}>     0.5          1    1
##  6 <{D},
##     {B},
```

```
##      F}> => <{A}>     0.5          1    1
##  7 <{D},
##     {B}> => <{A}>     0.5          1    1
##
```

The usual measures of confidence and lift are used.

Chapter 7

Cluster Analysis

This chapter introduces cluster analysis using K-means, hierarchical clustering and DBSCAN. We will discuss how to choose the number of clusters and how to evaluate the quality clusterings. In addition, we will introduce more clustering algorithms and how clustering is influenced by outliers.

The corresponding chapter of the data mining textbook is available online: Chapter 7: Cluster Analysis: Basic Concepts and Algorithms.[1]

Packages Used in this Chapter

```
pkgs <- c("cluster", "dbscan", "e1071", "factoextra", "fpc",
          "GGally", "kernlab", "mclust", "mlbench", "scatterpie",
          "seriation", "tidyverse")

pkgs_install <- pkgs[!(pkgs %in% installed.packages()[,"Package"])]
if(length(pkgs_install)) install.packages(pkgs_install)
```

The packages used for this chapter are:

- *cluster* (Maechler et al. 2023)
- *dbscan* (Hahsler and Piekenbrock 2024)
- *e1071* (Meyer et al. 2024)
- *factoextra* (Kassambara and Mundt 2020)
- *fpc* (Hennig 2024)
- *GGally* (Schloerke et al. 2024)
- *kernlab* (Karatzoglou, Smola, and Hornik 2024)
- *mclust* (Fraley, Raftery, and Scrucca 2024)
- *mlbench* (Leisch and Dimitriadou 2024)
- *scatterpie* (Yu 2024)

[1] https://www-users.cs.umn.edu/~kumar001/dmbook/ch7_clustering.pdf

- *seriation* (Hahsler, Buchta, and Hornik 2024)
- *tidyverse* (Wickham 2023c)

7.1 Overview

Cluster analysis or clustering[2] is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters).

Clustering is also called unsupervised learning, because it tries to directly learns the structure of the data and does not rely on the availability of a correct answer or class label as supervised learning does. Clustering is often used for exploratory analysis or to preprocess data by grouping.

You can read the free sample chapter from the textbook (Tan, Steinbach, and Kumar 2005): Chapter 7. Cluster Analysis: Basic Concepts and Algorithms[3]

7.1.1 Data Preparation

```
library(tidyverse)
```

We will use here a small and very clean toy dataset called Ruspini which is included in the R package **cluster**.

```
data(ruspini, package = "cluster")
```

The Ruspini data set, consisting of 75 points in four groups that is popular for illustrating clustering techniques. It is a very simple data set with well separated clusters. The original dataset has the points ordered by group. We can shuffle the data (rows) using `sample_frac` which samples by default 100%.

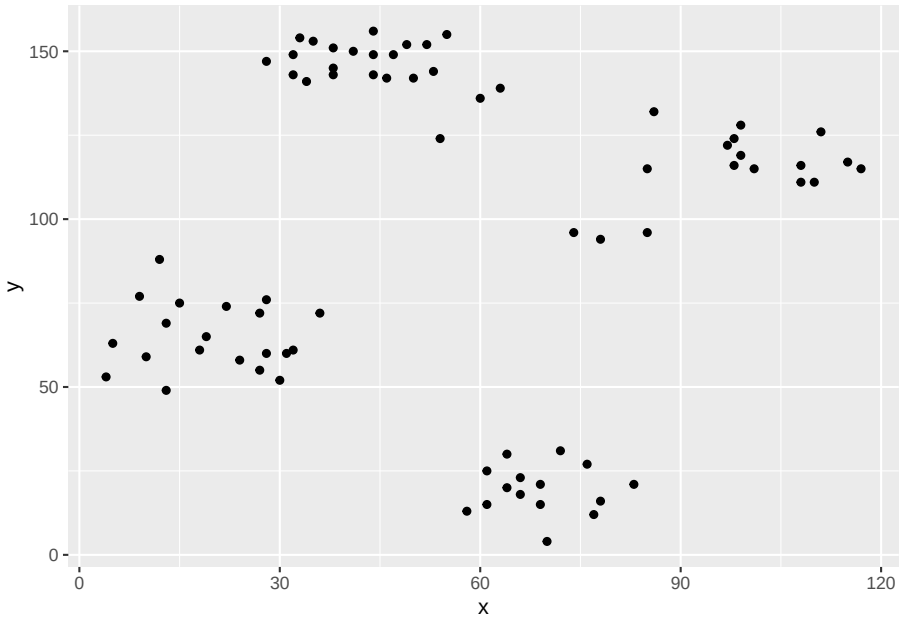
```
ruspini <- as_tibble(ruspini) |>
  sample_frac()
ruspini
## # A tibble: 75 x 2
##        x     y
##    <int> <int>
##  1    66    18
##  2    32   143
##  3    28    76
##  4    38   143
##  5    41   150
##  6   108   116
##  7   110   111
##  8   115   117
```

[2] https://en.wikipedia.org/wiki/Cluster_analysis
[3] https://www-users.cs.umn.edu/~kumar001/dmbook/ch7_clustering.pdf

```
##  9    35   153
## 10     9    77
## # i 65 more rows
```

7.1.2 Data cleaning

```
ggplot(ruspini, aes(x = x, y = y)) + geom_point()
```




```
summary(ruspini)
##        x                y        
##  Min.   :  4.0   Min.   :  4.0  
##  1st Qu.: 31.5   1st Qu.: 56.5  
##  Median : 52.0   Median : 96.0  
##  Mean   : 54.9   Mean   : 92.0  
##  3rd Qu.: 76.5   3rd Qu.:141.5  
##  Max.   :117.0   Max.   :156.0
```

For most clustering algorithms it is necessary to handle missing values and outliers (e.g., remove the observations). For details see Section "Outlier removal" below. This data set has not missing values or strong outlier and looks like it has some very clear groups.

7.1.3 Scale data

Clustering algorithms use distances and the variables with the largest number range will dominate distance calculation. The summary above shows that this is not an issue for the Ruspini dataset with both, x and y, being roughly between 0 and 150. Most data analysts will still scale each column in the data to zero mean and unit standard deviation (z-scores[4]).

Note: The standard `scale()` function scales a whole data matrix so we implement a function for a single vector and apply it to all numeric columns.

```
## I use this till tidyverse implements a scale function
scale_numeric <- function(x) {
  x |> mutate(across(where(is.numeric),
                     function(y) as.vector(scale(y))))
}
```

```
ruspini_scaled <- ruspini |>
  scale_numeric()
summary(ruspini_scaled)
##        x                 y
##  Min.   :-1.6681   Min.   :-1.8074
##  1st Qu.:-0.7665   1st Qu.:-0.7295
##  Median :-0.0944   Median : 0.0816
##  Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.: 0.7088   3rd Qu.: 1.0158
##  Max.   : 2.0366   Max.   : 1.3136
```

After scaling, most z-scores will fall in the range $[-3, 3]$ (z-scores are measured in standard deviations from the mean), where 0 means average.

7.2 K-means

k-means[5] implicitly assumes Euclidean distances. We use $k = 4$ clusters and run the algorithm 10 times with random initialized centroids. The best result is returned.

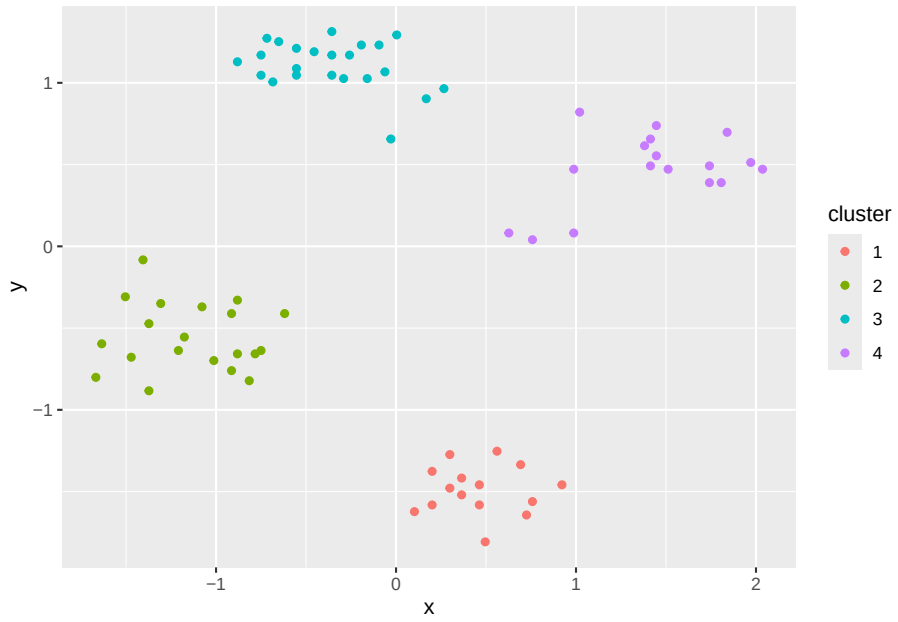
```
km <- kmeans(ruspini_scaled, centers = 4, nstart = 10)
km
## K-means clustering with 4 clusters of sizes 15, 20, 23, 17
##
## Cluster means:
##         x       y
## 1  0.4607 -1.4912
## 2 -1.1386 -0.5560
```

[4] https://en.wikipedia.org/wiki/Standard_score
[5] https://en.wikipedia.org/wiki/K-means_clustering

```
## 3 -0.3595  1.1091
## 4  1.4194  0.4693
##
## Clustering vector:
##  [1] 1 3 2 3 3 4 4 4 3 2 2 1 3 2 1 4 3 2 3 4 3 3 1 1 2 4 3 4
## [29] 3 1 1 4 1 4 2 4 2 1 2 1 4 2 3 3 2 2 4 3 2 2 2 4 4 3 1 4
## [57] 3 1 3 3 2 2 3 4 2 4 2 2 3 1 3 1 3 1 3
##
## Within cluster sum of squares by cluster:
## [1] 1.082 2.705 2.659 3.641
##  (between_SS / total_SS =  93.2 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"
## [4] "withinss"     "tot.withinss" "betweenss"
## [7] "size"         "iter"         "ifault"
```

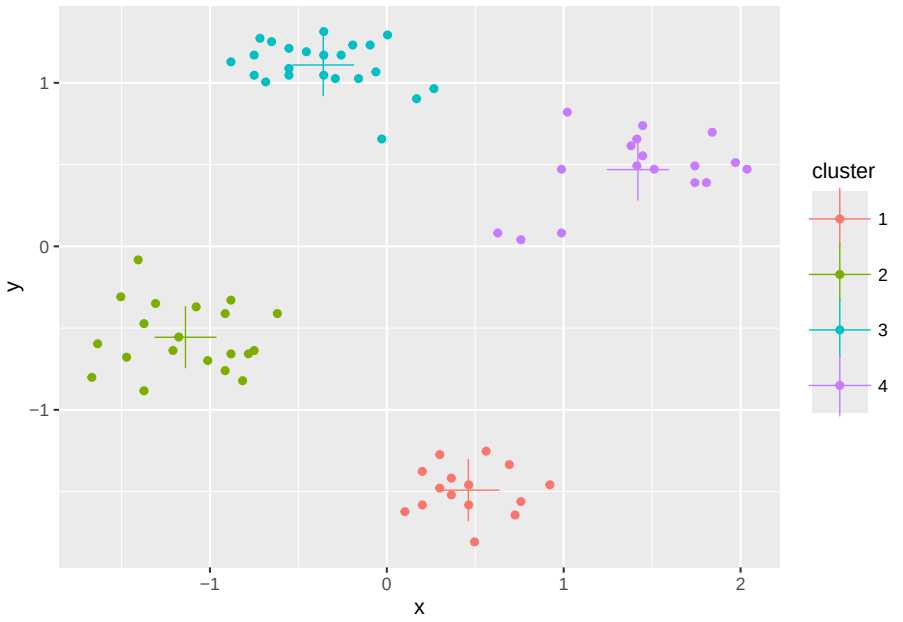
`km` is an R object implemented as a list. The clustering vector contains the cluster assignment for each data row and can be accessed using `km$cluster`. I add the cluster assignment as a column to the scaled dataset (I make it a factor since it represents a nominal label).

```
ruspini_clustered <- ruspini_scaled |>
  add_column(cluster = factor(km$cluster))
ruspini_clustered
## # A tibble: 75 x 3
##         x      y cluster
##     <dbl>  <dbl> <fct>
##  1  0.365 -1.52  1
##  2 -0.750  1.05  3
##  3 -0.881 -0.329 2
##  4 -0.553  1.05  3
##  5 -0.455  1.19  3
##  6  1.74   0.492 4
##  7  1.81   0.390 4
##  8  1.97   0.513 4
##  9 -0.652  1.25  3
## 10 -1.50  -0.309 2
## # i 65 more rows
ggplot(ruspini_clustered, aes(x = x, y = y)) +
  geom_point(aes(color = cluster))
```



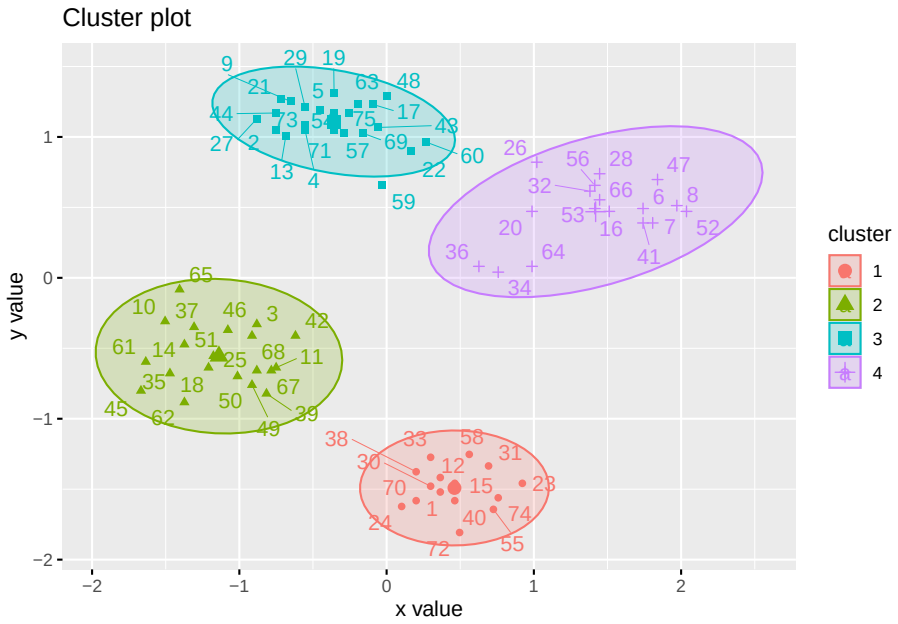
Add the centroids to the plot. Note that the second `geom_points` uses not the original data but specifies the centroids as its dataset.

```
centroids <- as_tibble(km$centers, rownames = "cluster")
centroids
## # A tibble: 4 x 3
##   cluster      x      y
##   <chr>    <dbl>  <dbl>
## 1 1        0.461 -1.49
## 2 2       -1.14  -0.556
## 3 3       -0.360  1.11
## 4 4        1.42   0.469
ggplot(ruspini_clustered, aes(x = x, y = y)) +
  geom_point(aes(color = cluster)) +
  geom_point(data = centroids, aes(x = x, y = y, color = cluster),
             shape = 3, size = 10)
```



The `factoextra` package provides also a good visualization with object labels and ellipses for clusters.

```
library(factoextra)
fviz_cluster(km, data = ruspini_scaled, centroids = TRUE,
             repel = TRUE, ellipse.type = "norm")
```

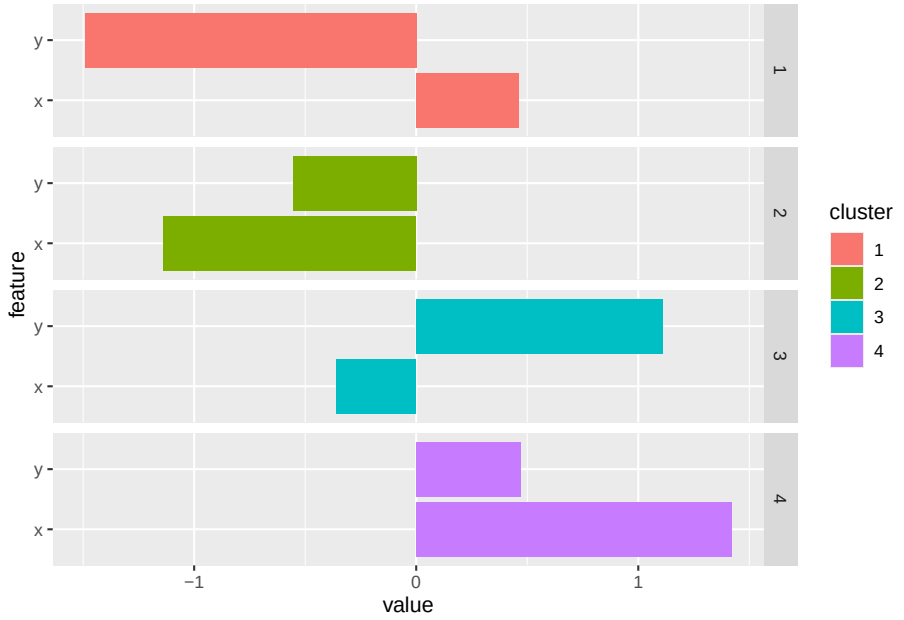


7.2.0.1 Inspect clusters

We inspect the clusters created by the 4-cluster k-means solution. The following code can be adapted to be used for other clustering methods.

7.2.0.1.1 Cluster Profiles Inspect the centroids with horizontal bar charts organized by cluster. To group the plots by cluster, we have to change the data format to the "long"-format using a pivot operation. I use colors to match the clusters in the scatter plots.

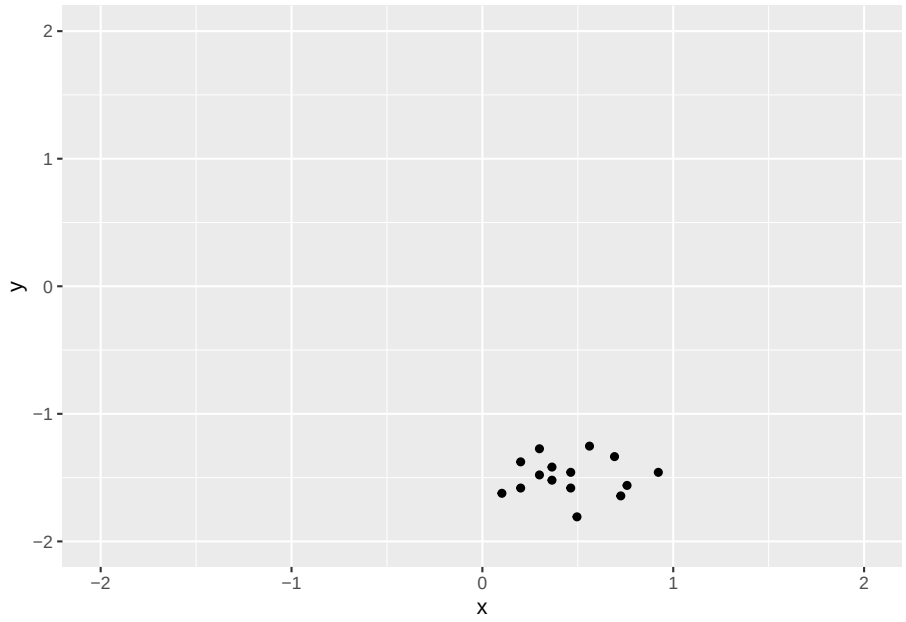
```
ggplot(pivot_longer(centroids,
                    cols = c(x, y),
                    names_to = "feature"),
  aes(x = value, y = feature, fill = cluster)) +
  geom_bar(stat = "identity") +
  facet_grid(rows = vars(cluster))
```



7.2.0.1.2 Extract a single cluster You need is to filter the rows corresponding to the cluster index. The next example calculates summary statistics and then plots all data points of cluster 1.

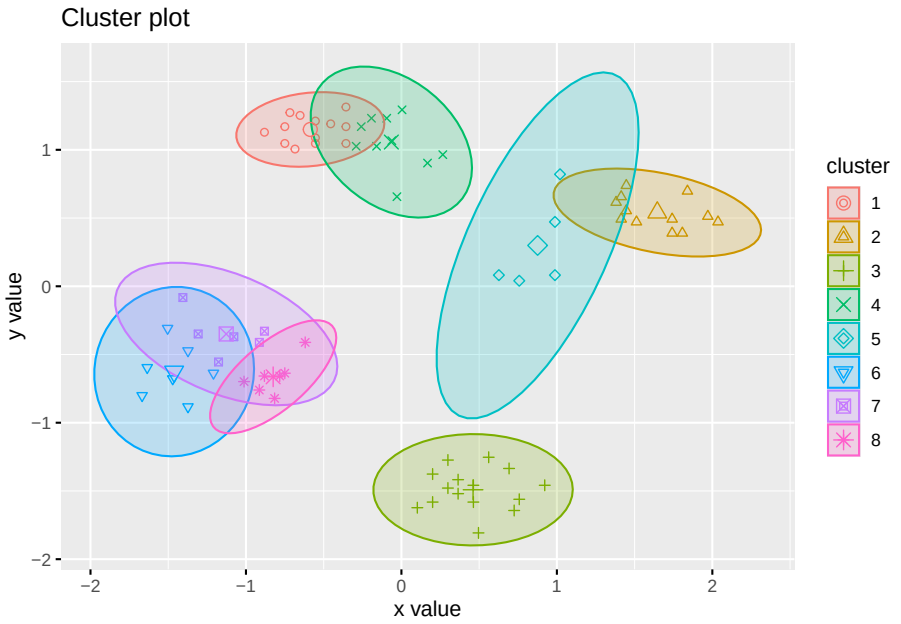
```
cluster1 <- ruspini_clustered |>
  filter(cluster == 1)
cluster1
## # A tibble: 15 x 3
##        x     y cluster
##    <dbl> <dbl> <fct>
##  1 0.365 -1.52 1
##  2 0.365 -1.42 1
##  3 0.463 -1.46 1
##  4 0.922 -1.46 1
##  5 0.102 -1.62 1
##  6 0.299 -1.48 1
##  7 0.692 -1.34 1
##  8 0.299 -1.27 1
##  9 0.201 -1.38 1
## 10 0.463 -1.58 1
## 11 0.725 -1.64 1
## 12 0.561 -1.25 1
## 13 0.201 -1.58 1
## 14 0.496 -1.81 1
## 15 0.758 -1.56 1
```

```
summary(cluster1)
##        x                y          cluster
##  Min.   :0.102   Min.   :-1.81   1:15
##  1st Qu.:0.299   1st Qu.:-1.58   2: 0
##  Median :0.463   Median :-1.48   3: 0
##  Mean   :0.461   Mean   :-1.49   4: 0
##  3rd Qu.:0.627   3rd Qu.:-1.40
##  Max.   :0.922   Max.   :-1.25
ggplot(cluster1, aes(x = x, y = y)) + geom_point() +
  coord_cartesian(xlim = c(-2, 2), ylim = c(-2, 2))
```



What happens if we try to cluster with 8 centers?

```
fviz_cluster(kmeans(ruspini_scaled, centers = 8), data = ruspini_scaled,
  centroids = TRUE,  geom = "point", ellipse.type = "norm")
```



7.3 Agglomerative Hierarchical Clustering

Hierarchical clustering starts with a distance matrix. `dist()` defaults to method="Euclidean". **Note:** Distance matrices become very large quickly (size and time complexity is $O(n^2)$ where n is the number if data points). It is only possible to calculate and store the matrix for small data sets (maybe a few hundred thousand data points) in main memory. If your data is too large then you can use sampling.

```
d <- dist(ruspini_scaled)
```

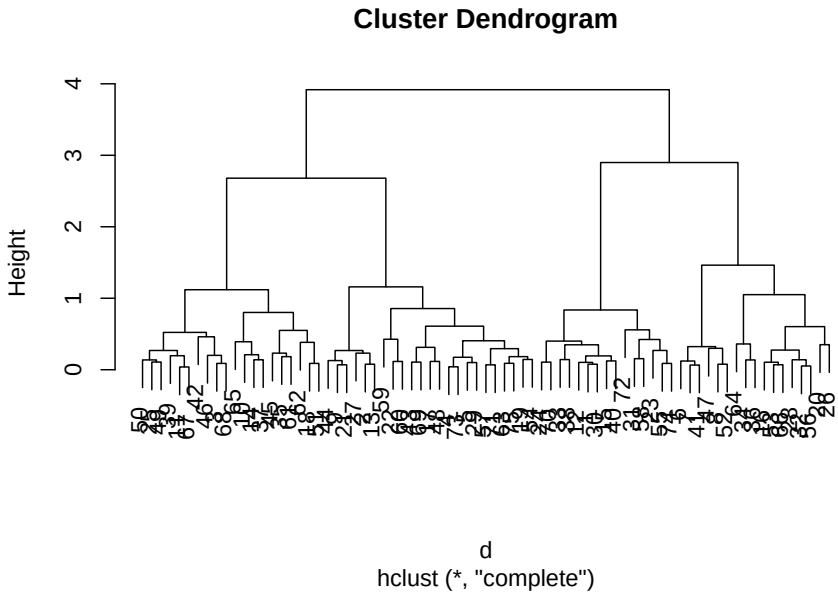
`hclust()` implements agglomerative hierarchical clustering[6]. We cluster using complete link.

```
hc <- hclust(d, method = "complete")
```

Hierarchical clustering does not return cluster assignments but a dendrogram. The standard plot function plots the dendrogram.

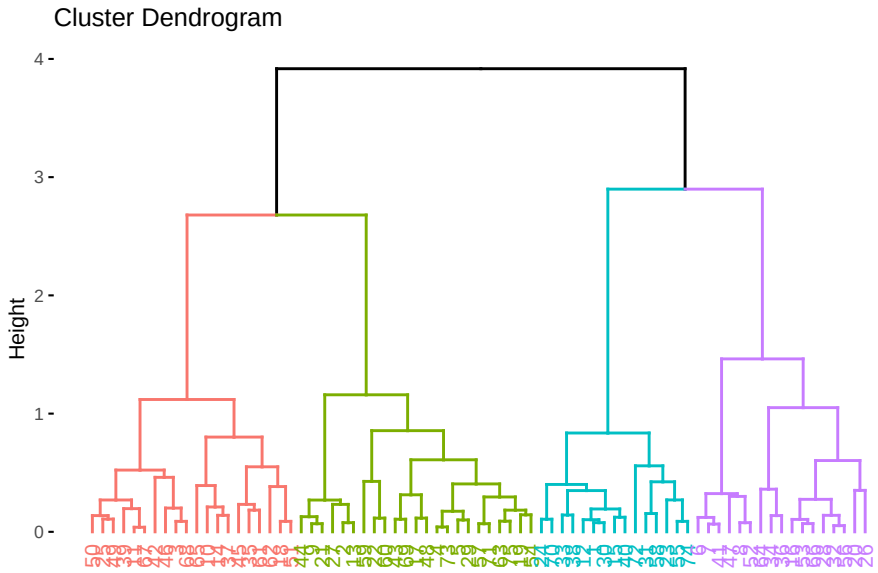
[6] https://en.wikipedia.org/wiki/Hierarchical_clustering

```
plot(hc)
```



Use `factoextra` (ggplot version). We can specify the number of clusters to visualize how the dendrogram will be cut into clusters.

```
fviz_dend(hc, k = 4)
```



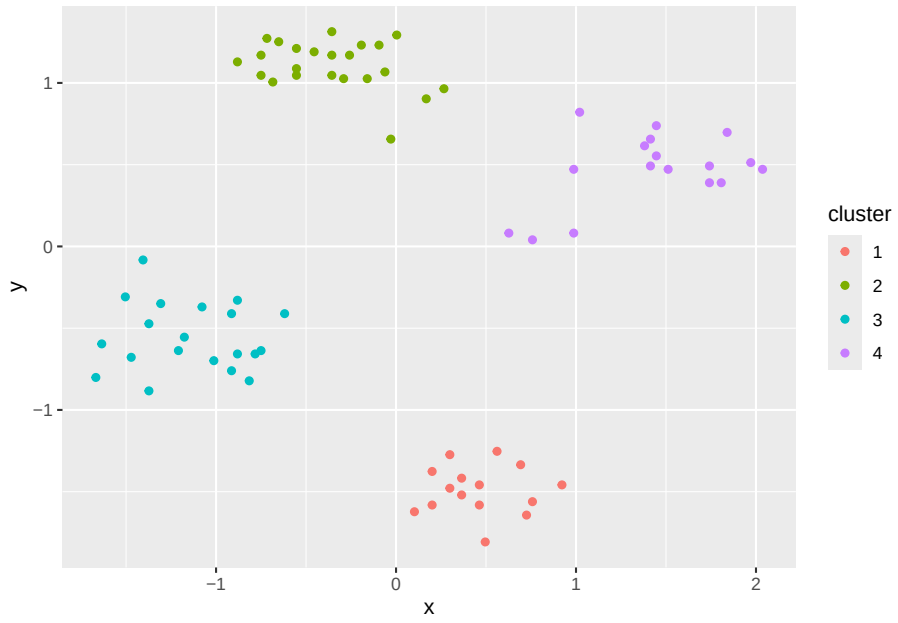
More plotting options for dendrograms, including plotting parts of large dendrograms can be found here.[7]

Extract cluster assignments by cutting the dendrogram into four parts and add the cluster id to the data.

```
clusters <- cutree(hc, k = 4)
cluster_complete <- ruspini_scaled |>
  add_column(cluster = factor(clusters))
cluster_complete
## # A tibble: 75 x 3
##         x      y cluster
##     <dbl>  <dbl> <fct>
##  1  0.365 -1.52  1
##  2 -0.750  1.05  2
##  3 -0.881 -0.329 3
##  4 -0.553  1.05  2
##  5 -0.455  1.19  2
##  6  1.74   0.492 4
##  7  1.81   0.390 4
##  8  1.97   0.513 4
##  9 -0.652  1.25  2
## 10 -1.50  -0.309 3
## # i 65 more rows
```

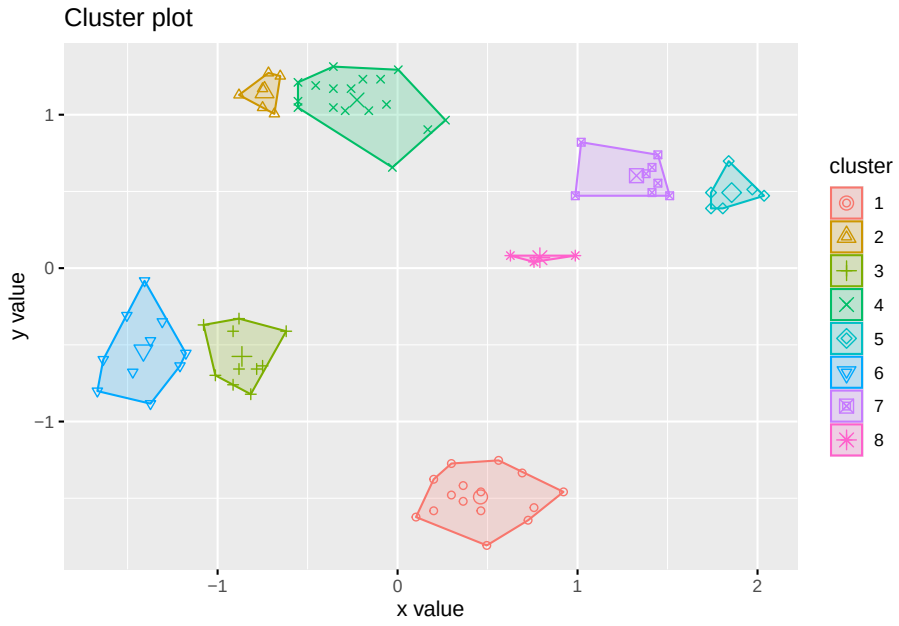
[7] https://rpubs.com/gaston/dendrograms

```
ggplot(cluster_complete, aes(x, y, color = cluster)) +
  geom_point()
```



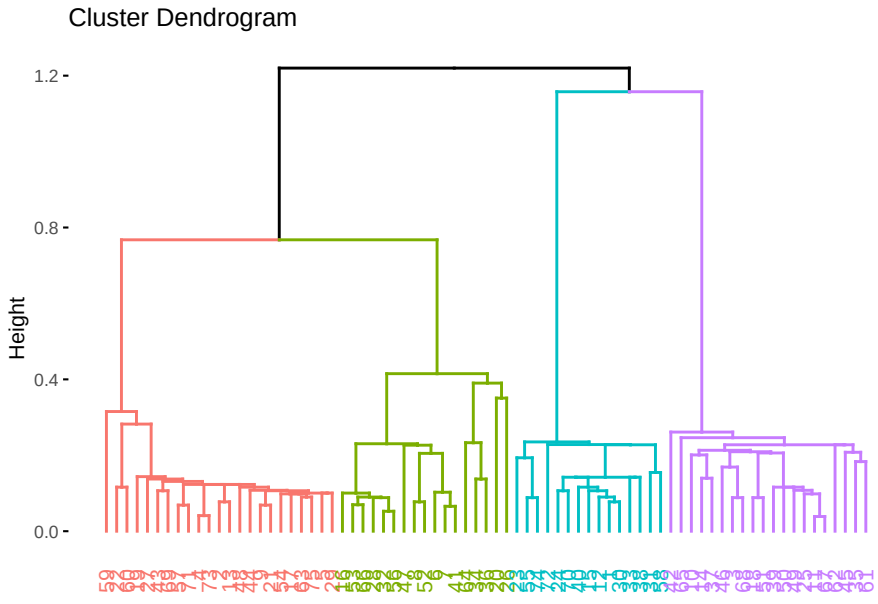
Try 8 clusters (Note: `fviz_cluster` needs a list with data and the cluster labels for hclust)

```
fviz_cluster(list(data = ruspini_scaled,
                  cluster = cutree(hc, k = 8)),
             geom = "point")
```



Clustering with single link

```
hc_single <- hclust(d, method = "single")
fviz_dend(hc_single, k = 4)
```



```
fviz_cluster(list(data = ruspini_scaled,
                  cluster = cutree(hc_single, k = 4)),
             geom = "point")
```

7.4 DBSCAN

```
library(dbscan)
##
## Attaching package: 'dbscan'
## The following object is masked from 'package:stats':
##
##     as.dendrogram
```

DBSCAN[8] stands for "Density-Based Spatial Clustering of Applications with Noise." It groups together points that are closely packed together and treats points in low-density regions as outliers.

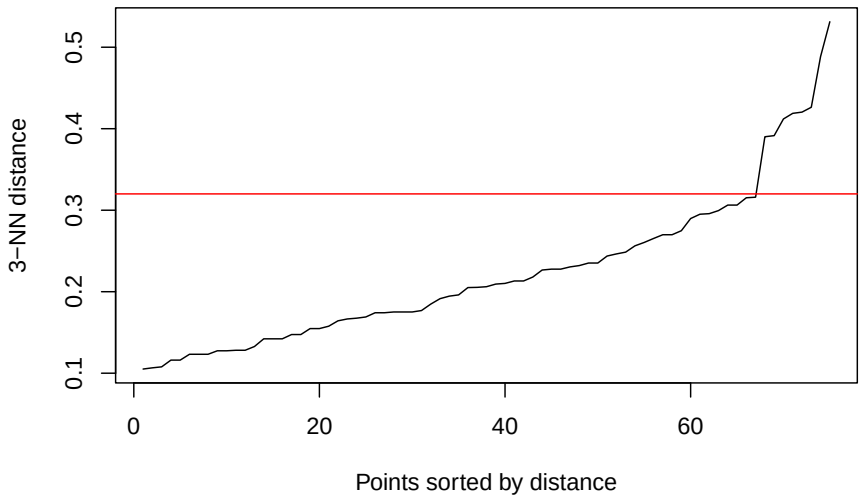
Parameters: minPts defines how many points in the epsilon neighborhood are needed to make a point a core point. It is often chosen as a smoothing parameter. I use here minPts = 4.

To decide on epsilon, the knee in the kNN distance plot is often used. Note that minPts contains the point itself, while the k-nearest neighbor does not. We

[8] https://en.wikipedia.org/wiki/DBSCAN

therefore have to use k = minPts - 1! The knee is around eps = .32.

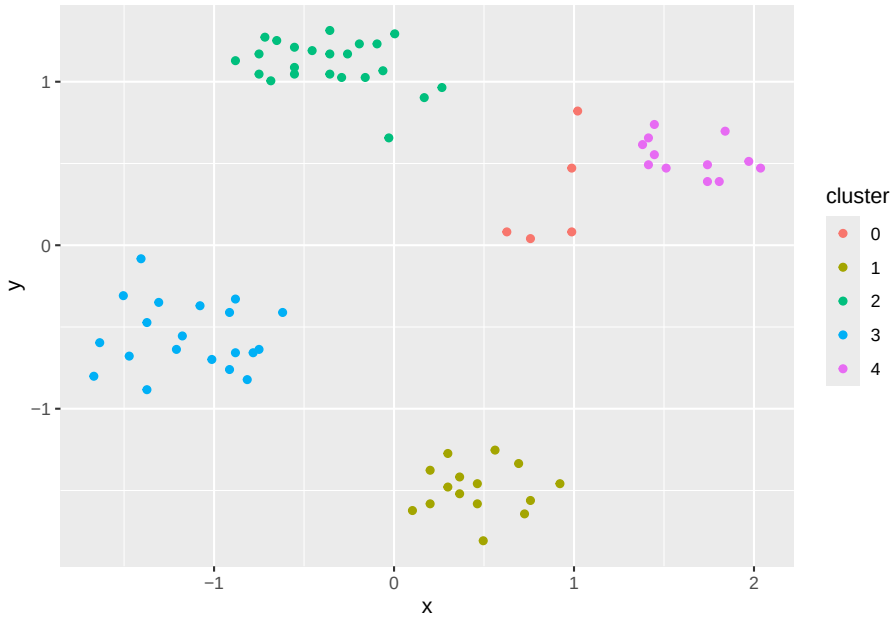
```
kNNdistplot(ruspini_scaled, k = 3)
abline(h = .32, col = "red")
```



run dbscan

```
db <- dbscan(ruspini_scaled, eps = .32, minPts = 4)
db
## DBSCAN clustering for 75 objects.
## Parameters: eps = 0.32, minPts = 4
## Using euclidean distances and borderpoints = TRUE
## The clustering contains 4 cluster(s) and 5 noise points.
##
##  0  1  2  3  4
##  5 15 23 20 12
##
## Available fields: cluster, eps, minPts, metric,
##                   borderPoints
str(db)
## List of 5
##  $ cluster     : int [1:75] 1 2 3 2 2 4 4 4 2 3 ...
##  $ eps         : num 0.32
##  $ minPts      : num 4
##  $ metric      : chr "euclidean"
##  $ borderPoints: logi TRUE
```

```
##  - attr(*, "class")= chr [1:2] "dbscan_fast" "dbscan"
ggplot(ruspini_scaled |> add_column(cluster = factor(db$cluster)),
  aes(x, y, color = cluster)) + geom_point()
```



Note: Cluster 0 represents outliers).

```
fviz_cluster(db, ruspini_scaled, geom = "point")
```

Play with eps (neighborhood size) and MinPts (minimum of points needed for core cluster)

7.5 Cluster Evaluation

7.5.1 Unsupervized Cluster Evaluation

The two most popular quality metrics are the within-cluster sum of squares (WCSS) used as the optimization objective by k-means[9] and the average silhouette width[10]. Look at `within.cluster.ss` and `avg.silwidth` below.

```
##library(fpc)
```

Notes:

- I do not load fpc since the NAMESPACE overwrites dbscan.
- The clustering (second argument below) has to be supplied as a vector with numbers (cluster IDs) and cannot be a factor (use `as.integer()` to convert the factor to an ID).

```
fpc::cluster.stats(d, km$cluster)
## $n
## [1] 75
##
```

[9] https://en.wikipedia.org/wiki/K-means_clustering
[10] https://en.wikipedia.org/wiki/Silhouette_(clustering)

```
## $cluster.number
## [1] 4
##
## $cluster.size
## [1] 15 20 23 17
##
## $min.cluster.size
## [1] 15
##
## $noisen
## [1] 0
##
## $diameter
## [1] 0.8359 1.1193 1.1591 1.4627
##
## $average.distance
## [1] 0.3564 0.4824 0.4286 0.5806
##
## $median.distance
## [1] 0.3380 0.4492 0.3934 0.5024
##
## $separation
## [1] 1.1577 1.1577 0.7676 0.7676
##
## $average.toother
## [1] 2.308 2.157 2.149 2.293
##
## $separation.matrix
##       [,1]  [,2]   [,3]   [,4]
## [1,] 0.000 1.158 1.9577 1.3084
## [2,] 1.158 0.000 1.2199 1.3397
## [3,] 1.958 1.220 0.0000 0.7676
## [4,] 1.308 1.340 0.7676 0.0000
##
## $ave.between.matrix
##       [,1]  [,2]  [,3]  [,4]
## [1,] 0.000 1.874 2.750 2.220
## [2,] 1.874 0.000 1.887 2.772
## [3,] 2.750 1.887 0.000 1.925
## [4,] 2.220 2.772 1.925 0.000
##
## $average.between
## [1] 2.219
##
## $average.within
```

```
## [1] 0.463
##
## $n.between
## [1] 2091
##
## $n.within
## [1] 684
##
## $max.diameter
## [1] 1.463
##
## $min.separation
## [1] 0.7676
##
## $within.cluster.ss
## [1] 10.09
##
## $clus.avg.silwidths
##      1      2      3      4
## 0.8074 0.7211 0.7455 0.6813
##
## $avg.silwidth
## [1] 0.7368
##
## $g2
## NULL
##
## $g3
## NULL
##
## $pearsongamma
## [1] 0.8416
##
## $dunn
## [1] 0.5248
##
## $dunn2
## [1] 3.228
##
## $entropy
## [1] 1.373
##
## $wb.ratio
## [1] 0.2086
##
```

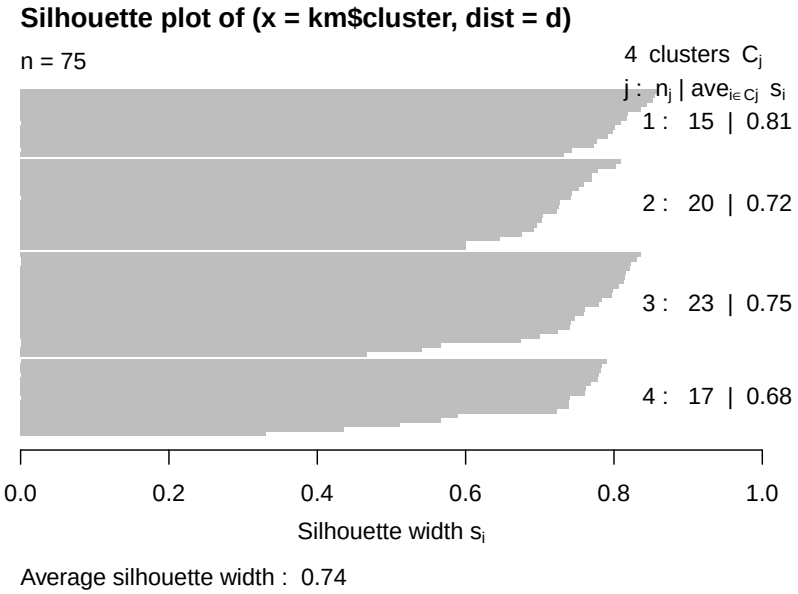
```
## $ch
## [1] 323.6
##
## $cwidegap
## [1] 0.2352 0.2612 0.3153 0.4150
##
## $widestgap
## [1] 0.415
##
## $sindex
## [1] 0.8583
##
## $corrected.rand
## NULL
##
## $vi
## NULL
```

Read `? cluster.stats` for an explanation of all the available indices.

```
sapply(
  list(
    km = km$cluster,
    hc_compl = cutree(hc, k = 4),
    hc_single = cutree(hc_single, k = 4)
  ),
  FUN = function(x)
    fpc::cluster.stats(d, x))[c("within.cluster.ss", "avg.silwidth"), ]
##                   km     hc_compl hc_single
## within.cluster.ss 10.09  10.09    10.09
## avg.silwidth      0.7368 0.7368   0.7368
```

Next, we look at the silhouette using a silhouette plot.

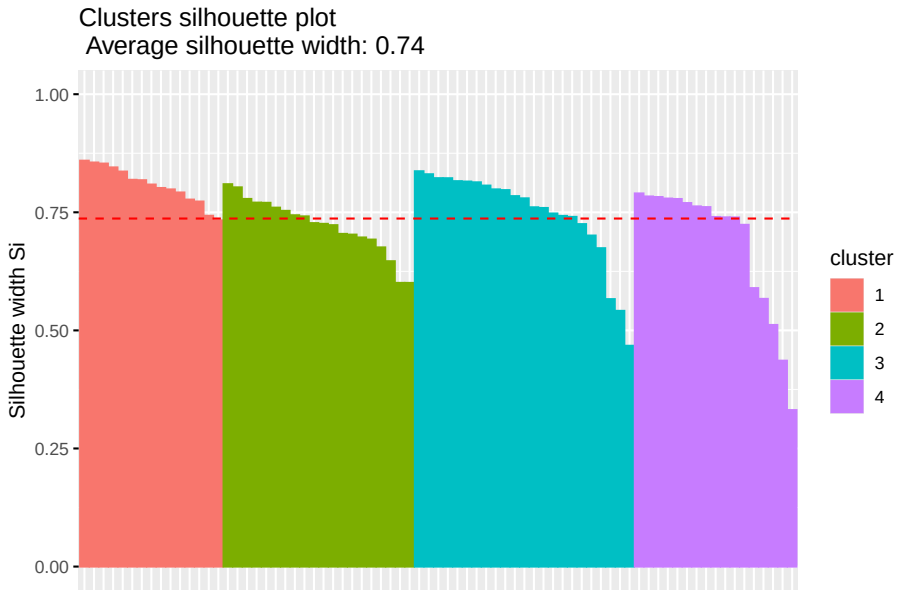
```
library(cluster)
##
## Attaching package: 'cluster'
## The following object is masked _by_ '.GlobalEnv':
##
##     ruspini
plot(silhouette(km$cluster, d))
```



Note: The silhouette plot does not show correctly in R Studio if you have too many objects (bars are missing). I will work when you open a new plotting device with `windows()`, `x11()` or `quartz()`.

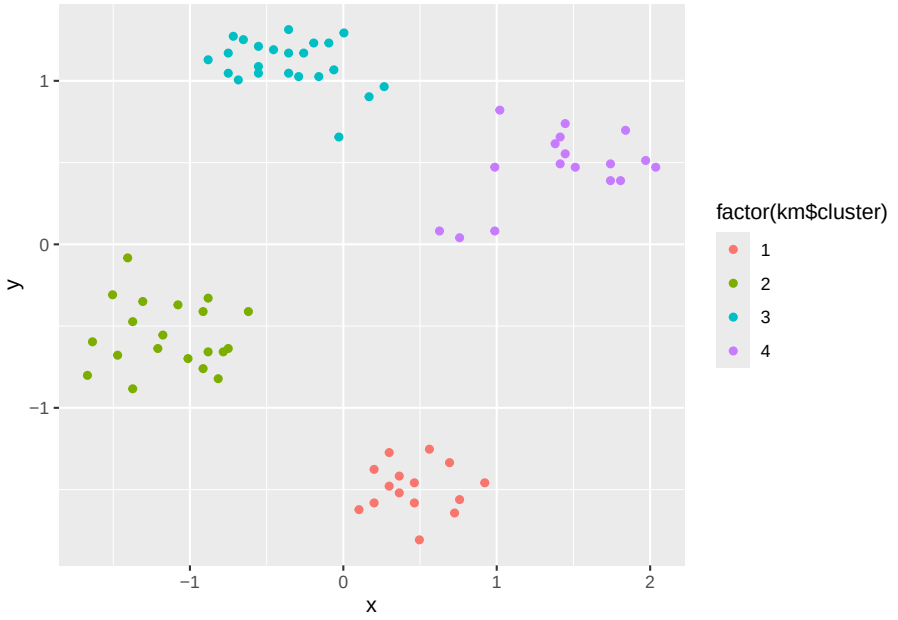
ggplot visualization using `factoextra`

```
fviz_silhouette(silhouette(km$cluster, d))
##   cluster size ave.sil.width
## 1       1   15          0.81
## 2       2   20          0.72
## 3       3   23          0.75
## 4       4   17          0.68
```




```
ggplot(ruspini_scaled,
       aes(x, y, color = factor(km$cluster))) +
  geom_point()
```

7.5.2 Unsupervised Cluster Evaluation using the Proximity Matrix



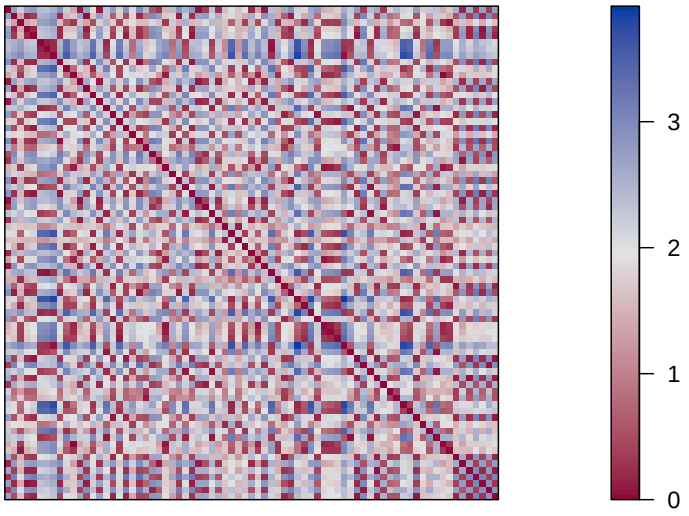

```
d <- dist(ruspini_scaled)
```

Inspect the distance matrix between the first 5 objects.

```
as.matrix(d)[1:5, 1:5]
##       1      2     3      4      5
## 1 0.000 2.7982 1.723 2.7258 2.8315
## 2 2.798 0.0000 1.382 0.1967 0.3282
## 3 1.723 1.3819 0.000 1.4142 1.5781
## 4 2.726 0.1967 1.414 0.0000 0.1742
## 5 2.832 0.3282 1.578 0.1742 0.0000
```

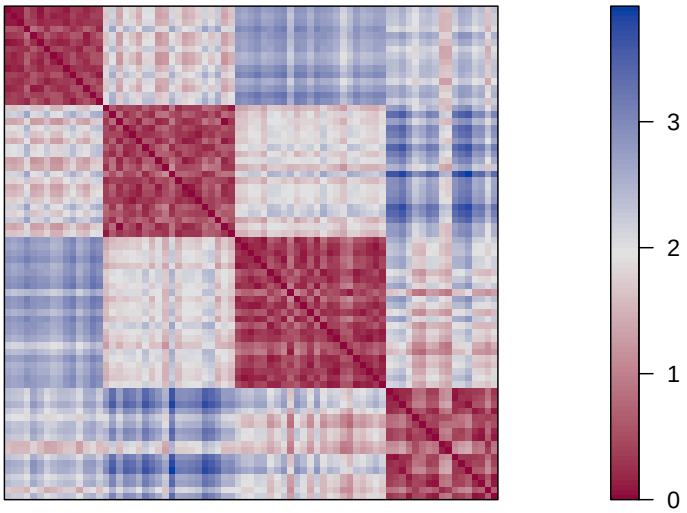
A false-color image visualizes each value in the matrix as a pixel with the color representing the value.

```
library(seriation)
pimage(d, col = bluered(100))
```



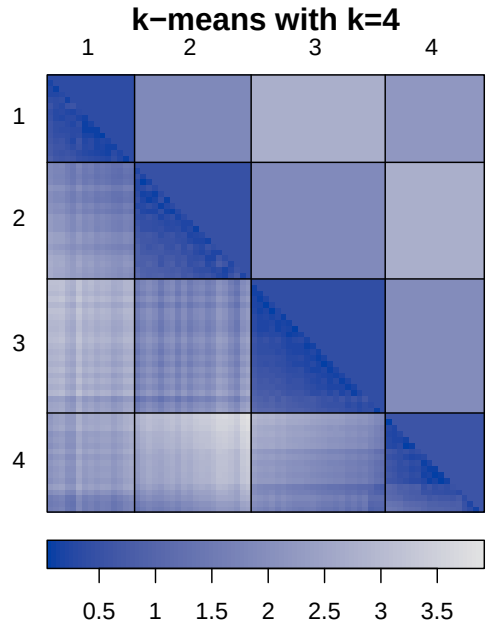
Rows and columns are the objects as they are ordered in the data set. The diagonal represents the distance between an object and itself and has by definition a distance of 0 (dark line). Visualizing the unordered distance matrix does not show much structure, but we can reorder the matrix (rows and columns) using the k-means cluster labels from cluster 1 to 4. A clear block structure representing the clusters becomes visible.

```
pimage(d, order=order(km$cluster), col = bluered(100))
```



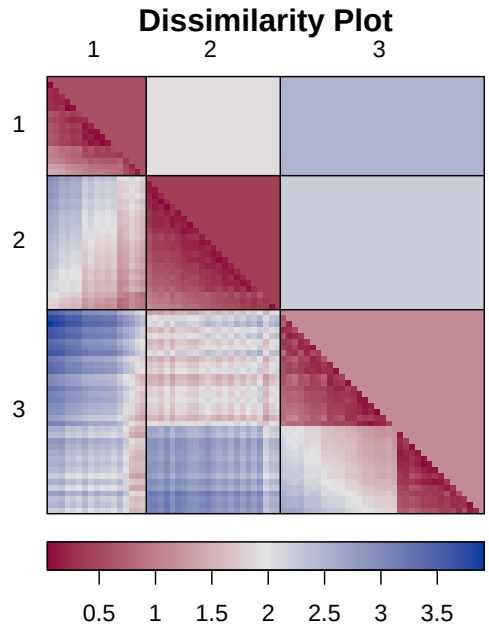
Plot function `dissplot` in package **seriation** rearranges the matrix and adds lines and cluster labels. In the lower half of the plot, it shows average dissimilarities between clusters. The function organizes the objects by cluster and then reorders clusters and objects within clusters so that more similar objects are closer together.

```
dissplot(d, labels = km$cluster,
         options = list(main = "k-means with k=4"))
```

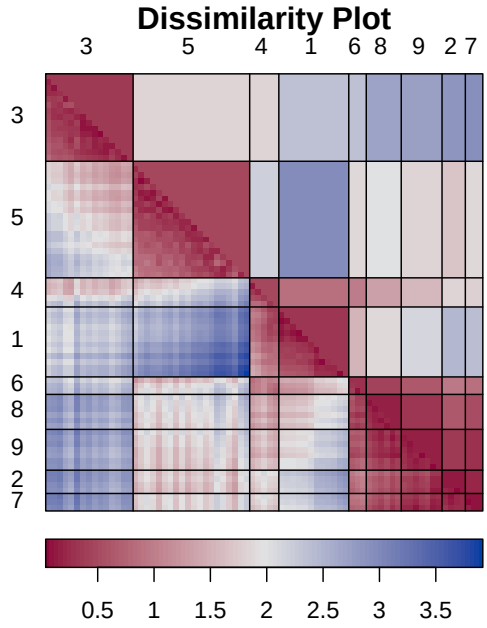


The reordering by `dissplot` makes the misspecification of k visible as blocks.

```
dissplot(d,
         labels = kmeans(ruspini_scaled, centers = 3)$cluster,
         col = bluered(100))
```

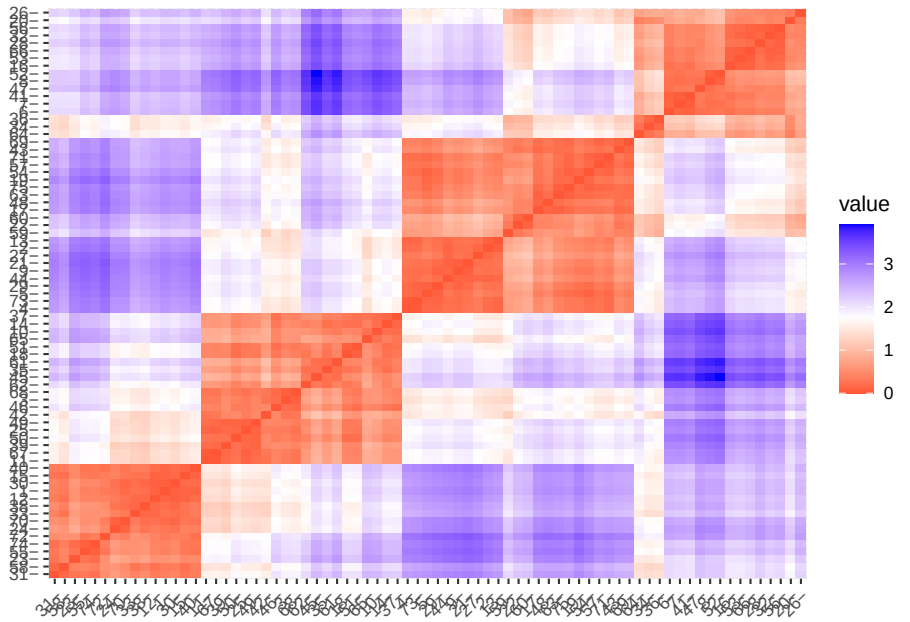


```
dissplot(d,
         labels = kmeans(ruspini_scaled, centers = 9)$cluster,
         col = bluered(100))
```



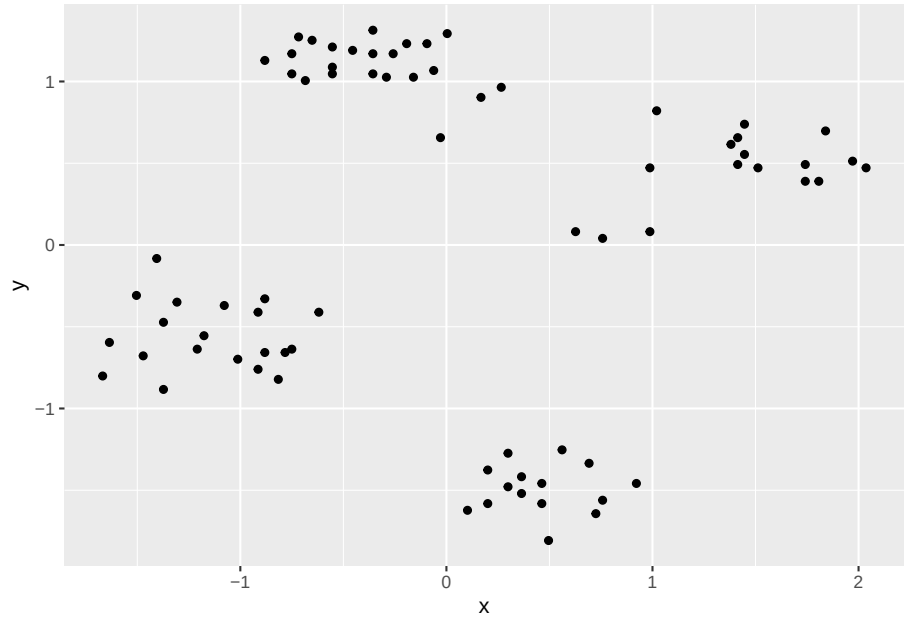
Using `factoextra`

```
fviz_dist(d)
```



7.5.3 Determining the Correct Number of Clusters

```
ggplot(ruspini_scaled, aes(x, y)) + geom_point()
```

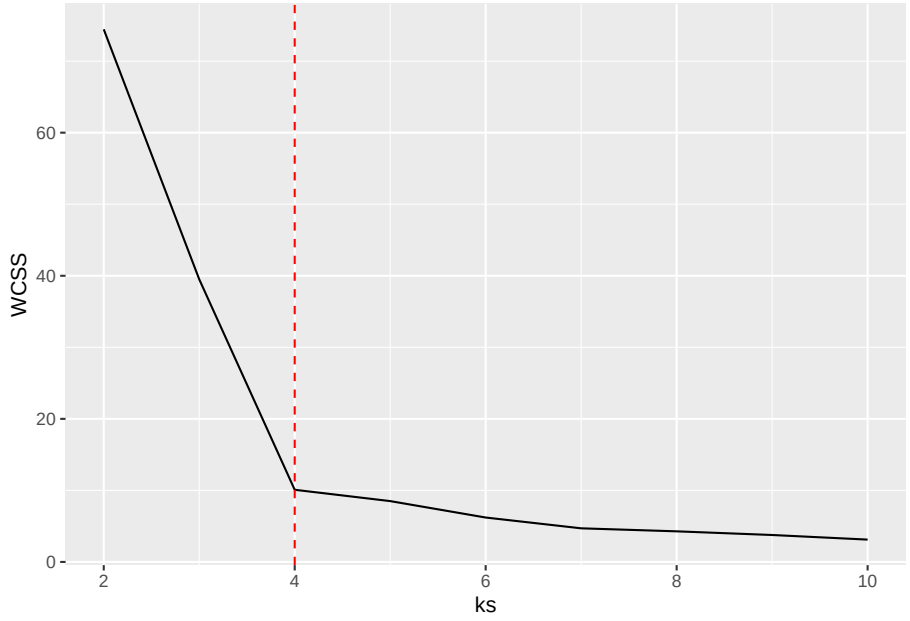


```
## We will use different methods and try 1-10 clusters.
set.seed(1234)
ks <- 2:10
```

7.5.3.1 Elbow Method: Within-Cluster Sum of Squares

Calculate the within-cluster sum of squares for different numbers of clusters and look for the knee or elbow[11] in the plot. (`nstart = 5` just repeats k-means 5 times and returns the best solution)

```
WCSS <- sapply(ks, FUN = function(k) {
  kmeans(ruspini_scaled, centers = k, nstart = 5)$tot.withinss
  })

ggplot(tibble(ks, WCSS), aes(ks, WCSS)) +
  geom_line() +
  geom_vline(xintercept = 4, color = "red", linetype = 2)
```

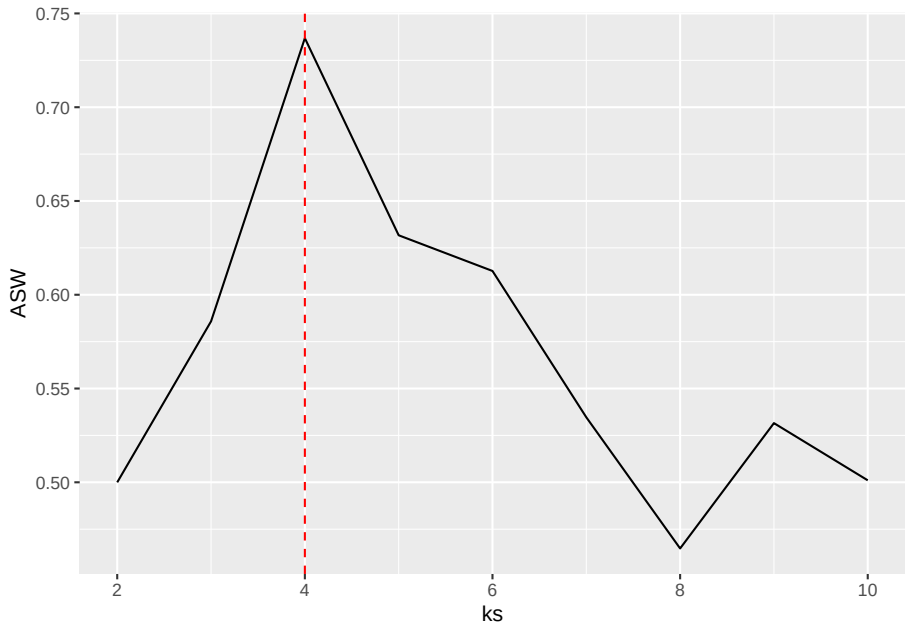


7.5.3.2 Average Silhouette Width

Plot the average silhouette width for different number of clusters and look for the maximum in the plot.

[11] https://en.wikipedia.org/wiki/Elbow_method_(clustering)

```
ASW <- sapply(ks, FUN=function(k) {
  fpc::cluster.stats(d,
                     kmeans(ruspini_scaled,
                            centers = k,
                            nstart = 5)$cluster)$avg.silwidth
  })

best_k <- ks[which.max(ASW)]
best_k
## [1] 4
ggplot(tibble(ks, ASW), aes(ks, ASW)) +
  geom_line() +
  geom_vline(xintercept = best_k, color = "red", linetype = 2)
```



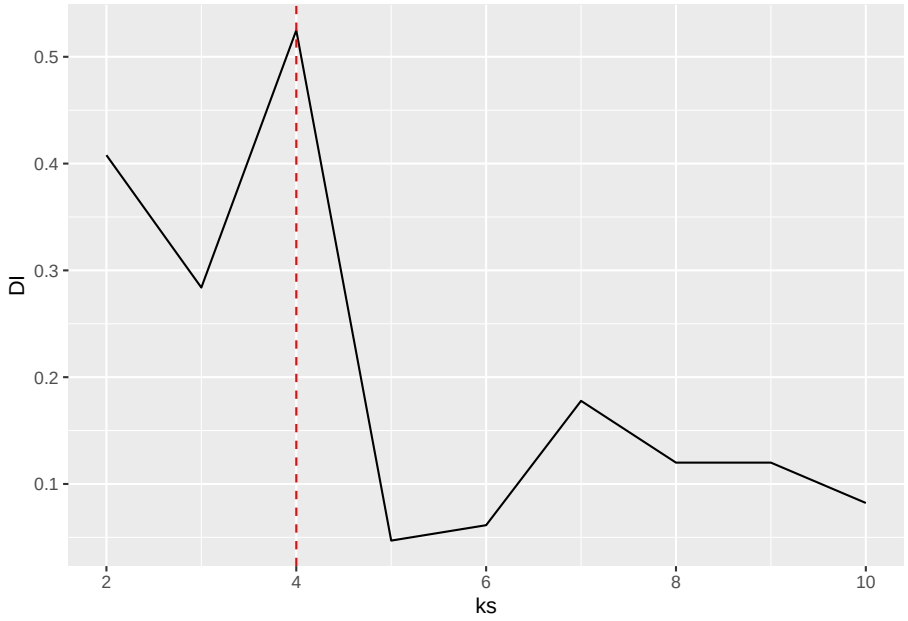
7.5.3.3 Dunn Index

Use Dunn index[12] (another internal measure given by min. separation/ max. diameter)

```
DI <- sapply(ks, FUN = function(k) {
  fpc::cluster.stats(d,
                     kmeans(ruspini_scaled, centers = k,
                            nstart = 5)$cluster)$dunn
})
```

[12] https://en.wikipedia.org/wiki/Dunn_index

```
best_k <- ks[which.max(DI)]
ggplot(tibble(ks, DI), aes(ks, DI)) +
  geom_line() +
  geom_vline(xintercept = best_k, color = "red", linetype = 2)
```

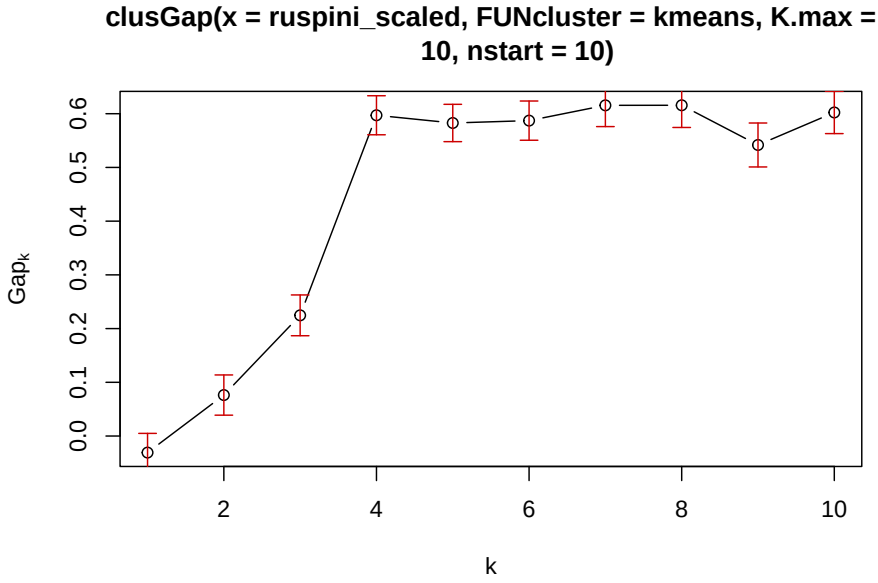


7.5.3.4 Gap Statistic

Compares the change in within-cluster dispersion with that expected from a null model (see `? clusGap`). The default method is to choose the smallest k such that its value Gap(k) is not more than 1 standard error away from the first local maximum.

```
library(cluster)
k <- clusGap(ruspini_scaled,
             FUN = kmeans,
             nstart = 10,
             K.max = 10)
k
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = ruspini_scaled, FUNcluster = kmeans, K.max = 10, nstart = 10)
## B=100 simulated reference sets, k = 1..10; spaceH0="scaledPCA"
##  --> Number of clusters (method 'firstSEmax', SE.factor=1): 4
##        logW E.logW      gap  SE.sim
##  [1,] 3.498  3.467 -0.03080 0.03570
##  [2,] 3.074  3.150  0.07625 0.03744
```

```
##  [3,] 2.678  2.903  0.22466 0.03796
##  [4,] 2.106  2.704  0.59710 0.03633
##  [5,] 1.987  2.570  0.58271 0.03474
##  [6,] 1.864  2.451  0.58713 0.03651
##  [7,] 1.732  2.348  0.61558 0.03954
##  [8,] 1.640  2.256  0.61570 0.04132
##  [9,] 1.630  2.172  0.54177 0.04090
## [10,] 1.491  2.093  0.60230 0.03934
plot(k)
```



Note: these methods can also be used for hierarchical clustering.

There have been many other methods and indices proposed to determine the number of clusters. See, e.g., package NbClust[13].

7.5.4 Clustering Tendency

Most clustering algorithms will always produce a clustering, even if the data does not contain a cluster structure. It is typically good to check cluster tendency before attempting to cluster the data.

We use again the smiley data.

```
library(mlbench)
shapes <- mlbench.smiley(n = 500, sd1 = 0.1, sd2 = 0.05)$x
```

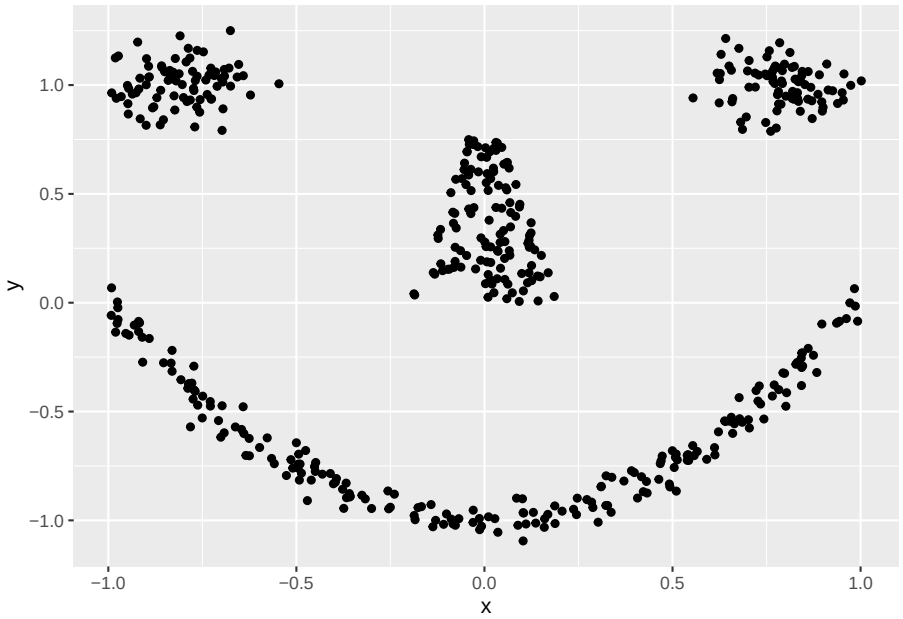
[13] https://cran.r-project.org/package=NbClust

```
colnames(shapes) <- c("x", "y")
shapes <- as_tibble(shapes)
```

7.5.4.1 Scatter plots

The first step is visual inspection using scatter plots.

```
ggplot(shapes, aes(x = x, y = y)) + geom_point()
```



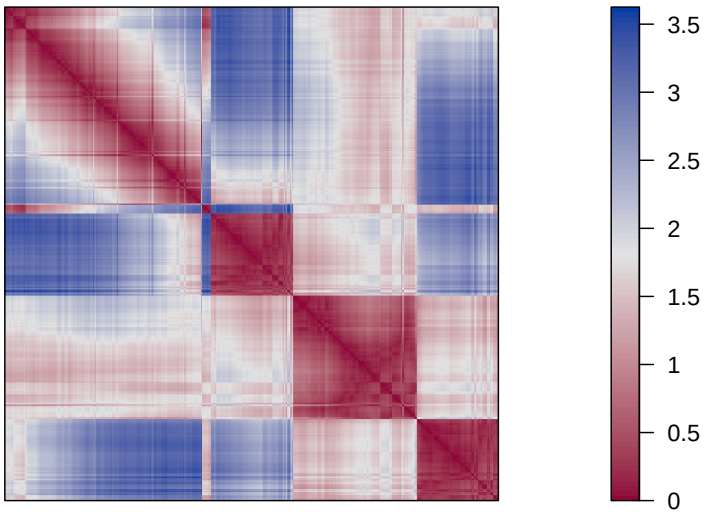
Cluster tendency is typically indicated by several separated point clouds. Often an appropriate number of clusters can also be visually obtained by counting the number of point clouds. We see four clusters, but the mouth is not convex/spherical and thus will pose a problems to algorithms like k-means.

If the data has more than two features then you can use a pairs plot (scatterplot matrix) or look at a scatterplot of the first two principal components using PCA.

7.5.4.2 Visual Analysis for Cluster Tendency Assessment (VAT)

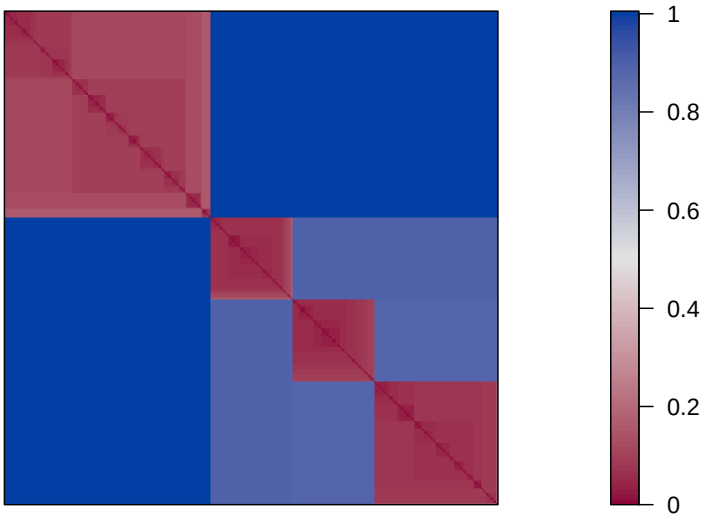
VAT reorders the objects to show potential clustering tendency as a block structure (dark blocks along the main diagonal). We scale the data before using Euclidean distance.

```
library(seriation)

d_shapes <- dist(scale(shapes))
VAT(d_shapes, col = bluered(100))
```



iVAT uses the largest distances for all possible paths between two objects instead of the direct distances to make the block structure better visible.

```
iVAT(d_shapes, col = bluered(100))
```



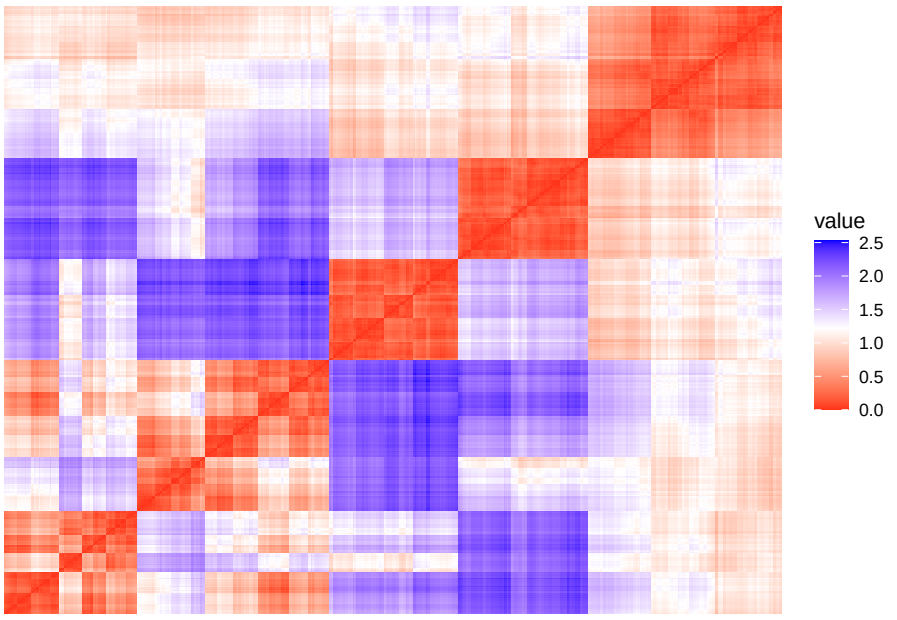
7.5.4.3 Hopkins statistic

`factoextra` can also create a VAT plot and calculate the Hopkins statistic[14] to assess clustering tendency. For the Hopkins statistic, a sample of size n is drawn from the data and then compares the nearest neighbor distribution with a simulated dataset drawn from a random uniform distribution (see detailed explanation[15]). A values $>.5$ indicates usually a clustering tendency.

```
get_clust_tendency(shapes, n = 10)
## $hopkins_stat
## [1] 0.9074
##
## $plot
```

[14] https://en.wikipedia.org/wiki/Hopkins_statistic

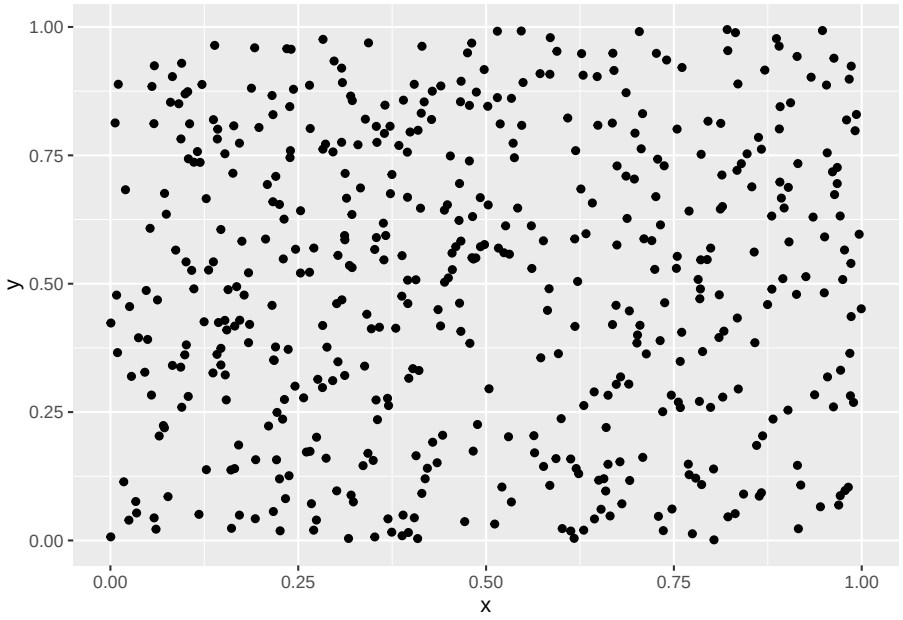
[15] https://www.datanovia.com/en/lessons/assessing-clustering-tendency/#statistical-methods



Both plots show a strong cluster structure with 4 clusters.

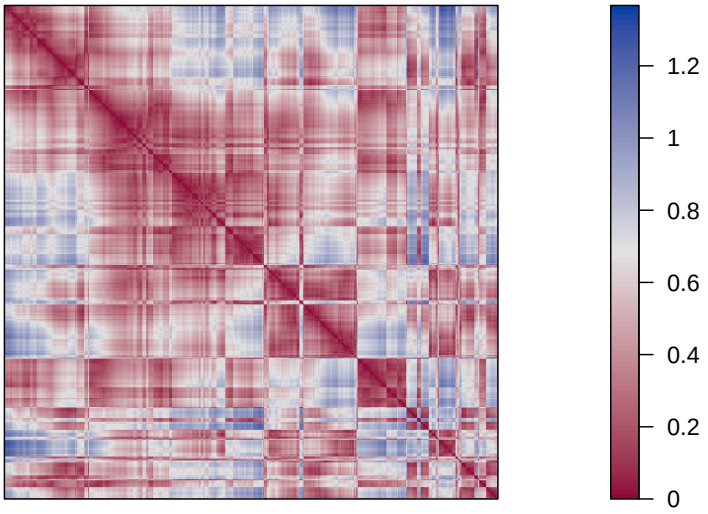
```
data_random <- tibble(x = runif(500), y = runif(500))
ggplot(data_random, aes(x, y)) + geom_point()
```

7.5.4.4 Data Without Clustering Tendency

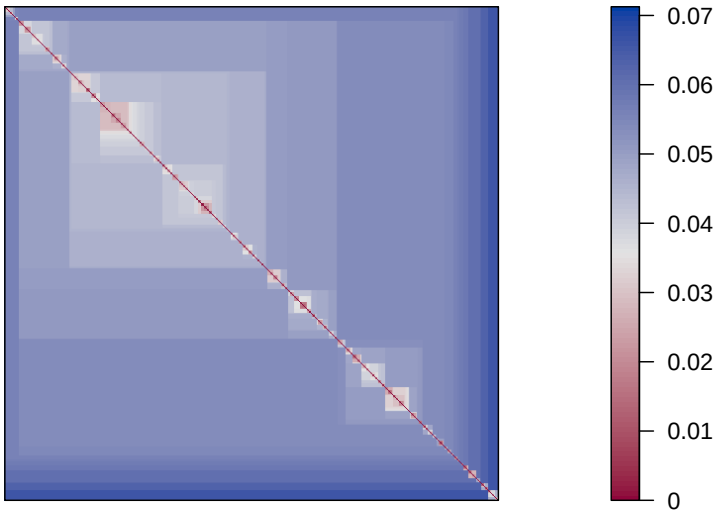


No point clouds are visible, just noise.

```
d_random <- dist(data_random)
VAT(d_random, col = bluered(100))
```



```
iVAT(d_random, col = bluered(100))
```



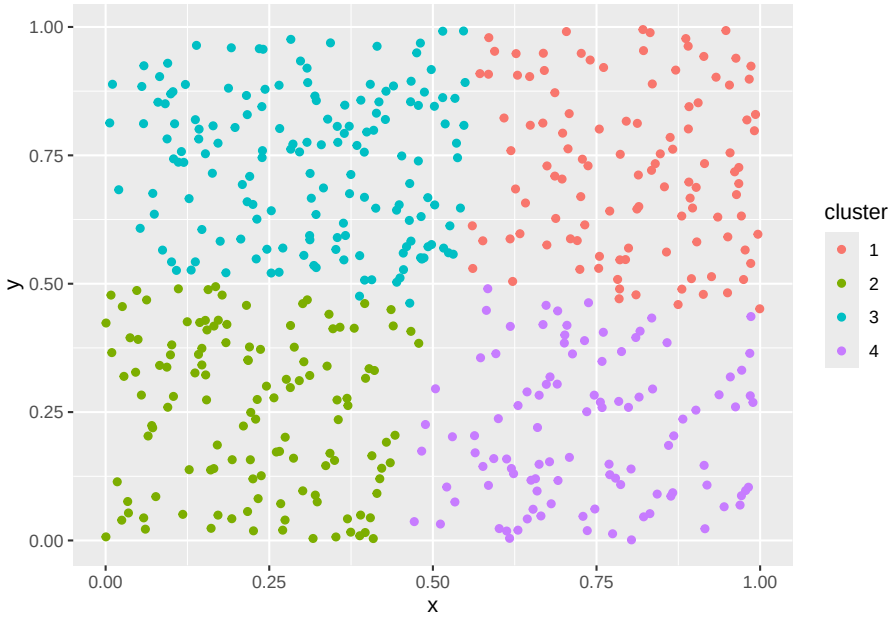

```
get_clust_tendency(data_random, n = 10, graph = FALSE)
## $hopkins_stat
## [1] 0.4642
##
## $plot
## NULL
```

There is very little clustering structure visible indicating low clustering tendency and clustering should not be performed on this data. However, k-means can be used to partition the data into k regions of roughly equivalent size. This can be used as a data-driven discretization of the space.

7.5.4.5 k-means on Data Without Clustering Tendency

What happens if we perform k-means on data that has no inherent clustering structure?

```
km <- kmeans(data_random, centers = 4)

random_clustered<- data_random |>
  add_column(cluster = factor(km$cluster))
ggplot(random_clustered, aes(x = x, y = y, color = cluster)) +
  geom_point()
```



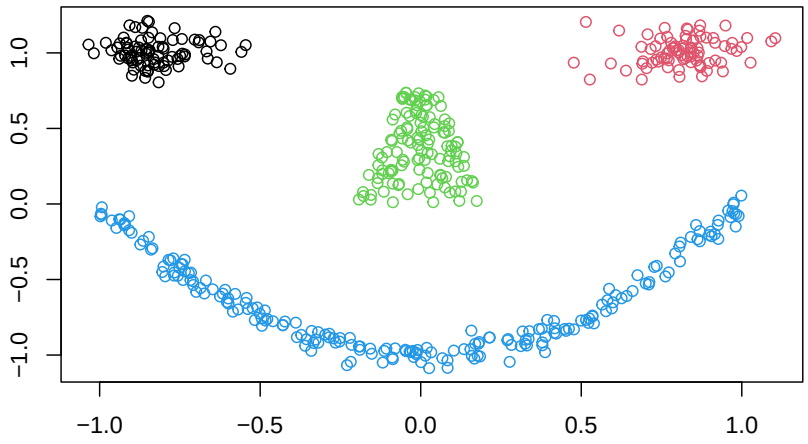
k-means discretizes the space into similarly sized regions.

7.5.5 Supervised Measures of Cluster Validity

Also called external cluster validation since it uses ground truth information. That is, the user has an idea how the data should be grouped. This could be a known class label not provided to the clustering algorithm.

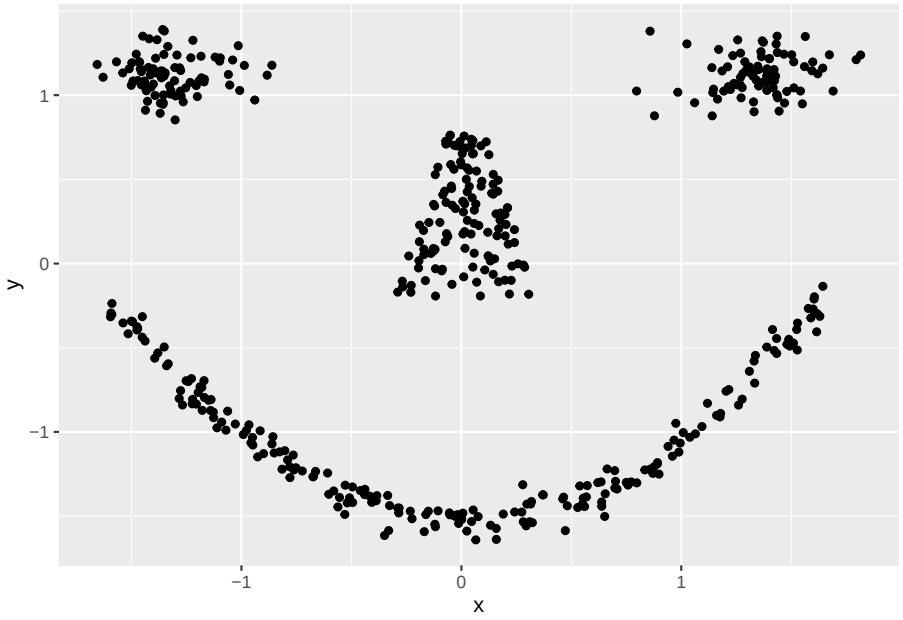
We use an artificial data set with known groups.

```
library(mlbench)
set.seed(1234)
shapes <- mlbench.smiley(n = 500, sd1 = 0.1, sd2 = 0.05)
plot(shapes)
```



Prepare data

```
truth <- as.integer(shapes$class)
shapes <- shapes$x
colnames(shapes) <- c("x", "y")

shapes <- shapes |> scale() |> as_tibble()

ggplot(shapes, aes(x, y)) + geom_point()
```

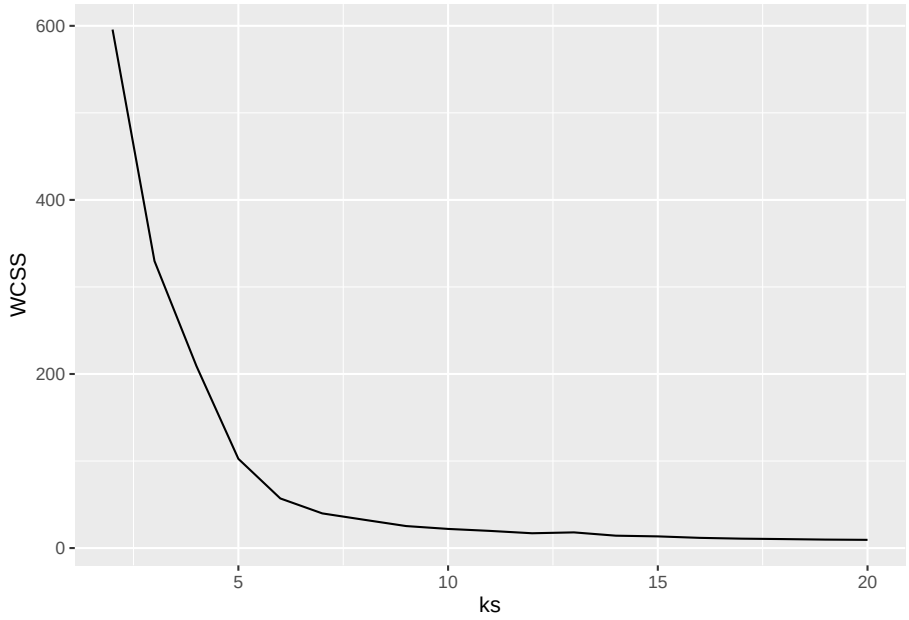


Find optimal number of Clusters for k-means

```
ks <- 2:20
```

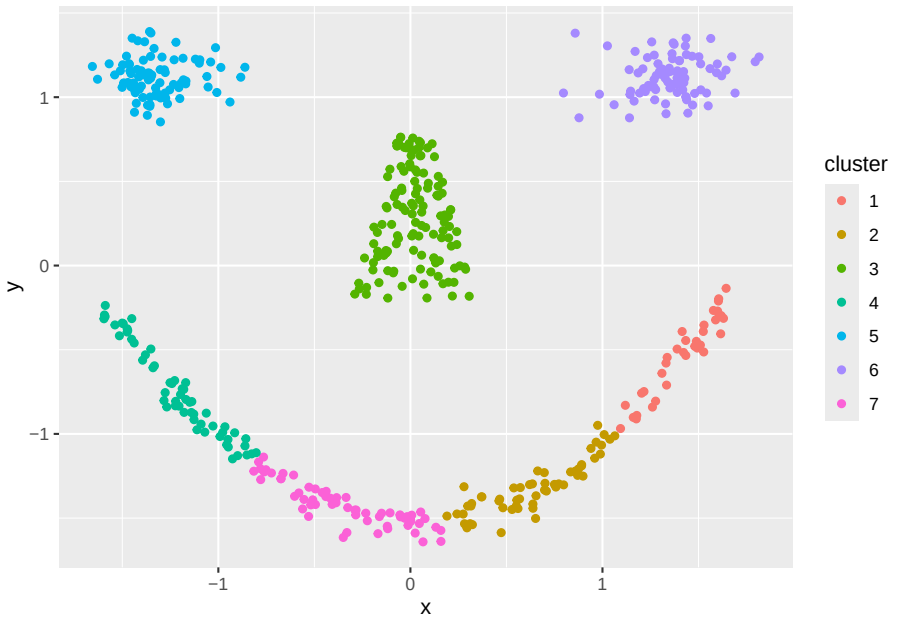
Use within sum of squares (look for the knee)

```
WCSS <- sapply(ks, FUN = function(k) {
  kmeans(shapes, centers = k, nstart = 10)$tot.withinss
})

ggplot(tibble(ks, WCSS), aes(ks, WCSS)) + geom_line()
```



Looks like it could be 7 clusters

```
km <- kmeans(shapes, centers = 7, nstart = 10)

ggplot(shapes |> add_column(cluster = factor(km$cluster)),
       aes(x, y, color = cluster)) +
  geom_point()
```

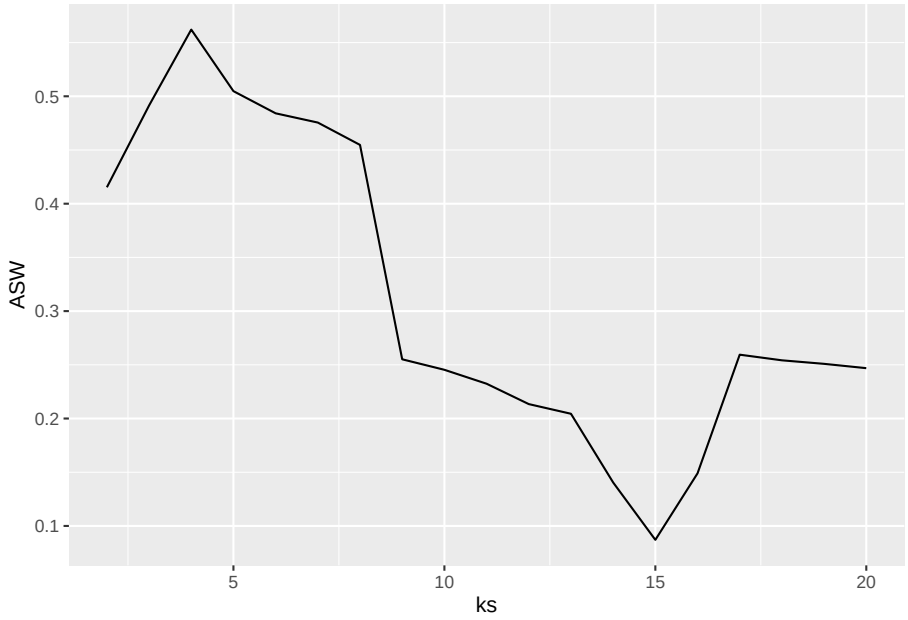


Hierarchical clustering: We use single-link because of the mouth is non-convex and chaining may help.

```
d <- dist(shapes)
hc <- hclust(d, method = "single")
```

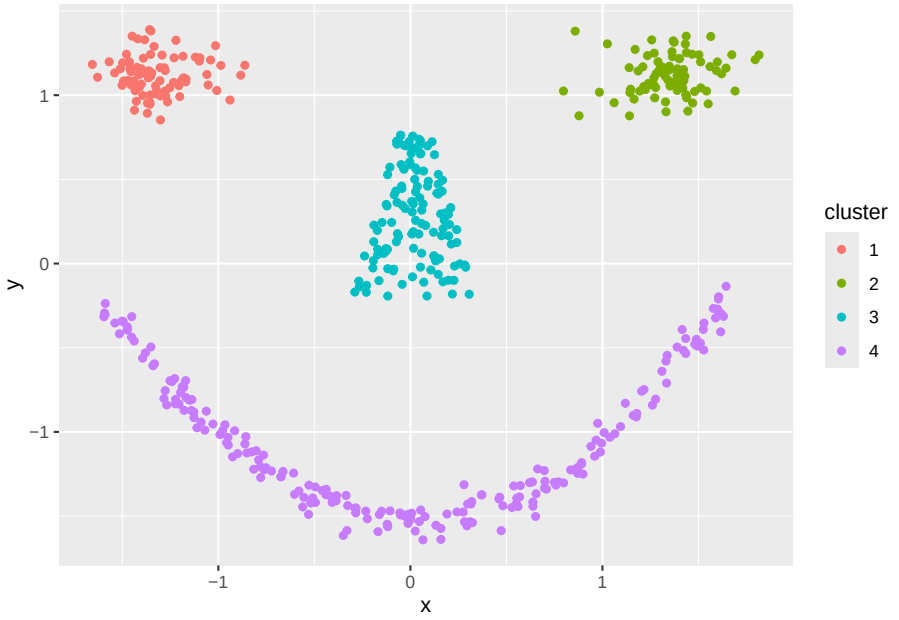
Find optimal number of clusters

```
ASW <- sapply(ks, FUN = function(k) {
  fpc::cluster.stats(d, cutree(hc, k))$avg.silwidth
})

ggplot(tibble(ks, ASW), aes(ks, ASW)) + geom_line()
```



The maximum is clearly at 4 clusters.

```
hc_4 <- cutree(hc, 4)

ggplot(shapes |> add_column(cluster = factor(hc_4)),
       aes(x, y, color = cluster)) +
  geom_point()
```



Compare with ground truth with the corrected (=adjusted) Rand index (ARI)[16], the variation of information (VI) index[17], entropy[18] and purity[19].

`cluster_stats` computes ARI and VI as comparative measures. I define functions for entropy and purity here:

```
entropy <- function(cluster, truth) {
  k <- max(cluster, truth)
  cluster <- factor(cluster, levels = 1:k)
  truth <- factor(truth, levels = 1:k)
  w <- table(cluster)/length(cluster)

  cnts <- sapply(split(truth, cluster), table)
  p <- sweep(cnts, 1, rowSums(cnts), "/")
  p[is.nan(p)] <- 0
  e <- -p * log(p, 2)

  sum(w * rowSums(e, na.rm = TRUE))
}

purity <- function(cluster, truth) {
  k <- max(cluster, truth)
```

[16] https://en.wikipedia.org/wiki/Rand_index#Adjusted_Rand_index
[17] https://en.wikipedia.org/wiki/Variation_of_information
[18] https://en.wikipedia.org/wiki/Entropy_(information_theory)
[19] https://en.wikipedia.org/wiki/Cluster_analysis#External_evaluation

```
  cluster <- factor(cluster, levels = 1:k)
  truth <- factor(truth, levels = 1:k)
  w <- table(cluster)/length(cluster)

  cnts <- sapply(split(truth, cluster), table)
  p <- sweep(cnts, 1, rowSums(cnts), "/")
  p[is.nan(p)] <- 0

  sum(w * apply(p, 1, max))
}
```

calculate measures (for comparison we also use random "clusterings" with 4 and 6 clusters)

```
random_4 <- sample(1:4, nrow(shapes), replace = TRUE)
random_6 <- sample(1:6, nrow(shapes), replace = TRUE)

r <- rbind(
  kmeans_7 = c(
    unlist(fpc::cluster.stats(d, km$cluster,
                              truth, compareonly = TRUE)),
    entropy = entropy(km$cluster, truth),
    purity = purity(km$cluster, truth)
    ),
  hc_4 = c(
    unlist(fpc::cluster.stats(d, hc_4,
                              truth, compareonly = TRUE)),
    entropy = entropy(hc_4, truth),
    purity = purity(hc_4, truth)
    ),
  random_4 = c(
    unlist(fpc::cluster.stats(d, random_4,
                              truth, compareonly = TRUE)),
    entropy = entropy(random_4, truth),
    purity = purity(random_4, truth)
    ),
  random_6 = c(
    unlist(fpc::cluster.stats(d, random_6,
                              truth, compareonly = TRUE)),
    entropy = entropy(random_6, truth),
    purity = purity(random_6, truth)
    )
  )
r
##          corrected.rand     vi entropy purity
## kmeans_7       0.638229 0.5709  0.2286 0.4639
```

```
## hc_4           1.000000 0.0000  0.0000 1.0000
## random_4      -0.003235 2.6832  1.9883 0.2878
## random_6      -0.002125 3.0763  1.7281 0.1436
```

Notes:

- Hierarchical clustering found the perfect clustering.
- Entropy and purity are heavily impacted by the number of clusters (more clusters improve the metric).
- The corrected rand index shows clearly that the random clusterings have no relationship with the ground truth (very close to 0). This is a very helpful property.

Read `? cluster.stats` for an explanation of all the available indices.

7.6 More Clustering Algorithms*

Note: Some of these methods are covered in Chapter 8 of the textbook.

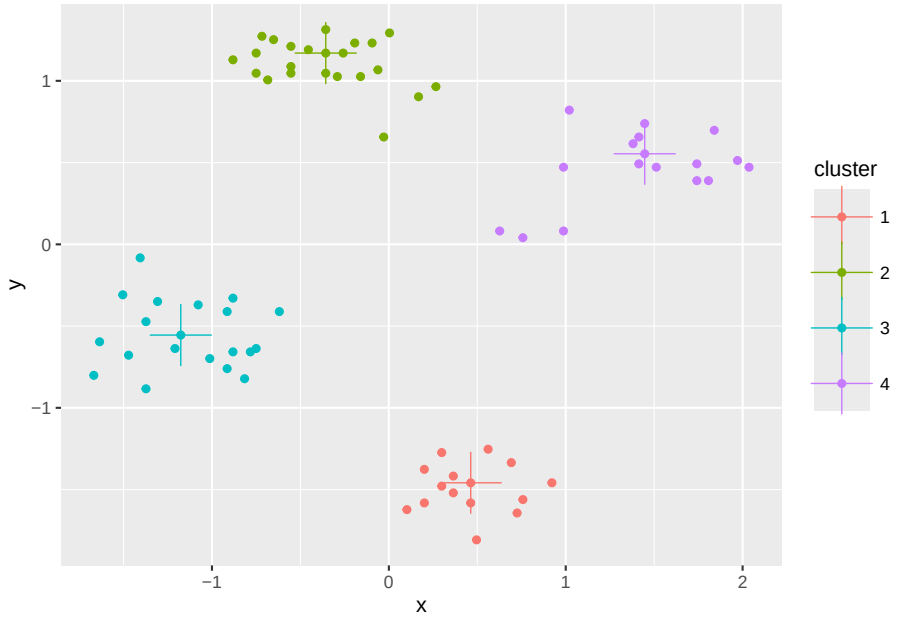
7.6.1 Partitioning Around Medoids (PAM)

PAM[20] tries to solve the k-medoids problem. The problem is similar to k-means, but uses medoids instead of centroids to represent clusters. Like hierarchical clustering, it typically works with precomputed distance matrix. An advantage is that you can use any distance metric not just Euclidean distances. **Note:** The medoid is the most central data point in the middle of the cluster.

```
library(cluster)

d <- dist(ruspini_scaled)
str(d)
##  'dist' num [1:2775] 2.8 1.72 2.73 2.83 2.44 ...
##  - attr(*, "Size")= int 75
##  - attr(*, "Diag")= logi FALSE
##  - attr(*, "Upper")= logi FALSE
##  - attr(*, "method")= chr "Euclidean"
##  - attr(*, "call")= language dist(x = ruspini_scaled)
p <- pam(d, k = 4)
p
## Medoids:
##      ID
## [1,] 15 15
## [2,] 54 54
## [3,] 51 51
## [4,] 66 66
```

[20] https://en.wikipedia.org/wiki/K-medoids

```
## Clustering vector:
##  [1] 1 2 3 2 2 4 4 4 2 3 3 1 2 3 1 4 2 3 2 4 2 2 1 1 3 4 2 4
## [29] 2 1 1 4 1 4 3 4 3 1 3 1 4 3 2 2 3 3 4 2 3 3 3 4 4 2 1 4
## [57] 2 1 2 2 3 3 2 4 3 4 3 3 2 1 2 1 2 1 2
## Objective function:
##  build   swap
## 0.4423 0.3187
##
## Available components:
## [1] "medoids"    "id.med"     "clustering" "objective"
## [5] "isolation"  "clusinfo"   "silinfo"    "diss"
## [9] "call"
ruspini_clustered <- ruspini_scaled |>
  add_column(cluster = factor(p$cluster))

medoids <- as_tibble(ruspini_scaled[p$medoids, ],
                     rownames = "cluster")
medoids
## # A tibble: 4 x 3
##   cluster      x      y
##   <chr>    <dbl>  <dbl>
## 1 1        0.463 -1.46
## 2 2       -0.357  1.17
## 3 3       -1.18  -0.555
## 4 4        1.45   0.554
ggplot(ruspini_clustered, aes(x = x, y = y, color = cluster)) +
  geom_point() +
  geom_point(data = medoids, aes(x = x, y = y, color = cluster),
             shape = 3, size = 10)
```



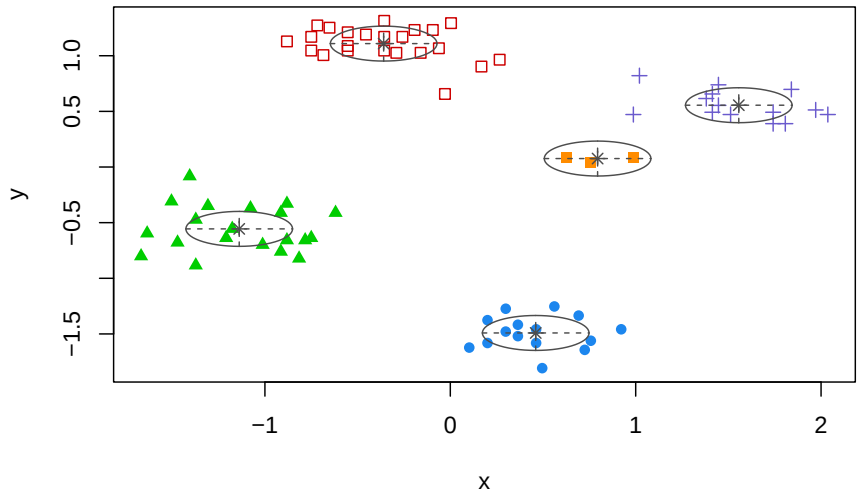

```
## __Note:__ `fviz_cluster` needs the original data.
fviz_cluster(c(p, list(data = ruspini_scaled)),
             geom = "point",
             ellipse.type = "norm")
```

```
library(mclust)
## Package 'mclust' version 6.1.1
## Type 'citation("mclust")' for citing this R package in publications.
##
## Attaching package: 'mclust'
## The following object is masked from 'package:purrr':
##
##     map
```

Gaussian mixture models[21] assume that the data set is the result of drawing data from a set of Gaussian distributions where each distribution represents a cluster. Estimation algorithms try to identify the location parameters of the distributions and thus can be used to find clusters. `Mclust()` uses Bayesian Information Criterion (BIC) to find the number of clusters (model selection). BIC uses the likelihood and a penalty term to guard against overfitting.

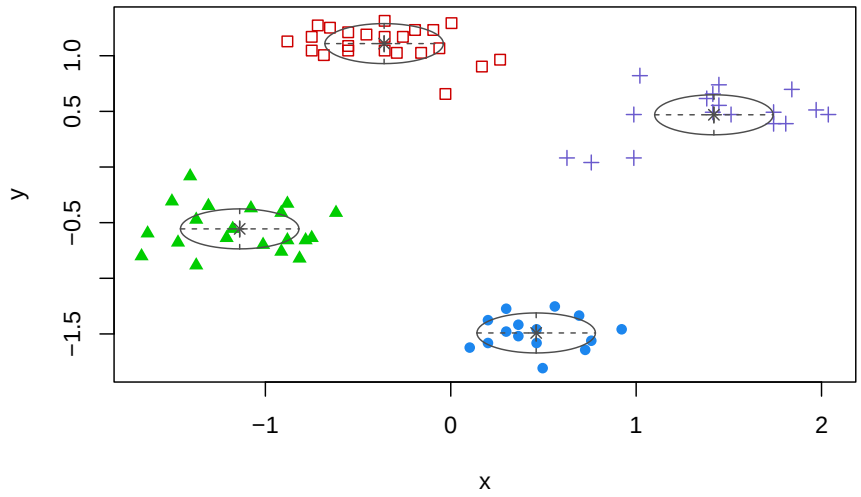
```
m <- Mclust(ruspini_scaled)
summary(m)
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust EEI (diagonal, equal volume and shape) model with 5
## components:
##
##  log-likelihood  n df    BIC    ICL
##          -91.26 75 16 -251.6 -251.7
##
## Clustering table:
##  1  2  3  4  5
## 15 23 20 14  3
plot(m, what = "classification")
```

[21] https://en.wikipedia.org/wiki/Mixture_model#Multivariate_Gaussian_mixture_model



Rerun with a fixed number of 4 clusters

```
m <- Mclust(ruspini_scaled, G=4)
summary(m)
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust EEI (diagonal, equal volume and shape) model with 4
## components:
##
##  log-likelihood  n df    BIC    ICL
##          -101.6 75 13 -259.3 -259.3
##
## Clustering table:
##  1  2  3  4
## 15 23 20 17
plot(m, what = "classification")
```



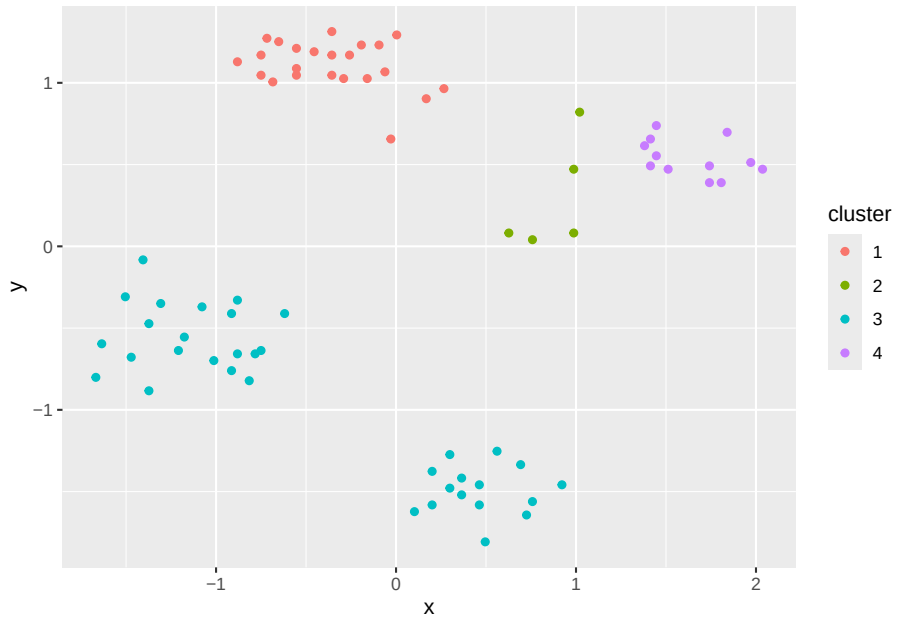
7.6.3 Spectral clustering

Spectral clustering[22] works by embedding the data points of the partitioning problem into the subspace of the k largest eigenvectors of a normalized affinity/kernel matrix. Then uses a simple clustering method like k-means.

```
library("kernlab")
##
## Attaching package: 'kernlab'
## The following object is masked from 'package:arulesSequences':
##
##     size
## The following object is masked from 'package:scales':
##
##     alpha
## The following object is masked from 'package:arules':
##
##     size
## The following object is masked from 'package:purrr':
##
##     cross
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

[22] https://en.wikipedia.org/wiki/Spectral_clustering

```
cluster_spec <- specc(as.matrix(ruspini_scaled), centers = 4)
cluster_spec
## Spectral Clustering object of class "specc"
##
##  Cluster memberships:
##
## 3 1 3 1 1 4 4 4 1 3 3 3 1 3 3 4 1 3 1 2 1 1 3 3 3 2 1 4 1 3 3 4 3 2 3 2 3 3 3 3 4 3 1 1 3 3 4
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  45.2704818907988
##
## Centers:
##         [,1]    [,2]
## [1,] -0.3595  1.1091
## [2,]  0.8760  0.2992
## [3,] -0.4532 -0.9568
## [4,]  1.6459  0.5401
##
## Cluster size:
## [1] 23  5 35 12
##
## Within-cluster sum of squares:
## [1] 49.920  1.444 35.792 14.334
ggplot(ruspini_scaled |>
         add_column(cluster = factor(cluster_spec)),
       aes(x, y, color = cluster)) +
  geom_point()
```



7.6.4 Fuzzy C-Means Clustering

The fuzzy clustering[23] version of the k-means clustering problem. Each data point has a degree of membership to for each cluster.

```
library("e1071")

cluster_cmeans <- cmeans(as.matrix(ruspini_scaled), centers = 4)
cluster_cmeans
## Fuzzy c-means clustering with 4 clusters
##
## Cluster centers:
##         x       y
## 1 -0.3763  1.1143
## 2 -1.1371 -0.5550
## 3  0.4552 -1.4760
## 4  1.5048  0.5161
##
## Memberships:
##                1         2         3         4
##  [1,] 1.347e-03 0.0031669 0.9936326 1.853e-03
##  [2,] 9.103e-01 0.0483863 0.0168060 2.448e-02
##  [3,] 4.507e-02 0.9045083 0.0339768 1.645e-02
##  [4,] 9.754e-01 0.0120656 0.0047501 7.761e-03
```

[23] https://en.wikipedia.org/wiki/Fuzzy_clustering

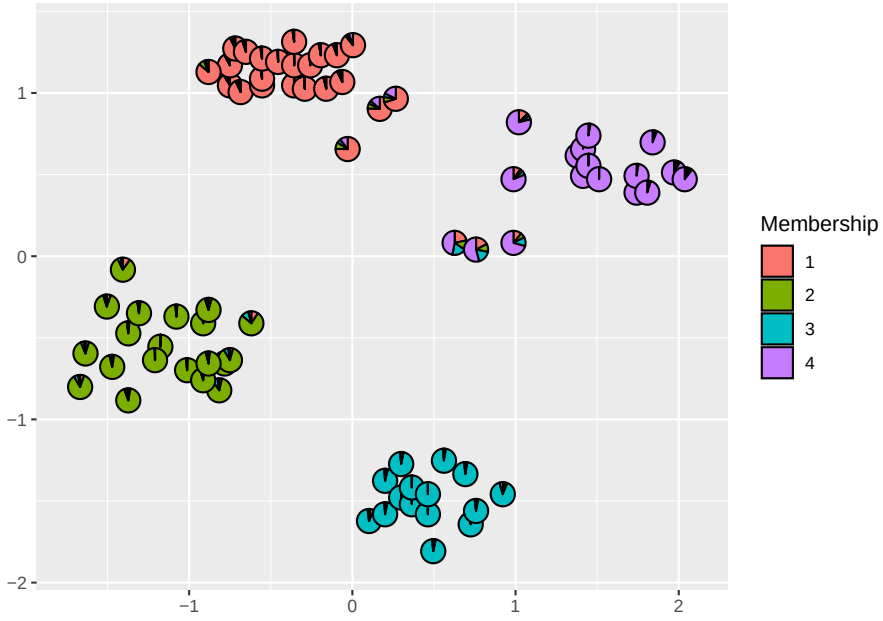
```
##  [5,] 9.924e-01 0.0033852 0.0014974 2.767e-03
##  [6,] 1.130e-02 0.0058687 0.0099606 9.729e-01
##  [7,] 1.929e-02 0.0106788 0.0192344 9.508e-01
##  [8,] 3.390e-02 0.0184311 0.0318361 9.158e-01
##  [9,] 9.467e-01 0.0256400 0.0103557 1.729e-02
## [10,] 5.308e-02 0.8953105 0.0336350 1.798e-02
## [11,] 4.259e-02 0.8727668 0.0633407 2.130e-02
## [12,] 1.664e-03 0.0038611 0.9921767 2.298e-03
## [13,] 9.294e-01 0.0376067 0.0133140 1.973e-02
## [14,] 1.710e-02 0.9626040 0.0138103 6.486e-03
## [15,] 5.052e-05 0.0001096 0.9997657 7.423e-05
## [16,] 5.072e-04 0.0002500 0.0004110 9.988e-01
## [17,] 9.395e-01 0.0204605 0.0114695 2.852e-02
## [18,] 3.143e-03 0.9920939 0.0034031 1.360e-03
## [19,] 9.763e-01 0.0095410 0.0046349 9.540e-03
## [20,] 9.619e-02 0.0392634 0.0536233 8.109e-01
## [21,] 9.245e-01 0.0371551 0.0146299 2.371e-02
## [22,] 7.551e-01 0.0672224 0.0448207 1.329e-01
## [23,] 2.343e-02 0.0384838 0.8921803 4.591e-02
## [24,] 1.727e-02 0.0498125 0.9125420 2.038e-02
## [25,] 2.104e-02 0.9397734 0.0290833 1.010e-02
## [26,] 1.267e-01 0.0394189 0.0461300 7.877e-01
## [27,] 8.622e-01 0.0758529 0.0256687 3.625e-02
## [28,] 1.483e-02 0.0061529 0.0087250 9.703e-01
## [29,] 9.753e-01 0.0114709 0.0048172 8.408e-03
## [30,] 3.347e-03 0.0082410 0.9839899 4.423e-03
## [31,] 1.016e-02 0.0183427 0.9537440 1.775e-02
## [32,] 7.444e-03 0.0032215 0.0047483 9.846e-01
## [33,] 1.011e-02 0.0241504 0.9523625 1.337e-02
## [34,] 1.739e-01 0.1075109 0.1773920 5.412e-01
## [35,] 2.687e-02 0.9343409 0.0272622 1.153e-02
## [36,] 2.177e-01 0.1283095 0.1837593 4.702e-01
## [37,] 2.256e-02 0.9540722 0.0155207 7.844e-03
## [38,] 1.083e-02 0.0287389 0.9470184 1.341e-02
## [39,] 3.837e-02 0.8665163 0.0740304 2.108e-02
## [40,] 1.396e-03 0.0030791 0.9934964 2.029e-03
## [41,] 1.387e-02 0.0075736 0.0135378 9.650e-01
## [42,] 9.312e-02 0.7683908 0.0971198 4.137e-02
## [43,] 9.272e-01 0.0247830 0.0139391 3.405e-02
## [44,] 9.187e-01 0.0419845 0.0155193 2.380e-02
## [45,] 5.522e-02 0.8604234 0.0593853 2.497e-02
## [46,] 1.357e-02 0.9712836 0.0102381 4.907e-03
## [47,] 2.683e-02 0.0130794 0.0205423 9.395e-01
## [48,] 8.915e-01 0.0333616 0.0199931 5.510e-02
## [49,] 2.248e-02 0.9302691 0.0358176 1.143e-02
```

```
## [50,] 9.545e-03 0.9731702 0.0127746 4.511e-03
## [51,] 4.474e-04 0.9989319 0.0004367 1.839e-04
## [52,] 4.091e-02 0.0229222 0.0405198 8.957e-01
## [53,] 2.457e-03 0.0011605 0.0018410 9.945e-01
## [54,] 9.977e-01 0.0009643 0.0004512 8.879e-04
## [55,] 1.087e-02 0.0205932 0.9503615 1.818e-02
## [56,] 8.074e-03 0.0034562 0.0050420 9.834e-01
## [57,] 9.889e-01 0.0046264 0.0021820 4.268e-03
## [58,] 9.017e-03 0.0173388 0.9591009 1.454e-02
## [59,] 7.514e-01 0.0920666 0.0519047 1.047e-01
## [60,] 7.016e-01 0.0713875 0.0509703 1.760e-01
## [61,] 4.917e-02 0.8877275 0.0431104 1.999e-02
## [62,] 3.002e-02 0.9148676 0.0405064 1.461e-02
## [63,] 9.693e-01 0.0112515 0.0059276 1.354e-02
## [64,] 1.100e-01 0.0654687 0.1188630 7.056e-01
## [65,] 9.817e-02 0.8288352 0.0452769 2.771e-02
## [66,] 1.324e-03 0.0006093 0.0009438 9.971e-01
## [67,] 3.663e-02 0.8900971 0.0549599 1.831e-02
## [68,] 2.538e-02 0.9426532 0.0220708 9.896e-03
## [69,] 9.597e-01 0.0151626 0.0078897 1.728e-02
## [70,] 9.526e-03 0.0254633 0.9531428 1.187e-02
## [71,] 9.964e-01 0.0015599 0.0007050 1.322e-03
## [72,] 1.136e-02 0.0249351 0.9472518 1.646e-02
## [73,] 9.787e-01 0.0103272 0.0041351 6.878e-03
## [74,] 1.111e-02 0.0203875 0.9492343 1.926e-02
## [75,] 9.885e-01 0.0044836 0.0022372 4.751e-03
##
## Closest hard clustering:
##  [1] 3 1 2 1 1 4 4 4 1 2 2 3 1 2 3 4 1 2 1 4 1 1 3 3 2 4 1 4
## [29] 1 3 3 4 3 4 2 4 2 3 2 3 4 2 1 1 2 2 4 1 2 2 2 4 4 1 3 4
## [57] 1 3 1 1 2 2 1 4 2 4 2 2 1 3 1 3 1 3 1
##
## Available components:
## [1] "centers"     "size"        "cluster"     "membership"
## [5] "iter"        "withinerror" "call"
```

Plot membership (shown as small pie charts)

```
library("scatterpie")
## scatterpie v0.2.4 Learn more at https://yulab-smu.top/
##
## Please cite:
##
## D Wang, G Chen, L Li, S Wen, Z Xie, X Luo, L
## Zhan, S Xu, J Li, R Wang, Q Wang, G Yu. Reducing language
## barriers, promoting information absorption, and
```

```
## communication using fanyi. Chinese Medical Journal. 2024,
## 137(16):1950-1956. doi: 10.1097/CM9.0000000000003242
ggplot()  +
  geom_scatterpie(
    data = cbind(ruspini_scaled, cluster_cmeans$membership),
    aes(x = x, y = y),
    cols = colnames(cluster_cmeans$membership),
    legend_name = "Membership") +
  coord_equal()
```



7.7 Outliers in Clustering*

Most clustering algorithms perform complete assignment (i.e., all data points need to be assigned to a cluster). Outliers will affect the clustering. It is useful to identify outliers and remove strong outliers prior to clustering. A density based method to identify outlier is LOF[24] (Local Outlier Factor). It is related to dbscan and compares the density around a point with the densities around its neighbors (you have to specify the neighborhood size k). The LOF value for a regular data point is 1. The larger the LOF value gets, the more likely the point is an outlier.

```
library(dbscan)
```

[24] https://en.wikipedia.org/wiki/Local_outlier_factor

Add a clear outlier to the scaled Ruspini dataset that is 10 standard deviations above the average for the x axis.

```
ruspini_scaled_outlier <- ruspini_scaled |> add_case(x=10,y=0)
```

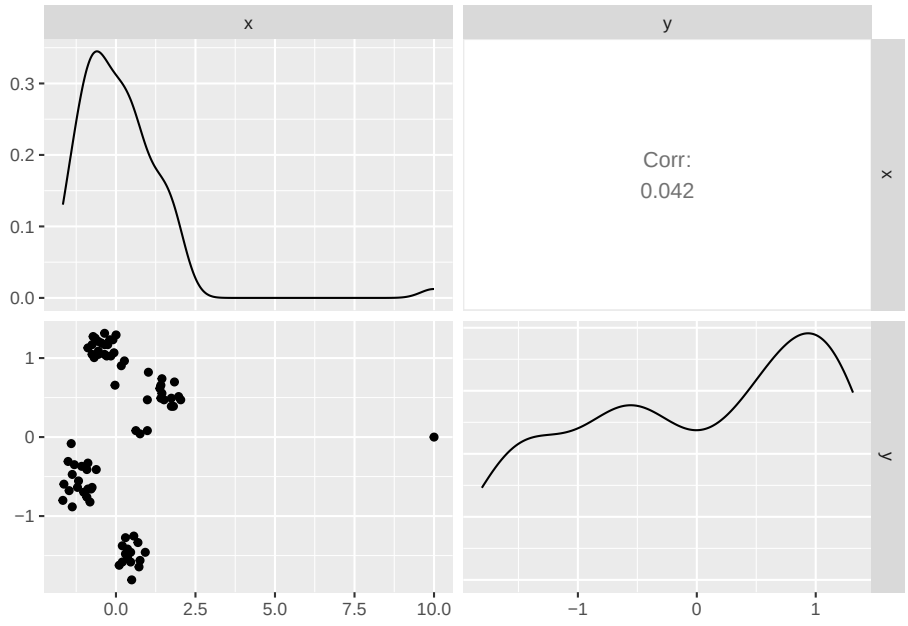
7.7.1 Visual inspection of the data

Outliers can be identified using summary statistics, histograms, scatterplots (pairs plots), and boxplots, etc. We use here a pairs plot (the diagonal contains smoothed histograms). The outlier is visible as the single separate point in the scatter plot and as the long tail of the smoothed histogram for `x` (we would expect most observations to fall in the range

$$-3, 3$$

in normalized data).

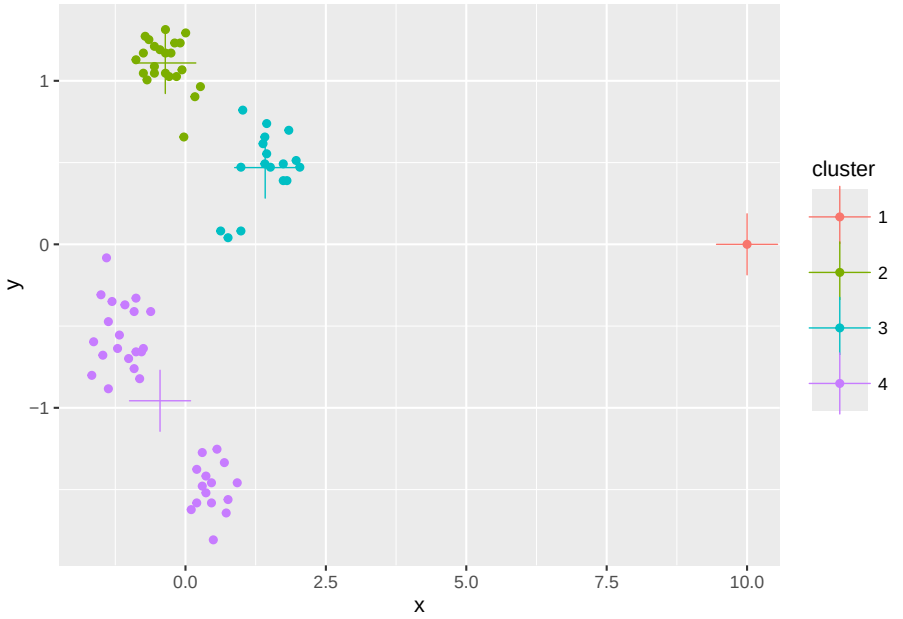
```
library("GGally")
ggpairs(ruspini_scaled_outlier, progress = FALSE)
```



The outlier is a problem for k-means

```
km <- kmeans(ruspini_scaled_outlier, centers = 4, nstart = 10)
ruspini_scaled_outlier_km <- ruspini_scaled_outlier|>
  add_column(cluster = factor(km$cluster))
centroids <- as_tibble(km$centers, rownames = "cluster")
```

```
ggplot(ruspini_scaled_outlier_km, aes(x = x, y = y, color = cluster)) +
  geom_point() +
  geom_point(data = centroids,
             aes(x = x, y = y, color = cluster),
             shape = 3, size = 10)
```



This problem can be fixed by increasing the number of clusters and removing small clusters in a post-processing step or by identifying and removing outliers before clustering.

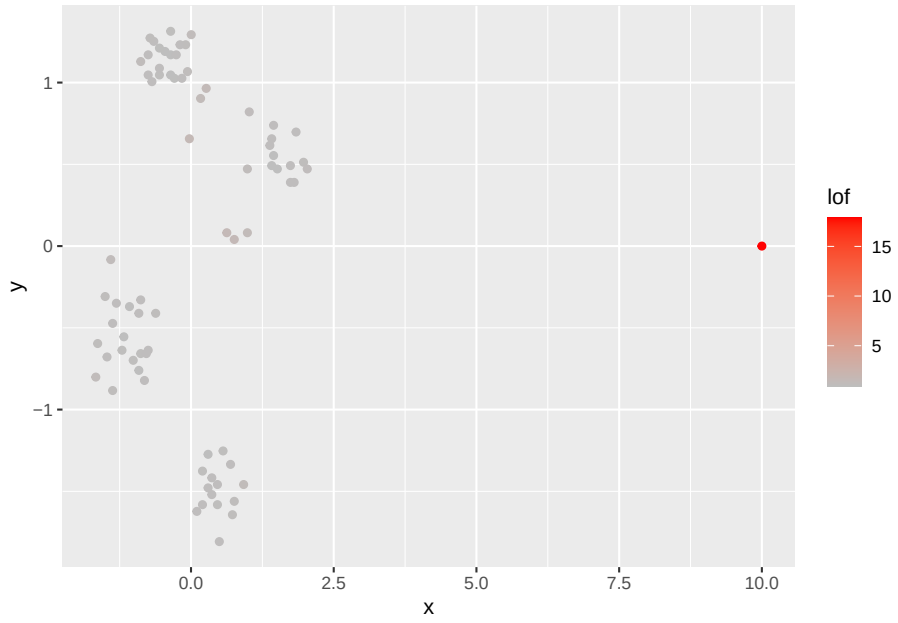
7.7.2 Local Outlier Factor (LOF)

The Local Outlier Factor[25] is related to concepts of DBSCAN can help to identify potential outliers. Calculate the LOF (I choose a local neighborhood size of 10 for density estimation),

```
lof <- lof(ruspini_scaled_outlier, minPts= 10)
lof
##  [1]  1.0145  1.0396  1.0008  0.9815  0.9427  1.0206  1.0041
##  [8]  1.0842  1.0213  1.0881  1.0234  0.9972  1.0340  1.0393
## [15]  0.9939  0.9385  1.0702  0.9825  0.9850  1.1603  1.0680
## [22]  1.3973  1.2518  1.0214  0.9767  1.1429  1.1750  0.9982
## [29]  0.9653  0.9928  1.0262  1.0022  0.9625  1.5716  1.0509
## [36]  1.5976  1.0200  0.9809  1.0413  0.9864  1.0055  1.0830
```

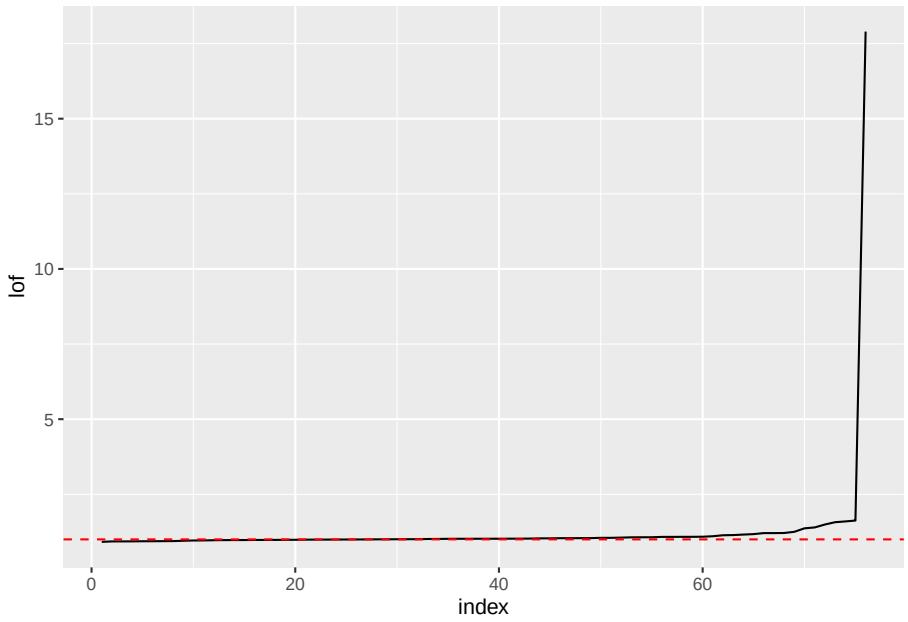
[25] https://en.wikipedia.org/wiki/Local_outlier_factor

```
## [43]  1.1045  1.0340  1.2090  0.9175  1.0062  1.2088  0.9536
## [50]  0.9778  0.9411  1.1375  0.9377  0.9319  1.0708  0.9920
## [57]  0.9716  0.9758  1.6258  1.4943  1.0852  1.0828  1.0515
## [64]  1.3682  1.2124  0.9329  1.0198  1.0127  1.0409  0.9957
## [71]  0.9336  1.0238  0.9903  1.0593  1.0187 17.8995
ggplot(ruspini_scaled_outlier |> add_column(lof = lof),
       aes(x, y, color = lof)) +
  geom_point() +
  scale_color_gradient(low = "gray", high = "red")
```



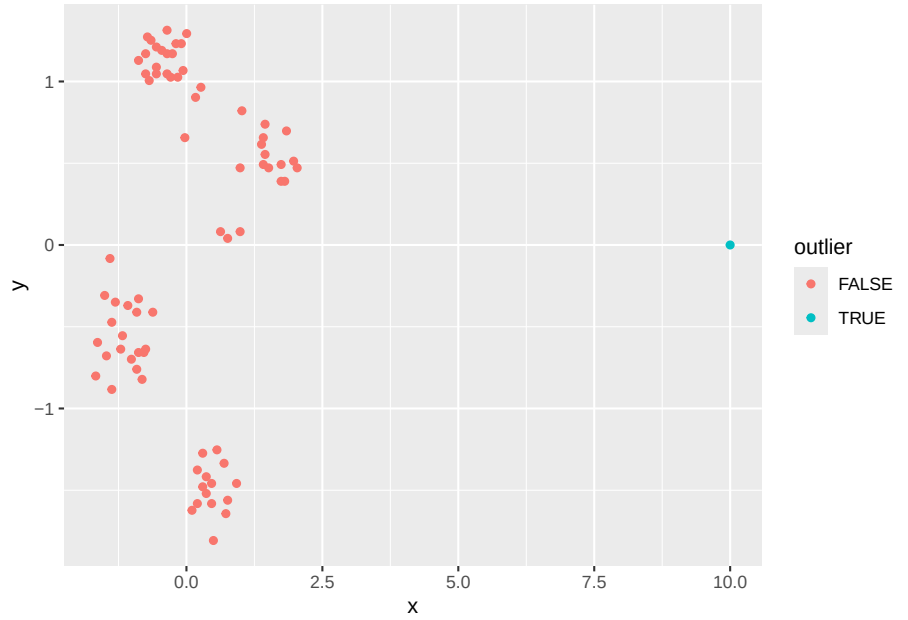
Plot the points sorted by increasing LOF and look for a knee.

```
ggplot(tibble(index = seq_len(length(lof)), lof = sort(lof)),
       aes(index, lof)) +
  geom_line() +
  geom_hline(yintercept = 1, color = "red", linetype = 2)
```



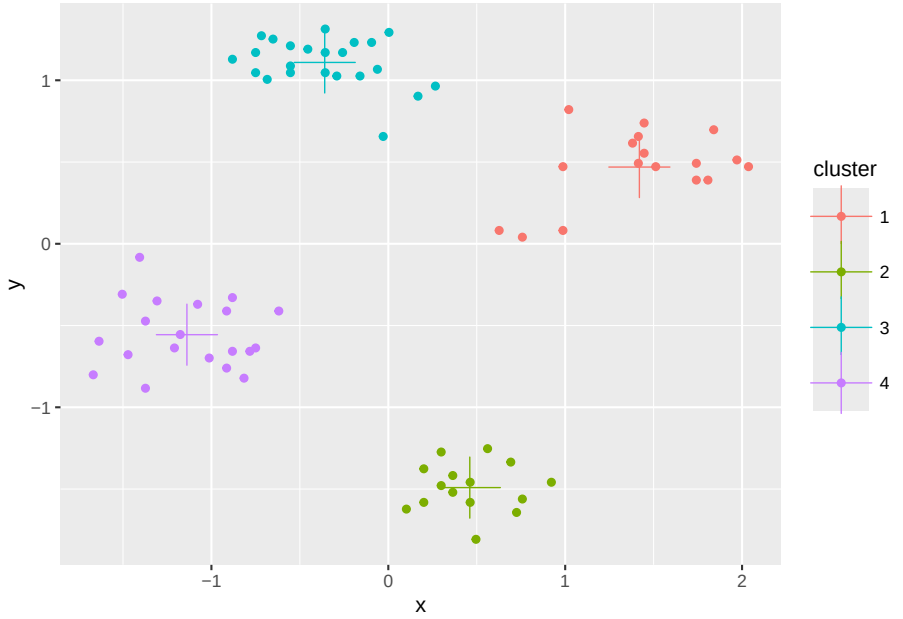
Choose a threshold above 1.

```
ggplot(ruspini_scaled_outlier |> add_column(outlier = lof >= 2),
       aes(x, y, color = outlier)) +
  geom_point()
```



Analyze the found outliers (they might be interesting data points) and then cluster the data without them.

```
ruspini_scaled_clean <- ruspini_scaled_outlier  |> filter(lof < 2)

km <- kmeans(ruspini_scaled_clean, centers = 4, nstart = 10)
ruspini_scaled_clean_km <- ruspini_scaled_clean|>
  add_column(cluster = factor(km$cluster))
centroids <- as_tibble(km$centers, rownames = "cluster")

ggplot(ruspini_scaled_clean_km, aes(x = x, y = y, color = cluster)) +
  geom_point() +
  geom_point(data = centroids,
             aes(x = x, y = y, color = cluster),
             shape = 3, size = 10)
```



There are many other outlier removal strategies available. See, e.g., package outliers[26].

7.8 Exercises*

We will again use the Palmer penguin data for the exercises.

[26] https://cran.r-project.org/package=outliers

```
library(palmerpenguins)
head(penguins)
## # A tibble: 6 x 8
##   species island    bill_length_mm bill_depth_mm
##   <chr>   <chr>              <dbl>         <dbl>
## 1 Adelie  Torgersen           39.1          18.7
## 2 Adelie  Torgersen           39.5          17.4
## 3 Adelie  Torgersen           40.3          18
## 4 Adelie  Torgersen           NA            NA
## 5 Adelie  Torgersen           36.7          19.3
## 6 Adelie  Torgersen           39.3          20.6
## # i 4 more variables: flipper_length_mm <dbl>,
## #   body_mass_g <dbl>, sex <chr>, year <dbl>
```

Create a R markdown file with the code and discussion for the following below.

1. What features do you use for clustering? What about missing values? Discuss your answers. Do you need to scale the data before clustering? Why?
2. What distance measure do you use to reflect similarities between penguins? See Measures of Similarity and Dissimilarity in Chapter 2.
3. Apply k-means clustering. Use an appropriate method to determine the number of clusters. Compare the clustering using unscaled data and scaled data. What is the difference? Visualize and describe the results.
4. Apply hierarchical clustering. Create a dendrogram and discuss what it means.
5. Apply DBSCAN. How do you choose the parameters? How well does it work?

Chapter 8

Regression*

This chapter introduces the regression problem and multiple linear regression. In addition, alternative models like regression trees and regularized regression are discussed.

Packages Used in this Chapter

```
pkgs <- c('lars')

pkgs_install <- pkgs[!(pkgs %in% installed.packages()[,"Package"])]
if(length(pkgs_install)) install.packages(pkgs_install)
```

The packages used for this chapter are:

- *lars* (Hastie and Efron 2022)

8.1 Introduction

Classification predicts one of a small set of discrete labels (e.g., yes or no). Regression is also a supervised learning problem, but the goal is to predict the value of a continuous value given a set of predictors. We start with the popular linear regression and will later discuss alternative regression methods.

Linear regression models the value of a dependent variable y (also called response) as as linear function of independent variables $X_1, X_2, ..., X_p$ (also called regressors, predictors, exogenous variables or covariates). Given n observations the model is:

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \epsilon_i$$

where β_0 is the intercept, β is a p-dimensional parameter vector learned from the data, and ϵ is the error term (called residuals).

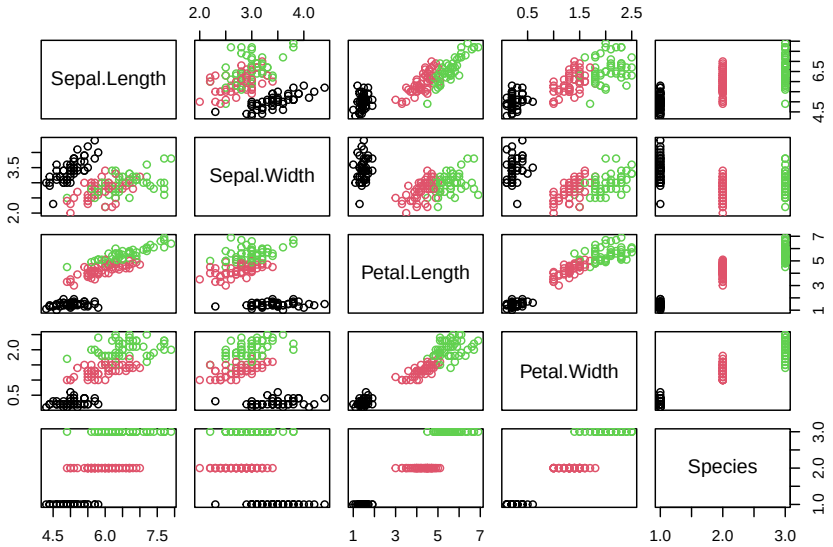
Linear regression makes several assumptions:

- *Weak exogeneity:* Predictor variables are assumed to be error free.
- *Linearity:* There is a linear relationship between dependent and independent variables.
- *Homoscedasticity:* The variance of the error (ϵ) does not change (increase) with the predicted value.
- *Independence of errors:* Errors between observations are uncorrelated.
- *No multicollinearity of predictors:* Predictors cannot be perfectly correlated or the parameter vector cannot be identified. *Note* that highly correlated predictors lead to unstable results and should be avoided.

8.2 A First Linear Regression Model

Load and shuffle data (flowers are in order by species)

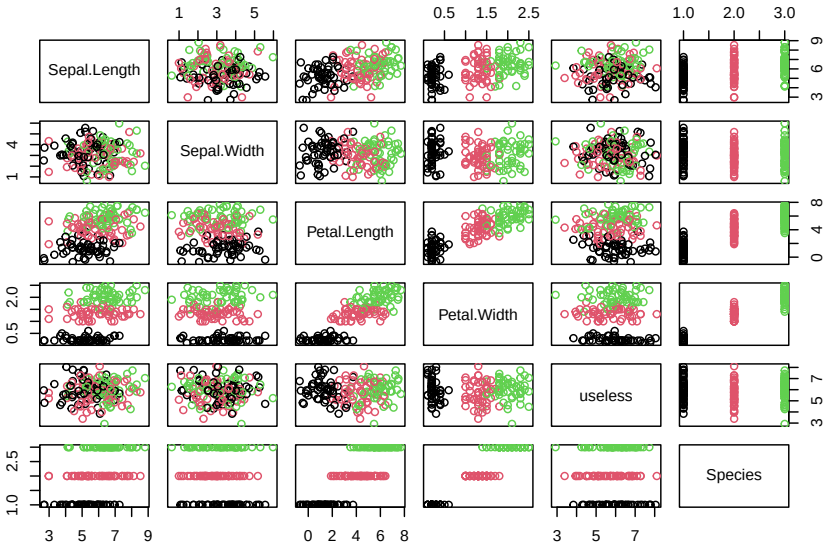
```
data(iris)
set.seed(2000) # make the sampling reproducible

x <- iris[sample(1:nrow(iris)),]
plot(x, col=x$Species)
```



Make the data a little messy and add a useless feature

```
x[,1] <- x[,1] + rnorm(nrow(x))
x[,2] <- x[,2] + rnorm(nrow(x))
```

```
x[,3] <- x[,3] + rnorm(nrow(x))
x <- cbind(x[,-5],
           useless = mean(x[,1]) + rnorm(nrow(x)),
           Species = x[,5])

plot(x, col=x$Species)
```




```
summary(x)
##   Sepal.Length   Sepal.Width     Petal.Length
##  Min.   :2.68   Min.   :0.611   Min.   :-0.713
##  1st Qu.:5.05   1st Qu.:2.306   1st Qu.: 1.876
##  Median :5.92   Median :3.171   Median : 4.102
##  Mean   :5.85   Mean   :3.128   Mean   : 3.781
##  3rd Qu.:6.70   3rd Qu.:3.945   3rd Qu.: 5.546
##  Max.   :8.81   Max.   :5.975   Max.   : 7.708
##   Petal.Width      useless          Species
##  Min.   :0.1   Min.   :2.92   setosa     :50
##  1st Qu.:0.3   1st Qu.:5.23   versicolor:50
##  Median :1.3   Median :5.91   virginica :50
##  Mean   :1.2   Mean   :5.88
##  3rd Qu.:1.8   3rd Qu.:6.57
##  Max.   :2.5   Max.   :8.11
head(x)
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 85         2.980       1.464        5.227         1.5
```

```
## 104        5.096       3.044        5.187        1.8
## 30         4.361       2.832        1.861        0.2
## 53         8.125       2.406        5.526        1.5
## 143        6.372       1.565        6.147        1.9
## 142        6.526       3.697        5.708        2.3
##     useless    Species
## 85    5.712 versicolor
## 104   6.569  virginica
## 30    4.299     setosa
## 53    6.124 versicolor
## 143   6.553  virginica
## 142   5.222  virginica
```

Create some training and learning data

```
train <- x[1:100,]
test <- x[101:150,]
```

Can we predict Petal.Width using the other variables?

lm uses a formula interface see ?lm for description

```
model1 <- lm(Petal.Width ~ Sepal.Length
            + Sepal.Width + Petal.Length + useless,
            data = train)
model1
##
## Call:
## lm(formula = Petal.Width ~ Sepal.Length + Sepal.Width + Petal.Length +
##     useless, data = train)
##
## Coefficients:
##  (Intercept)  Sepal.Length   Sepal.Width  Petal.Length
##     -0.20589       0.01957       0.03406       0.30814
##      useless
##      0.00392
coef(model1)
##  (Intercept) Sepal.Length  Sepal.Width Petal.Length
##    -0.205886     0.019568     0.034062     0.308139
##      useless
##     0.003917
```

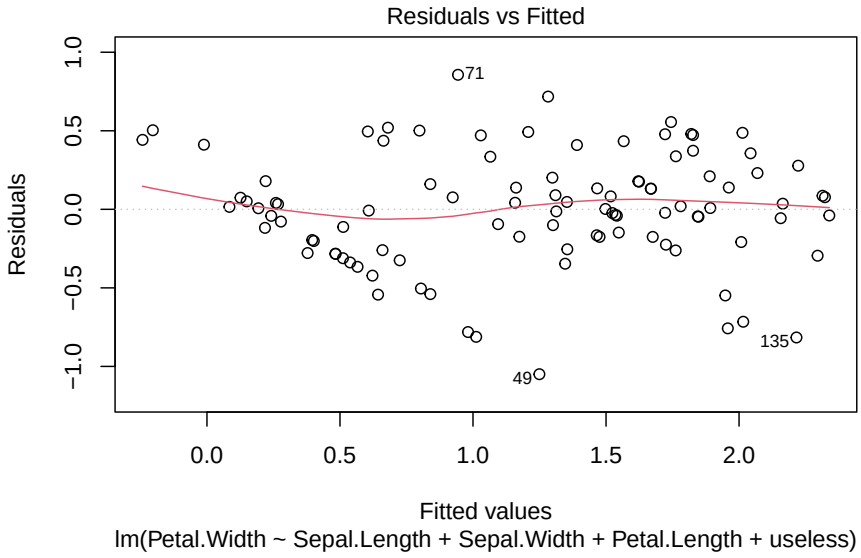
Summary shows:

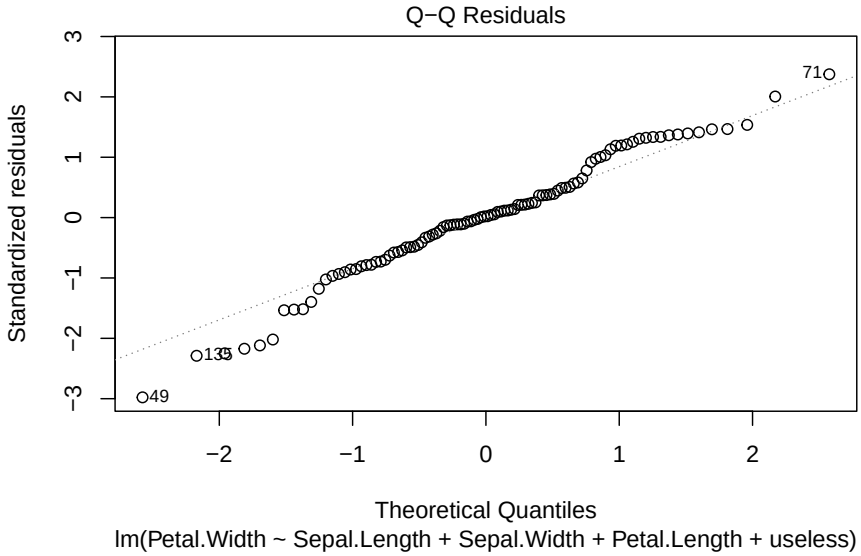
- Which coefficients are significantly different from 0
- R-squared (coefficient of determination): Proportion of the variability of the dependent variable explained by the model. It is better to look at the adjusted R-square (adjusted for number of dependent vars.)

```
summary(model1)
##
## Call:
## lm(formula = Petal.Width ~ Sepal.Length + Sepal.Width + Petal.Length +
##     useless, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.0495 -0.2033  0.0074  0.2038  0.8564
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.20589    0.28723   -0.72     0.48
## Sepal.Length  0.01957    0.03265    0.60     0.55
## Sepal.Width   0.03406    0.03549    0.96     0.34
## Petal.Length  0.30814    0.01819   16.94   <2e-16 ***
## useless       0.00392    0.03558    0.11     0.91
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.366 on 95 degrees of freedom
## Multiple R-squared:  0.778,  Adjusted R-squared:  0.769
## F-statistic: 83.4 on 4 and 95 DF,  p-value: <2e-16
```

Plotting the model produces diagnostic plots (see `? plot.lm`). For example to check for homoscedasticity (residual vs predicted value scatter plot should produce a close to horizontal line) and if the residuals are approximately normally distributed (Q-Q plot should be close to the straight line).

```
plot(model1, which = 1:2)
```





8.3 Comparing Nested Models

Here we create a simpler model by using only three predictors.

```
model2 <- lm(Petal.Width ~ Sepal.Length +
               Sepal.Width + Petal.Length,
             data = train)
summary(model2)
##
## Call:
## lm(formula = Petal.Width ~ Sepal.Length + Sepal.Width + Petal.Length,
##     data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.0440 -0.2024  0.0099  0.1998  0.8513
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -0.1842     0.2076   -0.89     0.38
## Sepal.Length   0.0199     0.0323    0.62     0.54
## Sepal.Width    0.0339     0.0353    0.96     0.34
## Petal.Length   0.3080     0.0180   17.07   <2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.365 on 96 degrees of freedom
## Multiple R-squared:  0.778,  Adjusted R-squared:  0.771
## F-statistic:  112 on 3 and 96 DF,  p-value: <2e-16
```

We can remove the intercept by adding `-1` to the formula.

```
model3 <- lm(Petal.Width ~ Sepal.Length +
               Sepal.Width + Petal.Length - 1,
             data = train)
summary(model3)
##
## Call:
## lm(formula = Petal.Width ~ Sepal.Length + Sepal.Width + Petal.Length -
##     1, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.0310 -0.1961 -0.0051  0.2264  0.8246
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## Sepal.Length -0.00168    0.02122   -0.08     0.94
## Sepal.Width   0.01859    0.03073    0.61     0.55
```

```
## Petal.Length  0.30666    0.01796   17.07   <2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.364 on 97 degrees of freedom
## Multiple R-squared:  0.938,  Adjusted R-squared:  0.936
## F-statistic:  486 on 3 and 97 DF,  p-value: <2e-16
```

here is a very simple model:

```
model4 <- lm(Petal.Width ~ Petal.Length -1,
             data = train)
summary(model4)
##
## Call:
## lm(formula = Petal.Width ~ Petal.Length - 1, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.9774 -0.1957  0.0078  0.2536  0.8455
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## Petal.Length  0.31586    0.00822    38.4   <2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.362 on 99 degrees of freedom
## Multiple R-squared:  0.937,  Adjusted R-squared:  0.936
## F-statistic: 1.47e+03 on 1 and 99 DF,  p-value: <2e-16
```

Compare models (Null hypothesis: all treatments=models have the same effect). **Note:** This only works for *nested models*. Models are nested only if one model contains all the predictors of the other model.

```
anova(model1, model2, model3, model4)
## Analysis of Variance Table
##
## Model 1: Petal.Width ~ Sepal.Length + Sepal.Width + Petal.Length + useless
## Model 2: Petal.Width ~ Sepal.Length + Sepal.Width + Petal.Length
## Model 3: Petal.Width ~ Sepal.Length + Sepal.Width + Petal.Length - 1
## Model 4: Petal.Width ~ Petal.Length - 1
##   Res.Df  RSS Df Sum of Sq    F Pr(>F)
## 1     95 12.8
```

```
## 2     96 12.8 -1   -0.0016 0.01   0.91
## 3     97 12.9 -1   -0.1046 0.78   0.38
## 4     99 13.0 -2   -0.1010 0.38   0.69
```

Models 1 is not significantly better than model 2. Model 2 is not significantly better than model 3. Model 3 is not significantly better than model 4! Use model 4 (simplest model)

8.4 Stepwise Variable Selection

Automatically looks for the smallest AIC (Akaike information criterion)

```
s1 <- step(lm(Petal.Width ~ . -Species, data = train))
## Start:  AIC=-195.9
## Petal.Width ~ (Sepal.Length + Sepal.Width + Petal.Length + useless +
##     Species) - Species
##
##                Df Sum of Sq  RSS    AIC
## - useless       1       0.0 12.8 -197.9
## - Sepal.Length  1       0.0 12.8 -197.5
## - Sepal.Width   1       0.1 12.9 -196.9
## <none>                      12.8 -195.9
## - Petal.Length  1      38.6 51.3  -58.7
##
## Step:  AIC=-197.9
## Petal.Width ~ Sepal.Length + Sepal.Width + Petal.Length
##
##                Df Sum of Sq  RSS    AIC
## - Sepal.Length  1       0.1 12.8 -199.5
## - Sepal.Width   1       0.1 12.9 -198.9
## <none>                      12.8 -197.9
## - Petal.Length  1      38.7 51.5  -60.4
##
## Step:  AIC=-199.5
## Petal.Width ~ Sepal.Width + Petal.Length
##
##                Df Sum of Sq  RSS    AIC
## - Sepal.Width   1       0.1 12.9 -200.4
## <none>                      12.8 -199.5
## - Petal.Length  1      44.7 57.5  -51.3
##
## Step:  AIC=-200.4
## Petal.Width ~ Petal.Length
##
##                Df Sum of Sq  RSS    AIC
```

```
## <none>                       12.9 -200.4
## - Petal.Length  1      44.6 57.6  -53.2
summary(s1)
##
## Call:
## lm(formula = Petal.Width ~ Petal.Length, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.9848 -0.1873  0.0048  0.2466  0.8343
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.0280     0.0743    0.38     0.71
## Petal.Length   0.3103     0.0169   18.38   <2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.364 on 98 degrees of freedom
## Multiple R-squared:  0.775,  Adjusted R-squared:  0.773
## F-statistic:  338 on 1 and 98 DF,  p-value: <2e-16
```

8.5 Modeling with Interaction Terms

What if two variables are only important together? Interaction terms are modeled with `:` or `*` in the formula (they are literally multiplied). See `? formula`.

```
model5 <- step(lm(Petal.Width ~ Sepal.Length *
                    Sepal.Width * Petal.Length,
            data = train))
## Start:  AIC=-196.4
## Petal.Width ~ Sepal.Length * Sepal.Width * Petal.Length
##
##                                         Df Sum of Sq  RSS
## <none>                                               11.9
## - Sepal.Length:Sepal.Width:Petal.Length  1     0.265 12.2
##                                          AIC
## <none>                                  -196
## - Sepal.Length:Sepal.Width:Petal.Length -196
summary(model5)
##
## Call:
## lm(formula = Petal.Width ~ Sepal.Length * Sepal.Width * Petal.Length,
##     data = train)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1064 -0.1882  0.0238  0.1767  0.8577
##
## Coefficients:
##                                       Estimate Std. Error
## (Intercept)                            -2.4207     1.3357
## Sepal.Length                            0.4484     0.2313
## Sepal.Width                             0.5983     0.3845
## Petal.Length                            0.7275     0.2863
## Sepal.Length:Sepal.Width               -0.1115     0.0670
## Sepal.Length:Petal.Length              -0.0833     0.0477
## Sepal.Width:Petal.Length               -0.0941     0.0850
## Sepal.Length:Sepal.Width:Petal.Length   0.0201     0.0141
##                                       t value Pr(>|t|)
## (Intercept)                             -1.81    0.073 .
## Sepal.Length                             1.94    0.056 .
## Sepal.Width                              1.56    0.123
## Petal.Length                             2.54    0.013 *
## Sepal.Length:Sepal.Width                -1.66    0.100 .
## Sepal.Length:Petal.Length               -1.75    0.084 .
## Sepal.Width:Petal.Length                -1.11    0.272
## Sepal.Length:Sepal.Width:Petal.Length    1.43    0.157
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.36 on 92 degrees of freedom
## Multiple R-squared:  0.792,  Adjusted R-squared:  0.777
## F-statistic: 50.2 on 7 and 92 DF,  p-value: <2e-16
anova(model5, model4)
## Analysis of Variance Table
##
## Model 1: Petal.Width ~ Sepal.Length * Sepal.Width * Petal.Length
## Model 2: Petal.Width ~ Petal.Length - 1
##   Res.Df  RSS Df Sum of Sq    F Pr(>F)
## 1     92 11.9
## 2     99 13.0 -7     -1.01 1.11   0.36
```

Model 5 is not significantly better than model 4

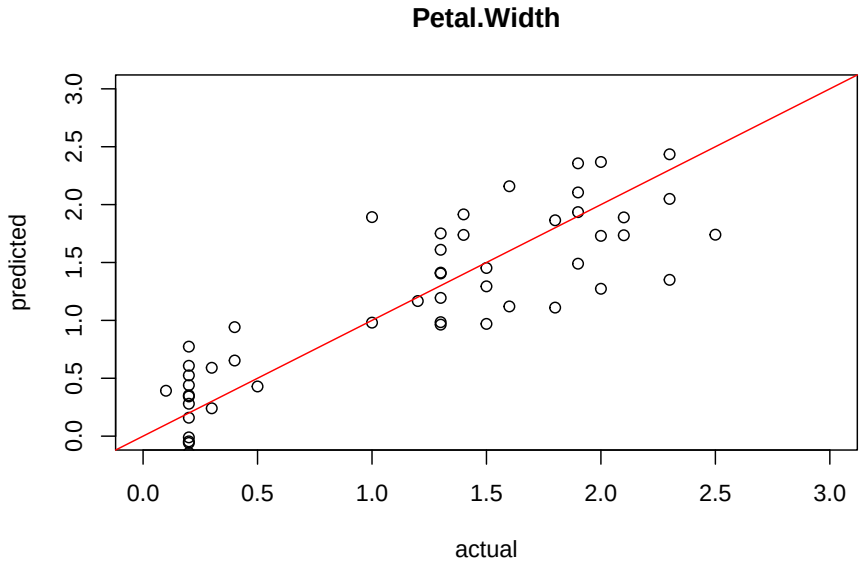
```
test[1:5,]
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 128        8.017       1.541        3.515         1.8
## 92         5.268       4.064        6.064         1.4
## 50         5.461       4.161        1.117         0.2
## 134        6.055       2.951        4.599         1.5
## 8          4.900       5.096        1.086         0.2
##     useless    Species
## 128   6.110  virginica
## 92    4.938 versicolor
## 50    6.373     setosa
## 134   5.595  virginica
## 8     7.270     setosa
test[1:5,]$Petal.Width
## [1] 1.8 1.4 0.2 1.5 0.2
predict(model4, test[1:5,])
##    128     92     50    134      8
## 1.1104 1.9155 0.3529 1.4526 0.3429
```

Calculate the root-mean-square error (RMSE): less is better

```
RMSE <- function(predicted, true) mean((predicted-true)^2)^.5
RMSE(predict(model4, test), test$Petal.Width)
## [1] 0.3874
```

Compare predicted vs. actual values

```
plot(test[,"Petal.Width"], predict(model4, test),
  xlim=c(0,3), ylim=c(0,3),
  xlab = "actual", ylab = "predicted",
  main = "Petal.Width")
abline(0,1, col="red")
```




```
cor(test[,"Petal.Width"], predict(model4, test))
## [1] 0.8636
```

8.7 Using Nominal Variables

Dummy coding is used for factors (i.e., levels are translated into individual 0-1 variable). The first level of factors is automatically used as the reference and the other levels are presented as 0-1 dummy variables called contrasts.

```
levels(train$Species)
## [1] "setosa"     "versicolor" "virginica"
```

`model.matrix` is used internally to create the dummy coding.

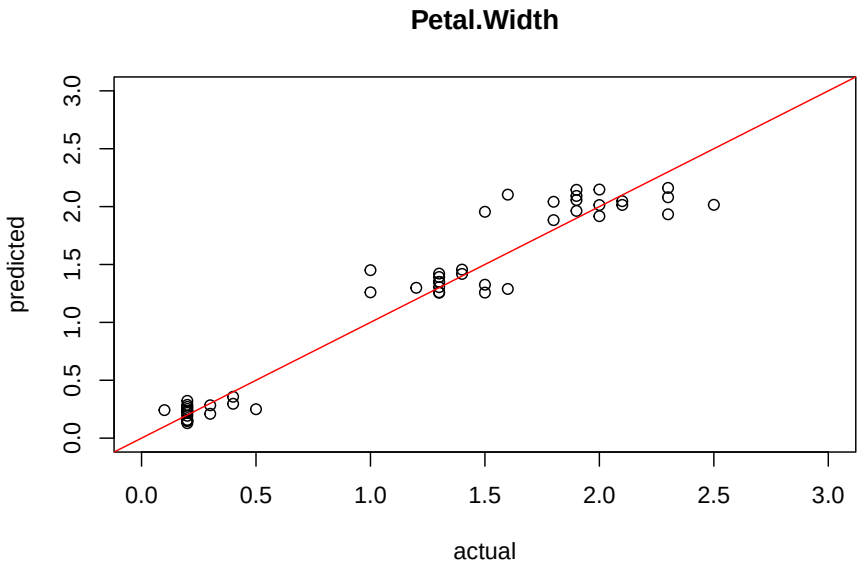
```
head(model.matrix(Petal.Width ~ ., data=train))
##     (Intercept) Sepal.Length Sepal.Width Petal.Length
## 85            1        2.980       1.464        5.227
## 104           1        5.096       3.044        5.187
## 30            1        4.361       2.832        1.861
## 53            1        8.125       2.406        5.526
## 143           1        6.372       1.565        6.147
## 142           1        6.526       3.697        5.708
##     useless Speciesversicolor Speciesvirginica
## 85    5.712                 1                0
## 104   6.569                 0                1
```

```
## 30    4.299                 0                 0
## 53    6.124                 1                 0
## 143   6.553                 0                 1
## 142   5.222                 0                 1
```

Note that there is no dummy variable for species Setosa, because it is used as the reference. It is often useful to set the reference level. A simple way is to use the function `relevel` to change which factor is listed first.

```
model6 <- step(lm(Petal.Width ~ ., data=train))
## Start:  AIC=-308.4
## Petal.Width ~ Sepal.Length + Sepal.Width + Petal.Length + useless +
##     Species
##
##                Df Sum of Sq   RSS  AIC
## - Sepal.Length  1      0.01  3.99 -310
## - Sepal.Width   1      0.01  3.99 -310
## - useless       1      0.02  4.00 -310
## <none>                       3.98 -308
## - Petal.Length  1      0.47  4.45 -299
## - Species       2      8.78 12.76 -196
##
## Step:  AIC=-310.2
## Petal.Width ~ Sepal.Width + Petal.Length + useless + Species
##
##                Df Sum of Sq   RSS  AIC
## - Sepal.Width   1      0.01  4.00 -312
## - useless       1      0.02  4.00 -312
## <none>                       3.99 -310
## - Petal.Length  1      0.49  4.48 -300
## - Species       2      8.82 12.81 -198
##
## Step:  AIC=-311.9
## Petal.Width ~ Petal.Length + useless + Species
##
##                Df Sum of Sq   RSS  AIC
## - useless       1      0.02  4.02 -313
## <none>                       4.00 -312
## - Petal.Length  1      0.48  4.48 -302
## - Species       2      8.95 12.95 -198
##
## Step:  AIC=-313.4
## Petal.Width ~ Petal.Length + Species
##
##                Df Sum of Sq   RSS  AIC
## <none>                       4.02 -313
```

```
## - Petal.Length  1      0.50  4.52 -304
## - Species       2      8.93 12.95 -200
model6
##
## Call:
## lm(formula = Petal.Width ~ Petal.Length + Species, data = train)
##
## Coefficients:
##       (Intercept)       Petal.Length  Speciesversicolor
##            0.1597             0.0664             0.8938
##  Speciesvirginica
##            1.4903
summary(model6)
##
## Call:
## lm(formula = Petal.Width ~ Petal.Length + Species, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.7208 -0.1437  0.0005  0.1254  0.5460
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)         0.1597     0.0441    3.62  0.00047 ***
## Petal.Length        0.0664     0.0192    3.45  0.00084 ***
## Speciesversicolor   0.8938     0.0746   11.98  < 2e-16 ***
## Speciesvirginica    1.4903     0.1020   14.61  < 2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.205 on 96 degrees of freedom
## Multiple R-squared:  0.93,   Adjusted R-squared:  0.928
## F-statistic:  427 on 3 and 96 DF,  p-value: <2e-16
RMSE(predict(model6, test), test$Petal.Width)
## [1] 0.1885
plot(test[,"Petal.Width"], predict(model6, test),
  xlim=c(0,3), ylim=c(0,3),
  xlab = "actual", ylab = "predicted",
  main = "Petal.Width")
abline(0,1, col="red")
```




```
cor(test[,"Petal.Width"], predict(model6, test))
## [1] 0.9696
```

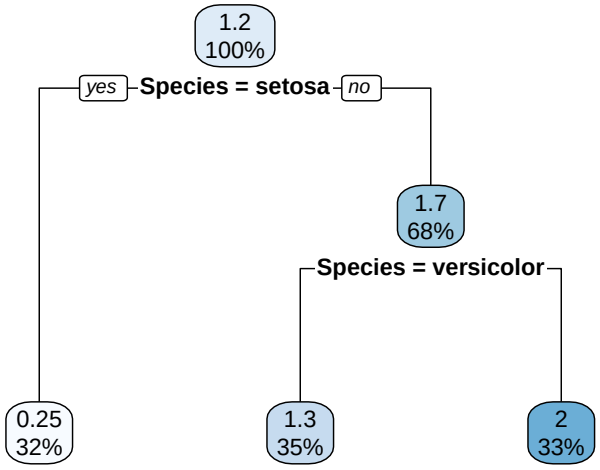
8.8 Alternative Regression Models

8.8.1 Regression Trees

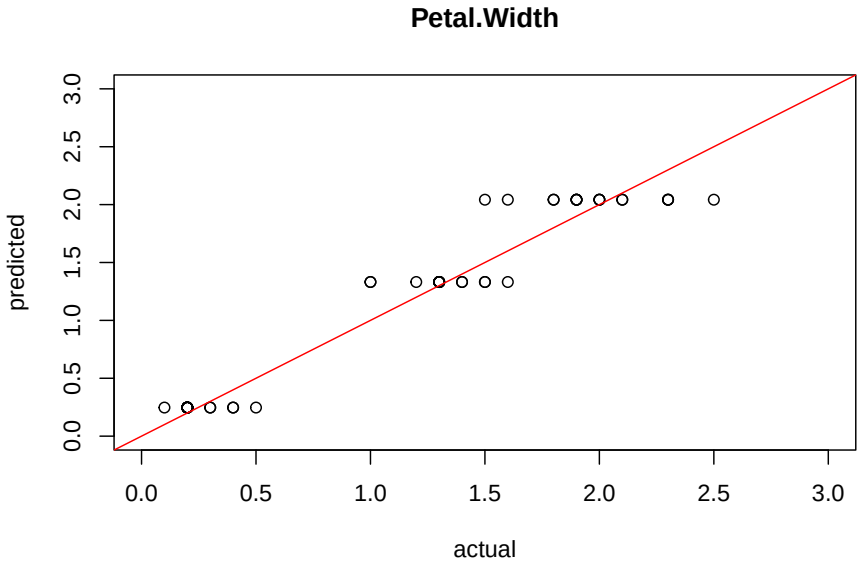
Many models we use for classification can also perform regression to produce piece-wise predictors. For example CART:

```
library(rpart)
library(rpart.plot)
model7 <- rpart(Petal.Width ~ ., data = train,
  control = rpart.control(cp = 0.01))
model7
## n= 100
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 100 57.5700 1.2190
##   2) Species=setosa 32  0.3797 0.2469 *
##   3) Species=versicolor,virginica 68 12.7200 1.6760
##     6) Species=versicolor 35  1.5350 1.3310 *
##     7) Species=virginica 33  2.6010 2.0420 *
```

```
rpart.plot(model7)
```




```
RMSE(predict(model7, test), test$Petal.Width)
## [1] 0.182
plot(test[,"Petal.Width"], predict(model7, test),
  xlim = c(0,3), ylim = c(0,3),
  xlab = "actual", ylab = "predicted",
  main = "Petal.Width")
abline(0,1, col = "red")
```




```
cor(test[,"Petal.Width"], predict(model7, test))
## [1] 0.9717
```

Note: This is not a nested model of the linear regressions so we cannot do ANOVA to compare the models!

8.8.2 Regularized Regression

LASSO and LAR try to reduce the number of parameters using a regularization term (see `lars` in package lars and https://en.wikipedia.org/wiki/Elastic_net_regularization)

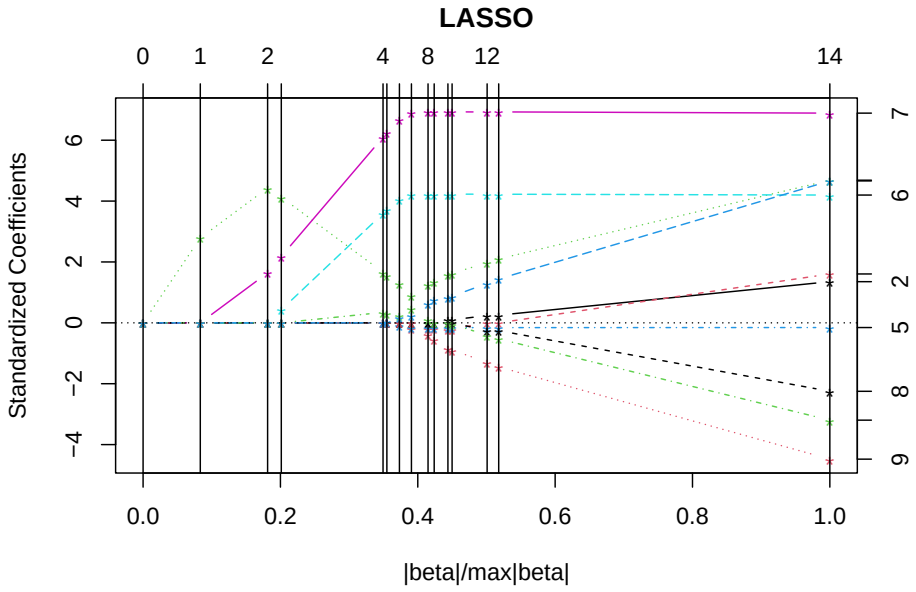
```
library(lars)
## Loaded lars 1.3
```

create a design matrix (with dummy variables and interaction terms). `lm` did this automatically for us, but for this `lars` implementation we have to do it manually.

```
x <- model.matrix(~ . + Sepal.Length*Sepal.Width*Petal.Length ,
  data = train[, -4])
head(x)
##     (Intercept) Sepal.Length Sepal.Width Petal.Length
## 85            1        2.980       1.464        5.227
## 104           1        5.096       3.044        5.187
## 30            1        4.361       2.832        1.861
## 53            1        8.125       2.406        5.526
```

```
## 143           1        6.372       1.565        6.147
## 142           1        6.526       3.697        5.708
##     useless Speciesversicolor Speciesvirginica
## 85    5.712                1                0
## 104   6.569                0                1
## 30    4.299                0                0
## 53    6.124                1                0
## 143   6.553                0                1
## 142   5.222                0                1
##     Sepal.Length:Sepal.Width Sepal.Length:Petal.Length
## 85                     4.362                    15.578
## 104                   15.511                    26.431
## 30                    12.349                     8.114
## 53                    19.546                    44.900
## 143                    9.972                    39.168
## 142                   24.126                    37.253
##     Sepal.Width:Petal.Length
## 85                     7.650
## 104                   15.787
## 30                     5.269
## 53                    13.294
## 143                    9.620
## 142                   21.103
##     Sepal.Length:Sepal.Width:Petal.Length
## 85                                  22.80
## 104                                 80.45
## 30                                  22.98
## 53                                 108.01
## 143                                 61.30
## 142                                137.72
y <- train[, 4]

model_lars <- lars(x, y)
summary(model_lars)
## LARS/LASSO
## Call: lars(x = x, y = y)
##    Df  Rss      Cp
## 0   1 57.6 1225.20
## 1   2 28.1  549.74
## 2   3 11.6  172.16
## 3   4 10.1  140.94
## 4   5  4.2    5.49
## 5   6  4.1    6.08
## 6   7  4.0    5.14
## 7   8  3.9    6.21
```

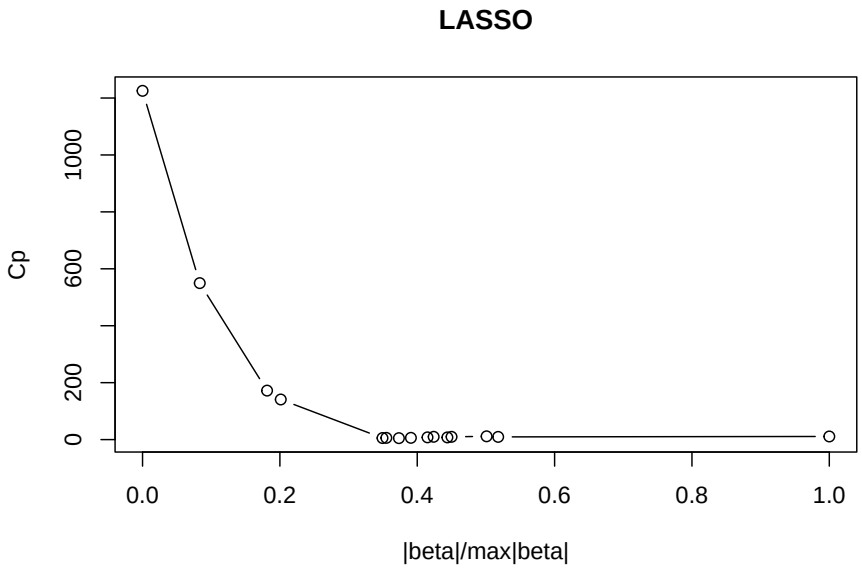
```
## 8   9  3.9    7.94
## 9  10  3.9    9.87
## 10  9  3.9    7.75
## 11 10  3.9    9.73
## 12 11  3.9   11.58
## 13 10  3.9    9.54
## 14 11  3.9   11.00
model_lars
##
## Call:
## lars(x = x, y = y)
## R-squared: 0.933
## Sequence of LASSO moves:
##      Petal.Length Speciesvirginica Speciesversicolor
## Var             4                7                 6
## Step            1                2                 3
##      Sepal.Width:Petal.Length
## Var                         10
## Step                         4
##      Sepal.Length:Sepal.Width:Petal.Length useless
## Var                                     11       5
## Step                                     5       6
##      Sepal.Width Sepal.Length:Petal.Length Sepal.Length
## Var            3                         9            2
## Step           7                         8            9
##      Sepal.Width:Petal.Length Sepal.Width:Petal.Length
## Var                       -10                        10
## Step                       10                        11
##      Sepal.Length:Sepal.Width Sepal.Width Sepal.Width
## Var                         8          -3           3
## Step                       12          13          14
plot(model_lars)
```



the plot shows how variables are added (from left to right to the model)

find best model (using Mallows's Cp statistic, see https://en.wikipedia.org/wiki/Mallows\XeTeXglyph\numexpr\XeTeXcharglyph"0027\relax{}s_Cp)

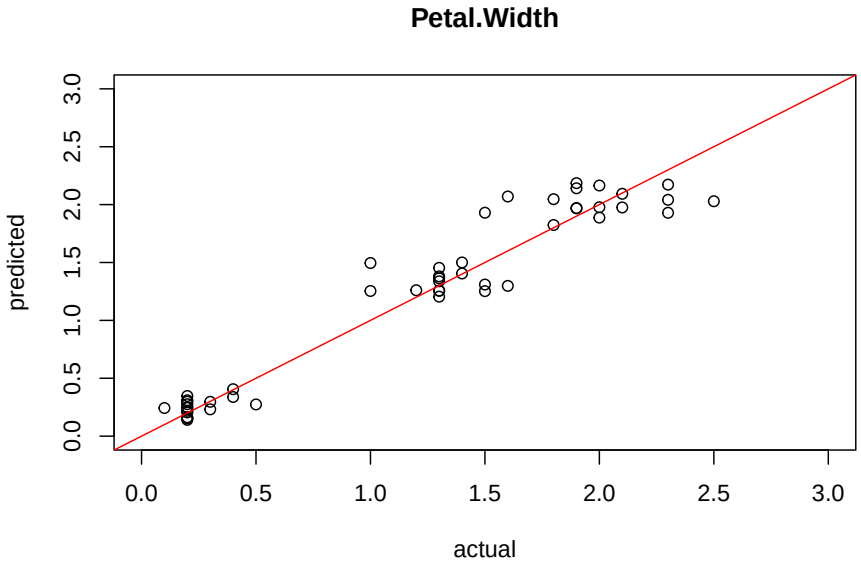
```
plot(model_lars, plottype = "Cp")
```




```
best <- which.min(model_lars$Cp)
coef(model_lars, s = best)
##                              (Intercept)
##                                0.0000000
##                             Sepal.Length
##                                0.0000000
##                              Sepal.Width
##                                0.0000000
##                             Petal.Length
##                                0.0603837
##                                  useless
##                               -0.0086551
##                      Speciesversicolor
##                                0.8463786
##                       Speciesvirginica
##                                1.4181850
##               Sepal.Length:Sepal.Width
##                                0.0000000
##              Sepal.Length:Petal.Length
##                                0.0000000
##               Sepal.Width:Petal.Length
##                                0.0029688
## Sepal.Length:Sepal.Width:Petal.Length
##                                0.0003151
```

make predictions

```
x_test <- model.matrix(~ . + Sepal.Length*Sepal.Width*Petal.Length,
  data = test[, -4])
predict(model_lars, x_test[1:5,], s = best)
## $s
## 6
## 7
##
## $fraction
##      6
## 0.4286
##
## $mode
## [1] "step"
##
## $fit
##    128     92     50    134      8
## 1.8237 1.5003 0.2505 1.9300 0.2439
test[1:5, ]$Petal.Width
## [1] 1.8 1.4 0.2 1.5 0.2
RMSE(predict(model_lars, x_test, s = best)$fit, test$Petal.Width)
## [1] 0.1907
plot(test[,"Petal.Width"],
     predict(model_lars, x_test, s = best)$fit,
     xlim=c(0,3), ylim=c(0,3),
     xlab = "actual", ylab = "predicted",
     main = "Petal.Width")
abline(0,1, col = "red")
```




```
cor(test[,"Petal.Width"],
    predict(model_lars, x_test, s = best)$fit)
## [1] 0.9686
```

8.8.3 Other Types of Regression

- Robust regression: robust against violation of assumptions like heteroscedasticity and outliers (`roblm` and `robglm` in package robustbase)
- Generalized linear models (`glm`). An example is logistic regression discussed in the next chapter.
- Nonlinear least squares (`nlm`)

8.9 Exercises

We will again use the Palmer penguin data for the exercises.

```
library(palmerpenguins)
head(penguins)
## # A tibble: 6 x 8
##   species island    bill_length_mm bill_depth_mm
##   <chr>   <chr>              <dbl>         <dbl>
## 1 Adelie  Torgersen           39.1          18.7
## 2 Adelie  Torgersen           39.5          17.4
## 3 Adelie  Torgersen           40.3          18
## 4 Adelie  Torgersen           NA            NA
```

```
## 5 Adelie  Torgersen           36.7          19.3
## 6 Adelie  Torgersen           39.3          20.6
## # i 4 more variables: flipper_length_mm <dbl>,
## #   body_mass_g <dbl>, sex <chr>, year <dbl>
```

Create an R markdown document that performs the following:

1. Create a linear regression model to predict the weight of a penguin (`body_mass_g`).
2. How high is the R-squared. What does it mean.
3. What variables are significant, what are not?
4. Use stepwise variable selection to remove unnecessary variables.
5. Predict the weight for the following new penguin:

```
new_penguin <- tibble(
  species = factor("Adelie",
    levels = c("Adelie", "Chinstrap", "Gentoo")),
  island = factor("Dream",
    levels = c("Biscoe", "Dream", "Torgersen")),
 bill_length_mm = 39.8,
 bill_depth_mm = 19.1,
 flipper_length_mm = 184,
 body_mass_g = NA,
 sex = factor("male", levels = c("female", "male")),
 year = 2007
)
new_penguin
## # A tibble: 1 x 8
##   species island bill_length_mm bill_depth_mm
##   <fct>   <fct>           <dbl>         <dbl>
## 1 Adelie  Dream            39.8          19.1
## # i 4 more variables: flipper_length_mm <dbl>,
## #   body_mass_g <lgl>, sex <fct>, year <dbl>
```

6. Create a regression tree. Look at the tree and explain what it does. Then use the regression tree to predict the weight for the above penguin.

Chapter 9

Logistic Regression*

This chapter introduces the popular classification method logistic regression.

Packages Used in this Chapter

This chapter only uses R's base functionality and does not need extra packages.

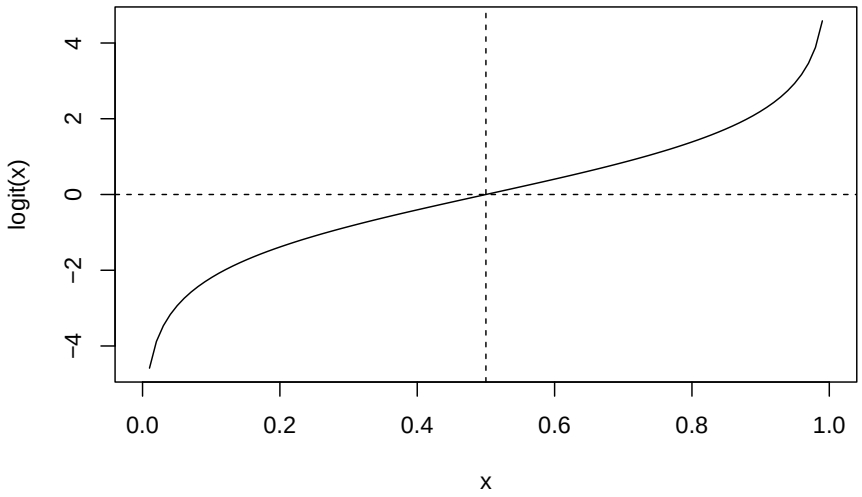
9.1 Introduction

Logistic regression contains the word regression, but it is actually a probabilistic statistical classification model to predict a binary outcome (a probability) given a set of features. It is a very powerful model that can be fit very quickly. It is one of the first classification models you should try on new data.

Logistic regression can be thought of as a linear regression with the log odds ratio (logit) of the binary outcome as the dependent variable:

$$logit(p) = ln(\frac{p}{1-p}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ...$$

```
logit  <- function(p) log(p/(1-p))
x <- seq(0, 1, length.out = 100)
plot(x, logit(x), type = "l")
abline(v=0.5, lty = 2)
abline(h=0, lty = 2)
```



This is equivalent to modeling the probability of the outcome p by

$$p = \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots}}{1+e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots}} = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots)}}$$

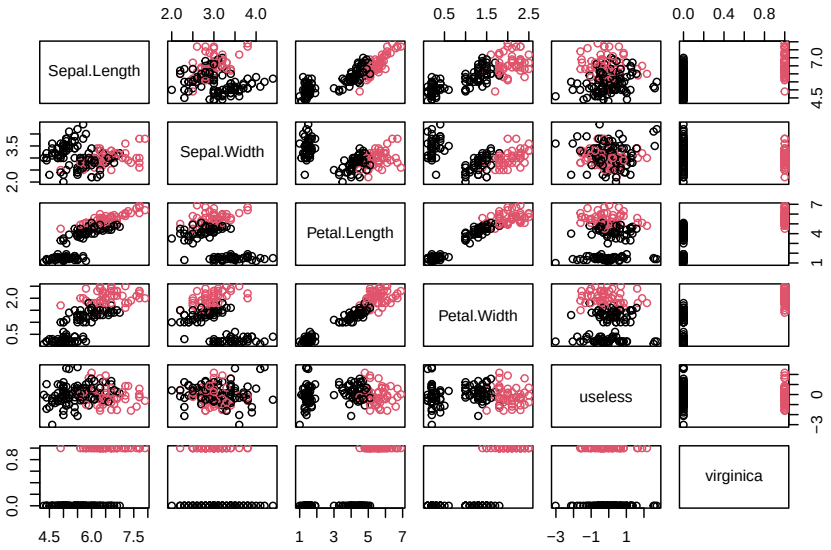
9.2 Data Preparation

Load and shuffle data. We also add a useless variable to see if the logistic regression removes it.

```
data(iris)
set.seed(100) # for reproducability

x <- iris[sample(1:nrow(iris)),]
x <- cbind(x, useless = rnorm(nrow(x)))
```

Make Species into a binary classification problem so we will classify if a flower is of species Virginica

```
x$virginica <- x$Species == "virginica"
x$Species <- NULL
plot(x, col=x$virginica+1)
```



9.3 Create a Logistic Regression Model

Logistic regression is a generalized linear model (GLM) with logit as the link function and a binomial error model.

```
model <- glm(virginica ~ .,
  family = binomial(logit), data=x)
## Warning: glm.fit: fitted probabilities numerically 0 or 1
## occurred
```

About the warning: glm.fit: fitted probabilities numerically 0 or 1 occurred means that the data is possibly linearly separable.

```
model
##
## Call:  glm(formula = virginica ~ ., family = binomial(logit), data = x)
##
## Coefficients:
##  (Intercept)  Sepal.Length   Sepal.Width  Petal.Length
##      -41.649        -2.531        -6.448         9.376
##  Petal.Width       useless
##       17.696         0.098
##
## Degrees of Freedom: 149 Total (i.e. Null);  144 Residual
## Null Deviance:       191
```

```
## Residual Deviance: 11.9  AIC: 23.9
```

Check which features are significant?

```
summary(model)
##
## Call:
## glm(formula = virginica ~ ., family = binomial(logit), data = x)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -41.649     26.556   -1.57    0.117
## Sepal.Length   -2.531      2.458   -1.03    0.303
## Sepal.Width    -6.448      4.794   -1.34    0.179
## Petal.Length    9.376      4.763    1.97    0.049 *
## Petal.Width    17.696     10.632    1.66    0.096 .
## useless         0.098      0.807    0.12    0.903
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 190.954  on 149  degrees of freedom
## Residual deviance:  11.884  on 144  degrees of freedom
## AIC: 23.88
##
## Number of Fisher Scoring iterations: 12
```

AIC can be used for model selection

9.4 Stepwise Variable Selection

```
model2 <- step(model, data = x)
## Start:  AIC=23.88
## virginica ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width +
##     useless
## Warning: glm.fit: fitted probabilities numerically 0 or 1
## occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1
## occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1
## occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1
## occurred
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1
## occurred
##                Df Deviance  AIC
## - useless       1     11.9 21.9
## - Sepal.Length  1     13.2 23.2
## <none>                11.9 23.9
## - Sepal.Width   1     14.8 24.8
## - Petal.Width   1     22.4 32.4
## - Petal.Length  1     25.9 35.9
## Warning: glm.fit: fitted probabilities numerically 0 or 1
## occurred
##
## Step:  AIC=21.9
## virginica ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width
## Warning: glm.fit: fitted probabilities numerically 0 or 1
## occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1
## occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1
## occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1
## occurred
##                Df Deviance  AIC
## - Sepal.Length  1     13.3 21.3
## <none>                11.9 21.9
## - Sepal.Width   1     15.5 23.5
## - Petal.Width   1     23.8 31.8
## - Petal.Length  1     25.9 33.9
## Warning: glm.fit: fitted probabilities numerically 0 or 1
## occurred
##
## Step:  AIC=21.27
## virginica ~ Sepal.Width + Petal.Length + Petal.Width
## Warning: glm.fit: fitted probabilities numerically 0 or 1
## occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1
## occurred
##                Df Deviance  AIC
## <none>                13.3 21.3
## - Sepal.Width   1     20.6 26.6
## - Petal.Length  1     27.4 33.4
## - Petal.Width   1     31.5 37.5
summary(model2)
##
## Call:
```

```
## glm(formula = virginica ~ Sepal.Width + Petal.Length + Petal.Width,
##     family = binomial(logit), data = x)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -50.53      23.99   -2.11    0.035 *
## Sepal.Width     -8.38       4.76   -1.76    0.079 .
## Petal.Length     7.87       3.84    2.05    0.040 *
## Petal.Width     21.43      10.71    2.00    0.045 *
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 190.954  on 149  degrees of freedom
## Residual deviance:  13.266  on 146  degrees of freedom
## AIC: 21.27
##
## Number of Fisher Scoring iterations: 12
```

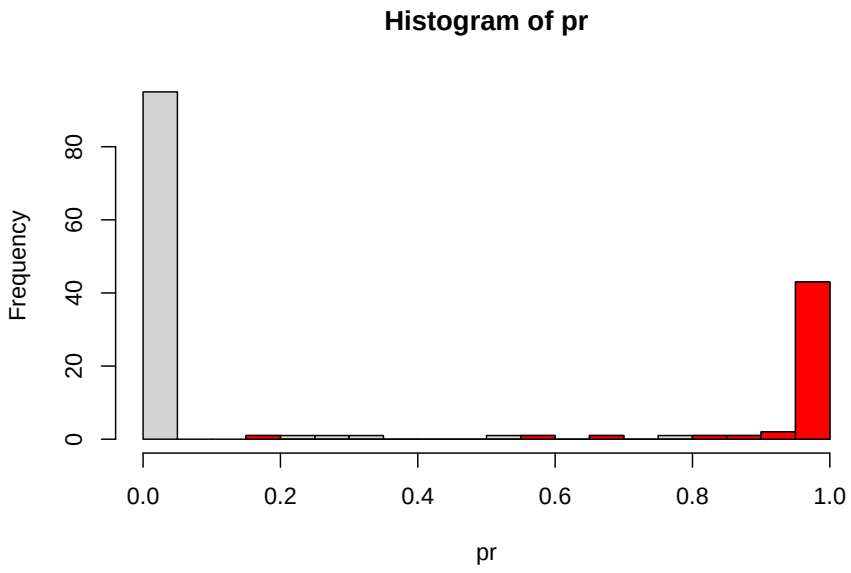
The estimates $(\beta_0, \beta_1, \dots)$ are log-odds and can be converted into odds using $exp(\beta)$. A negative log-odds ratio means that the odds go down with an increase in the value of the predictor. A predictor with a positive log-odds ratio increases the odds. In this case, the odds of looking at a Virginica iris goes down with Sepal.Width and increases with the other two predictors.

9.5 Calculate the Response

Note: we do here in-sample testing on the data we learned the data from. To get a generalization error estimate you should use a test set or cross-validation!

```
pr <- predict(model2, x, type="response")
round(pr, 2)
##  102  112    4   55   70   98  135    7   43  140   51   25
## 1.00 1.00 0.00 0.00 0.00 0.00 0.86 0.00 0.00 1.00 0.00 0.00
##    2   68  137   48   32   85   91  121   16  116   66  146
## 0.00 0.00 1.00 0.00 0.00 0.00 0.00 1.00 0.00 1.00 0.00 1.00
##   93   45   30  124  126   87   95   97  120   29   92   31
## 0.00 0.00 0.00 0.98 1.00 0.00 0.00 0.00 0.93 0.00 0.00 0.00
##   54   41  105  113   24  142  143   63   65    9  150   20
## 0.00 0.00 1.00 1.00 0.00 1.00 1.00 0.00 0.00 0.00 0.96 0.00
##   14   78   88    3   36   27   46   59   96   69   47  147
## 0.00 0.54 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.20 0.00 1.00
##  129  136   12  141  130   56   22   82   53   99    5   44
```

```
## 1.00 1.00 0.00 1.00 0.99 0.00 0.00 0.00 0.00 0.00 0.00 0.00
##   28   52  139   42   15   57   75   37   26  110  100  149
## 0.00 0.00 0.67 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 1.00
##  132  107   35   58  127  111  144   86  114   71  123  119
## 1.00 0.60 0.00 0.00 0.92 1.00 1.00 0.00 1.00 0.28 1.00 1.00
##   18    8  128   83  138   19  115   23   89   62   80  104
## 0.00 0.00 0.82 0.00 1.00 0.00 1.00 0.00 0.00 0.00 0.00 1.00
##   40   17   94  133   60   81  118  125  122   49  148   61
## 0.00 0.00 0.00 1.00 0.00 0.00 1.00 1.00 1.00 0.00 1.00 0.00
##   10  109  106   72   13   77   79   39  134   84   67  117
## 0.00 1.00 1.00 0.00 0.00 0.00 0.00 0.00 0.16 0.79 0.00 1.00
##  108  101  103   76    1   50  131   90   34   38    6   64
## 1.00 1.00 1.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00
##   33  145   74   11   21   73
## 0.00 1.00 0.00 0.00 0.00 0.32
hist(pr, breaks=20)
hist(pr[x$virginica==TRUE], col="red", breaks=20, add=TRUE)
```



9.6 Check Classification Performance

We calculate the predicted class by checking if the probability is larger than .5.

```
pred <- pr > .5
```

Now er can create a confusion table and calculate the accuracy.

```
tbl <- table(actual = x$virginica, predicted = pr>.5)
tbl
##        predicted
## actual  FALSE TRUE
##   FALSE    98    2
##   TRUE      1   49
sum(diag(tbl))/sum(tbl)
## [1] 0.98
```

We can also use caret's more advanced function `confusionMatrix()`. Our code above uses `logical` vectors. but foo caret, we need to make sure that both, the reference and the predictions are coded as `factor`.

```
caret::confusionMatrix(
  reference = factor(x$virginica, levels = c(TRUE, FALSE)),
  data = factor(pr>.5, levels = c(TRUE, FALSE)))
## Confusion Matrix and Statistics
##
##           Reference
## Prediction TRUE FALSE
##      TRUE    49     2
##      FALSE    1    98
##
##                Accuracy : 0.98
##                  95% CI : (0.943, 0.996)
##     No Information Rate : 0.667
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.955
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.980
##             Specificity : 0.980
##          Pos Pred Value : 0.961
##          Neg Pred Value : 0.990
##              Prevalence : 0.333
##          Detection Rate : 0.327
##    Detection Prevalence : 0.340
##       Balanced Accuracy : 0.980
##
##        'Positive' Class : TRUE
##
```

We see that the model performs well with a very high accuracy and kappa value.

9.7 Regularized Logistic Regression

Glmnet fits generalized linear models (including logistic regression) using regularization via penalized maximum likelihood. The regularization parameter λ is a hyperparameter and glmnet can use cross-validation to find an appropriate value. glmnet does not have a function interface, so we have to supply a matrix for `X` and a vector of responses for `y`.

```
library(glmnet)
## Loaded glmnet 4.1-8
X <- as.matrix(x[, 1:5])
y <- x$virginica

fit <- cv.glmnet(X, y, family = "binomial")
fit
##
## Call:  cv.glmnet(x = X, y = y, family = "binomial")
##
## Measure: Binomial Deviance
##
##      Lambda Index Measure     SE Nonzero
## min 0.00164    59   0.126 0.0456       5
## 1se 0.00664    44   0.167 0.0422       3
```

There are several selection rules for lambda, we look at the coefficients of the logistic regression using the lambda that gives the most regularized model such that the cross-validated error is within one standard error of the minimum cross-validated error.

```
coef(fit, s = fit$lambda.1se)
## 6 x 1 sparse Matrix of class "dgCMatrix"
##                    s1
## (Intercept)  -16.961
## Sepal.Length   .
## Sepal.Width   -1.766
## Petal.Length   2.197
## Petal.Width    6.820
## useless        .
```

A dot means 0. We see that the predictors Sepal.Length and useless are not used in the prediction giving a models similar to stepwise variable selection above.

A predict function is provided. We need to specify what regularization to use and that we want to predict a class label.

```
predict(fit, newx = X[1:5,], s = fit$lambda.1se, type = "class")
##     s1
## 102 "TRUE"
```

```
## 112 "TRUE"
## 4   "FALSE"
## 55  "FALSE"
## 70  "FALSE"
```

Glmnet provides supports many types of generalized linear models. Examples can be found in the article An Introduction to glmnet[1].

9.8 Exercises

We will again use the Palmer penguin data for the exercises.

```
library(palmerpenguins)
head(penguins)
## # A tibble: 6 x 8
##   species island    bill_length_mm bill_depth_mm
##   <chr>   <chr>              <dbl>         <dbl>
## 1 Adelie  Torgersen           39.1          18.7
## 2 Adelie  Torgersen           39.5          17.4
## 3 Adelie  Torgersen           40.3          18
## 4 Adelie  Torgersen           NA            NA
## 5 Adelie  Torgersen           36.7          19.3
## 6 Adelie  Torgersen           39.3          20.6
## # i 4 more variables: flipper_length_mm <dbl>,
## #   body_mass_g <dbl>, sex <chr>, year <dbl>
```

Create an R markdown document that performs the following:

1. Create a test and a training data set (see section Holdout Method in Chapter 3).
2. Create a logistic regression using the training set to predict the variable sex.
3. Use stepwise variable selection. What variables are selected?
4. What do the parameters for for each of the selected features tell you?
5. Predict the sex of the penguins in the test set. Create a confusion table and calculate the accuracy and discuss how well the model works.

[1] https://glmnet.stanford.edu/articles/glmnet.html

References

Agrawal, Rakesh, Tomasz Imielinski, and Arun Swami. 1993. "Mining Association Rules Between Sets of Items in Large Databases." In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, 207–16. Washington, D.C., United States: ACM Press.

Bates, Douglas, Martin Maechler, and Mikael Jagan. 2024. *Matrix: Sparse and Dense Matrix Classes and Methods*. https://Matrix.R-forge.R-project.org.

Blake, Catherine L., and Christopher J. Merz. 1998. *UCI Repository of Machine Learning Databases*. Irvine, CA: University of California, Irvine, Department of Information; Computer Sciences.

Breiman, Leo, Adele Cutler, Andy Liaw, and Matthew Wiener. 2024. *randomForest: Breiman and Cutlers Random Forests for Classification and Regression*. https://www.stat.berkeley.edu/~breiman/RandomForests/.

Buchta, Christian, and Michael Hahsler. 2024. *arulesSequences: Mining Frequent Sequences*. https://CRAN.R-project.org/package=arulesSequences.

Carr, Dan, Nicholas Lewin-Koh, and Martin Maechler. 2024. *Hexbin: Hexagonal Binning Routines*. https://github.com/edzer/hexbin.

Chen, Tianqi, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, et al. 2024. *Xgboost: Extreme Gradient Boosting*. https://github.com/dmlc/xgboost.

Chen, Ying-Ju, Fadel M. Megahed, L. Allison Jones-Farmer, and Steven E. Rigdon. 2023. *Basemodels: Baseline Models for Classification and Regression*. https://github.com/Ying-Ju/basemodels.

Fraley, Chris, Adrian E. Raftery, and Luca Scrucca. 2024. *Mclust: Gaussian Mixture Modelling for Model-Based Clustering, Classification, and Density Estimation*. https://mclust-org.github.io/mclust/.

Friedman, Jerome, Trevor Hastie, Rob Tibshirani, Balasubramanian Narasimhan, Kenneth Tay, Noah Simon, and James Yang. 2023. *Glmnet: Lasso and Elastic-Net Regularized Generalized Linear Models*. https://glmnet.stanford.edu.

Friedman, Jerome, Robert Tibshirani, and Trevor Hastie. 2010. "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software* 33 (1): 1–22. https://doi.org/10.18637/jss.v033.i01.

Grolemund, Garrett, and Hadley Wickham. 2011. "Dates and Times Made Easy with lubridate." *Journal of Statistical Software* 40 (3): 1–25. https:

//www.jstatsoft.org/v40/i03/.

Hahsler, Michael. 2017a. "An Experimental Comparison of Seriation Methods for One-Mode Two-Way Data." *European Journal of Operational Research* 257 (1): 133–43. https://doi.org/10.1016/j.ejor.2016.08.066.

———. 2017b. "ArulesViz: Interactive Visualization of Association Rules with R." *R Journal* 9 (2): 163–75. https://doi.org/10.32614/RJ-2017-047.

———. 2021. *An R Companion for Introduction to Data Mining*. Online Book. https://mhahsler.github.io/Introduction_to_Data_Mining_R_Examples/book.

———. 2024. *arulesViz: Visualizing Association Rules and Frequent Itemsets*. https://github.com/mhahsler/arulesViz.

Hahsler, Michael, Christian Buchta, Bettina Gruen, and Kurt Hornik. 2024. *Arules: Mining Association Rules and Frequent Itemsets*. https://github.com/mhahsler/arules.

Hahsler, Michael, Christian Buchta, and Kurt Hornik. 2024. *Seriation: Infrastructure for Ordering Objects Using Seriation*. https://github.com/mhahsler/seriation.

Hahsler, Michael, Sudheer Chelluboina, Kurt Hornik, and Christian Buchta. 2011. "The Arules r-Package Ecosystem: Analyzing Interesting Patterns from Large Transaction Datasets." *Journal of Machine Learning Research* 12: 1977–81. https://jmlr.csail.mit.edu/papers/v12/hahsler11a.html.

Hahsler, Michael, Bettina Gruen, and Kurt Hornik. 2005. "Arules – A Computational Environment for Mining Association Rules and Frequent Item Sets." *Journal of Statistical Software* 14 (15): 1–25. https://doi.org/10.18637/jss.v014.i15.

Hahsler, Michael, Bettina Grün, and Kurt Hornik. 2005. "Arules – A Computational Environment for Mining Association Rules and Frequent Item Sets." *Journal of Statistical Software* 14 (15): 1–25. http://www.jstatsoft.org/v14/i15/.

Hahsler, Michael, Kurt Hornik, and Christian Buchta. 2008. "Getting Things in Order: An Introduction to the r Package Seriation." *Journal of Statistical Software* 25 (3): 1–34. https://doi.org/10.18637/jss.v025.i03.

Hahsler, Michael, and Matthew Piekenbrock. 2024. *Dbscan: Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and Related Algorithms*. https://github.com/mhahsler/dbscan.

Hahsler, Michael, Matthew Piekenbrock, and Derek Doran. 2019. "dbscan: Fast Density-Based Clustering with R." *Journal of Statistical Software* 91 (1): 1–30. https://doi.org/10.18637/jss.v091.i01.

Hastie, Trevor, and Brad Efron. 2022. *Lars: Least Angle Regression, Lasso and Forward Stagewise*. https://doi.org/10.1214/009053604000000067.

Hennig, Christian. 2024. *Fpc: Flexible Procedures for Clustering*. https://www.unibo.it/sitoweb/christian.hennig/en/.

Hornik, Kurt. 2023. *RWeka: R/Weka Interface*. https://CRAN.R-project.org/package=RWeka.

Hornik, Kurt, Christian Buchta, and Achim Zeileis. 2009. "Open-Source Machine Learning: R Meets Weka." *Computational Statistics* 24 (2): 225–32.

https://doi.org/10.1007/s00180-008-0119-7.

Horst, Allison, Alison Hill, and Kristen Gorman. 2022. *Palmerpenguins: Palmer Archipelago (Antarctica) Penguin Data*. https://allisonhorst.github.io/palmerpenguins/.

Hothorn, Torsten, Peter Buehlmann, Sandrine Dudoit, Annette Molinaro, and Mark Van Der Laan. 2006. "Survival Ensembles." *Biostatistics* 7 (3): 355–73. https://doi.org/10.1093/biostatistics/kxj011.

Hothorn, Torsten, Kurt Hornik, Carolin Strobl, and Achim Zeileis. 2024. *Party: A Laboratory for Recursive Partytioning*. http://party.R-forge.R-project.org.

Hothorn, Torsten, Kurt Hornik, and Achim Zeileis. 2006. "Unbiased Recursive Partitioning: A Conditional Inference Framework." *Journal of Computational and Graphical Statistics* 15 (3): 651–74. https://doi.org/10.1198/106186006X133933.

Kalinowski, Tomasz, JJ Allaire, and François Chollet. 2024. *Keras3: R Interface to Keras*. https://keras3.posit.co/.

Karatzoglou, Alexandros, Alex Smola, and Kurt Hornik. 2024. *Kernlab: Kernel-Based Machine Learning Lab*. https://CRAN.R-project.org/package=kernlab.

Karatzoglou, Alexandros, Alex Smola, Kurt Hornik, and Achim Zeileis. 2004. "Kernlab – an S4 Package for Kernel Methods in R." *Journal of Statistical Software* 11 (9): 1–20. https://doi.org/10.18637/jss.v011.i09.

Kassambara, Alboukadel. 2023. *Ggcorrplot: Visualization of a Correlation Matrix Using Ggplot2*. http://www.sthda.com/english/wiki/ggcorrplot-visualization-of-a-correlation-matrix-using-ggplot2.

Kassambara, Alboukadel, and Fabian Mundt. 2020. *Factoextra: Extract and Visualize the Results of Multivariate Data Analyses*. http://www.sthda.com/english/rpkgs/factoextra.

Kuhn, Max. 2023. *Caret: Classification and Regression Training*. https://github.com/topepo/caret/.

Kuhn, Max, and Ross Quinlan. 2023. *C50: C5.0 Decision Trees and Rule-Based Models*. https://topepo.github.io/C5.0/.

Kuhn, and Max. 2008. "Building Predictive Models in r Using the Caret Package." *Journal of Statistical Software* 28 (5): 1–26. https://doi.org/10.18637/jss.v028.i05.

Leisch, Friedrich, and Evgenia Dimitriadou. 2024. *Mlbench: Machine Learning Benchmark Problems*. https://CRAN.R-project.org/package=mlbench.

Liaw, Andy, and Matthew Wiener. 2002. "Classification and Regression by randomForest." *R News* 2 (3): 18–22. https://CRAN.R-project.org/doc/Rnews/.

Maechler, Martin, Peter Rousseeuw, Anja Struyf, and Mia Hubert. 2023. *Cluster: "Finding Groups in Data": Cluster Analysis Extended Rousseeuw Et Al.* https://svn.r-project.org/R-packages/trunk/cluster/.

Meyer, David, and Christian Buchta. 2022. *Proxy: Distance and Similarity Measures*. https://CRAN.R-project.org/package=proxy.

Meyer, David, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and

Friedrich Leisch. 2024. *E1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. https://CRAN.R-project.org/package=e1071.

Milborrow, Stephen. 2024. *Rpart.plot: Plot Rpart Models: An Enhanced Version of Plot.rpart*. http://www.milbo.org/rpart-plot/index.html.

Müller, Kirill, and Hadley Wickham. 2023. *Tibble: Simple Data Frames*. https://tibble.tidyverse.org/.

R Core Team. 2024. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Ripley, Brian. 2023. *Nnet: Feed-Forward Neural Networks and Multinomial Log-Linear Models*. http://www.stats.ox.ac.uk/pub/MASS4/.

Ripley, Brian, and Bill Venables. 2024. *MASS: Support Functions and Datasets for Venables and Ripley's MASS*. http://www.stats.ox.ac.uk/pub/MASS4/.

Robin, Xavier, Natacha Turck, Alexandre Hainard, Natalia Tiberti, Frédérique Lisacek, Jean-Charles Sanchez, and Markus Müller. 2011. "pROC: An Open-Source Package for r and s+ to Analyze and Compare ROC Curves." *BMC Bioinformatics* 12: 77.

———. 2023. *pROC: Display and Analyze ROC Curves*. https://xrobin.github.io/pROC/.

Roever, Christian, Nils Raabe, Karsten Luebke, Uwe Ligges, Gero Szepannek, Marc Zentgraf, and David Meyer. 2023. *klaR: Classification and Visualization*. https://statistik.tu-dortmund.de.

Romanski, Piotr, Lars Kotthoff, and Patrick Schratz. 2023. *FSelector: Selecting Attributes*. https://github.com/larskotthoff/fselector.

Sarkar, Deepayan. 2008. *Lattice: Multivariate Data Visualization with r*. New York: Springer. http://lmdvr.r-forge.r-project.org.

———. 2023. *Lattice: Trellis Graphics for r*. https://lattice.r-forge.r-project.org/.

Schloerke, Barret, Di Cook, Joseph Larmarange, Francois Briatte, Moritz Marbach, Edwin Thoen, Amos Elberg, and Jason Crowley. 2024. *GGally: Extension to Ggplot2*. https://ggobi.github.io/ggally/.

Scrucca, Luca, Chris Fraley, T. Brendan Murphy, and Adrian E. Raftery. 2023. *Model-Based Clustering, Classification, and Density Estimation Using mclust in R*. Chapman; Hall/CRC. https://doi.org/10.1201/9781003277965.

Sievert, Carson. 2020. *Interactive Web-Based Data Visualization with r, Plotly, and Shiny*. Chapman; Hall/CRC. https://plotly-r.com.

Sievert, Carson, Chris Parmer, Toby Hocking, Scott Chamberlain, Karthik Ram, Marianne Corvellec, and Pedro Despouy. 2024. *Plotly: Create Interactive Web Graphics via Plotly.js*. https://plotly-r.com.

Simon, Noah, Jerome Friedman, Robert Tibshirani, and Trevor Hastie. 2011. "Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent." *Journal of Statistical Software* 39 (5): 1–13. https://doi.org/10.18637/jss.v039.i05.

Spinu, Vitalie, Garrett Grolemund, and Hadley Wickham. 2023. *Lubridate: Make Dealing with Dates a Little Easier*. https://lubridate.tidyverse.org.

Strobl, Carolin, Anne-Laure Boulesteix, Thomas Kneib, Thomas Augustin, and Achim Zeileis. 2008. "Conditional Variable Importance for Random Forests." *BMC Bioinformatics* 9 (307). https://doi.org/10.1186/1471-2105-9-307.

Strobl, Carolin, Anne-Laure Boulesteix, Achim Zeileis, and Torsten Hothorn. 2007. "Bias in Random Forest Variable Importance Measures: Illustrations, Sources and a Solution." *BMC Bioinformatics* 8 (25). https://doi.org/10.1186/1471-2105-8-25.

Tan, Pang-Ning, Michael S. Steinbach, Anuj Karpatne, and Vipin Kumar. 2017. *Introduction to Data Mining*. 2nd Edition. Pearson. https://www-users.cs.umn.edu/~kumar001/dmbook.

Tan, Pang-Ning, Michael S. Steinbach, and Vipin Kumar. 2005. *Introduction to Data Mining*. 1st Edition. Addison-Wesley. https://www-users.cs.umn.edu/~kumar001/dmbook/firsted.php.

Tay, J. Kenneth, Balasubramanian Narasimhan, and Trevor Hastie. 2023. "Elastic Net Regularization Paths for All Generalized Linear Models." *Journal of Statistical Software* 106 (1): 1–31. https://doi.org/10.18637/jss.v106.i01.

Therneau, Terry, and Beth Atkinson. 2023. *Rpart: Recursive Partitioning and Regression Trees*. https://github.com/bethatkinson/rpart.

Tillé, Yves, and Alina Matei. 2023. *Sampling: Survey Sampling*. https://CRAN.R-project.org/package=sampling.

Venables, W. N., and B. D. Ripley. 2002a. *Modern Applied Statistics with s*. Fourth. New York: Springer. https://www.stats.ox.ac.uk/pub/MASS4/.

———. 2002b. *Modern Applied Statistics with s*. Fourth. New York: Springer. https://www.stats.ox.ac.uk/pub/MASS4/.

Venables, W. N., D. M. Smith, and the R Core Team. 2021. *An Introduction to R*.

Weihs, Claus, Uwe Ligges, Karsten Luebke, and Nils Raabe. 2005. "klaR Analyzing German Business Cycles." In *Data Analysis and Decision Support*, edited by D. Baier, R. Decker, and L. Schmidt-Thieme, 335–43. Berlin: Springer-Verlag.

Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org.

———. 2023a. *Forcats: Tools for Working with Categorical Variables (Factors)*. https://forcats.tidyverse.org/.

———. 2023b. *Stringr: Simple, Consistent Wrappers for Common String Operations*. https://stringr.tidyverse.org.

———. 2023c. *Tidyverse: Easily Install and Load the Tidyverse*. https://tidyverse.tidyverse.org.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.

Wickham, Hadley, Mine Çetinkaya-Rundel, and Garrett Grolemund. 2023. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. 2st ed. O'Reilly Media, Inc. https://r4ds.hadley.nz/.

Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske

Takahashi, Claus Wilke, Kara Woo, Hiroaki Yutani, Dewey Dunnington, and Teun van den Brand. 2024. *Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. https://ggplot2.tidyverse.org.

Wickham, Hadley, Romain François, Lionel Henry, Kirill Müller, and Davis Vaughan. 2023. *Dplyr: A Grammar of Data Manipulation*. https://dplyr.tidyverse.org.

Wickham, Hadley, and Lionel Henry. 2023. *Purrr: Functional Programming Tools*. https://purrr.tidyverse.org/.

Wickham, Hadley, Jim Hester, and Jennifer Bryan. 2024. *Readr: Read Rectangular Text Data*. https://readr.tidyverse.org.

Wickham, Hadley, Thomas Lin Pedersen, and Dana Seidel. 2023. *Scales: Scale Functions for Visualization*. https://scales.r-lib.org.

Wickham, Hadley, Davis Vaughan, and Maximilian Girlich. 2024. *Tidyr: Tidy Messy Data*. https://tidyr.tidyverse.org.

Wilkinson, Leland. 2005. *The Grammar of Graphics (Statistics and Computing)*. Berlin, Heidelberg: Springer-Verlag. https://doi.org/10.1007/0-387-28695-0.

Witten, Ian H., and Eibe Frank. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd ed. San Francisco: Morgan Kaufmann.

Yu, Guangchuang. 2024. *Scatterpie: Scatter Pie Plot*. https://CRAN.R-project.org/package=scatterpie.

Zaki, Mohammed J. 2000. "Sequence Mining in Categorical Domains: Incorporating Constraints." In *Proceedings of the Ninth International Conference on Information and Knowledge Management*, 422–29. CIKM '00. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/354756.354849.

Zeileis, Achim, Torsten Hothorn, and Kurt Hornik. 2008. "Model-Based Recursive Partitioning." *Journal of Computational and Graphical Statistics* 17 (2): 492–514. https://doi.org/10.1198/106186008X319331.