

# **RStatsbook**

Andreas Busjahn

2026-01-13

# Table of contents

<b>Preface</b>	<b>8</b>
<b>1 Introduction</b>	<b>9</b>
1.1 Installation of R . . . . .	9
1.2 Installation of RStudio Desktop (UI) . . . . .	9
1.3 Installation of Quarto (Reportgenerator) . . . . .	9
1.4 Installation of git: . . . . .	9
1.5 Setup RStudio . . . . .	10
1.6 Projects on your computer . . . . .	11
1.6.1 Option 1: Create manually . . . . .	11
1.6.2 Option 2: Create from a github repository . . . . .	12
<b>2 Some useful resources</b>	<b>13</b>
2.1 Books . . . . .	13
2.2 Ressources . . . . .	13
<b>3 Syntax rules / basic things to know about R</b>	<b>14</b>
3.1 <i>Script preparation / basic setup</i> . . . . .	14
3.2 <i>Numeric operations</i> . . . . .	15
3.3 <i>Variables</i> . . . . .	16
3.3.1 <i>Variable names</i> . . . . .	16
3.3.2 <i>Basic classes of data</i> . . . . .	16
3.3.3 <i>Exercise: variables / Assignments</i> . . . . .	19
3.3.4 <i>Exercise: Factors</i> . . . . .	21
3.3.5 <i>Indexing variables</i> . . . . .	25
3.3.6 <i>Usage of variables</i> . . . . .	26
3.4 <i>Functions</i> . . . . .	27
3.4.1 <i>Function usage</i> . . . . .	27
3.4.2 <i>Functions combined</i> . . . . .	29
3.4.3 <i>Writing functions</i> . . . . .	31
3.5 <i>More complex data types, created by functions</i> . . . . .	34
3.5.1 <i>Matrix</i> . . . . .	34
3.6 <i>Exercise: Matrix</i> . . . . .	36
3.6.1 <i>Data frame</i> . . . . .	37
3.6.2 <i>Tibble</i> . . . . .	40
3.6.3 <i>Exercise: tibble</i> . . . . .	51
3.6.4 <i>List</i> . . . . .	52

3.7	<i>Control structures</i>	55
3.7.1	<i>Loops</i>	55
3.7.2	<i>Conditions</i>	59
3.7.3	<i>Exercise: for and while loops, and if/ifelse/case_xxx</i>	63
<b>4</b>	<b>Regular expressions</b>	<b>66</b>
4.1	Intro	66
4.1.1	Some basic examples:	66
4.1.2	An example for their use in renaming variables:	68
4.2	Exercise	69
<b>5</b>	<b>Importing data</b>	<b>70</b>
5.1	Import from text files (.txt, .csv)	70
5.2	Import from Excel	71
5.2.1	Tidy Excel files	71
5.2.2	Excel file with units row	71
5.3	ODS files	73
5.3.1	Import from SPSS/SAS	73
<b>6</b>	<b>Changing structure wide &lt;-&gt; long</b>	<b>76</b>
6.1	Example 1: single repeated measure	77
6.2	Example 2: several repeated measures	77
6.3	Example 3: long to wide	79
6.4	More examples	80
6.4.1	Step 1: Create example data:	80
6.4.2	Step 2: Transform that data to a long form:	81
6.4.3	Step 3 Transform long to wide	82
6.5	Exercise: Reshaping Data	83
6.5.1	Part 1: From Long to Wide ( <code>pivot_wider()</code> )	83
<b>7</b>	<b>Visualize data with ggplot</b>	<b>85</b>
7.1	Example data	85
7.2	Basic structure of a ggplot call	86
7.3	fill vs. color	91
7.4	Color systems	93
7.4.1	External color definitions from ggsci	96
7.5	Exporting ggplots	99
7.6	Other geoms	99
7.7	Exercise 1	109
7.8	Combining and finetuning aesthetics	109
7.9	Positioning elements	117
7.10	Order of layers	122
7.11	Local aesthetics for layers	126
7.12	Faceting (splitting) plots	128
7.12.1	<code>facet_grid</code>	128
7.12.2	<code>facet_wrap</code>	131

7.12.3	Controlling scales in facets (default: scales=“fixed”)	134
7.13	Exercise 2	136
7.14	Showing summaries	136
7.15	Indicating significances	143
7.16	Theme definitions / changes	149
7.17	Combining figures with patchwork	154
7.18	ggplots in loops	156
<b>8</b>	<b>Grouping of variables by type / distribution / use</b>	<b>161</b>
8.1	Test for Normal distribution	161
8.1.1	Testing a single variable	161
8.1.2	Testing several variables	164
8.2	Exercise: Distribution of penguin measures	169
8.3	Picking column names and positions	169
<b>9</b>	<b>Descriptive statistics</b>	<b>171</b>
9.1	Typical descriptives	171
9.2	Reading in data	171
9.3	Graphical exploration should start before descriptive statistics	171
9.4	Gaussian variables	172
9.5	A little theory	172
9.5.1	Simple function calls	173
9.5.2	Combined reporting	174
9.6	Ordinal variables	175
9.7	Categorical variables	177
9.8	Summarize data	179
<b>10</b>	<b>Exercises</b>	<b>185</b>
<b>11</b>	<b>Summarize / across</b>	<b>186</b>
<b>12</b>	<b>Simple test statistics</b>	<b>194</b>
12.1	Tests require hypotheses	195
12.1.1	Null hypothesis ?	196
12.2	Quantitative measures with Gaussian distribution	196
12.3	Ordinal data	205
12.4	Categorial data	208
<b>13</b>	<b>Exercise</b>	<b>212</b>
<b>14</b>	<b>Covariance / Correlation</b>	<b>213</b>
14.1	Covariance	213
14.2	Correlation	213
14.3	Vizualizations	215
<b>15</b>	<b>Intro to linear models</b>	<b>220</b>
15.1	Setup	220

15.2 Import / Preparation . . . . .	220
15.3 Graphical exploration . . . . .	221
15.4 Linear Models . . . . .	224
15.4.1 Linear regression . . . . .	224
15.4.2 ANOVA . . . . .	229
15.4.3 LM with continuous AND categorical IV . . . . .	235
15.5 compare_n_numvars() as reporting option . . . . .	241
15.6 Model exploration with package performance . . . . .	242
<b>16 Exercise</b>	<b>245</b>
<b>17 Interaction in linear models</b>	<b>246</b>
17.1 No age effect, no treatment effect, interaction treatment*agegroup . . . . .	246
17.2 Age effect, no treatment effect, interaction treatment*agegroup . . . . .	250
17.3 Age effect, treatment effect, interaction treatment*agegroup . . . . .	255
17.4 How to specify interaction in multivariable models . . . . .	262
<b>18 Logistic regression</b>	<b>266</b>
18.1 Odds vs. probability . . . . .	266
18.2 Data preparation . . . . .	267
18.3 Build model . . . . .	270
18.4 Create structure for ggplot . . . . .	272
18.5 create forest plot . . . . .	273
18.6 Create predictions . . . . .	274
18.7 Regression tree as alternative to glm . . . . .	278
18.8 Jackknife . . . . .	284
<b>19 Linear Mixed Models</b>	<b>286</b>
19.1 Import / Preparation . . . . .	286
19.2 Random intercept model . . . . .	288
19.2.1 Package nlme . . . . .	288
19.2.2 Package lme4 . . . . .	288
19.2.3 Visualization of fixed and random effects . . . . .	290
19.3 Random slope model . . . . .	293
19.3.1 Visualization of fixed and random effects . . . . .	294
<b>20 Machine Learning with R: Basic concepts</b>	<b>297</b>
20.1 structured vs. unstructured data . . . . .	297
20.2 supervised vs. unsupervised methods . . . . .	297
20.3 Resampling methods . . . . .	297
20.3.1 Permutation tests . . . . .	297
20.3.2 Bootstrapping . . . . .	298
20.3.3 Jackknife . . . . .	301
20.3.4 k-fold CV . . . . .	302
<b>21 Markov Chain Monte Carlo: Metropolis algorithm simulation</b>	<b>304</b>
21.1 Rule definition . . . . .	305

21.2 Data structures for simulation . . . . .	306
21.3 Simulation . . . . .	306
21.4 Results . . . . .	306
<b>22 k nearest neighbors knn</b>	<b>320</b>
22.1 Data preparation . . . . .	321
22.2 Exploratory plots . . . . .	321
22.3 Scaling . . . . .	325
22.3.1 caret function preProcess . . . . .	325
22.3.2 tidymodel approach . . . . .	326
22.3.3 preprocessCore function normalize.quantiles . . . . .	326
22.3.4 Visual comparison of the scaled data . . . . .	327
22.4 Modelling . . . . .	329
22.4.1 Definition of predictor variables (IV) . . . . .	329
22.4.2 Data splitting . . . . .	329
22.4.3 Model fitting . . . . .	330
22.4.4 Model evaluation . . . . .	333
22.4.5 Alternative approach to modelling . . . . .	334
22.4.6 Evaluation of the alternative approach . . . . .	334
22.4.7 Adding predictor variables . . . . .	336
<b>23 Regression and classification trees / Random forests</b>	<b>340</b>
23.1 Regression trees . . . . .	340
23.1.1 Graphical exploration . . . . .	340
23.1.2 Modelling . . . . .	341
23.1.3 Model evaluation . . . . .	345
23.2 RT for continuous outcomes . . . . .	347
23.3 Random forest . . . . .	350
23.3.1 Modelling . . . . .	350
23.3.2 Model evaluation . . . . .	351
<b>24 Boosted regression trees</b>	<b>354</b>
<b>25 Principal Components Analysis</b>	<b>372</b>
25.1 Exploration of correlations between predictor variables . . . . .	373
25.2 Two variable example . . . . .	376
25.3 PCA bioconductor style . . . . .	393
<b>26 Linear discriminant analysis LDA</b>	<b>397</b>
<b>27 Cluster analysis</b>	<b>405</b>
27.1 kmeans . . . . .	405
27.2 HClust . . . . .	416
27.3 Unified from easystats . . . . .	424
<b>28 caret package for machine learning</b>	<b>427</b>
28.1 Parallel computing setup . . . . .	428

28.2 Define global modelling options . . . . .	428
28.3 Model tuning and training . . . . .	430
28.3.1 K-Nearest Neighbors (KNN) . . . . .	430
28.3.2 Extreme Gradient Boosting (XGBoost) . . . . .	434
28.3.3 Random Forest (RF) . . . . .	439
28.3.4 Linear Discriminant Analysis (LDA) . . . . .	443
28.3.5 Support Vector Machine (SVM) . . . . .	447
28.3.6 Naive Bayes . . . . .	451
28.3.7 Neural Network (NN) . . . . .	456
28.4 Stopping parallel computing . . . . .	678
28.5 Model comparison . . . . .	678
28.6 Using the best model for prediction . . . . .	681

# **Preface**

This booklet is meant as companion to my R / statistics seminars. It is NOT a complete guide to either R or statistical data analysis.

# 1 Introduction

Chapters will follow my usual schedule, starting from R basics, introducing ggplot, data import, data preparation and cleaning, descriptive statistics, simple test statistics, then progression to linear models (regression/ANOVA), generalized linear models with logistic regression as example, linear mixed effect models, and machine learning.

But first things first, installation of necessary and useful tools and setup of our environment.

## 1.1 Installation of R

[The Comprehensive R Archive Network \(r-project.org\)](#)

If using Windows, install Rtools as well (3 GB)

If on linux, R2U is useful:

[CRAN as Ubuntu Binaries - r2u \(eddelbuettel.github.io\)](#)

## 1.2 Installation of RStudio Desktop (UI)

[RStudio Desktop - Posit](#)

## 1.3 Installation of Quarto (Reportgenerator)

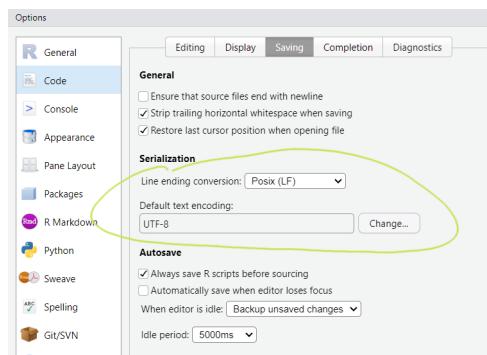
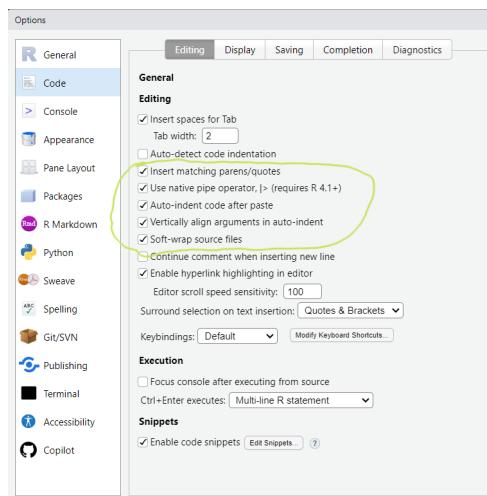
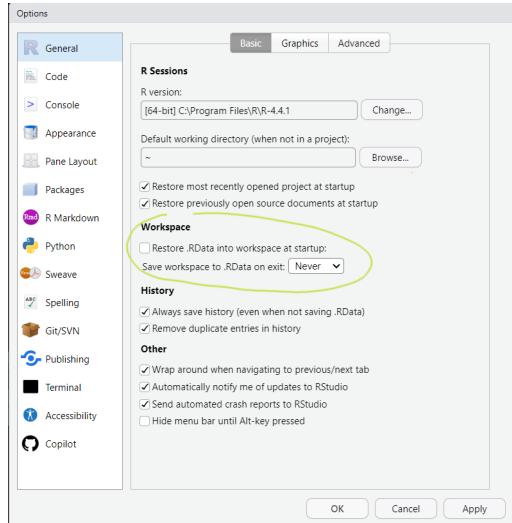
[Quarto - Get Started](#)

## 1.4 Installation of git:

[Git - Downloads \(git-scm.com\)](#)

## 1.5 Setup RStudio

There are many options available to adjust the UI to your liking, some default settings should be changed. There are global and project specific options, both can be found under Menu /Tools. Some useful (IMHO) changes are highlighted:



There are MANY more settings to adjust RStudios appearance and behavior to your liking.

## 1.6 Projects on your computer

It is terribly important (or at least helpful) to be organized ! Different analyses for e.g. new experiments or new data sets should be separated, demo analyses and exercises as well. Project structures should be consistent to make re-use of scripts easier.

As a starting point, I suggest creating a folder for everything R-related, e.g. **Rstuff**

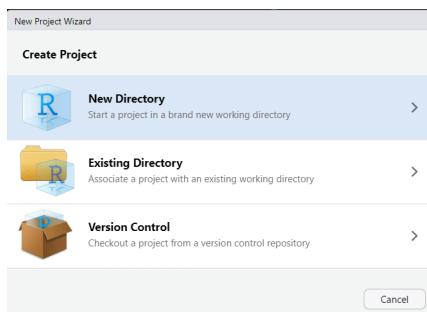
### 1.6.1 Option 1: Create manually

- Define location for project folder (often somewhere under C:/Users/SomeName/Documents),  
for seminar projects this should be the folder you just created, Rstuff or whatever name you used.
- In that folder, create a new folder, e.g. Rexercises
- Create useful sub-folders, I recommend /RScripts and /Data as minimal structure; be consistent in naming!!

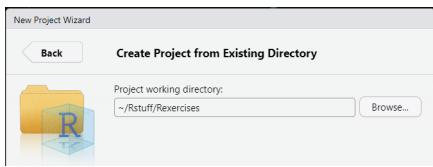
You should have a folder structure something like this:

- Rstuff
  - Rexercises
    - \* RScripts
    - \* Data
  - SomeProject
    - \* RScripts
    - \* Data

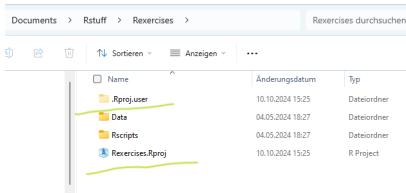
Now you are ready to create a project in RStudio:



As there is an existing directory, this is what you select.



In this new window you browse to the root folder of your project and create your project.  
This will create new (possibly hidden) entries:



### 1.6.2 Option 2: Create from a github repository

e.g. <https://github.com/abusjahn/RStatsbook>

Here you have to define source and new location.

## 2 Some useful resources

### 2.1 Books

There are plenty of books available, check out e.g.

[R for Data Science](#)

[Modern Statistics for Modern Biology](#)

[Modern R with the tidyverse](#)

[ggplot2: Elegant Graphics for Data Analysis](#)

[The big book of R](#)

[Fundamentals of Data Visualization \(clauswilke.com\)](#)

[Statistical Inference via Data Science \(moderndive.com\)](#)

[Data Science Live Book \(datascienceheroes.com\)](#)

[Biostatistics for Biomedical Research \(hbiostat.org\)](#)

[Welcome | Data Science at the Command Line, 2e \(jeroenjanssens.com\)](#)

### 2.2 Ressources

[Cheatsheets - Posit](#)

[Contributed Cheatsheets \(rstudio.github.io\)](#)

[R-bloggers](#)

[Bioconductor - Home](#)

# **3 *Syntax rules / basic things to know about R***

This is going to be the boring technical stuff... We'll get to the more interesting topics in the next chapters.

## **3.1 *Script preparation / basic setup***

At the beginning of (almost) every script we define packages to be used. This could be done by either

- checking if packages needed are installed and otherwise do so, followed by function `library(packagename)`

OR

- simplifying this using function `p_load()` from package pacman; if you want to create fool-proof scripts, check for pacman and install if needed.

```
# ↑ this is the head of a code chunk
Sys.setenv(LANG = "en_EN.UTF-8") # to get errors/warnings in English
if (!requireNamespace("pacman", quietly = TRUE)) {
  install.packages("pacman")
}
# library("pacman") adds all functions from package to namespace
# use("pacman", "p_load") # starting with R 4.5.0, selected functions can be extracted
pacman::p_load(
  conflicted, # tests/solutions for name conflicts
  tidyverse, # metapackage
  wrappedtools, # my own tools package
  randomNames # used to create pseudo names
)
conflict_scout()
```

3 conflicts

```
* `filter()`/: dplyr and stats

* `lag()`/: dplyr and stats
```

```
* `mean_cl_boot()` : wrappedtools and ggplot2
```

```
conflicts_prefer(  
  dplyr::filter,  
  stats::lag  
)
```

```
[conflicted] Will prefer dplyr::filter over any other package.
```

```
[conflicted] Will prefer stats::lag over any other package.
```

## 3.2 Numeric operations

```
### simple calculations ####  
2 + 5
```

```
[1] 7
```

```
3 * 5
```

```
[1] 15
```

```
15 / 3 # not 15:3!!, would create vector 15,14,13 ... 3
```

```
[1] 5
```

```
3^2
```

```
[1] 9
```

```
9^0.5
```

```
[1] 3
```

```
10 %% 3 # modulo
```

```
[1] 1
```

## 3.3 Variables

### 3.3.1 Variable names

Naming things is harder than you may expect. Try to be verbose and consistent in language and style. Commonly used are snake\_case\_style, CamelCaseStyle, and kebab-case-style.

Decide about computer-friendly (syntactical) or human-friendly names, illegal names can be used inside backticks: ‘measure [unit]’. My preference is syntactical for script variables and humane for data variables, e.g. column names, print labels etc.

There are rules for valid syntactical names:

- UPPERCASE and lowercase are distinguished
- start with letter or symbols as .\_\_ , but not with a number
- no mathematical symbols or brackets allowed

To store some value into a variable, use the assignment operator `<-` ; while it possible to use `=` or `->` , this is rather unusual. Assignments are silent, so either a call of the variable, or `print()` / `cat()` function are needed to inspect. Alternatively, put brackets around assignment: (varname `<-` content).

```
### Variable names #####
test <- 1
test1 <- 1
# 1test <- 2 # wrong, would result in error
`1test` <- 2 # this would be possible
test_1 <- 5
test.1 <- 2
`test-1` <- 6
`test(1)` <- 5
Test <- "bla"
HereAreFilteredData <- "" # CamelCase
here_are_filtered_data <- "test" # snake_case
`Weight [kg]` <- 67
```

### 3.3.2 Basic classes of data

R is ‘guessing’ the suitable type of data from input. This should be checked after e.g. importing data! If elements of different classes are found, the more inclusive is used. There are functions to change / force a type if needed.

The `class()` function returns the class of an object, which determines how it behaves with respect to functions like `print()`. The class of an object can be changed by using generic functions and methods.

The `typeof()` function returns the basic data type of an object, which determines how it is stored in memory. The basic data type of an object cannot be changed.

The `str()` function shows class and examples of an object.

### 3.3.2.1 *Guessed classes*

```
float_num <- 123.456  
class(float_num)
```

```
[1] "numeric"
```

```
typeof(float_num)
```

```
[1] "double"
```

```
str(float_num)
```

```
num 123
```

```
int_num <- 123L # L specifies integer, guessing requires more values  
class(int_num)
```

```
[1] "integer"
```

```
typeof(int_num)
```

```
[1] "integer"
```

```
str(int_num)
```

```
int 123
```

```
integer(length = 3)
```

```
[1] 0 0 0
```

```
result <- 9^(1 / 2)
result
```

```
[1] 3
```

```
print(result)
```

```
[1] 3
```

```
cat(result)
```

```
3
```

```
char_var <- "some words"
class(char_var)
```

```
[1] "character"
```

```
typeof(char_var)
```

```
[1] "character"
```

```
character(length = 5)
```

```
[1] "" "" "" "" "
```

```
logical_var <- TRUE # can be abbreviated to T
logical_var2 <- FALSE # or F, seen as bad style
class(logical_var)
```

```
[1] "logical"
```

```
typeof(logical_var)
```

```
[1] "logical"
```

```
logical(length = 3)
```

```
[1] FALSE FALSE FALSE
```

```
# logicals usually are defined by conditions:  
int_num < float_num
```

```
[1] TRUE
```

```
# all numbers are true but 0  
as.logical(c(0, 1, 5, -7.45678)) # c() combines values into a vector
```

```
[1] FALSE TRUE TRUE TRUE
```

### 3.3.3 Exercise: *variables / Assignments*

(Please do exercises in scripts in a different project!)

**Scenario:** Imagine you've just received a new sample in the lab, and you need to record some basic information about it in R. This will help you get familiar with how R stores different kinds of data.

#### Learning Objectives:

- Learn how to create variables (objects) in R using the assignment operator (<-).
- Understand different basic data types (classes) in R: `numeric`, `character`, `logical`.
- Practice viewing the content and class of your variables.

#### Instructions:

##### 1. Assign a Sample ID:

- Create a variable called `sample_id`.
- Assign it a unique identifier for your sample. Since it's text, make sure to put it in quotes, e.g., "ExpA\_S001".

##### 2. Record the Organism:

- Create a variable called `organism_name`.
- Assign the scientific name of the organism your sample came from. Again, use quotes, e.g., "Saccharomyces cerevisiae".

### 3. Record a Measurement:

- You've measured the length of a cell from this sample in micrometers.
- Create a variable called `cell_length_um`.
- Assign a numerical value (without quotes) for the length, e.g., 7.5.

### 4. Record a Binary Condition:

- Was this sample grown in the presence of a specific nutrient (e.g., glucose)?
- Create a variable called `grown_on_glucose`.
- Assign a logical value: TRUE if yes, FALSE if no (these are special keywords in R, no quotes needed). E.g., TRUE.

### 5. Check Your Work (Content and Class):

- After creating each variable, type the variable name on a new line and press Enter to see its content.
- Use the `class()` function to check what type of data R thinks each variable holds. For example:

```
- class(sample_id)  
- class(organism_name)  
- class(cell_length_um)  
- class(grown_on_glucose)
```

Factor: categorical variables with limited sets of distinct values (called levels), internally stored as integers. Everything intended to group subjects or representing categories (like species, tissue type, treatment) should be stored as factor for efficient storage and proper statistical handling in R. In Python, Pandas Categorical and dictionaries are related constructs.

Package `forcats` provides nice tools for factors!

```
factor_var <- factor(c("m", "m", "f", "m", "f", "f", "?"))  
factor_var
```

```
[1] m m f m f f ?  
Levels: ? f m
```

```
class(factor_var)
```

```
[1] "factor"
```

```
typeof(factor_var) # that is why factors can be called enumerated type
```

```
[1] "integer"
```

```
levels(factor_var)
```

```
[1] "?" "f" "m"
```

```
# factor definition can reorder, rename, and drop levels:  
factor_var2 <- factor(c("m", "m", "f", "m", "f", "f", "?"),  
  levels = c("m", "f"),  
  labels = c("male", "female"))  
factor_var2
```

```
[1] male   male   female male   female female <NA>  
Levels: male female
```

### 3.3.4 *Exercise: Factors*

**Scenario:** Beyond just individual measurements, biologists often need to categorize samples based on different experimental conditions, genetic backgrounds, or developmental stages. R uses a special data type called **factors** for this. Factors are crucial because they tell R that your data belongs to discrete groups, which is very important for statistical analysis!

**The “Integer with a Name Tag” Concept:** Think of it this way: R stores factors internally as numbers (integers), but it “tags” these numbers with meaningful labels (the “name tags”). For example, R might store “Male” as 1 and “Female” as 2, but it always displays “Male” and “Female” to you. This saves memory and makes computations efficient, while keeping your data interpretable.

#### Learning Objectives:

- Understand factors as a way to store **categorical data**.
- Learn how to create factors from character data.
- See how to inspect the **levels** (the “name tags”) of a factor.
- Discover how R internally represents factors as **integers**.
- Learn how to create **ordered factors** when the categories have a natural sequence.

#### Instructions:

### 1. Categorizing by Treatment Group (Unordered Factor):

- Imagine your samples are from different treatment groups: “Control”, “Drug A”, “Drug B”.
- First, create a *character variable* for the treatment of one sample  
*What is its class?*
- Now, convert `sample_treatment` into a **factor**. This tells R it’s a category.  
*What is its class now? Notice how R also lists “Levels” when you print it.*
- To see all the possible “name tags” (categories) R knows for this factor, use `levels()`  
*Why do you think R automatically inferred “Control”, “Drug A”, “Drug B” as levels, even though you only assigned “Drug A”? (Hint: It hasn’t; it only knows “Drug A” for now. We’ll fix this in the next step!)*
- **The “Integer Tag” Reveal:** To see the hidden integer that R uses for “Drug A”, convert the factor to a numeric type:  
*What number did you get? This number corresponds to the alphabetical position of “Drug A” among the levels R currently sees (Drug A).*

### 2. Defining All Levels Explicitly:

- When you create a factor, it’s good practice to tell R *all* the possible levels upfront, even if a particular sample only has one of them. This ensures consistency.
- Re-create `sample_treatment_factor`, but this time specifying all the `levels`:  
*Now, what number do you get from `as.numeric()`? How does it relate to the levels you explicitly defined? This demonstrates how the integer tag links to its “name tag” position in the defined levels.*

### 3. Categorizing by Developmental Stage (Ordered Factor):

- Sometimes, categories have a natural order. For example, developmental stages: “Larva”, “Pupa”, “Adult”. “Adult” is definitely “later” than “Larva”.
- Let’s create a factor for a sample’s developmental stage, making sure R understands the order:

*How does the `class()` output differ from `sample_treatment_factor_full`? What integer did “Pupa” get, and why?*

#### Key Takeaways:

- Use **character** for free text (like comments or unique IDs).
- Use **factor** for **categorical data** (like experimental groups, sexes, genotypes, species names in a fixed list).

- Factors are essential for statistical models as they correctly identify groups for comparisons.
- `levels()` shows you the “name tags.”
- `as.numeric()` shows you the underlying “integer tags.”
- Use `ordered = TRUE` within the `factor()` function when your categories have a meaningful order (e.g., “Low” < “Medium” < “High”).

*Dates / Time:*

```
(date_var <- Sys.Date())
```

```
[1] "2026-01-13"
```

```
class(date_var)
```

```
[1] "Date"
```

```
typeof(date_var)
```

```
[1] "double"
```

```
class(Sys.time())
```

```
[1] "POSIXct" "POSIXt"
```

```
typeof(Sys.time())
```

```
[1] "double"
```

*Mixed classes:*

If the data that goes into a variable has more than 1 class, the more general class is selected.

```
test2 <- c(1, 2, "a", "b")
class(test2)
```

```
[1] "character"
```

```
test2
```

```
[1] "1" "2" "a" "b"
```

### 3.3.4.1 *Forcing / casting classes*

If the assigned class is not correct or appropriate, it can be changed. Casting functions usually start with `as`. When creating variables filled with NA, use casting functions or specific variants of NA to force type!

```
(test <- c(1, 2, 3, "a", "b", "c"))
```

```
[1] "1" "2" "3" "a" "b" "c"
```

```
(test_n <- as.numeric(test))
```

```
Warning: NAs introduced by coercion
```

```
[1] 1 2 3 NA NA NA
```

```
as.numeric(factor_var)
```

```
[1] 3 3 2 3 2 2 1
```

```
as.character(10:19)
```

```
[1] "10" "11" "12" "13" "14" "15" "16" "17" "18" "19"
```

```
# NAs  
(test_NA1 <- rep(NA, 10))
```

```
[1] NA NA NA NA NA NA NA NA NA
```

```
class((test_NA1))
```

```
[1] "logical"
```

```

class(NA_real_)

[1] "numeric"

(test_NA2 <- rep(NA_real_, 10))

[1] NA NA

class((test_NA2))

[1] "numeric"

class(NA_integer_)

[1] "integer"

class(NA_character_)

[1] "character"

class(NA_Date_) # from package lubridate

[1] "Date"

```

### 3.3.5 *Indexing variables*

The most general kind of indexing is by position, starting with **1**. Negative numbers result in exclusion of position(s). Position indices are provided within square brackets. The index can (and usually will) be a variable instead of hard coded numbers.

```

(numbers1 <- c(5, 3, 6, 8, 2, 1))

[1] 5 3 6 8 2 1

numbers1[1]

[1] 5

```

```
numbers1[1:3]
```

```
[1] 5 3 6
```

```
numbers1[-c(1, 3)]
```

```
[1] 3 8 2 1
```

```
numbers2 <- 1:3  
numbers1[numbers2]
```

```
[1] 5 3 6
```

```
# numbers1[1,2] #Error: incorrect number of dimensions
```

To get first or last entries, head() and tail() can be used. By default 6 entries are returned.

```
tail(x = numbers1, n = 1)
```

```
[1] 1
```

```
head(x = numbers1, n = 3)
```

```
[1] 5 3 6
```

```
nth(x = numbers1, n = -2) # 2nd to last
```

```
[1] 2
```

### 3.3.6 Usage of variables

Variables are like placeholders for their content, so that you don't have to remember where you left things. Operations on variables are operations on their content. Changing the content of a variable does not automatically save those changes back to the variable, this needs to be done explicitly!

```
numbers1 + 100 # not stored anywhere, just printed
```

```
[1] 105 103 106 108 102 101
```

```
numbers1 + numbers2 # why does this even work?
```

```
[1] 6 5 9 9 4 4
```

When combining variables of different length, the short one is recycled, so the numbers2 is added to the first 3 elements of numbers2, then is reused and added to the remaining 3 elements. If the length of the longer is not a multiple of the shorter, there will be a warning.

```
c(2, 4, 6, 8) + 1
```

```
[1] 3 5 7 9
```

```
c(2, 4, 6, 8) + c(1, 2)
```

```
[1] 3 6 7 10
```

```
c(2, 4, 6, 8) + c(1, 2, 3)
```

Warning in c(2, 4, 6, 8) + c(1, 2, 3): longer object length is not a multiple of shorter object length

```
[1] 3 6 9 9
```

## 3.4 Functions

### 3.4.1 Function usage

Functions have the same naming rules as variables, but the name is always followed by opening/closing round brackets, within those brackets function parameters/arguments can be specified to provide input or control behavior:

*FunctionName(parameter1=x1,parameter2=x2,x3,...)*

Most functions have named arguments, those argument names may be omitted as long as parameter values are supplied in the defined order. Arguments may have predefined default values, see `help!` Some functions like `c()` use unnamed arguments.

```
c("my", "name") # unnamed
```

```
[1] "my"    "name"
```

```
# ?mean  
mean(x = c(3, 5, 7, NA)) # using default parameters
```

```
[1] NA
```

```
mean(x = c(3, 5, 7, NA), na.rm = TRUE) # overriding default parameter
```

```
[1] 5
```

```
mean(na.rm = TRUE, x = c(3, 5, 7, NA)) # changed order of arguments
```

```
[1] 5
```

```
mean(c(3, 5, 7, NA), na.rm = TRUE) # name of 1st argument omitted
```

```
[1] 5
```

```
sd(c(3, 5, 7, NA), na.rm = TRUE)
```

```
[1] 2
```

```
# same logic as mean, partially the same arguments  
median(1:100, TRUE)
```

```
[1] 50.5
```

```
# omitting arguments influences readability of a function, careful!  
t <- c(1:10, 100)  
quantile(x = t, probs = c(.2, .8))
```

```
20% 80%  
3     9
```

```
# putting text elements together  
paste("some text", 1:3)
```

```
[1] "some text 1" "some text 2" "some text 3"
```

```
paste0("some text", 1:3)
```

```
[1] "some text1" "some text2" "some text3"
```

```
paste("some text", 1:3, sep = ": ")
```

```
[1] "some text: 1" "some text: 2" "some text: 3"
```

```
paste("some text", 1:3, sep = ": ", collapse = "; ")
```

```
[1] "some text: 1; some text: 2; some text: 3"
```

```
paste("some text", 1:3, sep = ": ", collapse = "\n") |> cat("\n")
```

```
some text: 1  
some text: 2  
some text: 3
```

```
paste("mean", "SD", sep = " \u00b1 ")
```

```
[1] "mean \u00b1 SD"
```

### 3.4.2 Functions combined

Functions often just solve one problem or task, so usually we need to combine them to, for instance, filter our data, then calculate some statistics, and finally tabulize the results. This can be done by nesting or piping. Piping (creation of pipelines or production belts) makes reading/understanding scripts easier, as it shows order of information flowing from one function to the next, often visualized with a special symbol |>

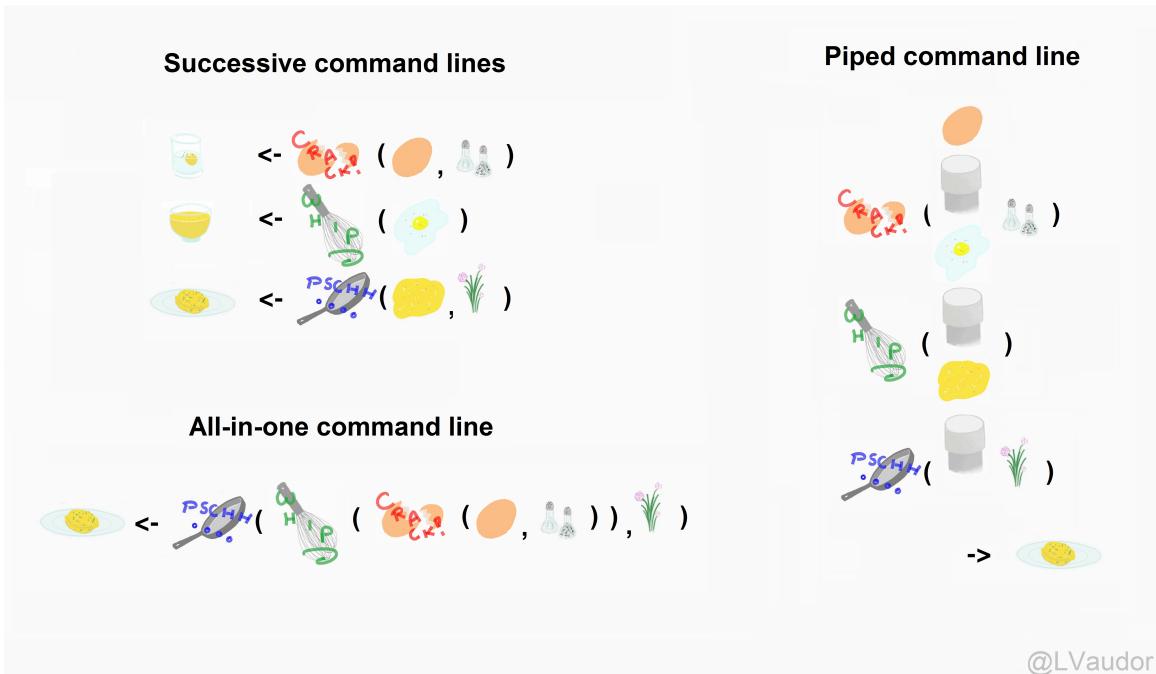


Figure 3.1: *Piping functions*

```
# functions may be nested:
floor(
  as.numeric(
    Sys.Date() -
    as.Date("1985/12/10")
  ) /
  365.25
)
```

[1] 40

```
# or (usually better) piped:
mtcars |> # inbuild example data, use F1!
  mutate(am = factor(am,
    levels = c(0, 1),
    labels = c(
      "automatic",
      "manual"
    )
  )) |> # #change into better class
  filter(vs == 1) |> # filter out V-shaped
  group_by(am) |> # ask for grouped analysis
```

```

summarize(across(
  .cols = c(wt, mpg, qsec, disp),
  .fns = meansd
)) |>
pivot_longer(cols = -am, names_to = "Measure") |> # put variables in rows
pivot_wider(
  id_cols = Measure, names_from = am, # put groups in cols
  values_from = value
)

```

```

# A tibble: 4 x 3
  Measure automatic manual
  <chr>    <chr>    <chr>
1 wt        3.2 ± 0.3 2.0 ± 0.4
2 mpg       21 ± 2    28 ± 5
3 qsec      20 ± 1    19 ± 1
4 disp      175 ± 49  90 ± 19

```

If a sequence of functions is used often, combining them into a new function is advisable, e.g. this function is a combination of descriptive and test statistics:

```

# can be combined into higher order functions:
compare2numvars(
  data = mtcars,
  dep_vars = c("mpg", "wt", "qsec"),
  indep_var = "am",
  add_n = TRUE,
  gaussian = TRUE
)

```

```

# A tibble: 3 x 5
  Variable desc_all          `am 0`          `am 1`          p
  <chr>    <chr>          <chr>          <chr>          <chr>
1 mpg      20 ± 6 [n = 32]  17 ± 4 [n = 19]  24 ± 6 [n = 13]  0.001
2 wt       3.2 ± 1.0 [n = 32] 3.8 ± 0.8 [n = 19]  2.4 ± 0.6 [n = 13]  0.001
3 qsec     18 ± 2 [n = 32]   18 ± 2 [n = 19]   17 ± 2 [n = 13]   0.206

```

### 3.4.3 Writing functions

Functions can be thought of as blocks/chunks of code with defined in- and output. Functions intended for general use (e.g. published in a package) should be enhanced by error prevention / handling and documentation.

```
# FunctionName<-function(parameters...){definition}
division <- function(y, x) {
  return(x / y)
}
(Sys.Date() - as.Date("1958/12/10")) |>
  as.numeric() |>
  division(y = 365.25, x = _) |>
  floor()
```

[1] 67

```
mean <- function(values) {
  return(base::mean(values, na.rm = TRUE))
}

mean(c(1, 2, 3, NA))
```

[1] 2

```
rm(mean) # to revert to original function base::mean

mark_sign <- function(SignIn) {
  SignIn <- as.numeric(SignIn)
  if (is.na(SignIn)) {
    SignOut <- "wrong input, stupido!"
  } else {
    # if (!is.na(SignIn)) {
    SignOut <- "n.s."
    if (SignIn <= 0.1) {
      SignOut <- "+"
    }
    if (SignIn <= 0.05) {
      SignOut <- "*"
    }
    if (SignIn <= 0.01) {
      SignOut <- "**"
    }
    if (SignIn <= 0.001) {
      SignOut <- "***"
    }
  }
  return(SignOut)
}
```

```
mark_sign(SignIn = 0.035)
```

```
[1] "*"
```

```
mark_sign(SignIn = "0.35")
```

```
[1] "n.s."
```

```
mark_sign(SignIn = "p=3,5%") # wrong input
```

Warning in mark\_sign(SignIn = "p=3,5%"): NAs introduced by coercion

```
[1] "wrong input, stupido!"
```

*different implementation*

```
markSign0 <- function(SignIn, plabel = c("n.s.", "+", "*", "**", "***")) {  
  SignIn <- suppressWarnings(  
    as.numeric(SignIn)  
  )  
  SignOut <- cut(SignIn,  
    breaks = c(-Inf, .001, .01, .05, .1, 1),  
    labels = rev(plabel)  
  )  
  return(SignOut)  
}  
  
markSign0(SignIn = c(0.035, 0.00002, .234))
```

```
[1] *      ***  n.s.  
Levels: *** ** * + n.s.
```

```
markSign0(SignIn = "0.35")
```

```
[1] n.s.  
Levels: *** ** * + n.s.
```

```
markSign0(SignIn = "p=3,5%") # wrong input
```

```
[1] <NA>  
Levels: *** ** * + n.s.
```

```
# source("F:/Aktenschrank/Analysen/R/myfunctions.R")
```

## 3.5 More complex data types, created by functions

### 3.5.1 Matrix

A matrix is a 2-dimensional data structure, where all elements are of the same class.

#### 3.5.1.1 Creation

```
my1.Matrix <-  
  matrix(  
    data = 1:12,  
    # nrow=4, # this is not needed, can be derived from data  
    ncol = 3,  
    byrow = TRUE, # data are put into row 1 first  
    dimnames = list(  
      paste0("row", 1:4),  
      paste0("col", 1:3)  
    )  
  )  
print(my1.Matrix)
```

	col1	col2	col3
row1	1	2	3
row2	4	5	6
row3	7	8	9
row4	10	11	12

```
data <- seq(from = 1, to = 100, by = 1) # 1:100  
nrow <- 20  
matrix(  
  data = data,  
  nrow = nrow,  
  byrow = FALSE, # data are put into column 1 first
```

```

    dimnames = list(
      paste0("row", 1:nrow),
      paste0("col", 1:(length(data) / nrow))
    )
  ) |>
  head()

```

	col1	col2	col3	col4	col5
row1	1	21	41	61	81
row2	2	22	42	62	82
row3	3	23	43	63	83
row4	4	24	44	64	84
row5	5	25	45	65	85
row6	6	26	46	66	86

```

my2.Matrix <- matrix(c(1, 2, 3, 11, 12, 13),
  nrow = 2, ncol = 3
) # byrow=FALSE, specified but default
my2.Matrix

```

	[,1]	[,2]	[,3]
[1,]	1	3	12
[2,]	2	11	13

### 3.5.1.2 Indexing

Addressing a matrix is done with [row\_index, column\_index]

```
my1.Matrix[2, 3] # Index:[row,column]
```

```
[1] 6
```

```
my1.Matrix[2, ] # all columns
```

col1	col2	col3
4	5	6

```
my1.Matrix[, 2] # all rows
```

row1	row2	row3	row4
2	5	8	11

```
my1.Matrix[c(1, 3), -2] # exclude column 2
```

```
  col1 col3  
row1    1    3  
row3    7    9
```

```
my1.Matrix[1, 1] <- NA # Index can be used for writing as well
```

## 3.6 Exercise: Matrix

**Scenario:** Imagine you are a field biologist studying plant distribution. You've set up several quadrats (square frames) in your study area and counted the number of individuals for three different plant species in each quadrat.

### Instructions:

#### 1. Create Your Data:

- You have the following counts:
  - **Species A:** Quadrat 1: 12, Quadrat 2: 8, Quadrat 3: 15
  - **Species B:** Quadrat 1: 5, Quadrat 2: 10, Quadrat 3: 7
  - **Species C:** Quadrat 1: 20, Quadrat 2: 14, Quadrat 3: 18
- Create a numeric vector for each species' counts (e.g., `species_A_counts <- c(12, 8, 15)`).

#### 2. Create a Matrix:

- Combine these three vectors into a single matrix. Name this matrix `quadrat_data`.
- Make sure each row represents a species and each column represents a quadrat.
- Set the number of rows (`nrow`) to 3 and the number of columns (`ncol`) to 3.

#### 3. Name Rows and Columns:

- Assign row names to your matrix: "SpeciesA", "SpeciesB", "SpeciesC".
- Assign column names to your matrix: "Quadrat1", "Quadrat2", "Quadrat3".

#### 4. Index Your Matrix (Access Data):

- **Access a single element:** Get the count of Species B in Quadrat 2.
- **Access a full row:** Get all counts for Species A.

- **Access a full column:** Get all counts from Quadrat 3.
- **Access multiple elements/rows/columns:**
  - Get the counts for Species A and Species C in Quadrat 1 and Quadrat 2.

### 3.6.1 Data frame

A data frame has 2 dimensions, it can handle various data types (1 per columns). This structure is rather superseded by tibbles (see below).

#### 3.6.1.1 Creation

Data frames are defined by creating and filling columns, functions can be used (and piped) to create content.

```
patientN <- 15
(myTable <- data.frame(
  patientCode = paste0("pat", 1:patientN),
  Var1 = 1, # gets recycled
  Var2 = NA_Date_
)) |> head()
```

	patientCode	Var1	Var2
1	pat1	1	<NA>
2	pat2	1	<NA>
3	pat3	1	<NA>
4	pat4	1	<NA>
5	pat5	1	<NA>
6	pat6	1	<NA>

```
str(myTable)
```

```
'data.frame': 15 obs. of 3 variables:
$ patientCode: chr  "pat1" "pat2" "pat3" "pat4" ...
$ Var1        : num  1 1 1 1 1 1 1 1 1 1 ...
$ Var2        : Date, format: NA NA ...
```

```
set.seed(101)
myTable <- data.frame(
  patientCode = paste0("pat", 1:patientN),
  Age = runif(n = patientN, min = 18, max = 65) |> floor(),
```

```

Sex = factor(rep(x = NA, times = patientN),
  levels = c("m", "f")
),
`sysRR (mmHg)` = round(rnorm(n = patientN, mean = 140, sd = 10)),
check.names = FALSE
)
head(myTable)

```

	patientCode	Age	Sex	sysRR (mmHg)
1	pat1	35	<NA>	142
2	pat2	20	<NA>	132
3	pat3	51	<NA>	122
4	pat4	48	<NA>	157
5	pat5	29	<NA>	144
6	pat6	32	<NA>	148

### 3.6.1.2 Indexing

Beside the numeric index, columns can be addressed by name. This can be done by either `dfname$colname` (for the content of a single column) or `dfname[, "colname"]` for 1 or more columns.

```
myTable[1:5, 1]
```

```
[1] "pat1" "pat2" "pat3" "pat4" "pat5"
```

```
myTable[1:5, ]
```

	patientCode	Age	Sex	sysRR (mmHg)
1	pat1	35	<NA>	142
2	pat2	20	<NA>	132
3	pat3	51	<NA>	122
4	pat4	48	<NA>	157
5	pat5	29	<NA>	144

```
myTable[, 1:2]
```

	patientCode	Age
1	pat1	35
2	pat2	20

```
3      pat3  51
4      pat4  48
5      pat5  29
6      pat6  32
7      pat7  45
8      pat8  33
9      pat9  47
10     pat10 43
11     pat11 59
12     pat12 51
13     pat13 52
14     pat14 61
15     pat15 39
```

```
myTable$patientCode[1:5]
```

```
[1] "pat1" "pat2" "pat3" "pat4" "pat5"
```

```
myTable[1:5, "patientCode"]
```

```
[1] "pat1" "pat2" "pat3" "pat4" "pat5"
```

```
# returns vector of values for a single column, data.frame otherwise
myTable["patientCode"] # returns df
```

```
patientCode
1      pat1
2      pat2
3      pat3
4      pat4
5      pat5
6      pat6
7      pat7
8      pat8
9      pat9
10     pat10
11     pat11
12     pat12
13     pat13
14     pat14
15     pat15
```

```
columns <- c("Sex", "Age")
myTable[1:5, columns]
```

```
Sex Age
1 <NA> 35
2 <NA> 20
3 <NA> 51
4 <NA> 48
5 <NA> 29
```

```
myTable[1:5, c("patientCode", "Age")]
```

```
patientCode Age
1 pat1 35
2 pat2 20
3 pat3 51
4 pat4 48
5 pat5 29
```

```
myTable[, 1] <- paste0("Code", 1:patientN)
```

### 3.6.2 *Tibble*

Tibbles are a modern and efficient data structure that extends data frames, providing enhanced features and performance for data manipulation and analysis.

#### 3.6.2.1 *Creation*

```
patientN <- 25
set.seed(3105)
rawdata <- tibble(
  PatID = paste("P", 1:patientN), # as in data.frame
  Sex = sample(
    x = c("male", "female"), # random generator
    size = patientN, replace = TRUE,
    prob = c(.7, .3)
  ),
  Ethnicity = sample(
    x = 1:6,
    size = patientN,
```

```

    replace = TRUE,
    prob = c(.01, .01, .05, .03, .75, .15)
),
# random assignments
`Given name` = randomNames(
  n = patientN,
  gender = Sex,
  # this is a reference to column Sex
  ethnicity = Ethnicity,
  which.names = "first"
),
`Family name` = randomNames(
  n = patientN,
  ethnicity = Ethnicity,
  which.names = "last"
),
Treatment = sample(
  x = c("Placebo", "Verum"),
  size = patientN,
  replace = TRUE
),
`sysRR (mmHg)` = round(rnorm(n = patientN, mean = 140, sd = 10)) -
  (Treatment == "Verum") * 15,
`diaRR (mmHg)` = round(rnorm(n = patientN, mean = 80, sd = 10)) -
  (Treatment == "Verum") * 10,
HR = round(rnorm(n = patientN, mean = 90, sd = 7))
)
rawdata

```

```

# A tibble: 25 x 9
  PatID Sex   Ethnicity `Given name` `Family name` Treatment `sysRR (mmHg)`
  <chr> <chr>     <int> <chr>      <chr>       <chr>          <dbl>
1 P 1   male        5 Austin      Collins     Verum        135
2 P 2   male        5 Peter       Atkins     Verum        129
3 P 3   male        5 Jeffrey    Williamson Verum        128
4 P 4   male        1 Hament     Maez       Verum        133
5 P 5   female      5 Jennifer   Allen      Placebo      159
6 P 6   male        5 Nathan     Potter     Placebo      153
7 P 7   male        5 Joshua     Trujillo   Verum        126
8 P 8   male        5 Thomas     Martin     Placebo      158
9 P 9   male        5 Sander     Stickels   Placebo      121
10 P 10  male       5 Brandon    Morgan     Verum        106
# i 15 more rows
# i 2 more variables: `diaRR (mmHg)` <dbl>, HR <dbl>

```

```
colnames(rawdata)
```

```
[1] "PatID"          "Sex"           "Ethnicity"       "Given name"      "Family name"  
[6] "Treatment"     "sysRR (mmHg)"  "diaRR (mmHg)"  "HR"
```

```
cn() # shortcut from wrappedtools
```

```
[1] "PatID"          "Sex"           "Ethnicity"       "Given name"      "Family name"  
[6] "Treatment"     "sysRR (mmHg)"  "diaRR (mmHg)"  "HR"
```

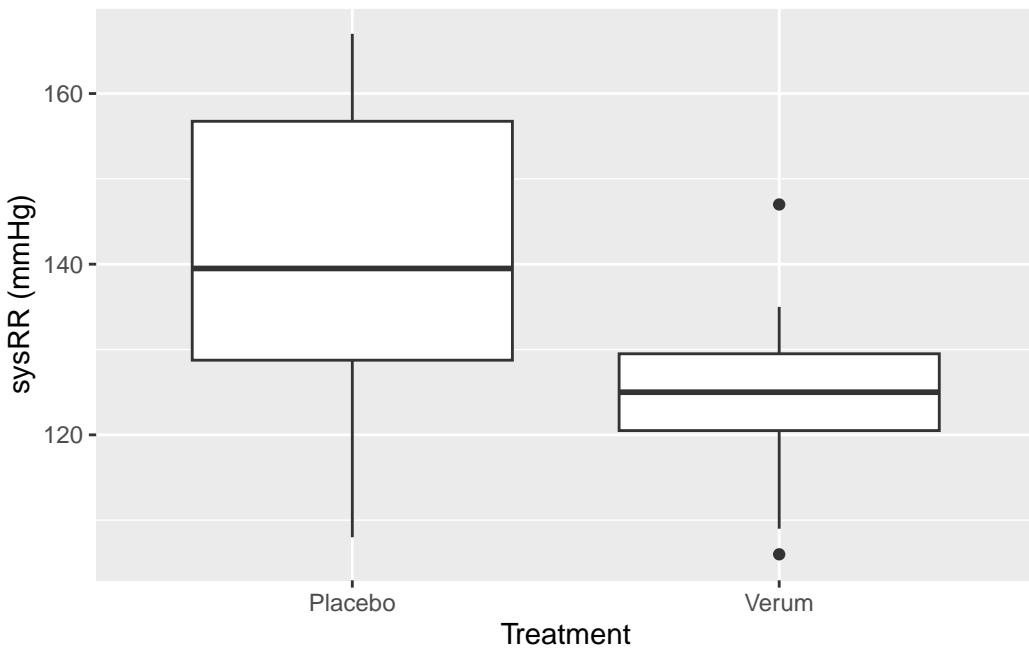
```
# example of data management for a tibble, recoding ethnicity:
```

```
rawdata <- rawdata |>
```

```
  mutate(  
    Ethnicity = factor(  
      Ethnicity,  
      levels = 1:6,  
      labels = c(  
        "American Indian or Native Alaskan",  
        "Asian or Pacific Islander",  
        "Black (not Hispanic)",  
        "Hispanic",  
        "White (not Hispanic)",  
        "Middle-Eastern, Arabic"  
      ))  
    )  
  )
```

```
# quick visual inspection
```

```
ggplot(rawdata, aes(x = Treatment, y = `sysRR (mmHg)`)) +  
  geom_boxplot()
```



### 3.6.2.2 Indexing

The same rules as for the data frame, but more consistent behavior.

```
rawdata[1:5, 1:2]
```

```
# A tibble: 5 x 2
  PatID Sex
  <chr> <chr>
1 P 1   male
2 P 2   male
3 P 3   male
4 P 4   male
5 P 5   female
```

```
rawdata[, 6]
```

```
# A tibble: 25 x 1
  Treatment
  <chr>
1 Verum
2 Verum
3 Verum
```

```
4 Verum
5 Placebo
6 Placebo
7 Verum
8 Placebo
9 Placebo
10 Verum
# i 15 more rows
```

```
rawdata[6]
```

```
# A tibble: 25 x 1
  Treatment
  <chr>
 1 Verum
 2 Verum
 3 Verum
 4 Verum
 5 Placebo
 6 Placebo
 7 Verum
 8 Placebo
 9 Placebo
10 Verum
# i 15 more rows
```

```
rawdata[[6]]
```

```
[1] "Verum"    "Verum"    "Verum"    "Verum"    "Placebo"   "Placebo"   "Verum"
[8] "Placebo"   "Placebo"   "Verum"    "Placebo"   "Placebo"   "Verum"    "Placebo"
[15] "Verum"    "Verum"    "Verum"    "Verum"    "Placebo"   "Placebo"   "Verum"
[22] "Verum"    "Verum"    "Placebo"  "Verum"
```

```
rawdata$`Family name`
```

```
[1] "Collins"   "Atkins"    "Williamson" "Maez"      "Allen"
[6] "Potter"     "Trujillo"   "Martin"     "Stickels"   "Morgan"
[11] "al-Kanan"   "Foster"    "O'Donnell"  "Doubleday"  "Wurz"
[16] "Gentry"     "Good"      "Miller"     "Italiano"   "Kircher"
[21] "Barr"       "Copley"    "Cook"       "Thomas"     "Max"
```

Differences in addressing data frames and tibbles:

- tibble and [ always returns tibble
- tibble and [[ always returns vector
- data.frame and [ may return data.frame (if >1 column) or vector
- data.frame and [[ always returns vector

```
rawdata_df <- as.data.frame(rawdata)
rawdata[2] # returns Tibble with 1 column
```

```
# A tibble: 25 x 1
  Sex
  <chr>
1 male
2 male
3 male
4 male
5 female
6 male
7 male
8 male
9 male
10 male
# i 15 more rows
```

```
rawdata[[2]] # returns vector
```

```
[1] "male"    "male"    "male"    "male"    "female"  "male"    "male"    "male"
[9] "male"    "male"    "male"    "male"    "male"    "male"    "female"  "male"
[17] "male"    "male"    "female"  "female"  "female"  "male"    "male"    "female"
[25] "female"
```

```
rawdata[, 2] # returns Tibble with 1 column
```

```
# A tibble: 25 x 1
  Sex
  <chr>
1 male
2 male
3 male
4 male
5 female
6 male
```

```
7 male
8 male
9 male
10 male
# i 15 more rows
```

```
rawdata[, 2:3] # returns tibble with 2 columns
```

```
# A tibble: 25 x 2
  Sex    Ethnicity
  <chr>  <fct>
1 male   White (not Hispanic)
2 male   White (not Hispanic)
3 male   White (not Hispanic)
4 male   American Indian or Native Alaskan
5 female White (not Hispanic)
6 male   White (not Hispanic)
7 male   White (not Hispanic)
8 male   White (not Hispanic)
9 male   White (not Hispanic)
10 male  White (not Hispanic)
# i 15 more rows
```

```
rawdata_df[2] # returns DF with 1 column
```

```
      Sex
1    male
2    male
3    male
4    male
5  female
6    male
7    male
8    male
9    male
10   male
11   male
12   male
13   male
14   male
15 female
16   male
17   male
```

```

18 male
19 female
20 female
21 female
22 male
23 male
24 female
25 female

rawdata_df[[2]] # returns vector

[1] "male"   "male"   "male"   "male"   "female" "male"   "male"   "male"
[9] "male"   "male"   "male"   "male"   "male"   "male"   "female" "male"
[17] "male"   "male"   "female" "female" "female" "male"   "male"   "female"
[25] "female"

rawdata_df[, 2] # returns vector

[1] "male"   "male"   "male"   "male"   "female" "male"   "male"   "male"
[9] "male"   "male"   "male"   "male"   "male"   "male"   "female" "male"
[17] "male"   "male"   "female" "female" "female" "male"   "male"   "female"
[25] "female"

rawdata_df[, 2:3] # returns DF with 2 columns

      Sex          Ethnicity
1    male  White (not Hispanic)
2    male  White (not Hispanic)
3    male  White (not Hispanic)
4    male American Indian or Native Alaskan
5  female  White (not Hispanic)
6    male  White (not Hispanic)
7    male  White (not Hispanic)
8    male  White (not Hispanic)
9    male  White (not Hispanic)
10   male  White (not Hispanic)
11   male Middle-Eastern, Arabic
12   male  White (not Hispanic)
13   male  White (not Hispanic)
14   male  White (not Hispanic)
15 female  White (not Hispanic)
16   male  White (not Hispanic)

```

```

17 male           White (not Hispanic)
18 male           White (not Hispanic)
19 female         White (not Hispanic)
20 female         White (not Hispanic)
21 female         White (not Hispanic)
22 male           White (not Hispanic)
23 male           White (not Hispanic)
24 female         White (not Hispanic)
25 female         White (not Hispanic)

```

There are specific functions for ‘picking’ columns or rows, especially useful in pipes.

```
rawdata |> select(PatID:Ethnicity, `sysRR (mmHg)` :HR)
```

```
# A tibble: 25 x 6
  PatID Sex   Ethnicity      `sysRR (mmHg)` `diaRR (mmHg)`    HR
  <chr> <chr> <fct>          <dbl>          <dbl> <dbl>
1 P 1   male  White (not Hispanic)     135            82   90
2 P 2   male  White (not Hispanic)     129            66   80
3 P 3   male  White (not Hispanic)     128            47   94
4 P 4   male  American Indian or Native A~ 133            67   92
5 P 5   female White (not Hispanic)     159            68   95
6 P 6   male  White (not Hispanic)     153            89  100
7 P 7   male  White (not Hispanic)     126            86   86
8 P 8   male  White (not Hispanic)     158            64   90
9 P 9   male  White (not Hispanic)     121            70   91
10 P 10  male  White (not Hispanic)    106            57   90
# i 15 more rows
```

```
rawdata |>
  select(PatID:Ethnicity, `sysRR (mmHg)` :HR) |>
  slice(1:5)
```

```
# A tibble: 5 x 6
  PatID Sex   Ethnicity      `sysRR (mmHg)` `diaRR (mmHg)`    HR
  <chr> <chr> <fct>          <dbl>          <dbl> <dbl>
1 P 1   male  White (not Hispanic)     135            82   90
2 P 2   male  White (not Hispanic)     129            66   80
3 P 3   male  White (not Hispanic)     128            47   94
4 P 4   male  American Indian or Native A~ 133            67   92
5 P 5   female White (not Hispanic)     159            68   95
```

```
rawdata |> select(contains("RR", ignore.case = F))
```

```
# A tibble: 25 x 2
`sysRR (mmHg)` `diaRR (mmHg)`
<dbl>          <dbl>
1       135          82
2       129          66
3       128          47
4       133          67
5       159          68
6       153          89
7       126          86
8       158          64
9       121          70
10      106          57
# i 15 more rows
```

```
rawdata |> select(ends_with("r"))
```

```
# A tibble: 25 x 1
HR
<dbl>
1   90
2   80
3   94
4   92
5   95
6  100
7   86
8   90
9   91
10  90
# i 15 more rows
```

```
rawdata |> select(-contains("name"))
```

```
# A tibble: 25 x 7
  PatID Sex   Ethnicity Treatment `sysRR (mmHg)` `diaRR (mmHg)`    HR
  <chr> <chr>   <fct>     <chr>           <dbl>          <dbl> <dbl>
1 P 1   male   White (not Hispan~ Verum            135            82   90
2 P 2   male   White (not Hispan~ Verum            129            66   80
3 P 3   male   White (not Hispan~ Verum            128            47   94
```

```

4 P 4   male   American Indian o~ Verum           133      67    92
5 P 5   female  White (not Hispan~ Placebo       159      68    95
6 P 6   male   White (not Hispan~ Placebo       153      89    100
7 P 7   male   White (not Hispan~ Verum        126      86    86
8 P 8   male   White (not Hispan~ Placebo       158      64    90
9 P 9   male   White (not Hispan~ Placebo       121      70    91
10 P 10  male   White (not Hispan~ Verum       106      57    90
# i 15 more rows

```

```
rawdata |> select(where(is.numeric))
```

```

# A tibble: 25 x 3
`sysRR (mmHg)` `diaRR (mmHg)`   HR
                <dbl>          <dbl> <dbl>
1              135            82    90
2              129            66    80
3              128            47    94
4              133            67    92
5              159            68    95
6              153            89   100
7              126            86    86
8              158            64    90
9              121            70    91
10             106            57    90
# i 15 more rows

```

```
rawdata |> select(`sysRR (mmHg)`)
```

```

# A tibble: 25 x 1
`sysRR (mmHg)`
                <dbl>
1              135
2              129
3              128
4              133
5              159
6              153
7              126
8              158
9              121
10             106
# i 15 more rows

```

```
rawdata |> select(contains("r"), -contains("rr"))
```

```
# A tibble: 25 x 2
  Treatment    HR
  <chr>      <dbl>
1 Verum        90
2 Verum        80
3 Verum        94
4 Verum        92
5 Placebo     95
6 Placebo     100
7 Verum        86
8 Placebo     90
9 Placebo     91
10 Verum       90
# i 15 more rows
```

```
rawdata |> pull(`sysRR (mmHg)`)
```

```
[1] 135 129 128 133 159 153 126 158 121 106 137 140 109 108 122 112 147 121 167
[20] 139 130 120 122 126 125
```

### 3.6.3 *Exercise: tibble*

Think of a cruet\_stand / Gewürzmenage

- define n\_elements <- 5\*10^3
- create a tibble “menage” with columns saltshaker, peppercaster and n\_elements each for saltgrain and pepperflake
- print saltshaker (tibble with 1 columns)
- print salt (content of column, all saltgrains)
- print 100 saltgrains



### 3.6.4 List

While matrix, data.frames, and tibbles always have the same number of rows for each column, sometimes different lengths are required. A list can handle all kinds of data with different number of elements for each sublist. This is a typical output format for statistical functions and is useful for collecting e.g. result tables or figures. Package rlist provides useful tools.

#### 3.6.4.1 Creation

```
shopping <- list(
  beverages = c(
    "beer", "water",
    "gin(not Gordons!!)", "tonic"
  ),
  snacks = c("chips", "pretzels"),
  nonfood = c("DVDs", "Akku"),
  mengen = 1:10,
  volumen = rnorm(50, 100, 2)
)
shopping
```

```
$beverages
[1] "beer"           "water"          "gin(not Gordons!!)"
[4] "tonic"
```

```

$snacks
[1] "chips"      "pretzels"

$nonfood
[1] "DVDs"       "Akku"

$mengen
[1] 1 2 3 4 5 6 7 8 9 10

$volumen
[1] 100.90487 99.00849 100.10589 99.40560 99.43487 102.74559 100.82905
[8] 102.18185 98.31797 98.94987 100.47771 103.06376 99.59920 99.84288
[15] 101.44851 101.38885 100.61377 99.37406 101.59994 98.19405 96.22721
[22] 99.50355 100.53137 101.34237 96.98513 101.88614 99.97876 102.21222
[29] 98.64954 102.05933 99.66065 100.00861 99.25600 99.84849 98.23560
[36] 100.44805 99.19763 99.96392 99.66506 100.63817 103.00076 98.21063
[43] 101.42869 99.00265 99.49460 99.99962 100.80369 102.78947 97.25760
[50] 99.93733

```

### 3.6.4.2 Indexing

```
shopping$snacks
```

```
[1] "chips"      "pretzels"
```

```
shopping[1] # returns a list
```

```

$beverages
[1] "beer"           "water"          "gin(not Gordons!!)"
[4] "tonic"

```

```
shopping[[1]] # returns a vector
```

```
[1] "beer"           "water"          "gin(not Gordons!!)"
[4] "tonic"
```

```
str(shopping[1])
```

```

List of 1
$ beverages: chr [1:4] "beer" "water" "gin(not Gordons!!)" "tonic"

```

```
str(shopping[[1]])
```

```
chr [1:4] "beer" "water" "gin(not Gordons!!)" "tonic"
```

```
str(shopping$beverages)
```

```
chr [1:4] "beer" "water" "gin(not Gordons!!)" "tonic"
```

```
shopping[1] [2]
```

```
$<NA>  
NULL
```

```
shopping[[1]][2]
```

```
[1] "water"
```

```
shopping$beverages[2]
```

```
[1] "water"
```

```
t_out <- t.test(  
  x = rnorm(n = 20, mean = 10, sd = 1),  
  y = rnorm(20, 12, 1)  
)  
str(t_out)
```

```
List of 10  
 $ statistic : Named num -5.73  
   ..- attr(*, "names")= chr "t"  
 $ parameter : Named num 37.3  
   ..- attr(*, "names")= chr "df"  
 $ p.value   : num 1.43e-06  
 $ conf.int  : num [1:2] -2.66 -1.27  
   ..- attr(*, "conf.level")= num 0.95  
 $ estimate  : Named num [1:2] 10.2 12.1  
   ..- attr(*, "names")= chr [1:2] "mean of x" "mean of y"
```

```
$ null.value : Named num 0
..- attr(*, "names")= chr "difference in means"
$ stderr      : num 0.343
$ alternative: chr "two.sided"
$ method      : chr "Welch Two Sample t-test"
$ data.name   : chr "rnorm(n = 20, mean = 10, sd = 1) and rnorm(20, 12, 1)"
- attr(*, "class")= chr "htest"
```

```
t_out$p.value
```

```
[1] 1.426456e-06
```

```
t_out |> pluck("p.value")
```

```
[1] 1.426456e-06
```

## 3.7 Control structures

### 3.7.1 Loops

Repetitive tasks like computation of descriptive statistics over many variables or repeated simulations of data can be declared inside of a loop. There are functions (like summarize(across(...))) that create those repetitions internally, but often doing this explicitly improves readability or helps solving various tasks like describing AND plotting data.

#### 3.7.1.1 for-loop

In a for-loop, we can define the number of runs in advance, e.g. by the number of variables to describe. There are 2 ways/styles, how to define this number:

1. by creating an index variable with an integer vector 1,2,3, ... number of runs/variables
2. by creating an index containing e.g. colnames

```
# integer index
print("### Game of Loops ###")
```

```
[1] "### Game of Loops ###"
```

```

for (season_i in 1:3) {
  cat(paste("GoL Season", season_i, "\n"))
  for (episode_i in 1:5) {
    cat(paste0(
      "  GoL S.", season_i,
      " Episode ", episode_i, "\n"
    ))
  }
  cat("\n")
}

```

GoL Season 1  
 GoL S.1 Episode 1  
 GoL S.1 Episode 2  
 GoL S.1 Episode 3  
 GoL S.1 Episode 4  
 GoL S.1 Episode 5

GoL Season 2  
 GoL S.2 Episode 1  
 GoL S.2 Episode 2  
 GoL S.2 Episode 3  
 GoL S.2 Episode 4  
 GoL S.2 Episode 5

GoL Season 3  
 GoL S.3 Episode 1  
 GoL S.3 Episode 2  
 GoL S.3 Episode 3  
 GoL S.3 Episode 4  
 GoL S.3 Episode 5

```

# content index
## names of elements
for (col_i in colnames(rawdata)) {
  print(str(rawdata |> pull(col_i)))
}

```

```

chr [1:25] "P 1" "P 2" "P 3" "P 4" "P 5" "P 6" "P 7" "P 8" "P 9" "P 10" ...
NULL
chr [1:25] "male" "male" "male" "male" "female" "male" "male" "male" ...
NULL
Factor w/ 6 levels "American Indian or Native Alaskan",...: 5 5 5 1 5 5 5 5 5 ...

```

```

NULL
chr [1:25] "Austin" "Peter" "Jeffrey" "Hament" "Jennifer" "Nathan" ...
NULL
chr [1:25] "Collins" "Atkins" "Williamson" "Maez" "Allen" "Potter" ...
NULL
chr [1:25] "Verum" "Verum" "Verum" "Verum" "Placebo" "Placebo" "Verum" ...
NULL
num [1:25] 135 129 128 133 159 153 126 158 121 106 ...
NULL
num [1:25] 82 66 47 67 68 89 86 64 70 57 ...
NULL
num [1:25] 90 80 94 92 95 100 86 90 91 90 ...
NULL

```

```

## content of elements
for (col_i in shopping) {
  print(col_i)
}

```

```

[1] "beer"           "water"          "gin(not Gordons!!)"
[4] "tonic"
[1] "chips"         "pretzels"
[1] "DVDs"          "Akku"
[1] 1 2 3 4 5 6 7 8 9 10
[1] 100.90487 99.00849 100.10589 99.40560 99.43487 102.74559 100.82905
[8] 102.18185 98.31797 98.94987 100.47771 103.06376 99.59920 99.84288
[15] 101.44851 101.38885 100.61377 99.37406 101.59994 98.19405 96.22721
[22] 99.50355 100.53137 101.34237 96.98513 101.88614 99.97876 102.21222
[29] 98.64954 102.05933 99.66065 100.00861 99.25600 99.84849 98.23560
[36] 100.44805 99.19763 99.96392 99.66506 100.63817 103.00076 98.21063
[43] 101.42869 99.00265 99.49460 99.99962 100.80369 102.78947 97.25760
[50] 99.93733

```

```

# automatic creation of integer index from elements
# for(col_i in 1:ncol(rawdata)){
for (col_i in seq_along(colnames(rawdata))) {
  print(colnames(rawdata)[col_i])
}

```

```

[1] "PatID"
[1] "Sex"
[1] "Ethnicity"
[1] "Given name"

```

```
[1] "Family name"  
[1] "Treatment"  
[1] "sysRR (mmHg)"  
[1] "diaRR (mmHg)"  
[1] "HR"
```

```
for (col_i in seq_len(length(colnames(rawdata)))) {  
  print(colnames(rawdata)[col_i])  
}
```

```
[1] "PatID"  
[1] "Sex"  
[1] "Ethnicity"  
[1] "Given name"  
[1] "Family name"  
[1] "Treatment"  
[1] "sysRR (mmHg)"  
[1] "diaRR (mmHg)"  
[1] "HR"
```

```
for (col_i in 1:length(colnames(rawdata))) {  
  print(colnames(rawdata)[col_i])  
}
```

```
[1] "PatID"  
[1] "Sex"  
[1] "Ethnicity"  
[1] "Given name"  
[1] "Family name"  
[1] "Treatment"  
[1] "sysRR (mmHg)"  
[1] "diaRR (mmHg)"  
[1] "HR"
```

```
# edge-case of 0 elements -> 0 runs  
n_gaussvars <- 0  
for (col_i in seq_len(n_gaussvars)) {  
  print(colnames(rawdata)[col_i])  
}  
  
n_gaussvars <- 0  
for (col_i in 1:n_gaussvars) {
```

```
    print(colnames(rawdata)[col_i])  
}
```

```
[1] "PatID"  
character(0)
```

### 3.7.1.2 *while-loops*

If not number of repetitions is know, but a condition.

```
test <- 0  
while (test < 10) {  
  print(test)  
  test <- test + 1  
}
```

```
[1] 0  
[1] 1  
[1] 2  
[1] 3  
[1] 4  
[1] 5  
[1] 6  
[1] 7  
[1] 8  
[1] 9
```

### 3.7.2 *Conditions*

#### 3.7.2.1 *if else*

We can run code if condition(s) are true:

```
sex <- "female"  
if (sex == "male") {  
  print("Male")  
} else {  
  print("Female")  
}
```

```
[1] "Female"
```

```
if (sex == "male") {  
  print("Male")  
}
```

```
if (sex != "male") {  
  print("Female")  
}
```

```
[1] "Female"
```

```
testvar <- 4  
if (testvar %in% c(1, 3, 5)) {  
  print("uneven")  
} else {  
  print("probably even")  
}
```

```
[1] "probably even"
```

```
TRUE & FALSE # AND
```

```
[1] FALSE
```

```
all(TRUE, FALSE)
```

```
[1] FALSE
```

```
(1 < 10) & (sex == "male")
```

```
[1] FALSE
```

```
all(1 < 10, sex == "male")
```

```
[1] FALSE
```

```
TRUE | FALSE # OR
```

```
[1] TRUE
```

```
any(TRUE, FALSE)
```

```
[1] TRUE
```

```
(1 > 10) | (1 < 5)
```

```
[1] TRUE
```

```
age <- 5  
(sex == "female" & age <= 12) | (sex == "male" & age <= 14)
```

```
[1] TRUE
```

```
if (  
  any(  
    all(sex == "female", age <= 12),  
    all(sex == "male", age <= 14)  
) ) {  
  cat("kid is still growing\n")  
}
```

```
kid is still growing
```

### 3.7.2.2 *ifelse*

We can get text conditionally:

```
test <- "female"  
print(ifelse(test == sex == "male",  
            yes = "is male",  
            no = "is female"  
) )
```

```
[1] "is female"
```

```
p <- .12  
paste0(  
  "That is ",
```

```

ifelse(test = p <= .05, yes = "", no = "not "),
"significant"
)

```

```
[1] "That is not significant"
```

```

if (p > .05) {
  sign_out <- "not "
} else {
  sign_out <- ""
}
paste0(
  "That is ",
  sign_out,
  "significant"
)

```

```
[1] "That is not significant"
```

### 3.7.2.3 *case\_when* / *case\_match*

When there are many tests to do, *case\_when* or *case\_match* are nice replacements for *ifelse*. While *case\_when* allows complex conditions, *case\_match* is used for simple comparisons:

```

rawdata <- mutate(
  .data = rawdata,
  Hypertension = case_when(
    `sysRR (mmHg)` < 120 & `diaRR (mmHg)` < 70 ~ "normotensive",
    `sysRR (mmHg)` < 160 & `diaRR (mmHg)` <= 80 ~ "borderline",
    .default = "hypertensive"
  ),
  `prescribe something?` = case_match(
    Hypertension,
    "hypertensive" ~ "yes",
    "borderline" ~ "possibly",
    "normotensive" ~ "no"
  )
)
rawdata |>
  select(contains("RR"), Hypertension, contains("pres"))

```

```
# A tibble: 25 x 4
`sysRR (mmHg)` `diaRR (mmHg)` Hypertension `prescribe something?`
<dbl>          <dbl> <chr>           <chr>
1      135          82 hypertensive yes
2      129          66 borderline   possibly
3      128          47 borderline   possibly
4      133          67 borderline   possibly
5      159          68 borderline   possibly
6      153          89 hypertensive yes
7      126          86 hypertensive yes
8      158          64 borderline   possibly
9      121          70 borderline   possibly
10     106          57 normotensive no
# i 15 more rows
```

```
p <- 0.07
paste0(
  "That is ",
  case_when(p <= .05 ~ "", p <= .1 ~ "borderline ", .default = "not "),
  "significant"
)
```

```
[1] "That is borderline significant"
```

### 3.7.3 Exercise: for and while loops, and if/ifelse/case\_xxx

#### 3.7.3.1 1. Analyzing Gene Expression Data (Using for loop and if/else)

**Scenario:** You have a dataset of gene expression levels (e.g., RNA-seq counts) for several genes across different samples. You want to identify genes that are either highly expressed or lowly expressed based on a threshold.

**Task:**

- **Create Sample Data:** Generate a numeric vector representing expression levels for 20 genes (mean=100, SD=30).
- **Classify Genes by their Expression Level:**
  - Using a `for` loop, iterate through each gene's expression level.
  - Inside the loop, use an `if/else` statement to categorize each gene:
    - \* If expression is above 120, print `paste("Gene", i, "is highly expressed:", gene_expression[i])`.

- \* If expression is below 70, print `paste("Gene", i, "is lowly expressed:", gene_expression[i])`.
- \* Otherwise, print `paste("Gene", i, "is moderately expressed:", gene_expression[i])`.
- Create a similar loop using `case_when` to simplify the rules.

### 3.7.3.2 2. Simulating Population Growth (Using while loop)

**Scenario:** You want to simulate the growth of a bacterial population over time until it reaches a certain threshold.

**Task:**

- **Set Initial Conditions:**
  - Start with `population_size <- 100`.
  - Set a `growth_rate <- 1.1` (meaning 10% increase per generation).
  - Set a `carrying_capacity <- 1000`.
  - Initialize `generation <- 0`.
- **Simulate Growth:**
  - Use a `while` loop that continues as long as `population_size < carrying_capacity`.
  - Inside the loop:
    - Update `population_size <- population_size * growth_rate`.
    - Increment `generation <- generation + 1`.
    - Print the `population_size` and `generation` at each step.
- **Add a Condition:** When at the last run of the loop, print a message indicating how many generations it took to reach the carrying capacity.

### 3.7.3.3 3. Classifying Species Based on Traits (Using for loop and ifelse)

**Scenario:** You have a dataset of different plant species and two of their traits: leaf length (cm) and number of petals. You want to classify them into broad groups.

**Task:**

- **Create Sample Data:**
  - `species <- c("Oak", "Maple", "Cherry", "Rose", "Lily", "Daisy", "Sunflower")`

```
- leaf_length <- c(15, 12, 8, 4, 18, 5, 25)
- petal_count <- c(0, 0, 5, 20, 6, 30, 0) (0 for trees, flowers have petals)
```

- **Categorize Species:**

- Create an empty vector `species_type <- character(length(species))` to store the results.
- Use a `for` loop to iterate through each species.
- Inside the loop, use nested `ifelse` statements to determine `species_type` based on these rules:
  - If `petal_count[i] == 0`, it's a “Tree”.
  - If `petal_count[i] > 10`, it's a “Many-petaled Flower”.
  - If `petal_count[i] > 0`, it's a “Few-petaled Flower”.
- Assign the result to `species_type[i]`.
- **Display Results:** After the loop, create a data frame or tibble with `species`, `leaf_length`, `petal_count`, and the new `species_type` column.

# 4 Regular expressions

## 4.1 Intro

[Regex cheatsheet](#)

[stringr cheatsheet](#)

Regular expressions are a **powerful** tool for searching and manipulating text data. They allow you to define specific patterns within sequences, which is particularly useful when analyzing biological data such as DNA or protein sequences. But more mundane, they are terribly useful for practical tasks like finding or renaming variables, correcting common typos, and checking input patterns.

Basic functions like `grep()` or `gsub()` are difficult to use in pipelines and not very intuitive, tidyverse functions from stringr like `str_detect()` or `str_replace()` are more verbose and easier to use.

**Key symbols and their meanings:**

- . (period): Matches any single character except a newline. For example, “A.T” would match “AAT”, “AGT”, “ACT”, etc.
- + means 1 ore more occurrences (+), ? means zero or 1 occurrence
- [] (square brackets): Defines a character class. Any character within the brackets will match. For example, “[ATGC]” matches any DNA nucleotide.
- {} (curly braces): Specify the number of occurrences of the preceding element. For example, “A{3}” matches exactly three consecutive “A”s, like in “AAA”.
- \d: Matches any digit (0-9). For example, “\d+” matches one or more digits, which could be useful for finding numerical identifiers in protein databases.
- \D: Matches any non-digit character.

### 4.1.1 Some basic examples:

```
pacman::p_load(tidyverse)
starttext <- "Did grandma eat all the pizza?"
str_detect(string = starttext, pattern = "pizza")
```

```
[1] TRUE
```

```
str_detect(string = starttext, pattern = "pasta")
```

```
[1] FALSE
```

```
str_detect(string = starttext, pattern = "e.+a\\?\\$")
```

```
[1] TRUE
```

```
str_view(string = starttext, pattern = "e.+a\\?\\$")
```

```
[1] | Did grandma <eat all the pizza?>
```

```
str_detect(string = starttext, pattern = "grand[mp]a")
```

```
[1] TRUE
```

```
str_view(string = starttext, pattern = "grand[mp]a")
```

```
[1] | Did <grandma> eat all the pizza?
```

```
str_replace(  
  string = starttext,  
  pattern = "ma",  
  replacement = "pa"  
)
```

```
[1] "Did grandpa eat all the pizza?"
```

```
str_replace(  
  string = starttext,  
  pattern = " all ",  
  replacement = " half "  
)
```

```
[1] "Did grandma eat half the pizza?"
```

```

str_replace(
  string = starttext,
  pattern = "^(\\w+) (\\w+) (.+)",
  replacement = "\\2 \\1 \\3"
) |>
  str_to_sentence()

```

[1] "Grandma did eat all the pizza?"

```

str_replace(
  string = starttext,
  pattern = "^(\\w+) (\\w*) (.* )\\?",
  replacement = "\\2 \\1 \\3!"
) |>
  str_to_sentence()

```

[1] "Grandma did eat all the pizza!"

```

str_replace_all(
  string = starttext,
  pattern = c(
    "ma" = "pa",
    "all" = "half",
    "izz" = "ast"
  )
)

```

[1] "Did grandpa eat half the pasta?"

#### 4.1.2 An example for their use in renaming variables:

```

temptibble <- tibble(
  cup_weigh = seq(10, 20, .5),
  CAPSIZE_cm = 5,
  height_of_cup_cm = rnorm(21, 10, .01)
)
colnames(temptibble)

```

[1] "cup\_weigh" "CAPSIZE\_cm" "height\_of\_cup\_cm"

```

rename_with(
  .data = temptibble,
  .fn = ~ str_replace_all(
    .x,
    c(
      "CAP" = "CUP",
      "_(cm)" = " [\\"1]",
      "(.*)_.+_(.+)( .*)" = "\\"2\\"1\\"3",
      "_" = ""
    )
  ) |>
  str_to_sentence()
) |>
  colnames()

```

```
[1] "Cupweigh"      "Cupsized [cm]"   "Cupheight [cm]"
```

## 4.2 Exercise

```

testset1 <- c(
  "Meier", "Mayer", "Maier", "Meyer", "Mayr", "Hans Meier",
  "Maya", "Mayor", "Faltermeyer", "Meierhoven"
)
# find all variations of the name "Meier" (not Maya or Mayor etc)

testset2 <- c("weight_mm", "height_cm", "age_yr", "temp_c")
# replace _ with space
# replace _ with space and add unit in brackets

testset3 <- c("1980_12_30", "13.04.2005", "2005/04/25", "24121990")
# transform into YYYY-MM-DD

testset4 <- c("pw2000", "That1sb3tt3r", "M@kesSense?", "NoDigits@this1")
# test pwd strength, rules: Upper, lower, special char, number, min 8 char long

```

# 5 Importing data

```
pacman::p_load(conflicted, tidyverse, wrappedtools,  
    readxl, readODS, foreign, haven, here)
```

## 5.1 Import from text files (.txt, .csv)

There are base functions like `read.csv()` and tidyverse-based updated ones like `read_csv()`. Different versions like `read_csv2()` or `read_delim()` have different settings for delimiters, number formats etc.

```
rawdata <- read_csv2(here("data/Medtest_e.csv"))
```

i Using `'',''` as decimal and `'..'` as grouping mark. Use ``read_delim()`` for more control.

```
Rows: 28 Columns: 25  
-- Column specification -----  
Delimiter: ";"  
chr (1): sex  
dbl (24): randomcode, included, finalized, testmedication, size, weight, sys...
```

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.

```
Medtest_e <- read_delim(here("data/Medtest_e.csv"),  
    delim = ";", escape_double = FALSE, locale = locale(date_names = "de",  
        decimal_mark = ",", grouping_mark = ".")  
    trim_ws = TRUE)
```

```
Rows: 28 Columns: 25  
-- Column specification -----  
Delimiter: ";"  
chr (1): sex  
dbl (24): randomcode, included, finalized, testmedication, size, weight, sys...
```

```
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.
```

```
View(Medtest_e)
```

## 5.2 Import from Excel

### 5.2.1 Tidy Excel files

```
rawdata <- read_excel(path = here("data/Medtest_e.xlsx")) |>  
  rename_with(.fn = ~str_replace_all(.x, pattern = "_",  
                                     replacement = " ")) |>  
  rename_with(.fn = str_to_title,  
              .cols = !contains(c("BP", "BMI", "NY"))) |>  
  rename(`Size (cm)` = Size, #newname=oldname  
        `Weight (kg)` = Weight) |>  
  select(-`Sex M`)  
  saveRDS(rawdata, file = here("data/rawdata.rds"))
```

### 5.2.2 Excel file with units row

1. Import names section and data section separately
2. Loop over all columns
  1. test for existence of unit, if not NA
  2. paste 1st row, "[, 2nd row,]"
3. Use 1st row as colnames for data

```
cn_temp <- read_excel(path = here("data/Medtest_e.xlsx"),  
                      range = "A1:Y2", col_names = FALSE,  
                      sheet = 2)
```

```
New names:  
* `` -> `...1`  
* `` -> `...2`  
* `` -> `...3`  
* `` -> `...4`  
* `` -> `...5`  
* `` -> `...6`
```

```
* `` -> `...7`  
* `` -> `...8`  
* `` -> `...9`  
* `` -> `...10`  
* `` -> `...11`  
* `` -> `...12`  
* `` -> `...13`  
* `` -> `...14`  
* `` -> `...15`  
* `` -> `...16`  
* `` -> `...17`  
* `` -> `...18`  
* `` -> `...19`  
* `` -> `...20`  
* `` -> `...21`  
* `` -> `...22`  
* `` -> `...23`  
* `` -> `...24`  
* `` -> `...25`
```

```
for(col_i in colnames(cn_temp)){  
  if(!is.na(cn_temp[1, col_i])){  
    cn_temp[1, col_i] <-  
      paste0(cn_temp[1, col_i], " [", cn_temp[2, col_i], "]")  
  }  
}  
  
rawdata <- read_excel(path = here("data/Medtest_e.xlsx"),  
                      skip = 2, col_names = FALSE,  
                      sheet = 2)
```

```
New names:  
* `` -> `...1`  
* `` -> `...2`  
* `` -> `...3`  
* `` -> `...4`  
* `` -> `...5`  
* `` -> `...6`  
* `` -> `...7`  
* `` -> `...8`  
* `` -> `...9`  
* `` -> `...10`  
* `` -> `...11`  
* `` -> `...12`  
* `` -> `...13`
```

```
* `` -> `...14`  
* `` -> `...15`  
* `` -> `...16`  
* `` -> `...17`  
* `` -> `...18`  
* `` -> `...19`  
* `` -> `...20`  
* `` -> `...21`  
* `` -> `...22`  
* `` -> `...23`  
* `` -> `...24`  
* `` -> `...25`
```

```
colnames(rawdata) <- cn_temp[1,]
```

## 5.3 ODS files

Package readODS provides similar functionality for OpenOffice/LibreOffice files.

### 5.3.1 Import from SPSS/SAS

Import from SPSS generic files is implemented in various packages:

- foreign::read.spss is a more base approach,
  - on the positive side it has an option to read in value labels
  - on the other hand it returns lists or data frames, so casting into tibble is advised
- haven::read\_sav comes from tidyverse
  - variable- and value-labels are imported into attributes
  - as\_factor uses value labels

```
import1 <- read.spss(file = here("data/Zellbeads.sav"),  
                      to.data.frame = TRUE,  
                      use.value.labels = TRUE)
```

Zurückkodierung von CP1252

```
str(import1)
```



```
import2 <- mutate(import2,  
                  Bedingung=as_factor(Bedingung))  
str(import2$Bedingung)
```

Factor w/ 3 levels "Kontrolle","AngII",...: 1 1 1 1 1 1 1 1 1 1 ...

## 6 Changing structure wide <-> long

There are many examples and explanations in the tidyR cheatsheet

<https://rstudio.github.io/cheatsheets/html/tidyr.html>

and the vignette:

<https://tidyverse.org/articles/pivot.html>

When working with repeated measures (e.g. follow-ups or changes over shorter periods of time) there are two typical formats:

1. wide data:

ID	Var1Time1	Var1Time2	Var2Time1	Var2Time2
P1				
P2				
P3				

2. long data

ID	Time	Var1	Var2
P1	1		
P1	2		
P2	1		
P2	2		
P3	1		
P3	2		

While long data can be seen as the tidier version and is necessary for many statistical procedures, the wide format makes computation of differences and procedures as the t-test for dependent samples easier. Package `tidyR` provides the functions `pivot_wider` and `pivot_longer` for conversions between those forms. Another use-case is the plotting of several variables into ggplot facets, which can be achieved by combining those variables into a single column. Summarizing several variables and grouped data may result in a wide table with groups as rows and variables as columns, contrary to the common opposite form, another use-case.

```
pacman::p_load(conflicted,tidyverse, wrappedtools)
```

## 6.1 Example 1: single repeated measure

```
n <- 3
wide_data <- tibble(ID = paste("P",1:n),
                      Var1 = LETTERS[1:n],
                      Var2Time1 = rnorm(n = n, mean = 100, sd = 15),
                      Var2Time2 = Var2Time1 + rnorm(n,10,5))
wide_data

# A tibble: 3 x 4
#>   ID   Var1  Var2Time1  Var2Time2
#>   <chr> <chr>     <dbl>      <dbl>
#> 1 P 1    A        103.       113.
#> 2 P 2    B        70.9       78.5
#> 3 P 3    C        88.9       90.2
```

```
long_data <- pivot_longer(
  data = wide_data,
  cols = contains("Time")
)
long_data
```

```
# A tibble: 6 x 4
#>   ID   Var1  name      value
#>   <chr> <chr> <chr>     <dbl>
#> 1 P 1    A    Var2Time1 103.
#> 2 P 1    A    Var2Time2 113.
#> 3 P 2    B    Var2Time1  70.9
#> 4 P 2    B    Var2Time2  78.5
#> 5 P 3    C    Var2Time1  88.9
#> 6 P 3    C    Var2Time2  90.2
```

## 6.2 Example 2: several repeated measures

```
set.seed(42)
wide_data <- tibble(ID = paste("P",1:n),
                      Var1Time1 = rnorm(n = n, mean = 100, sd = 15),
                      Var1Time2 = Var1Time1 + rnorm(n,10,5),
                      Var2Time1 = rnorm(n = n, mean = 10, sd = 2),
                      Var2Time2 = Var2Time1 + rnorm(n,0,1),
                      Var3 = LETTERS[1:n])
```

```
wide_data
```

```
# A tibble: 3 x 6
  ID    Var1Time1 Var1Time2 Var2Time1 Var2Time2 Var3
  <chr>     <dbl>     <dbl>     <dbl>     <dbl> <chr>
1 P 1       121.      134.      13.0      13.0 A
2 P 2        91.5      104.      9.81      11.1 B
3 P 3       105.      115.      14.0      16.3 C
```

```
# version with intermediate step:
very_long_data <- pivot_longer(
  data = wide_data,
  cols = contains("Time"),
  names_to = c("Variable", "Time"),
  names_pattern = "(Var\\d+)(Time[12])",
  values_to = "Value"
)
very_long_data
```

```
# A tibble: 12 x 5
  ID    Var3 Variable Time   Value
  <chr> <chr> <chr>   <chr>  <dbl>
1 P 1   A     Var1    Time1  121.
2 P 1   A     Var1    Time2  134.
3 P 1   A     Var2    Time1  13.0
4 P 1   A     Var2    Time2  13.0
5 P 2   B     Var1    Time1  91.5
6 P 2   B     Var1    Time2  104.
7 P 2   B     Var2    Time1  9.81
8 P 2   B     Var2    Time2  11.1
9 P 3   C     Var1    Time1  105.
10 P 3  C     Var1    Time2  115.
11 P 3  C     Var2    Time1  14.0
12 P 3  C     Var2    Time2  16.3
```

```
long_data <- pivot_wider(very_long_data,
                         names_from = Variable,
                         values_from = Value)
long_data
```

```
# A tibble: 6 x 5
  ID    Var3 Time   Var1  Var2
  <chr> <chr> <dbl> <dbl> <dbl>
```

```

<chr> <chr> <chr> <dbl> <dbl>
1 P 1     A      Time1 121. 13.0
2 P 1     A      Time2 134. 13.0
3 P 2     B      Time1 91.5 9.81
4 P 2     B      Time2 104. 11.1
5 P 3     C      Time1 105. 14.0
6 P 3     C      Time2 115. 16.3

```

Alternatively in 1 step, value column names will be extracted from parts of the wide colnames.  
This requires either a name pattern or a separator:

```

long_data <- pivot_longer(
  data=wide_data,
  cols=contains("Time"),
  names_to = c(".value","Time"), # .value will be replaced dynamically
  #names_sep = "Time"
  names_pattern = "(Var\\d+)(Time\\d+)"
)
long_data

```

```

# A tibble: 6 x 5
  ID    Var3  Time   Var1  Var2
  <chr> <chr> <chr> <dbl> <dbl>
1 P 1     A      Time1 121. 13.0
2 P 1     A      Time2 134. 13.0
3 P 2     B      Time1 91.5 9.81
4 P 2     B      Time2 104. 11.1
5 P 3     C      Time1 105. 14.0
6 P 3     C      Time2 115. 16.3

```

### 6.3 Example 3: long to wide

```

wide_again_data <- pivot_wider(
  data = long_data,
  names_from = Time,
  values_from = Var1:Var2,
  names_glue = "{.value}@{Time}"
)
wide_again_data

```

```

# A tibble: 3 x 6
  ID    Var3  `Var1@Time1` `Var1@Time2` `Var2@Time1` `Var2@Time2`
  <chr> <chr> <dbl> <dbl> <dbl> <dbl>
1 P 1     A      121. 13.0 105. 14.0
2 P 1     A      134. 13.0 115. 16.3
3 P 2     B      91.5 9.81 104. 11.1

```

		<dbl>	<dbl>	<dbl>
1	P 1	A	121.	134.
2	P 2	B	91.5	104.
3	P 3	C	105.	115.
			13.0	13.0
			9.81	11.1
			14.0	16.3

## 6.4 More examples

### 6.4.1 Step 1: Create example data:

- 5 subjects per group, 2 groups A/B
- 3 measurements weight (V1, V2, V3) with random numbers,
  - means 46, 50, 51
  - SDs 2
- 2 measurements length (V1, V3) #no visit 2!
  - means 120, 135
  - SDs 3

```
set.seed(42)
n <- 10
rawdata <-
  tibble(ID=paste("Pat",seq_len(n), sep="#"),
         groups=rep(c("A","B"), each=n/2),
         weight_V1=rnorm(n = n,mean = 46,sd = 2),
         weight_V2=rnorm(n = n,mean = 50,sd = 2),
         weight_V3=rnorm(n = n,mean = 51,sd = 2),
         size_V1=rnorm(n = n,mean = 120,sd = 3),
         size_V3=rnorm(n = n,mean = 135,sd = 3))

head(rawdata)
```

```
# A tibble: 6 x 7
  ID    groups weight_V1 weight_V2 weight_V3 size_V1 size_V3
  <chr> <chr>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1 Pat#1 A        48.7      52.6      50.4     121.     136.
2 Pat#2 A        44.9      54.6      47.4     122.     134.
3 Pat#3 A        46.7      47.2      50.7     123.     137.
4 Pat#4 A        47.3      49.4      53.4     118.     133.
5 Pat#5 A        46.8      49.7      54.8     122.     131.
6 Pat#6 B        45.8      51.3      50.1     115.     136.
```

#### 6.4.2 Step 2: Transform that data to a long form:

- 1 column for weight
- 1 column for length
- 1 column for measurement time named “Visit”

```
# with intermediate super_long step
rawdata_long <-
  # to superlong
  pivot_longer(data = rawdata,
    cols = contains("V"),
    names_to = c("what_was_measured",
                "Visit"),
    names_sep = "_",
    values_to = "weight_or_size") |>
  # from superlong to long
  pivot_wider(names_from = what_was_measured,
              values_from = weight_or_size)
head(rawdata_long)
```

```
# A tibble: 6 x 5
  ID   groups Visit weight size
  <chr> <chr>  <chr>  <dbl> <dbl>
1 Pat#1 A      V1     48.7 121.
2 Pat#1 A      V2     52.6  NA
3 Pat#1 A      V3     50.4 136.
4 Pat#2 A      V1     44.9 122.
5 Pat#2 A      V2     54.6  NA
6 Pat#2 A      V3     47.4 134.
```

```
# single step approach
rawdata_long2 <-
  pivot_longer(data = rawdata,
    cols = contains("V"),
    names_to = c(".value","Visit"),
    # .value will be replaced by weigh or size
    names_sep = "_")
head(rawdata_long2)
```

```
# A tibble: 6 x 5
  ID   groups Visit weight size
  <chr> <chr>  <chr>  <dbl> <dbl>
1 Pat#1 A      V1     48.7 121.
2 Pat#1 A      V2     52.6  NA
```

```

3 Pat#1 A      V3      50.4 136.
4 Pat#2 A      V1      44.9 122.
5 Pat#2 A      V2      54.6 NA
6 Pat#2 A      V3      47.4 134.

```

#### 6.4.3 Step 3 Transform long to wide

```

# 2-steps
rawdata_wide <-
  pivot_longer(rawdata_long,
    cols = c(weight, size),
    names_to = "what_was_measured",
    values_to = "weight_or_size") |>
  pivot_wider(names_from=c(what_was_measured,Visit),
    # names created from 2 sources
    values_from = weight_or_size,
    names_sep = "_")
head(rawdata_wide)

```

```

# A tibble: 6 x 8
  ID   groups weight_V1 size_V1 weight_V2 size_V2 weight_V3 size_V3
  <chr> <chr>     <dbl>   <dbl>     <dbl>   <dbl>     <dbl>   <dbl>
1 Pat#1 A       48.7    121.      52.6    NA       50.4    136.
2 Pat#2 A       44.9    122.      54.6    NA       47.4    134.
3 Pat#3 A       46.7    123.      47.2    NA       50.7    137.
4 Pat#4 A       47.3    118.      49.4    NA       53.4    133.
5 Pat#5 A       46.8    122.      49.7    NA       54.8    131.
6 Pat#6 B       45.8    115.      51.3    NA       50.1    136.

```

```

# 1step option
rawdata_wide2 <-
  pivot_wider(rawdata_long,
    values_from = c(weight,size),
    # values come from 2 sources, names will used in names_glue
    names_from=Visit,
    names_glue=".value}_{Visit}")
head(rawdata_wide2)

```

```

# A tibble: 6 x 8
  ID   groups weight_V1 weight_V2 weight_V3 size_V1 size_V2 size_V3
  <chr> <chr>     <dbl>     <dbl>     <dbl>   <dbl>   <dbl>   <dbl>
1 Pat#1 A       48.7      52.6      50.4    121.     NA     136.
2 Pat#2 A       44.9      54.6      47.4    122.     NA     134.

```

3 Pat#3 A	46.7	47.2	50.7	123.	NA	137.
4 Pat#4 A	47.3	49.4	53.4	118.	NA	133.
5 Pat#5 A	46.8	49.7	54.8	122.	NA	131.
6 Pat#6 B	45.8	51.3	50.1	115.	NA	136.

## 6.5 Exercise: Reshaping Data

**Scenario:** You are a biologist studying the growth of chicks under different diets. The `ChickWeight` dataset in R contains observations on the weight of chicks over time, grouped by individual chick and diet. Understanding how to reshape data is crucial for different types of analyses and visualizations.

**Goal:** Practice transforming data between “long” and “wide” formats using `pivot_wider()` and `pivot_longer()`.

### 6.5.1 Part 1: From Long to Wide (`pivot_wider()`)

Sometimes, you might want to see all the measurements for a single individual (e.g., a chick) laid out in one row, with each time point becoming a separate column. This “wide” format can be useful for quick visual comparison of an individual’s trajectory or for certain statistical tests that expect a specific column structure.

#### Instructions:

##### 1. Inspect the original data:

- Type `head(ChickWeight)` and `str(ChickWeight)` to understand its current “long” format (each row is a single observation of weight at a specific time for a specific chick).
- Notice the `Time` and `weight` columns.

##### 2. Transform to Wide Format:

- Use `pivot_wider()` to transform the `ChickWeight` dataset.
- You want `Time` values to become new column names.
- You want the `weight` values to fill these new columns.
- Each row should represent a unique `Chick` and `Diet`.
- **Hint:** Think about `id_cols`, `names_from`, and `values_from`.

##### 3. Reflect:

- What are the new column names?
- What does `NA` mean in this new `chick_weight_wide` dataset? (Hint: Not all chicks were measured at all time points, or some started later).

### 6.5.1.1 Part 2: From Wide to Long (`pivot_longer()`)

Imagine you received data from a collaborator where each time point (e.g., Day 0, Day 2, Day 4, etc.) is its own column. For many biological analyses, especially for plotting time series with `ggplot2` or running mixed-effects models, you need this data in a “long” format, where all the measurements are in a single `weight` column, and there’s a separate `Time` column indicating when that measurement was taken.

#### Instructions:

1. **Start with the wide data:** We will use the `chick_weight_wide` tibble you created in Part 1.
2. **Transform to Long Format:**
  - Use `pivot_longer()` on `chick_weight_wide` (or `chick_weight_wide_example`).
  - You want to gather all the columns that represent `Time` points (e.g., 0, 2, 4, ..., 21) into two new columns: one for the `Time` itself and one for the `weight` measurement.
  - **Hint:** Think about `cols`, `names_to`, and `values_to`. You might also need `names_transform` to convert the `Time` column back to a numeric type.
3. **Reflect:**
  - Compare `chick_weight_long` to the original `ChickWeight` dataset. Are they identical in structure (ignoring the order of columns/rows which might vary slightly)?

# 7 Visualize data with ggplot

While there are various packages providing visualizations, here we are focusing on `ggplot2` (grammar of graphics) as a very flexible and versatile approach with many extensions implemented in additional packages. See e.g. <https://exts.ggplot2.tidyverse.org/>.

```
pacman::p_load(conflicted, tidyverse, here,
                 grid, gridExtra, car,
                 ggsci, ggsignif, ggthemes, ggridges,
                 # ganimate,
                 ggforce,
                 ggbeeswarm,
                 wrappedtools,
                 emojiifont,
                 patchwork,
                 GGally)
conflicts_prefer(dplyr::filter,
                  ggplot2::mean_cl_boot) # solves name conflict
```

```
[conflicted] Will prefer dplyr::filter over any other package.
[conflicted] Will prefer ggplot2::mean_cl_boot over any other package.
```

## 7.1 Example data

The typical examples use either diamonds from `ggplot2` or mtcars from `datasets`. There are help files for both.

```
head(diamonds)

# A tibble: 6 x 10
  carat     cut      color clarity depth table price     x     y     z
  <dbl>    <ord>    <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1 0.23 Ideal     E     SI2     61.5    55    326  3.95  3.98  2.43
2 0.21 Premium   E     SI1     59.8    61    326  3.89  3.84  2.31
3 0.23 Good      E     VS1     56.9    65    327  4.05  4.07  2.31
4 0.29 Premium   I     VS2     62.4    58    334  4.2    4.23  2.63
5 0.31 Good      J     SI2     63.3    58    335  4.34  4.35  2.75
6 0.24 Very Good J     VVS2    62.8    57    336  3.94  3.96  2.48
```

```
head(mtcars)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

Exercises will be using the penguins data set from package palmerpenguins. Beware that the same data is now part of package datasets, but with different column names!

## 7.2 Basic structure of a ggplot call

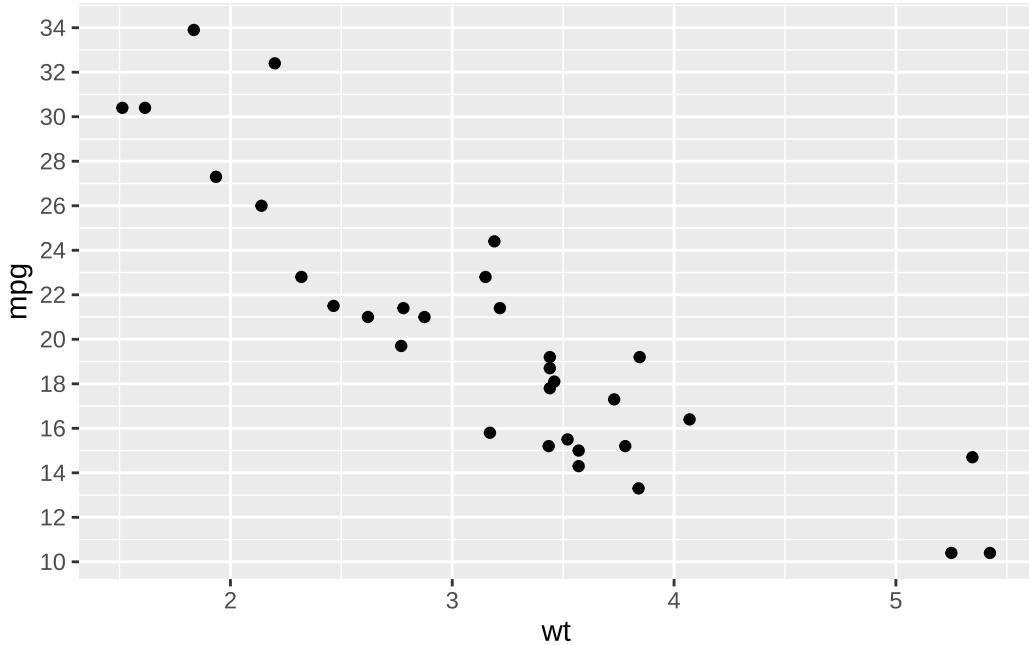
To create a figure, we at least use 2 function calls:

1. `ggplot(data = my_data, mapping = aes(x=..., y=..., color=..., shape=...))` to define data and inside `aes()` some global defaults for aesthetic mappings, this (sort of) creates the canvas to draw on
2. `geom_xxx()` to define the geometry to be used to show the data, e.g. bar, boxplot, point

When mapping data to aesthetics, the class of data matters: Numerical data are interpreted as continuous, so a color heatmap is mapped rather than discrete colors, grouping of data requires factors / characters.

Extensive example:

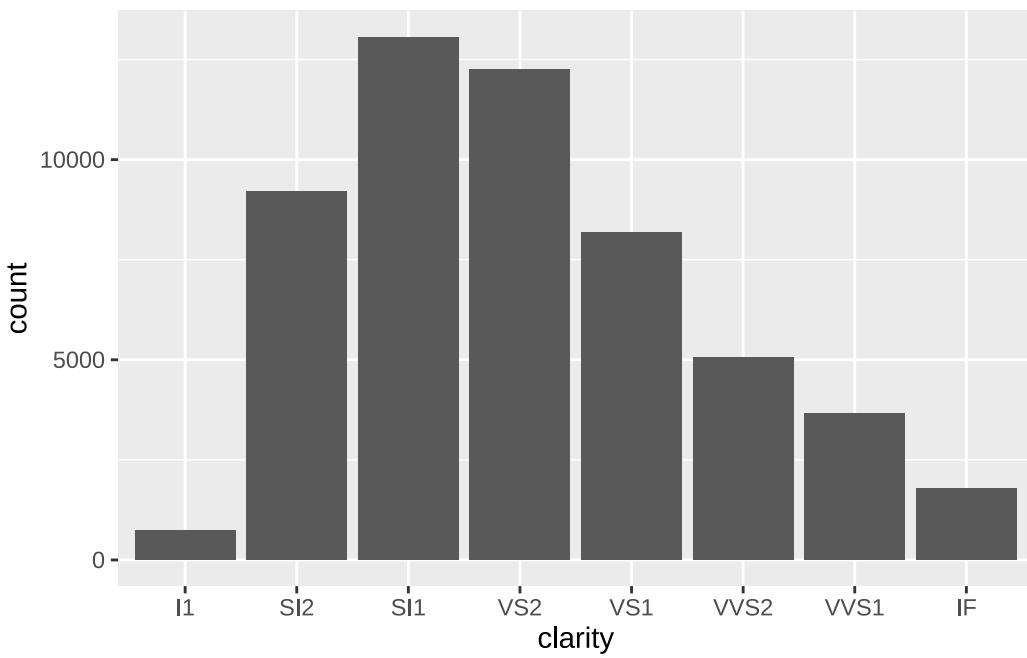
```
ggplot(  
  data = mtcars,  
  aes(x = wt, y = mpg))+  
  geom_point()  
  scale_y_continuous(breaks = seq(0,50,2))+  
  theme(legend.position = "bottom")
```



```
ggplot( #base definition of plot
  data = mtcars, # what data set to use
  aes(x = wt, y = mpg)) + # which variables bound to aesthetics
  geom_point() + # which geometric form to show the data (bar, boxplot ...)
  scale_y_continuous(breaks = seq(0,50,2)) + # tuning of translation of data into aesthetics
  theme(legend.position = "bottom") # all non-data aspects of a plot
```

Minimal example:

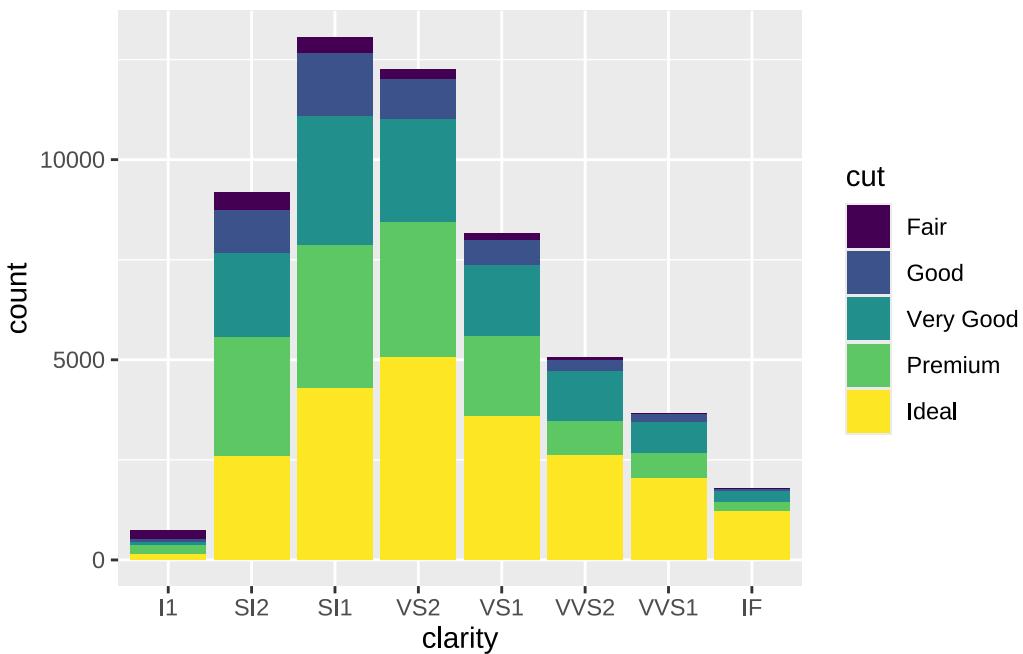
```
ggplot(data=diamonds,mapping = aes(x=clarity))+  
  geom_bar()
```



`geom_bar()` inherits the global aesthetic `x` (*build a x-axis based on values in column clarity*) and does not need a y-axis definition, as it uses some in-build statistics (“count”) and defines `y`.

We can add additional aesthetic parameters like `fill=`. This automatically creates sub-groups for counting:

```
ggplot(data=diamonds,aes(x=clarity,fill=cut))+  
  geom_bar()
```



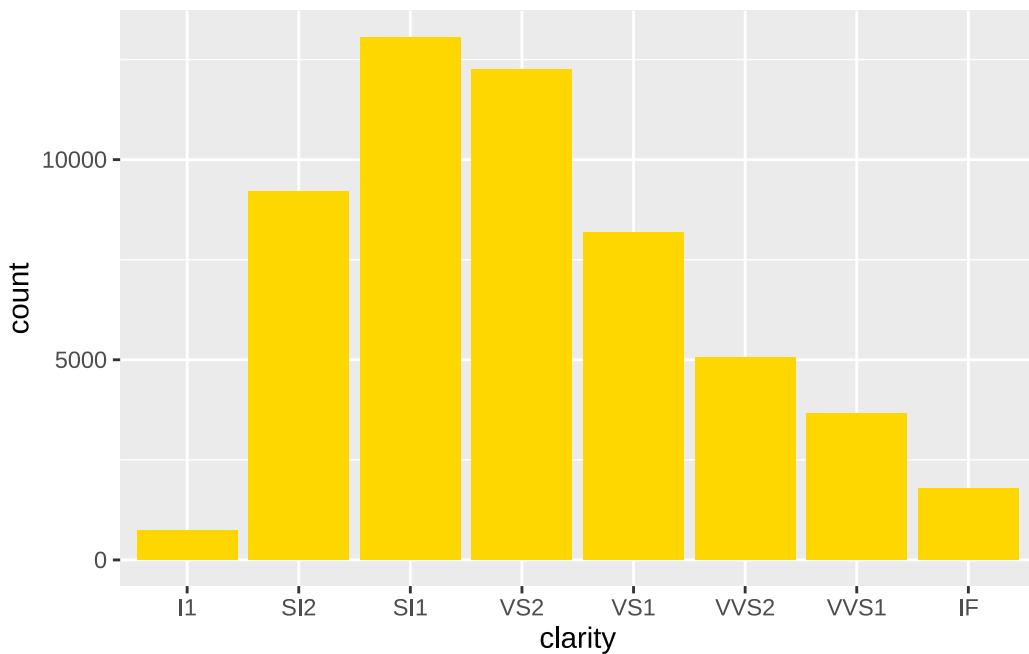
This graph represents this count table:

```
diamonds |>
  group_by(clarity,cut) |>
  count() |>
  pivot_wider(names_from = clarity,values_from=n)
```

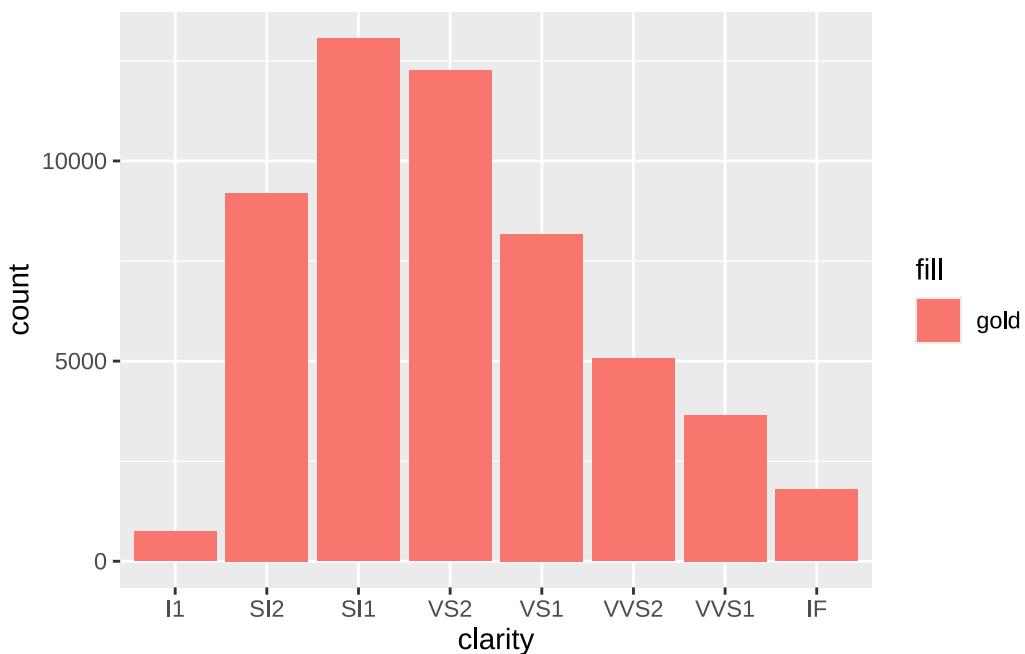
```
# A tibble: 5 x 9
# Groups:   cut [5]
  cut      I1    SI2    SI1    VS2    VS1    VVS2    VVS1    IF
  <ord>  <int> <int> <int> <int> <int> <int> <int>
1 Fair     210    466   408   261   170    69     17     9
2 Good     96    1081   1560   978   648   286    186    71
3 Very Good  84    2100   3240  2591  1775  1235   789   268
4 Premium   205   2949   3575  3357  1989   870   616   230
5 Ideal     146   2598   4282  5071  3589  2606  2047  1212
```

Aesthetic parameters can represent data / have some meaning (as cut quality), but they can be defined to reflect your taste rather than data. In that case, you define them outside of `aes()`. Careful, as this may lead to confusion:

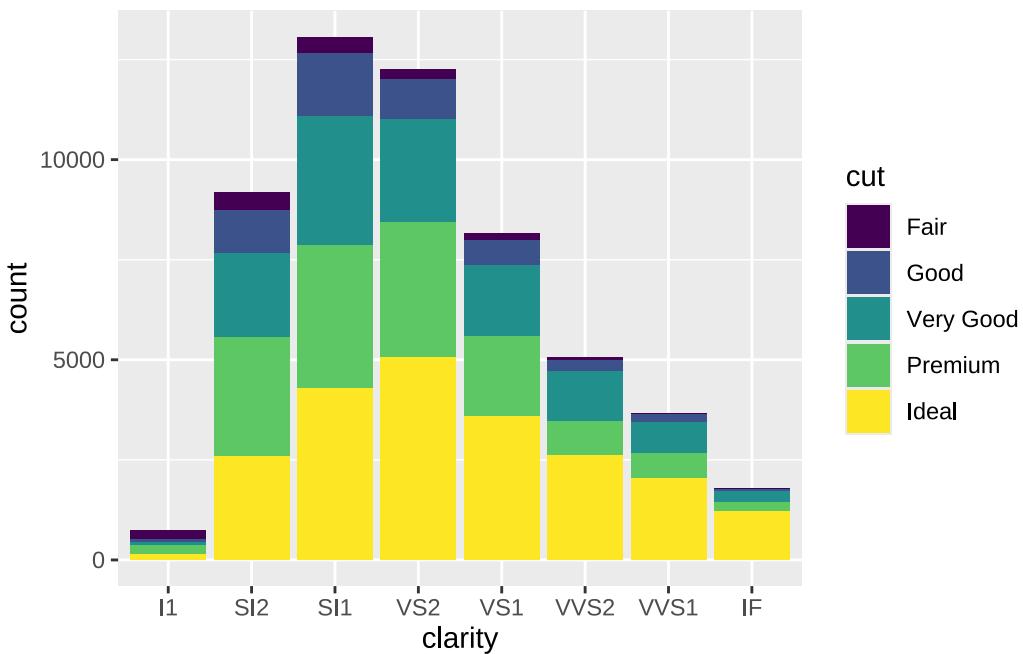
```
#aesthetics outside aes
ggplot(data=diamonds,aes(x=clarity))+  
  geom_bar(fill="gold")
```



```
ggplot(data=diamonds,aes(x=clarity))+
  geom_bar(aes(fill="gold")) #should be outside aes!
```



```
ggplot(data=diamonds,aes(x=clarity))+
  geom_bar(aes(fill=cut)) # may be defined locally as well globally
```



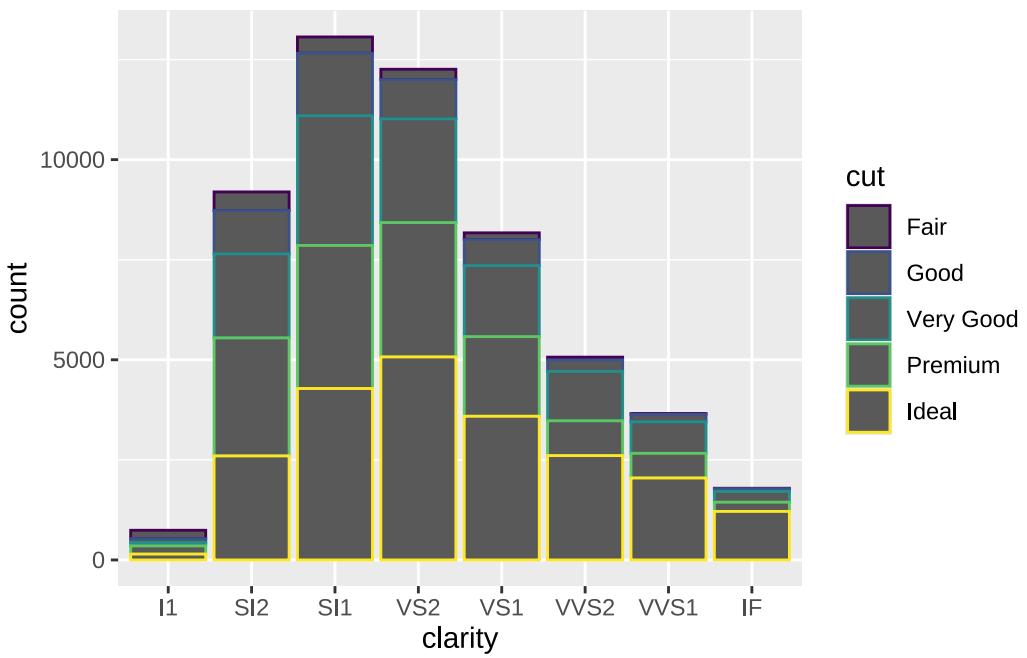
### 7.3 fill vs. color

Some elements (as e.g. the bar or boxplot) know 2 color elements:

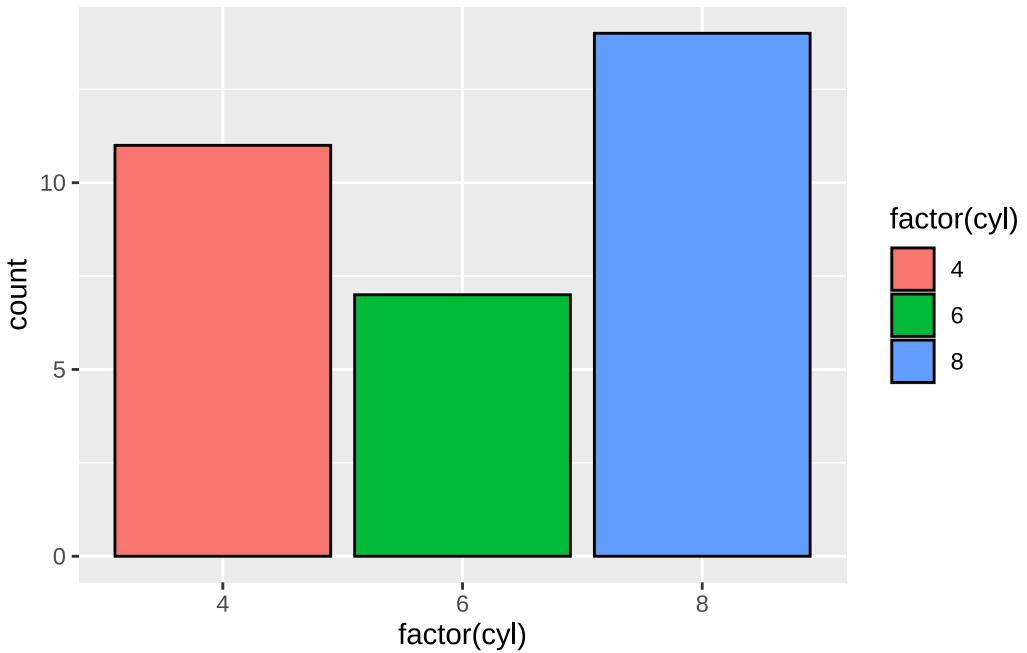
- inner color, defined by `fill`
- outer frame color, defined by `color`

Other elements (as e.g. the line) only have a single color definition, specified by `color`. And for some elements (as e.g. dots), it depends. See help for points for examples.

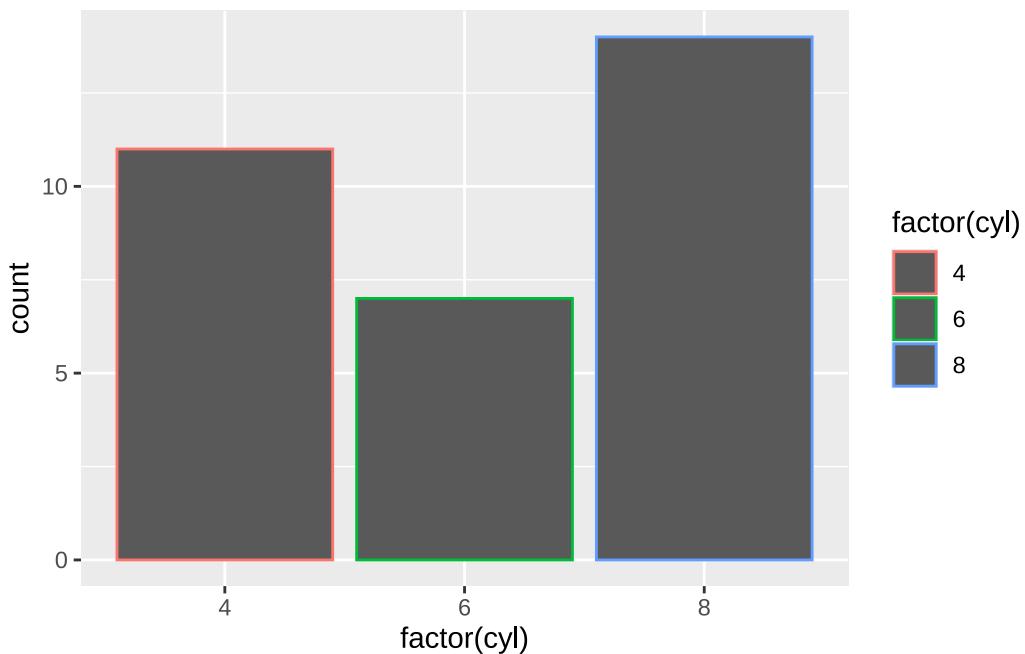
```
ggplot(data=diamonds,aes(x=clarity,color=cut))+  
  geom_bar()
```



```
ggplot(data=mtcars,aes(factor(cyl),fill=factor(cyl)))+
  geom_bar(color="black")
```



```
ggplot(data=mtcars,aes(factor(cyl),color=factor(cyl)))+
  geom_bar()
```



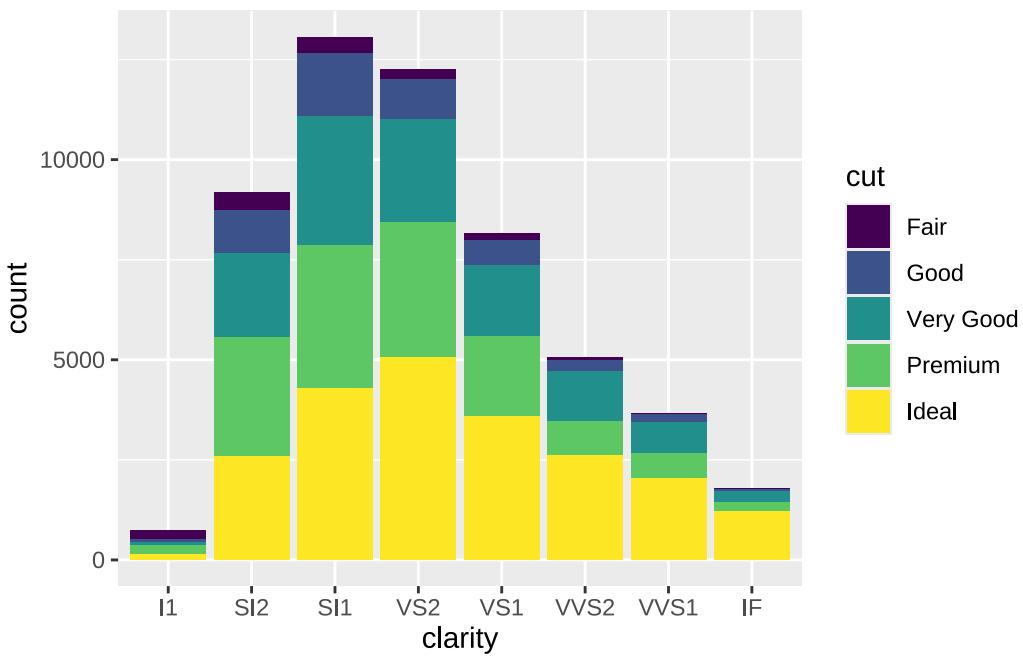
## 7.4 Color systems

ggplot2 comes with various color definitions, many external packages extend that. Manual definition of colors is possible as well. Redefining the mapping between data and aesthetics can be done with `scale_...` functions.

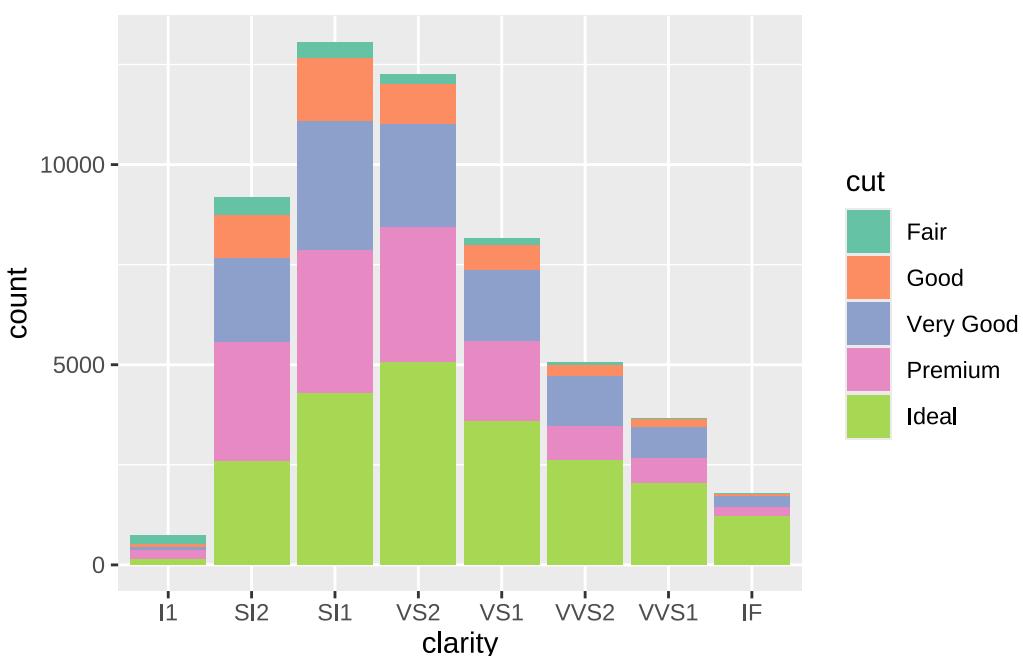
[comprehensive overview for color palettes](#)

For the demonstration, I store a plot into a variable, this includes all data and plot definitions, nothing like jpg!

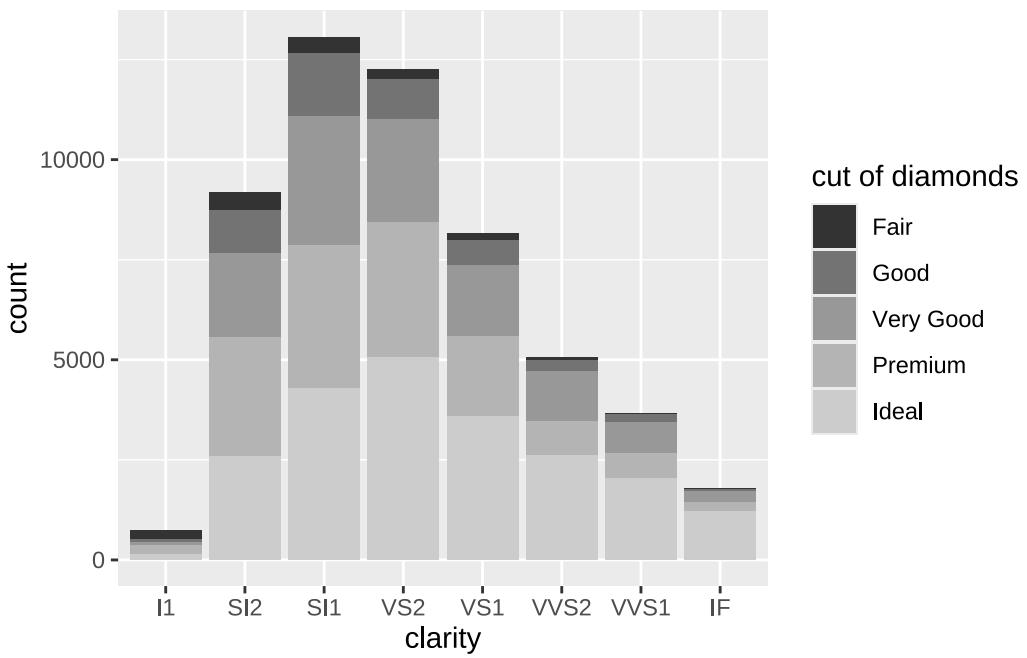
```
(plottemp <- ggplot(data=diamonds, aes(x=clarity, fill=cut))+
  geom_bar())
```



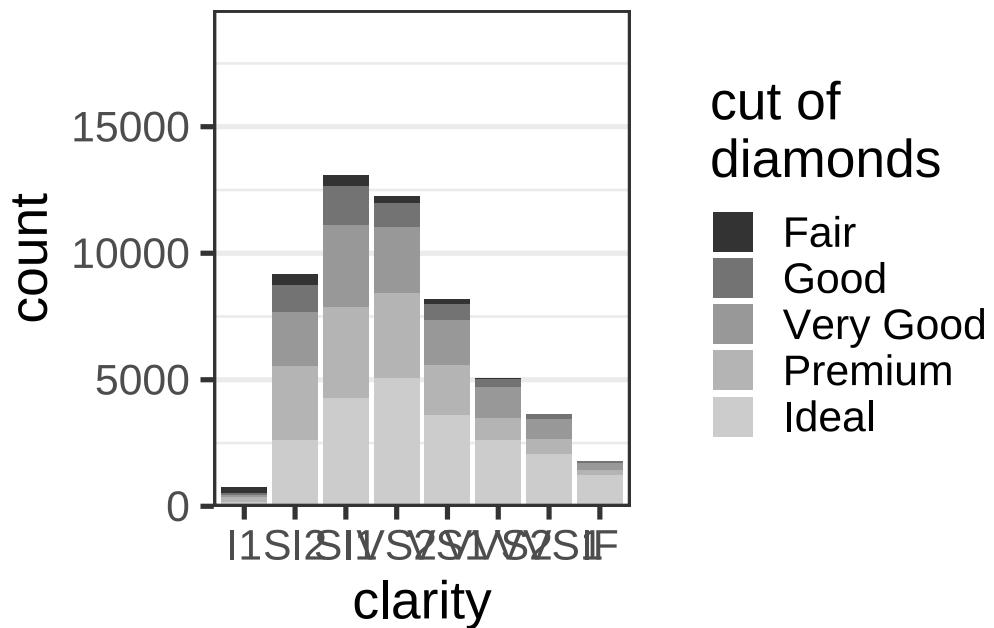
```
plottemp + scale_fill_brewer(palette="Set2") #in-built "scale" for fill
```



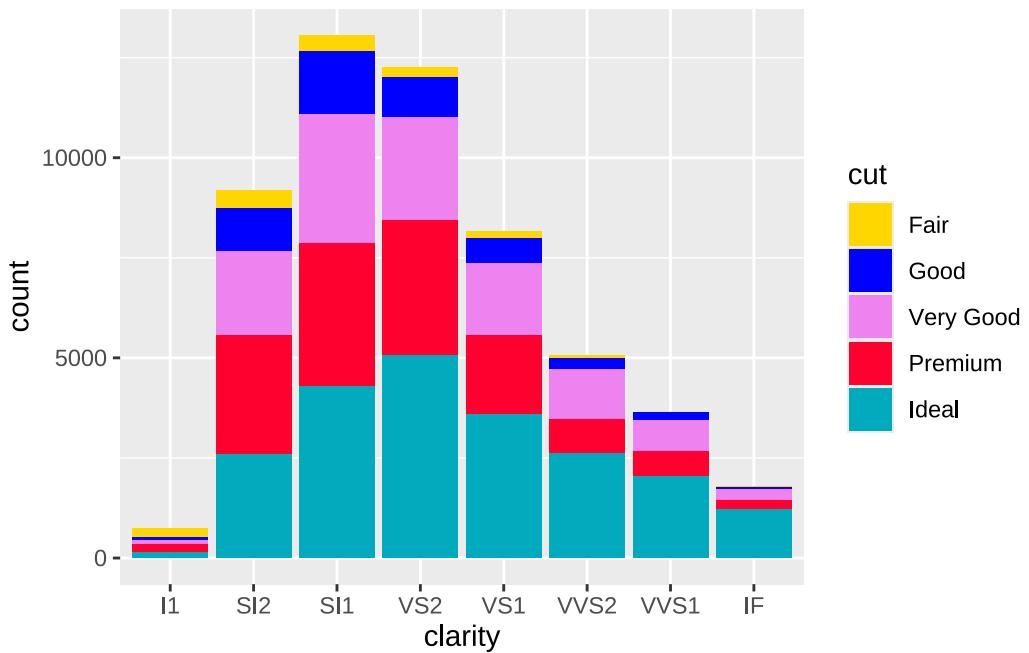
```
plottemp + scale_fill_grey(name = "cut of diamonds")
```



```
plottemp + scale_fill_grey(name = "cut of diamonds") +
  scale_y_continuous(expand = expansion(mult = c(0,.5)))+ # rescaling y
  theme_bw(base_size = 20) +
  theme(panel.grid.major.x = element_blank())
```

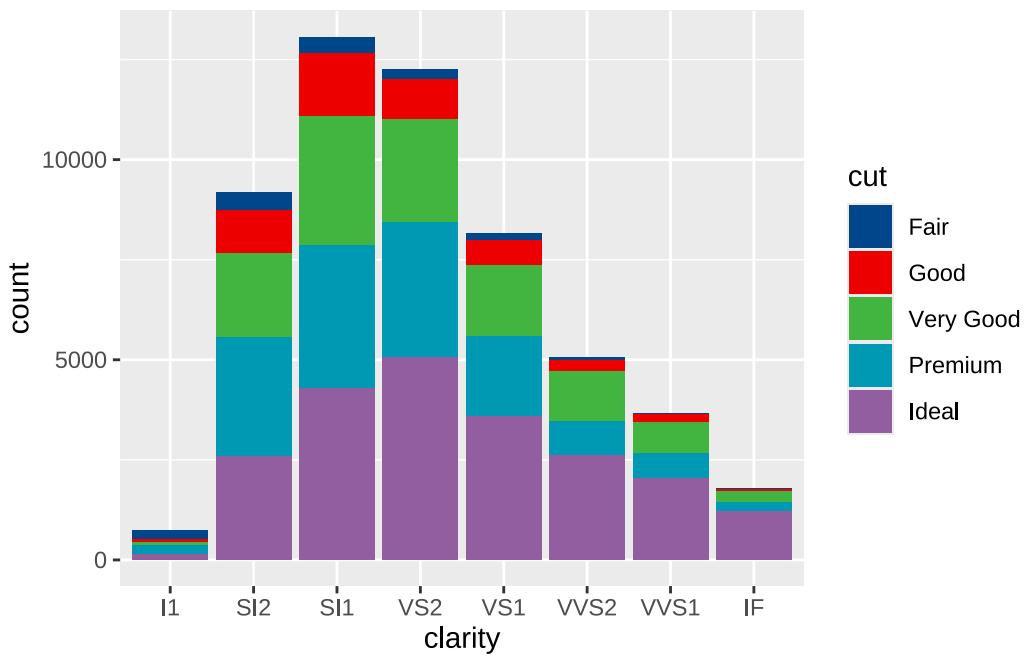


```
my_colors <- c("gold",'blue','violet',"#FF012F","#01AABC")
plottemp+ scale_fill_manual(values=my_colors)
```

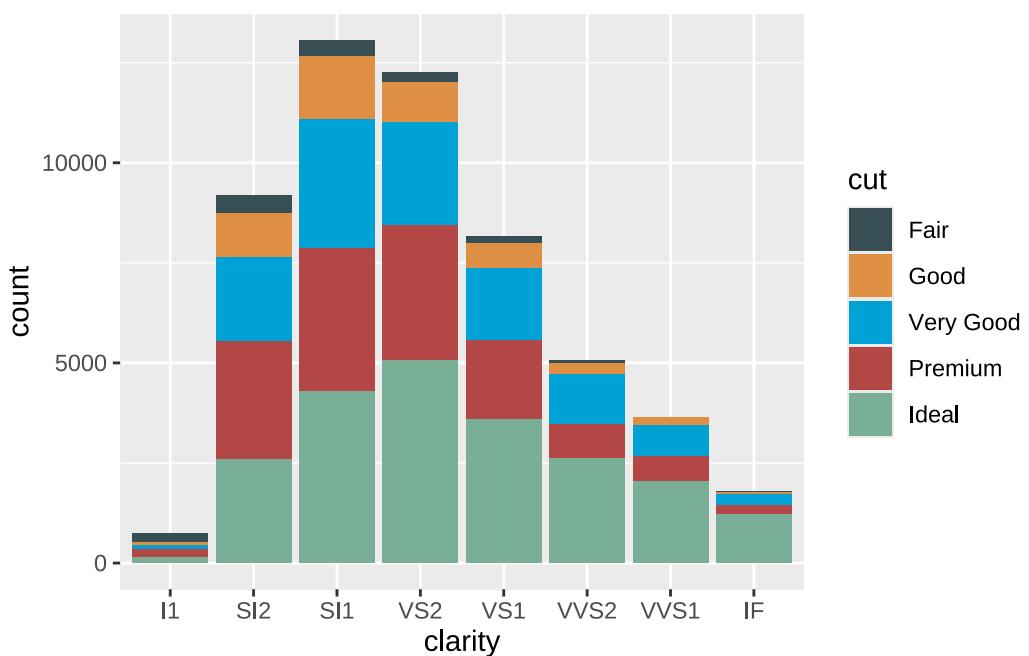


#### 7.4.1 External color definitions from ggsci

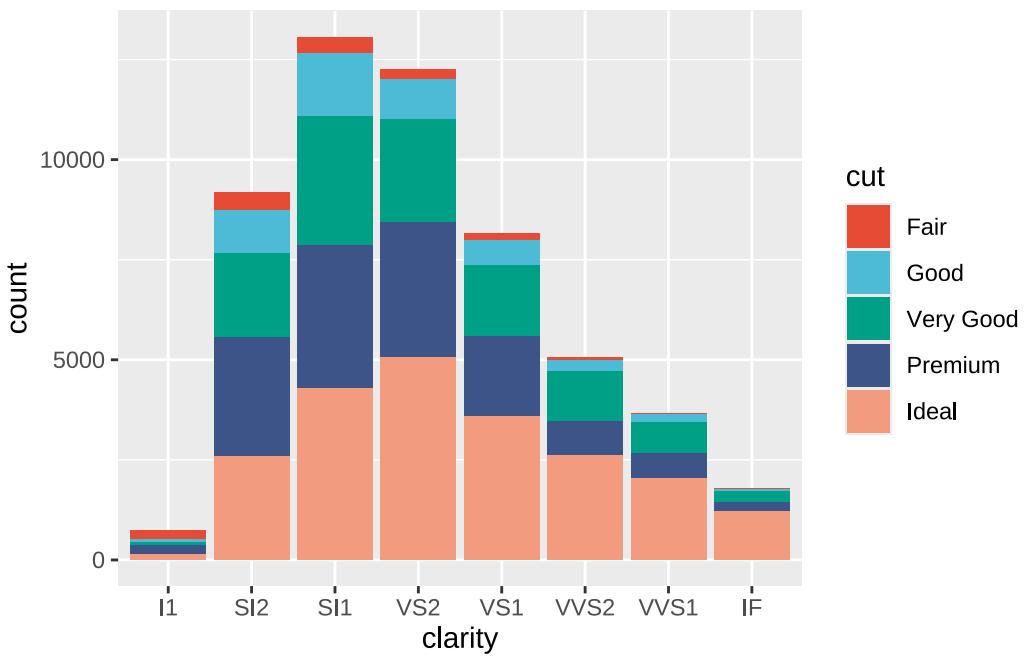
```
plottemp+scale_fill_lancet()
```



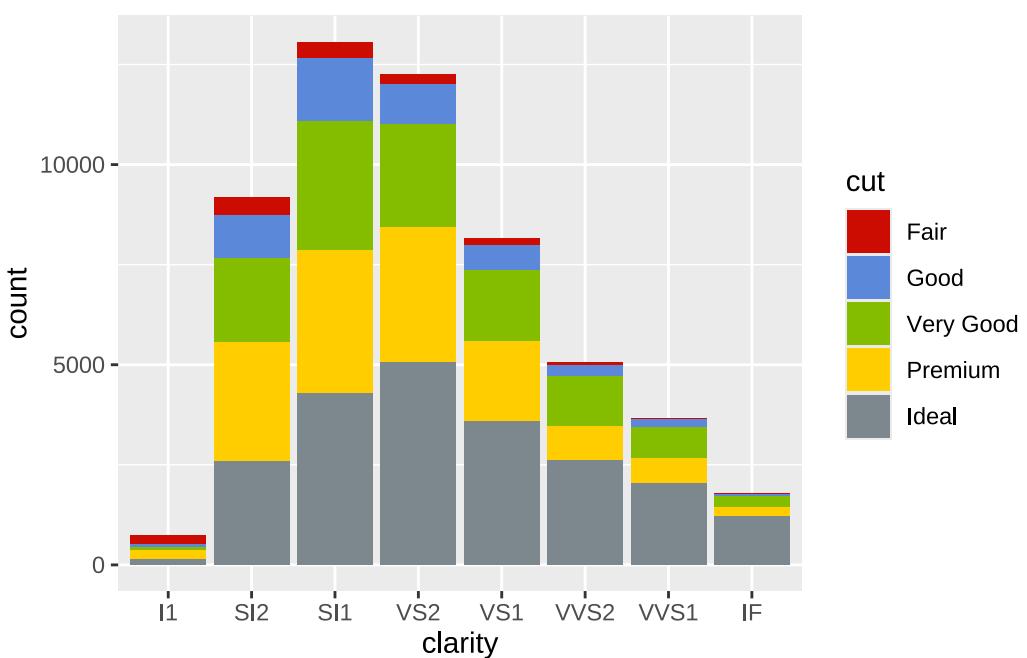
```
plottemp+scale_fill_jama()
```



```
plottemp+scale_fill_npg()
```



```
(printplot <- plottemp+scale_fill_startrek())
```



## 7.5 Exporting ggplots

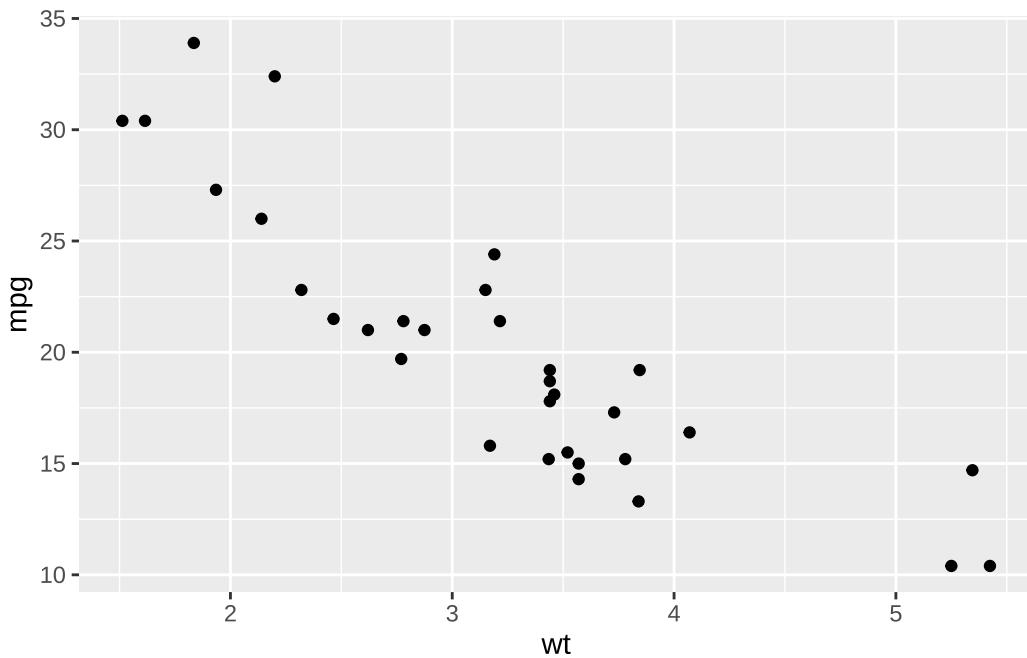
There are two distinct ways, using `ggsave()` or more generally creating an external graphic device with e.g. `png()` / `tiff()` / `pdf()`:

```
ggsave(filename = here("Graphs/ggtestplot.png"),
       plot = printplot,
       width=20,height=20,
       units="cm",dpi=150)
ggsave(filename = here("Graphs/ggtestplot.tiff"),
       plot = printplot,
       width=20,height=20,
       units="cm",dpi=600)
ggsave(filename = here("Graphs/ggtestplot_c.tiff"),
       plot = printplot,
       width=20,height=20, compression="lzw",
       units="cm",dpi=600)
# alternative:
png(filename = here("Graphs/ggtestplot2.png"),
     width = 20,height = 20,units = "cm",res = 150)
plottemp
dev.off()
```

## 7.6 Other geoms

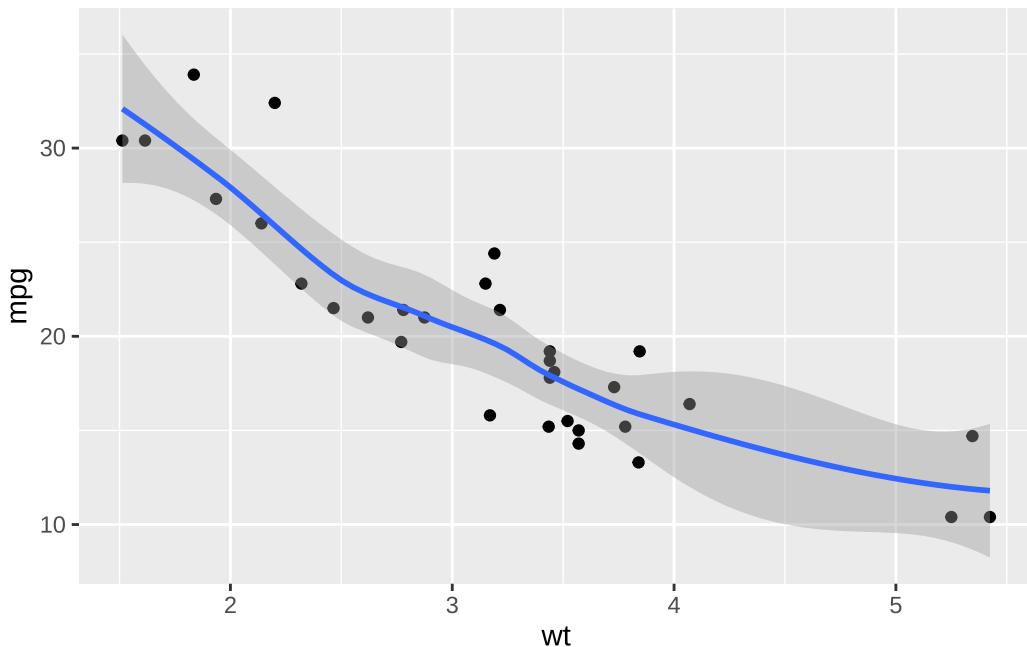
Common forms of plots are barplots, boxplots, scatterplots (possibly with regression line), and density-plots. For plotting dots for groups, there are various options to avoid over-plotting of repeated data, with the `beeswarm` as my preference.

```
ggplot(data=mtcars,aes(x = wt,y = mpg))+  
  geom_point()
```



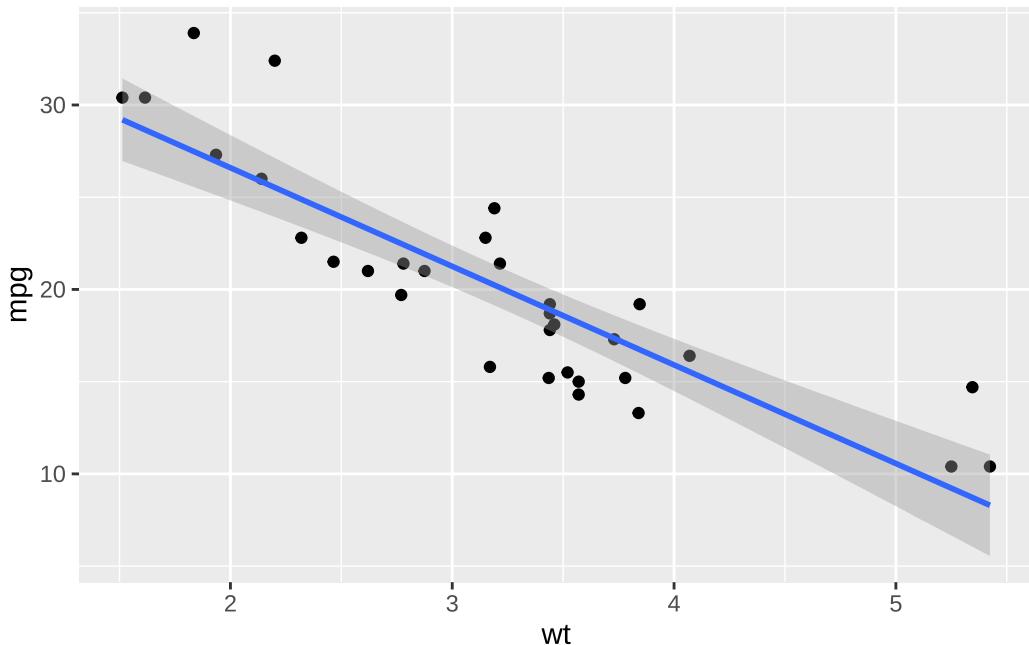
```
ggplot(data=mtcars,aes(x = wt,y = mpg))+  
  geom_point() +  
  geom_smooth()
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

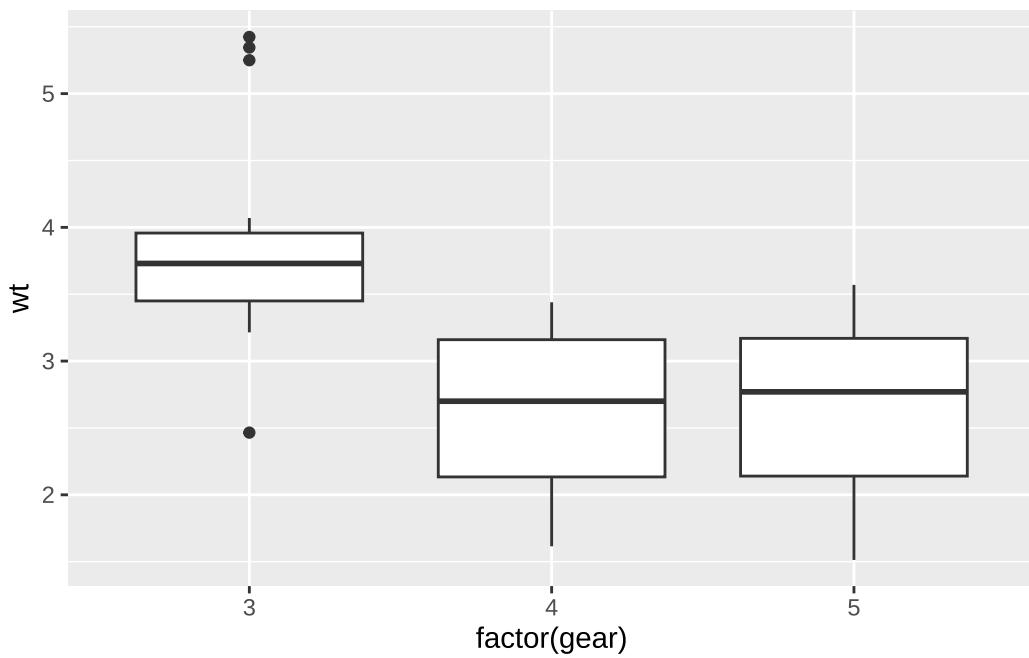


```
ggplot(data=mtcars,aes(x = wt,y = mpg))+  
  geom_point() +  
  geom_smooth(method="lm")
```

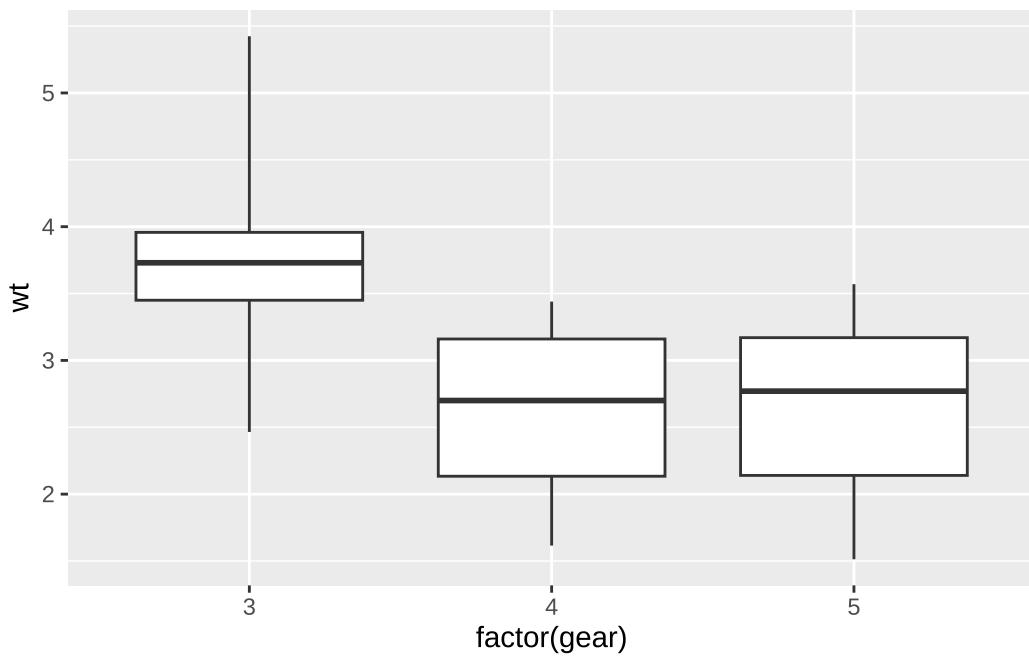
```
`geom_smooth()` using formula = 'y ~ x'
```



```
ggplot(mtcars,aes(x = factor(gear),y = wt))+  
  geom_boxplot() #default 1.5 IQR
```

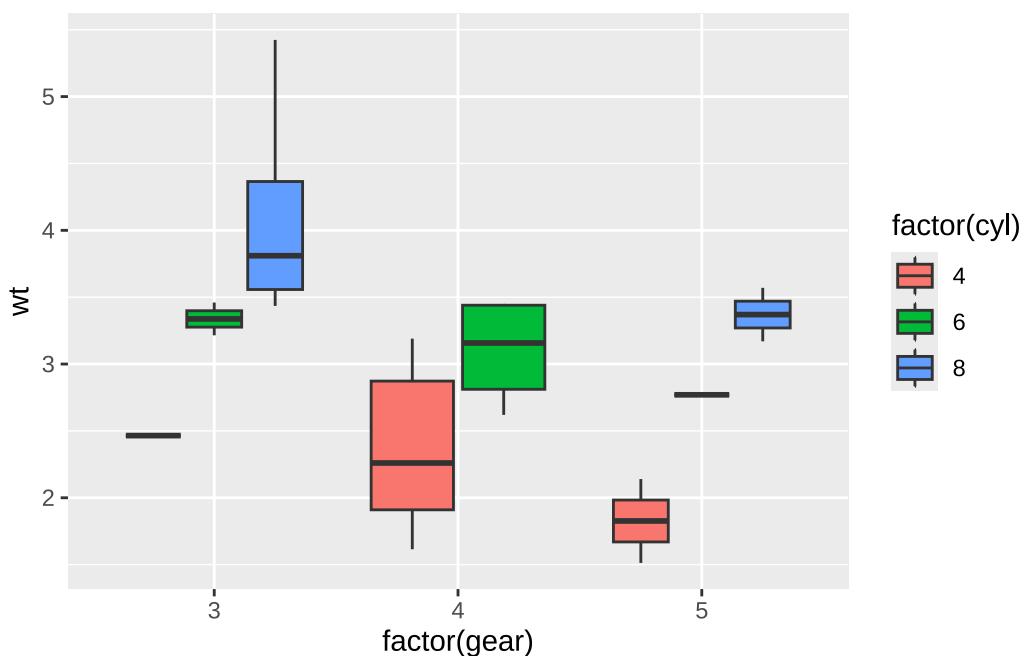


```
ggplot(mtcars,aes(x = factor(gear),y = wt))+  
  geom_boxplot(coef=3) # this extends range of expected values
```

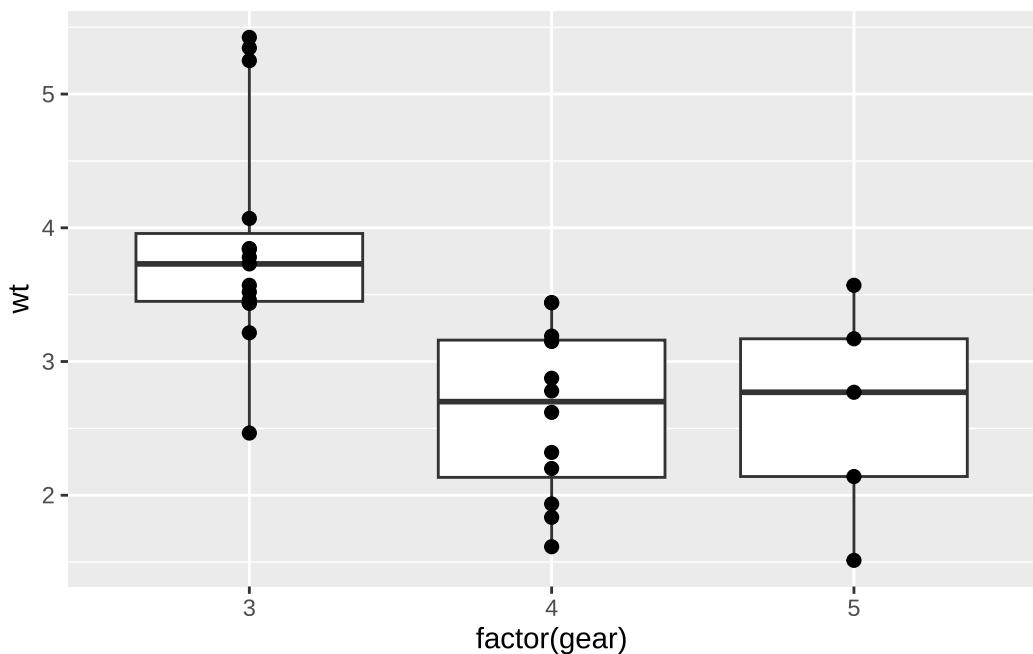


```
ggplot(mtcars,aes(x = factor(gear),y = wt,  
  fill=factor(cyl)))+
```

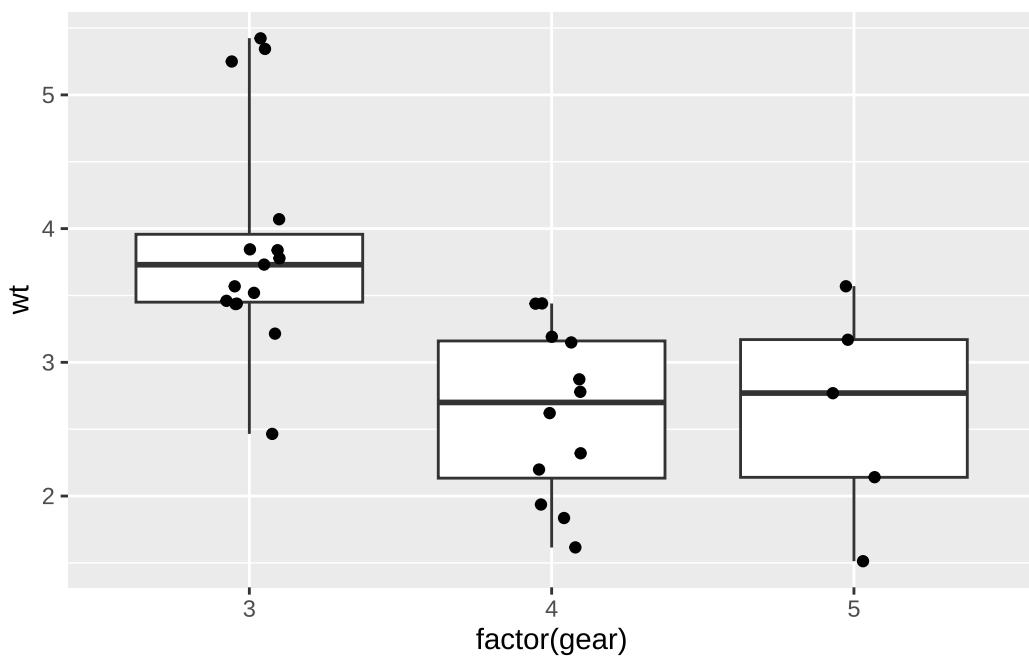
```
geom_boxplot(coef=3) # group by cyl, as it is mapped to fill
```



```
ggplot(mtcars,aes(x = factor(gear),y = wt))+  
  geom_boxplot(coef=3)+  
  geom_point(size=2) # may contain overlapping points
```

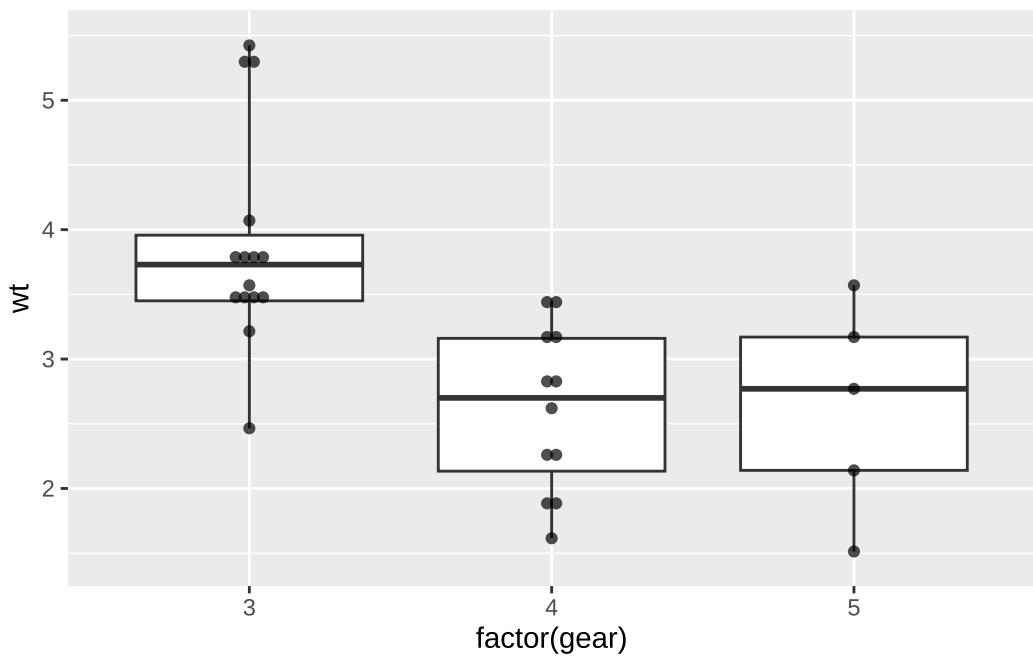


```
ggplot(mtcars,aes(x = factor(gear),y = wt))+  
  geom_boxplot(coef=3)+  
  geom_point(position = position_jitter(width = .1))
```

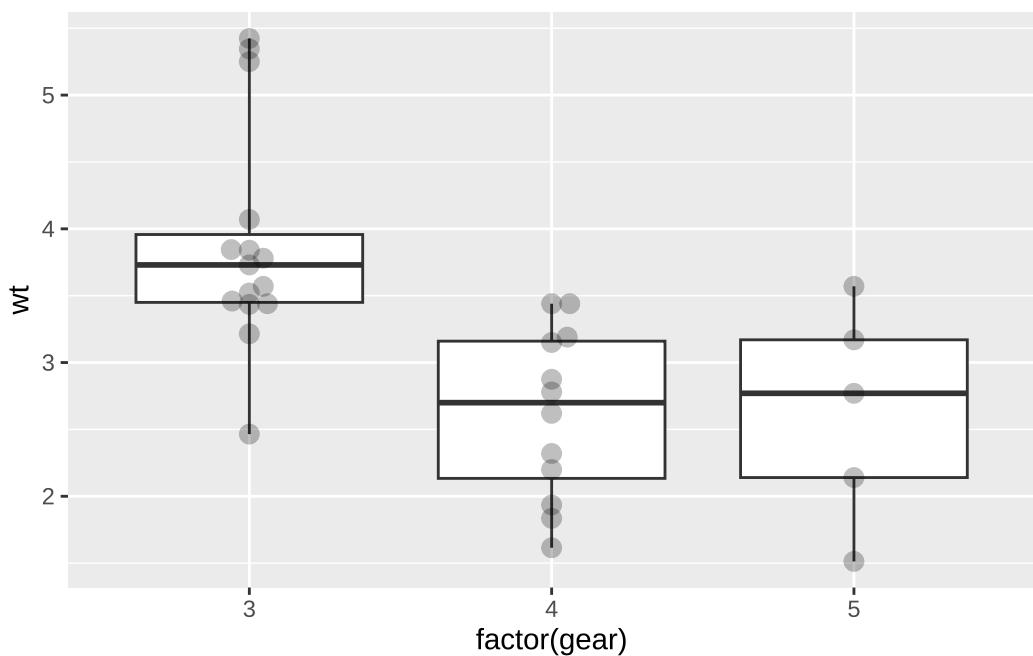


```
ggplot(mtcars,aes(x = factor(gear),y = wt))+  
  geom_boxplot(coef=3)+  
  geom_dotplot(alpha=.7, # group similar(ish) data on a line  
               binaxis = "y",stackdir = "center",  
               stackratio = .9,dotsize = .6)
```

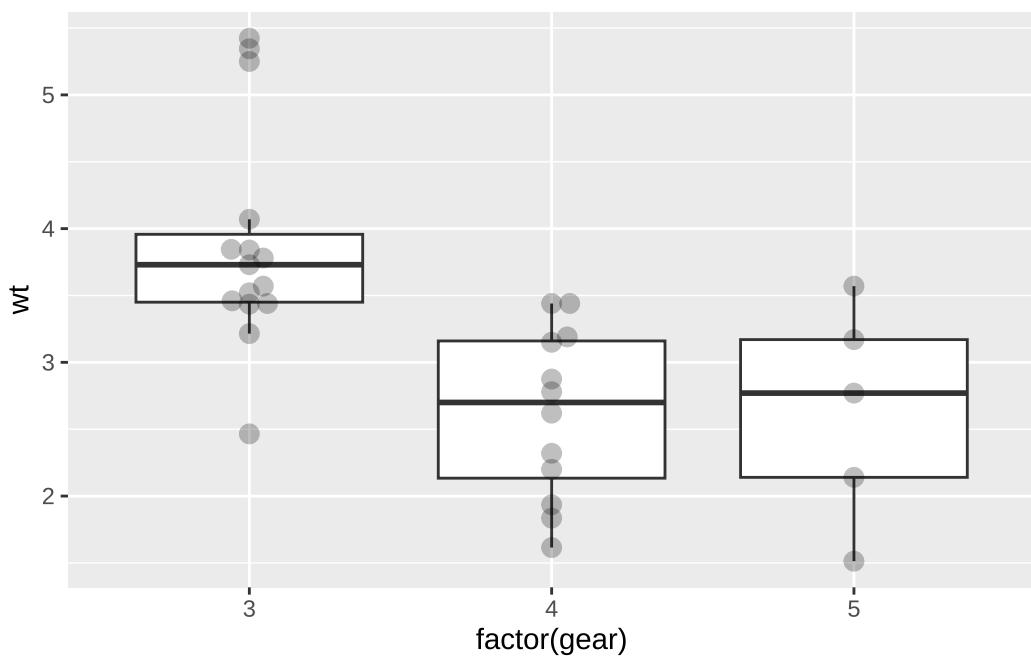
Bin width defaults to 1/30 of the range of the data. Pick better value with `binwidth`.



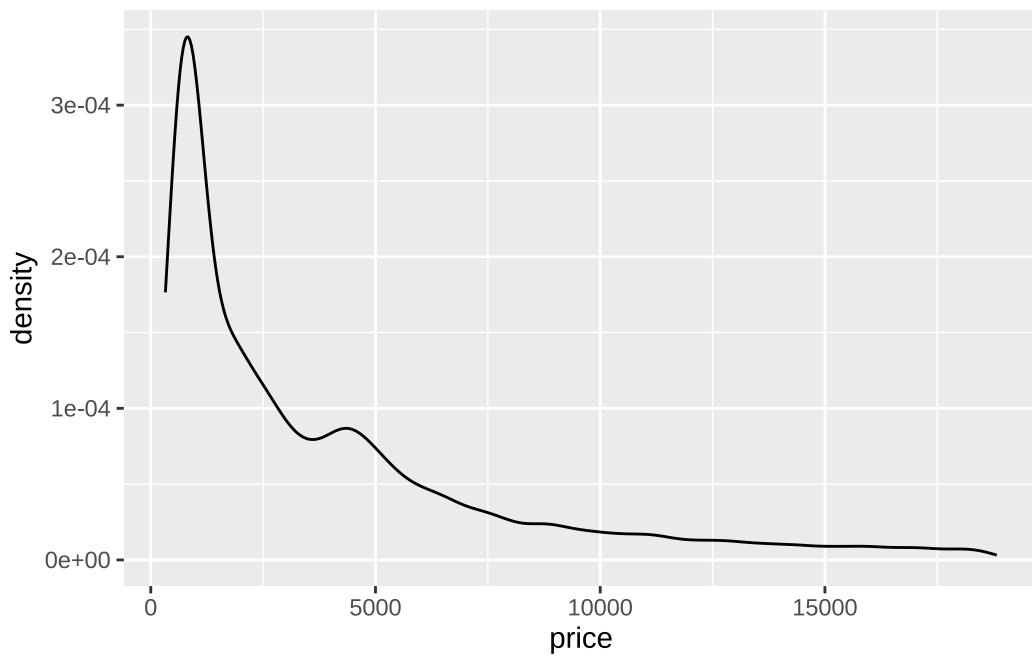
```
ggplot(mtcars,aes(x = factor(gear),y = wt))+  
  geom_boxplot(coef=3)+  
  ggbeeswarm::geom_beeswarm(cex = 2,size=3,alpha=.25)
```



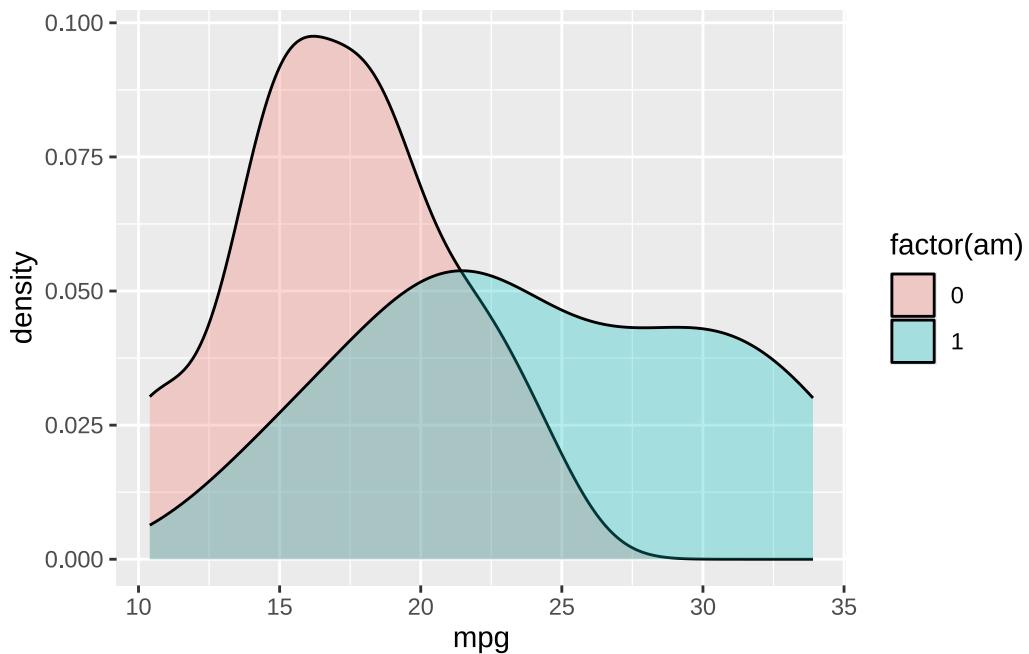
```
ggplot(mtcars,aes(x = factor(gear),y = wt))+  
  geom_boxplot(outlier.alpha = 0)+ # to avoid plotting outliers twice  
  geom_beeswarm(cex = 2,size=3,alpha=.25)
```



```
#density plot  
ggplot(diamonds,aes(price))+  
  geom_density()
```

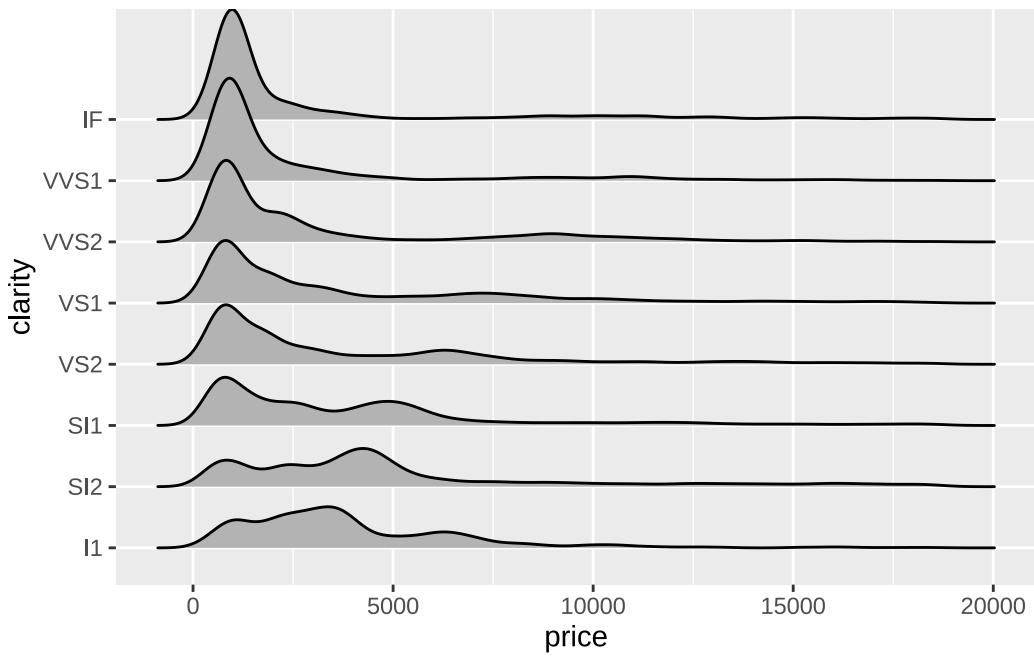


```
ggplot(mtcars,aes(mpg, fill=factor(am)))+
  geom_density(alpha=.3)
```

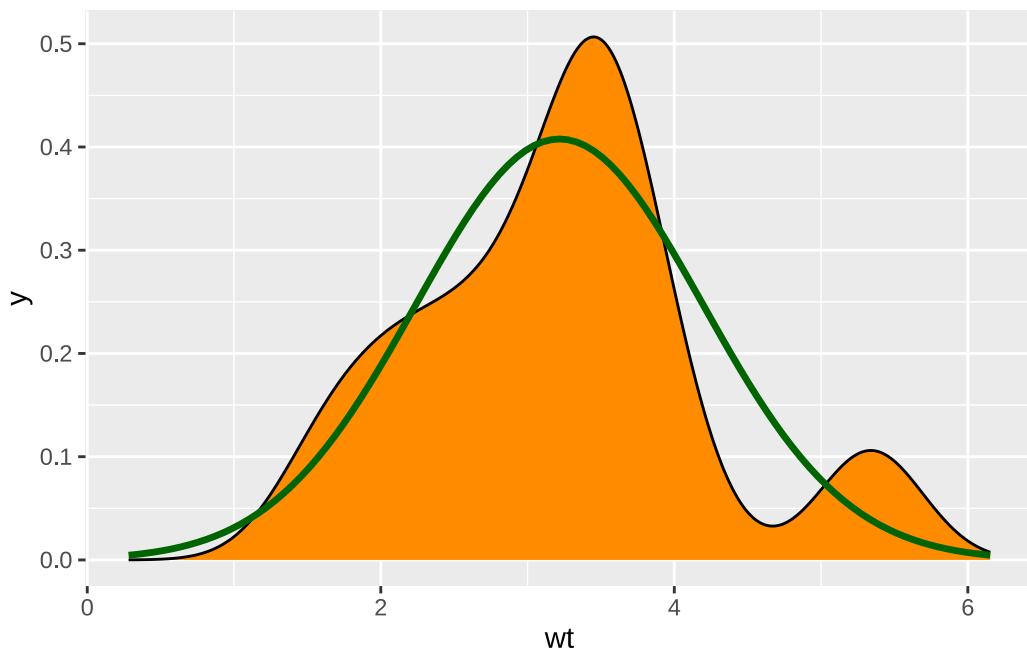


```
ggplot(diamonds,aes(price,y=clarity))+
  geom_density_ridges()
```

Picking joint bandwidth of 403



```
#empirical vs. theoretical distribution
ggplot(mtcars, aes(wt))+
  geom_density(fill = "darkorange")+
  stat_function(fun = dnorm,
    args = list(mean = mean(mtcars$wt),
               sd = sd(mtcars$wt)),
    color = "darkgreen",
    linewidth = 1.2)+
  scale_x_continuous(
    limits = c(mean(mtcars$wt)-3*sd(mtcars$wt),
              mean(mtcars$wt)+3*sd(mtcars$wt)))
```



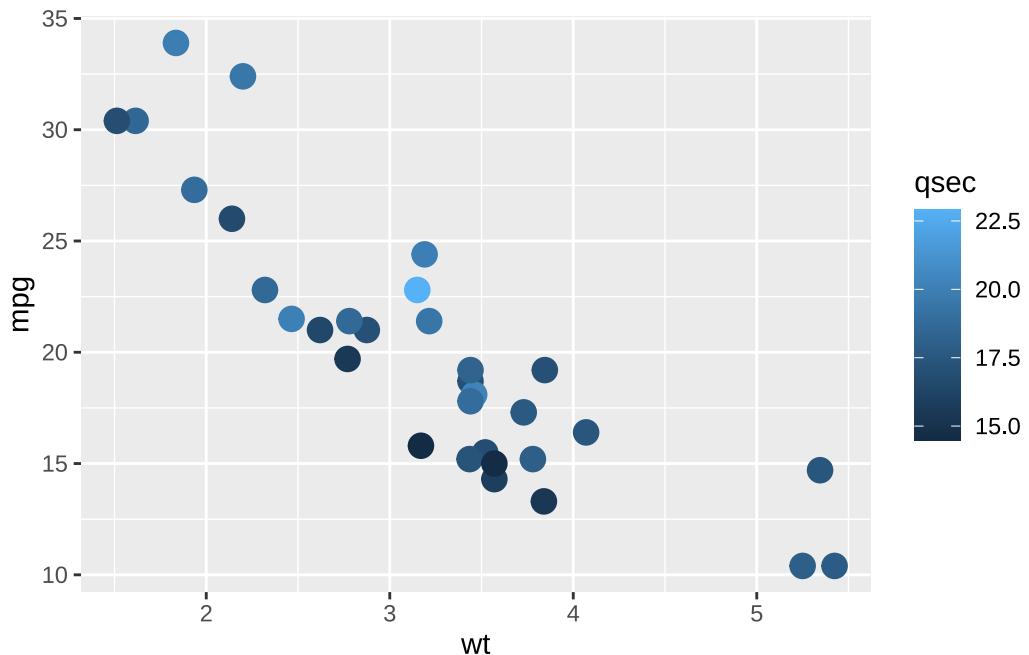
## 7.7 Exercise 1

Vizualize the following:

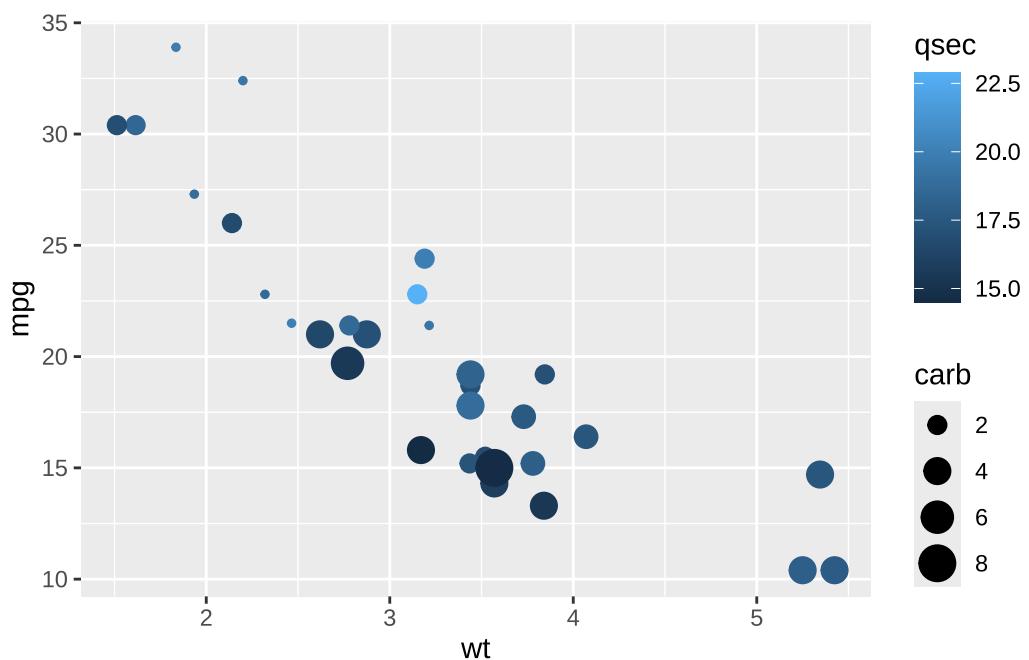
- count of sex within species
- boxplot+beeswarm for weight and species
- boxplot+beeswarm weight for species AND sex
- scatterplot for flipper length vs. body mass
- add a regression line to that plot
- do the same scatterplot and regression grouped by species and sex
- optionally, define your own colors for species (scale or name or rgb)

## 7.8 Combining and finetuning aesthetics

```
ggplot(data=mtcars,aes(wt, mpg,color=qsec))+  
  geom_point(size=4) #outside aes!
```

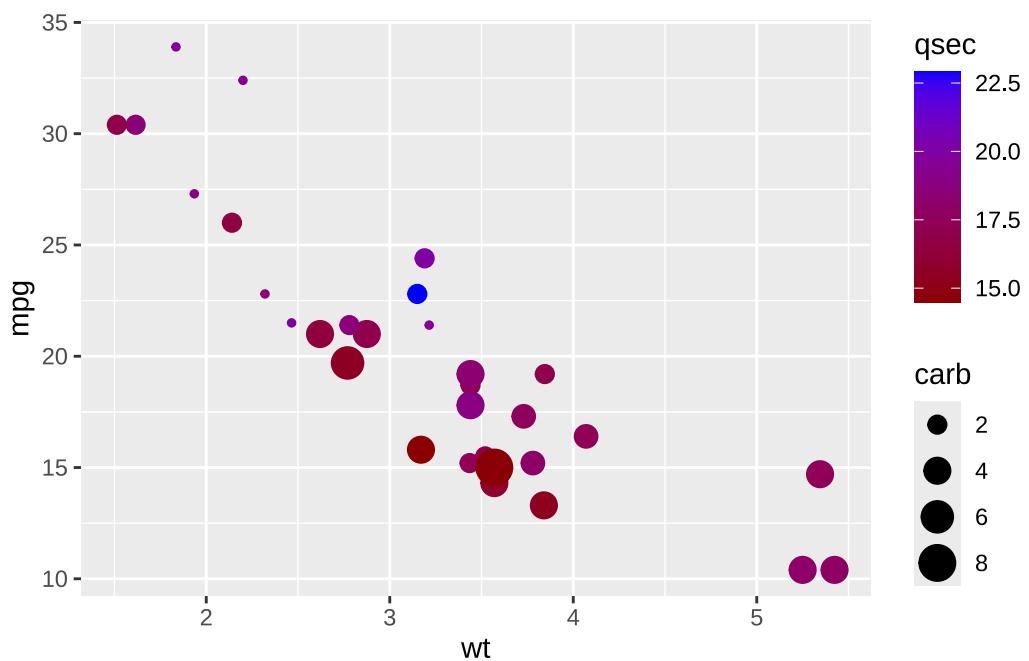


```
ggplot(data=mtcars,aes(wt, mpg,color=qsec, size=carb))+  
  geom_point()
```

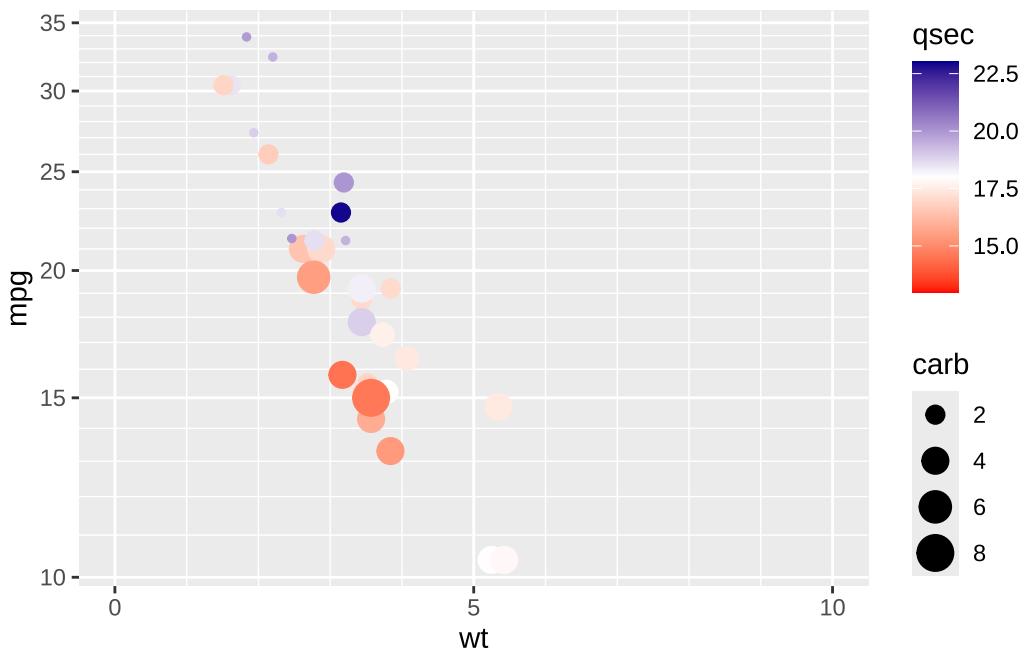


```
ggplot(data=mtcars,aes(wt, mpg,color=qsec, size=carb))+  
  scale_color_gradient(low="darkred",high="blue")+
```

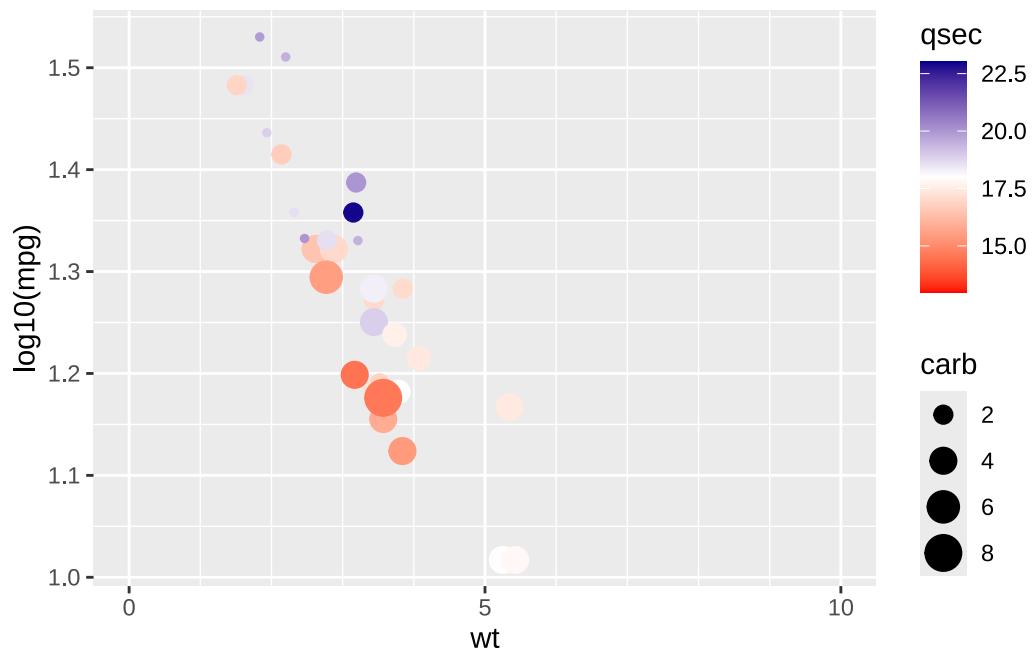
```
geom_point()
```



```
ggplot(data=mtcars,aes(wt, mpg,color=qsec, size=carb))+  
  scale_color_gradient2(low="red",high="darkblue",  
                        mid="white",  
                        limits=c(13,23),midpoint=18)+  
  geom_point()+  
  scale_x_continuous(breaks = seq(-10^3,10^3,5),  
                     minor_breaks=seq(-10^3,10^3,1),  
                     limits = c(0,10))+  
  scale_y_log10(  
    breaks=seq(0,100,5),  
    minor_breaks=seq(0,100,1))
```

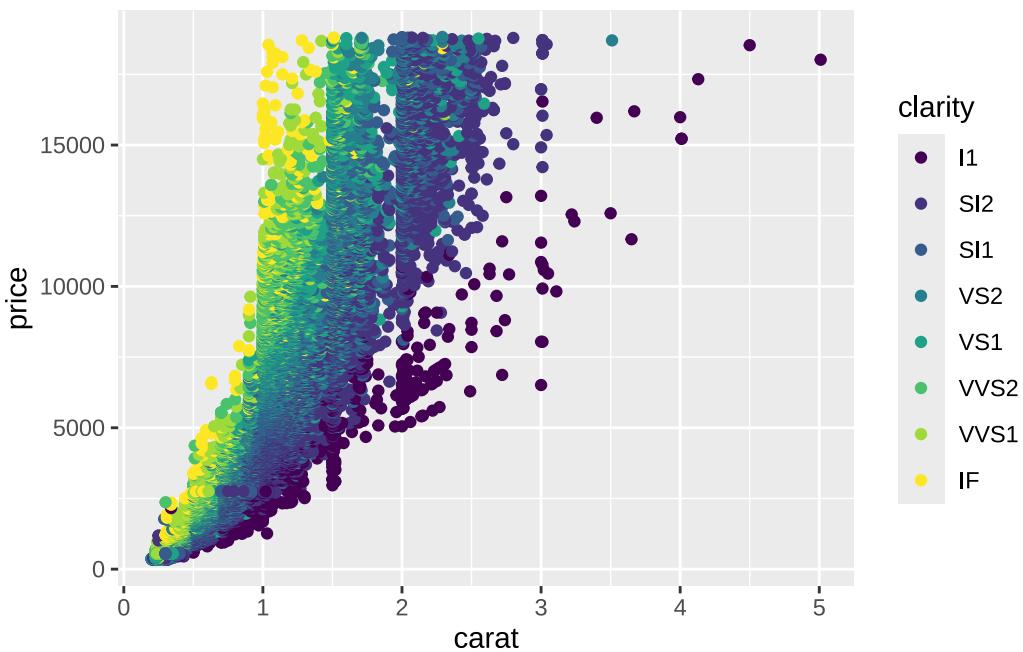


```
ggplot(data=mtcars,aes(wt, log10(mpg),color=qsec, size=carb))+  
  scale_color_gradient2(low="red",high="darkblue",  
                        mid="white",  
                        limits=c(13,23),midpoint=18)+  
  geom_point() +  
  scale_x_continuous(breaks = seq(0,100,5),  
                     minor_breaks=seq(0,100,1),  
                     limits = c(0,10))#+
```

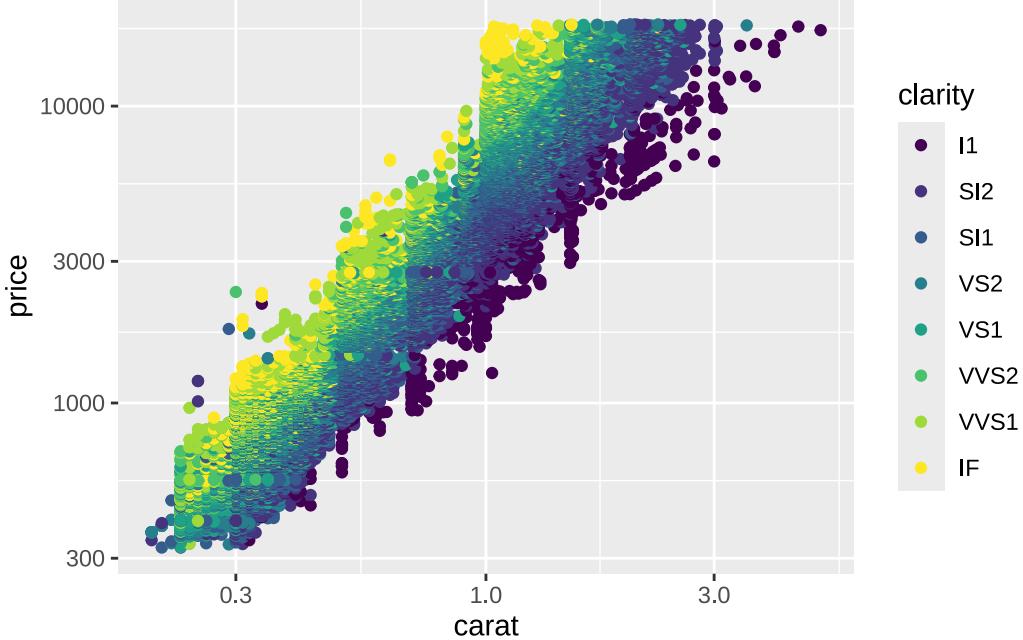


```
# scale_y_log10(minor_breaks=seq(0,100,1))

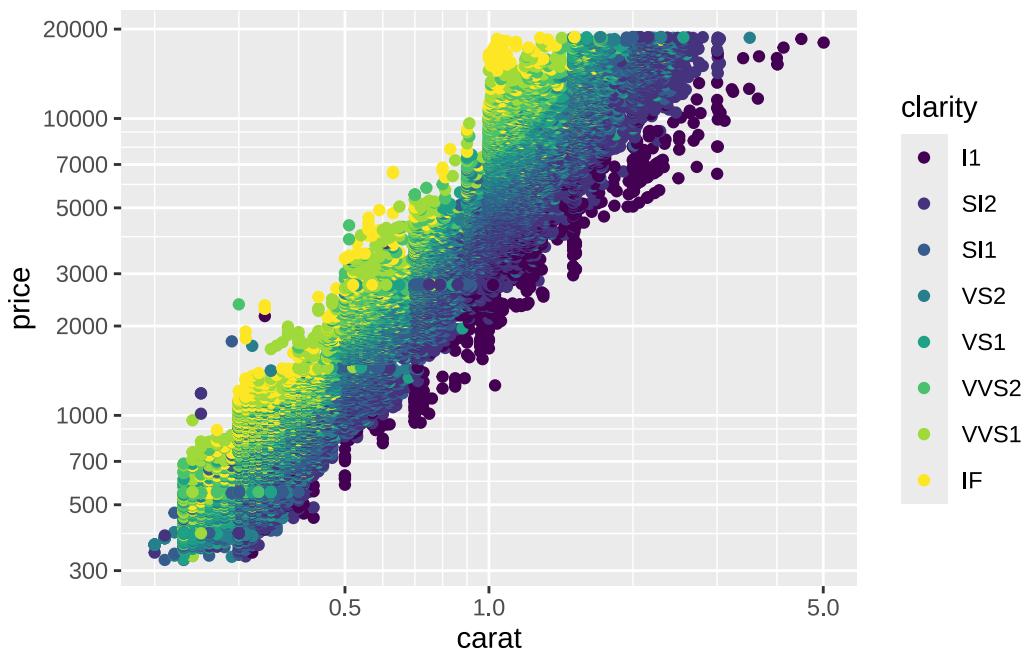
ggplot(diamonds,aes(carat,price,color=clarity))+  
  geom_point()
```



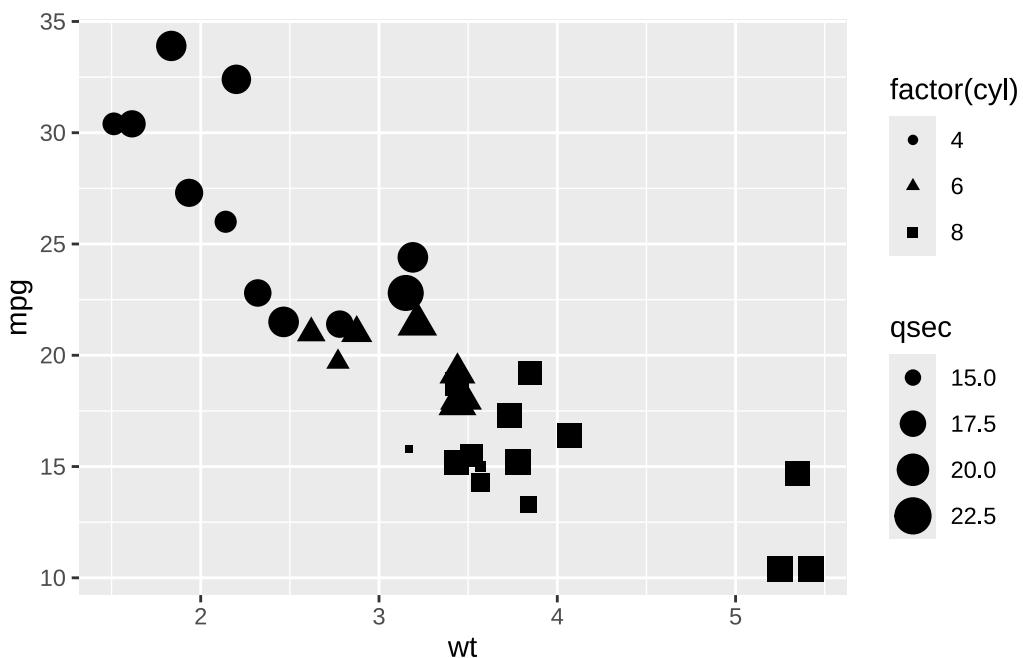
```
ggplot(diamonds,aes(carat,price,color=clarity))+  
  geom_point() +  
  scale_x_log10() +  
  scale_y_log10()
```



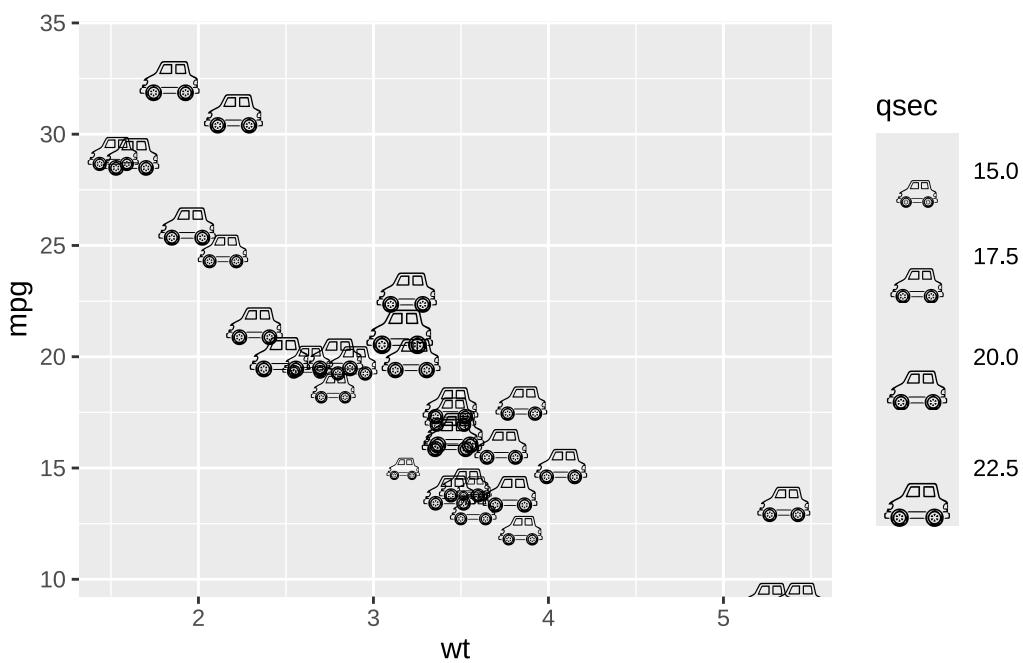
```
ggplot(diamonds,aes(carat,price,color=clarity))+  
  geom_point() +  
  scale_x_log10(  
    breaks=logrange_15,  
    minor_breaks=logrange_123456789) +  
  scale_y_log10(  
    breaks=logrange_12357,  
    minor_breaks=logrange_123456789)
```



```
# use different aesthetic mappings  
ggplot(data=mtcars,  
        aes(wt, mpg, size=qsec, shape=factor(cyl)))+  
  geom_point()
```



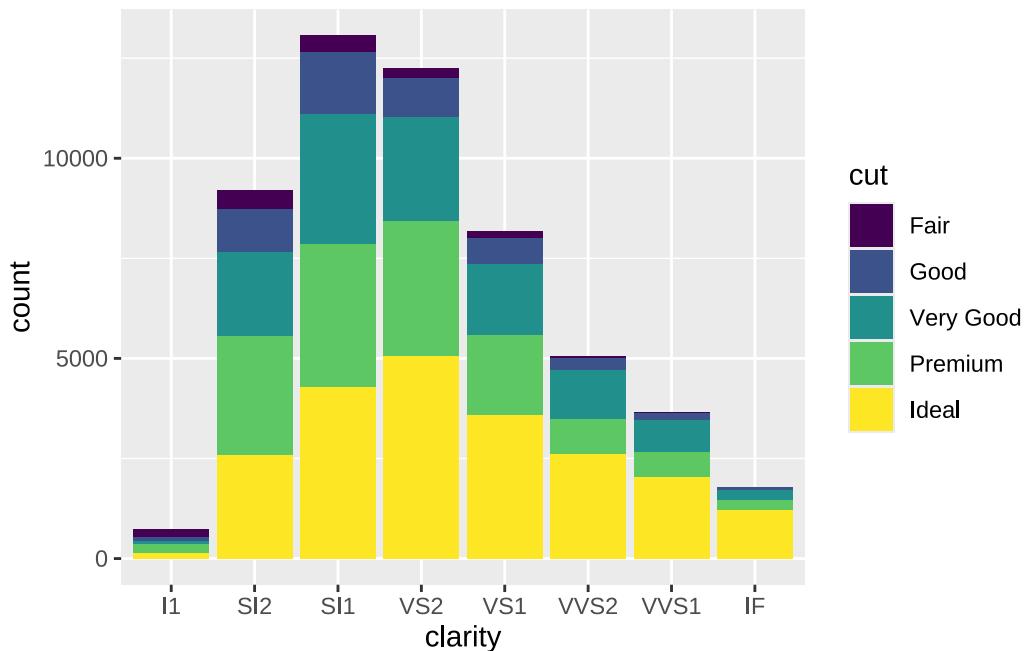
```
ggplot(data=mtcars,aes(wt, mpg, size=qsec))+  
  geom_text(family="EmojiOne",label="\U1F697") +  
  scale_size_continuous(range = c(5,10))
```



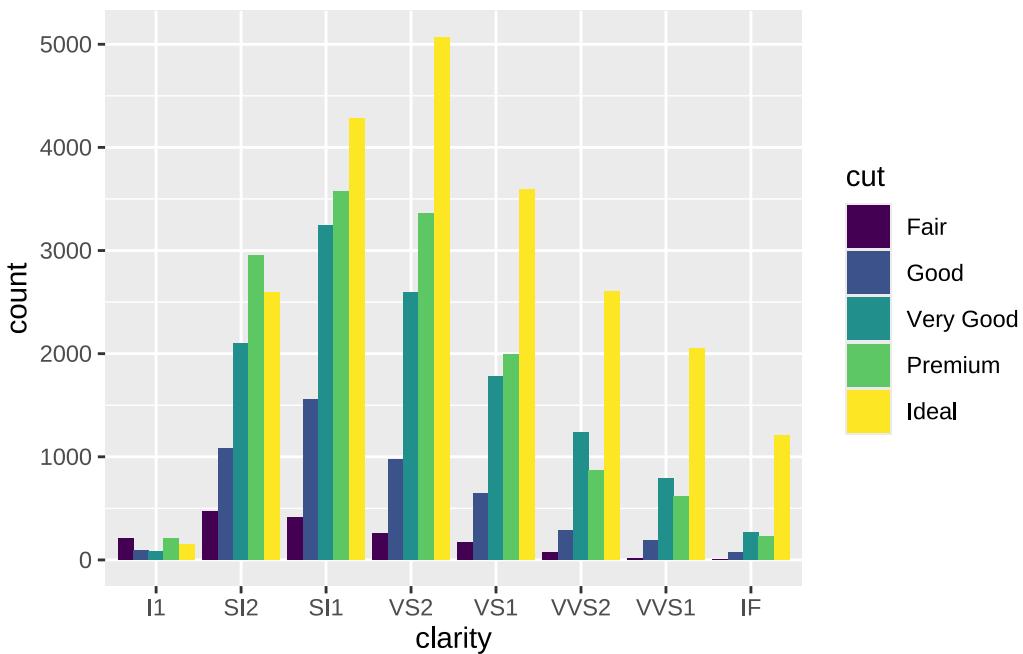
## 7.9 Positioning elements

The position arguments allows stacking, dodging, jittering and exact positioning of elements. Positioning is an essential part of storytelling, the same data can be presented with different focus.

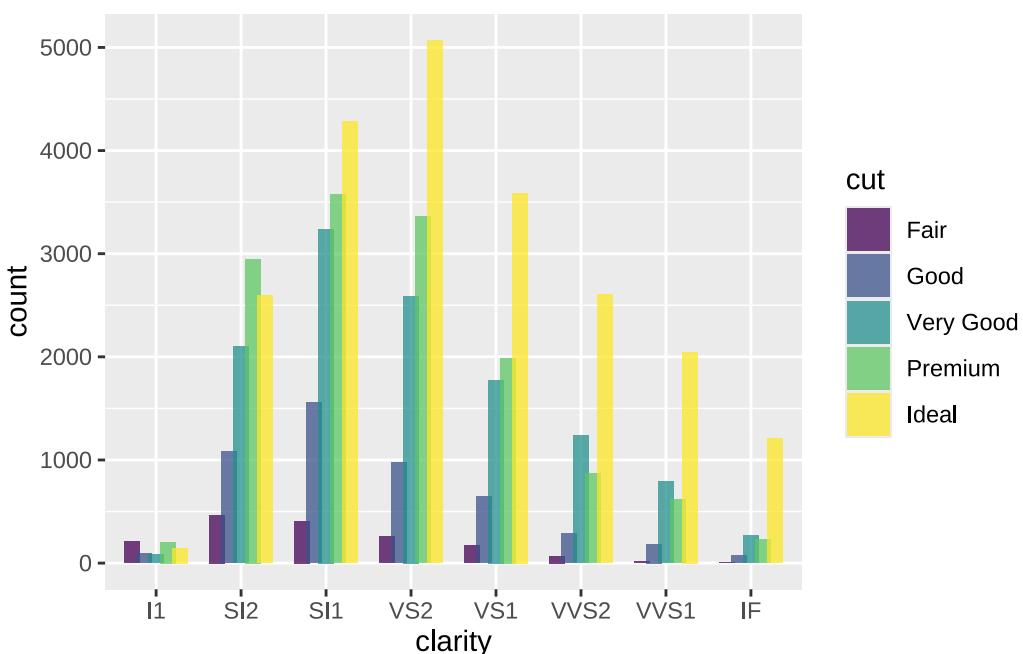
```
p<-ggplot(data=diamonds,aes(clarity,fill=cut))  
p+geom_bar(position="stack") # default for bar
```



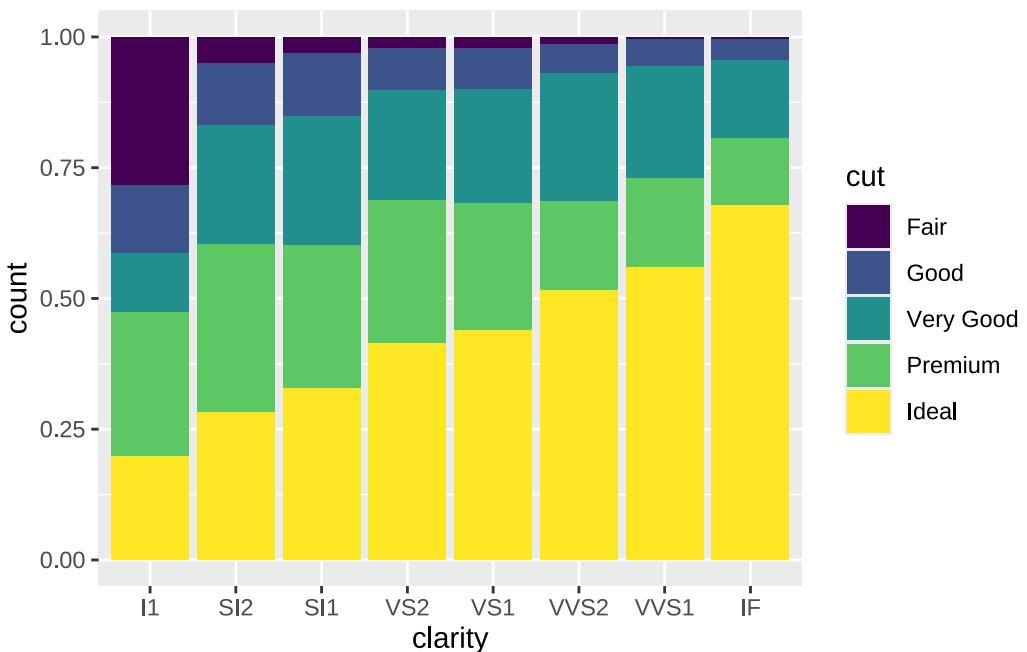
```
p+geom_bar(position="dodge")
```



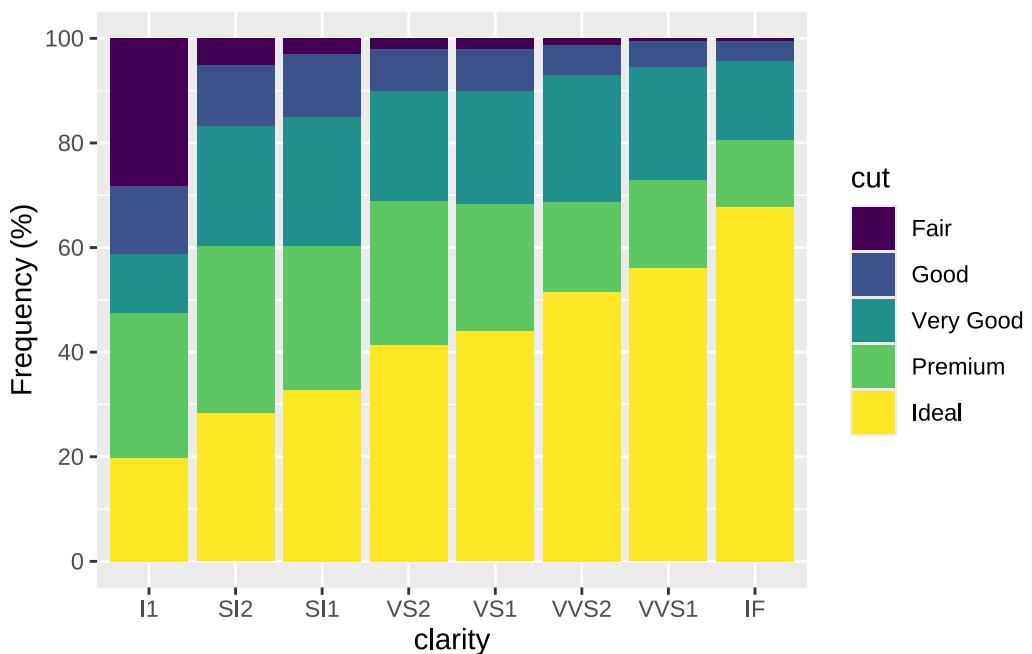
```
p+geom_bar(position=position_dodge(width = 0.7), alpha=.75)
```



```
p+geom_bar(position="fill") #y-axis labeling needs tuning
```



```
p+geom_bar(position="fill")+
  scale_y_continuous(name = "Frequency (%)",
                     breaks=seq(0,1,.2), #steps
                     labels=seq(0,100,20))
```

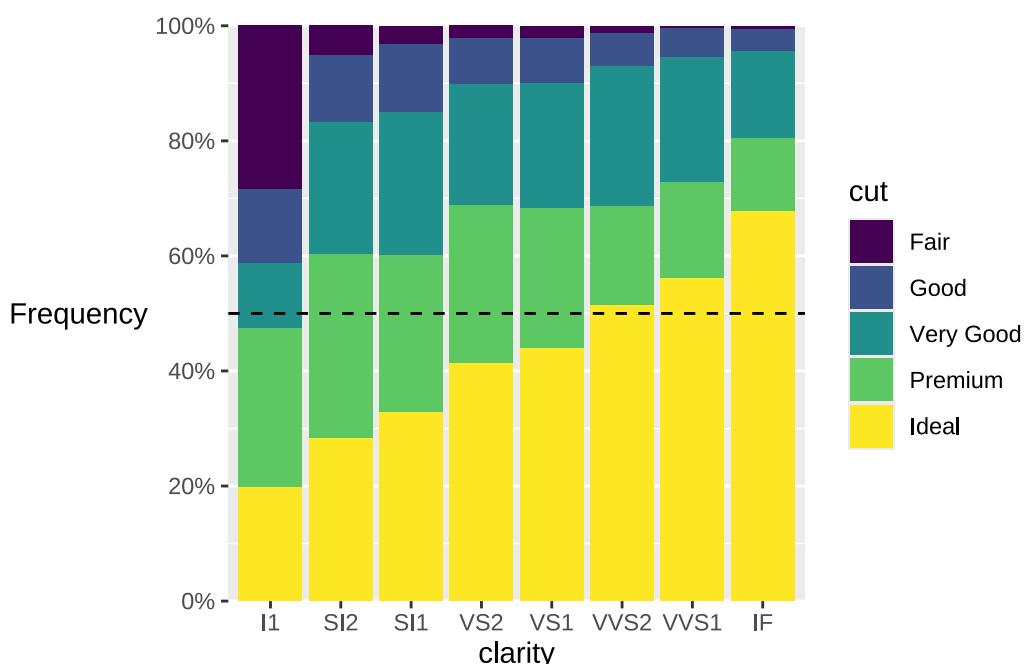


```

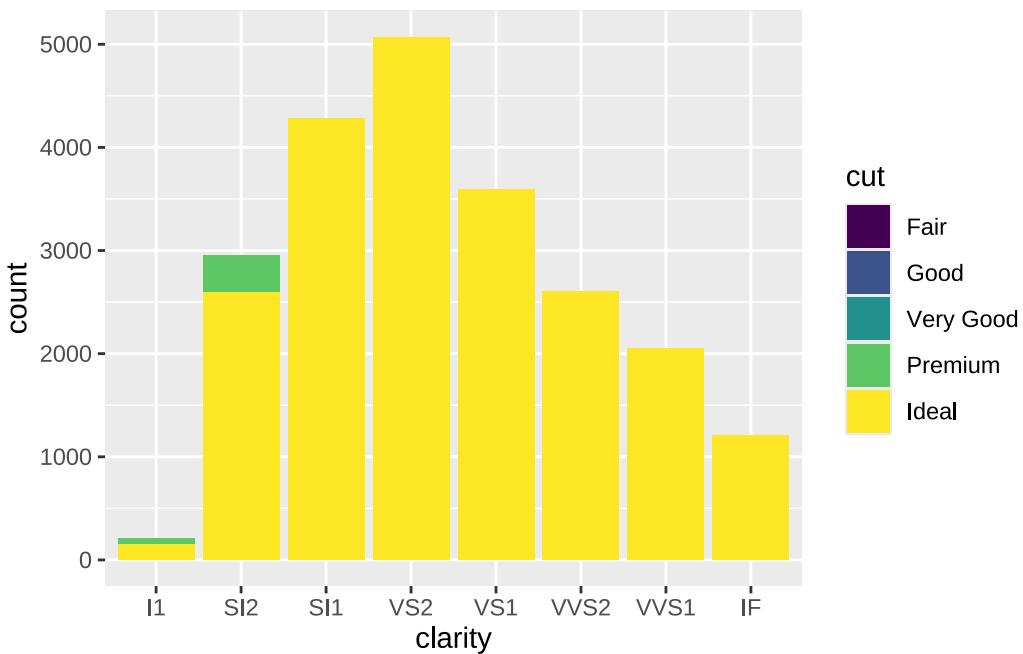
p+geom_bar(position="fill")+
  scale_y_continuous("Frequency",
                     breaks=seq(0,1,.2),
                     labels=scales::percent,
                     expand=expansion(mult = c(0,0)))+
  theme(axis.title.y = element_text(angle = 0,
                                    vjust = .5))+  

  geom_hline(yintercept = .5, linetype=2) # e.g. for reference lines

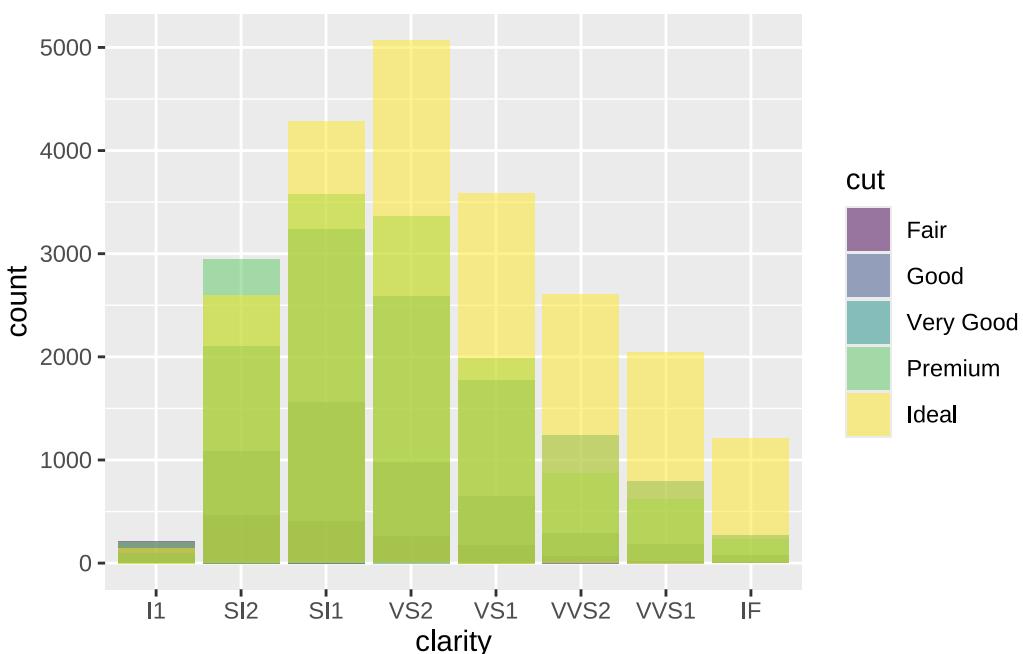
```



```
p+geom_bar(position="identity") # bad idea!
```

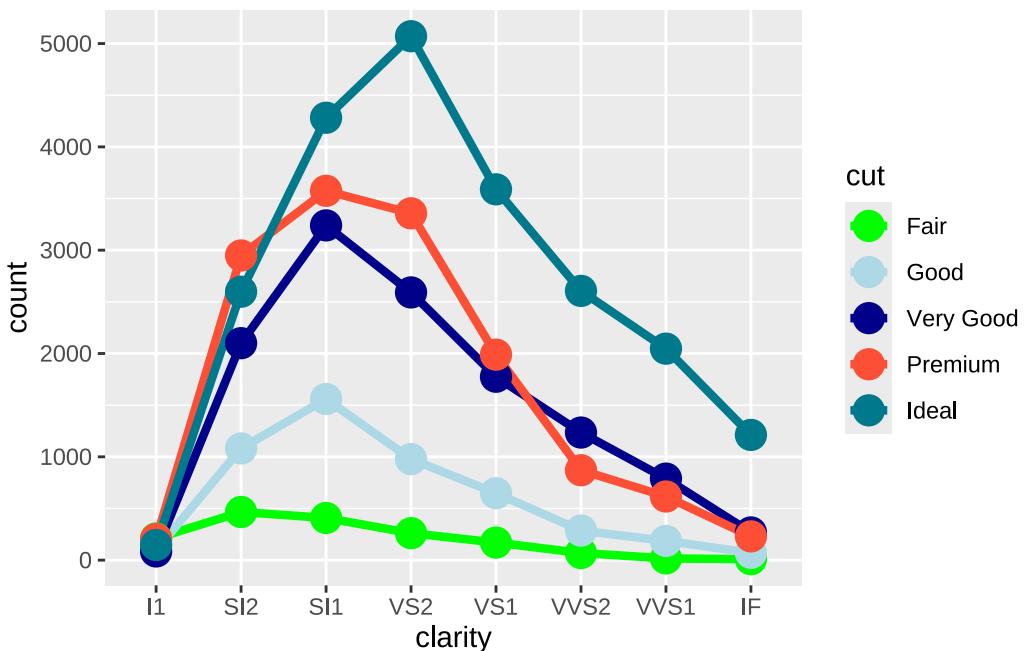


```
p+geom_bar(position="identity",alpha=.5) # even worse!!
```



```
ggplot(data=diamonds,aes(clarity,color=cut, group=cut))+  
  geom_line(stat="count",position="identity",lwd=1.5)+  
  geom_point(stat="count",size=5)+
```

```
scale_color_manual(values = c("green","lightblue",
                             "darkblue",
                             "rgb(253,79,54,
                               maxColorValue = 255),
                             "#00798d"))
```

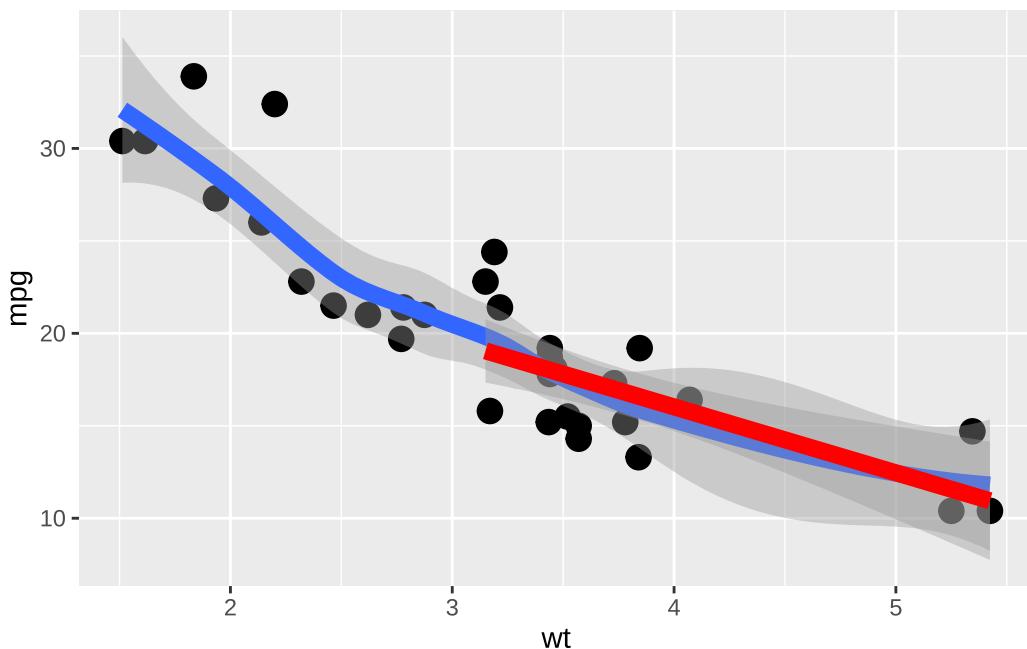


## 7.10 Order of layers

When combining various geoms, the order is important, as elements are not transparent by default.

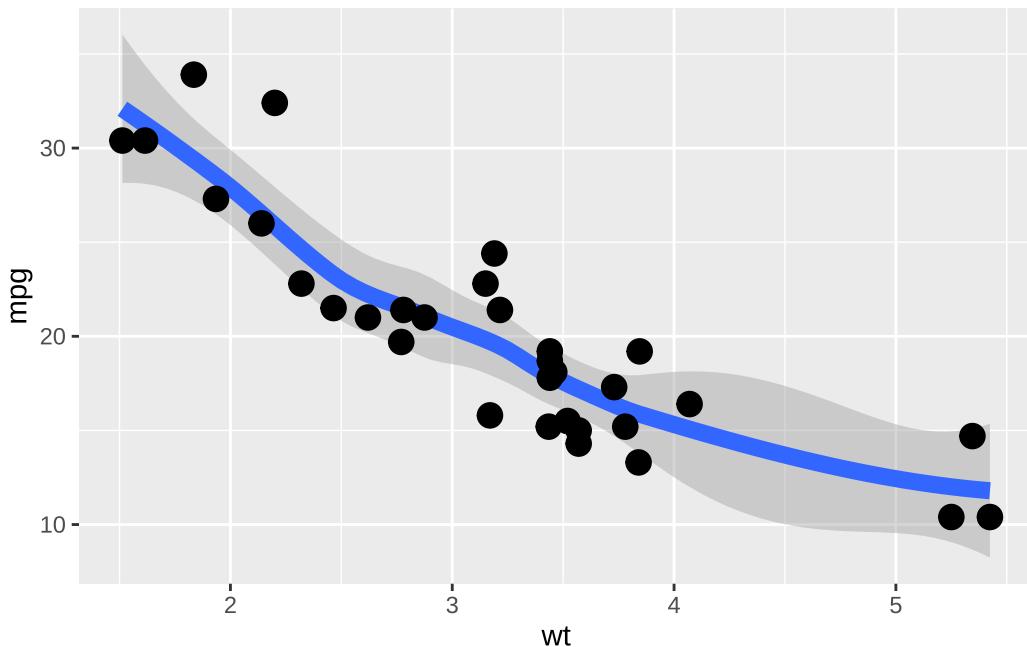
```
ggplot(data=mtcars,aes(wt, mpg))+  
  geom_point(size=4)+  
  geom_smooth(linewidth=3)+ # line overlaps points  
  geom_smooth(data=mtcars |> filter(wt>3), #picks a sub-sample  
              method="lm", linewidth=3, color="red")
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'  
`geom_smooth()` using formula = 'y ~ x'
```



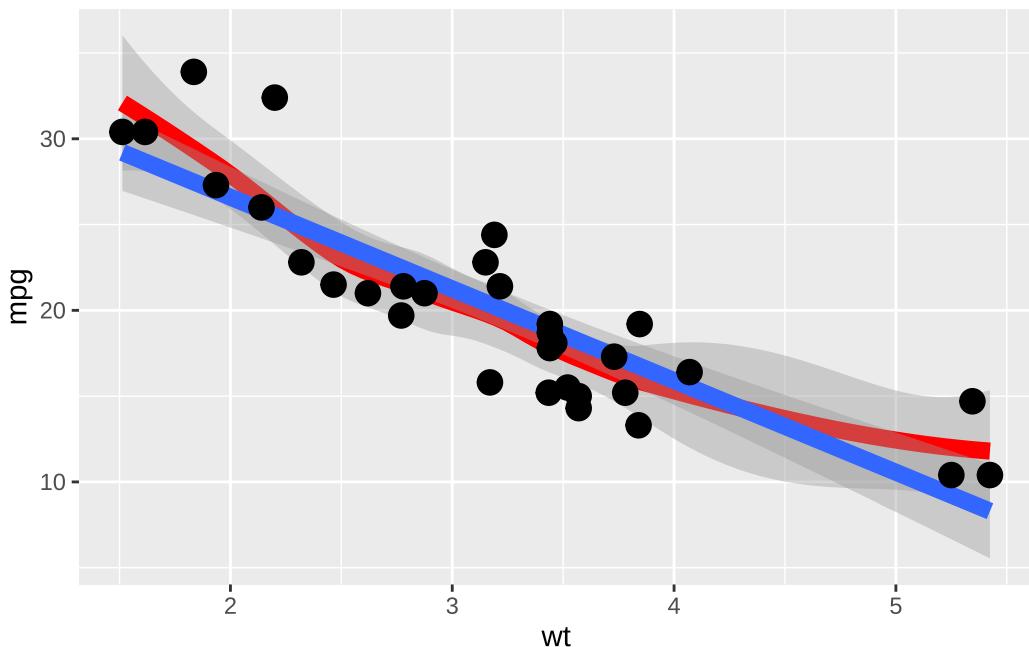
```
ggplot(data=mtcars,aes(wt, mpg))+  
  geom_smooth(linewidth=3)+  
  geom_point(size=4)
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



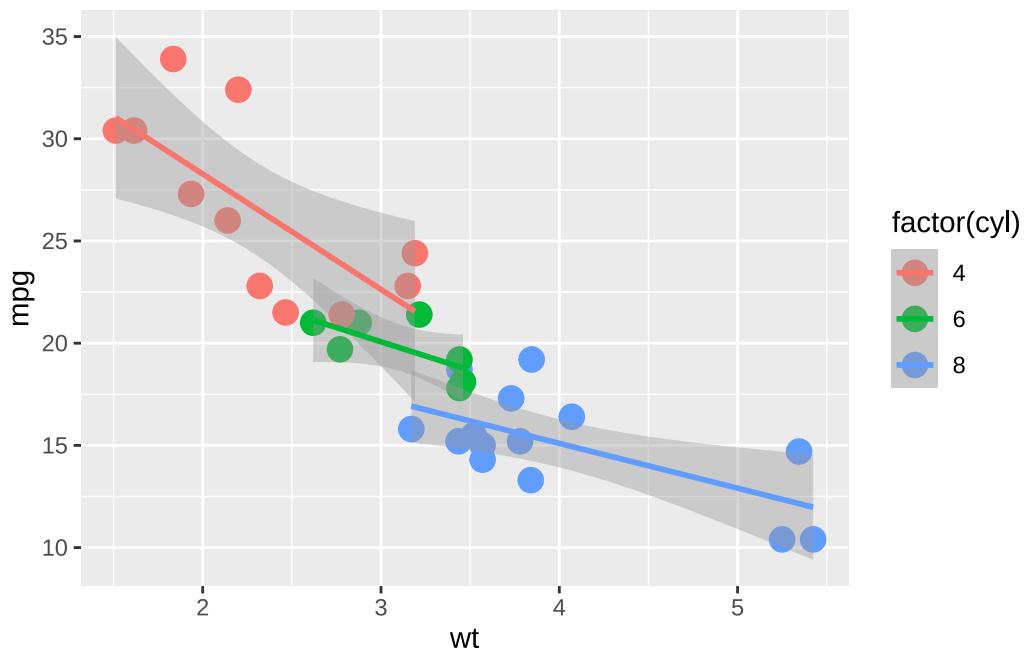
```
ggplot(data=mtcars,aes(wt, mpg))+  
  geom_smooth(linewidth=3,color="red") +  
  geom_smooth(method="lm", linewidth=3) +  
  geom_point(size=4)
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'  
`geom_smooth()` using formula = 'y ~ x'
```

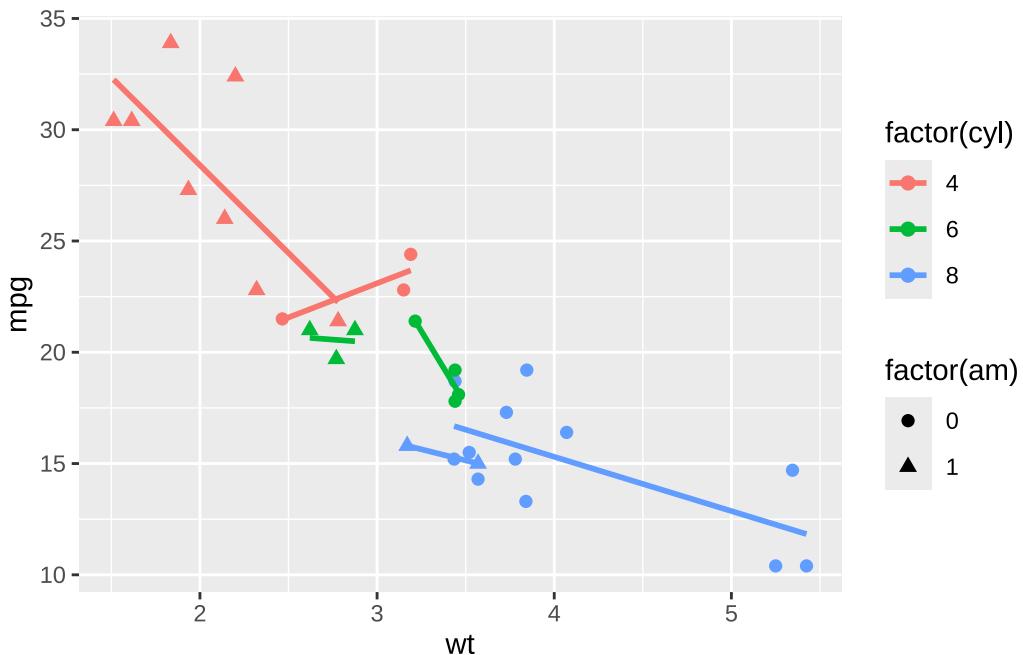


```
ggplot(data=mtcars,aes(wt, mpg,  
                      color=factor(cyl)))+  
  geom_point(size=4)+  
  geom_smooth(method="lm", linewidth=1)
```

```
`geom_smooth()` using formula = 'y ~ x'
```



```
ggplot(data=mtcars,aes(wt, mpg,
                        color=factor(cyl),
                        shape=factor(am)))+
  geom_point(size=2)+
  geom_smooth(method="lm", linewidth=1, se=FALSE)
`geom_smooth()` using formula = 'y ~ x'
```

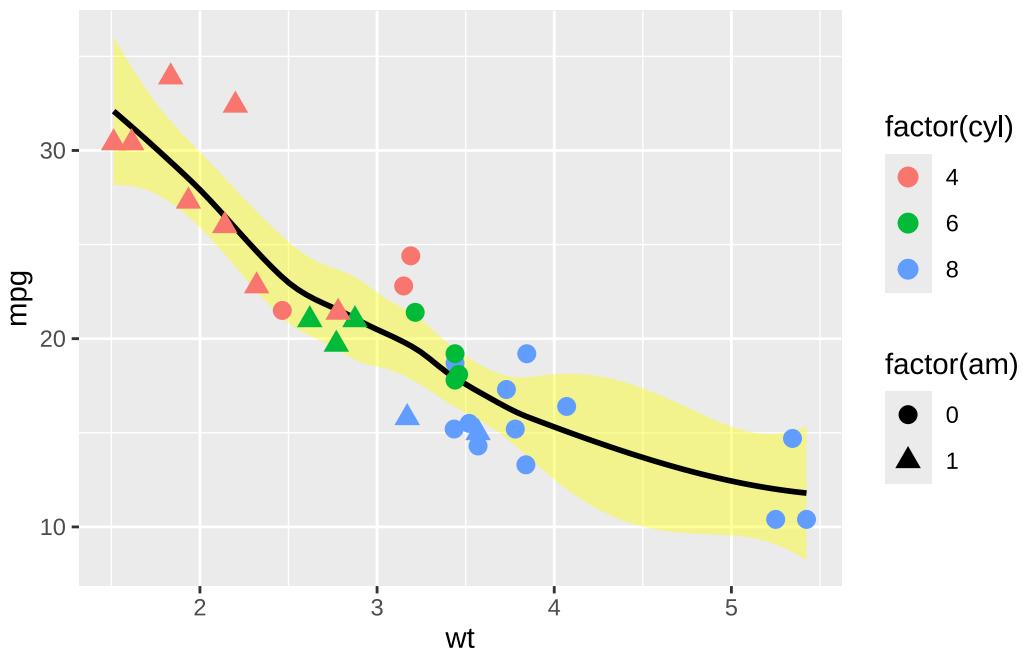


## 7.11 Local aesthetics for layers

```
#? lm for all?
ggplot(data=mtcars,aes(wt, mpg))+
  geom_smooth(size=1,color="black",fill="yellow")+
  geom_point(size=3,aes(color=factor(cyl),shape=factor(am))) #aes for geom only
```

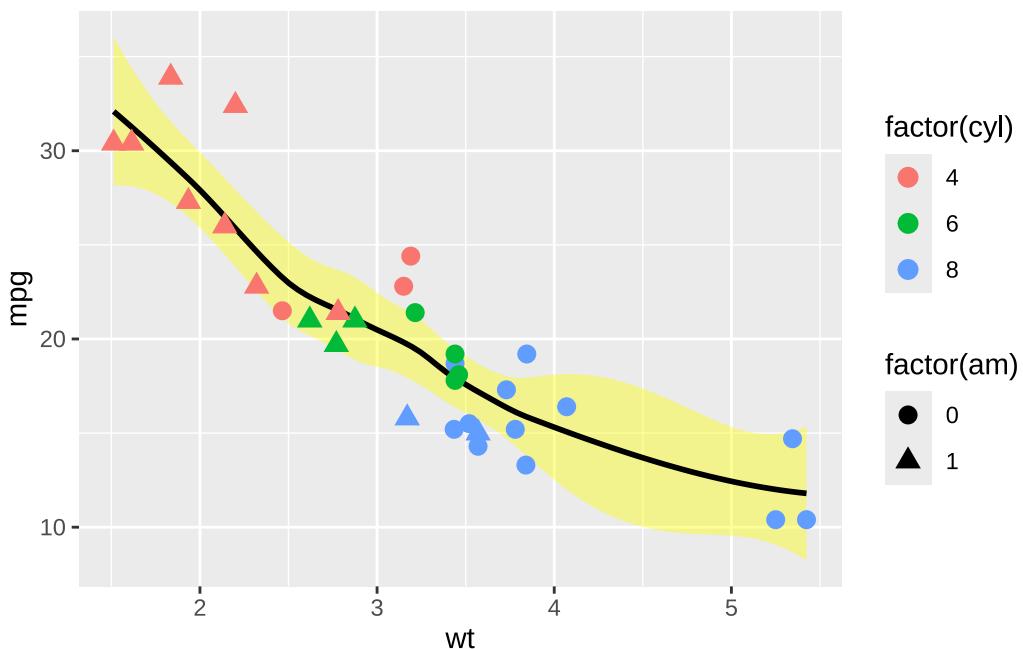
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.  
i Please use `linewidth` instead.

`geom\_smooth()` using method = 'loess' and formula = 'y ~ x'



```
ggplot(data=mtcars, aes(wt, mpg, color=factor(cyl)))+
  geom_smooth(size=1, color="black", fill="yellow")+ # global color overwritten
  geom_point(size=3, aes(shape=factor(am)))
```

`geom\_smooth()` using method = 'loess' and formula = 'y ~ x'



## 7.12 Faceting (splitting) plots

Visualizing many groups can lead to confusing / too-busy plots, splitting is often an alternative. Visualizing many variables at the same time can be achieved with facets as well (after pivot\_longer).

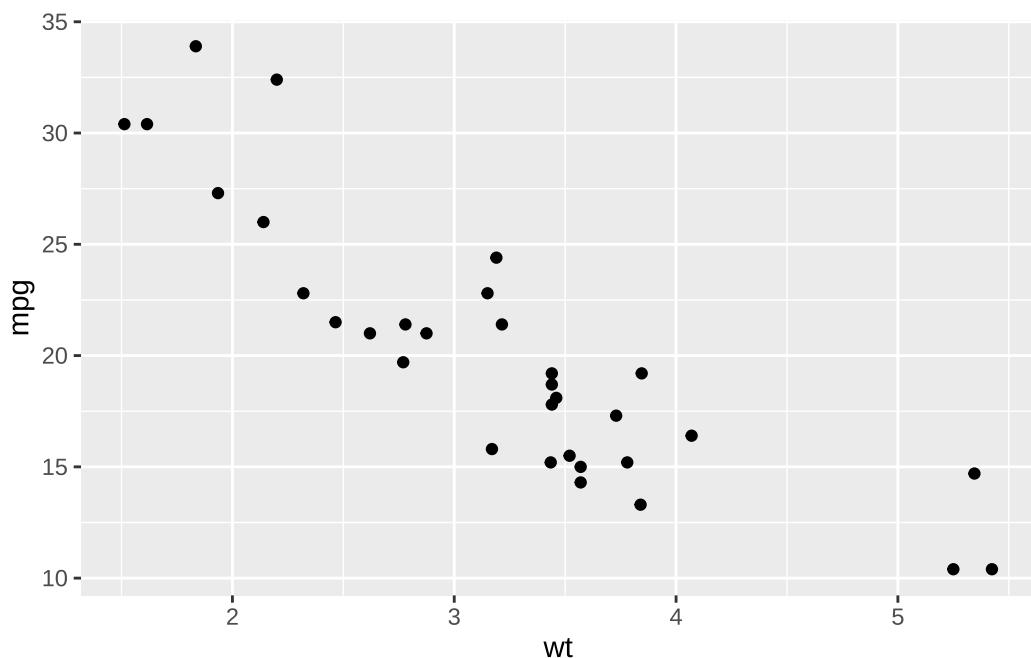
### 7.12.1 facet\_grid

Grids are specified by defining variables for rows and/or columns, empty combinations still are shown.

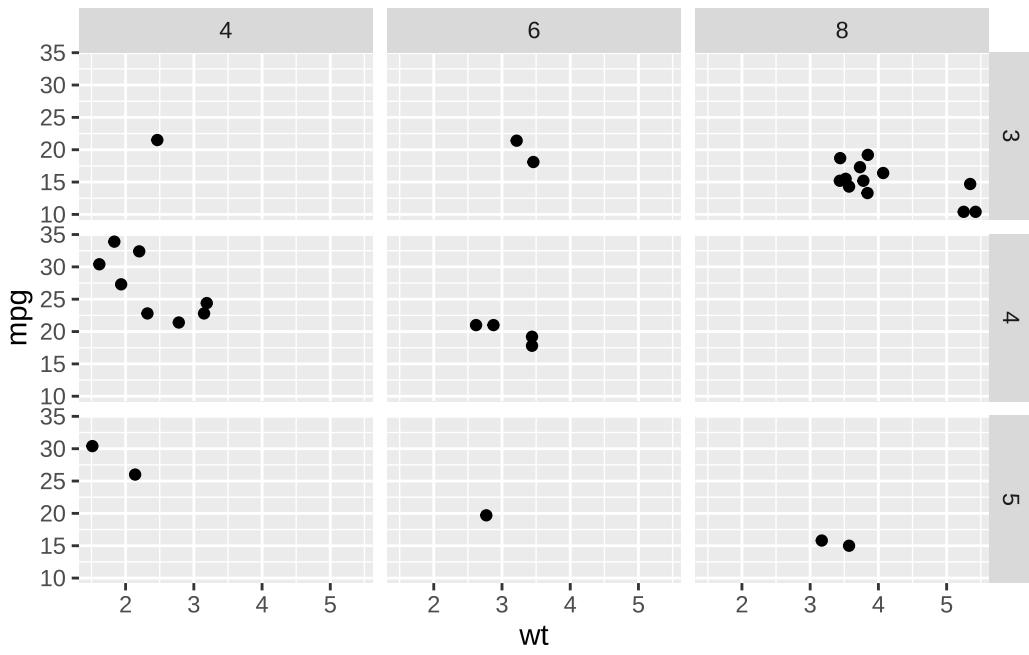
Labeling of facets often requires name and content to be informative.

Margins (taking all elements together) can be shown for rows and/or columns.

```
(plot_tmp <- ggplot(mtcars, aes(wt, mpg)) +  
  geom_point())
```



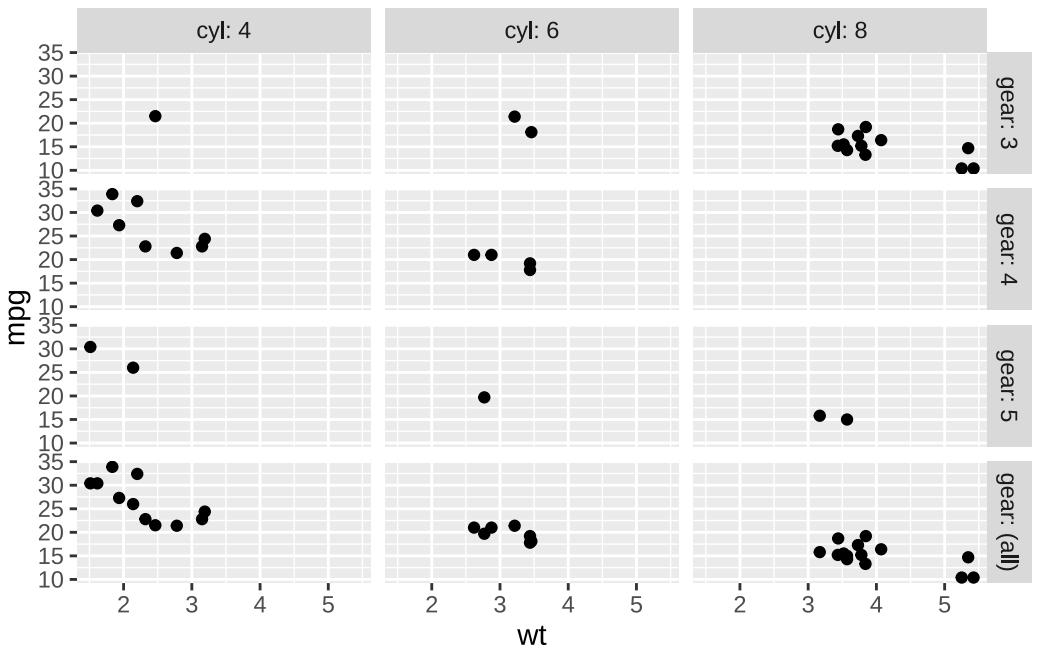
```
plot_tmp + facet_grid(rows = vars(gear),  
                      cols = vars(cyl))
```



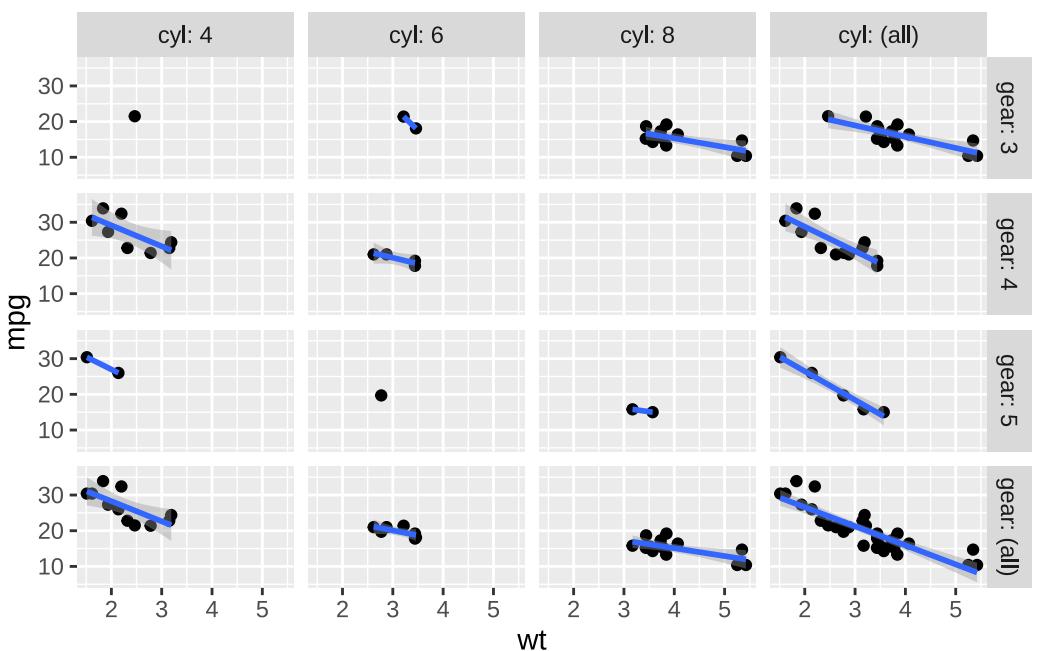
```
cat("facet labeling improved:\n")
```

facet labeling improved:

```
plot_tmp + facet_grid(rows = vars(gear),  
                      cols = vars(cyl),  
                      labeller=label_both,margins="gear")
```

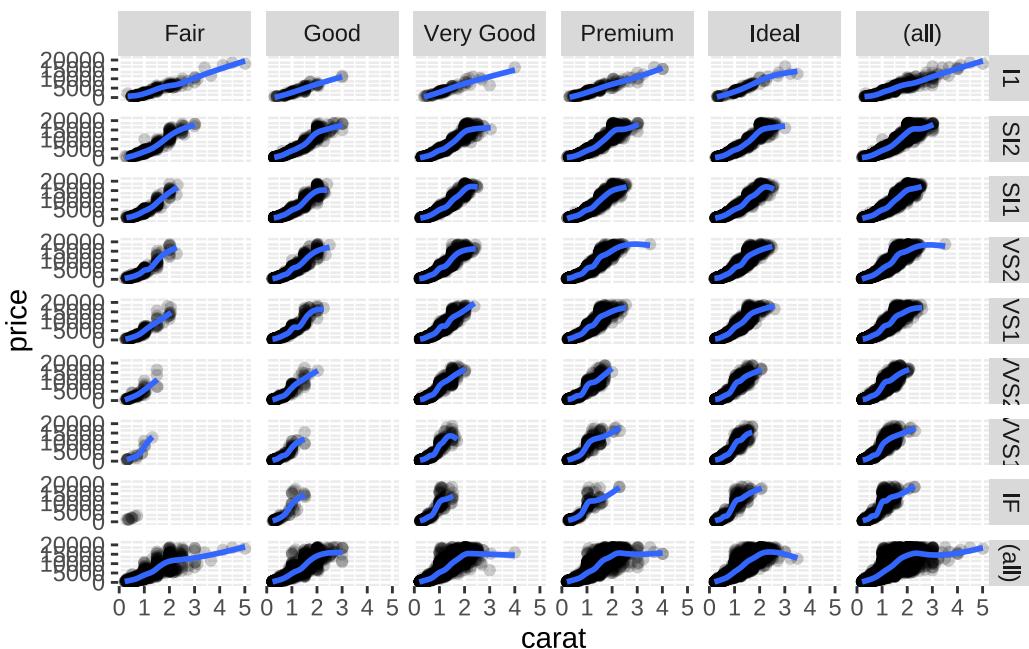


```
# options(warn=-1)
plot_tmp + geom_smooth(method="lm") +
  facet_grid(rows = vars(gear),
             cols = vars(cyl),
             labeller=label_both, margins=TRUE)
```



```
# options(warn=0)

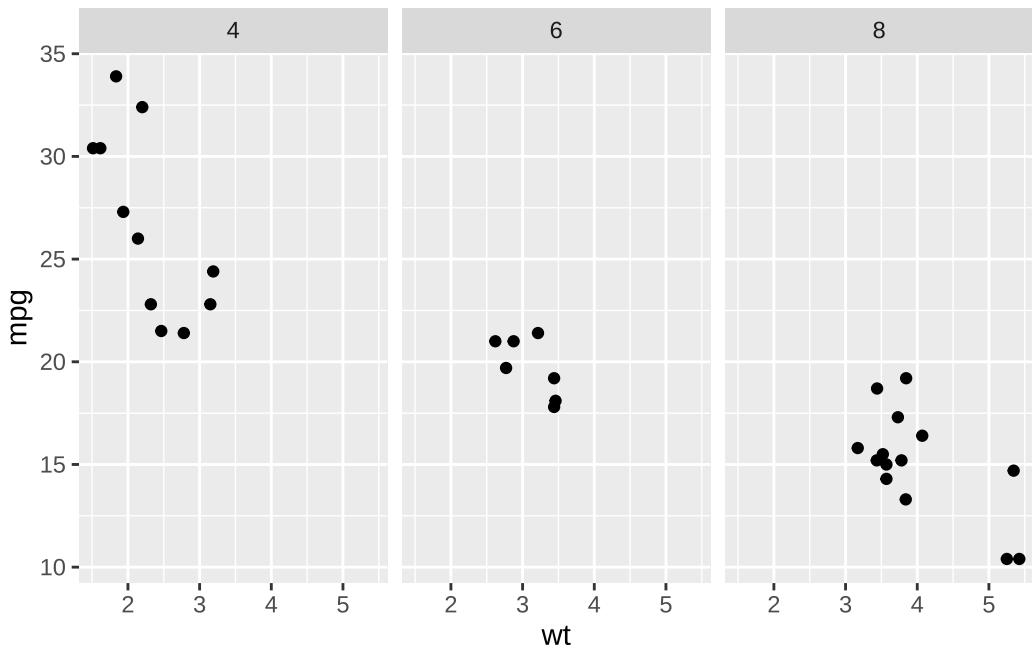
ggplot(diamonds,aes(carat,price))+
  geom_point(alpha=.2)+
  geom_smooth()+
  facet_grid(rows = vars(clarity),
             cols = vars(cut),
             margins=TRUE)
```



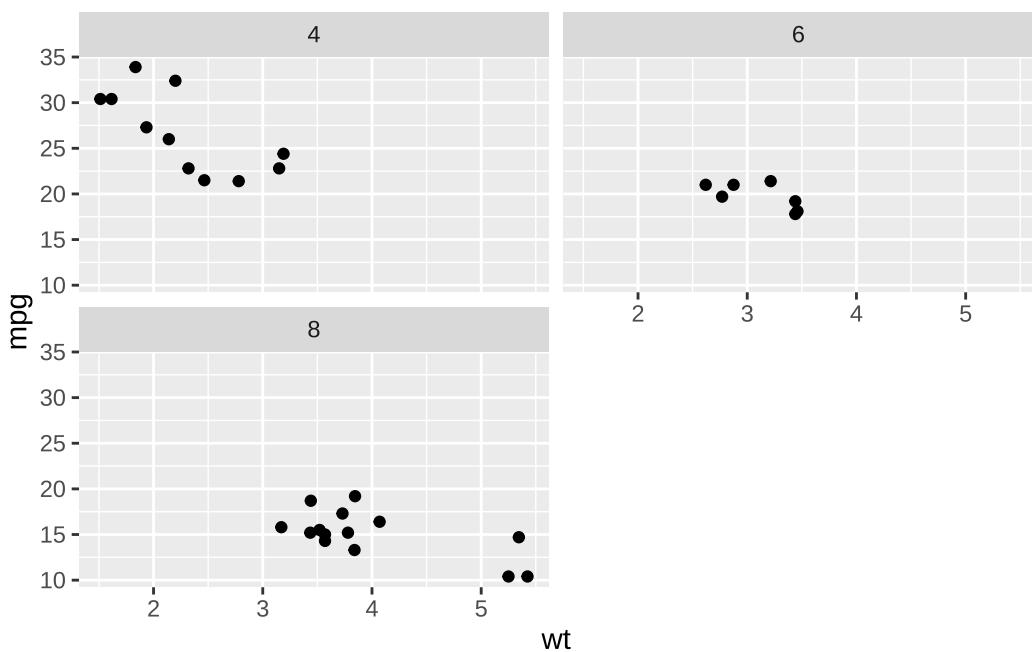
### 7.12.2 `facet_wrap`

When showing many facets, wrapping around after some is useful, less systematic than `grid`.

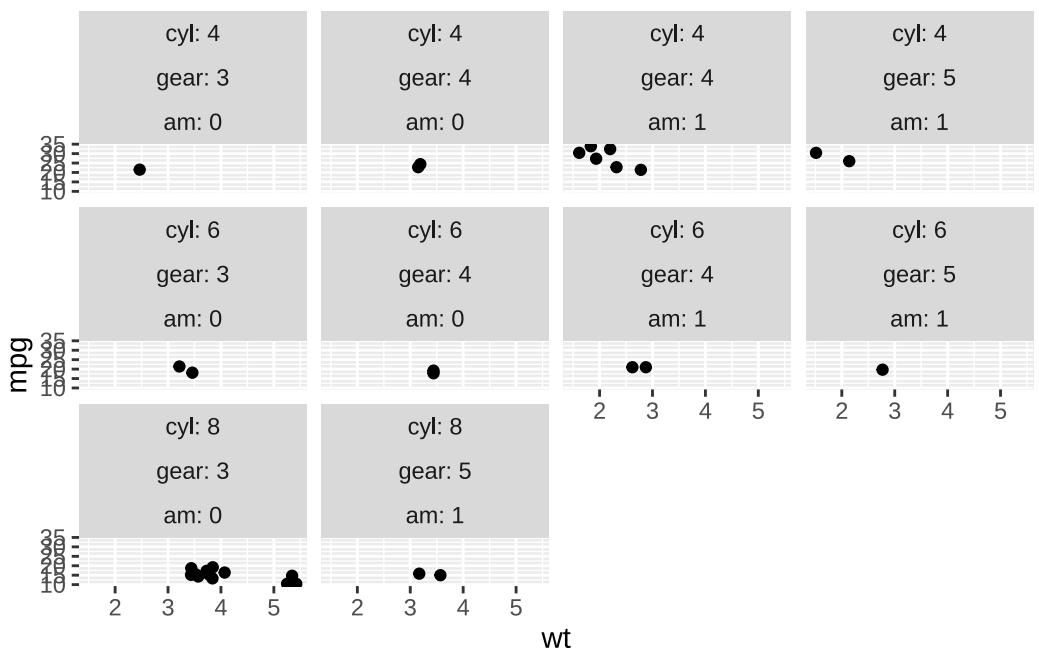
```
plot_tmp + facet_wrap(facets = vars(cyl))
```



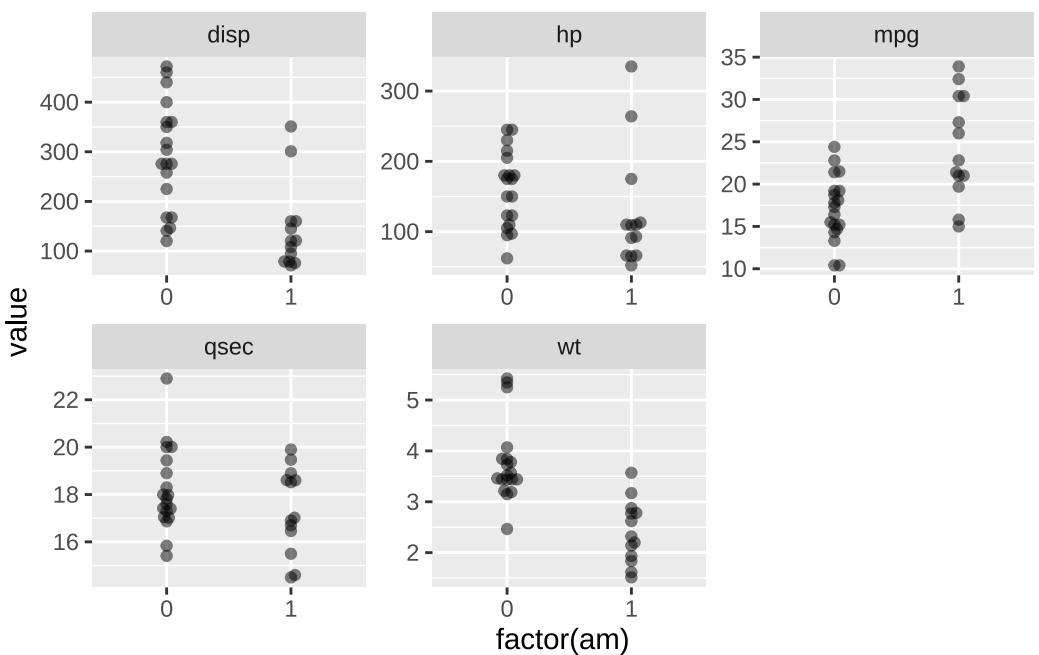
```
plot_tmp + facet_wrap(facets = vars(cyl), ncol=2)
```



```
# empty combination is dropped
plot_tmp + facet_wrap(facets=vars(cyl,gear,am),labeler=label_both)
```

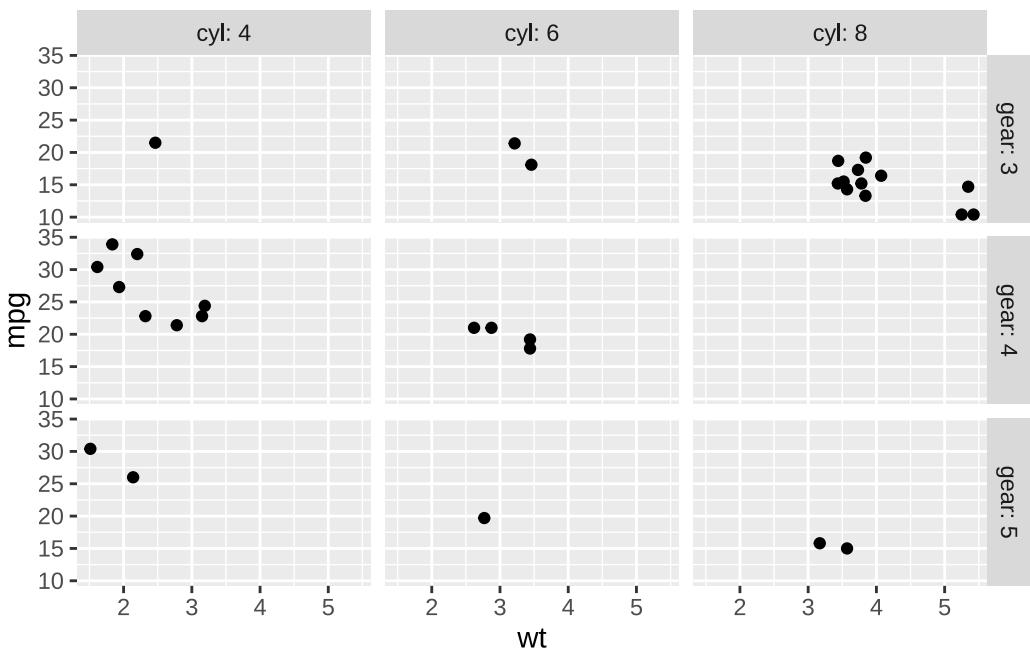


```
#combining variables
mtcars |>
  pivot_longer(cols = c(wt, mpg, hp, disp, qsec)) |> #view()
  ggplot(aes(x=factor(am), y=value))+
  geom_beeswarm(alpha=.5, cex=2)+
  facet_wrap(facets = vars(name), scales="free")
```

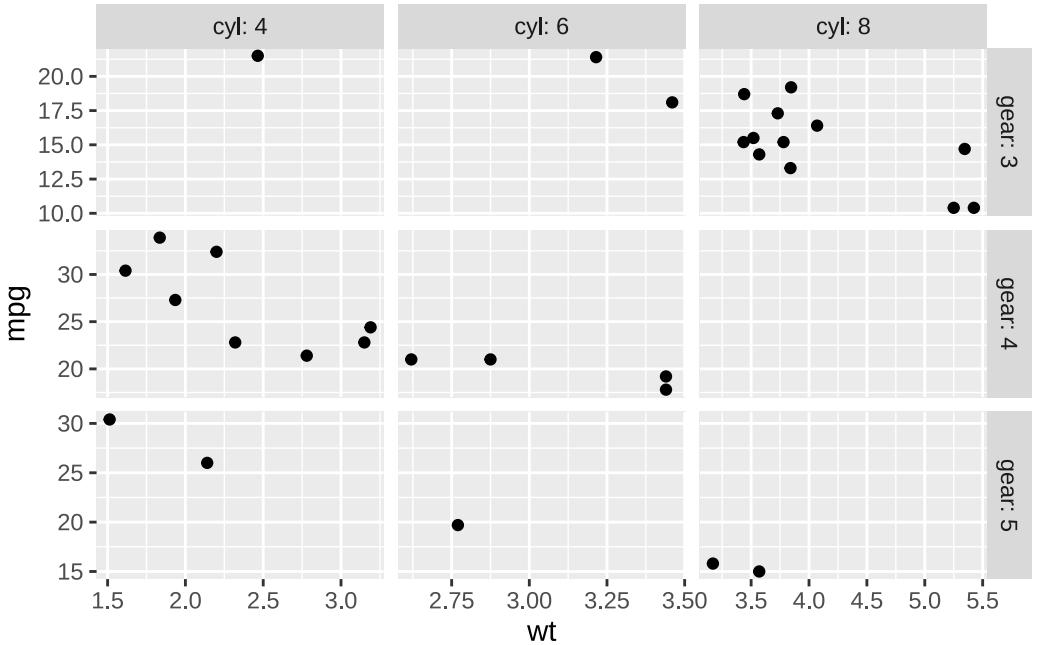


### 7.12.3 Controlling scales in facets (default: scales="fixed")

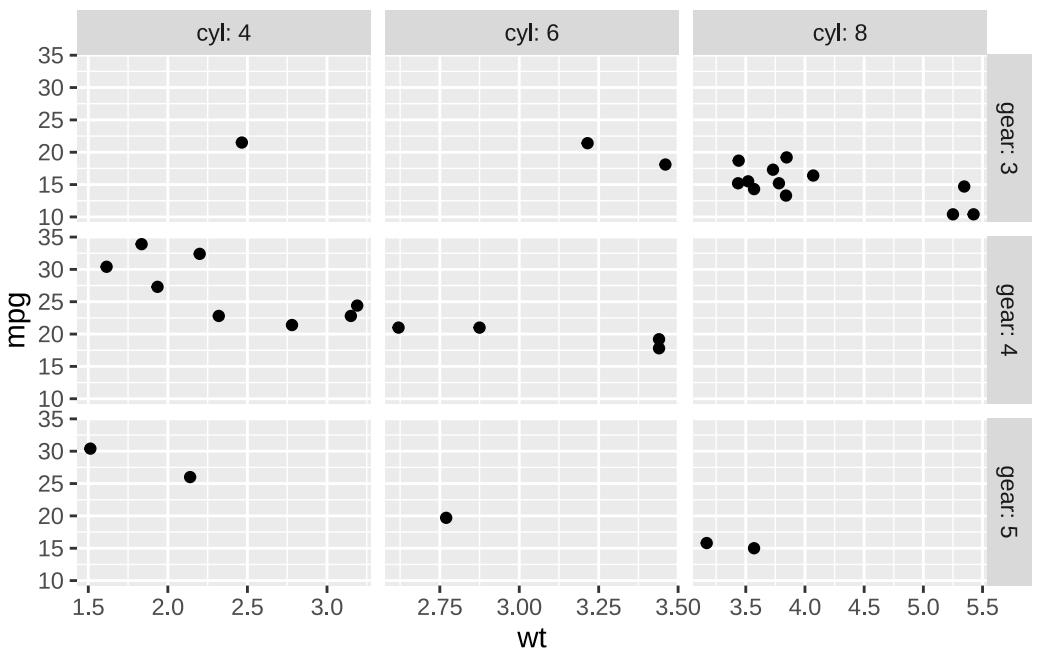
```
plot_tmp + facet_grid(rows=vars(gear),cols=vars(cyl),  
                      labeller=label_both, scales="fixed")
```



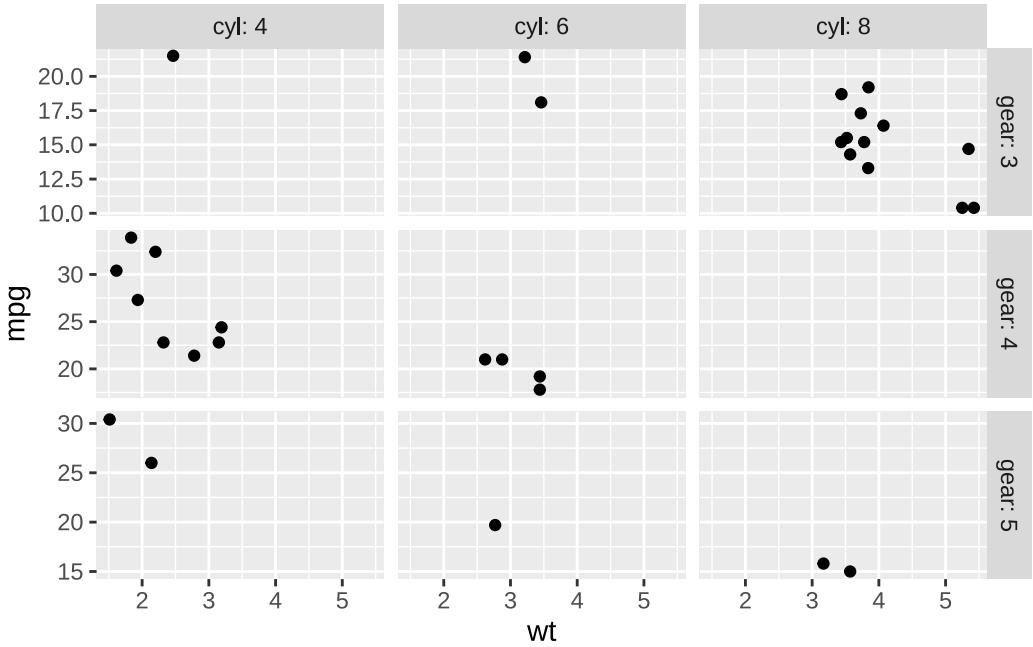
```
plot_tmp + facet_grid(rows=vars(gear),cols=vars(cyl),  
                      labeller=label_both, scales="free")
```



```
plot_tmp + facet_grid(rows=vars(gear),cols=vars(cyl),
                      labeller=label_both, scales="free_x")
```



```
plot_tmp + facet_grid(rows=vars(gear),cols=vars(cyl),
                      labeller=label_both, scales="free_y")
```



## 7.13 Exercise 2

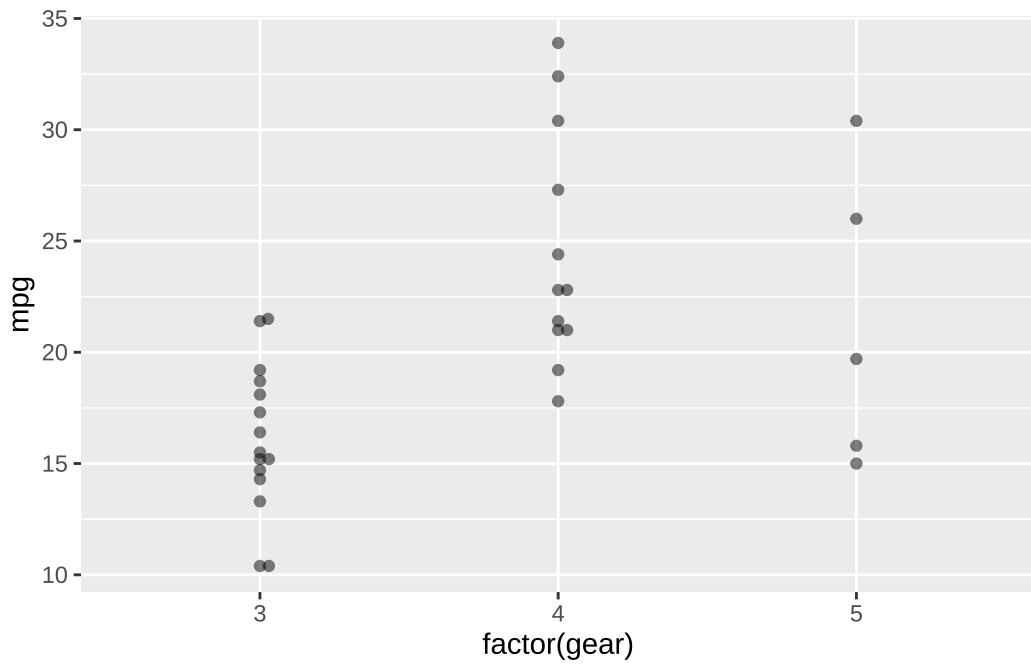
- distribution of body mass by species and sex (density, histogram)
- without / with facetting

## 7.14 Showing summaries

While plotting underlying rawdata is pretty informative, adding summary statistics guides the viewer. Error bars help to evaluate differences visible, but need to be labelled!!

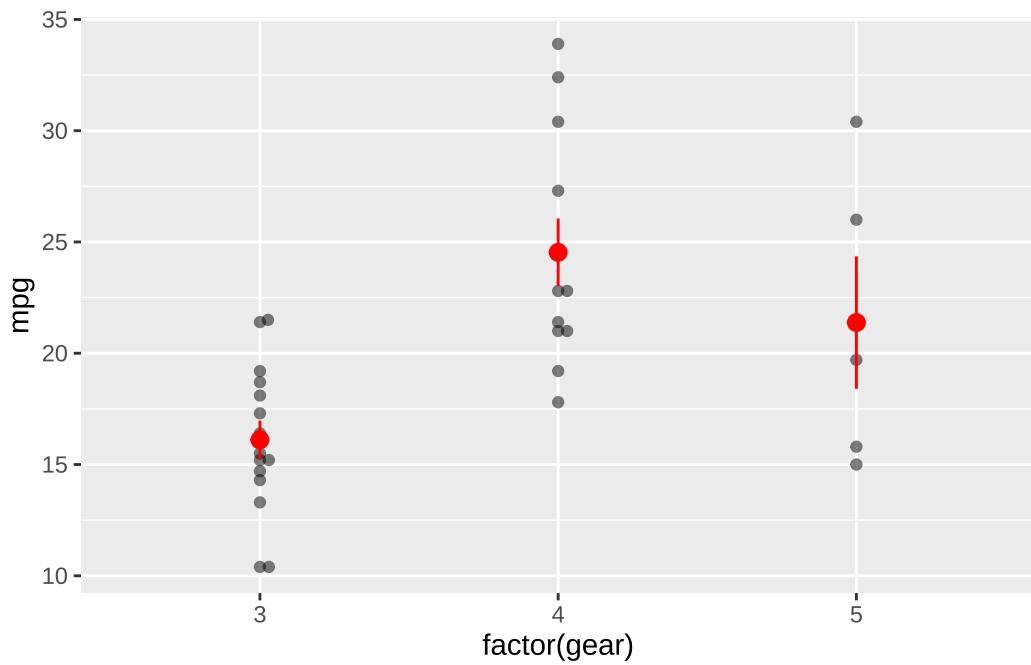
Functions for summary statistics (mean\_se, mean\_cl\_normal, mean\_cl\_boot etc.) are build on top of Hmisc functions. So this package is needed but not automatically installed with ggplot2.

```
(plottemp <- ggplot(mtcars,aes(factor(gear),mpg))+  
  geom_beeswarm(alpha=.5))
```

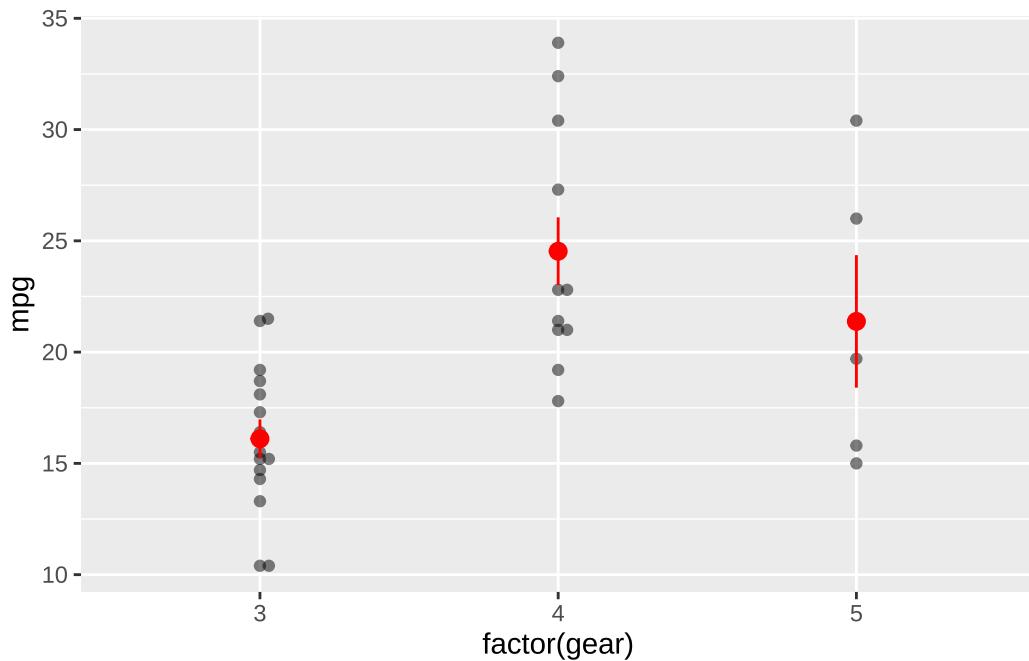


```
plottemp+stat_summary(color="red")
```

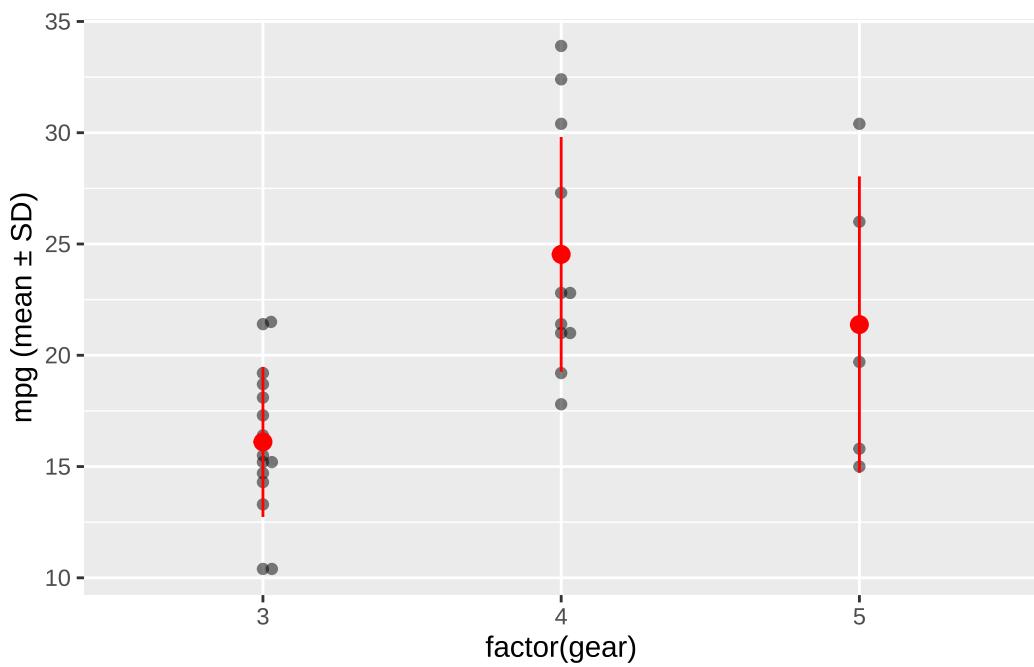
No summary function supplied, defaulting to `mean\_se()`



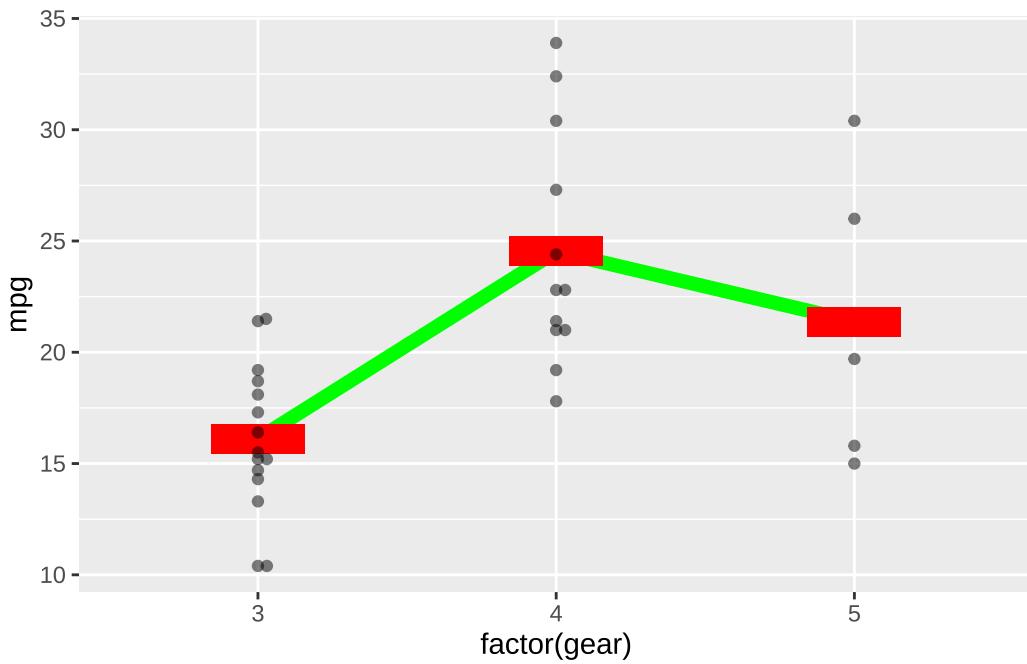
```
plottemp+stat_summary(fun.data="mean_se",
                      color="red")
```



```
plottemp+stat_summary(fun.data="mean_sdl",
                      fun.args=list(mult=1),
                      color="red")+
  ylab("mpg (mean \u00b1 SD)")
```



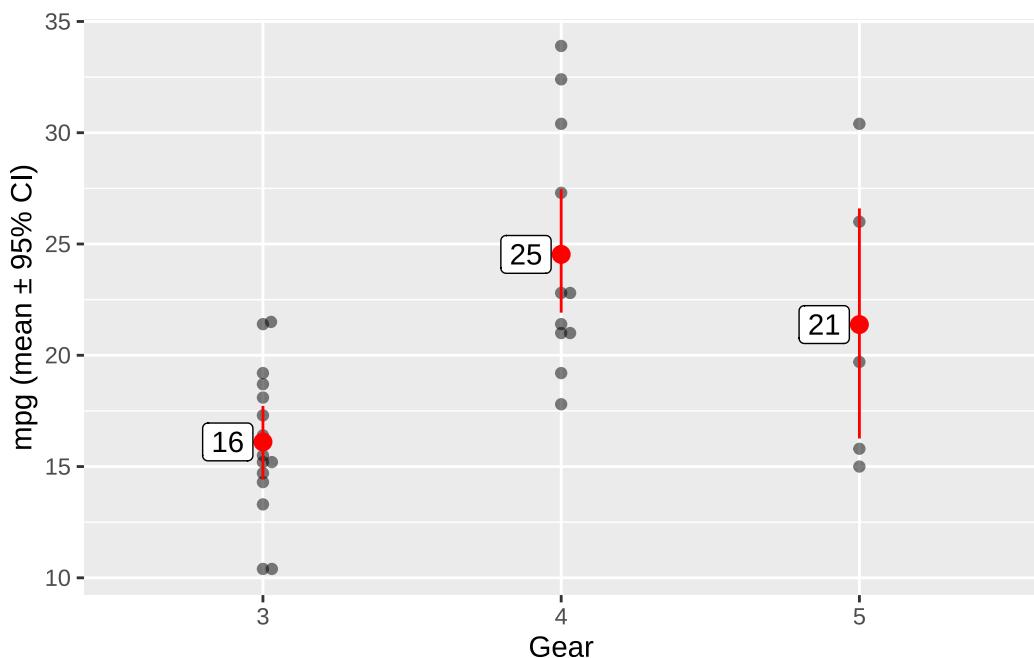
```
ggplot(mtcars,aes(factor(gear),mpg))+  
  stat_summary(geom = "line", aes(group="all"),  
               size=2.5,  
               fun = "mean",color="green") +  
  stat_summary(geom = "point", shape="-",  
               size=50,  
               fun = "mean",color="red") +  
  geom_beeswarm(alpha=.5)
```



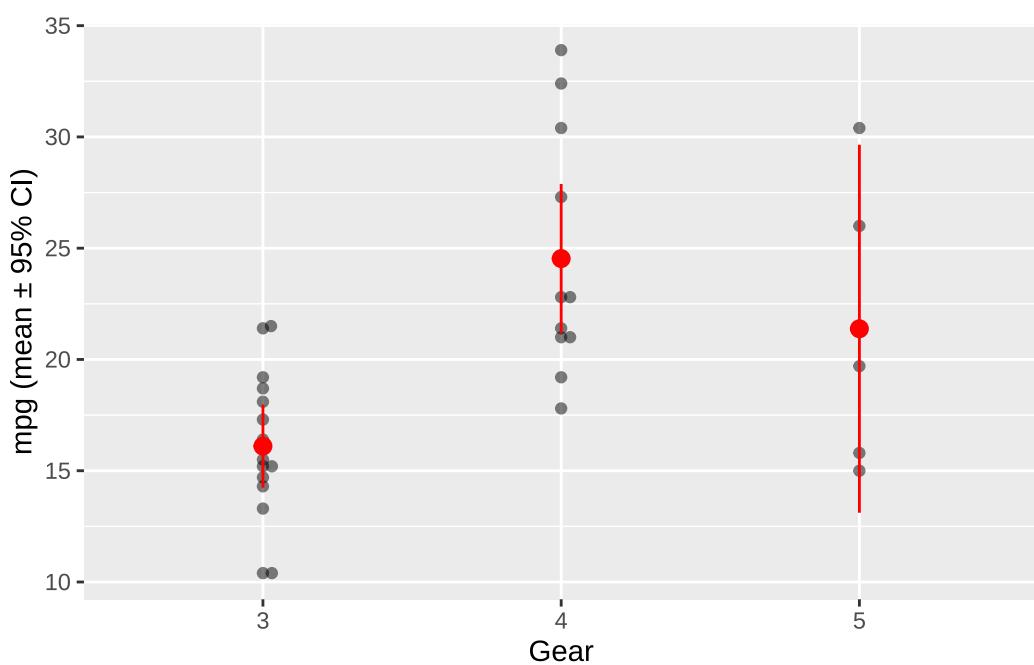
```

means <- mtcars |>
  group_by(gear) |>
  summarise(mean=round(mean(mpg),3),sd=sd(mpg))
plottemp+stat_summary(fun.data="mean_cl_boot",
                      fun.args=list(B=10^4),
                      color="red")+
  geom_label(data=means,
             aes(factor(gear),mean,label=round(mean)),
             hjust=1.2)+
  ylab("mpg (mean \u00b1 95% CI)")+
  xlab("Gear")

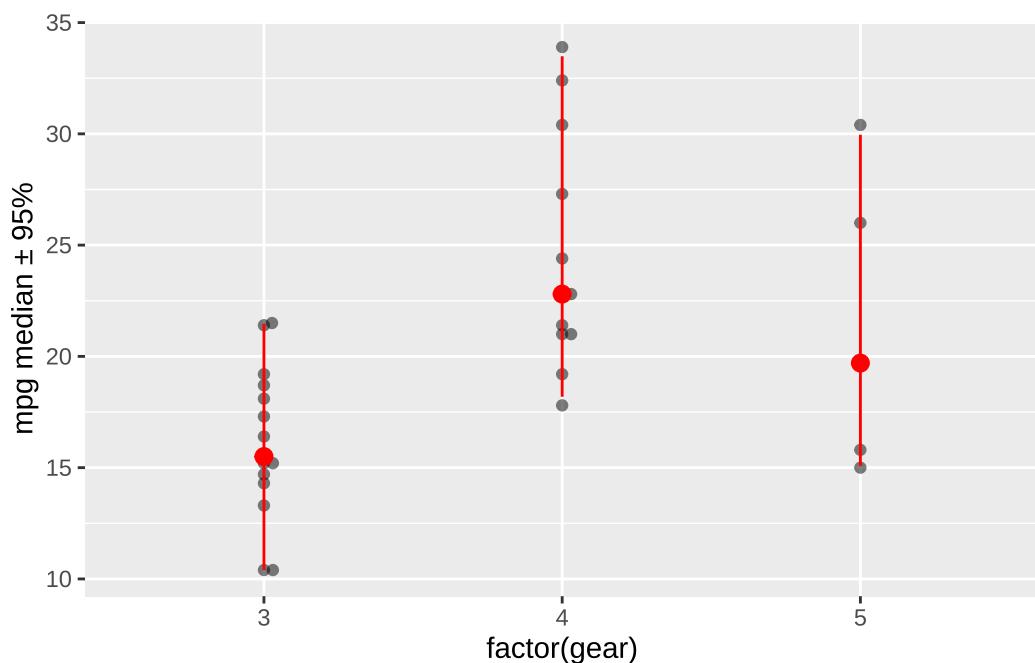
```



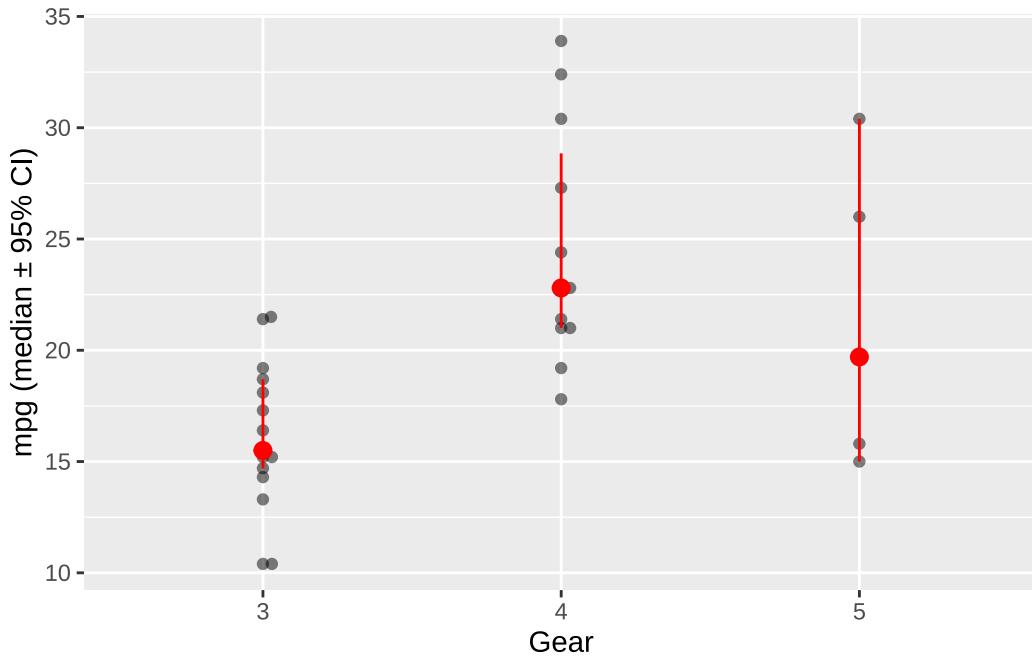
```
plottemp+stat_summary(fun.data="mean_cl_normal",color="red")+
  ylab("mpg (mean \u00b1 95% CI)")+
  xlab("Gear")
```



```
plottemp+stat_summary(fun.data="median_hilow",color="red")+
  ylab("mpg median \u00b1 95%")
```



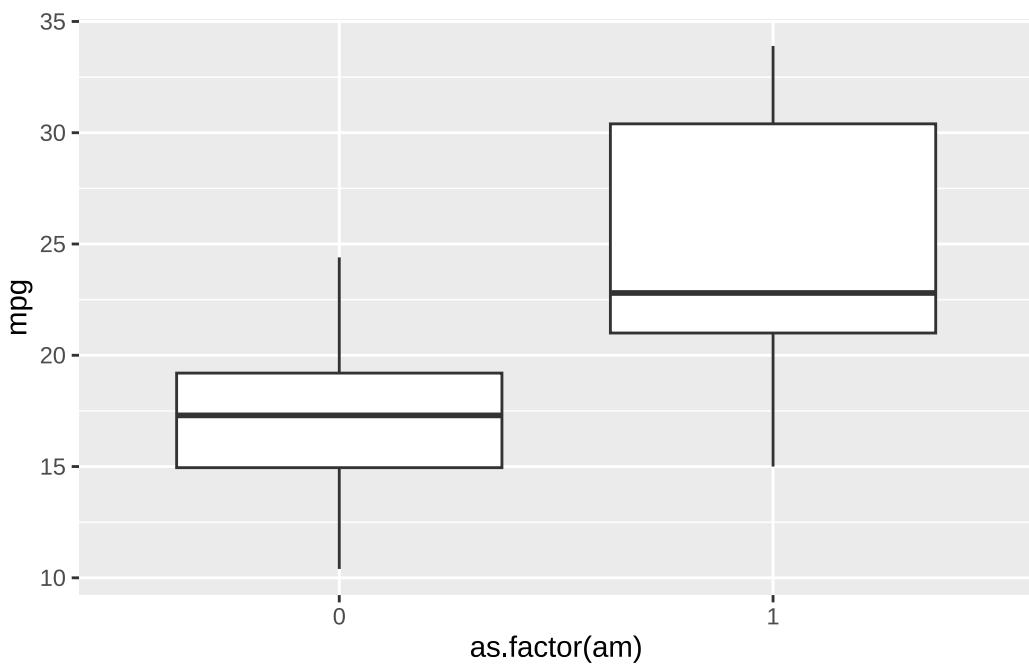
```
# geom_pointrange()
plottemp+stat_summary(fun.data="median_cl_boot_gg",color="red")+
  ylab("mpg (median \u00b1 95% CI)")+
  xlab("Gear")
```



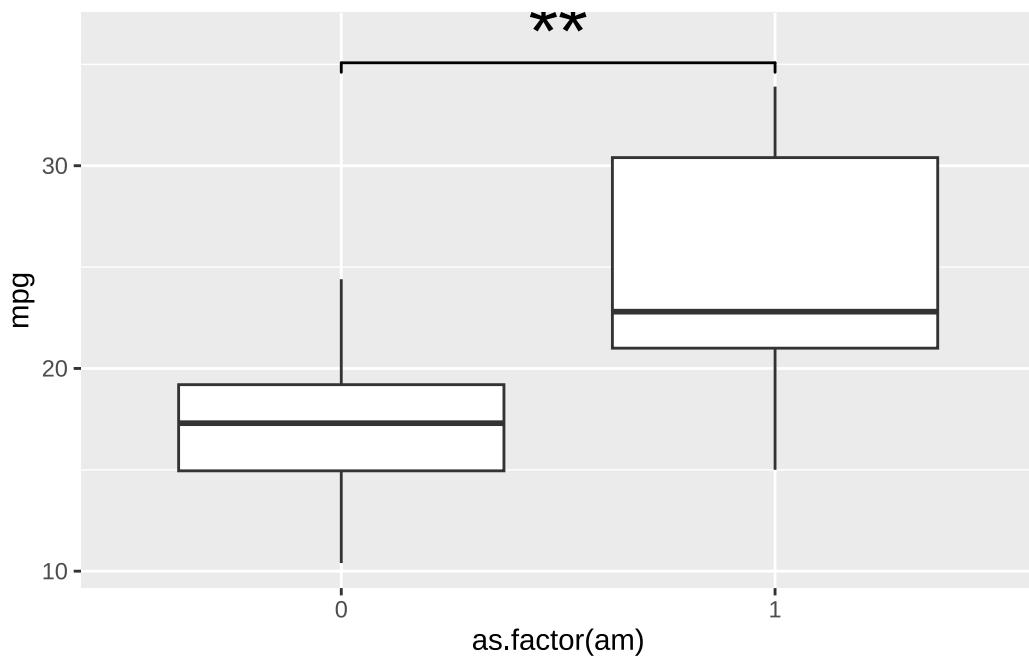
## 7.15 Indicating significances

Package ggsign makes it easier to add significance brackets (no more photoshopping), it either computes p-values or takes them from your testing (and this is what you should always be doing!).

```
# ggsign ####
p <- round(
  wilcox.test(mtcars$mpg~mtcars$am, exact = FALSE)$p.value,
  5)
(plottemp <- ggplot(mtcars,aes(as.factor(am),mpg))+  
  geom_boxplot())
```

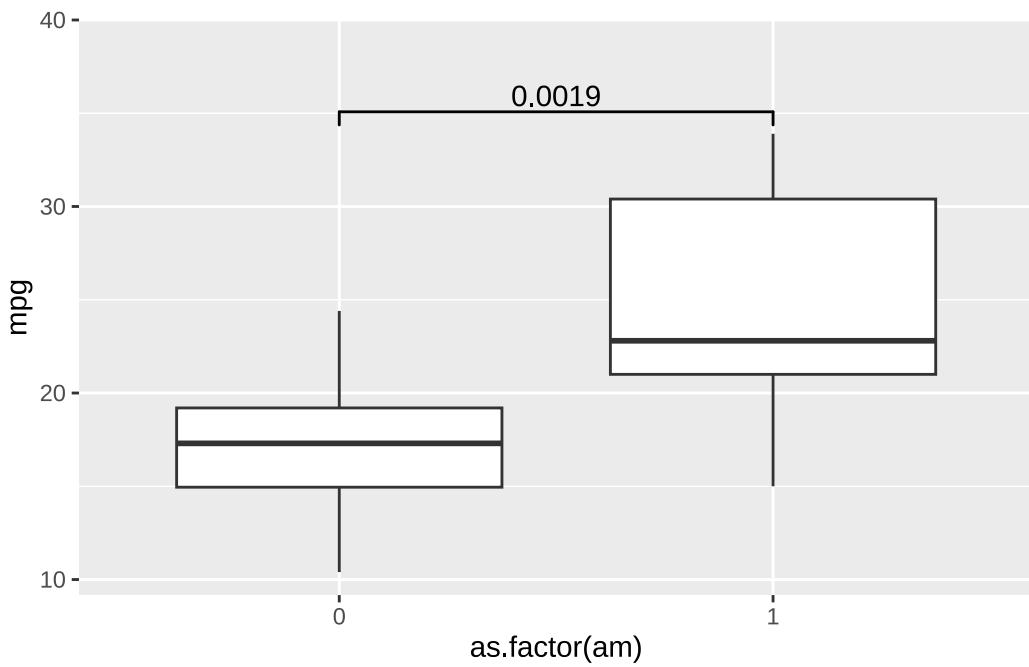


```
plottemp+geom_signif(  
  comparisons=list(c(1,2)),  
  # aes(y=0),  
  textsize = rel(10), vjust = .0,  
  #y_position=max(mtcars$mpg)+3,  
  # annotations=paste0("p = ", p),  
  annotations=markSign(p),  
  # annotations=p,  
  tip_length=.02)+  
  scale_y_continuous(expand = expansion(mult=c(0.05,.1)))
```



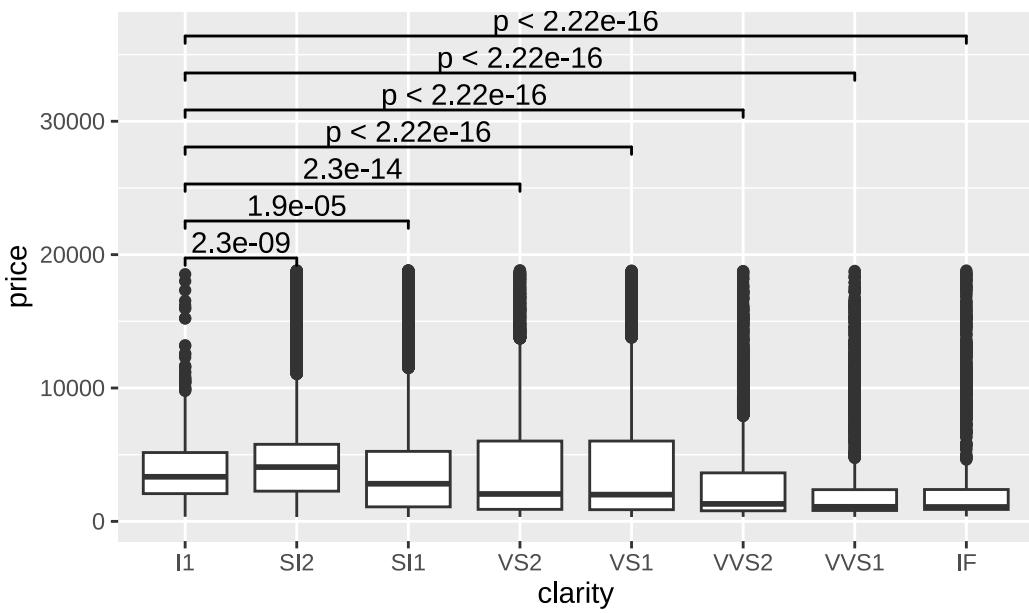
```
plottemp + geom_signif(  
  comparisons=list(1:2))+  
  scale_y_continuous(expand = expansion(mult=c(0.05,.2)))
```

Warning in wilcox.test.default(c(21.4, 18.7, 18.1, 14.3, 24.4, 22.8, 19.2, :  
kann bei Bindungen keinen exakten p-Wert Berechnen



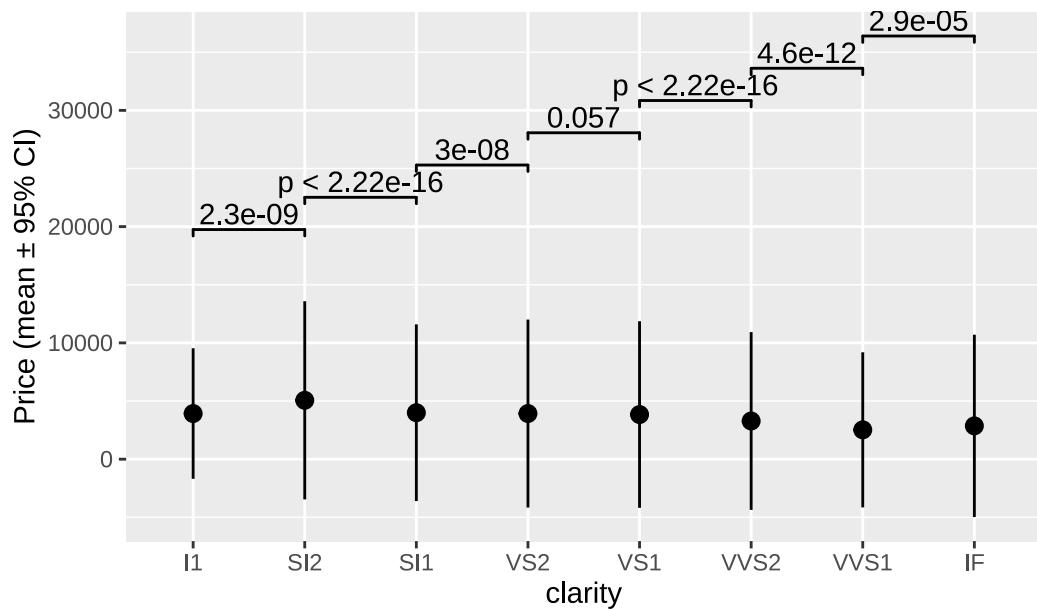
```
ggplot(diamonds,aes(clarity, price))+  
  geom_boxplot() +  
  # stat_summary(fun.data=mean_sdl)+  
  geom_signif(comparisons=list(c(1,2),c(1,3),c(1,4),c(1,5),  
                                c(1,6),c(1,7),c(1,8)),  
              step_increase = 0.15)+  
  ggtitle("nominal p-values!!")
```

nominal p-values!!



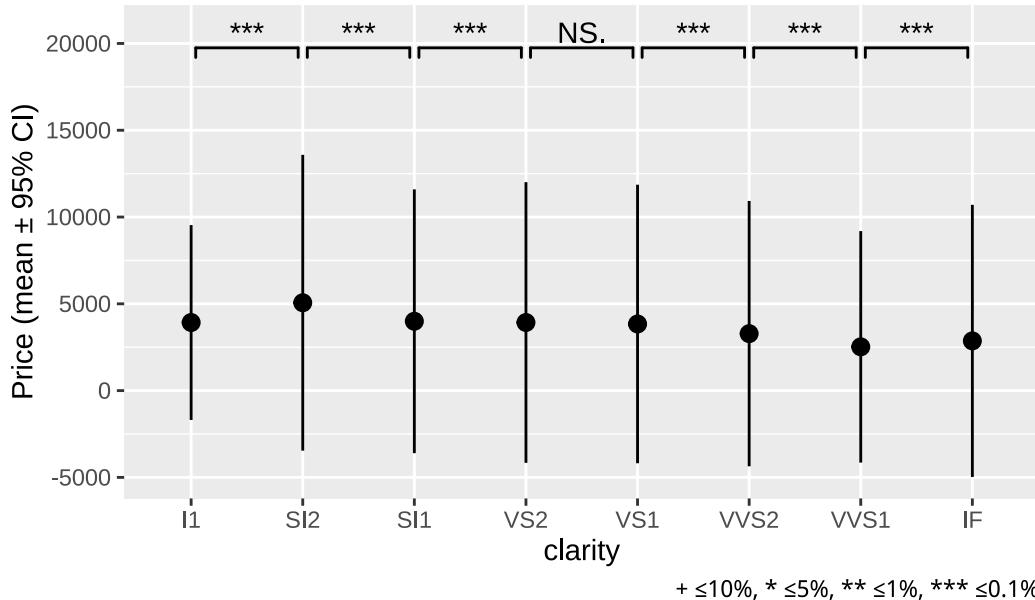
```
ggplot(diamonds,aes(clarity, price))+  
  stat_summary(fun.data=mean_sdl)+  
  ylab("Price (mean \u00b1 95% CI)")+  
  geom_signif(comparisons=list(c(1,2),c(2,3),c(3,4),c(4,5),  
                               c(5,6),c(6,7),c(7,8)),  
              step_increase = 0.15)+  
  ggtitle("nominal p-values!!")
```

## nominal p-values!!



```
ggplot(diamonds,aes(clarity, price))+  
  stat_summary(fun.data=mean_sdl)+  
  ylab("Price (mean \u00b1 95% CI)")+  
  geom_signif(comparisons=list(c(1,2),c(2,3),c(3,4),c(4,5),  
                               c(5,6),c(6,7),c(7,8)),  
              map_signif_level = TRUE,  
              extend_line = -.005)+  
  ggtitle("nominal p-values!!") +  
  scale_y_continuous(expand = expansion(mult=c(0.05,.1)))+  
  labs(caption = "+ \u226410%, * \u22645%, ** \u22641%, *** \u22640.1%")
```

nominal p-values!!

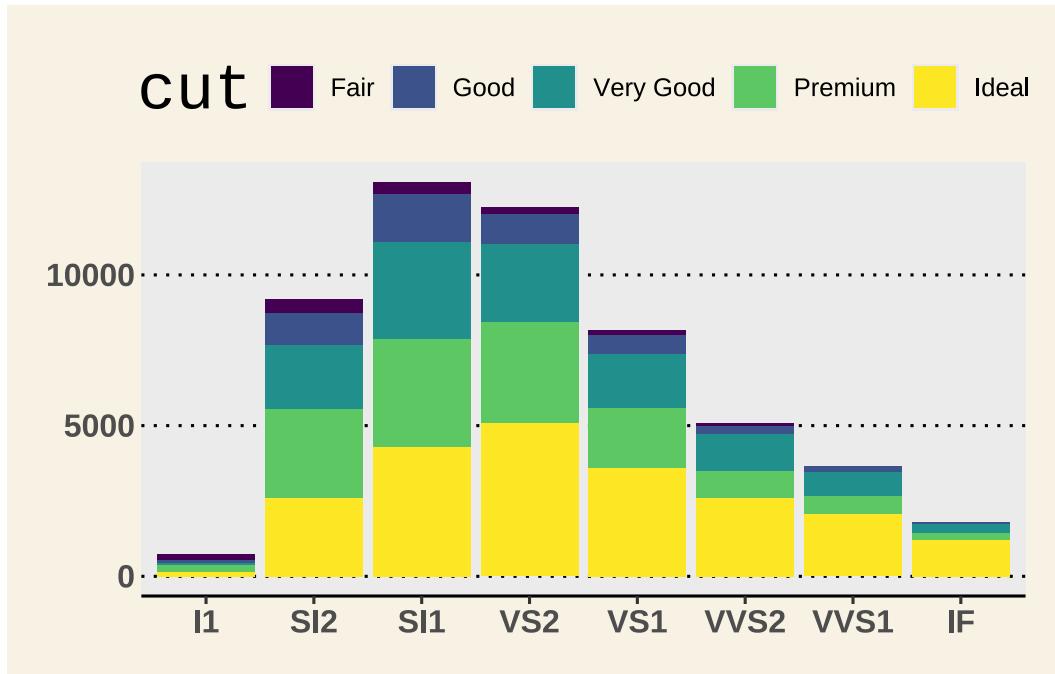


## 7.16 Theme definitions / changes

Themes define everything not-data-related in your figures, like margins, fonts, background color etc. There are many predefined themes, and all can be customized. You can change a theme for all plots to come (`theme_update()`) or just a single plot ( `+theme()`)

There is an informative blog post [Styling plots](#) that is a good starter.

```
old <- theme_set(theme_wsj())
ggplot(data=diamonds,aes(x=clarity,fill=cut))+  
  geom_bar()
```

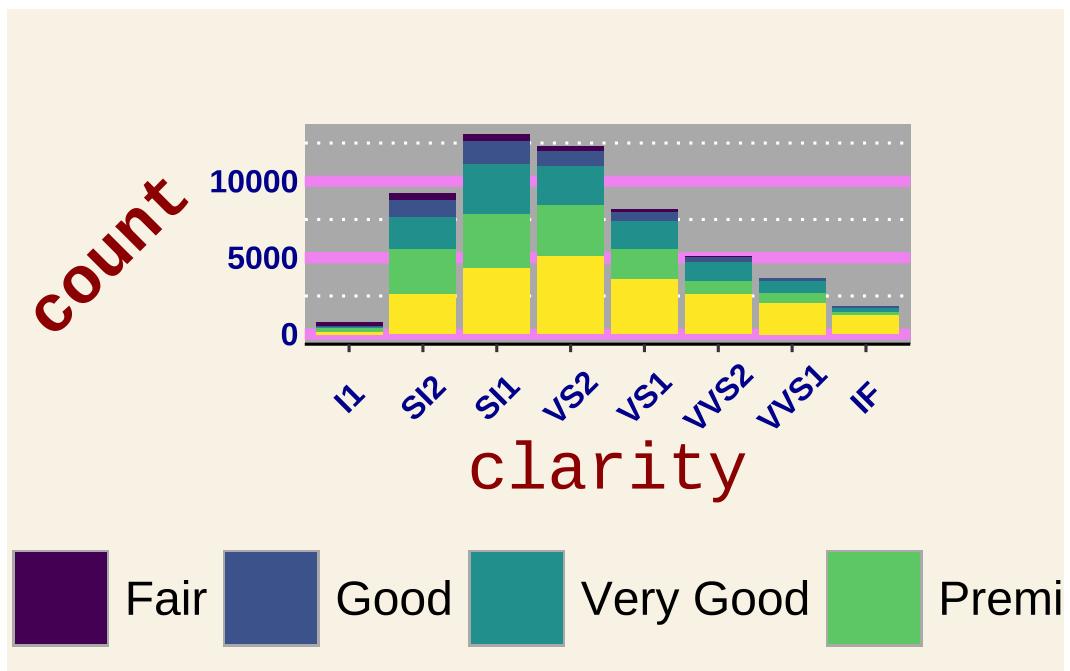


```

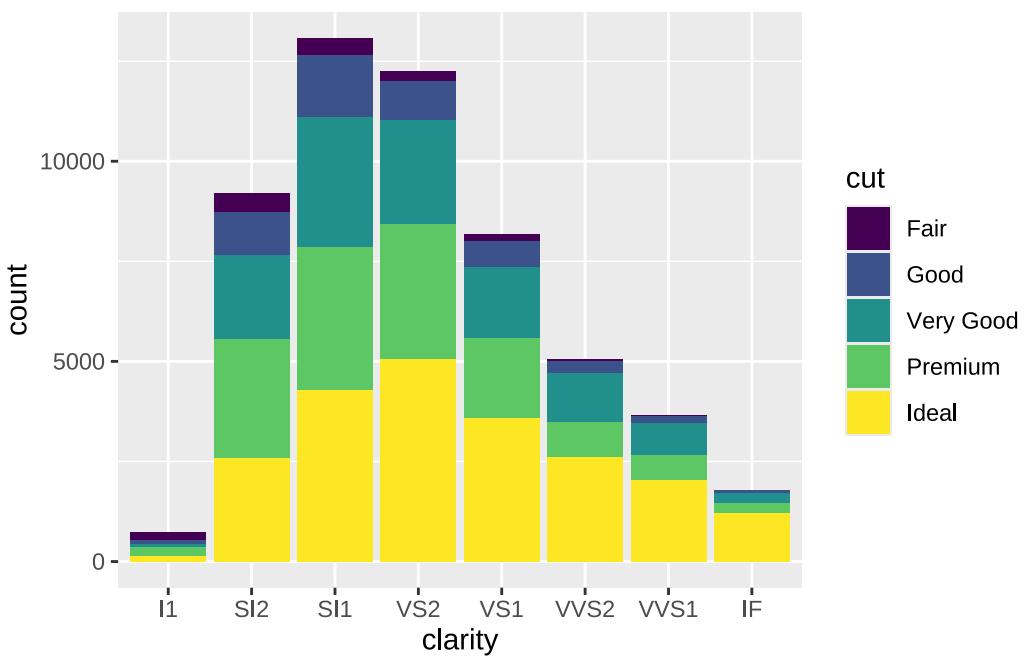
theme_update(legend.position="bottom",
             axis.text=element_text(colour = "darkblue",
                                    size=12),
             axis.text.x=element_text(vjust=0.5,angle=45,
                                      family="sans",
                                      face = "bold"),
             axis.title=element_text(size=25,
                                    color="darkred"),
             plot.margin=unit(c(3,4,0.5,0.3),"lines"),      #N,E,S,W
             axis.title.y=element_text(vjust=0.4,angle=45,
                                       face="bold"),
             legend.key.size=unit(2.5, "lines"),
             panel.background=element_rect(fill="darkgrey"),
             panel.grid.minor = element_line(colour="white"),
             panel.grid.major = element_line(
               linetype=1,
               color="violet", linewidth = 2),
             legend.text = element_text(size = 18),
             legend.title=element_text(size=30, color="pink"))

ggplot(data=diamonds,aes(x=clarity,fill=cut))+
  geom_bar()

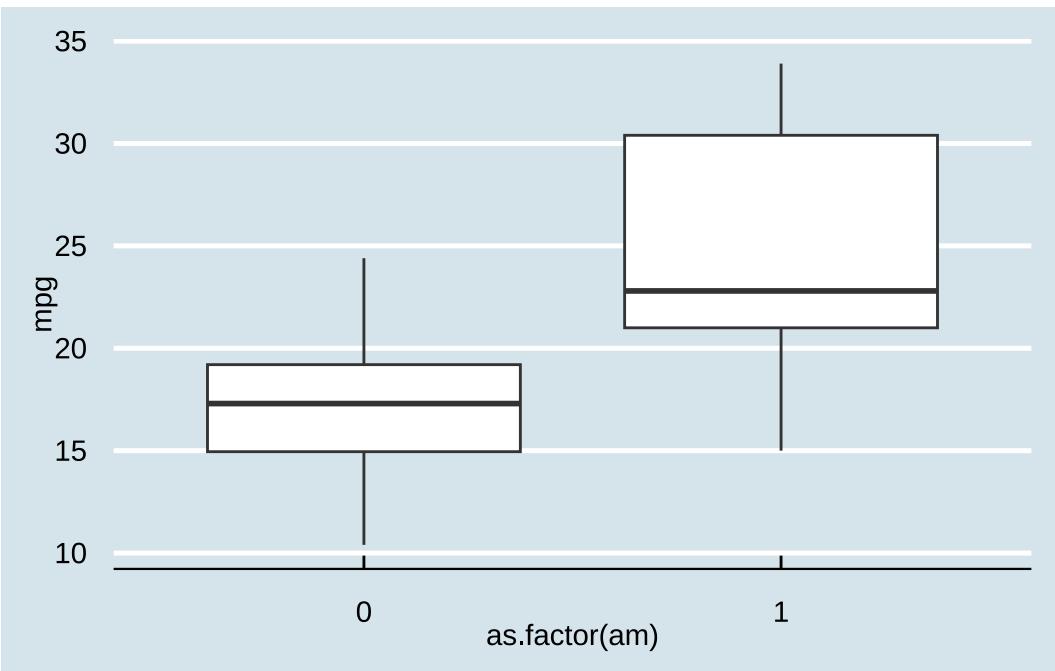
```



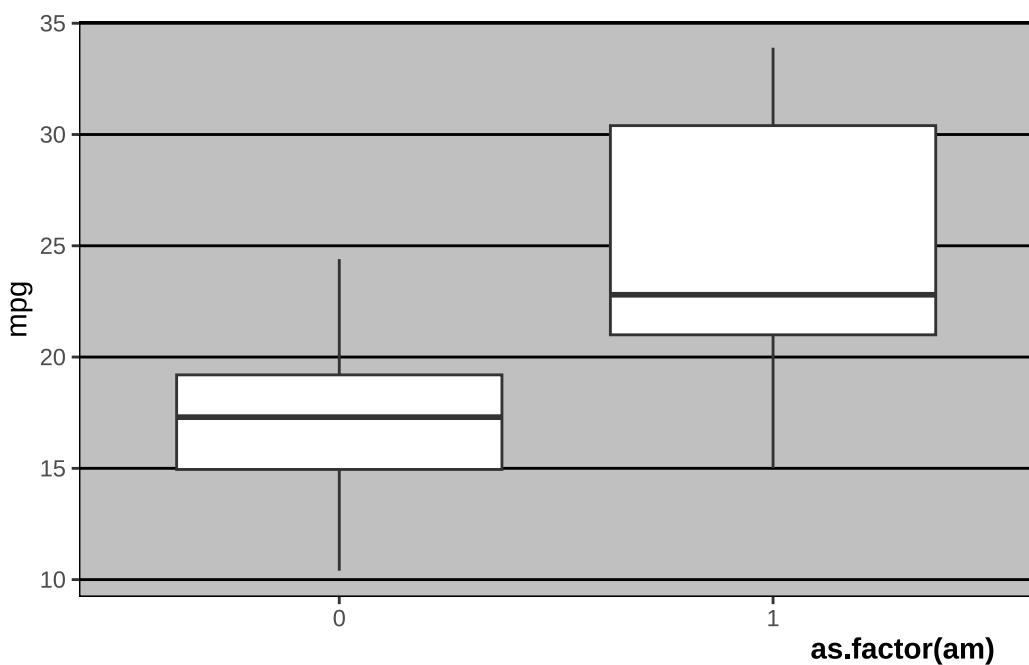
```
theme_set(theme_grey())
#theme_set(old)
ggplot(data=diamonds,aes(x=clarity,fill=cut))+  
  geom_bar()
```



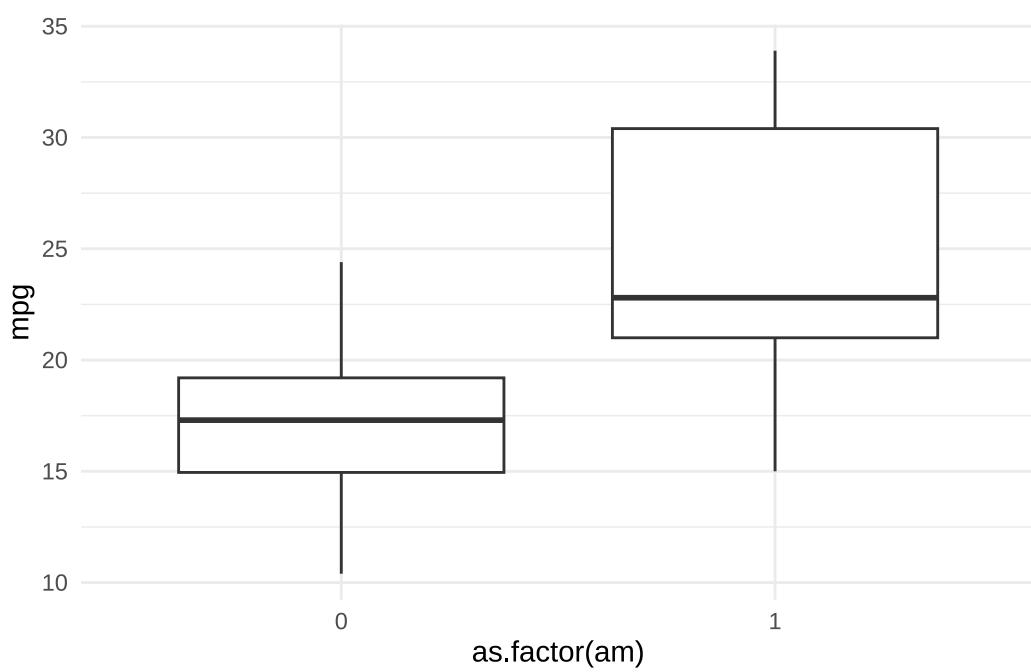
```
# ggthemes #####
plottemp+theme_economist()
```



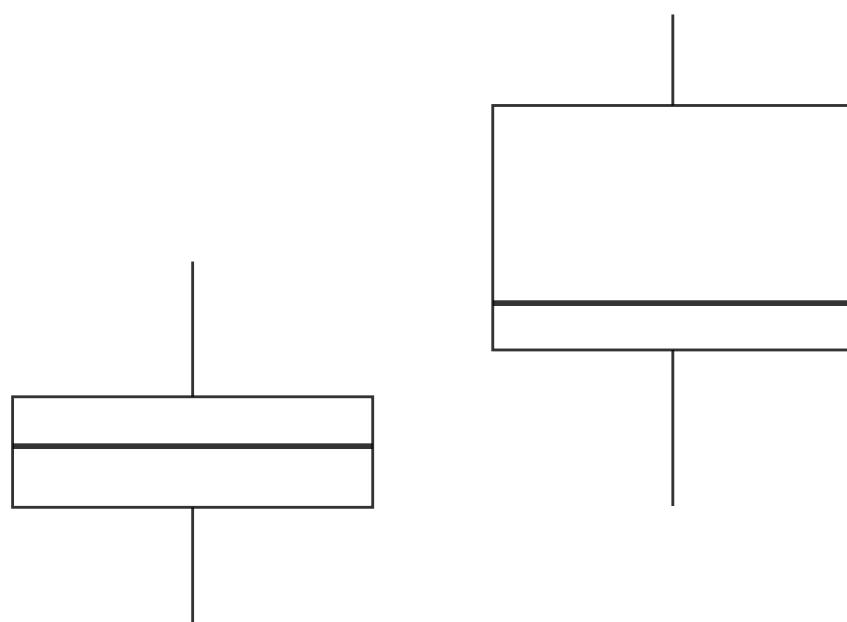
```
plottemp+theme_excel()+
  theme(axis.title.x = element_text(face="bold", hjust=0.95))
```



```
plottemp+theme_minimal()
```



```
plottemp+theme_void()
```



```
## install from Github
# devtools::install_github("jespermaag/gganatogram")
# gganatogram::gganatogram(
#   organism = "human", fill = "color",
#   data=tibble(organ = c("nerve", "brain"),
#   type = c("nervous system", "nervous system"),
#   colour = c("purple", "orange"),
#   value = c(1, 1)))+
#   theme_void()
# plottemp+theme_base()
# plottemp+theme_bw()+
#   theme(panel.grid.major.x = element_blank())

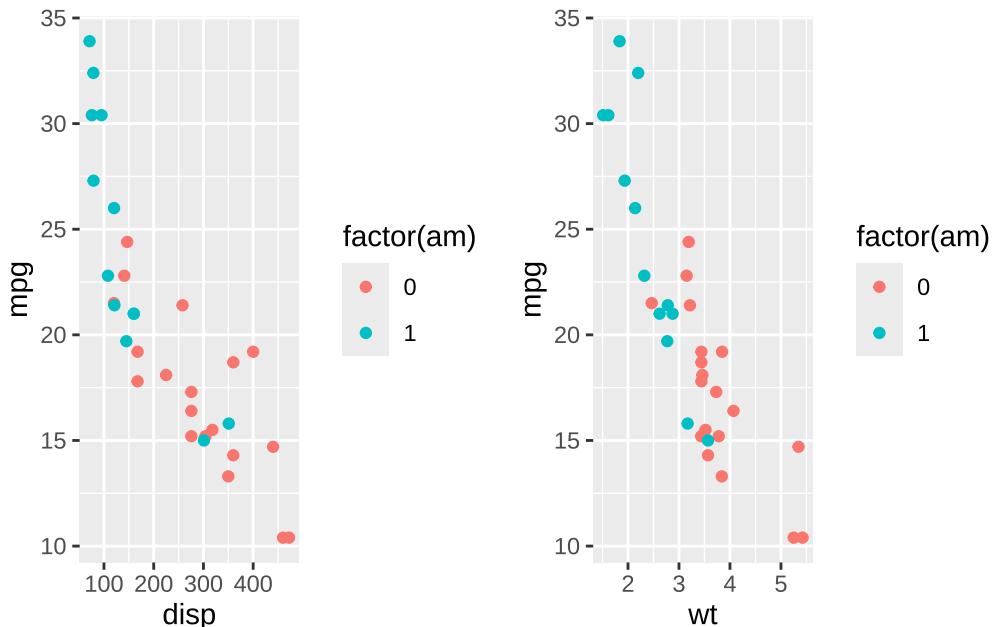
# https://www.data-imaginist.com/2019/a-flurry-of-facets/

# https://github.com/thomasp85/gganimate
```

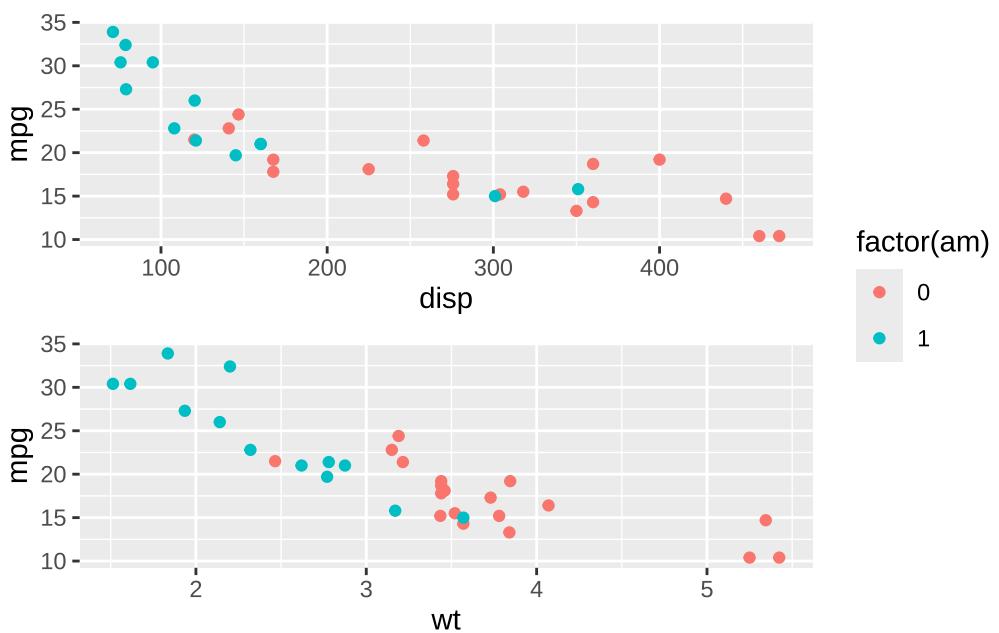
## 7.17 Combining figures with patchwork

More details on [github](#)

```
p1 <- ggplot(mtcars) +
  geom_point(aes(disp, mpg, color=factor(am)))
p2 <- ggplot(mtcars) +
  geom_point(aes(wt, mpg, color=factor(am)))
p1 | p2
```



```
(p1 / p2) +
  plot_layout(guides = "collect")
```



```
p3 <- ggplot(mtcars) +
  geom_smooth(aes(disp, qsec, color=factor(am)))
```

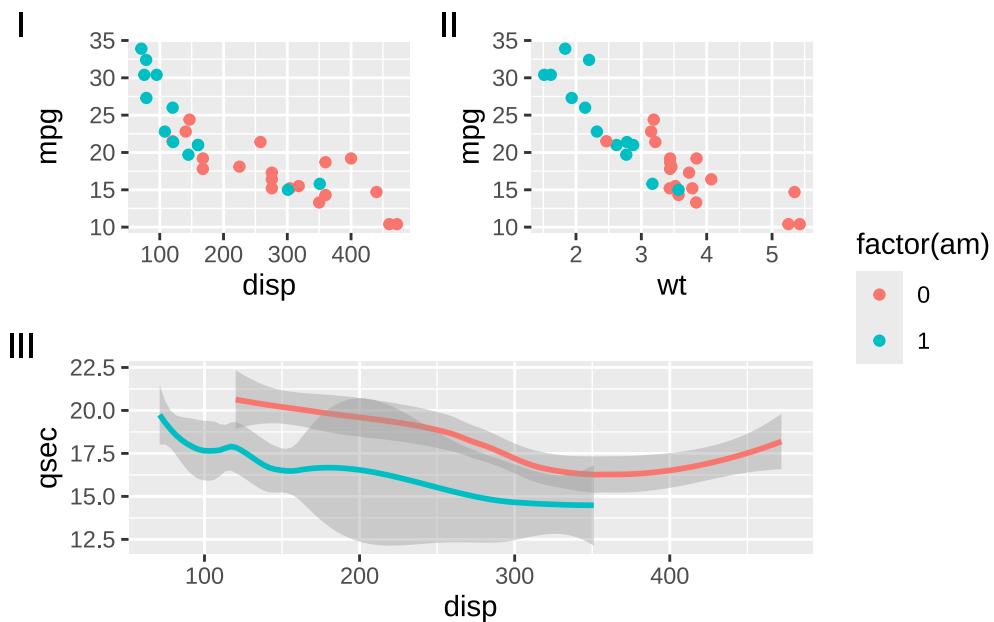
```

# p4 <- ggplot(mtcars) +
#   geom_bar(aes(factor(carb), fill=factor(am)))

(p1 | p2) /
  (p3 + guides(color = "none")) +
  plot_annotation(tag_levels = "I") +
  plot_layout(guides = "collect")

```

`geom\_smooth()` using method = 'loess' and formula = 'y ~ x'



## 7.18 ggplots in loops

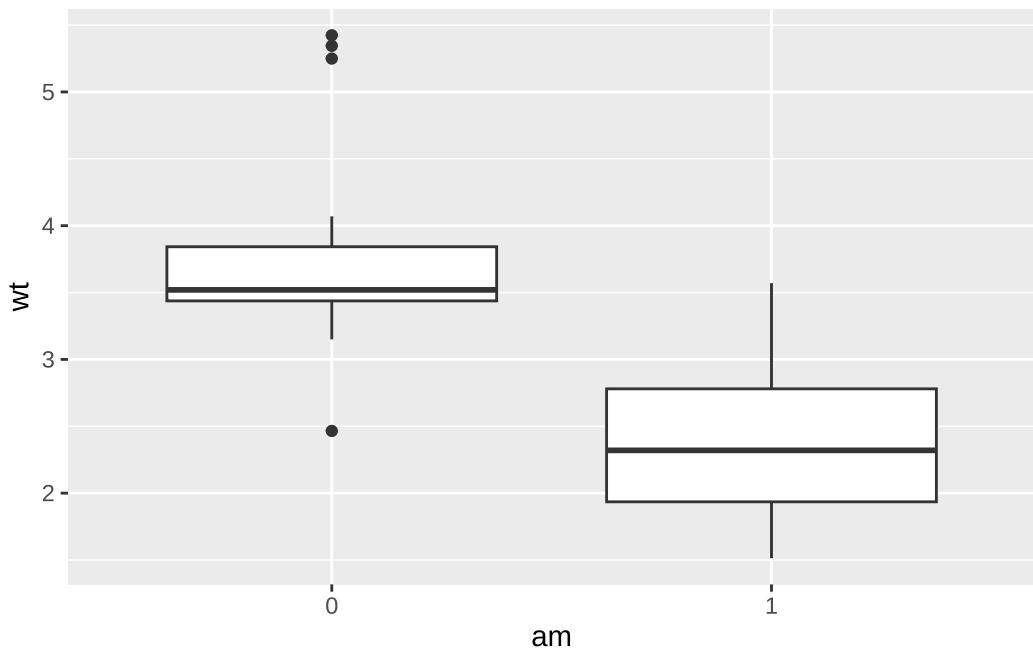
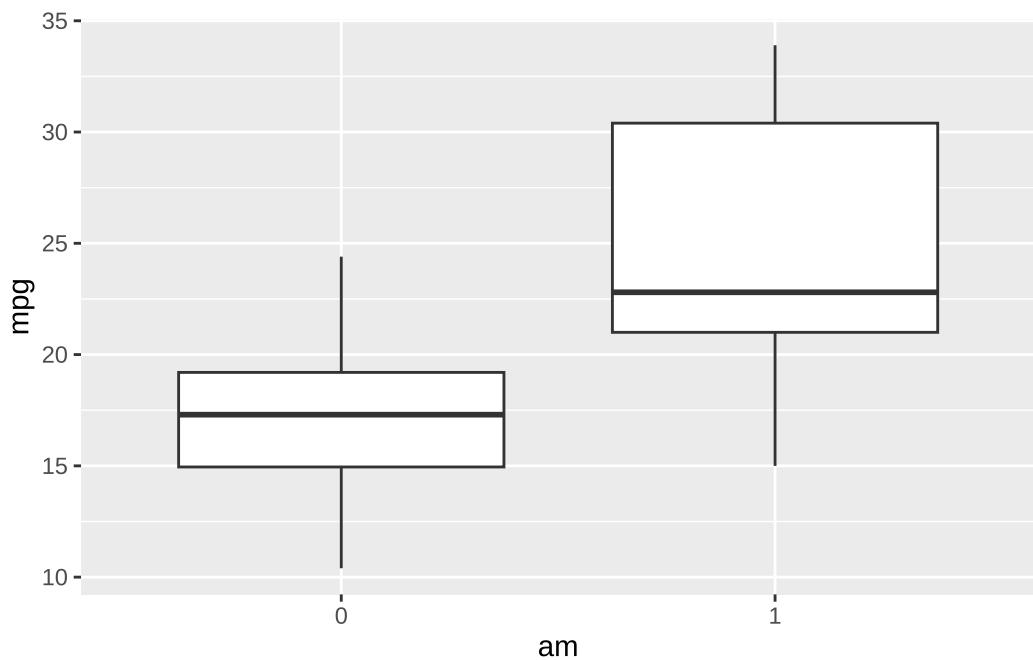
inside aes(), ggplot expects variables referring to columns in the data. In a loop, the loop index is often a character representation of a column name. So inside aes(), there is a variable containing a variable name. To make that work, we use .data as a reference to the data ggplot is working on, and the loop index inside double square brackets as its index:

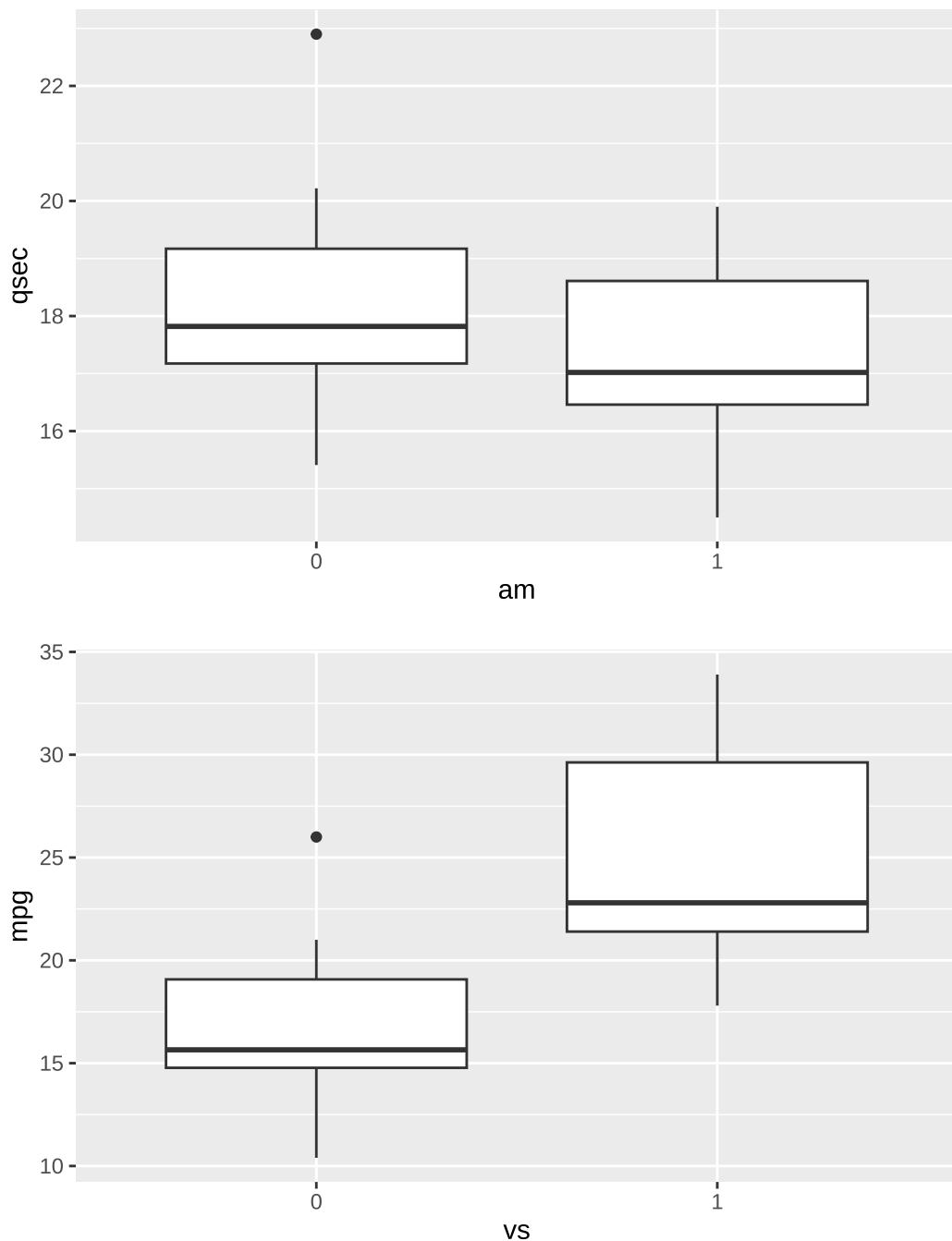
```

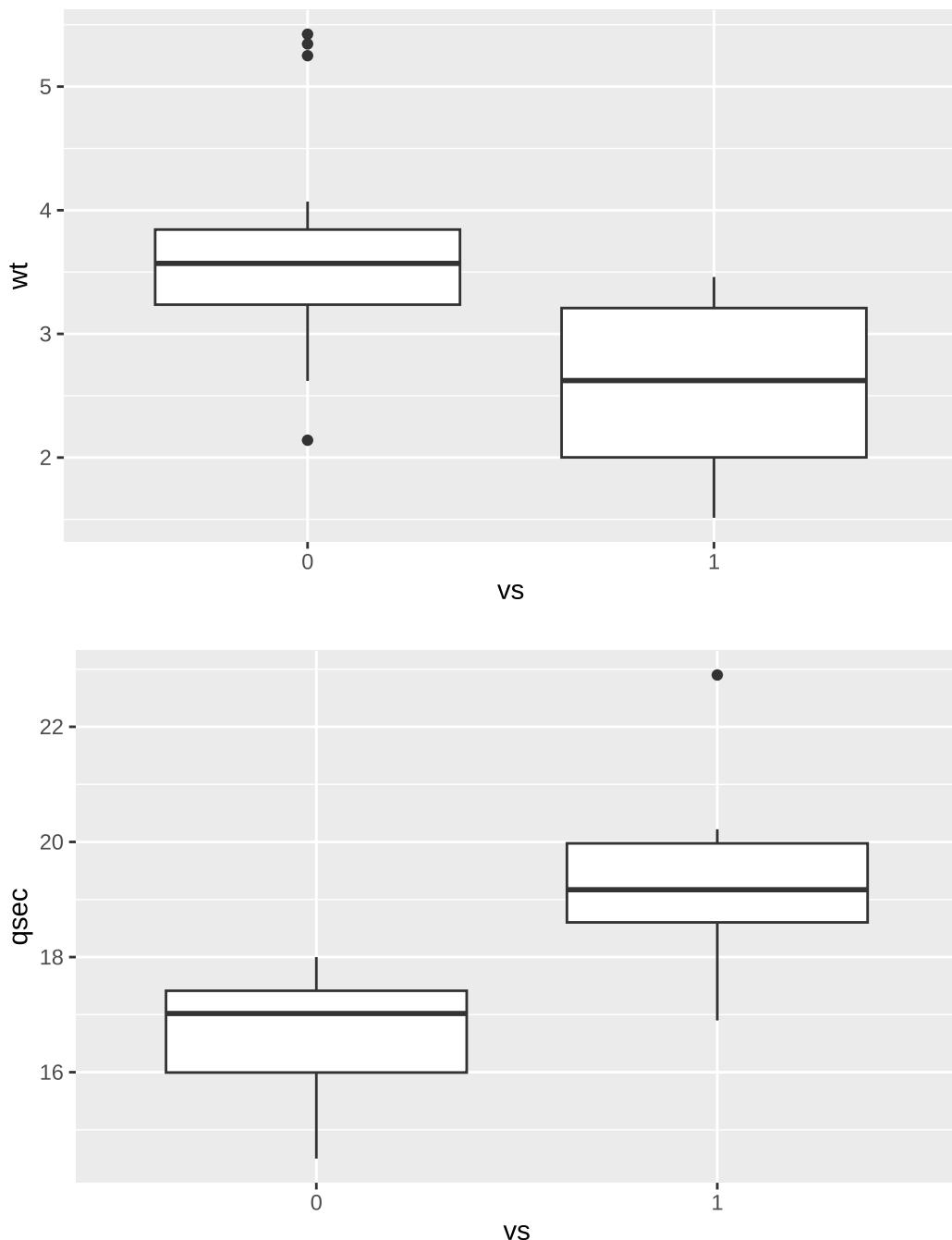
for(group_i in c("am", "vs")){
  for(var_i in c("mpg", "wt", "qsec")){
    plot_temp <-
      mutate(mtcars,
        am = factor(am),
        vs = factor(vs)) |>

```

```
    ggplot(aes(x = .data[[group_i]], y = .data[[var_i]]))+
      geom_boxplot()
  print(plot_temp)
}
}
```





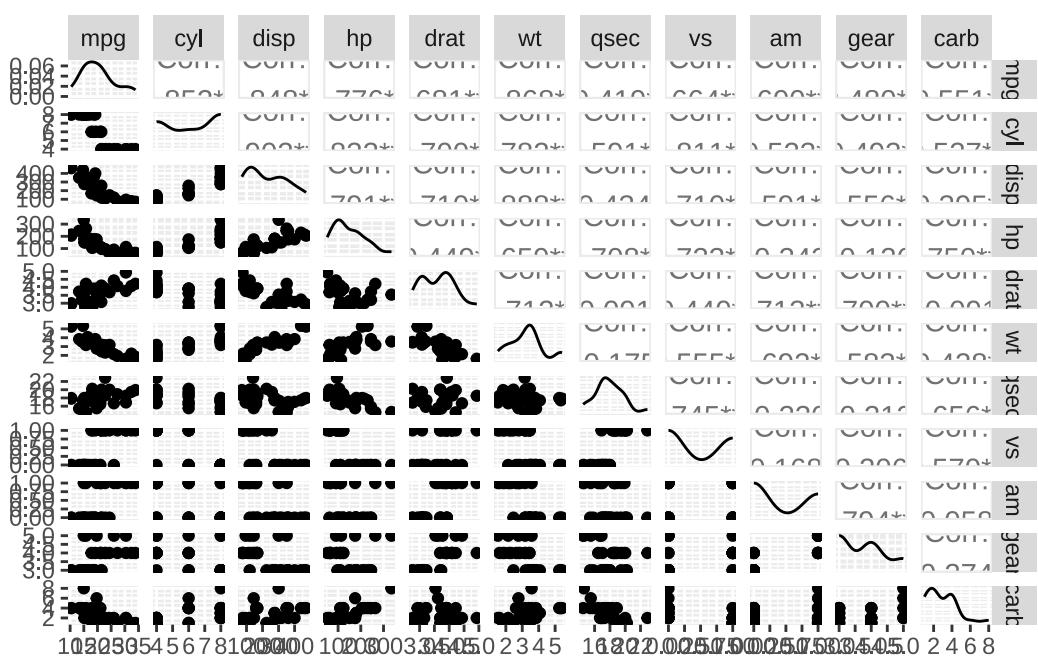


The function `aes_string()`, which expected a character representation instead of a variable name, is deprecated, but may still be suggested by chatbots

[more facets...](#)

[animations...](#)

```
GGally::ggpairs(mtcars)
```



# 8 Grouping of variables by type / distribution / use

```
pacman::p_load(conflicted, wrappedtools, tidyverse, here)
conflicts_prefer(dplyr::filter)
```

[conflicted] Will prefer dplyr::filter over any other package.

```
rawdata <- readRDS(here('data/rawdata.rds'))
```

## 8.1 Test for Normal distribution

### 8.1.1 Testing a single variable

Before computing some test-statistics, a graphical exploration should be done by e.g. density plots.

There are a number of tests for Normal distribution, all testing the Null hypothesis of data coming from a population with Normal distribution. So small p-values lead to rejection of the Null and indicate deviation from normality. Kolmogorov-Smirnov-test (for larger sample sizes) and Shapiro-Wilk-test (for smaller samples) will be used as examples.

Other tests would be e.g. Anderson-Darling, and the Cramer-von Mises test, see package `nortest`.

```
ks.test(x = rawdata$`Size (cm)`,
        "pnorm",
        mean=mean(rawdata$`Size (cm)`),
        na.rm = TRUE),
        sd=sd(rawdata$`Size (cm)`),
        na.rm = TRUE))
```

Warning in ks.test.default(x = rawdata\$`Size (cm)` , "pnorm", mean =  
mean(rawdata\$`Size (cm)` , : ties should not be present for the one-sample  
Kolmogorov-Smirnov test

Asymptotic one-sample Kolmogorov-Smirnov test

```
data: rawdata$`Size (cm)`  
D = 0.13284, p-value = 0.7064  
alternative hypothesis: two-sided
```

```
ksnormal(rawdata$`Size (cm)`, lillie = FALSE)
```

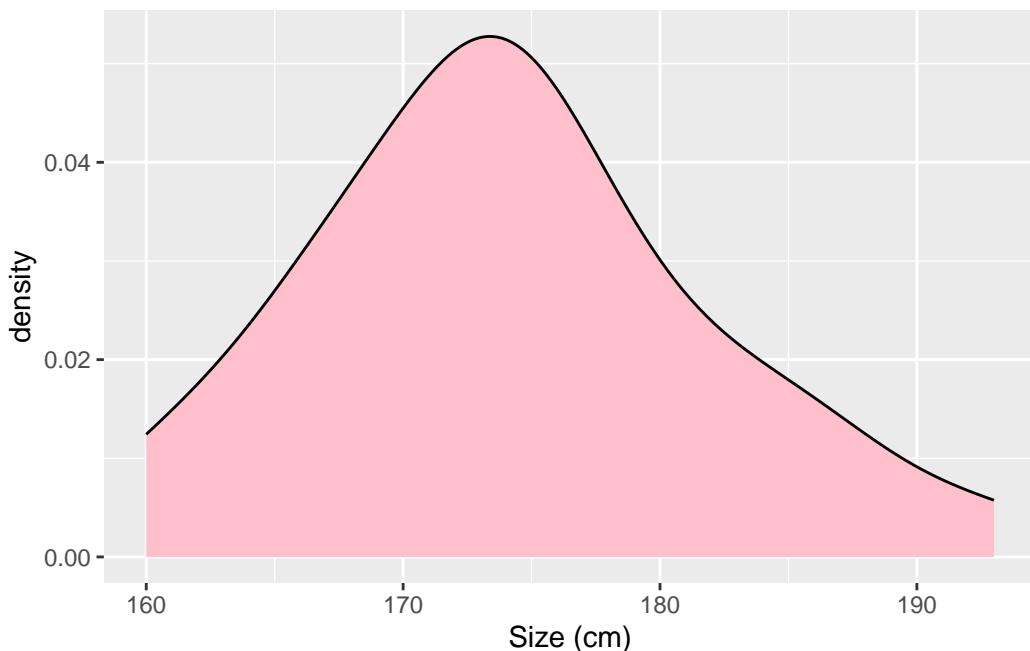
```
p_Normal_KS  
0.7063825
```

```
shapiro.test(rawdata$`Size (cm)`)
```

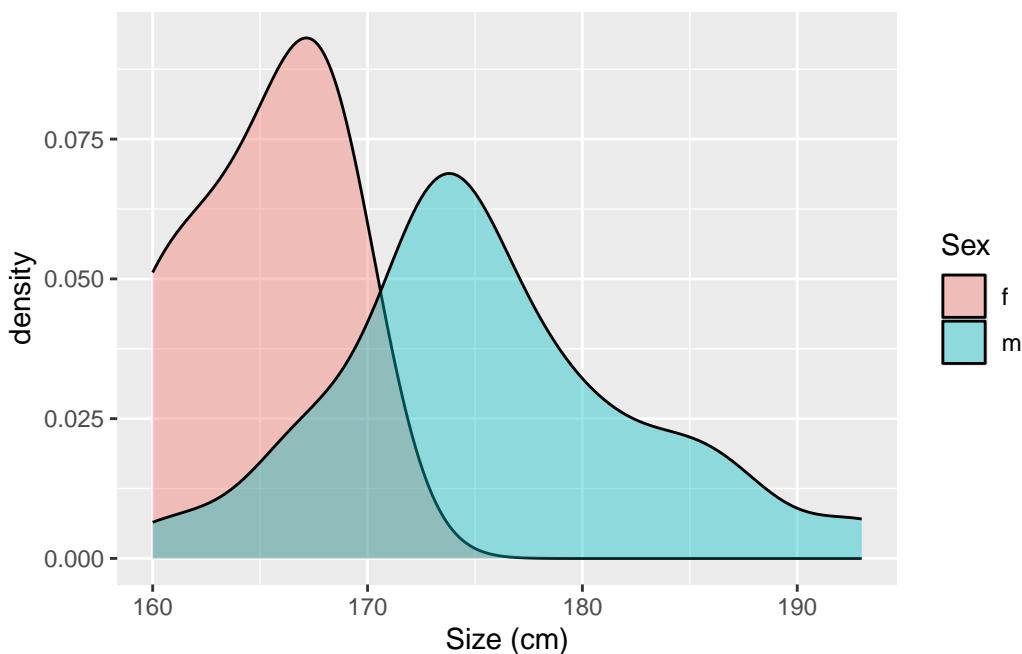
Shapiro-Wilk normality test

```
data: rawdata$`Size (cm)`  
W = 0.9766, p-value = 0.7627
```

```
ggplot(rawdata, aes(x = `Size (cm)`)) +  
  geom_density(fill="pink")
```



```
ggplot(rawdata,aes(x = `Size (cm)`,fill=Sex))+  
  geom_density(alpha=.4)
```



If severe group difference can be expected (case/control, sex ...), exploration and analyses should be done in subgroups.

```
rawdata |> filter(Sex=="m") |>  
  pull(`Size (cm)`) |>  
  ksnormal()
```

```
p_Normal_Lilliefors  
0.1180981
```

```
rawdata |>  
  group_by(Sex) |>  
  summarize(  
    n=n(),  
    p_KS = ksnormal(`Size (cm)`,lillie = FALSE),  
    `pGauss (Shapiro)` = shapiro.test(`Size (cm)`) |>  
    pluck("p.value"))
```

```
# A tibble: 2 x 4  
Sex      n  p_KS `pGauss (Shapiro)`  
<chr> <int> <dbl>          <dbl>
```

```

1 f          4 0.905          0.272
2 m          24 0.575         0.638

```

### 8.1.2 Testing several variables

To explore larger data sets, it may be useful to test all numerical variables for normality, this can be done in a loop or with the across-function. As a start for the loop-solution we can get the names and positions for all (or selected) numerical variables with the ColSeeker-function from wrappedtools.

```

numvars <-
  ColSeeker(data = rawdata, # can be omitted, as it is the default
            varclass = "numeric")
numvars$index

[1] 1 2 3 4 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

head(numvars$names)

[1] "Randomcode"      "Included"        "Finalized"       "Testmedication"
[5] "Size (cm)"       "Weight (kg)"

numvars$count

[1] 23

```

Loops can be created with either a numeric counter-like index or content-based index.

Loop Version 1:

```

## result table v1, pre-filled
resulttable1 <- tibble(
  Variables=numvars$names,
  pKS=NA_real_,
  pSh=NA_real_
)
## loop version 1
for(var_i in seq_len(numvars$count)){
  resulttable1$pKS[var_i] <-
    ksnormal(rawdata[[numvars$names[var_i]]])
  resulttable1$pSh[var_i] <-
    shapiro.test(rawdata |>

```

```

        pull(numvars$names[var_i]))$p.value
}
head(resulttable1)

```

```

# A tibble: 6 x 3
  Variables      pKS      pSh
  <chr>       <dbl>    <dbl>
1 Randomcode  9.74e- 1 3.10e- 1
2 Included     4.88e-24 2.25e-11
3 Finalized   1.25e-21 1.86e- 9
4 Testmedication 6.96e- 9 4.31e- 7
5 Size (cm)   2.33e- 1 7.63e- 1
6 Weight (kg) 2.98e- 1 8.96e- 2

```

```

resulttable1 |>
  mutate(pKS=formatP(pKS,ndigits=5),
         pSh=formatP(pSh,mark = T))

```

```

# A tibble: 23 x 3
  Variables      pKS      pSh
  <chr>       <chr>    <chr>
1 Randomcode  0.97369 0.310 n.s.
2 Included     0.00001 0.001 *** 
3 Finalized   0.00001 0.001 *** 
4 Testmedication 0.00001 0.001 *** 
5 Size (cm)   0.23305 0.763 n.s.
6 Weight (kg) 0.29772 0.090 +
7 sysBP V0    0.46905 0.256 n.s.
8 diaBP V0    0.11863 0.095 +
9 Lv Edv Mri  0.25374 0.167 n.s.
10 Lv Esv Mri 0.00288 0.001 ***#
# i 13 more rows

```

Loop Version 2:

```

## result table v2, just structure
resulttable2 <- tibble(Measures=NA_character_,
                      pKS_Placebo=NA_character_,
                      pKS_Verum=NA_character_,
                      .rows = 0)
for(var_i in numvars$names){
  ks_tmp <- by(data = rawdata[[var_i]],
                INDICES=rawdata$Testmedication,

```

```

        FUN=ksnormal,
        lillie=FALSE)
resulttable2 <-add_row(resulttable2,
                        Measures=var_i,
                        pKS_Placebo=ks_tmp[[1]] |>
                          formatP(), # added rounding/formatting
                        pKS_Verum=ks_tmp[[2]] |>
                          formatP())
}
head(resulttable2)

```

```

# A tibble: 6 x 3
  Measures     pKS_Placebo pKS_Verum
  <chr>       <chr>      <chr>
1 Randomcode   0.994      0.996
2 Included     0.001      0.001
3 Finalized    0.003      0.001
4 Testmedication 0.001    0.001
5 Size (cm)    0.955      0.964
6 Weight (kg)  0.553      0.793

```

across() - Version:

`across()` in R (from `dplyr`) is a powerful tool that lets you apply the same operation to multiple columns in your data frame, similar to using loops or list comprehensions with Pandas DataFrames in Python. `across()` (sort of) loops over variables / columns and applies function(s), it can be used inside summarize, mutate, filter.

```

resulttable1a <-
  rawdata |>
  summarize(across(.cols=all_of(numvars$names[-(1:4)]),
                  .fns = list(
                    pKS=~ksnormal(.x) |>
                      formatP(mark = TRUE),
                    pSh=~shapiro.test(.x) |>
                      pluck("p.value") |>
                      formatP(mark = TRUE)))) |>
#change output structure to long form
pivot_longer(everything(),
             names_to=c("Variable",".value"),
             names_sep = "_")
# pivot_longer(everything(),
#             #           names_to=c("Variable","test"), #.variable
#             #           names_sep = "_") |>
# pivot_wider(names_from=test, values_from=value)

```

```
head(resulttable1a)
```

```
# A tibble: 6 x 3
  Variable      pKS      pSh
  <chr>        <chr>    <chr>
1 Size (cm)   0.233 n.s. 0.763 n.s.
2 Weight (kg) 0.298 n.s. 0.090 +
3 sysBP V0    0.469 n.s. 0.256 n.s.
4 diaBP V0    0.119 n.s. 0.095 +
5 Lv Edv Mri  0.254 n.s. 0.167 n.s.
6 Lv Esv Mri  0.003 **  0.001 ***
```

```
head(resulttable1)
```

```
# A tibble: 6 x 3
  Variables      pKS      pSh
  <chr>        <dbl>    <dbl>
1 Randomcode   9.74e- 1 3.10e- 1
2 Included     4.88e-24 2.25e-11
3 Finalized   1.25e-21 1.86e- 9
4 Testmedication 6.96e- 9 4.31e- 7
5 Size (cm)   2.33e- 1 7.63e- 1
6 Weight (kg)  2.98e- 1 8.96e- 2
```

```
resulttable2a <-
  rawdata |>
  mutate(Testmedication=factor(Testmedication,
                                levels=c(0,1),
                                labels=c('Placebo','Verum'))) |>
  # mutate(Testmedication=case_match(Testmedication,
  #                                   0~"Placebo",
  #                                   1~"Verum")) |>
  group_by(Testmedication) |>
  summarize(across(all_of(numvars$names[-(1:4)]),
                  .fns = ~ksnormal(.x) |>
                  formatP(mark = TRUE))) |>
  pivot_longer(-Testmedication,
               names_to="Measure") |>
  pivot_wider(names_from=Testmedication,
              values_from=value)
head(resulttable2a)
```

```
# A tibble: 6 x 3
  Measure     Placebo     Verum
  <chr>       <chr>       <chr>
1 Size (cm)   0.678 n.s.  0.715 n.s.
2 Weight (kg) 0.087 +    0.303 n.s.
3 sysBP V0    0.085 +    0.277 n.s.
4 diaBP V0    0.143 n.s.  0.628 n.s.
5 Lv Edv Mri  0.985 n.s.  0.433 n.s.
6 Lv Esv Mri  0.035 *    0.004 **
```

```
head(resulttable2)
```

```
# A tibble: 6 x 3
  Measures      pKS_Placebo pKS_Verum
  <chr>        <chr>        <chr>
1 Randomcode   0.994        0.996
2 Included     0.001        0.001
3 Finalized   0.003        0.001
4 Testmedication 0.001      0.001
5 Size (cm)   0.955        0.964
6 Weight (kg) 0.553        0.793
```

```
resulttable3 <-
  rawdata |>
  group_by(Testmedication) |>
  summarize(across(all_of(numvars$names[-(1:4)]),
    .fns = list(
      Mean=~mean(.x, na.rm=TRUE) |>
        roundR(5),
      Median=~median(.x, na.rm=TRUE) |>
        roundR(5),
      pKS=~ksnormal(.x) |>
        formatP(mark = TRUE),
      pSh=~shapiro.test(.x) |>
        pluck("p.value") |>
        formatP(mark = TRUE)))) |>
  pivot_longer(-Testmedication,
    names_to=c("Variable","test"), #Variable,.value
    names_sep = "_") |>
  pivot_wider(names_from=test, values_from=value) |>
  arrange(Variable)
head(resulttable3)
```

```
# A tibble: 6 x 6
  Testmedication Variable      Mean   Median pKS      pSh
  <dbl> <chr>        <chr>  <chr>  <chr>      <chr>
1          0 Age       60.429 64.000 0.057 + 0.425 n.s.
2          1 Age       60.429 60.000 0.604 n.s. 0.282 n.s.
3          0 BMI       30.244 29.246 0.680 n.s. 0.430 n.s.
4          1 BMI       28.016 27.484 0.880 n.s. 0.862 n.s.
5          0 Ferritin Lab 258.21 220.00 0.112 n.s. 0.176 n.s.
6          1 Ferritin Lab 305.23 222.00 0.001 *** 0.003 **
```

```
rm(numvars)
```

## 8.2 Exercise: Distribution of penguin measures

- Using the penguin data, testing for which numerical measures a test for a *gaussian* distribution is meaningful
- For those measures, test Normality in the total sample as well as for subgroups defined by species and sex
  1. single measure, all penguins, plot and test
  2. single measure, by species/sex, plot and test
  3. all measures within species, plot and test
  4. all measures within species and sex, plot and test

## 8.3 Picking column names and positions

Based on data inspection, testing, and background knowledge, variables can be sorted into scale levels:

```
gaussvars <- ColSeeker(
  data = rawdata,    # can be omitted, as it is the default
  namepattern = c("si", "we", "BMI", "BP", "mri"),
  casesensitive = FALSE)

ordvars <- ColSeeker(data = rawdata,
                      namepattern = c("Age", "Lab"))

factvars <- ColSeeker(data = rawdata,
                      namepattern = c("Sex", "med", "NYHA"),
                      returnclass = TRUE)
```

```
rawdata <- mutate(rawdata,
                  across(all_of(factvars$names),
                        ~factor(.x)))
```

To make data accessible for other scripts, data can be saved:

```
save(rawdata, list = ls(pattern = "vars"),
      file = here("data/bookdata1.RData"))
```

# 9 Descriptive statistics

Descriptive statistics are used to summarize and organize data in a manner that is meaningful and useful. They provide simple summaries about the sample and the measures, such as mean, median, standard deviation, or frequencies. Furthermore, they allow for the presentation of quantitative descriptions in a manageable form, aiding in understanding the data distribution and central tendency. For group comparisons, they will inform about direction and magnitude of differences.

## 9.1 Typical descriptives

- `mean()` / `sd()` / `meansd()`
- `median()` / `quantile()` / `median_quart()`
- `table()` / `prop.table()` / `cat_desc_stats()`

## 9.2 Reading in data

```
pacman::p_load(conflicted,tidyverse,wrappedtools,
                flextable, here)
set_flextable_defaults(font.size = 9,
                       padding.bottom = 1,
                       padding.top = 3,
                       padding.left = 3,
                       padding.right = 4
)
load(here("data/bookdata1.RData"))
```

## 9.3 Graphical exploration should start before descriptive statistics

```
ggplot(rawdata,aes(`sysBP V0`, `diaBP V0`))+  
  geom_point() +  
  geom_smooth(se=F) +  
  geom_smooth(method="lm",color="red",  
             fill="gold", alpha=.15)
```

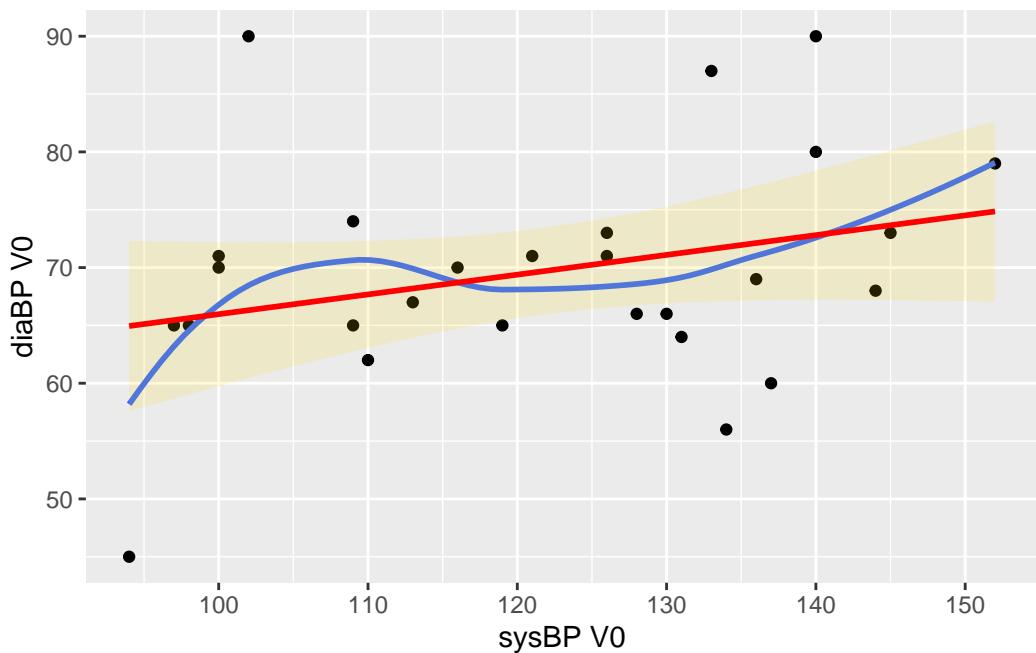
```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
Warning: Removed 1 row containing non-finite outside the scale range  
(`stat_smooth()`).
```

```
`geom_smooth()` using formula = 'y ~ x'
```

```
Warning: Removed 1 row containing non-finite outside the scale range  
(`stat_smooth()`).
```

```
Warning: Removed 1 row containing missing values or values outside the scale range  
(`geom_point()`).
```



## 9.4 Gaussian variables

## 9.5 A little theory

---

**Sample size n:** per variable, if there are NAs

**Mean:** central tendency, the expected  $\frac{\sum x}{n}$  typical value

---

<b>Variance:</b> measure for variability/heterogeneity of data	$\frac{\sum (x - \text{mean})^2}{n-1}$
<b>Standard deviation SD:</b> the <i>typical</i> weighted deviation from the mean	$\sqrt{\text{Var}}$

---



---

<b>Standard error of the mean SEM:</b> how reliable is the mean <i>estimate</i> , what would be the expected SD of means from repeated experiments?	$\frac{SD}{\sqrt{n}}$
<b>Median:</b> Split between lower/upper 50% of data	
<b>Quartiles:</b> Split at 25%/50%/75% of data (more general: <b>Quantiles</b> , e.g. <b>Percentiles</b> ), used in boxplot	various computational approaches

---

### 9.5.1 Simple function calls

```
(mean_size <- mean(rawdata$`Size (cm)`))
```

```
[1] 174.1071
```

```
(sd_size <- sd(rawdata$`Size (cm)`))
```

```
[1] 7.771454
```

```
min(rawdata$`Size (cm)`)
```

```
[1] 160
```

```
SEM(rawdata$`Size (cm)`)
```

```
[1] 1.468667
```

### 9.5.2 Combined reporting

For publishable tables you should round the numbers to a reasonable number of digits. Function `roundR()` is more flexible than base `round()`, as it determines the number of digits necessary to obtain the desired precision. The `level` argument allows for rounding to a specific number of non-zero digits. The `.german` argument changes the decimal point to a comma.

```
round(mean_size,digits = 2)
```

```
[1] 174.11
```

```
roundR(mean_size,level = 2)
```

```
[1] "174"
```

Usually mean and sd are reported together, function `meansd()` computes, rounds, and pastes the statistics in one go, arguments allow for flexible reporting:

```
meansd(rawdata$`Size (cm)`, roundDig = 4,  
       range = TRUE, add_n = TRUE)
```

```
[1] "174.1 ± 7.8 [160.0 -> 193.0] [n = 28]"
```

```
meansd(rawdata$`sysBP V0`, roundDig = 4,  
       range = TRUE,  
       add_n = TRUE, .german = TRUE)
```

```
[1] "121,9 ± 16,9 [94,0 -> 152,0] [n = 27]"
```

```
meanse(rawdata$`Size (cm)`, roundDig = 4)
```

```
[1] "174.1 ± 1.5"
```

## 9.6 Ordinal variables

Mean and SD describe symmetric distributions, but are not appropriate for ordinal data. For non-gaussian measurements, median and quartiles are more appropriate. When the distribution is known (e.g. Poisson for count data, other descriptives like lambda may be more informative).

```
median(rawdata$`Size (cm)`)
```

```
[1] 173.5
```

```
quantile(rawdata$`Size (cm)`,probs = c(.25,.75))
```

```
25%      75%
168.00 178.25
```

```
median_quart(rawdata$`Size (cm)`)
```

```
[1] "174 (168/179)"
```

```
median_quart(rawdata$Age,range = T)
```

```
[1] "62 (53/67)  [43 -> 74]"
```

Median and quartiles are sometimes the better choice even when assuming a Normal distribution, if there are outliers.



## 9.7 Categorical variables

```
table(rawdata$Sex, useNA = "a")
```

```
f      m <NA>
4     24     0
```

```
sex_count <- table(rawdata$Sex, useNA = "ifany")
table(rawdata$`NYHA V2`,useNA = "always")
```

```
0      1      2      3 <NA>
1      6      2      2     17
```

```
table(rawdata$`NYHA V2`,useNA = "i")
```

```
0      1      2      3 <NA>
1      6      2      2     17
```

```
table(rawdata$`NYHA V2`,useNA = "no")
```

```
0 1 2 3
1 6 2 2
```

```
randomize <- table(rawdata$Sex, rawdata$Testmedication)
prop.table(sex_count)
```

```
f          m
0.1428571 0.8571429
```

```
prop.table(randomize,margin = 2)*100
```

```
0          1
f 14.28571 14.28571
m 85.71429 85.71429
```

```
cat_desc_stats(rawdata$`NYHA V2`)
```

```
$level
# A tibble: 4 x 1
  value
  <chr>
1 0
2 1
3 2
4 3

$freq
# A tibble: 4 x 1
  desc
  <chr>
1 1 (9%)
2 6 (55%)
3 2 (18%)
4 2 (18%)
```

```
cat_desc_stats(rawdata$Sex, singleline = TRUE)
```

```
$level
[1] "f m"

$freq
# A tibble: 1 x 1
  desc
  <glue>
1 4 (14%) 24 (86%)
```

```
rawdata |>
  mutate(Testmedication=factor(Testmedication,
                                levels=0:1,
                                labels=c("Placebo",
                                         "Verum"))) |>
  cat_desc_table(
    desc_vars = factvars$names) |>
  rename(`n (%)`=desc_all) |>
  flextable() |>
  align(i = ~`n (%)`!=" ", j = 1, align = "right") |>
  width(j = c(1,2), width = c(3,4), unit = "cm") |>
```

```
bg(~`n` (%)`==` "", bg='lightgrey')
```

Variable	n (%)
Testmedication	
Placebo	14 (50%)
Verum	14 (50%)
Sex	
f	4 (14.29%)
m	24 (85.71%)
NYHA V1	
0	2 (11.76%)
1	9 (52.94%)
2	3 (17.65%)
3	3 (17.65%)
NYHA V2	
0	1 (9.09%)
1	6 (54.55%)
2	2 (18.18%)
3	2 (18.18%)
NYHA V3	
1	6 (50%)
2	3 (25%)
3	3 (25%)

## 9.8 Summarize data

When creating tables with descriptive statistics, you usually report on more than just 1 variable in more than just 1 subgroup, and there may be more than 1 statistics to report. `Summarize()`, often in combination with `across()`, makes that task easier, usually in a pipeline. As the output will contain (n variables) \* (n functions), wide to long pivoting often will be used. (More on this in the next chapter):

```

# we pipe the data
rawdata |>
  # into the summarize function
  summarize(across(all_of(gaussvars$names), #which variables to analyse?
    .fns=list( #which functions to apply?
      # ~ changes a function into a template
      n=~n(),
      # .x is the placeholder for the actual values
      Mean=~mean(.x,na.rm=TRUE) |>
        roundR(textout = F),
      Median=~median(.x,na.rm=TRUE) |>
        roundR(textout = F),
      SD=~sd(.x,na.rm=TRUE) |>
        roundR(textout = F)))) |>
  # the wide table with n vars * n functions columns is reshaped
  pivot_longer(
    cols = everything(), # transform all columns into long form
    names_to=c("Variable",".value"), # .value is extracting names for values
    names_sep="_") |>
  # pipe into formatted table
  flextable() |>
  set_table_properties(width=1, layout="autofit")

```

Variable	n	Mean	Median	SD
Size (cm)	28	174	174	7.8
Weight (kg)	28	88	84	15.0
sysBP V0	28	122	126	17.0
diaBP V0	28	70	69	9.8
Lv Edv Mri	28	206	193	70.0
Lv Esv Mri	28	110	77	70.0
Lv Ef Mri	28	50	55	15.0
Lv Ef Biplan Mri	28	49	51	14.0
sysBP V2	28	119	120	13.0
diaBP V2	28	66	68	8.6
BMI	28	29	28	4.3

```

rawdata |>
  group_by(Testmedication, Sex) |>
  summarise(WeightSummary=meansd(`Weight (kg)`, add_n = TRUE),
             .groups="drop") |>
  flextable()

```

Testmedication	Sex	WeightSummary
0	f	86 ± 9 [n = 2]
0	m	91 ± 14 [n = 12]
1	f	66 ± 3 [n = 2]
1	m	90 ± 14 [n = 12]

```

rawdata |>
  group_by(Sex) |>
  summarize(across(gaussvars$names,
                   .fns=~meansd(.x, range=T))) |>
  pivot_longer(cols=-Sex,
                names_to="Measure") |>
  pivot_wider(names_from=Sex) |>
  flextable() |>
  set_table_properties(width=1, layout="autofit")

```

Measure	f	m
Size (cm)	165 ± 4 [160 -> 168]	176 ± 7 [161 -> 193]
Weight (kg)	76 ± 13 [64 -> 93]	90 ± 14 [74 -> 120]
sysBP V0	118 ± 14 [102 -> 130]	123 ± 18 [94 -> 152]
diaBP V0	71 ± 13 [62 -> 90]	69 ± 9 [45 -> 90]
Lv Edv Mri	235 ± 72 [184 -> 286]	203 ± 71 [118 -> 379]
Lv Esv Mri	158 ± 56 [119 -> 198]	105 ± 71 [50 -> 294]
Lv Ef Mri	33 ± 3 [31 -> 35]	52 ± 15 [17 -> 74]
Lv Ef Biplan Mri	26 ± 11 [19 -> 34]	51 ± 13 [21 -> 69]
sysBP V2	114 ± 18 [96 -> 132]	120 ± 12 [100 -> 145]
diaBP V2	58 ± 12 [51 -> 72]	67 ± 8 [55 -> 80]

Measure	f	m
BMI	28 ± 5 [24 -> 35]	29 ± 4 [23 -> 41]

```
rawdata |>
  group_by(Sex) |>
  summarize(across(gaussvars$names,
    .fns=list(
      n=~n(),
      Mean=~mean(.x,na.rm=TRUE) |>
        roundR(textout = F),
      Median=~median(.x,na.rm=TRUE) |>
        roundR(textout = F),
      SD=~sd(.x,na.rm=TRUE) |>
        roundR(textout = F)))) |>
  pivot_longer(cols=-Sex,
    names_to=c("Variable","stat"),
    names_sep="_") |>
  pivot_wider(names_from=c(stat,Sex),
    # names_sep=" ",
    names_glue="{stat} ({Sex})",
    values_from=value) |>
  select(-starts_with("n")) |>
  flextable() |>
  set_table_properties(width=1, layout="autofit")
```

Variable	Mean (f)	Median (f)	SD (f)	Mean (m)	Median (m)	SD (m)
Size (cm)	165	166	3.8	176	174	7.2
Weight (kg)	76	74	13.0	90	86	14.0
sysBP V0	118	119	14.0	123	126	18.0
diaBP V0	71	66	13.0	69	70	9.5
Lv Edv MRI	235	235	72.0	203	193	71.0
Lv Esv MRI	158	158	56.0	105	71	71.0
Lv Ef MRI	33	33	2.8	52	56	15.0
Lv Ef Biplan MRI	26	26	11.0	51	54	13.0
sysBP V2	114	114	18.0	120	120	12.0
diaBP V2	58	52	12.0	67	68	7.8
BMI	28	27	4.7	29	29	4.4

```

compare2numvars(data = rawdata,
                 dep_vars = c( "Size (cm)", "Weight (kg)",
                             "sysBP V0", "diaBP V0"),
                 indep_var = "Sex",
                 gaussian = TRUE) |>
# select(-desc_all, -p) |>
rename(overall = desc_all) |>
rename_with(.fn = ~str_remove(.x, "Sex ")) |>
rename_with(.cols = "p", .fn = ~paste(.x, "- value")) |>
flextable() |>
set_table_properties(width=1, layout="autofit")

```

Variable	overall	f	m	p - value
Size (cm)	174 ± 8	165 ± 4	176 ± 7	0.009
Weight (kg)	88 ± 15	76 ± 13	90 ± 14	0.072
sysBP V0	122 ± 17	118 ± 14	123 ± 18	0.587
diaBP V0	70 ± 10	71 ± 13	69 ± 9	0.780

```

compare2qualvars(data = rawdata,
                  dep_vars = factvars$names[-2],
                  indep_var = factvars$names[2]) |>
flextable() |>
set_table_properties(width=1, layout="autofit")

```

Variable	desc_all	Sex f	Sex m	p
Testmedication				1.000
0	14 (50%)	2 (50%)	12 (50%)	
1	14 (50%)	2 (50%)	12 (50%)	
NYHA V1				0.094
0	2 (11.76%)	1 (33.33%)	1 (7.14%)	
1	9 (52.94%)	0 (0%)	9 (64.29%)	
2	3 (17.65%)	1 (33.33%)	2 (14.29%)	
3	3 (17.65%)	1 (33.33%)	2 (14.29%)	
NYHA V2				1.000
0	1 (9.09%)	0 (0%)	1 (12.5%)	

Variable	desc_all	Sex f	Sex m	p
1	6 (54.55%)	2 (66.67%)	4 (50%)	
2	2 (18.18%)	1 (33.33%)	1 (12.5%)	
3	2 (18.18%)	0 (0%)	2 (25%)	
NYHA V3				0.092
1	6 (50%)	0 (0%)	6 (66.67%)	
2	3 (25%)	1 (33.33%)	2 (22.22%)	
3	3 (25%)	2 (66.67%)	1 (11.11%)	

## 10 Exercises

Analyze the penguins data using summarize, functions can be any of mean/sd/median/meansd/median\_quart. Result tables should be pivoted if reasonable.

Describe species and sex with appropriate statistics as well.

- 1 function / 1 variable / no groups
- 2 functions / 1 variable / no groups
- 2 functions / 1 variable / subgroup species
- 1 function / 4 variables / no groups
- 1 function / 4 variables / subgroup species
- 2 functions / 4 variables / no subgroups
- 2 functions / 4 variables / subgroup species

# 11 Summarize / across

```
pacman::p_load(conflicted,tidyverse,wrappedtools,  
                flextable)
```

1 variable / 1 function / no groups

```
summarize(.data = mtcars,  
          MeanSD=meansd(mpg)) |>  
flextable()
```

MeanSD
20 ± 6

1 variable / 1 function / subgroups

```
mtcars |>  
group_by(am) |>  
summarize(MeanSD=meansd(mpg),  
          .groups = 'drop') |>  
flextable()
```

am	MeanSD
0	17 ± 4
1	24 ± 6

```
# groups in columns  
mtcars |>  
group_by(am) |>  
summarize(MeanSD=meansd(mpg),  
          .groups = 'drop') |>  
pivot_wider(names_from = am,  
            values_from = MeanSD,  
            names_glue = "am {.value}: {am}") |>  
flextable() |>
```

```
separate_header(split = ": ")
```

am (MeanSD)	
0	1
$17 \pm 4$	$24 \pm 6$

1 variable / 2 functions / no groups

```
summarize(.data = mtcars,
  `MeanSD mpg` = meansd(mpg, roundDig = 3),
  `MedianQuartiles mpg` = median_quart(mpg, roundDig = 3)) |>
flextable() |>
set_table_properties(width=.7, layout="autofit")
```

MeanSD mpg	MedianQuartiles mpg
$20.1 \pm 6.0$	19.2 (15.3/22.8)

1 variable / 2 functions / subgroups

```
mtcars |>
  group_by(am) |>
  summarize(`MeanSD mpg` = meansd(mpg),
    `MedianQuartiles mpg` = median_quart(mpg),
    .groups = 'drop') |>
flextable()
```

am	MeanSD mpg	Median- Quartiles mpg
0	$17 \pm 4$	17 (15/19)
1	$24 \pm 6$	23 (21/30)

```
# groups in columns
mtcars |>
  group_by(am) |>
  summarize(`MeanSD mpg` = meansd(mpg),
    `MedianQuartiles mpg` = median_quart(mpg),
    .groups = 'drop') |>
```

```

pivot_longer(
  cols = starts_with("M"),
  names_to = "Statistics",
  values_to = "estimate") |>
pivot_wider(names_from = am,
            values_from = estimate,
            names_glue = "am: {am}") |>
flextable()

```

Statistics	am: 0	am: 1
MeanSD	$17 \pm 4$	$24 \pm 6$
mpg	Median- Quartiles	17 (15/19) 23 (21/30)
mpg		

2 variables / 1 function / no groups

```

# no function arguments
mtcars |>
  summarize(across(.cols = c(mpg,disp),
                  .fns=meansd)) |>
  flextable() |>
  set_table_properties(width=.7, layout="autofit")

```

mpg	disp
$20 \pm 6$	$231 \pm 124$

```

# with function arguments
mtcars |>
  summarize(across(.cols=c(mpg,disp),
                  .fns=~meansd(.x,add_n = TRUE,range = TRUE))) |>
  flextable() |>
  set_table_properties(width=.7, layout="autofit")

```

mpg	disp
$20 \pm 6$ [10 -> 34] [n = 32]	$231 \pm 124$ [71 -> 472] [n = 32]

```
# Variables in rows
mtcars |>
  summarize(across(c(mpg,disp),
    .fns=~meansd(.x,add_n = TRUE))) |>
  pivot_longer(
    cols = everything(),
    names_to = "Variable",
    values_to = "MeanSD") |>
  flextable() |>
  set_table_properties(width=.7, layout="autofit")
```

	Variable MeanSD
mpg	20 ± 6 [n = 32]
disp	231 ± 124 [n = 32]

2 variables / 1 function / subgroups

```
mtcars |>
  group_by(am) |>
  summarize(across(c(mpg,disp),
    ~meansd(.x,add_n = TRUE)),
    .groups = 'drop') |>
  flextable() |>
  set_table_properties(width=.7, layout="autofit")
```

am	mpg	disp
0	17 ± 4 [n = 19]	290 ± 110 [n = 19]
1	24 ± 6 [n = 13]	144 ± 87 [n = 13]

```
# Variables in rows
mtcars |>
  group_by(am) |>
  summarize(across(c(mpg,disp),
    ~meansd(.x,add_n = TRUE)),
    .groups = 'drop') |>
  pivot_longer(
    cols = -am,
    names_to = "Variable",
    values_to = "MeanSD") |>
  pivot_wider(names_from = am,
```

```

    values_from = MeanSD,
    names_prefix = "am: ") |>
flextable() |>
set_table_properties(width=.7, layout="autofit")

```

	Variable am: 0	am: 1
mpg	17 ± 4 [n = 19]	24 ± 6 [n = 13]
disp	290 ± 110 [n = 19]	144 ± 87 [n = 13]

2 variables / 2 function / no groups

```

# with/without function arguments
mtcars |>
  summarize(across(
    c(mpg,disp),
    .fns=list(
      MeanSD=~meansd(.x,
                     add_n = TRUE),
      MedianQuart=median_quart),
    .names = "{.col}: {.fn}")) |>
  flextable() |>
  set_table_properties(width=1, layout="autofit")

```

mpg: MeanSD	mpg: MedianQuart	disp: MeanSD	disp: MedianQuart
20 ± 6 [n = 32]	19 (15/23)	231 ± 124 [n = 32]	196 (121/337)

```

# Variables in rows
mtcars |>
  summarize(across(
    c(mpg,disp),
    .fns=list(
      MeanSD=~meansd(.x,
                     add_n = TRUE),
      MedianQuart=median_quart))) |>
  pivot_longer(
    cols = everything(),
    names_to = c("Variable",".value"),
    names_sep="_") |>
  flextable() |>
  set_table_properties(width=1, layout="autofit")

```

Variable	MeanSD	MedianQuart
mpg	$20 \pm 6$ [n = 32]	19 (15/23)
disp	$231 \pm 124$ [n = 32]	196 (121/337)

2 variables / 2 function / subgroups

```
# with/without function arguments
mtcars |>
  group_by(am) |>
  summarize(across(
    c(mpg, disp),
    .fns=list(
      MeanSD=~meansd(.x,
                      add_n = TRUE),
      MedianQuart=median_quart)),
    .groups="drop") |>
  flextable() |>
  set_table_properties(width=1, layout="autofit")
```

am	mpg_MeanSD	mpg_MedianQuart	disp_MeanSD	disp_MedianQuart
0	$17 \pm 4$ [n = 19]	17 (15/19)	$290 \pm 110$ [n = 19]	276 (177/360)
1	$24 \pm 6$ [n = 13]	23 (21/30)	$144 \pm 87$ [n = 13]	120 (79/160)

```
# Variables in rows, groups in columns
mtcars |>
  group_by(am) |>
  summarize(across(
    c(mpg, disp),
    .fns=list(
      MeanSD=~meansd(.x,
                      add_n = TRUE),
      MedianQuart=median_quart)),
    .groups="drop") |>
  pivot_longer(
    cols = -am,
    names_to = c("Variable", ".value"),
    names_sep="_") |>
  pivot_wider(names_from = am,
              values_from = starts_with("M"),
              names_glue = "am: {am}_{.value}",
              names_vary="slowest") |>
```

```

flextable() |>
separate_header(split="[:_]") |>
set_table_properties(width=1, layout="autofit")

```

---

Variable	am		MedianQuart	MeanSD	MedianQuart
	0	1			
mpg	17 ± 4 [n = 19]	17 (15/19)	24 ± 6 [n = 13]	23 (21/30)	
disp	290 ± 110 [n = 19]	276 (177/360)	144 ± 87 [n = 13]	120 (79/160)	

---

```

# pivoting to have variables in rows V2

# with/without function arguments
result_long <-
mtcars |>
group_by(am) |>
summarize(across(
  c(mpg, disp),
  .fns=list(
    MeanSD=~meansd(.x,
                    add_n = TRUE),
    MedianQuart=median_quart))) |>
pivot_longer(cols = -c(am),
             names_to = c('Variable','.value'),
             names_sep="_",
             values_to = 'Value')
result_long |>
flextable() |>
merge_v(j=1) |>
set_table_properties(width=1, layout="autofit")

```

---

	am	Variable	MeanSD	MedianQuart
0	mpg	17 ± 4 [n = 19]	17 (15/19)	
	disp	290 ± 110 [n = 19]	276 (177/360)	
1	mpg	24 ± 6 [n = 13]	23 (21/30)	
	disp	144 ± 87 [n = 13]	120 (79/160)	

---

```

result <-
  result_long |>
  pivot_wider(names_from=am,
              names_prefix="am:",
              names_sep=" ",
              values_from=c(MeanSD, MedianQuart))
result |>
  flextable() |>
  separate_header(split="[ ]") |>
  set_table_properties(width=1, layout="autofit")

```

Variable	MeanSD		MedianQuart	
	am:0	am:1	am:0	am:1
mpg	17 ± 4 [n = 19]	24 ± 6 [n = 13]	17 (15/19)	23 (21/30)
disp	290 ± 110 [n = 19]	144 ± 87 [n = 13]	276 (177/360)	120 (79/160)



## 12 Simple test statistics

### 12.1 Tests require hypotheses



### 12.1.1 Null hypothesis ?

- Working hypothesis: This is what you expect!  
E.g. treatment is lowering blood pressure more than placebo, transgenic animals become obese, bio reactor A is more efficient than B, concentration of substance is correlated with speed of reaction ...
- Null hypothesis: This is what you test!  
No difference / relation, BP under therapy = BP under placebo

#### 4 possibilities:

- Null hypothesis correct, test false positive (case A): alpha-error
- Null hypothesis correct, test correct negative (case B)
- Null hypothesis false, test false negative (case C): beta-error
- Null hypothesis false, test correct positive (case D)

**Significance:** NOT probability of case A, but probability of your data given the NULL hypothesis, calculated from your data, conventionally  $<0.05$

**Power:** Probability of case D, *estimated* based on assumptions about effects and sample size, *calculation* would require knowledge of true difference, conventionally set at 0.80; this translates into a **20% risk of false negative results!**

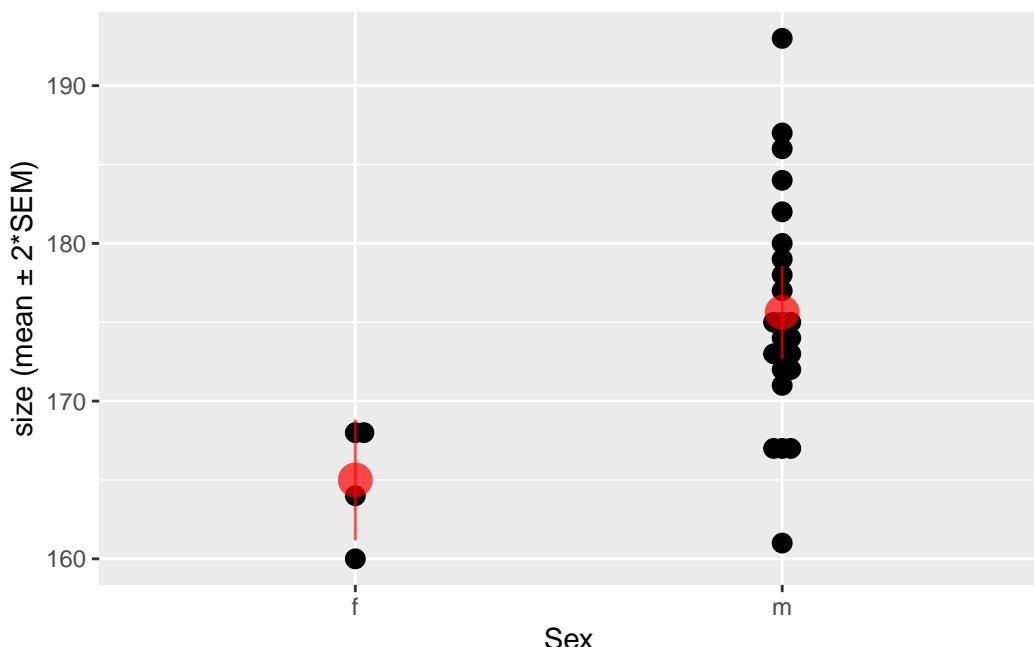
```
pacman::p_load(conflicted, plotrix, tidyverse, wrappedtools,
                 coin, ggsignif, patchwork, ggbeeswarm,
                 flextable, here)
#conflicted)
# conflict_prefer("filter", "dplyr")
load(here("data/bookdata1.RData"))
```

## 12.2 Quantitative measures with Gaussian distribution

### t-test

- Assumptions: Continuous data with Normal distribution
- 1 or 2 (independent or dependent) samples with/without equal variances
- how big is the mean difference relative to uncertainty?  
 $t = (\text{mean}_1 - \text{mean}_2)/\text{SEM}$
- t follows a t-distribution, allows estimation of probability of t under the NULL hypothesis

```
ggplot(rawdata,aes(x=Sex,y=`Size (cm)`))+  
  geom_beeswarm(size=3)+  
  stat_summary(color="red",size=1.2,alpha=.7,  
               fun.data="mean_se",fun.args=list(mult=2))+  
  ylab("size (mean \u00B1 2*SEM)")
```



```
rawdata |>  
  group_by(Sex) |>  
  summarize(MeanSE=meanse(`Size (cm)`),  
            SD=sd(`Size (cm)`))
```

```
# A tibble: 2 x 3  
  Sex    MeanSE     SD  
  <fct> <chr>   <dbl>  
1 f      165 ± 2  3.83  
2 m      176 ± 1  7.22
```

```
t.test(x = rawdata$`Size (cm)`[which(rawdata$Sex=="f")],  
       y = rawdata$`Size (cm)`[which(rawdata$Sex=="m")])
```

Welch Two Sample t-test

```
data: rawdata$`Size (cm)`[which(rawdata$Sex == "f")] and rawdata$`Size (cm)`[which(rawdat
t = -4.3967, df = 7.2767, p-value = 0.002887
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-16.295575 -4.954425
sample estimates:
mean of x mean of y
165.000 175.625
```

```
tOut<-t.test(rawdata$`Size (cm)`~rawdata$Sex)
tOut$p.value
```

```
[1] 0.00288704
```

```
# equal variances assumption?
vartestOut<-var.test(rawdata$`Size (cm)`~rawdata$Sex)
vartestOut
```

F test to compare two variances

```
data: rawdata$`Size (cm)` by rawdata$Sex
F = 0.2812, num df = 3, denom df = 23, p-value = 0.3232
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
0.07497668 3.97434769
sample estimates:
ratio of variances
0.281199
```

```
# car::leveneTest(rawdata$`Size (cm)`~rawdata$Sex)
# manual entry
t.test(rawdata$`Size (cm)`~rawdata$Sex,
      var.equal = TRUE)
```

Two Sample t-test

```
data: rawdata$`Size (cm)` by rawdata$Sex
t = -2.8446, df = 26, p-value = 0.008552
alternative hypothesis: true difference in means between group f and group m is not equal
95 percent confidence interval:
```

```
-18.302594 -2.947406
sample estimates:
mean in group f mean in group m
165.000      175.625
```

```
t.test(rawdata$`Size (cm)`~rawdata$Sex,
       var.equal = vartestOut$p.value>.05)
```

#### Two Sample t-test

```
data: rawdata$`Size (cm)` by rawdata$Sex
t = -2.8446, df = 26, p-value = 0.008552
alternative hypothesis: true difference in means between group f and group m is not equal
95 percent confidence interval:
-18.302594 -2.947406
sample estimates:
mean in group f mean in group m
165.000      175.625
```

```
# picked from test
t.test(rawdata$`Size (cm)`~rawdata$Sex,
       var.equal=var.test(
         rawdata$`Size (cm)`~rawdata$Sex)$p.value>.05)
```

#### Two Sample t-test

```
data: rawdata$`Size (cm)` by rawdata$Sex
t = -2.8446, df = 26, p-value = 0.008552
alternative hypothesis: true difference in means between group f and group m is not equal
95 percent confidence interval:
-18.302594 -2.947406
sample estimates:
mean in group f mean in group m
165.000      175.625
```

```
#combined function
t_var_test(data = rawdata,
            formula = "`Size (cm)`~Sex",
            cutoff = .1)
```

### Two Sample t-test

```
data: Size (cm) by Sex
t = -2.8446, df = 26, p-value = 0.008552
alternative hypothesis: true difference in means between group f and group m is not equal
95 percent confidence interval:
-18.302594 -2.947406
sample estimates:
mean in group f mean in group m
165.000          175.625
```

```
print(c(mean(rawdata$`sysBP V0`,na.rm=T),
       mean(rawdata$`sysBP V2`,na.rm=T)))
```

```
[1] 121.8519 119.4583
```

```
t.test(rawdata$`sysBP V0`,
       rawdata$`sysBP V2`,
       alternative="greater", # x>y
       paired=TRUE) #pairwise t-test, within subject
```

### Paired t-test

```
data: rawdata$`sysBP V0` and rawdata$`sysBP V2`
t = 0.88151, df = 23, p-value = 0.1936
alternative hypothesis: true mean difference is greater than 0
95 percent confidence interval:
-2.793386      Inf
sample estimates:
mean difference
2.958333
```

```
t.test(rawdata$`sysBP V0`,
       rawdata$`sysBP V2`,
       # alternative="greater", # x>y
       paired=T)$p.value/2 #pairwise t-test, within subject
```

```
[1] 0.1935805
```

```
t.test(rawdata$`Size (cm)`, mu = 173)
```

### One Sample t-test

```
data: rawdata$`Size (cm)`
t = 0.75384, df = 27, p-value = 0.4575
alternative hypothesis: true mean is not equal to 173
95 percent confidence interval:
171.0937 177.1206
sample estimates:
mean of x
174.1071
```

```
groupvars <- ColSeeker(namepattern = c("Sex", "Test"))

compare2numvars(data = rawdata, dep_vars = gaussvars$names,
                 indep_var = "Sex", gaussian = T,
                 round_desc = 3, n = T, mark=T) |>
  flextable() |>
  set_table_properties(width=1, layout="autofit")
```

Variable	n desc_all	n g1 Sex f	n g2 Sex m	p
Size (cm)	28 174 ± 8	4 165 ± 4	24 176 ± 7	0.009 **
Weight (kg)	28 88.4 ± 14.6	4 76.2 ± 13.1	24 90.4 ± 14.1	0.072 +
sysBP V0	27 122 ± 17	4 118 ± 14	23 123 ± 18	0.587 n.s.
diaBP V0	27 69.7 ± 9.8	4 71.0 ± 12.8	23 69.5 ± 9.5	0.780 n.s.
Lv Edv MRI	21 206 ± 70	2 235 ± 72	19 203 ± 71	0.559 n.s.
Lv Esv MRI	21 110 ± 70	2 158 ± 56	19 105 ± 71	0.322 n.s.
Lv Ef MRI	21 49.8 ± 14.9	2 33.0 ± 2.8	19 51.6 ± 14.6	0.095 +
Lv Ef Biplan MRI	21 48.7 ± 14.5	2 26.5 ± 10.6	19 51.0 ± 12.9	0.018 *
sysBP V2	24 119 ± 13	3 114 ± 18	21 120 ± 12	0.438 n.s.
diaBP V2	24 66.1 ± 8.6	3 58.3 ± 11.8	21 67.2 ± 7.8	0.097 +
BMI	28 29.1 ± 4.3	4 28.0 ± 4.7	24 29.3 ± 4.4	0.585 n.s.

```

compare2numvars(data = rawdata, dep_vars = gaussvars$names,
                 indep_var = "Testmedication", gaussian = T,
                 round_desc = 4) |>
flextable() |>
set_table_properties(width=1, layout="autofit")

```

Variable	desc_all	Testmedication 0	Testmedication 1	p
Size (cm)	174.1 ± 7.8	173.4 ± 6.4	174.8 ± 9.2	0.653
Weight (kg)	88.36 ± 14.59	90.50 ± 13.51	86.21 ± 15.81	0.448
sysBP V0	121.9 ± 16.9	122.0 ± 15.1	121.7 ± 19.0	0.966
diaBP V0	69.70 ± 9.75	69.85 ± 7.12	69.57 ± 11.97	0.943
Lv Edv Mri	206.4 ± 70.3	242.9 ± 76.9	173.2 ± 45.0	0.019
Lv Esv Mri	110.5 ± 70.3	137.8 ± 87.0	85.6 ± 40.5	0.108
Lv Ef Mri	49.81 ± 14.95	47.60 ± 18.33	51.82 ± 11.63	0.532
Lv Ef Biplan Mri	48.67 ± 14.47	47.44 ± 17.06	49.58 ± 12.92	0.747
sysBP V2	119.5 ± 12.7	122.5 ± 11.2	116.9 ± 13.7	0.297
diaBP V2	66.08 ± 8.63	66.00 ± 8.10	66.15 ± 9.39	0.966
BMI	29.13 ± 4.35	30.24 ± 5.25	28.02 ± 3.00	0.180

```

for(group_i in seq_len(groupvars$count)){
  resulttmp <-
    compare2numvars(data = rawdata,
                     dep_vars = gaussvars$names,
                     indep_var = groupvars$names[group_i], gaussian = T)
  # print(resulttmp)
  flextable(resulttmp) |>
    set_table_properties(width=1, layout="autofit") |>
    flex2rmd() #|> print()

  cat("\\\\newpage\\n\\n")
}

```

Variable	desc_all	Testmedication 0	Testmedication 1	p
Size (cm)	174 ± 8	173 ± 6	175 ± 9	0.653
Weight (kg)	88 ± 15	90 ± 14	86 ± 16	0.448
sysBP V0	122 ± 17	122 ± 15	122 ± 19	0.966

Variable	desc_all	Testmedication 0	Testmedication 1	p
diaBP V0	70 ± 10	70 ± 7	70 ± 12	0.943
Lv Edv Mri	206 ± 70	243 ± 77	173 ± 45	0.019
Lv Esv Mri	110 ± 70	138 ± 87	86 ± 41	0.108
Lv Ef Mri	50 ± 15	48 ± 18	52 ± 12	0.532
Lv Ef Biplan Mri	49 ± 14	47 ± 17	50 ± 13	0.747
sysBP V2	119 ± 13	122 ± 11	117 ± 14	0.297
diaBP V2	66 ± 9	66 ± 8	66 ± 9	0.966
BMI	29 ± 4	30 ± 5	28 ± 3	0.180

Variable	desc_all	Sex f	Sex m	p
Size (cm)	174 ± 8	165 ± 4	176 ± 7	0.009
Weight (kg)	88 ± 15	76 ± 13	90 ± 14	0.072
sysBP V0	122 ± 17	118 ± 14	123 ± 18	0.587
diaBP V0	70 ± 10	71 ± 13	69 ± 9	0.780
Lv Edv Mri	206 ± 70	235 ± 72	203 ± 71	0.559
Lv Esv Mri	110 ± 70	158 ± 56	105 ± 71	0.322
Lv Ef Mri	50 ± 15	33 ± 3	52 ± 15	0.095
Lv Ef Biplan Mri	49 ± 14	26 ± 11	51 ± 13	0.018
sysBP V2	119 ± 13	114 ± 18	120 ± 12	0.438
diaBP V2	66 ± 9	58 ± 12	67 ± 8	0.097
BMI	29 ± 4	28 ± 5	29 ± 4	0.585

## 12.3 Ordinal data

### Wilcoxon-test / Mann-Whitney U test

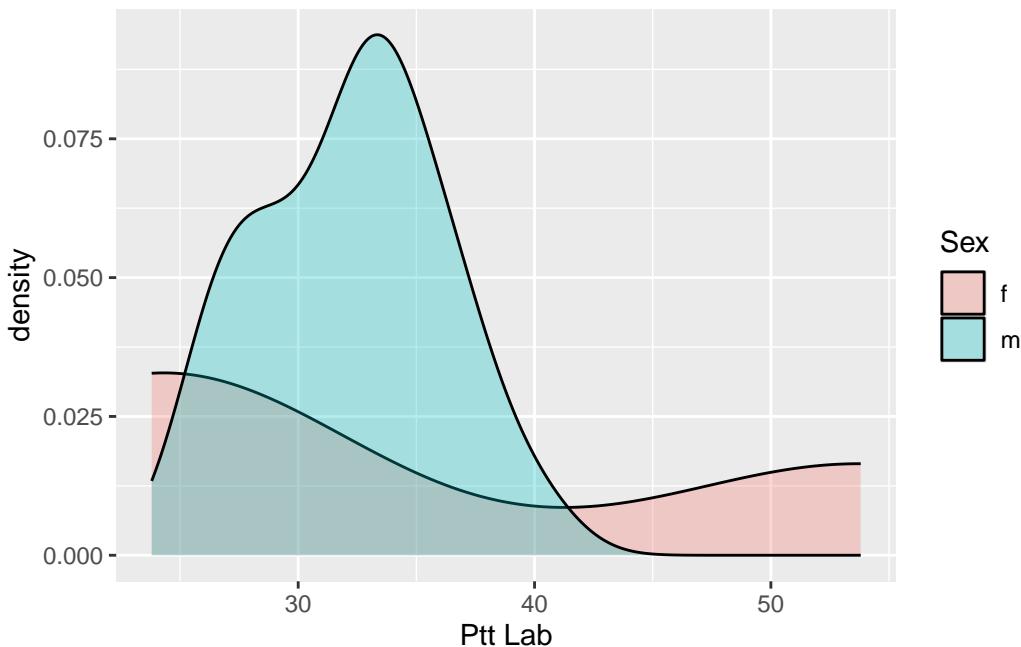
- nonparametric, no distribution is assumed
- based on rank-transformed data
- insensitive to extreme values

```
ordvars$names
```

```
[1] "Ptt Lab"           "Ferritin Lab"      "Iron Lab"        "Transferrin Lab"  
[5] "Age"
```

```
ggplot(rawdata,aes(`Ptt Lab`,fill=Sex))+  
  geom_density(alpha=.3)
```

Warning: Removed 1 row containing non-finite outside the scale range  
(`stat\_density()`).



```
by(data = rawdata[[ordvars$index[1]]],  
  INDICES = rawdata$Sex,FUN = median_quart)
```

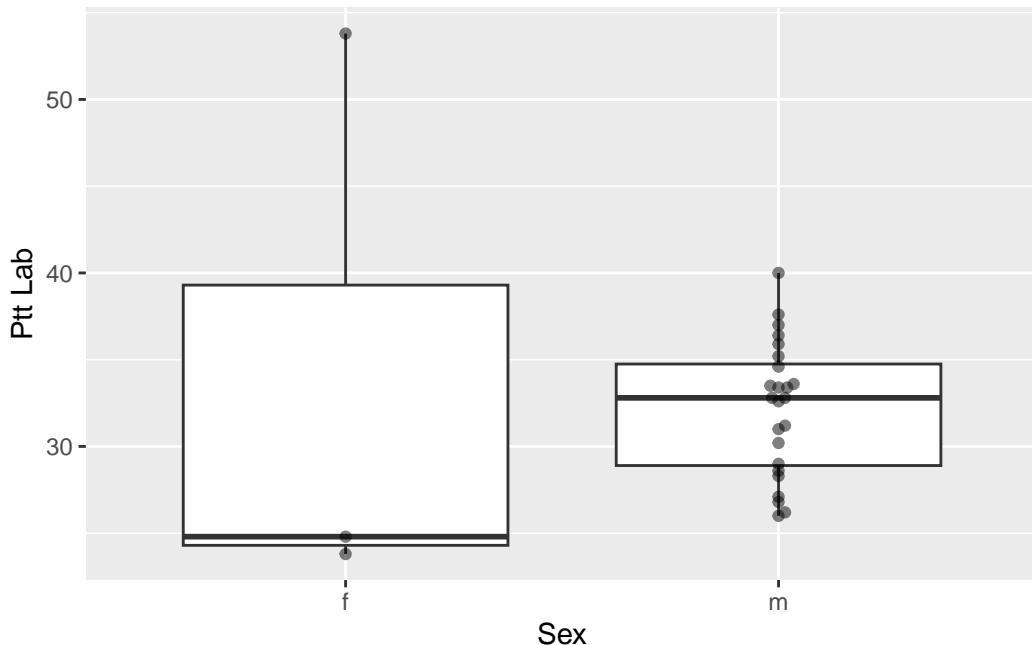
```
rawdata$Sex: f  
[1] "25 (24/49)"
```

```
-----  
rawdata$Sex: m  
[1] "33 (29/35)"
```

```
ggplot(rawdata,aes(Sex,`Ptt Lab`))+  
  geom_boxplot() +  
  geom_beeswarm(alpha=.5)
```

Warning: Removed 1 row containing non-finite outside the scale range  
(`stat\_boxplot()`).

Warning: Removed 1 row containing missing values or values outside the scale range  
(`geom\_point()`).



```
(uOut<-wilcox.test(  
  rawdata[[ordvars$names[1]]] ~ rawdata$Sex, exact=F))
```

Wilcoxon rank sum test with continuity correction  
data: rawdata[[ordvars\$names[1]]] by rawdata\$Sex

```
W = 24, p-value = 0.3748
alternative hypothesis: true location shift is not equal to 0
```

```
uOut$p.value
```

```
[1] 0.3748016
```

```
# coin::wilcox_test
(uOut2<-wilcox_test(`Ptt Lab`~Sex,
                      data=rawdata))
```

```
Asymptotic Wilcoxon-Mann-Whitney Test
```

```
data: Ptt Lab by Sex (f, m)
Z = -0.9261, p-value = 0.3544
alternative hypothesis: true mu is not equal to 0
```

```
pvalue(uOut2) #no list-object, but methods to extract infos like p
```

```
[1] 0.3543925
```

```
wilcox.test(`Ptt Lab`~Sex,exact=F,correct=F,
            data=rawdata)
```

```
Wilcoxon rank sum test
```

```
data: Ptt Lab by Sex
W = 24, p-value = 0.3544
alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(x=rawdata$`sysBP V0`,y=rawdata$`sysBP V2`,
            exact=FALSE,
            correct=TRUE,paired=TRUE)
```

```

Wilcoxon signed rank test with continuity correction

data: rawdata$`sysBP V0` and rawdata$`sysBP V2`
V = 143, p-value = 0.3478
alternative hypothesis: true location shift is not equal to 0

```

```

compare2numvars(data = rawdata,dep_vars = ordvars$names,n = F,
                 range = T,add_n = T,
                 indep_var = "Sex",gaussian = F) |>
flextable() |>
set_table_properties(width=1, layout="autofit")

```

Variable	desc_all	Sex f	Sex m
Ptt Lab	33 (28/35) [24 -> 54] [n = 27]	25 (24/49) [24 -> 54] [n = 3]	33 (29/35) [26 ->
Ferritin Lab	222 (162/339) [20 -> 1182] [n = 27]	138 (90/591) [81 -> 681] [n = 3]	224 (172/316) [2
Iron Lab	80 (61/102) [31 -> 191] [n = 27]	95 (92/103) [91 -> 105] [n = 3]	75 (60/99) [31 ->
Transferrin Lab	261 (233/276) [194 -> 343] [n = 27]	260 (227/299) [221 -> 307] [n = 3]	262 (235/276) [1
Age	62 (53/67) [43 -> 74] [n = 28]	66 (58/69) [53 -> 69] [n = 4]	60 (53/66) [43 ->

## 12.4 Categorical data

```
factvars$names
```

```
[1] "Testmedication" "Sex"           "NYHA V1"          "NYHA V2"
[5] "NYHA V3"
```

```
(crosstab<-table(rawdata$Sex,rawdata$Testmedication))
```

0	1	
f	2	2
m	12	12

```
chisq.test(crosstab,simulate.p.value=T,B=10^5) #empirical p-value
```

```
Pearson's Chi-squared test with simulated p-value (based on 1e+05  
replicates)
```

```
data: crosstab  
X-squared = 0, df = NA, p-value = 1
```

```
chisq.test(table(rawdata$Sex, rawdata$`NYHA V1`)) #based on table
```

```
Warning in chisq.test(table(rawdata$Sex, rawdata$`NYHA V1`)):  
Chi-Quadrat-Approximation kann inkorrekt sein
```

```
Pearson's Chi-squared test
```

```
data: table(rawdata$Sex, rawdata$`NYHA V1`)  
X-squared = 4.3849, df = 3, p-value = 0.2228
```

```
chisq.test(x=rawdata$Sex, y=rawdata$`NYHA V1`,  
simulate.p.value=T, B=10^5) #based on rawdata
```

```
Pearson's Chi-squared test with simulated p-value (based on 1e+05  
replicates)
```

```
data: rawdata$Sex and rawdata$`NYHA V1`  
X-squared = 4.3849, df = NA, p-value = 0.1755
```

```
fisher_out <- fisher.test(  
  table(rawdata$Sex, rawdata$`NYHA V1`))  
fisher_out$p.value
```

```
[1] 0.09558824
```

```
(crosstab1<-table(rawdata$Sex,  
  rawdata$`Weight (kg)`<=  
  median(rawdata$`Weight (kg)`)))
```

	FALSE	TRUE
f	1	3
m	13	11

```
(tabletestOut<-chisq.test(crosstab1, simulate.p.value=T,  
                           B=10^5))
```

Pearson's Chi-squared test with simulated p-value (based on 1e+05 replicates)

```
data: crosstab1  
X-squared = 1.1667, df = NA, p-value = 0.596
```

```
tabletestOut$p.value
```

```
[1] 0.595964
```

```
tabletestOut$expected
```

	FALSE	TRUE
f	2	2
m	12	12

```
tabletestOut$observed
```

	FALSE	TRUE
f	1	3
m	13	11

```
tabletestOut$statistic
```

X-squared  
1.166667

```
# if minimum(expected<5) then Fishers exact test  
if(min(tabletestOut$expected)<5) {  
  tabletestOut<-fisher.test(crosstab1)  
}  
tabletestOut$p.value
```

```
[1] 0.5955556
```

```

# report_cat
groupvar <- "Testmedication"

rawdata |>
  mutate(Testmedication=factor(Testmedication,
                                levels=0:1,
                                labels=c("Placebo","Verum"))) |>
  compare2qualvars(dep_vars = factvars$names[-1],
                    indep_var = groupvar, spacer = " ") |>
  rename_with(~str_remove(.x, "Testmedication")) |>
  rename(`Total sample`=desc_all) |>
  flextable() |>
  align(~p==" ", j = 1, align = "center") |>
  bg(~p!=" ",bg = "lightgrey")

```

Variable	Total sample	Placebo	Verum	p
<b>Sex</b>				1.000
f	4 (14.29%)	2 (14.29%)	2 (14.29%)	
m	24 (85.71%)	12 (85.71%)	12 (85.71%)	
<b>NYHA V1</b>				0.730
0	2 (11.76%)	0 (0%)	2 (20%)	
1	9 (52.94%)	4 (57.14%)	5 (50%)	
2	3 (17.65%)	1 (14.29%)	2 (20%)	
3	3 (17.65%)	2 (28.57%)	1 (10%)	
<b>NYHA V2</b>				0.826
0	1 (9.09%)	1 (20%)	0 (0%)	
1	6 (54.55%)	2 (40%)	4 (66.67%)	
2	2 (18.18%)	1 (20%)	1 (16.67%)	
3	2 (18.18%)	1 (20%)	1 (16.67%)	
<b>NYHA V3</b>				0.773
1	6 (50%)	2 (40%)	4 (57.14%)	
2	3 (25%)	1 (20%)	2 (28.57%)	
3	3 (25%)	2 (40%)	1 (14.29%)	

## 13 Exercise

Compare the 4 measurements in the penguin data between the sex as well as between 2 species (e.g Adelie vs. Gentoo). Treat the measures first as gaussian, then as ordinal.

Use the basic functions first for at least one phenotype before trying out compare2num-vars().

Look at the categorical data year, island, species and sex and run some tests on their independence.

# 14 Covariance / Correlation

```
pacman::p_load(conflicted, plotrix, tidyverse, wrappedtools,
                 coin, ggsignif, patchwork, ggbeeswarm,
                 flextable, here, corrr)
load(here("data/bookdata1.RData"))
```

## 14.1 Covariance

Covariance is a measure of the relationship between two random variables. The sign of the covariance shows the tendency in the linear relationship between the variables. The covariance between two variables is computed as follows:

```
cov(rawdata$`Weight (kg)`, rawdata$`Size (cm)`)
```

```
[1] 51.55291
```

```
cov(rawdata |> select(contains("BP")),
     use = "pairwise.complete.obs")
```

```
      sysBP V0 diaBP V0 sysBP V2 diaBP V2
sysBP V0 285.4387464 48.76211 102.23551 -0.5144928
diaBP V0 48.7621083 95.06268 -38.07065 28.7065217
sysBP V2 102.2355072 -38.07065 160.78080 -10.6050725
diaBP V2 -0.5144928 28.70652 -10.60507 74.5144928
```

## 14.2 Correlation

Correlation is a measure of the strength and direction of the linear relationship between two variables. Other than the covariance, it is standardized, so it is not affected by the scale of the variables. The correlation between two variables is computed as follows:

```
cor(rawdata$`Weight (kg)`, rawdata$`Size (cm)`)
```

```
[1] 0.454551
```

```
cor(rawdata |> select(contains("Mri")),
  use = "pairwise.complete.obs")
```

```
          Lv Edv Mri Lv Esv Mri  Lv Ef Mri Lv Ef Biplan Mri
Lv Edv Mri      1.0000000 0.9180259 -0.6653199      -0.6022713
Lv Esv Mri      0.9180259 1.0000000 -0.8913242      -0.8375638
Lv Ef Mri      -0.6653199 -0.8913242 1.0000000       0.9544592
Lv Ef Biplan Mri -0.6022713 -0.8375638  0.9544592      1.0000000
```

Significance of correlations can be tested:

```
cor.test(rawdata$`Weight (kg)`, rawdata$`Size (cm)`)
```

```
Pearson's product-moment correlation

data: rawdata$`Weight (kg)` and rawdata$`Size (cm)`
t = 2.6021, df = 26, p-value = 0.0151
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
0.09811224 0.70762684
sample estimates:
cor
0.454551
```

```
wrappedtools::cortestR(rawdata |> select(contains("BP")))
```

```
      sysBP V0    diaBP V0    sysBP V2 diaBP V2
sysBP V0      1
diaBP V0  0.296n.s.      1
sysBP V2      0.455* -0.294n.s.      1
diaBP V2 -0.003n.s.  0.326n.s. -0.097n.s.      1
```

```
wrappedtools::cortestR(rawdata |> select(contains("BP")),
  split=TRUE)
```

```

$corout
    sysBP V0 diaBP V0 sysBP V2 diaBP V2
sysBP V0      1
diaBP V0     0.296      1
sysBP V2     0.455   -0.294      1
diaBP V2    -0.003    0.326   -0.097      1

$pout
    sysBP V0 diaBP V0 sysBP V2 diaBP V2
sysBP V0      ***
diaBP V0     n.s.      ***
sysBP V2      *     n.s.      ***
diaBP V2     n.s.     n.s.     n.s.      ***

cor_out <-
  wrappedtools::cortestR(rawdata |> select(contains("BP")),
                         split=TRUE, sign_symbol=FALSE)
cor_out

```

```

$corout
    sysBP V0 diaBP V0 sysBP V2 diaBP V2
sysBP V0      1
diaBP V0     0.296      1
sysBP V2     0.455   -0.294      1
diaBP V2    -0.003    0.326   -0.097      1

$pout
    sysBP V0 diaBP V0 sysBP V2 diaBP V2
sysBP V0     0.001
diaBP V0     0.134    0.001
sysBP V2     0.025    0.163    0.001
diaBP V2     0.988    0.121    0.652    0.001

```

## 14.3 Vizualizations

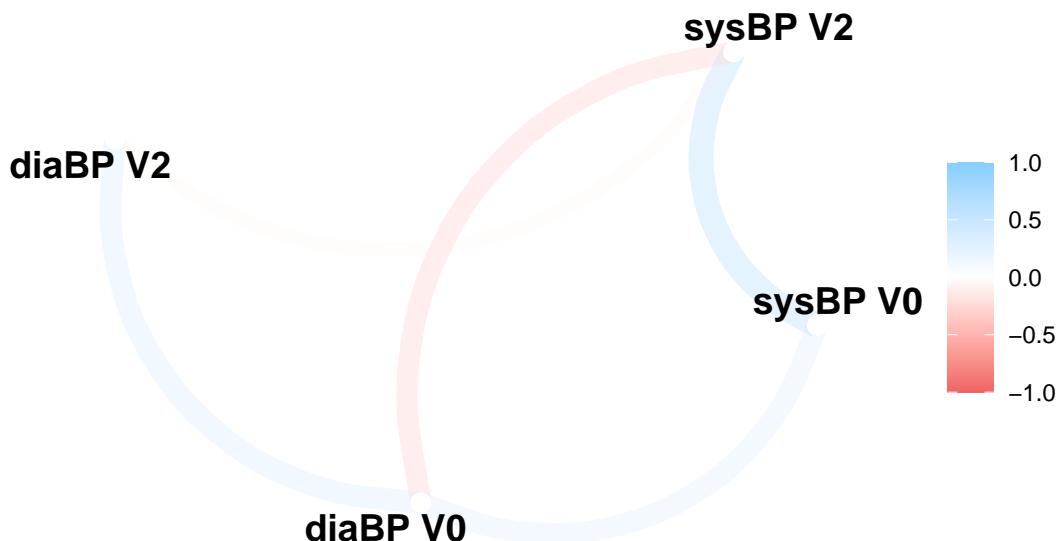
```

corrr::correlate(rawdata |> select(contains("BP"))) |>
  corrr::network_plot(min_cor = .05)

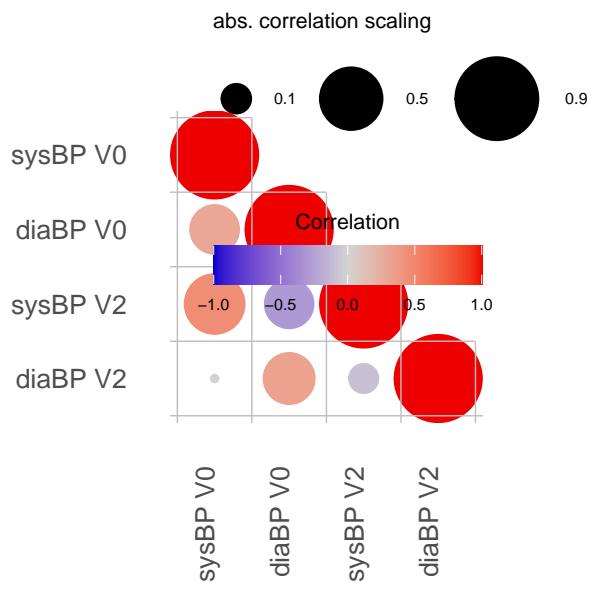
```

Correlation computed with  
 \* Method: 'pearson'  
 \* Missing treated using: 'pairwise.complete.obs'

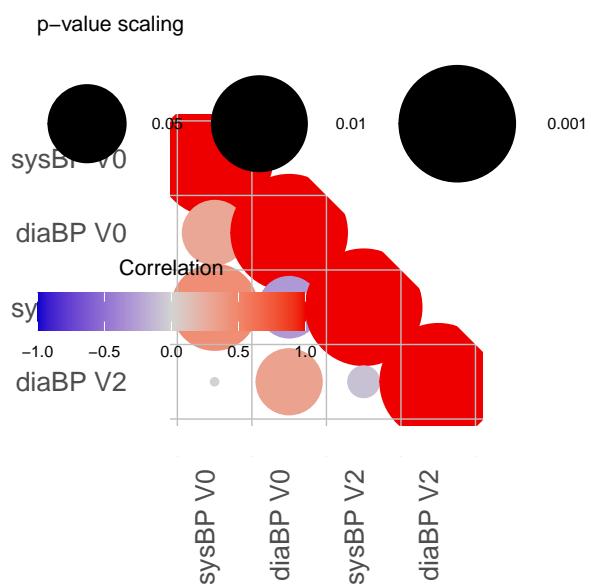
```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.  
i Please use `linewidth` instead.  
i The deprecated feature was likely used in the corrr package.  
Please report the issue at <https://github.com/tidymodels/corrr/issues>.
```



```
wrappedtools::ggcormat(cor_mat = cor_out$corout,  
maxpoint = 15)
```



```
wrappedtools::ggcormat(cor_mat = cor_out$corout,
                      p_mat = cor_out$pout,
                      maxpoint = 20)
```



```
GGally::ggpairs(rawdata |> select(contains("BP"))))
```

Warning: Removed 1 row containing non-finite outside the scale range  
(`stat\_density()`).

Warning: Removing 1 row that contained a missing value

Warning: Removed 4 rows containing missing values  
Removed 4 rows containing missing values

Warning: Removed 1 row containing missing values or values outside the scale range  
(`geom\_point()`).

Warning: Removed 1 row containing non-finite outside the scale range  
(`stat\_density()`).

Warning: Removed 4 rows containing missing values  
Removed 4 rows containing missing values

Warning: Removed 4 rows containing missing values or values outside the scale range  
(`geom\_point()`).

Removed 4 rows containing missing values or values outside the scale range  
(`geom\_point()`).

Warning: Removed 4 rows containing non-finite outside the scale range  
(`stat\_density()`).

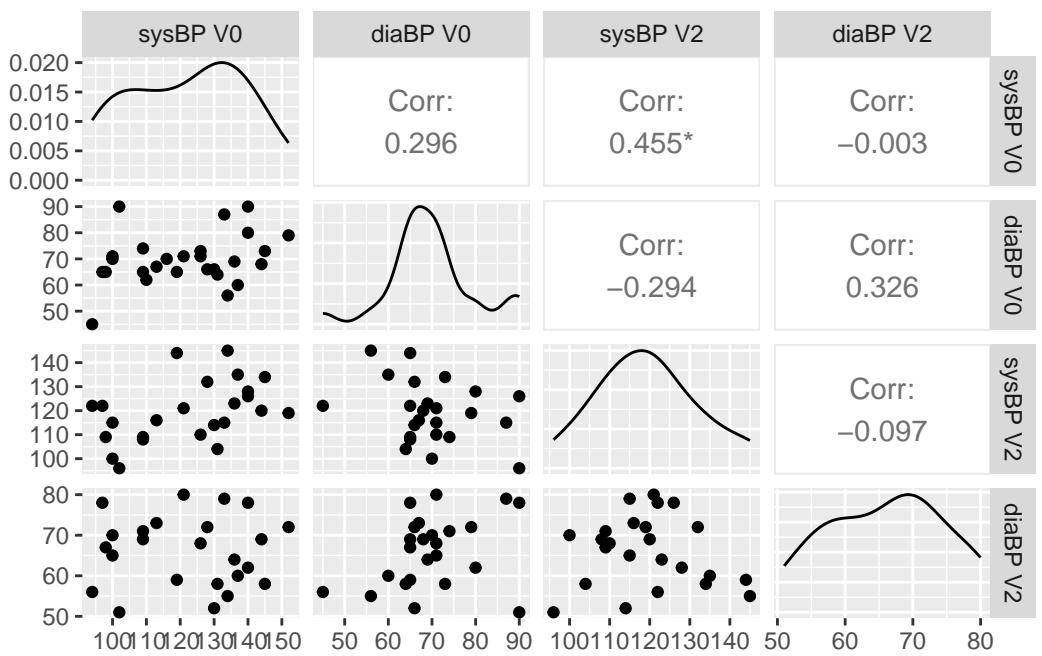
Warning: Removed 4 rows containing missing values

Warning: Removed 4 rows containing missing values or values outside the scale range  
(`geom\_point()`).

Removed 4 rows containing missing values or values outside the scale range  
(`geom\_point()`).

Removed 4 rows containing missing values or values outside the scale range  
(`geom\_point()`).

Warning: Removed 4 rows containing non-finite outside the scale range  
(`stat\_density()`).



# 15 Intro to linear models

In this chapter, linear models (including linear regression and ANOVA) will be introduced. Output is not optimized for print, but rather for interactive use.

## 15.1 Setup

All packages necessary will be invoked by `p_load`. Packages with only a single function call or potential for name conflicts can be unloaded, this way we still checked for their existence and installed them if need be.

```
pacman::p_load(conflicted,wrappedtools,car,nlme,broom, flextable,
                 multcomp,tidyverse,foreign,DescTools, ez,
                 ggbeeswarm,
                 lme4, nlme,merTools,
                 easystats, patchwork,here)#conflicted,
# rayshader,av)
# pacman::p_unload(DescTools, foreign)
# conflict_scout()
conflicts_prefer(dplyr::select,
                  dplyr::filter,
                  modelbased::standardize)
```

```
[conflicted] Will prefer dplyr::select over any other package.
[conflicted] Will prefer dplyr::filter over any other package.
[conflicted] Will prefer modelbased::standardize over any other package.
```

```
base_dir <- here::here()
```

## 15.2 Import / Preparation

Data are read from an SPSS file. Numeric column Passage is mutated into a factor as `Passage_F`, this is necessary for group comparisons in ANOVA. The call to `here()` expands the path to a file from the project directory to the full system path.

```

rawdata<-foreign::read.spss(file=here('data/Zellbeads.sav'),
                             use.value.labels=T,to.data.frame=T) |>
  as_tibble() |>
  dplyr::select(-ZahlZellen) |>
  rename(Growth=Wachstum,Treatment=Bedingung) |>
  mutate(Passage_F=factor(Passage),
         Treatment=fct_recode(Treatment,
                               Control="Kontrolle"))

```

Zurückkodierung von CP1252

## 15.3 Graphical exploration

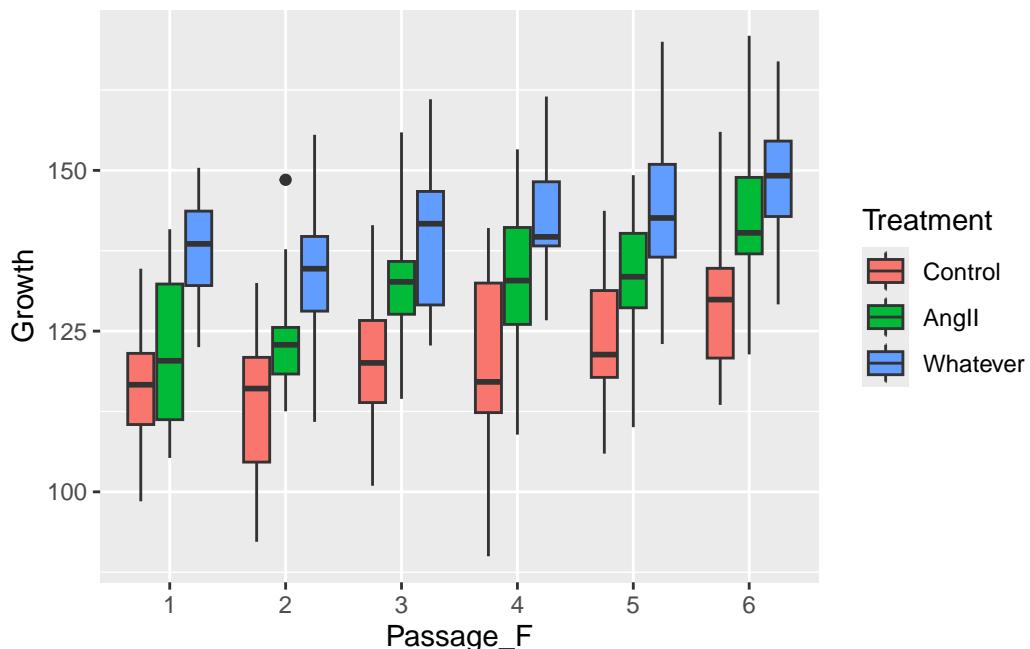
First impression of the data will be attempted by grouped boxplot, followed by interaction plots, both as basic and ggplot with variations.

```

ggplot(rawdata,aes(Passage_F,Growth, fill=Treatment))+  

  geom_boxplot(coef=3)

```



```

with(rawdata, interaction.plot(  

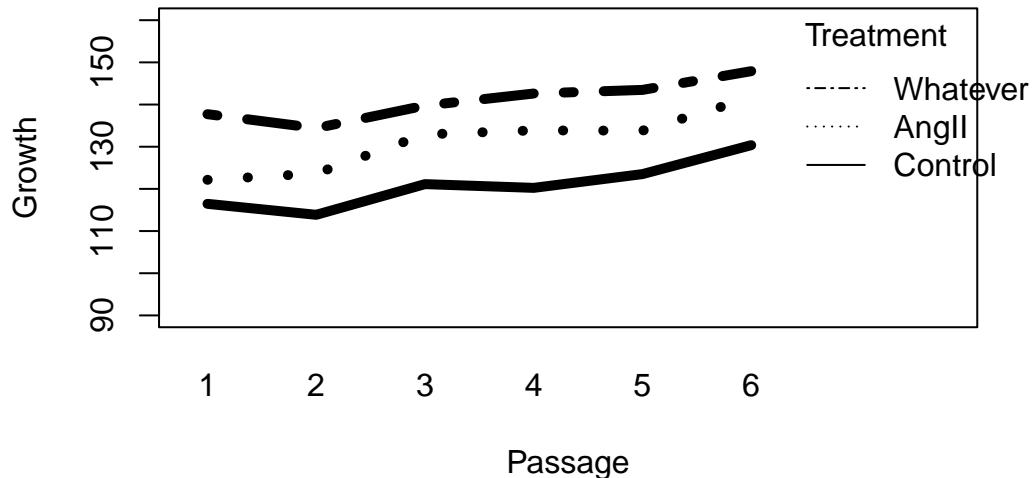
  x.factor=Passage, trace.factor=Treatment, response=Growth,  

  ylim = c(90, 160), lty = c(1,3,12), lwd = 5,  

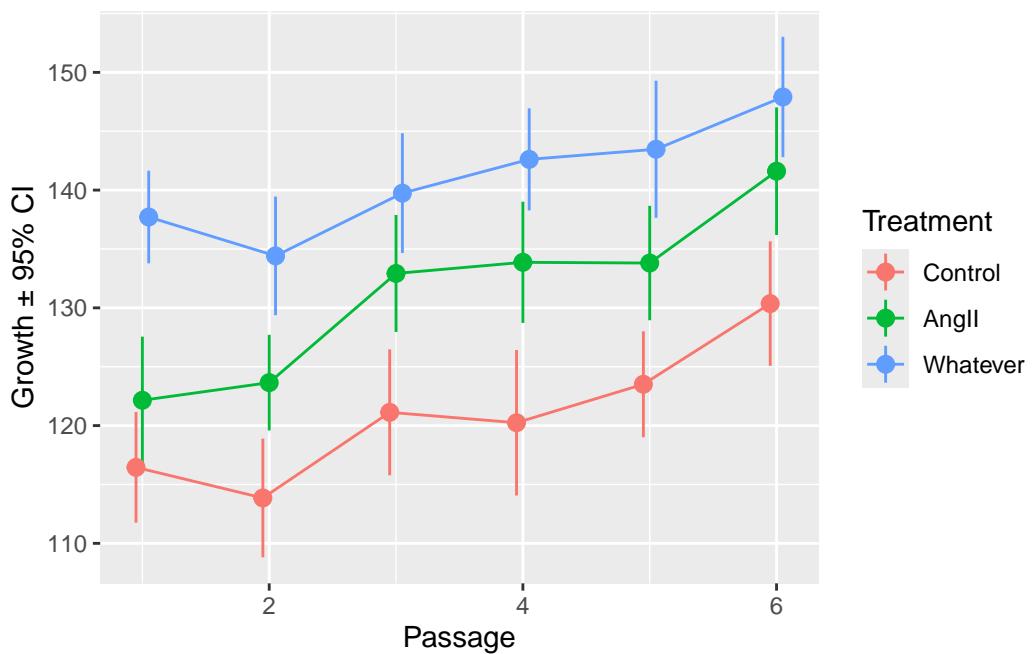
  ylab = "Growth", xlab = "Passage",

```

```
trace.label = "Treatment"))
```

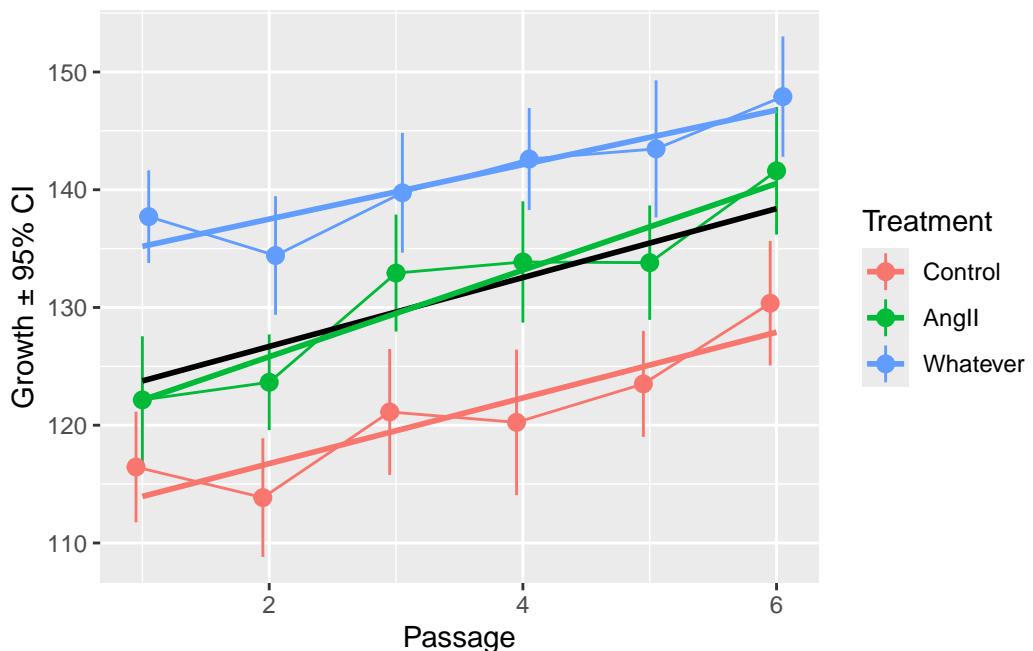


```
# p1<-ggplot(rawdata,aes(x=Passage,y=Growth))+  
#   stat_summary(geom='line',fun='mean',aes(color=Treatment))+  
#   stat_summary(geom='line',fun='mean')  
p1<-ggplot(rawdata,aes(x=Passage,y=Growth))+  
  stat_summary(geom='line',fun='mean',  
               aes(color=Treatment),  
               position=position_dodge(width = .15))+  
  stat_summary(aes(color=Treatment),  
               position=position_dodge(width = .15),  
               fun.data = "mean_cl_normal")  
  ylab('Growth \u00b1 95% CI')  
p1
```



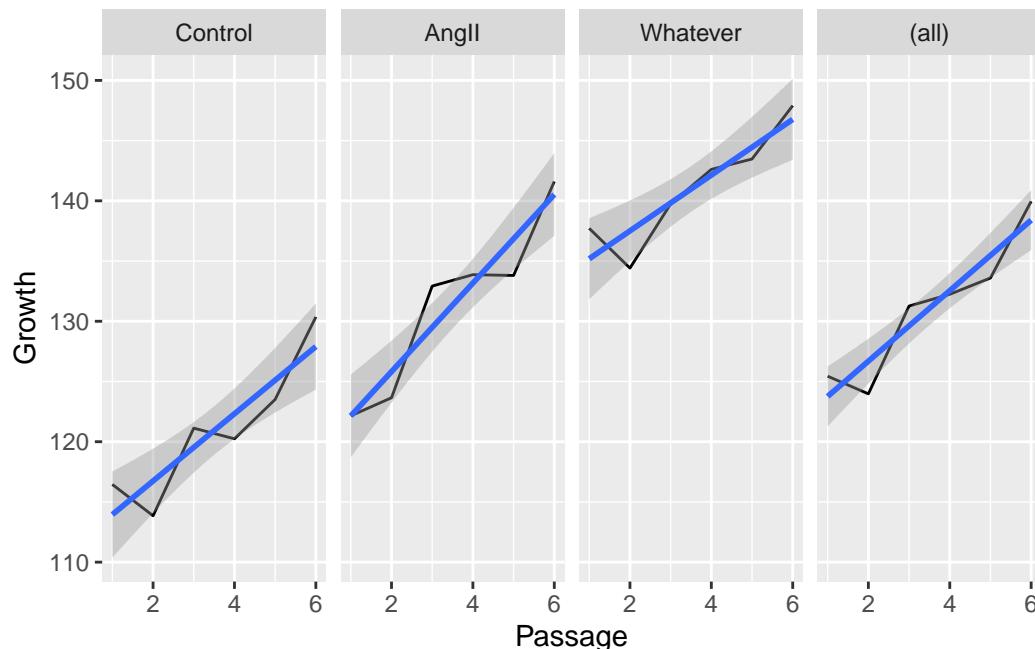
```
p1+geom_smooth(method='lm',color='black',se=F)+  
  geom_smooth(method='lm',aes(color=Treatment),se=F)
```

```
`geom_smooth()` using formula = 'y ~ x'  
`geom_smooth()` using formula = 'y ~ x'
```



```
ggplot(rawdata,aes(x=Passage,y=Growth))+  
  stat_summary(geom='line',fun='mean')+  
  geom_smooth(method='lm')+  
  facet_grid(cols = vars(Treatment), margins=T)
```

`geom\_smooth()` using formula = 'y ~ x'



## 15.4 Linear Models

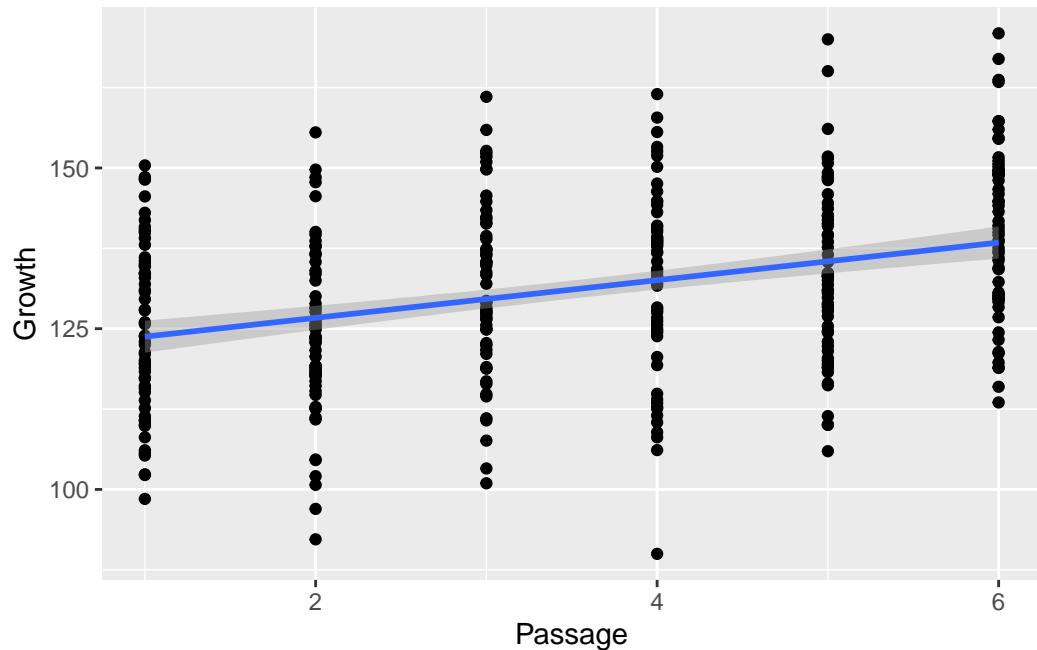
### 15.4.1 Linear regression

We will analyse the relation between dependent variable (DV) Growth and CONTINUOUS NUMERICAL independent variable (IV) Passage. This is traditionally called regression, here it is rather a regression-like linear model.

#### 15.4.1.1 Graphical exploration

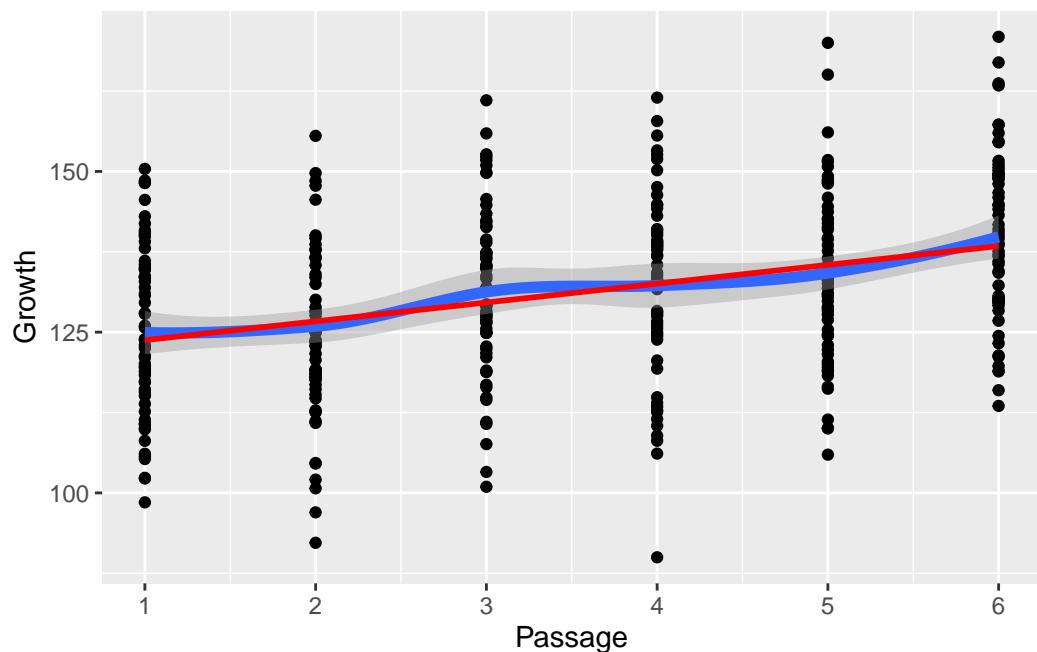
```
ggplot(rawdata,aes(Passage,Growth))+  
  geom_point()+  
  geom_smooth(method=lm)
```

```
`geom_smooth()` using formula = 'y ~ x'
```



```
ggplot(rawdata,aes(Passage,Growth))+  
  geom_point() +  
  scale_x_continuous(breaks=seq(0,10,1))+  
  geom_smooth(linewidth=2)+  
  geom_smooth(method=lm,se=F,color='red')
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'  
`geom_smooth()` using formula = 'y ~ x'
```



#### 15.4.1.2 Modelling

This takes 2 steps, building the model and computing p-values.

```
# model
(regressionOut<-lm(Growth~Passage,data=rawdata))
```

Call:  
`lm(formula = Growth ~ Passage, data = rawdata)`

Coefficients:  
`(Intercept) Passage`  
`120.834 2.927`

```
regressionOut$coefficients
```

(Intercept)	Passage
120.833844	2.927085

```
# model and p.value for slope, not recommended
tidy(regressionOut)
```

```
# A tibble: 2 x 5
  term      estimate std.error statistic   p.value
  <chr>     <dbl>     <dbl>     <dbl>     <dbl>
1 (Intercept) 121.      1.63      74.2  1.77e-219
2 Passage       2.93      0.418      7.00  1.26e- 11
```

```
# computation of SSQs and p-values, use this!
(anova_out<-anova(regressionOut))
```

### Analysis of Variance Table

Response: Growth

	Df	Sum Sq	Mean Sq	F value	Pr(>F)						
Passage	1	8996	8996.2	49.022	1.257e-11 ***						
Residuals	358	65698	183.5								
---											
Signif. codes:	0	'***'	0.001	'**'	0.01	'*'	0.05	'.'	0.1	' '	1

```
anova_out$`Pr(>F)` #|> na.omit()
```

```
[1] 1.257266e-11           NA
```

```
tidy(anova_out)
```

```
# A tibble: 2 x 6
  term      df    sumsq meansq statistic   p.value
  <chr>     <int>  <dbl>  <dbl>     <dbl>     <dbl>
1 Passage      1  8996.  8996.      49.0  1.26e-11
2 Residuals   358 65698. 184.       NA     NA
```

```
# summary(regressionOut)
# str(regressionOut)
```

#### 15.4.1.3 Adjusting

To take out the variance due to Passage effects, we can use the residuals and shift them to the original mean:

```
rawdata <-  
  mutate(rawdata,  
    growthAdj = regressionOut$residuals+mean(Growth))
```

```
summarise(rawdata,  
  across(contains('growth'),  
    ~meansd(.x,roundDig =4)))
```

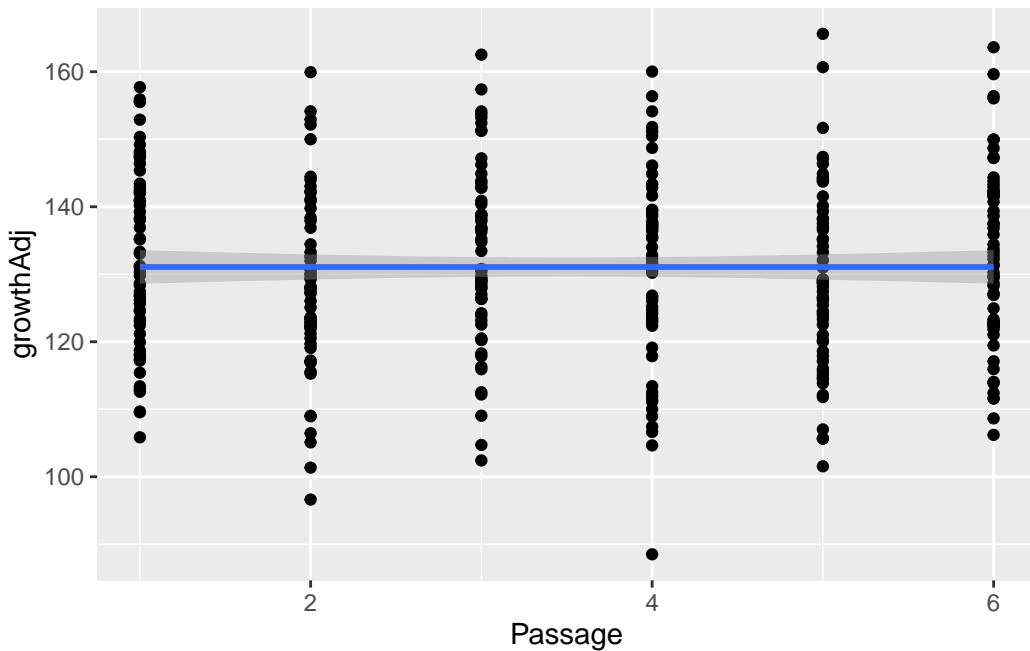
```
# A tibble: 1 x 2  
Growth      growthAdj  
<chr>       <chr>  
1 131.1 ± 14.4 131.1 ± 13.5
```

```
summarise(rawdata,  
  across(contains('growth'),  
    ~meanse(.x,roundDig =4)))
```

```
# A tibble: 1 x 2  
Growth      growthAdj  
<chr>       <chr>  
1 131.1 ± 0.8 131.1 ± 0.7
```

```
ggplot(rawdata,aes(Passage,growthAdj))+  
  geom_point() +  
  geom_smooth(method = 'lm')
```

```
`geom_smooth()` using formula = 'y ~ x'
```



```
lm(growthAdj~Passage,data=rawdata) |> tidy()
```

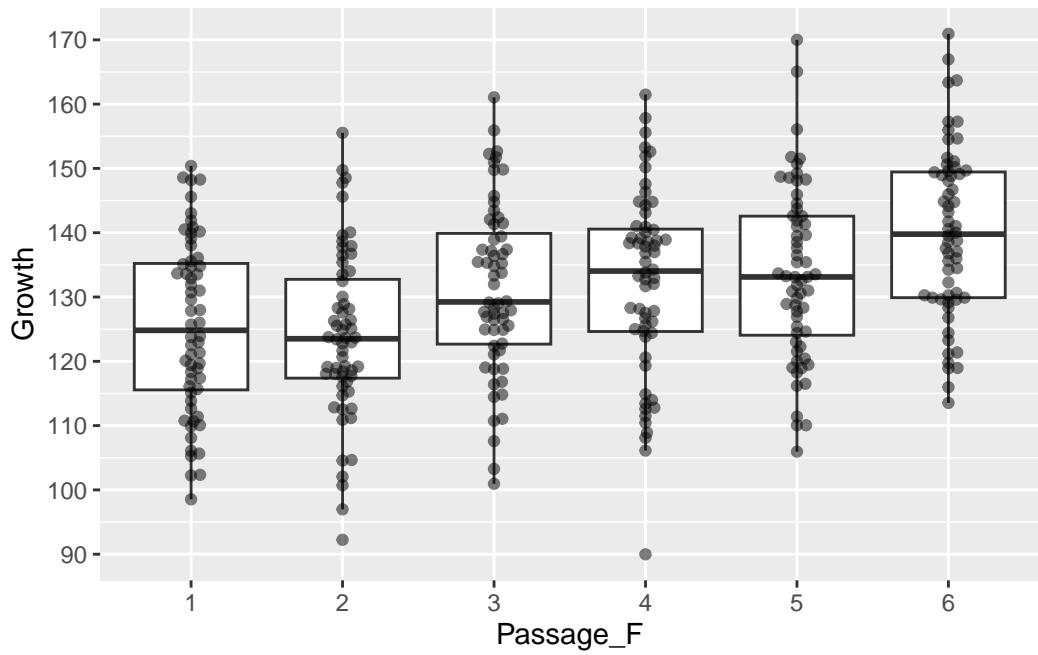
```
# A tibble: 2 x 5
  term      estimate std.error statistic   p.value
  <chr>     <dbl>     <dbl>     <dbl>     <dbl>
1 (Intercept) 1.31e+ 2     1.63     8.05e+ 1 2.04 e-231
2 Passage     6.80e-15    0.418    1.63e-14 1.000e+ 0
```

### 15.4.2 ANOVA

In the linear regression, we had Passage as a continuous IV, estimating a global ‘universal’ effect supposed to be constant. Now we look at Passage\_F and thus model a discrete IV, allowing for specific effects, and thereby comparing means between groups. So this is an ANOVA-like linear model.

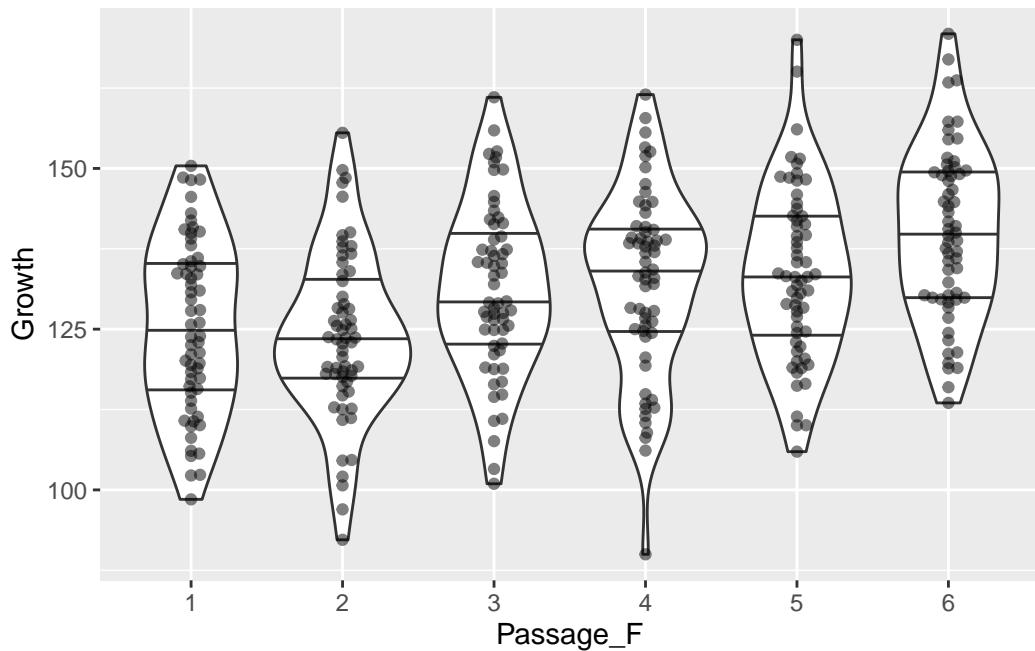
#### 15.4.2.1 Graphical exploration

```
ggplot(rawdata,aes(x = Passage_F, y = Growth))+  
  geom_boxplot(outlier.alpha = 0)+  
  geom_beeswarm(alpha=.5)+  
  scale_y_continuous(breaks=seq(0,1000,10))
```



```
ggplot(rawdata,aes(x = Passage_F, y = Growth))+  
  geom_violin(draw_quantiles = c(.25,.5,.75))+  
  geom_beeswarm(alpha=.5)
```

Warning: The `draw\_quantiles` argument of `geom\_violin()` is deprecated as of ggplot2 4.0.0.  
 i Please use the `quantiles.linetype` argument instead.



#### 15.4.2.2 Modelling

```
(AnovaOut<-lm(Growth~Passage_F, data=rawdata))
```

Call:  
`lm(formula = Growth ~ Passage_F, data = rawdata)`

Coefficients:

	(Intercept)	Passage_F2	Passage_F3	Passage_F4	Passage_F5	Passage_F6
	125.440	-1.467	5.824	6.801	8.156	14.520

```
tidy(AnovaOut)
```

```
# A tibble: 6 x 5
  term      estimate std.error statistic p.value
  <chr>      <dbl>    <dbl>     <dbl>   <dbl>
1 (Intercept) 125.       1.74     71.9  2.20e-213
2 Passage_F2  -1.47     2.47    -0.595 5.52e- 1
3 Passage_F3   5.82     2.47     2.36  1.87e- 2
4 Passage_F4   6.80     2.47     2.76  6.11e- 3
5 Passage_F5   8.16     2.47     3.31  1.04e- 3
6 Passage_F6  14.5      2.47     5.89  9.03e- 9
```

```
# summary(AnovaOut)
(t <- anova(AnovaOut))
```

Analysis of Variance Table

Response: Growth

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Passage_F	5	10134	2026.71	11.113	5.852e-10 ***
Residuals	354	64561	182.38		

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

```
t$`Pr(>F)`
```

```
[1] 5.851856e-10           NA
```

```
tidy(t)
```

```
# A tibble: 2 x 6
  term      df    sumsq   meansq statistic   p.value
  <chr>     <int>  <dbl>    <dbl>     <dbl>      <dbl>
1 Passage_F     5 10134.  2027.     11.1  5.85e-10
2 Residuals    354 64561. 182.      NA     NA
```

### 15.4.2.3 Post-hoc analyses

The p-value from our model only tests the global Null hypothesis of no differences between any group (all means are the same / all groups come from the same population). Post-hoc tests are used to figure out which groups are different. Those tests need to take multiple testing into account. Try to limit selection of tests!

```
# possible in a loop, but nominal p
t.test(rawdata$Growth[which(rawdata$Passage==1)],
       rawdata$Growth[which(rawdata$Passage==2)],
       var.equal = T)
```

Two Sample t-test

```
data: rawdata$Growth[which(rawdata$Passage == 1)] and rawdata$Growth[which(rawdata$Passag
```

```
t = 0.60679, df = 118, p-value = 0.5452
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-3.321297  6.255936
sample estimates:
mean of x mean of y
125.4396 123.9723
```

```
# all pairwise group combinations
pt_out<-pairwise.t.test(x=rawdata$Growth,
                         g=rawdata$Passage_F,
                         p.adjust.method='none')
pt_out
```

```
Pairwise comparisons using t tests with pooled SD

data: rawdata$Growth and rawdata$Passage_F

  1      2      3      4      5 
2 0.55215 -      -      -      - 
3 0.01871 0.00331 -      -      - 
4 0.00611 0.00088 0.69214 -      - 
5 0.00104 0.00011 0.34487 0.58296 - 
6 9e-09   3e-10   0.00048 0.00189 0.01025 

P value adjustment method: none
```

```
pairwise.t.test(x=rawdata$Growth,g=rawdata$Passage_F,
                 p.adjust.method='fdr')
```

```
Pairwise comparisons using t tests with pooled SD

data: rawdata$Growth and rawdata$Passage_F

  1      2      3      4      5 
2 0.62460 -      -      -      - 
3 0.02552 0.00621 -      -      - 
4 0.01018 0.00259 0.69214 -      - 
5 0.00259 0.00057 0.43109 0.62460 - 
6 6.8e-08 4.5e-09 0.00178 0.00405 0.01538 

P value adjustment method: fdr
```

```
pairwise.t.test(x=rawdata$Growth,g=rawdata$Passage_F,
                 p.adjust.method='bonferroni')
```

Pairwise comparisons using t tests with pooled SD

```
data: rawdata$Growth and rawdata$Passage_F
```

	1	2	3	4	5
2	1.0000	-	-	-	-
3	0.2807	0.0497	-	-	-
4	0.0917	0.0133	1.0000	-	-
5	0.0155	0.0017	1.0000	1.0000	-
6	1.4e-07	4.5e-09	0.0071	0.0283	0.1538

P value adjustment method: bonferroni

```
# comparison against reference group 1
pt_out$p.value[,1]
```

	2	3	4	5	6
5	5.521460e-01	1.871115e-02	6.110172e-03	1.036173e-03	9.031123e-09

```
# comparison against reference group 6
pt_out$p.value[5,]
```

	1	2	3	4	5
9	0.031123e-09	3.001066e-10	4.757018e-04	1.889098e-03	1.025037e-02

```
# comparison for selection
c(pt_out$p.value[1,1],pt_out$p.value[3,2],
  pt_out$p.value[5,1])
```

[1] 5.521460e-01 8.842382e-04 9.031123e-09

```
# comparison against next level
diag(pt_out$p.value)
```

[1] 0.55214600 0.00331248 0.69214393 0.58295615 0.01025037

```
# adjusting for multiple testing for selected comparisons  
p.adjust(diag(pt_out$p.value),method='fdr')
```

```
[1] 0.69214393 0.01656240 0.69214393 0.69214393 0.02562592
```

```
formatP(p.adjust(pt_out$p.value[,1],method='fdr'))
```

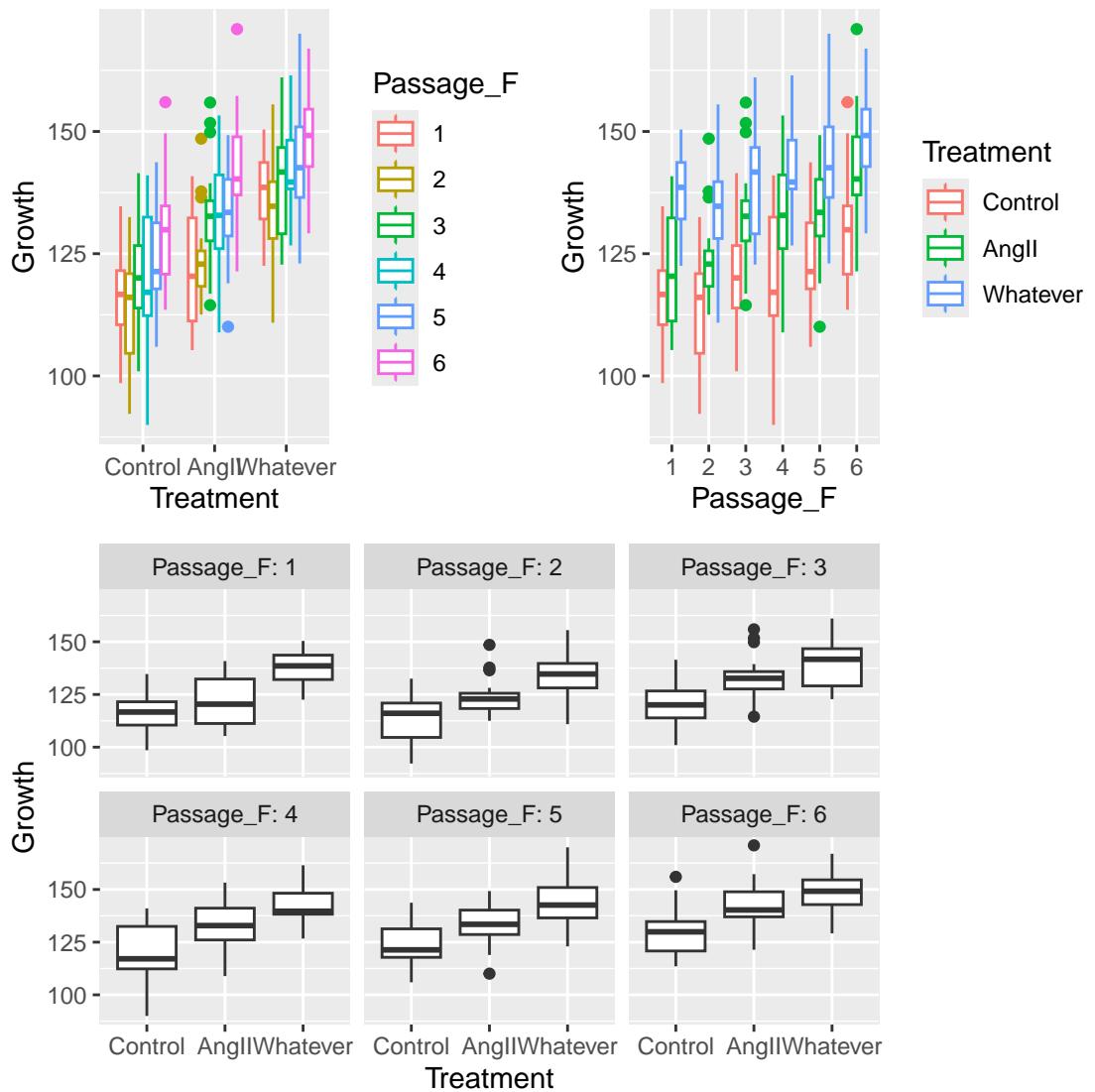
```
[1] "0.552" "0.023" "0.010" "0.003" "0.001"
```

### 15.4.3 LM with continuous AND categorical IV

Traditionally you may think of *regression OR ANOVA*, but they are no different and can be combined. This is called a general linear model. Multivariable models may contain interactions between independent variables V  $IV1*IV2$ .

#### 15.4.3.1 Graphical exploration

```
p0 <- ggplot(rawdata,aes(Treatment,Growth))+  
  geom_boxplot()  
p1 <- ggplot(rawdata,aes(Treatment,Growth, color=Passage_F))+  
  geom_boxplot()  
p2 <- ggplot(rawdata,aes(color=Treatment,Growth, x=Passage_F))+  
  geom_boxplot()  
p3 <- ggplot(rawdata,aes(Treatment,Growth))+  
  geom_boxplot()  
  facet_wrap(facets = vars(Passage_F), labeller='label_both')  
# from patchwork  
(p1+p2)/p3
```



### 15.4.3.2 Modelling

Models with (\*) and without (+) interaction are build and tested.

```
lmOut_interaction<-lm(Growth~Passage*Treatment,data=rawdata)
Anova(lmOut_interaction,type = 3)
```

Anova Table (Type III tests)

Response: Growth

Sum Sq	Df	F value	Pr(>F)
--------	----	---------	--------

```

(Intercept)      285160     1 2448.5613 < 2.2e-16 ***
Passage          2723      1   23.3855 1.981e-06 ***
Treatment        5635      2   24.1924 1.419e-10 ***
Passage:Treatment 335      2    1.4376    0.2389
Residuals       41227    354
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

```

#
lmOut_additive<-lm(Growth~Passage+Treatment,data=rawdata)
Anova_out <- Anova(lmOut_additive,type=2)
Anova_out$`Pr(>F)`
```

```
[1] 7.051564e-17 4.006053e-36           NA
```

```
tidy(Anova_out)
```

```
# A tibble: 3 x 5
  term      sumsq    df statistic p.value
  <chr>     <dbl> <dbl>     <dbl>     <dbl>
1 Passage    8996.     1      77.1  7.05e-17
2 Treatment  24137.    2     103.   4.01e-36
3 Residuals  41562.   356     NA     NA
```

```

# for comparison, here is the univariable model
lmOut_uni<-lm(Growth~Treatment,data=rawdata)
aOut<-Anova(lmOut_uni,type=3)
a_uni <- anova(lmOut_uni)
a_uni$`Pr(>F)`
```

```
[1] 5.549803e-31           NA
```

### 15.4.3.3 Post-hoc analyses

For multivariable models, pairwise.t.test() is not appropriate, Dunnet or Tukey tests (depending on hypothesis) are typical solutions.

```

glht_out <-
  summary(glht(model=lmOut_additive,
               linfct=mcp(Treatment='Dunnett'))))
glht_out$test$pvalues
```

```
[1] 1.316836e-12 5.551115e-16  
attr(,"error")  
[1] 1e-15
```

```
tidy(glht_out) |>  
  select(-null.value)
```

```
# A tibble: 2 x 6  
  term      contrast       estimate std.error statistic adj.p.value  
  <chr>    <chr>           <dbl>     <dbl>     <dbl>     <dbl>  
1 Treatment AngII - Control     10.4      1.39      7.46   1.32e-12  
2 Treatment Whatever - Control  20.1      1.39     14.4     5.55e-16
```

```
summary(glht(model=lmOut_additive,  
            linfct=mcp(Treatment='Tukey')))
```

### Simultaneous Tests for General Linear Hypotheses

#### Multiple Comparisons of Means: Tukey Contrasts

```
Fit: lm(formula = Growth ~ Passage + Treatment, data = rawdata)  
  
Linear Hypotheses:  
Estimate Std. Error t value Pr(>|t|)  
AngII - Control == 0     10.409     1.395    7.462  <1e-10 ***  
Whatever - Control == 0  20.052     1.395   14.375  <1e-10 ***  
Whatever - AngII == 0    9.643     1.395    6.913  <1e-10 ***  
---  
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
(Adjusted p values reported -- single-step method)
```

```
DescTools:::DunnettTest(Growth~Passage_F,data=rawdata)
```

```
Dunnett's test for comparing several treatments with a control :  
95% family-wise confidence level
```

```
$`1`  
  diff      lwr.ci      upr.ci      pval
```

```

2-1 -1.467320 -7.6899251 4.755285 0.9648
3-1 5.824059 -0.3985468 12.046664 0.0752 .
4-1 6.801105 0.5784996 13.023710 0.0265 *
5-1 8.156143 1.9335375 14.378748 0.0049 **
6-1 14.520106 8.2975011 20.742712 3.2e-08 ***

---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

```
DescTools:::DunnettTest(Growth~Treatment,data=rawdata)
```

```

Dunnett's test for comparing several treatments with a control :
95% family-wise confidence level

$Control
      diff     lwr.ci    upr.ci    pval
AngII-Control 10.40895 6.996811 13.82109 1e-10 ***
Whatever-Control 20.05199 16.639849 23.46413 <2e-16 ***

---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

```
pairwise.t.test(rawdata$Growth,rawdata$Treatment,p.adjust.method = 'n')
```

```

Pairwise comparisons using t tests with pooled SD

data: rawdata$Growth and rawdata$Treatment

          Control AngII
AngII      5.1e-11 -
Whatever < 2e-16 1.0e-09

P value adjustment method: none

```

```
# mean(rawdata$Growth[which(rawdata$Passage==1 &
                           # rawdata$Treatment=='Control')])  
anova_out$'Pr(>F)'
```

```
[1] 1.257266e-11      NA
```

```
#aOut$`Sum Sq`  
summary(lmOut_additive)
```

Call:  
lm(formula = Growth ~ Passage + Treatment, data = rawdata)

Residuals:

Min	1Q	Median	3Q	Max
-32.407	-7.793	-0.281	7.255	32.283

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	110.6802	1.5280	72.432	< 2e-16 ***
Passage	2.9271	0.3334	8.778	< 2e-16 ***
TreatmentAngII	10.4089	1.3949	7.462	6.59e-13 ***
TreatmentWhatever	20.0520	1.3949	14.375	< 2e-16 ***
---				

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.8 on 356 degrees of freedom  
Multiple R-squared: 0.4436, Adjusted R-squared: 0.4389  
F-statistic: 94.6 on 3 and 356 DF, p-value: < 2.2e-16

```
(result<-tibble(predictor=rownames(aOut),  
p=formatP(aOut$'Pr(>F)',ndigits=5)))
```

```
# A tibble: 3 x 2  
predictor p  
<chr> <chr>  
1 (Intercept) "0.00001"  
2 Treatment "0.00001"  
3 Residuals ""
```

```
broom::tidy(aOut)
```

```
# A tibble: 3 x 5  
term      sumsq    df statistic   p.value  
<chr>     <dbl> <dbl>     <dbl>      <dbl>  
1 (Intercept) 1754743.     1    12391.  2.91e-279  
2 Treatment    24137.     2      85.2  5.55e- 31  
3 Residuals    50558.   357       NA      NA
```

## 15.5 compare\_n\_numvars() as reporting option

```
raw_out <- compare_n_numvars(rawdata,
                                dep_vars = c("Growth", "growthAdj"),
                                indep_var = "Treatment",
                                gaussian = TRUE, add_n = TRUE)

raw_out$results |>
  select(Variiable, `p(ANOVA)` ~ multivar_p, contains("fn")) |>
  rename_with(~str_replace_all(.x,
                               c(" fn" = "", "Treatment" = ""))) |>
  flextable() |>
  add_footer_lines("Symbols b,c indicate significance in post-hoc test vs. group 2 or 3")
  set_table_properties(width=1, layout="autofit")
```

Variable	p(ANOVA)	Control	AngII	Whatever
Growth	0.001	121 ± 12 [n = 120] bc	131 ± 12 [n = 120] c	141 ± 11 [n = 120]
growthAdj	0.001	121 ± 11 [n = 120] bc	131 ± 11 [n = 120] c	141 ± 11 [n = 120]

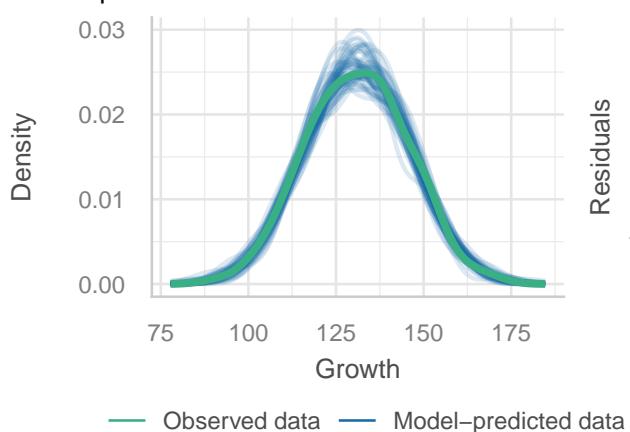
Symbols b,c indicate significance in post-hoc test vs. group 2 or 3

## 15.6 Model exploration with package performance

```
# x11() #interactive only!
# quartz() for mac
# from package performance
check_model(lm0ut_additive)
```

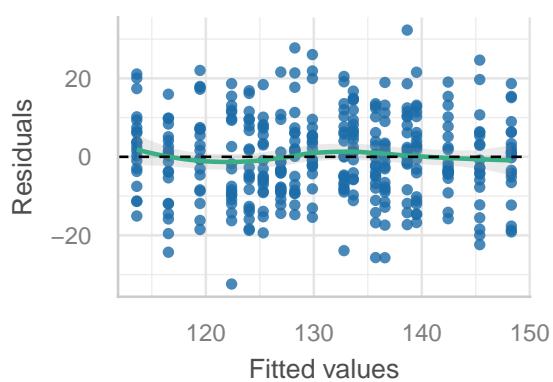
### Posterior Predictive Check

Model-predicted lines should resemble observed data



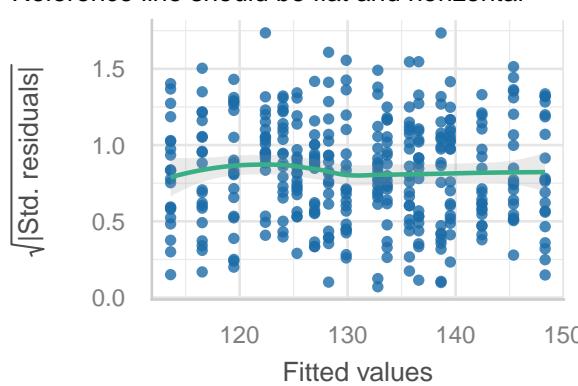
### Linearity

Reference line should be flat and horizontal



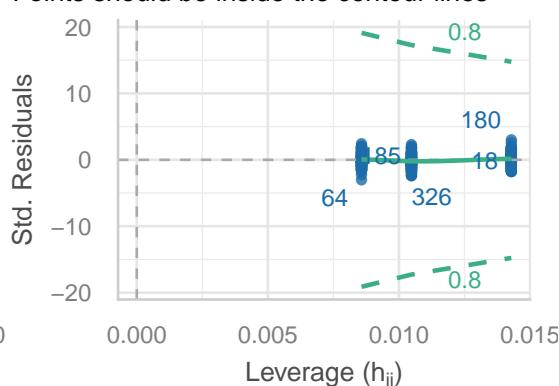
### Homogeneity of Variance

Reference line should be flat and horizontal



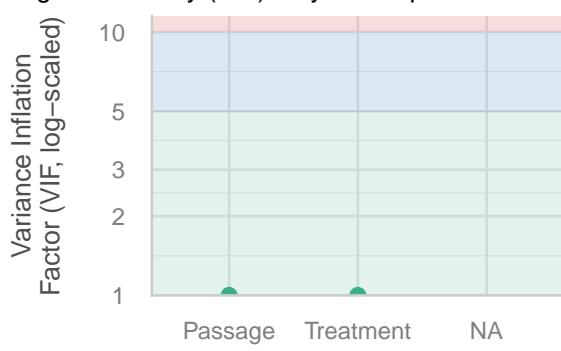
### Influential Observations

Points should be inside the contour lines



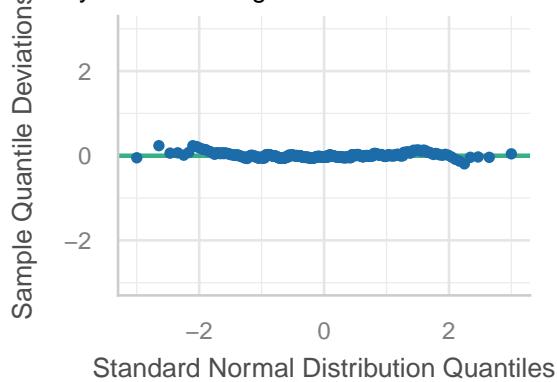
### Collinearity

High collinearity (VIF) may inflate parameter uncertainty



### Normality of Residuals

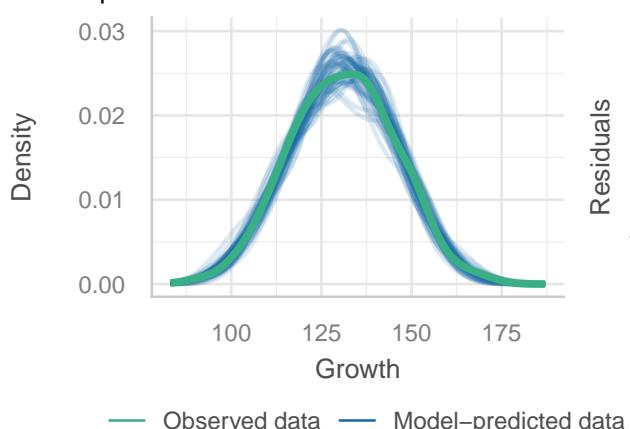
Points should fall along the line



```
check_model(lmOut_interaction)
```

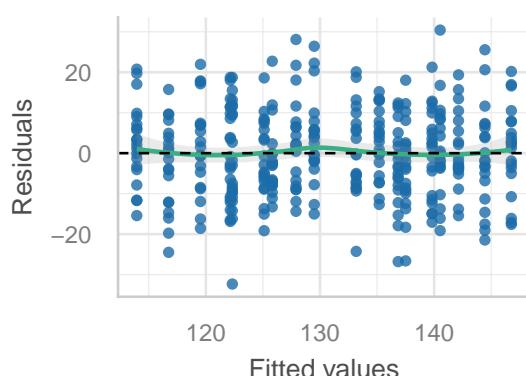
### Posterior Predictive Check

Model-predicted lines should resemble observed data



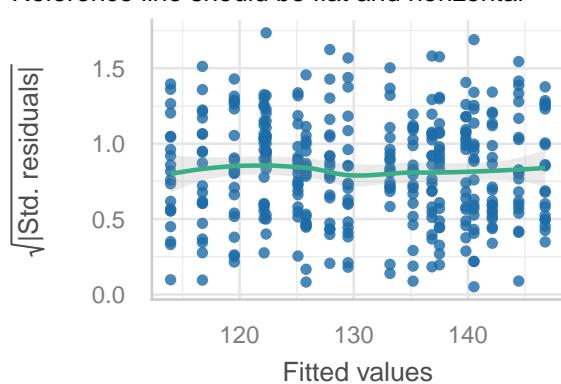
### Linearity

Reference line should be flat and horizontal



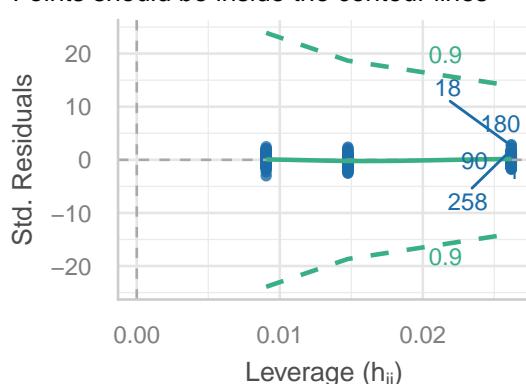
### Homogeneity of Variance

Reference line should be flat and horizontal



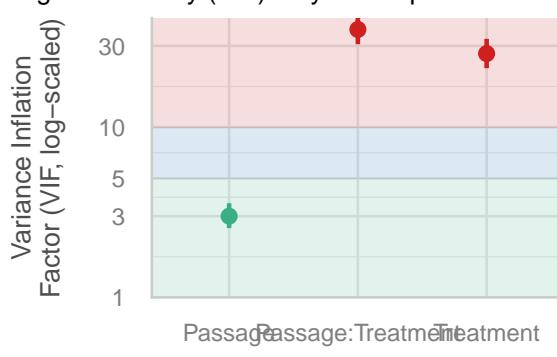
### Influential Observations

Points should be inside the contour lines



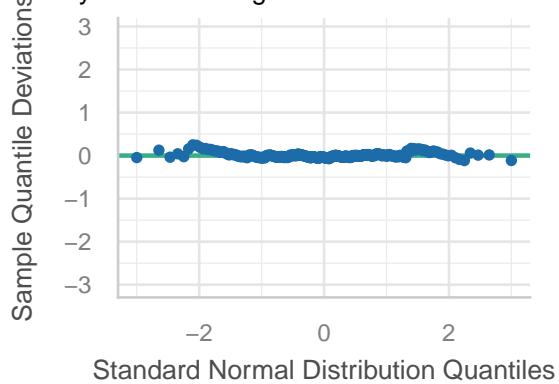
### Collinearity

High collinearity (VIF) may inflate parameter uncertainty



### Normality of Residuals

Points should fall along the line



```
# dev.off()
```

## 16 Exercise

In the penguin data, model relationship between

- DV flipper length and IV body weight
- DV bill depth and IV body weight
- DV flipper length and IV species
- DV flipper length and IV species and sex
- DV bill depth and IV species and sex
- DV flipper length and IV species and sex and body weight

When appropriate, do post-hoc analyses as well.

Vizualize results indicating significances.

# 17 Interaction in linear models

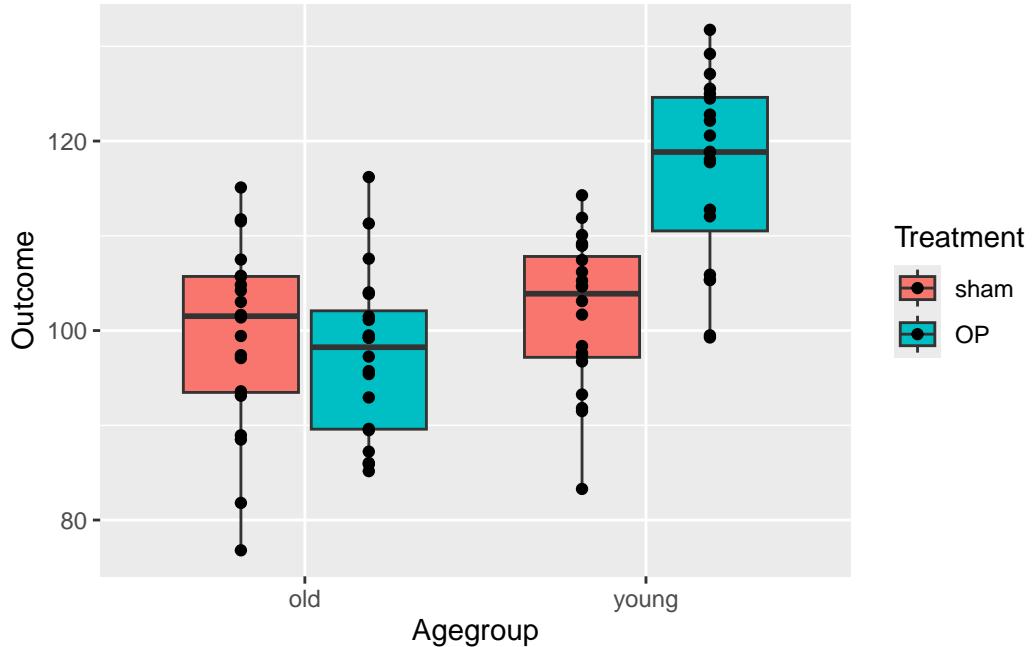
```
pacman::p_load(conflicted,tidyverse,car,multcomp,wrappedtools, broom)
conflicts_prefer(dplyr::select, dplyr::filter)
```

```
[conflicted] Will prefer dplyr::select over any other package.
[conflicted] Will prefer dplyr::filter over any other package.
```

For a better understanding, data with defined effect size and interactions will be simulated and analyzed.

## 17.1 No age effect, no treatment effect, interaction treatment\*agegroup

```
set.seed(101)
rawdata <- tibble(
  Agegroup=factor(
    rep(c('young','old'),each=40),
    levels=c('old','young')),
  Treatment=factor(
    rep(c('sham','OP'),40),
    levels = c('sham','OP')) ) |>
  mutate(Outcome=rnorm(n = 80,mean = 100,sd = 10) +
    ((Treatment=='OP')*
      (Agegroup=='young'))*20)
ggplot(rawdata,aes(x=Agegroup,y=Outcome,
                    fill=Treatment))+
  geom_boxplot()+
  geom_point(position=position_dodge(width=.75))
```



```
lmout <- lm(Outcome ~ Agegroup * Treatment,
             data = rawdata)
tidy(lmout) |>
  select(1:2)
```

```
# A tibble: 4 x 2
  term                  estimate
  <chr>                <dbl>
1 (Intercept)            99.5 
2 Agegroupyoung          2.41  
3 TreatmentOP           -2.04 
4 Agegroupyoung:TreatmentOP    17.3
```

```
anova(lmout) |> # this is WRONG!!!
tidy() |> slice(1:3) |>
  mutate(p.value=formatP(p.value, ndigits=3, mark=TRUE))
```

```
# A tibble: 3 x 6
  term                  df  sumsq meansq statistic p.value
  <chr>                <int> <dbl>  <dbl>     <dbl> <chr>
1 Agegroup               1  2443.   2443.     29.2  0.001 ***
2 Treatment              1   872.    872.      10.4  0.002 **
3 Agegroup:Treatment    1 1494.   1494.     17.9  0.001 ***
```

```
Anova(lmout,type = 3) |>
  tidy() |> slice(1:4) |>
  mutate(p.value=formatP(p.value,ndigits=3, mark=TRUE))
```

```
# A tibble: 4 x 5
  term            sumsq    df statistic p.value
  <chr>          <dbl>   <dbl>     <dbl> <chr>
1 (Intercept) 197833.     1  2368.    0.001 *** 
2 Agegroup      58.0      1   0.695  0.407 n.s.  
3 Treatment      41.7      1   0.499  0.482 n.s.  
4 Agegroup:Treatment 1494.     1   17.9   0.001 ***
```

```
summary(glht(model=lmout,
  linfct=mcp(Treatment='Tukey')))
```

Warning in mcp2matrix(model, linfct = linfct): covariate interactions found --  
default contrast might be inappropriate

Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts

Fit: lm(formula = Outcome ~ Agegroup \* Treatment, data = rawdata)

Linear Hypotheses:

	Estimate	Std. Error	t value	Pr(> t )
OP - sham == 0	-2.042	2.890	-0.707	0.482

(Adjusted p values reported -- single-step method)

```
summary(glht(model=lmout,
  linfct=mcp(Agegroup='Tukey')))
```

Warning in mcp2matrix(model, linfct = linfct): covariate interactions found --  
default contrast might be inappropriate

Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts

```
Fit: lm(formula = Outcome ~ Agegroup * Treatment, data = rawdata)
```

Linear Hypotheses:

	Estimate	Std. Error	t value	Pr(> t )
young - old == 0	2.409	2.890	0.834	0.407

(Adjusted p values reported -- single-step method)

## 17.2 Age effect, no treatment effect, interaction treatment\*agegroup

```

set.seed(1010)
rawdata <- tibble(
  Agegroup=factor(
    rep(c('young','middle','old'),each=40),
    levels=c('young','middle','old')),
  Treatment=factor(
    rep(c('sham','OP'),60),
    levels = c('sham','OP')),
  Outcome=rnorm(120,100,10)+  

    (Treatment=='OP')*  

    (Agegroup=='middle')*20+  

    (Agegroup=='old')*20
ggplot(rawdata,aes(x=Agegroup,y=Outcome,  

  fill=Treatment))+  

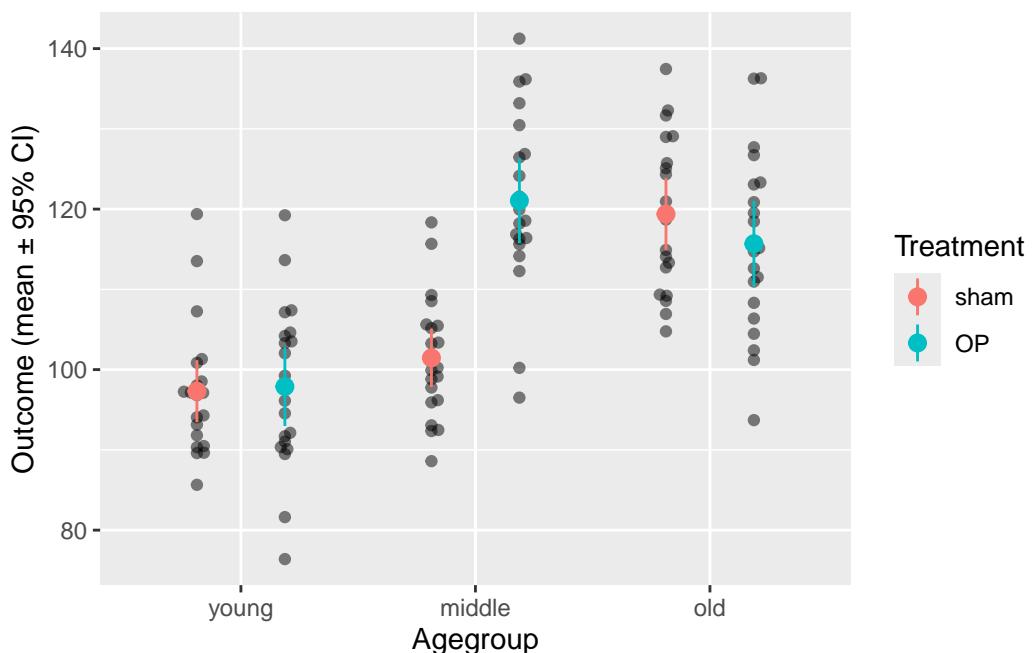
  # geom_boxplot()+
  ggbeeswarm::geom_beeswarm(dodge.width = .75, alpha=.5)+  

  stat_summary(aes(color=Treatment), fun.data= mean_cl_normal,  

  position=position_dodge(width = .75))+  

  ylab("Outcome (mean \u00b1 95% CI)")

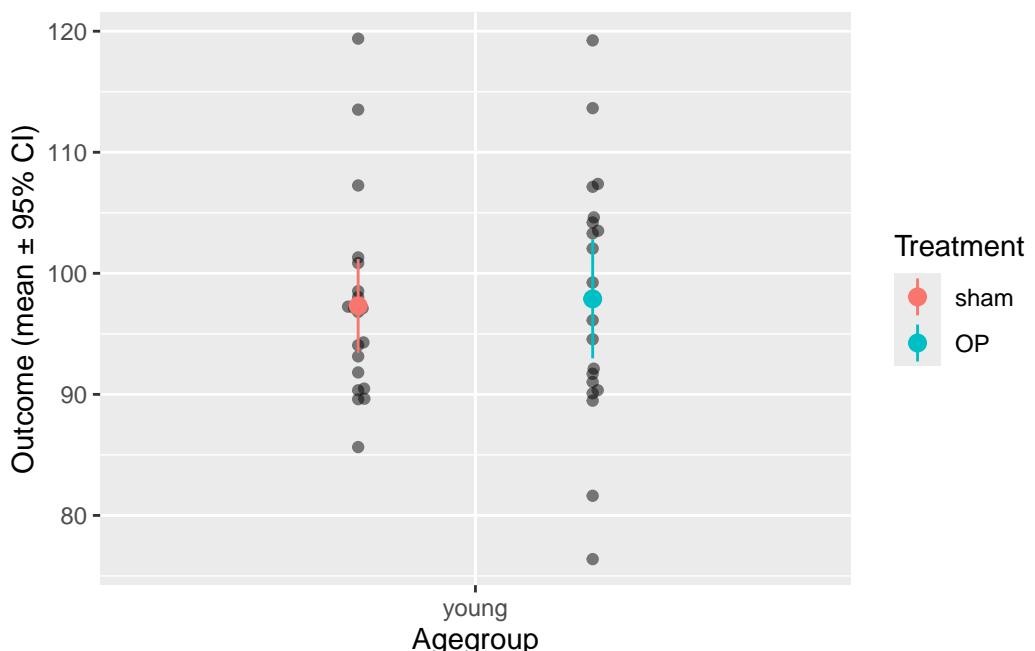
```



```

rawdata |> filter(Agegroup == "young") |>
  ggplot(aes(x=Agegroup,y=Outcome,
             fill=Treatment))+ 
  ggbeeswarm::geom_beeswarm(dodge.width = .75, alpha=.5) +
  stat_summary(aes(color=Treatment), fun.data= mean_cl_normal,
               position=position_dodge(width = .75))+
  ylab("Outcome (mean \u00b1 95% CI)")

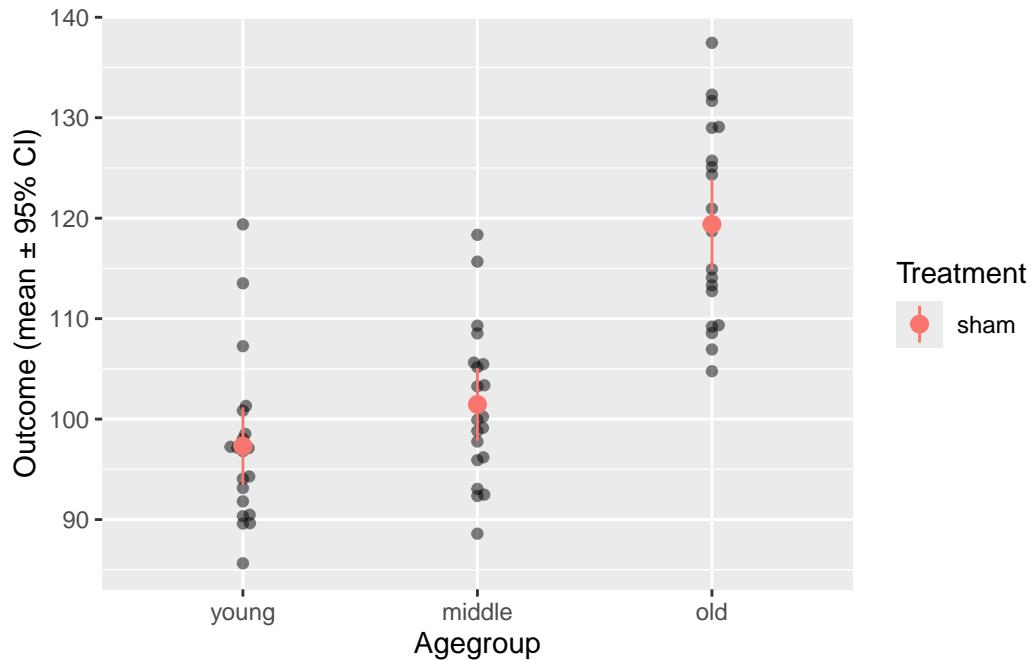
```



```

rawdata |> filter(Treatment== "sham") |>
  ggplot(aes(x=Agegroup,y=Outcome,
             fill=Treatment))+ 
  # geom_boxplot()+
  ggbeeswarm::geom_beeswarm(dodge.width = .75, alpha=.5) +
  stat_summary(aes(color=Treatment), fun.data= mean_cl_normal,
               position=position_dodge(width = .75))+
  ylab("Outcome (mean \u00b1 95% CI)")

```



```
lmout <- lm(Outcome ~ Agegroup * Treatment,
             data = rawdata)
tidy(lmout) |>
  select(1:2)
```

```
# A tibble: 6 x 2
  term                  estimate
  <chr>                <dbl>
1 (Intercept)            97.3
2 Agegroupmiddle         4.15 
3 Agegroupold            22.1 
4 TreatmentOP            0.578
5 Agegroupmiddle:TreatmentOP 19.0 
6 Agegroupold:TreatmentOP -4.28
```

```
anova(lmout) |> # this is WRONG!!!
tidy() |> slice(1:3) |>
  mutate(p.value=formatP(p.value, ndigits=3, mark=TRUE))
```

```
# A tibble: 3 x 6
  term                  df  sumsq meansq statistic p.value
  <chr>                 <int> <dbl>  <dbl>    <dbl> <chr>
1 Agegroup                  2  8310.   4155.     42.4  0.001 ***
2 Treatment                  1   904.    904.      9.22  0.003 **
```

```
3 Agegroup:Treatment      2 3074.  1537.     15.7  0.001 ***
```

```
Anova(lmout,type = 3) |>
  tidy() |> slice(1:4) |>
  mutate(p.value=formatP(p.value,ndigits=3, mark=TRUE))
```

```
# A tibble: 4 x 5
  term            sumsq    df statistic p.value
  <chr>          <dbl>   <dbl>      <dbl> <chr>
1 (Intercept)    189389.     1  1931.    0.001 ***
2 Agegroup       5504.      2   28.1     0.001 ***
3 Treatment       3.35      1   0.0341  0.854 n.s.
4 Agegroup:Treatment 3074.     2   15.7     0.001 ***
```

```
summary(glht(model=lmout,
  linfct=mcp(Treatment='Tukey')))
```

```
Warning in mcp2matrix(model, linfct = linfct): covariate interactions found --
default contrast might be inappropriate
```

#### Simultaneous Tests for General Linear Hypotheses

##### Multiple Comparisons of Means: Tukey Contrasts

```
Fit: lm(formula = Outcome ~ Agegroup * Treatment, data = rawdata)
```

##### Linear Hypotheses:

	Estimate	Std. Error	t value	Pr(> t )
OP - sham == 0	0.5784	3.1315	0.185	0.854
(Adjusted p values reported -- single-step method)				

```
summary(glht(model=lmout,
  linfct=mcp(Agegroup='Tukey')))
```

```
Warning in mcp2matrix(model, linfct = linfct): covariate interactions found --
default contrast might be inappropriate
```

Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts

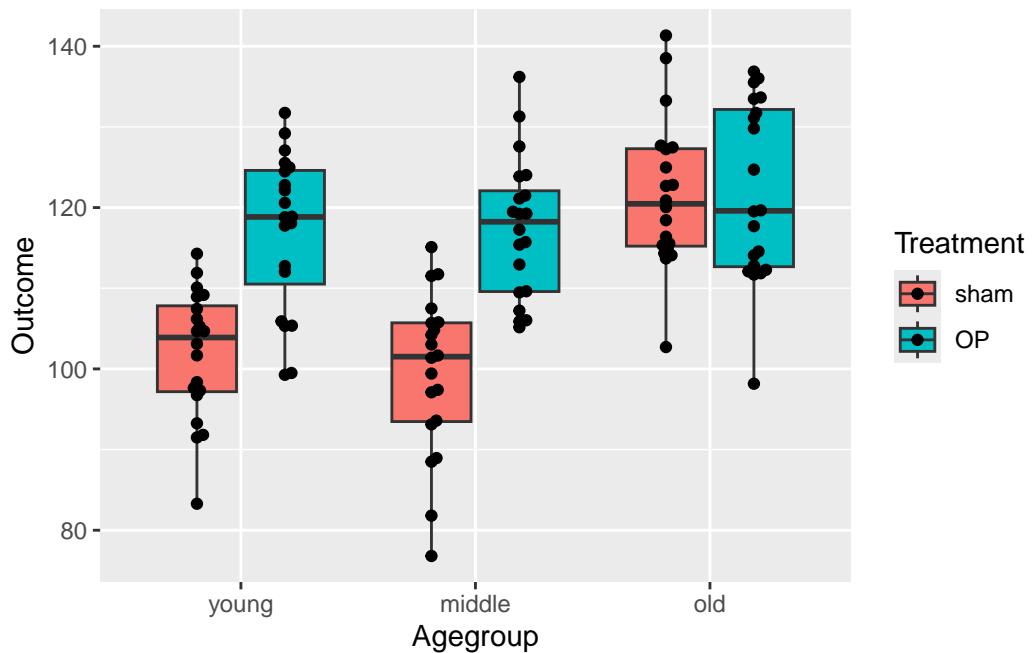
Fit: lm(formula = Outcome ~ Agegroup \* Treatment, data = rawdata)

Linear Hypotheses:

	Estimate	Std. Error	t value	Pr(> t )
middle - young == 0	4.149	3.131	1.325	0.384
old - young == 0	22.073	3.131	7.049	<1e-04 ***
old - middle == 0	17.924	3.131	5.724	<1e-04 ***
---				
Signif. codes:	0 '***'	0.001 '**'	0.01 '*'	0.05 '.'
	0.1 '	'	1	
(Adjusted p values reported -- single-step method)				

### 17.3 Age effect, treatment effect, interaction treatment\*agegroup

```
set.seed(101)
rawdata <- tibble(
  Agegroup=factor(
    rep(c('young','middle','old'),each=40),
    levels=c('young','middle','old')),
  Treatment=factor(
    rep(c('sham','OP'),60),
    levels = c('sham','OP')))) |>
  mutate(Outcome=rnorm(120,100,10) +
    (Treatment=='OP')*
    # (Agegroup %in% c('young','middle'))*
    (Agegroup!='old')*20+
    (Agegroup=='old')*20)
ggplot(rawdata,aes(x=Agegroup,y=Outcome,
                    fill=Treatment))+
  geom_boxplot()+
  ggbeeswarm::geom_beeswarm(dodge.width = .75)
```



```
suppressWarnings(
  ggplot(rawdata,aes(x=as.numeric(Agegroup),y=Outcome,fill=Treatment))+
  ggbeeswarm::geom_beeswarm(aes(shape=Treatment),
                             alpha=.5, dodge.width = .15)+
```

```
geom_smooth()+
  scale_x_continuous("Agegroup", breaks=1:3,
    labels=c('young','middle','old'))
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: pseudoinverse used at 0.99
```

```
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: neighborhood radius 2.01
```

```
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: reciprocal condition number 2.0996e-16
```

```
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: There are other near singularities as well. 4.0401
```

```
Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object))), : pseudoinverse used at
0.99
```

```
Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object))), : neighborhood radius
2.01
```

```
Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object))), : reciprocal condition
number 2.0996e-16
```

```
Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object))), : There are other near
singularities as well. 4.0401
```

```
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: pseudoinverse used at 0.99
```

```
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: neighborhood radius 2.01
```

```
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: reciprocal condition number 2.0996e-16
```

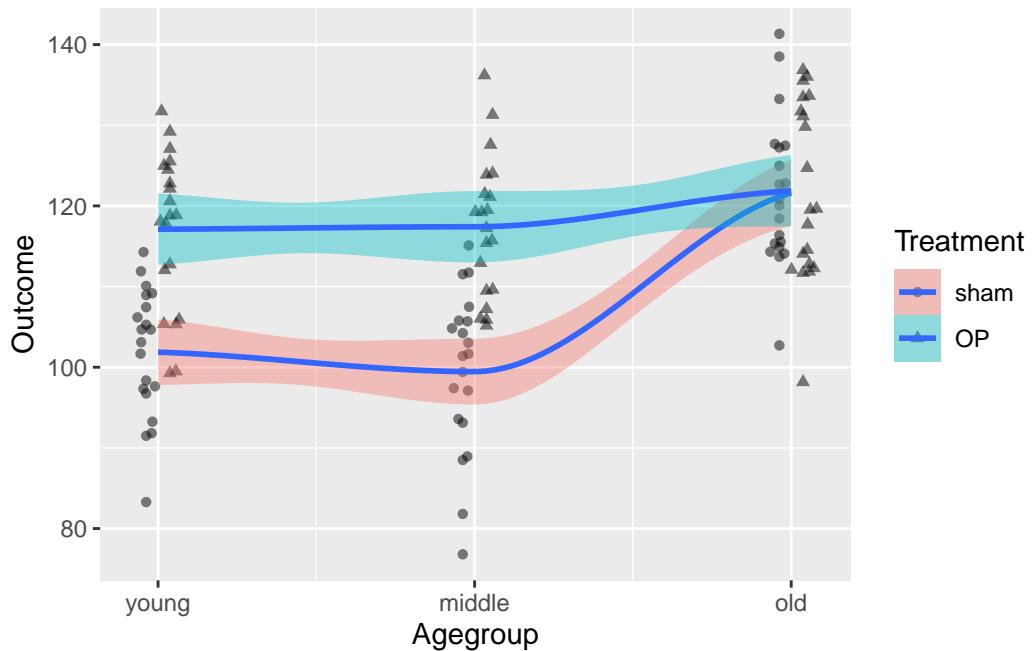
```
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: There are other near singularities as well. 4.0401
```

```
Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object))), : pseudoinverse used at
0.99
```

```
Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object))), : neighborhood radius
2.01
```

```
Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object))), : reciprocal condition
number 2.0996e-16
```

```
Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object))), : There are other near
singularities as well. 4.0401
```



```
lmout <- lm(Outcome ~ Agegroup * Treatment,
             data = rawdata)
tidy(lmout) |>
  select(1:2)
```

```
# A tibble: 6 x 2
  term                  estimate
  <chr>                 <dbl>
1 (Intercept)            102.
2 Agegroupmiddle        -2.41
3 Agegroupold           19.8 
4 TreatmentOP           15.2 
5 Agegroupmiddle:TreatmentOP 2.71
6 Agegroupold:TreatmentOP -15.0
```

```
anova(lmout) |> # this is WRONG!!!
tidy() |> slice(1:3) |>
  mutate(p.value=formatP(p.value, ndigits=3, mark=TRUE))
```

```
# A tibble: 3 x 6
  term                  df  sumsq meansq statistic p.value
  <chr>                 <int> <dbl>  <dbl>     <dbl> <chr>
1 Agegroup                2  4377.   2188.      24.3 0.001 ***
2 Treatment               1  3730.   3730.      41.4 0.001 ***
```

```
3 Agegroup:Treatment      2 1819.    910.      10.1 0.001 ***
```

```
Anova(lmout, type = 3) |>
  tidy() |> slice(1:4) |>
  mutate(p.value=formatP(p.value,ndigits=3, mark=TRUE))
```

```
# A tibble: 4 x 5
  term            sumsq   df statistic p.value
  <chr>          <dbl> <dbl>     <dbl> <chr>
1 (Intercept)    207533.    1     2301.  0.001 ***
2 Agegroup       5913.     2      32.8  0.001 ***
3 Treatment      2324.     1      25.8  0.001 ***
4 Agegroup:Treatment 1819.    2      10.1  0.001 ***
```

```
summary(glht(model=lmout,
  linfct=mcp(Treatment='Tukey')))
```

```
Warning in mcp2matrix(model, linfct = linfct): covariate interactions found --
default contrast might be inappropriate
```

#### Simultaneous Tests for General Linear Hypotheses

##### Multiple Comparisons of Means: Tukey Contrasts

```
Fit: lm(formula = Outcome ~ Agegroup * Treatment, data = rawdata)
```

##### Linear Hypotheses:

```
Estimate Std. Error t value Pr(>|t|)  
OP - sham == 0    15.245     3.003   5.076 1.52e-06 ***  
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
(Adjusted p values reported -- single-step method)
```

```
summary(glht(model=lmout,
  linfct=mcp(Agegroup='Tukey')))
```

```
Warning in mcp2matrix(model, linfct = linfct): covariate interactions found --
default contrast might be inappropriate
```

## Simultaneous Tests for General Linear Hypotheses

### Multiple Comparisons of Means: Tukey Contrasts

```
Fit: lm(formula = Outcome ~ Agegroup * Treatment, data = rawdata)

Linear Hypotheses:
Estimate Std. Error t value Pr(>|t|)
middle - young == 0   -2.409     3.003  -0.802   0.702
old - young == 0      19.750     3.003   6.576 <1e-05 ***
old - middle == 0    22.159     3.003   7.378 <1e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)
```

Ignoring the interaction in the model is no solution. Effect sizes will be wrongly estimated:

```
# falsch!!!
lmout_add <- lm(Outcome~Agegroup+Treatment,
                  data = rawdata)
tidy(lmout_add) |>
  select(1:2)

# A tibble: 4 x 2
  term       estimate
  <chr>     <dbl>
1 (Intercept) 104.
2 Agegroupmiddle -1.05
3 Agegroupold  12.3 
4 TreatmentOP  11.2 

anova(lmout) |> # this is WRONG!!!
  tidy() |> slice(1:2) |>
  mutate(p.value=formatP(p.value,ndigits=3, mark=TRUE))

# A tibble: 2 x 6
  term       df sumsq meansq statistic p.value
  <chr>     <int> <dbl>  <dbl>    <dbl> <chr>
1 Agegroup      2 4377.  2188.    24.3 0.001 ***
2 Treatment     1 3730.  3730.    41.4 0.001 ***
```

```
Anova(lmout,type = 2) |>
  tidy() |> slice(1:2) |>
  mutate(p.value=formatP(p.value,ndigits=3, mark=TRUE))
```

```
# A tibble: 2 x 5
  term      sumsq    df statistic p.value
  <chr>     <dbl> <dbl>     <dbl> <chr>
1 Agegroup  4377.     2      24.3 0.001 ***
2 Treatment 3730.     1      41.4 0.001 ***
```

## 17.4 How to specify interaction in multivariable models

```
# all possible interactions  
(lm_out <- lm(mpg~(wt*gear*factor(am)*cyl),  
              data=mtcars))
```

Call:

```
lm(formula = mpg ~ (wt * gear * factor(am) * cyl), data = mtcars)
```

Coefficients:

	wt	gear
(Intercept)	456.687	-158.453
factor(am)1		cyl
	-180.550	-16.129
wt:factor(am)1		gear:factor(am)1
	63.239	102.915
gear:cyl		factor(am)1:cyl
	5.764	-33.954
wt:gear:cyl		wt:factor(am)1:cyl
	-3.593	8.905
wt:gear:factor(am)1:cyl		gear:factor(am)1:cyl
	NA	3.456

```
(lm_out <- lm(mpg~(wt*factor(gear)*factor(am)*factor(cyl)),  
              data=mtcars))
```

Call:

```
lm(formula = mpg ~ (wt * factor(gear) * factor(am) * factor(cyl)),  
    data = mtcars)
```

Coefficients:

	(Intercept)
	39.880
wt	-7.456
factor(gear)4	-143.080
factor(gear)5	-149.462
factor(am)1	150.599

```

        factor(cyl)6
                      24.824
        factor(cyl)8
                      -14.821
        wt:factor(gear)4
                      47.456
        wt:factor(gear)5
                      49.599
        wt:factor(am)1
                      -49.160
        factor(gear)4:factor(am)1
                      NA
        factor(gear)5:factor(am)1
                      NA
        wt:factor(cyl)6
                      -6.013
        wt:factor(cyl)8
                      5.018
        factor(gear)4:factor(cyl)6
                      -72.234
        factor(gear)5:factor(cyl)6
                      -31.058
        factor(gear)4:factor(cyl)8
                      NA
        factor(gear)5:factor(cyl)8
                      -4.057
        factor(am)1:factor(cyl)6
                      21.011
        factor(am)1:factor(cyl)8
                      NA
        wt:factor(gear)4:factor(am)1
                      NA
        wt:factor(gear)5:factor(am)1
                      NA
        wt:factor(gear)4:factor(cyl)6
                      15.173
        wt:factor(gear)5:factor(cyl)6
                      NA
        wt:factor(gear)4:factor(cyl)8
                      NA
        wt:factor(gear)5:factor(cyl)8
                      NA
        wt:factor(am)1:factor(cyl)6
                      NA
        wt:factor(am)1:factor(cyl)8
                      NA

```

```

factor(gear)4:factor(am)1:factor(cyl)6
                               NA
factor(gear)5:factor(am)1:factor(cyl)6
                               NA
factor(gear)4:factor(am)1:factor(cyl)8
                               NA
factor(gear)5:factor(am)1:factor(cyl)8
                               NA
wt:factor(gear)4:factor(am)1:factor(cyl)6
                               NA
wt:factor(gear)5:factor(am)1:factor(cyl)6
                               NA
wt:factor(gear)4:factor(am)1:factor(cyl)8
                               NA
wt:factor(gear)5:factor(am)1:factor(cyl)8
                               NA

```

```

# only two-way interactions
(lm_out <- lm(mpg~(wt+gear+factor(am)+cyl)^2,
              data=mtcars))

```

Call:

```
lm(formula = mpg ~ (wt + gear + factor(am) + cyl)^2, data = mtcars)
```

Coefficients:

	wt	gear	factor(am)1
(Intercept)	57.58968	-17.65458	-7.80045
cyl	3.13308	4.90387	-11.51010
gear:factor(am)1	2.23288	gear:cyl	factor(am)1:cyl
		-1.43230	2.15419

```

#some selected interactions
(lm_out <- lm(mpg~wt*(gear+factor(am)+cyl),
              data=mtcars))

```

Call:

```
lm(formula = mpg ~ wt * (gear + factor(am) + cyl), data = mtcars)
```

Coefficients:

(Intercept)	wt	gear	factor(am)1	cyl
-------------	----	------	-------------	-----

50.53760	-8.43482	-5.80119	21.62110	-
1.04500				
wt:gear	wt:factor(am)1	wt:cyl		
2.03447	-7.59198	-0.00499		

# 18 Logistic regression

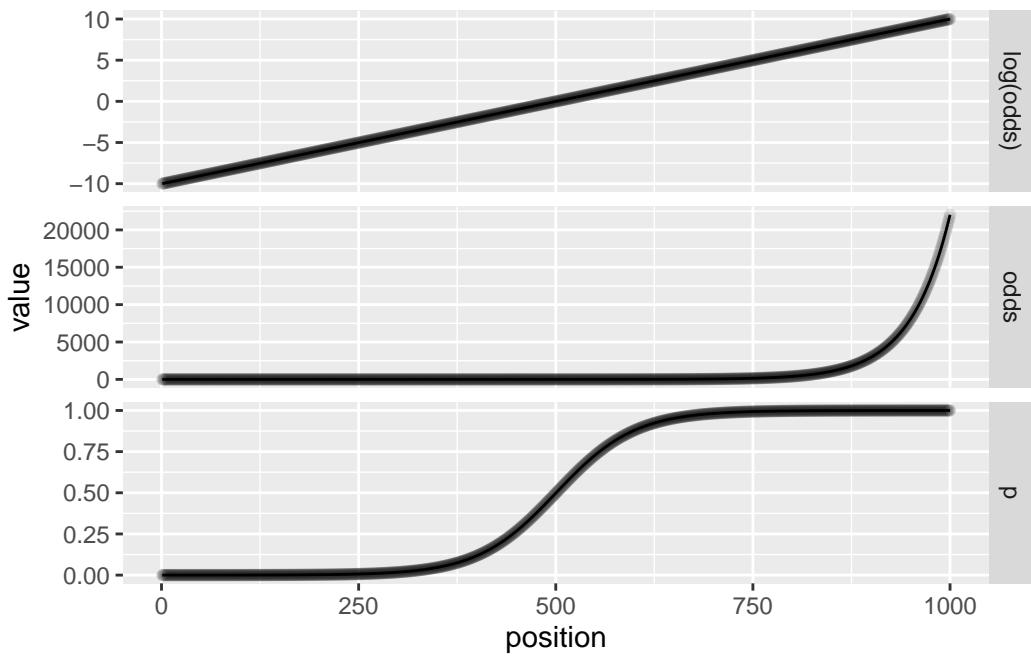
Generalized linear models will be introduced using logistic regression as an example. The data set `infert` contains information were the outcome follows a binomial distribution.

```
pacman::p_load(car, # Anova()
               wrappedtools, tidyverse, ggbeeswarm,
               rpart, rpart.plot,
               pROC # roc() ggroc()
               )
```

## 18.1 Odds vs. probability

Probabilities are usually expressed xx%, 20% risk for rain means that on 20 out of 100 days like this it rains. Odds are expressed as xx:yy, 20:80 means that on 20 out of 100 days like this it rains, while on 80 days it does not. The odds are calculated as  $p/(1-p)$ , where  $p$  is the probability. The log odds are calculated as  $\log(\text{odds})$ .

For many traits it is reasonable to assume a sigmoidal relationship between a risk (like LDL cholesterol) and the probability of an outcome like myocardial infarction. Those probabilities are restrained between 0 and 1. The odds are restrained at 0 but have not upper bound. The log odds are not restrained and are linearly related to the risk increase. While this may cause mental knots, it is useful mathematically and forms the basis for logistic regression, a generalized linear model for outcomes with binomial distribution.



## 18.2 Data preparation

Before the analysis, the data set is cleaned and prepared for the analysis. The age variable is transformed into pentayears, and the parity variable is lumped into two categories. Education is reversed and transformed into a factor.

```
rawdata <- infert |>
  as_tibble() |>
  select(-contains("stratum"))
head(rawdata)
```

```
# A tibble: 6 x 6
  education    age parity induced case spontaneous
  <fct>     <dbl>   <dbl>   <dbl> <dbl>       <dbl>
1 0-5yrs      26      6       1     1         2
2 0-5yrs      42      1       1     1         0
3 0-5yrs      39      6       2     1         0
4 0-5yrs      34      4       2     1         0
5 6-11yrs     35      3       1     1         1
6 6-11yrs     36      4       2     1         1
```

```
rawdata$age<-rawdata$age%/%5
```

```
[1] 25 40 35 30 35 35 20 30 20 25 25 35 30 25 30 25 30 25 25 40 40 35 25 35 25  
[26] 40 35 30 25 30 30 30 40 30 35 35 35 30 30 25 35 35 40 35 30 35 25 25 25 35  
[51] 20 35 25 25 25 35 25 25 25 35 25 30 30 25 30 20 25 35 25 30 25 30 35 25  
[76] 30 30 25 30 30 35 25 20 25 40 35 30 35 35 20 30 20 25 25 35 30 25 30 25 30  
[101] 25 25 40 40 35 25 35 25 40 35 30 25 30 30 40 30 35 35 30 30 25 35 35  
[126] 40 35 30 35 25 25 35 20 35 25 25 35 25 25 25 25 35 25 30 30 25 30 20  
[151] 25 35 25 30 25 30 30 25 30 30 35 25 25 20 25 40 35 30 35 35 20 30 20 25  
[176] 25 35 30 25 30 25 25 40 40 35 25 35 25 40 35 30 25 30 30 30 40 30 35  
[201] 35 35 30 30 25 35 35 40 35 30 35 25 25 25 35 20 35 25 25 25 35 25 25 25  
[226] 35 25 30 30 25 30 20 25 35 25 30 25 35 25 30 30 25 30 30 35 25 20
```

```
rawdata$age/5
```

```
[1] 5.2 8.4 7.8 6.8 7.0 7.2 4.6 6.4 4.2 5.6 5.8 7.4 6.2 5.8 6.2 5.4 6.0 5.2  
[19] 5.0 8.8 8.0 7.0 5.6 7.2 5.4 8.0 7.6 6.8 5.6 6.0 6.4 6.8 8.4 6.4 7.8 7.0  
[37] 7.2 6.8 6.0 5.6 7.8 7.0 8.2 7.4 6.0 7.4 5.6 5.4 5.2 7.6 4.8 7.2 5.4 5.6  
[55] 5.8 7.2 5.6 5.6 5.6 5.4 7.0 5.0 6.8 6.2 5.2 6.4 4.2 5.6 7.4 5.0 6.4 5.0  
[73] 6.2 7.6 5.2 6.2 6.2 5.0 6.2 6.8 7.0 5.8 4.6 5.2 8.4 7.8 6.8 7.0 7.2 4.6  
[91] 6.4 4.2 5.6 5.8 7.4 6.2 5.8 6.2 5.4 6.0 5.2 5.0 8.8 8.0 7.0 5.6 7.2 5.4  
[109] 8.0 7.6 6.8 5.6 6.0 6.4 6.8 8.4 6.4 7.8 7.0 7.2 6.8 6.0 5.6 7.8 7.0 8.2  
[127] 7.4 6.0 7.4 5.6 5.4 5.2 7.6 4.8 7.2 5.4 5.6 5.8 7.2 5.6 5.6 5.6 5.4 7.0  
[145] 5.0 6.8 6.2 5.2 6.4 4.2 5.6 7.4 5.0 6.4 5.0 6.2 5.2 6.2 6.2 5.0 6.2 6.8  
[163] 7.0 5.8 4.6 5.2 8.4 7.8 6.8 7.0 7.2 4.6 6.4 4.2 5.6 5.8 7.4 6.2 5.8 6.2  
[181] 5.4 6.0 5.2 5.0 8.8 8.0 7.0 5.6 7.2 5.4 8.0 7.6 6.8 5.6 6.0 6.4 6.8 8.4  
[199] 6.4 7.8 7.0 7.2 6.8 6.0 5.6 7.8 7.0 8.2 7.4 6.0 7.4 5.6 5.4 5.2 7.6 4.8  
[217] 7.2 5.4 5.6 5.8 7.2 5.6 5.6 5.4 7.0 5.0 6.8 6.2 5.2 6.4 4.2 5.6 7.4  
[235] 5.0 6.4 5.0 6.2 7.6 5.2 6.2 6.2 5.0 6.2 6.8 7.0 5.8 4.6
```

```
table(rawdata$parity)
```

```
1 2 3 4 5 6  
99 81 36 18 6 8
```

```
rawdata <- rawdata |>  
  mutate(  
    case=factor(case),  
    induced_f=factor(induced,  
      levels = c('0','1','2'),  
      labels = (c('none','one','two or more'))),  
    spontaneous_f=factor(spontaneous),  
    `age [pentayears]`=age/5,
```

```
education=forcats::fct_rev(education),  
parity_grp=forcats::fct_lump_n(as.character(parity),  
                                n = 2, other_level = '>2') |>  
fct_rev())
```

### 18.3 Build model

The model is built using `glm()` and the output is extracted and transformed. The model is tested using `Anova()` and `summary()`. The results are then prepared for plotting.

```
logreg_out <-
  glm(case~`age [pentayears]`+education+parity_grp+
    induced_f+spontaneous_f,
    family=binomial(), data=rawdata)
logreg_out
```

```
Call: glm(formula = case ~ `age [pentayears]` + education + parity_grp +
  induced_f + spontaneous_f, family = binomial(), data = rawdata)
```

Coefficients:

(Intercept)	`age [pentayears]`	education6-11yrs
	-5.8727	0.4394
education0-5yrs	parity_grp2	parity_grp1
	0.6082	2.7048
induced_fone	induced_ftwo or more	spontaneous_f1
	1.3591	2.0599
spontaneous_f2		
	4.3296	

Degrees of Freedom: 247 Total (i.e. Null); 238 Residual

Null Deviance: 316.2

Residual Deviance: 255.9 AIC: 275.9

```
#extract/transform model parameters
(ORs <- exp(logreg_out$coefficients))
```

(Intercept)	`age [pentayears]`	education6-11yrs
0.002815305	1.199607197	1.551793487
education0-5yrs	parity_grp2	parity_grp1
1.837105706	4.289689825	14.951228070
induced_fone	induced_ftwo or more	spontaneous_f1
3.892652439	16.931760140	7.844857609
spontaneous_f2		
75.916231594		

```
(CIs <- exp(confint(logreg_out)))
```

Waiting for profiling to be done...

	2.5 %	97.5 %
(Intercept)	1.860835e-04	0.03486434
`age [pentayears]`	8.849570e-01	1.63799477
education6-11yrs	8.051279e-01	3.03038853
education0-5yrs	3.735091e-01	8.28017336
parity_grp2	1.689458e+00	11.53626479
parity_grp1	4.663096e+00	53.09682931
induced_fone	1.726419e+00	9.15964545
induced_ftwo or more	4.809607e+00	65.00363723
spontaneous_f1	3.580230e+00	18.15518387
spontaneous_f2	2.133238e+01	311.72957158

```
#test model
>Anova_out <- Anova(logreg_out,type = 2) |>
  broom::tidy() |>
  mutate(p.value=formatP(p.value,ndigits = 5))
```

```
# A tibble: 5 x 4
  term          statistic    df p.value
  <chr>        <dbl> <dbl> <chr>
1 `age [pentayears]` 1.37     1 0.24200
2 education       1.89     2 0.38838
3 parity_grp      22.8    2 0.00001
4 induced_f       22.1    2 0.00002
5 spontaneous_f   60.2    2 0.00001
```

```
## test each OR
(sum_out <- summary(logreg_out))
```

Call:

```
glm(formula = case ~ `age [pentayears]` + education + parity_grp +
  induced_f + spontaneous_f, family = binomial(), data = rawdata)
```

Coefficients:

	Estimate	Std. Error	z value	Pr(> z )
(Intercept)	-5.8727	1.3291	-4.419	9.93e-06 ***
`age [pentayears]`	0.1820	0.1564	1.163	0.24467
education6-11yrs	0.4394	0.3370	1.304	0.19223
education0-5yrs	0.6082	0.7781	0.782	0.43442

```

parity_grp2          1.4562    0.4880   2.984  0.00284 ***
parity_grp1          2.7048    0.6186   4.373  1.23e-05 ***
induced_fone         1.3591    0.4236   3.208  0.00134 **
induced_ftwo or more 2.8292    0.6610   4.280  1.87e-05 ***
spontaneous_f1       2.0599    0.4124   4.994  5.90e-07 ***
spontaneous_f2       4.3296    0.6814   6.354  2.10e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

(Dispersion parameter for binomial family taken to be 1)

```

Null deviance: 316.17  on 247  degrees of freedom
Residual deviance: 255.91  on 238  degrees of freedom
AIC: 275.91

```

Number of Fisher Scoring iterations: 5

```
# broom::tidy(logreg_out)
```

## 18.4 Create structure for ggplot

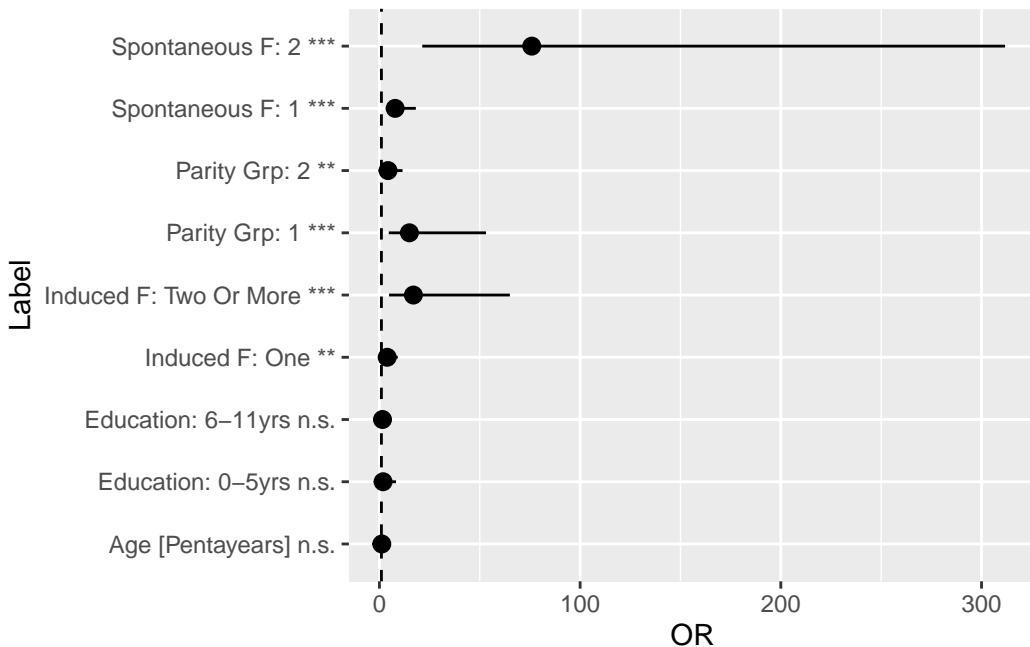
```

OR_plotdata <- tibble(
  Predictor=names(ORs)[-1] |>
    # make names nicer
    str_replace('_', ' ') |>
    str_replace_all(c(
      '(grp)(.)'='\\1: \\2',
      '(f)(.)'='\\1: \\2',
      '(n)(\\d)'='\\1: \\2',
      "``" = ""))
    str_to_title(),
  OR=ORs[-1],
  CI_low=CI[,1],
  CI_high=CI[,2],
  p=sum_out$coefficients[-1,4],
  Significance=markSign(p),
  Label=paste(Predictor,Significance))

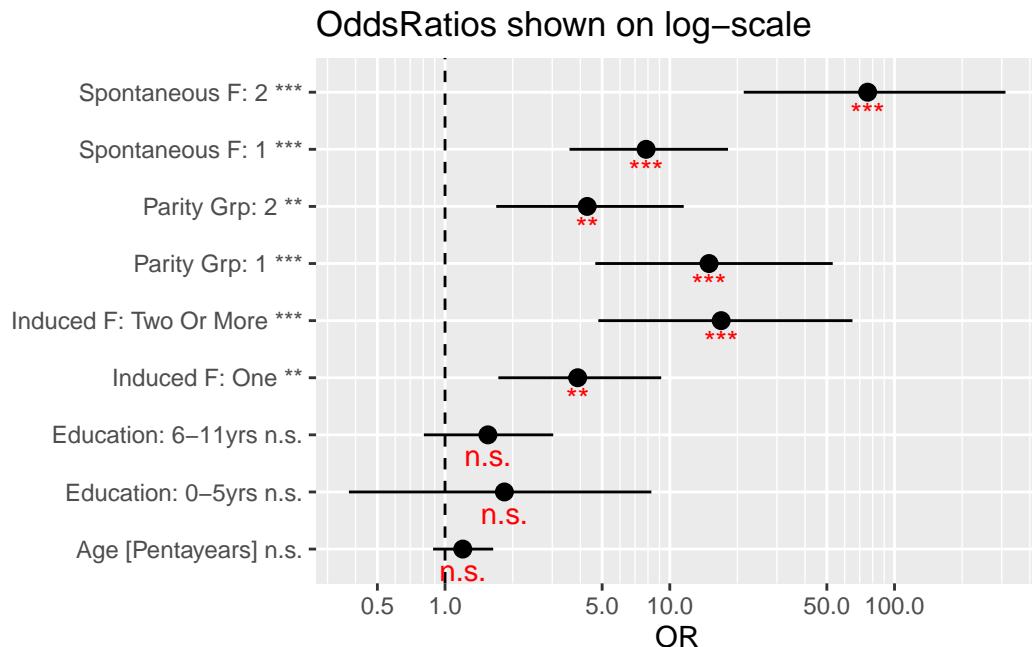
```

## 18.5 create forest plot

```
baseplot <-
  ggplot(OR_plotdata, aes(x = Label,y=OR))+ 
  geom_pointrange(aes(ymin=CI_low, ymax=CI_high))+ 
  geom_hline(yintercept = 1,linewidth=.5,linetype=2)+ 
  coord_flip()
baseplot
```



```
baseplot+
  scale_y_log10(breaks=logrange_15,
                 minor_breaks=logrange_123456789 )+
  geom_text(aes(label=Significance), vjust=1.5,color='red')+
  ggtitle('OddsRatios shown on log-scale')+
  xlab(NULL)
```



## 18.6 Create predictions

```
rawdata$pGLM <-
  predict(logreg_out, type = 'response') #predict probability 0-1
# run ROC for cutoff
roc_out <- roc(response=rawdata$case,
                 predictor=rawdata$pGLM)
```

Setting levels: control = 0, case = 1

Setting direction: controls < cases

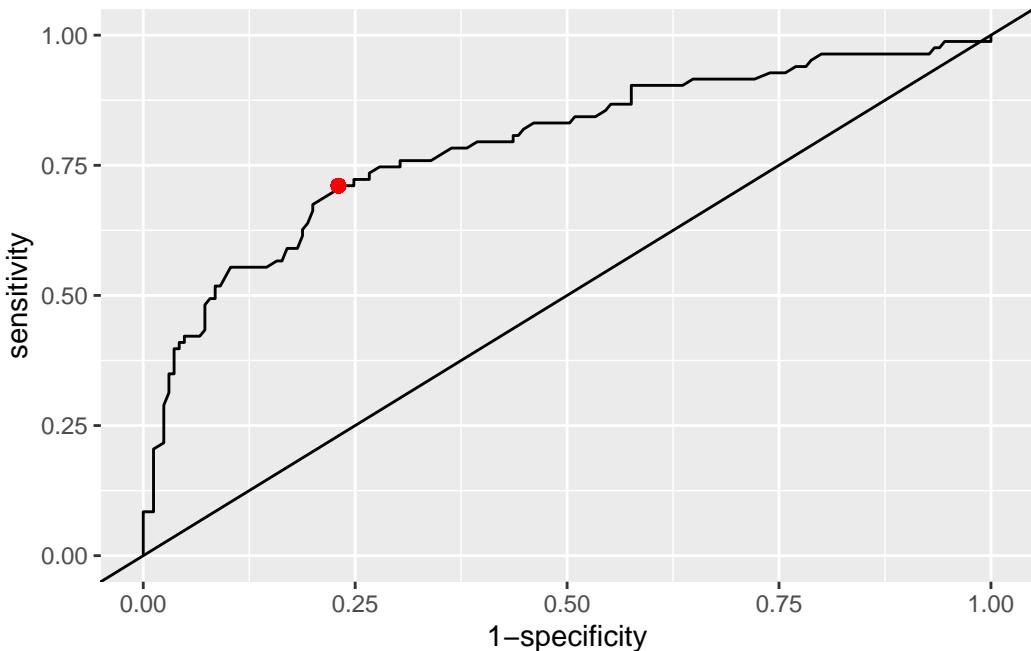
```
youden <- pROC::coords(roc_out,x='best',
                        best.method='youden')
youden
```

```
threshold specificity sensitivity
1 0.4141249    0.769697   0.7108434
```

```

ggroc(roc_out, legacy.axes = T) +
  geom_abline(slope = 1, intercept = 0) +
  geom_point(x=1-youden$specificity,
             y=youden$sensitivity, color='red', size=2 )

```



```

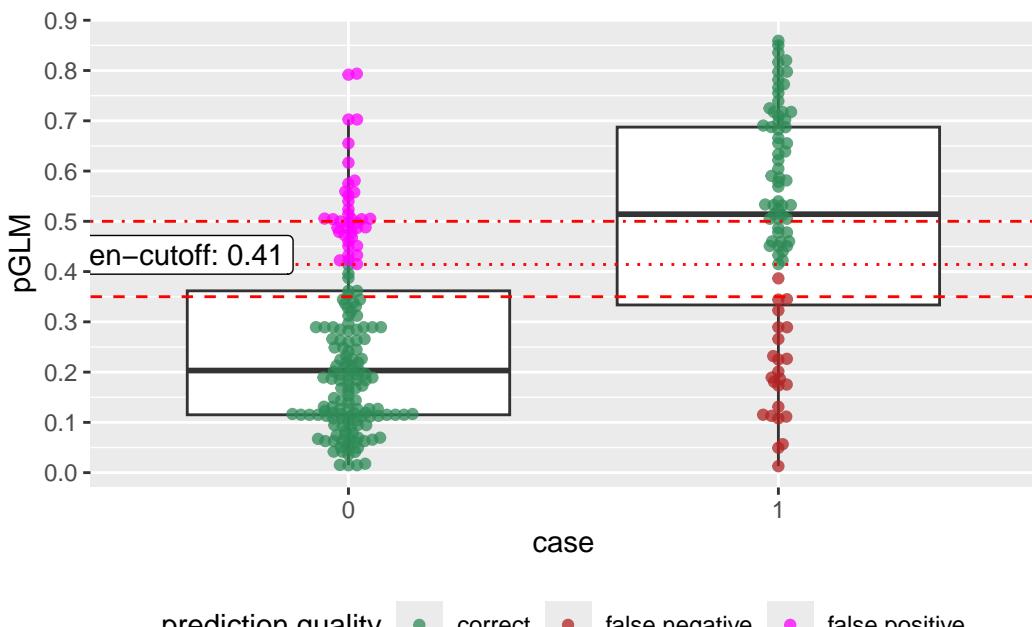
# plot predictions
rawdata |>
  mutate(`prediction quality` =
    case_when(case=="1" &
      pGLM<youden$threshold ~
        "false negative",
      case=="0" &
        pGLM>=youden$threshold ~
        "false positive",
      .default = 'correct' )) |>
  ggplot(aes(case,pGLM))+
  geom_boxplot(outlier.alpha = 0)+
  scale_y_continuous(breaks=seq(0,1,.1))+ 
  geom_beeswarm(alpha=.75,
                aes(color=`prediction quality`))+ 
  scale_color_manual(values=c("seagreen","firebrick","magenta"))+
  geom_hline(yintercept = c(.35, youden$threshold,.5),
             color='red',
             linetype=2:4)+
```

```

annotate(geom = "label",
         x = 1,y=youden$threshold,
         label=paste("Youden-cutoff:",
                     roundR(youden$threshold)),
         hjust=1.2,vjust=0.25)+  

theme(legend.position="bottom")

```



```

# ORhuman <-
# tibble(
#   Predictor=names(ORs),
#   OR=ORs,
#   OR_low=CI[,1],
#   OR_high=CI[,2]) |>
#   rowwise() |>
#   mutate(across(-Predictor,
#                 ~roundR(.x,level = 3)),
#         `OR(CI)` = paste0(OR," (",OR_low," / ",OR_high,")")) |>
#   ungroup() |>
#   pull(`OR(CI)`)

# ORhuman <-
#   paste0(map_chr(ORs,roundR),' (',
#         apply(CI,MARGIN = 1,
#               FUN=function(x){

```

```

#                               paste(roundR(x),collapse=' / ')}), ''))
ORreport <-
  tibble(
  Predictor=names(ORs),
  OR=ORs,
  OR_low=CI.s[,1],
  OR_high=CI.s[,2]) |>
  rowwise() |>
  mutate(across(-Predictor,
                ~roundR(.x,level = 3)),
         `OR(CI)` = paste0(OR," (",OR_low," / ",OR_high,")")) |>
  ungroup()

#
#  tibble(Predictor=rownames(CIs)[-1],
#         OR=ORs[-1],
#         low=CIs[-1,1],
#         high=CIs[-1,2],
#         `OR (CI95)`=NA)
# ORrounded <- apply(ORreport[,2:4],MARGIN = 1,roundR)
# ORreport$`OR (CI95)` <-
#   paste0(ORrounded[1,],' (',ORrounded[2,],'/',
#          ORrounded[3,],')')
#
ORreport |>
  flextable::flextable() |>
  flextable::set_table_properties(width = 1,layout = 'autofit')

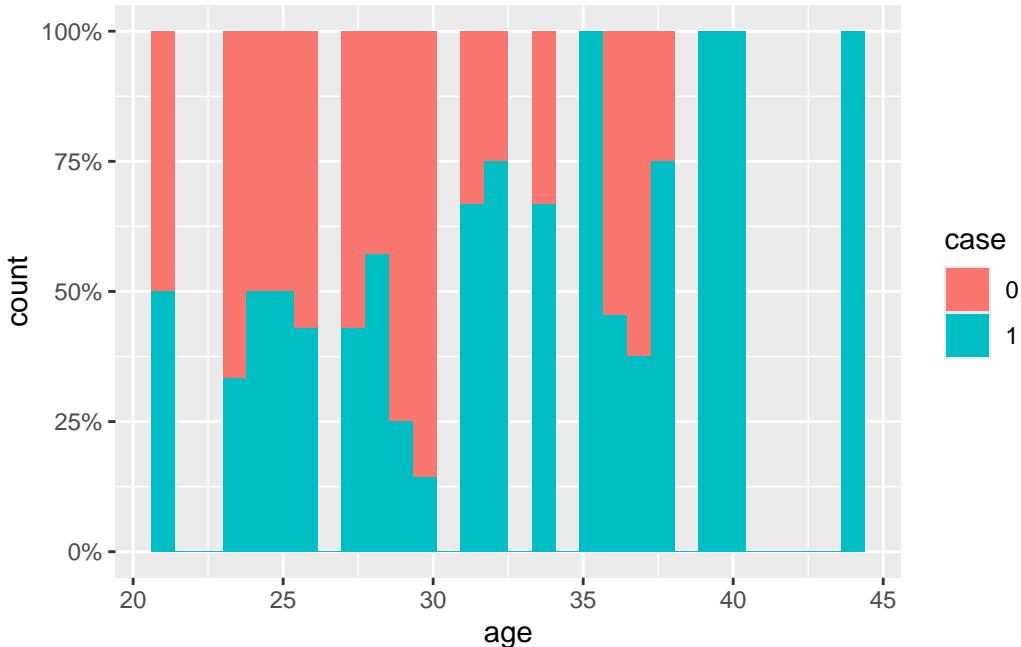
```

Predictor	OR	OR_low	OR_high	OR(CI)
(Intercept)	0.00282	0.000186	0.0349	0.00282 (0.000186 / 0.0349)
'age [pentayears]'	1.20	0.885	1.64	1.20 (0.885 / 1.64)
education6-11yrs	1.55	0.805	3.03	1.55 (0.805 / 3.03)
education0-5yrs	1.84	0.374	8.28	1.84 (0.374 / 8.28)
parity_grp2	4.29	1.69	11.5	4.29 (1.69 / 11.5)
parity_grp1	15.0	4.66	53.1	15.0 (4.66 / 53.1)
induced_fone	3.89	1.73	9.16	3.89 (1.73 / 9.16)
induced_ftwo or more	16.9	4.81	65.0	16.9 (4.81 / 65.0)
spontaneous_f1	7.84	3.58	18.2	7.84 (3.58 / 18.2)
spontaneous_f2	75.9	21.3	312	75.9 (21.3 / 312)

```
cat(' \\n\\n')
```

```
rawdata |>
  filter(spontaneous_f != "0") |>
  ggplot(aes(x = age, fill = case)) +
  geom_histogram(position = "fill") +
  scale_y_continuous(labels = scales::percent) #+
```

`stat\_bin()` using `bins = 30`. Pick better value `binwidth`.



```
# facet_grid(rows = vars(spontaneous_f))
```

## 18.7 Regression tree as alternative to `glm`

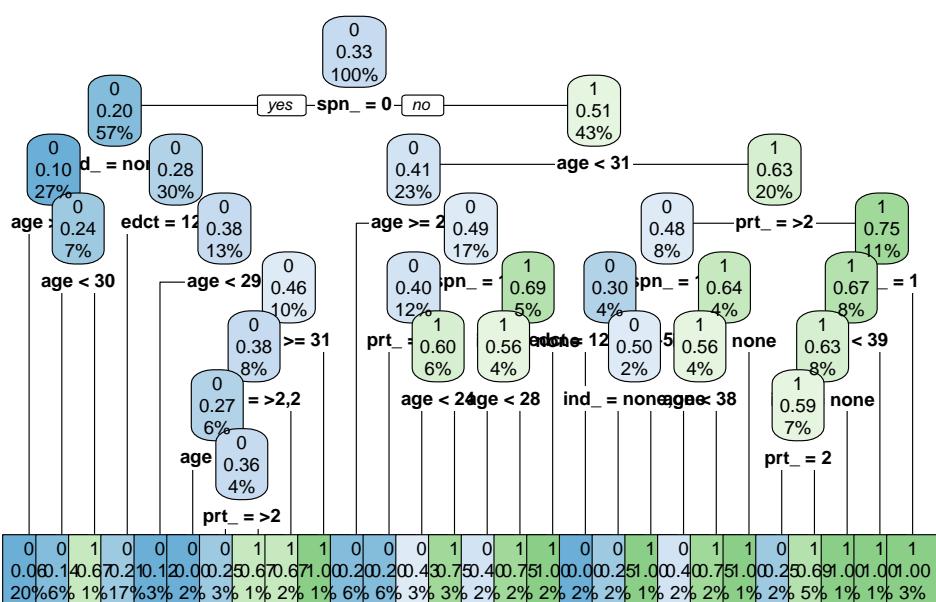
```
cn()
```

```
[1] "education"          "age"           "parity"         "induced"
[5] "case"              "spontaneous"    "induced_f"      "spontaneous_f"
[9] "age [pentayears]"  "parity_grp"     "pGLM"
```

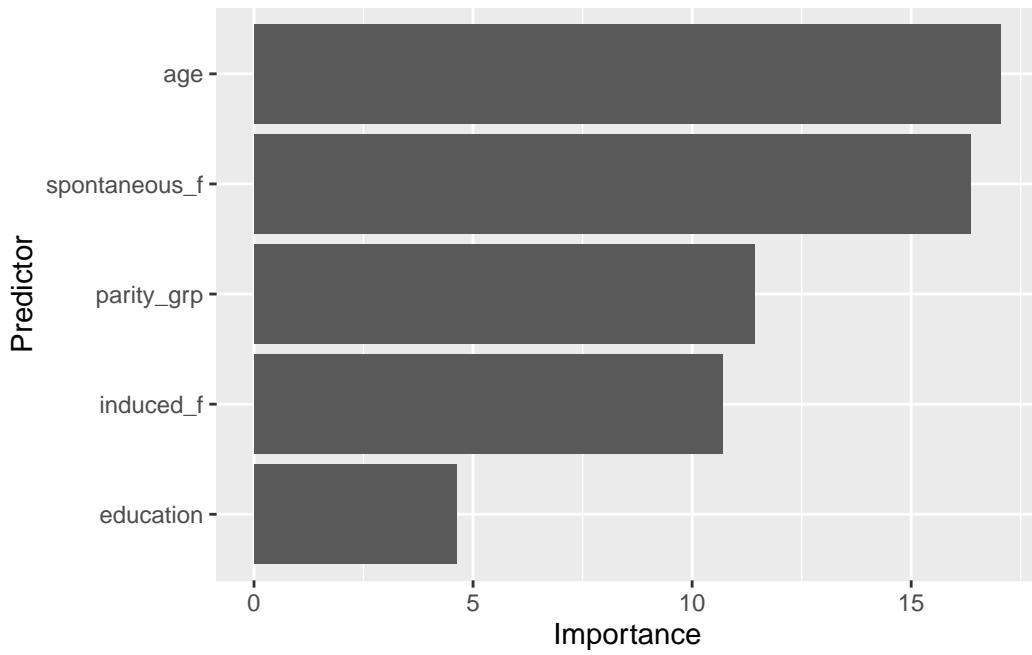
```

predvars <- ColSeeker(namepattern =
                        c("age$","edu","_grp","_f"))
rtformula <- paste("case~",
                     paste(predvars$btricked,collapse = "+"))
regtree_out<-rpart(rtformula,
                     minsplit=5,cp=.001,
                     data=rawdata)
rpart.plot(regtree_out,type = 2,tweak=2.0, varlen=4,facelen=5,leaf.round=0)

```



```
importance <-  
  as_tibble(regtree_out$variable.importance,  
             rownames='Predictor') |>  
  dplyr::rename('Importance'=2) |>  
  mutate(Predictor=fct_reorder(.f = Predictor,  
                               .x = Importance,  
                               .fun = min)) |>  
  arrange(desc(Importance))  
importance |>  
  ggplot(aes(Predictor,Importance))+  
  geom_col()  
  coord_flip()
```



```
rawdata$pRT <- predict(regtree_out)[,2]

#pROC
roc_out_rt <- roc(response=rawdata$case,
predictor=rawdata$pRT )
```

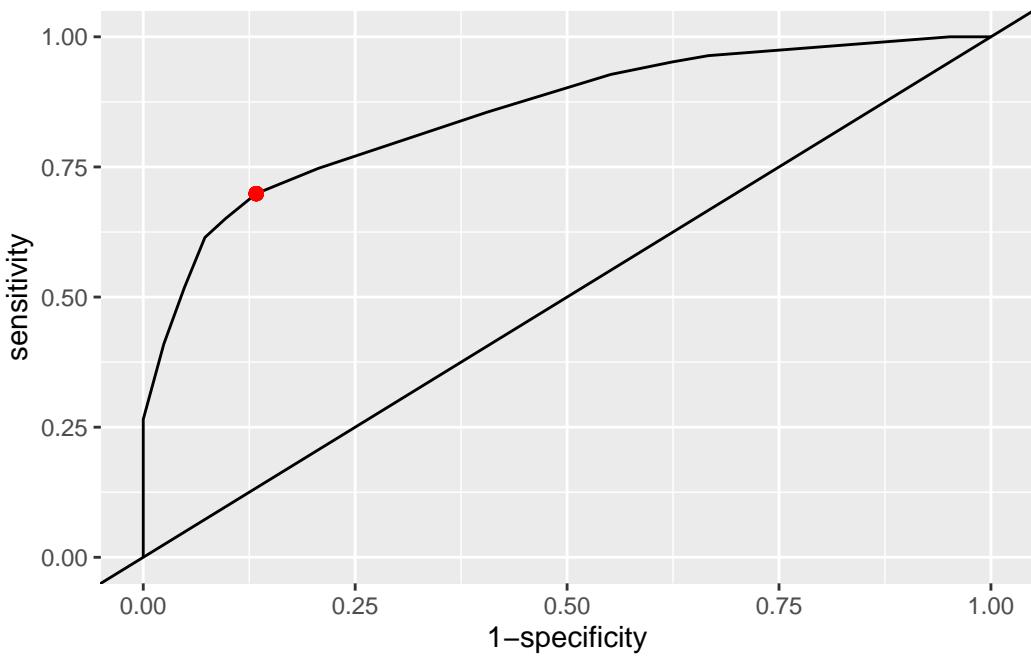
Setting levels: control = 0, case = 1

Setting direction: controls < cases

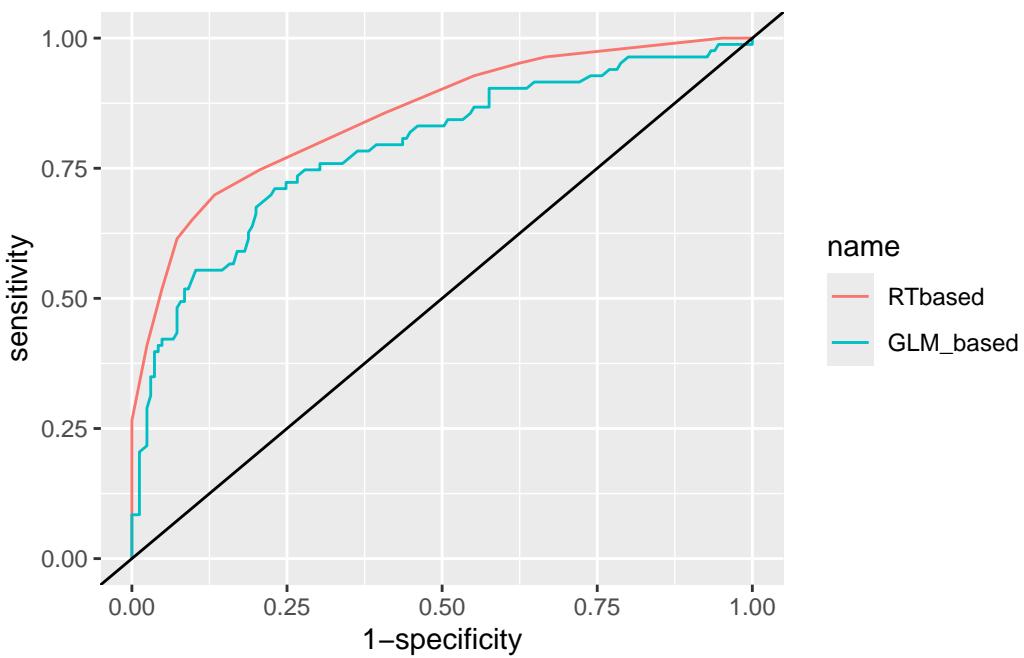
```
youden <- pROC::coords(roc_out_rt,x='best',
best.method='youden')
youden
```

	threshold	specificity	sensitivity
1	0.325	0.8666667	0.6987952

```
ggroc(roc_out_rt,legacy.axes = T) +
geom_abline(slope = 1,intercept = 0) +
geom_point(x=1-youden$specificity,
y=youden$sensitivity, color='red', size=2 )
```



```
ggroc(list(RTbased=roc_out_rt,GLM_based=roc_out),legacy.axes = T)+  
  geom_abline(slope = 1,intercept = 0)
```



```
ggplot(rawdata,aes(x=case,y=pRT))+  
  geom_boxplot(coef=3)+
```

```

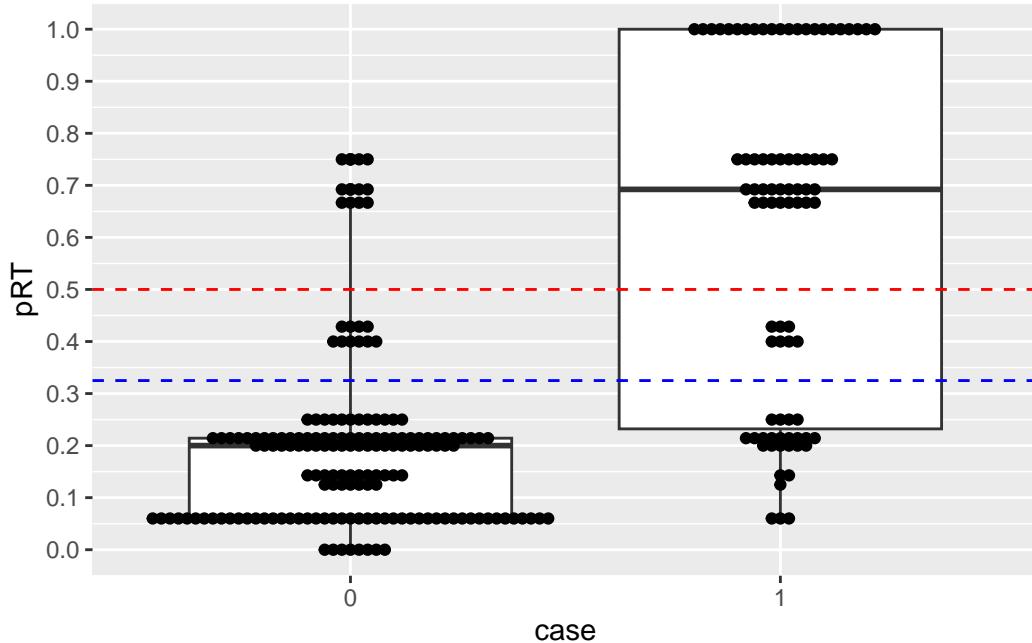
scale_y_continuous(breaks = seq(from = 0,to = 1,by = .1))+  

geom_hline(yintercept = c(.5,youden$threshold),  

color=c('red','blue'), linetype=2)+  

ggbeeswarm::geom_beeswarm()

```



```

ggplot(rawdata,aes(pGLM,pRT, color=case,shape=case))+  

geom_point(size=2)+  

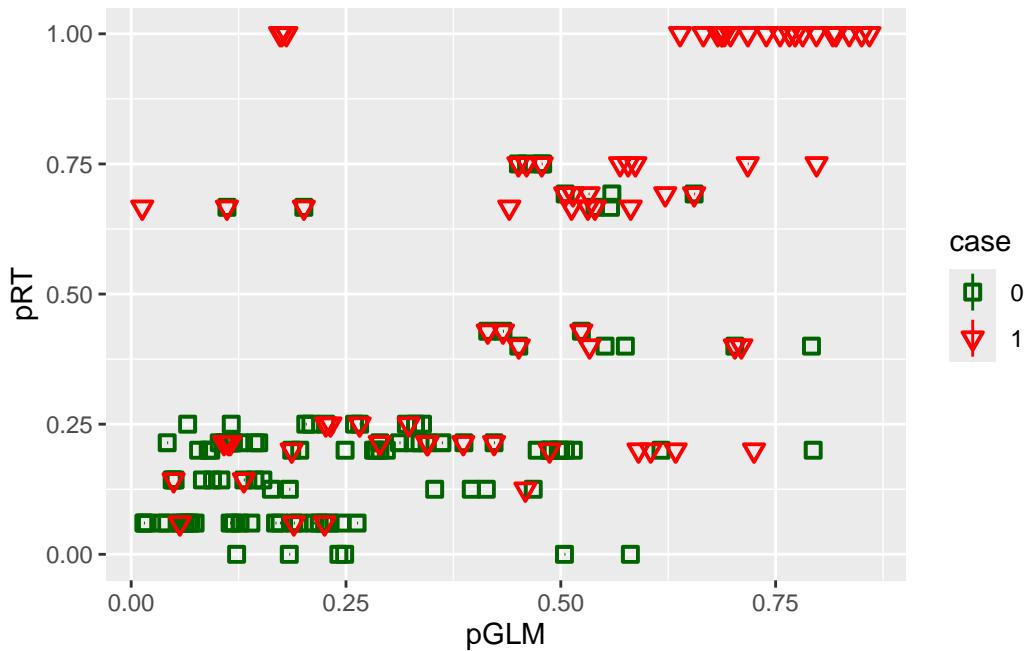
scale_color_manual(values = c('darkgreen','red'))+  

scale_shape_manual(values = c(0,6))+  

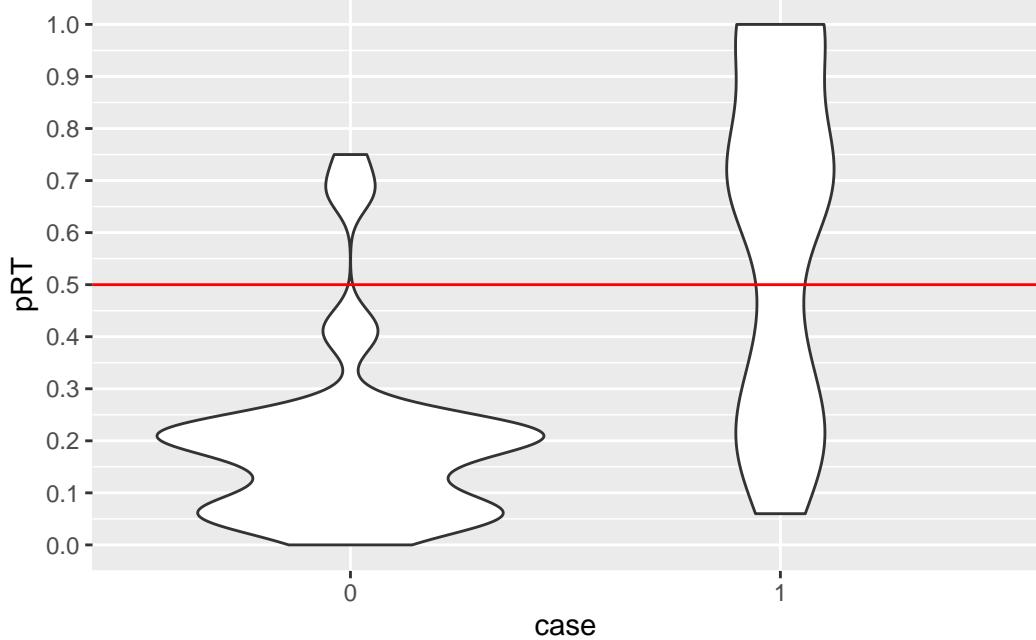
stat_summary(fun.data=mean_cl_boot)

```

Warning: Removed 139 rows containing missing values or values outside the scale range (`geom\_segment()`).



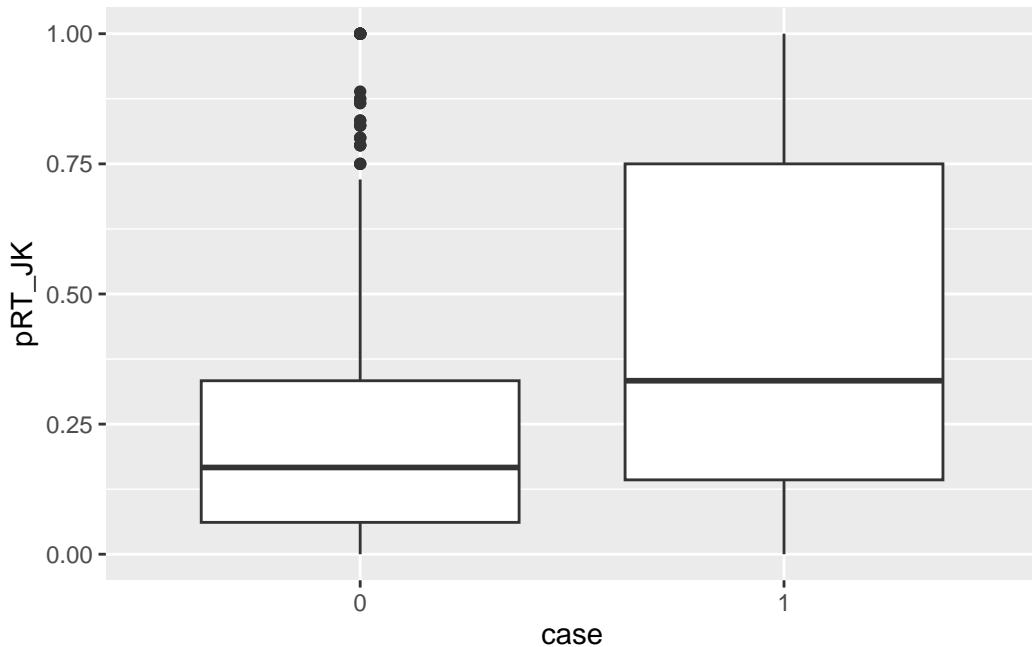
```
ggplot(rawdata,aes(x=case,y=pRT))+
  geom_violin()+
  scale_y_continuous(breaks = seq(0,to = 1,by = .1))+
```



## 18.8 Jackknife

```
rawdata$pRT_JK <- NA_real_
rawdata$pGLM_JK <- NA_real_
for(pat_i in 1: nrow(rawdata)){
  tempdata <- rawdata[-pat_i,]
  regtree_out_tmp<-rpart(rtformula,
                           minsplit=5,
                           cp=.001, data=tempdata)
  rawdata$pRT_JK[pat_i] <-
    predict(regtree_out_tmp,
           newdata = rawdata[pat_i,])[,2]

  glm_out_tmp<-glm(rtformula,
                     family = binomial(), data=tempdata)
  rawdata$pGLM_JK[pat_i] <-
    predict(glm_out_tmp,newdata = rawdata[pat_i,],
           type="response")
}
ggplot(rawdata,aes(case,pRT_JK))+
  geom_boxplot()
```



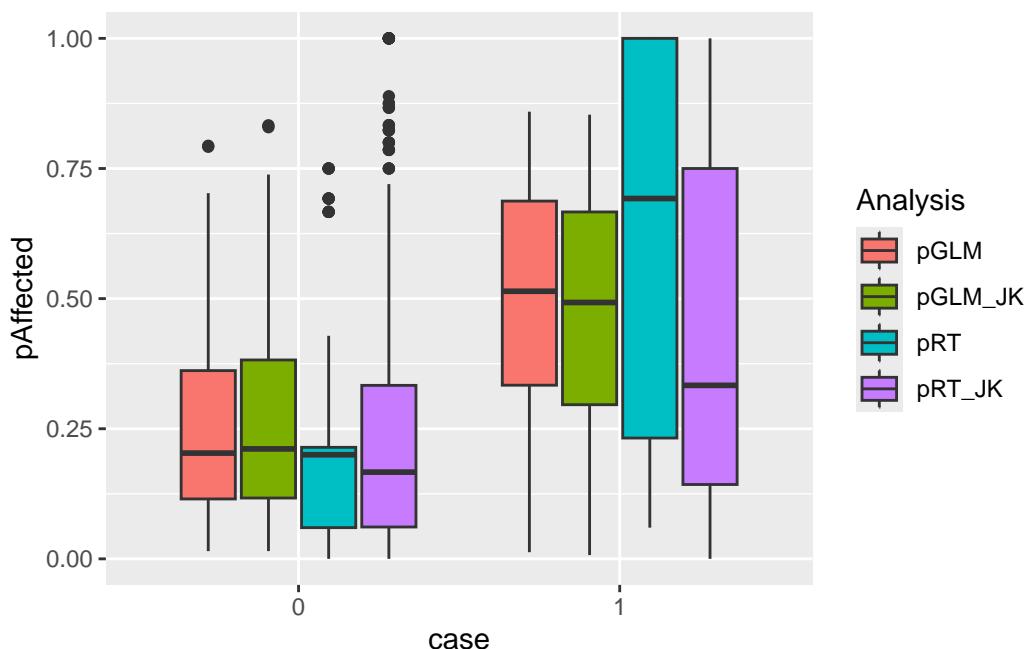
```
rawdata |>
  dplyr::select(case,pGLM,pGLM_JK, pRT_JK, pRT) |>
```

```

pivot_longer(cols = c(pGLM,pGLM_JK, pRT_JK,pRT),
             names_to = 'Analysis',
             values_to = 'pAffected') |>
ggplot(aes(case,pAffected,fill=Analysis))+  

geom_boxplot()

```



# 19 Linear Mixed Models

Linear Mixed-Effects Models (LMMs) are statistical tools used to analyze data where observations are grouped or repeated, allowing researchers to model both **fixed effects** (effects for which an estimate/prediction and a p-value are of interest, which represent population-level averages, like treatment type) and **random effects** (effects for which usually no prediction or p-value is required, which account for variation and non-independence among the groups, like individual subjects, families, or experimental batches). In biology, LMMs are essential for analyzing longitudinal studies (e.g., tracking an animal's growth over time), repeated measure experiments (e.g., measuring gene expression in the same tissue across different conditions, or exploring experimental variation between technical replicates), and ecological studies (e.g., accounting for variation between different field sites or populations).

```
pacman::p_load(conflicted, wrappedtools, car, broom,
                 multcomp, tidyverse, DescTools, haven,
                 ggbeeswarm,
                 lme4, nlme, merTools,
                 easystats, patchwork, here)

conflicts_prefer(modelbased::standardize,
                  dplyr::filter,
                  dplyr::select)
```

```
[conflicted] Will prefer modelbased::standardize over any other package.
[conflicted] Will prefer dplyr::filter over any other package.
[conflicted] Will prefer dplyr::select over any other package.
```

## 19.1 Import / Preparation

The data are the same as for the lm analyses. This time, passage effect is modeled as a random effect, as we are not interested in the effect of each passage, but want to account for the variation between passages. Future experiments are not expected to have the exact same passage differences. Passage numbers are arbitrary anyway. The numeric column Passage is mutated into a factor as Passage\_F, this is necessary for group comparisons in ANOVA.

```
rawdata<-read_sav(file=here('Data/Zellbeads.sav')) |>
  as_factor() |>
```

```

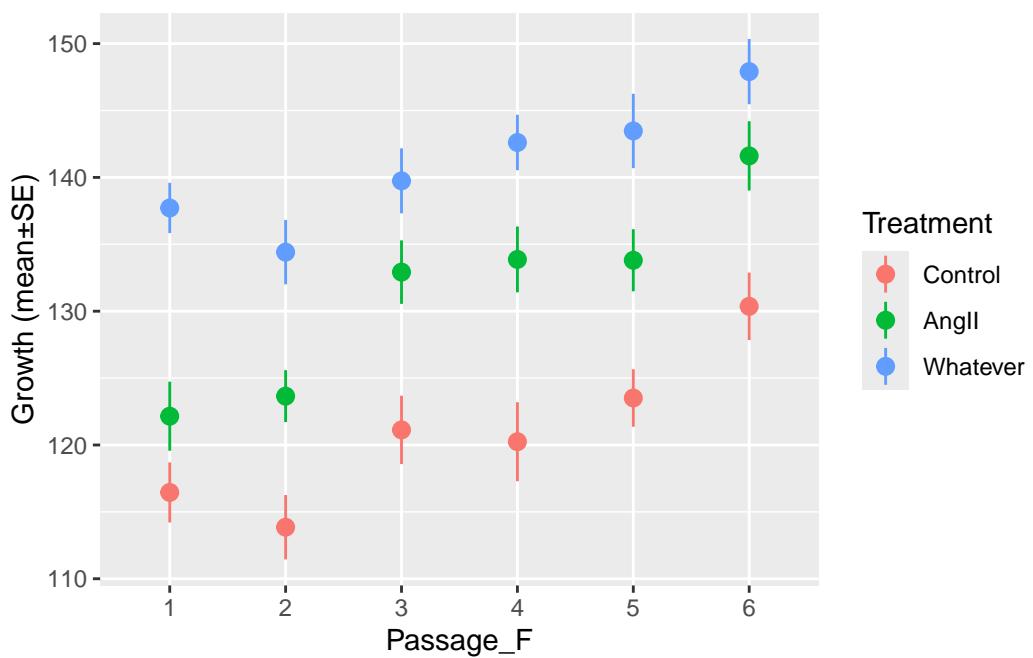
dplyr::select(-ZahlZellen) |>
  rename(Growth=Wachstum, Treatment=Bedingung) |>
  mutate(Passage_F=factor(Passage),
         Treatment=fct_recode(Treatment,
                               Control="Kontrolle"))

ggplot(rawdata,aes(Passage_F,Growth, color=Treatment))+  

  stat_summary(fun.data = mean_se)+  

  ylab("Growth (mean\u00b1SE)")

```



## 19.2 Random intercept model

### 19.2.1 Package nlme

```
lme_out <- nlme::lme(Growth~Treatment, data=rawdata,  
                      random = ~1|Passage_F) # RIntercept  
lme_out
```

```
Linear mixed-effects model fit by REML  
Data: rawdata  
Log-restricted-likelihood: -1367.644  
Fixed: Growth ~ Treatment  
          (Intercept) TreatmentAngII TreatmentWhatever  
            120.92499           10.40895          20.05199  
  
Random effects:  
Formula: ~1 | Passage_F  
          (Intercept) Residual  
StdDev:      5.644869 10.71643  
  
Number of Observations: 360  
Number of Groups: 6
```

```
Anova(lme_out)
```

```
Analysis of Deviance Table (Type II tests)  
  
Response: Growth  
          Chisq Df Pr(>Chisq)  
Treatment 210.17  2  < 2.2e-16 ***  
---  
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

### 19.2.2 Package lme4

```
lmer_out <- lme4::lmer(Growth~Treatment+(1|Passage_F),  
                       data=rawdata)  
  
lmer_out
```

```
Linear mixed model fit by REML ['lmerMod']
Formula: Growth ~ Treatment + (1 | Passage_F)
Data: rawdata
REML criterion at convergence: 2735.287
Random effects:
Groups      Name        Std.Dev.
Passage_F (Intercept) 5.645
Residual            10.716
Number of obs: 360, groups: Passage_F, 6
Fixed Effects:
(Intercept)    TreatmentAngII   TreatmentWhatever
              120.92           10.41            20.05
```

```
Anova(lmer_out)
```

Analysis of Deviance Table (Type II Wald chisquare tests)

```
Response: Growth
          Chisq Df Pr(>Chisq)
Treatment 210.17  2 < 2.2e-16 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

### 19.2.3 Visualization of fixed and random effects

Parameters can be conveniently extracted with `model_parameters()`. The intercept is the mean of the reference group (Control), the slopes are the differences to the reference group. The random effects are deviations from the overall intercept for each passage. The overall intercept and slope are shown as dashed line, the passage-specific intercepts with identical slopes as colored lines.

```
(lmer_out_param <- model_parameters(lmer_out, group_level = T))
```

```
# Fixed Effects
```

Parameter		Coefficient		SE		95% CI		t(355)		p
<hr/>										
(Intercept)		120.92		2.50		[116.00, 125.85]		48.30		< .001
Treatment [AngII]		10.41		1.38		[ 7.69, 13.13]		7.52		< .001
Treatment [Whatever]		20.05		1.38		[ 17.33, 22.77]		14.49		< .001

```
# Random Effects: Passage_F
```

Parameter		Coefficient		SE		95% CI
<hr/>						
(Intercept) [1]		-5.32		1.34		[-7.95, -2.69]
(Intercept) [2]		-6.70		1.34		[-9.34, -4.07]
(Intercept) [3]		0.17		1.34		[-2.46, 2.81]
(Intercept) [4]		1.10		1.34		[-1.54, 3.73]
(Intercept) [5]		2.37		1.34		[-0.26, 5.01]
(Intercept) [6]		8.38		1.34		[ 5.74, 11.01]

Uncertainty intervals (equal-tailed) and p-values (two-tailed) computed using a Wald t-distribution approximation.

```
intercept_all <- lmer_out_param$Coefficient[1]
slope_Ang <- lmer_out_param$Coefficient[2]
slope_What <- lmer_out_param$Coefficient[3]
plotdata <- tibble(
  Passage_F=lmer_out_param$Level[-(1:3)],
  intercept=lmer_out_param$Coefficient[-(1:3)]+
  intercept_all)

p1 <- rawdata |>
  filter(Treatment!="Whatever") |>
```

```

mutate(Treatment_n=case_match(Treatment, "Control" ~ 0,
                               .default =1)) |>
ggplot(aes(Treatment_n, Growth, color=Passage_F))+  

geom_beeswarm(alpha=.2, dodge.width = .2)+  

geom_abline(data = plotdata,  

            aes(color=Passage_F,  

                intercept=intercept,  

                slope=slope_Ang))+  

geom_abline(color='black',linetype=3,  

            aes(intercept=intercept_all,  

                slope=slope_Ang))+  

scale_x_continuous(breaks=0:1,  

                   limits = c(-.5,1.5),  

                   labels=c("Control","AngII"),  

                   minor_breaks = NULL)

p2 <- rawdata |>  

filter(Treatment!="AngII") |>  

mutate(Treatment_n=case_match(Treatment, "Control" ~ 0,
                               .default =1)) |>
ggplot(aes(Treatment_n, Growth, color=Passage_F))+  

geom_beeswarm(alpha=.2, dodge.width = .2)+  

geom_abline(data = plotdata,  

            aes(color=Passage_F,  

                intercept=intercept,  

                slope=slope_What))+  

geom_abline(color='black',linetype=3,  

            aes(intercept=intercept_all,  

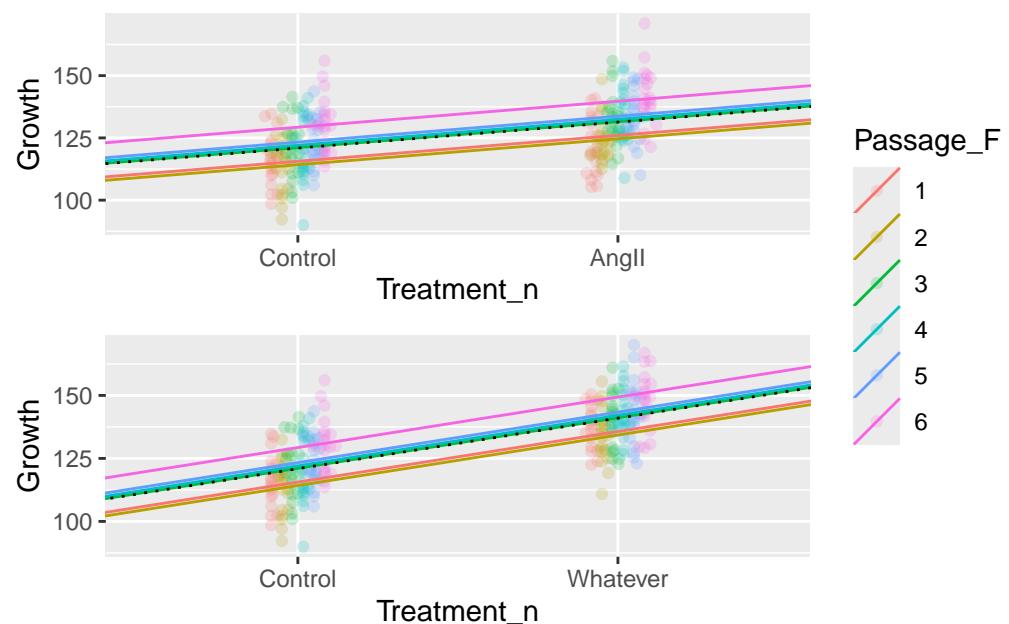
                slope=slope_What))+  

scale_x_continuous(breaks=0:1,  

                   limits = c(-.5,1.5),
                   labels=c("Control","Whatever"),
                   minor_breaks = NULL)

p1/p2+plot_layout(guides = "collect")

```



The regression lines for the passages are parallel but shifted, reflecting the random intercept model.

### 19.3 Random slope model

```
lmer_out2 <- lme4::lmer(Growth~Treatment+
                         (1+Treatment|Passage_F),
                         data=rawdata,
                         REML=FALSE)
```

```
boundary (singular) fit: see help('isSingular')
```

```
lmer_out2
```

```
Linear mixed model fit by maximum likelihood  ['lmerMod']
Formula: Growth ~ Treatment + (1 + Treatment | Passage_F)
Data: rawdata
      AIC      BIC      logLik -2*log(L)  df.resid
 2760.793 2799.654 -1370.397  2740.793      350
Random effects:
 Groups   Name        Std.Dev. Corr
 Passage_F (Intercept) 4.9984
          TreatmentAngII    1.3240    1.00
          TreatmentWhatever  0.9348  -1.00 -1.00
 Residual           10.6434
Number of obs: 360, groups: Passage_F, 6
Fixed Effects:
 (Intercept) TreatmentAngII TreatmentWhatever
      120.92          10.41            20.05
optimizer (nloptwrap) convergence code: 0 (OK) ; 0 optimizer warnings; 1 lme4 warnings
```

```
Anova(lmer_out2)
```

```
Analysis of Deviance Table (Type II Wald chisquare tests)
```

```
Response: Growth
          Chisq Df Pr(>Chisq)
Treatment 202.85  2 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

### 19.3.1 Visualization of fixed and random effects

```
(lmer_out_param2 <- model_parameters(lmer_out2, group_level = T))
```

# Fixed Effects

Parameter		Coefficient		SE		95% CI		t(350)		p
(Intercept)		120.92		2.26		[116.48, 125.37]		53.50		< .001
Treatment [AngII]		10.41		1.48		[ 7.50, 13.31]		7.05		< .001
Treatment [Whatever]		20.05		1.43		[ 17.25, 22.86]		14.06		< .001

# Random Effects: Passage\_F

Parameter		Coefficient		SE		95% CI
(Intercept) [1]		-5.37		1.27		[-7.87, -2.88]
(Intercept) [2]		-6.34		1.27		[-8.84, -3.85]
(Intercept) [3]		0.35		1.27		[-2.15, 2.84]
(Intercept) [4]		1.10		1.27		[-1.39, 3.60]
(Intercept) [5]		2.22		1.27		[-0.28, 4.72]
(Intercept) [6]		8.05		1.27		[ 5.55, 10.55]
TreatmentAngII [1]		-1.42		0.34		[-2.08, -0.76]
TreatmentAngII [2]		-1.68		0.34		[-2.34, -1.02]
TreatmentAngII [3]		0.09		0.34		[-0.57, 0.75]
TreatmentAngII [4]		0.29		0.34		[-0.37, 0.95]
TreatmentAngII [5]		0.59		0.34		[-0.07, 1.25]
TreatmentAngII [6]		2.13		0.34		[ 1.47, 2.79]
TreatmentWhatever [1]		1.00		0.24		[ 0.54, 1.47]
TreatmentWhatever [2]		1.19		0.24		[ 0.72, 1.65]
TreatmentWhatever [3]		-0.06		0.24		[-0.53, 0.40]
TreatmentWhatever [4]		-0.21		0.24		[-0.67, 0.26]
TreatmentWhatever [5]		-0.42		0.24		[-0.88, 0.05]
TreatmentWhatever [6]		-1.51		0.24		[-1.97, -1.04]

Uncertainty intervals (equal-tailed) and p-values (two-tailed) computed using a Wald t-distribution approximation.

```
intercept_all <- lmer_out_param2$Coefficient[1]
slope_Ang <- lmer_out_param2$Coefficient[2]
slope_What <- lmer_out_param2$Coefficient[3]
interceptdata <-
```

```

tibble(Passage_F=lmer_out_param2$Level[4:9],
       intercept=lmer_out_param2$Coefficient[4:9]+intercept_all,
       slopeAng=lmer_out_param2$Coefficient[10:15]+
         slope_Ang,
       slopeWhat=lmer_out_param2$Coefficient[16:21]+
         slope_What)

p1 <- rawdata |>
  filter(Treatment!="Whatever") |>
  mutate(Treatment_n=case_match(Treatment, "Control" ~ 0,
                                 .default =1)) |>
  ggplot(aes(Treatment_n, Growth, color=Passage_F))+  

  geom_beeswarm(alpha=.2, dodge.width = .2)+  

  geom_abline(data = interceptdata,
              aes(color=Passage_F,
                  intercept=intercept,
                  slope=slopeAng))+  

  geom_abline(color='black',
              aes(intercept=intercept_all,
                  slope=slope_Ang))+  

  scale_x_continuous(breaks=0:1,
                     limits = c(-.5,1.5),
                     labels=c("Control","AngII"))

p2 <- rawdata |>
  filter(Treatment!="AngII") |>
  mutate(Treatment_n=case_match(Treatment, "Control" ~ 0,
                                 .default =1)) |>
  ggplot(aes(Treatment_n, Growth, color=Passage_F))+  

  geom_beeswarm(alpha=.2, dodge.width = .2)+  

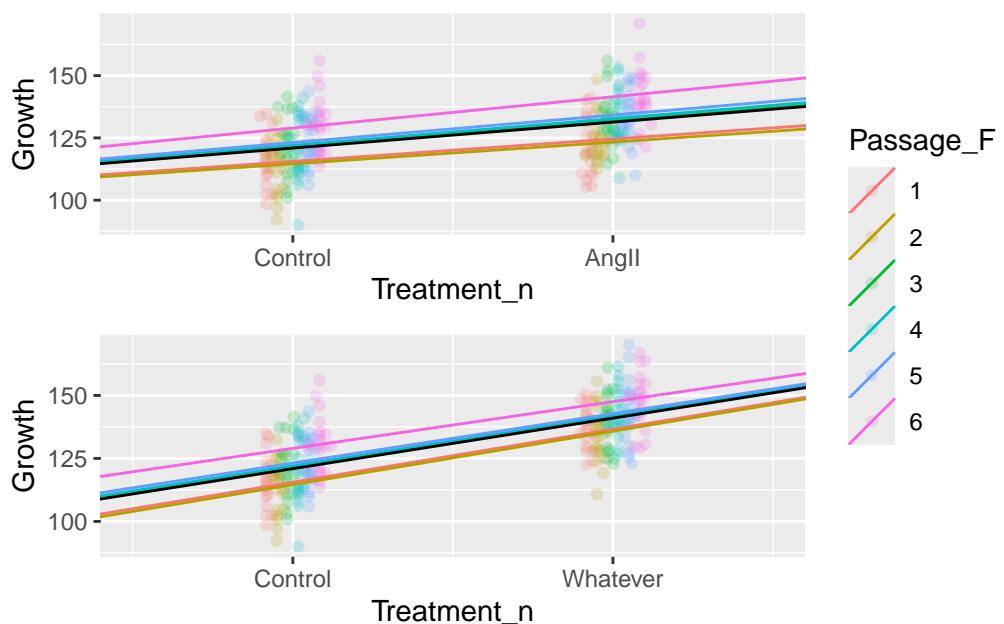
  geom_abline(data = interceptdata,
              aes(color=Passage_F,
                  intercept=intercept,
                  slope=slopeWhat))+  

  geom_abline(color='black',
              aes(intercept=intercept_all,
                  slope=slope_What))+  

  scale_x_continuous(breaks=0:1,
                     limits = c(-.5,1.5),
                     labels=c("Control","Whatever"))

p1/p2+plot_layout(guides = "collect")

```



This time, the Passage-related lines have individual intercepts AND slopes, indicating that the treatment effects are not exactly identical within the passages.

# 20 Machine Learning with R: Basic concepts

## 20.1 structured vs. unstructured data

## 20.2 supervised vs. unsupervised methods

## 20.3 Resampling methods

- Creation of new samples based on one observed sample.
- Helps in creating new synthetic datasets for training machine learning models and to estimate properties of a dataset when the dataset is unknown, difficult to estimate, or when the sample size of the dataset is small.
- Resampling Methods
  - Permutation tests (also re-randomization tests)
  - Bootstrapping
  - Jackknife (Leave one out validation)
  - Cross validation

### 20.3.1 Permutation tests

- Permutation reshuffles the observed cases, sampling without replacement, e.g. Random combination of 2 classes (like disease / risk)
- Original:

	Class 1	Class 2
Affected	Exposed	
Affected	Exposed	
Affected	-	
Control	-	
Control	-	

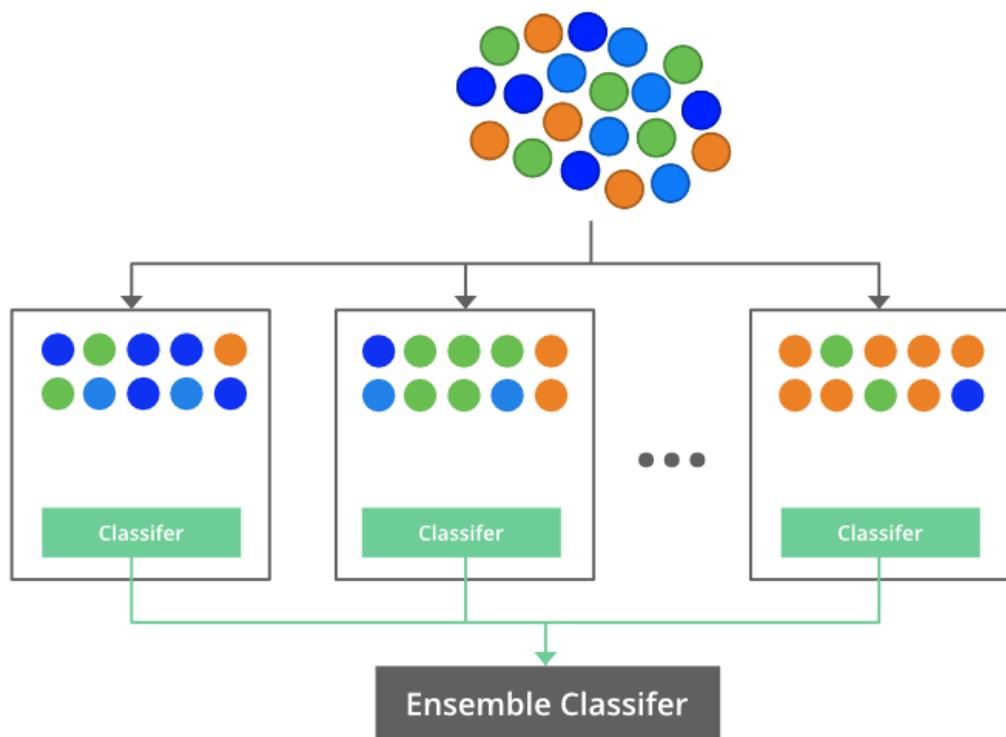
- Permutation 1

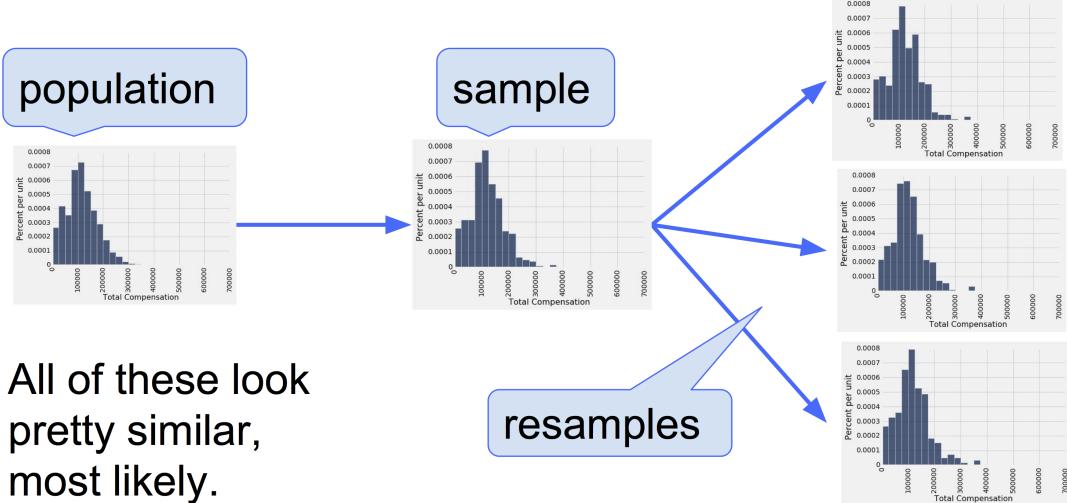
	Class 1	Class 2
Affected	-	
Affected	Exposed	
Affected	-	
Control	-	
Control	Exposed	

- repeated x times

### 20.3.2 Bootstrapping

- Bootstrapping selects from the population of observed cases, sampling with replacement.
- Bootstrap is a powerful statistical tool used to quantify the uncertainty of a given model.





All of these look pretty similar, most likely.

```
pacman::p_load(wrappedtools, boot, Hmisc, tidyverse)
```

```
n_subjects <- 50
real_mean <- 170
real_sd <- 10
n_boot <- 10^4
set.seed(12345)
experimental_data <-
  rnorm(n = n_subjects, mean = real_mean, sd = real_sd)
meansd(experimental_data, roundDig = 4)
```

```
[1] "171.8 ± 11.0"
```

```
meanse(experimental_data, roundDig = 4)
```

```
[1] "171.8 ± 1.6"
```

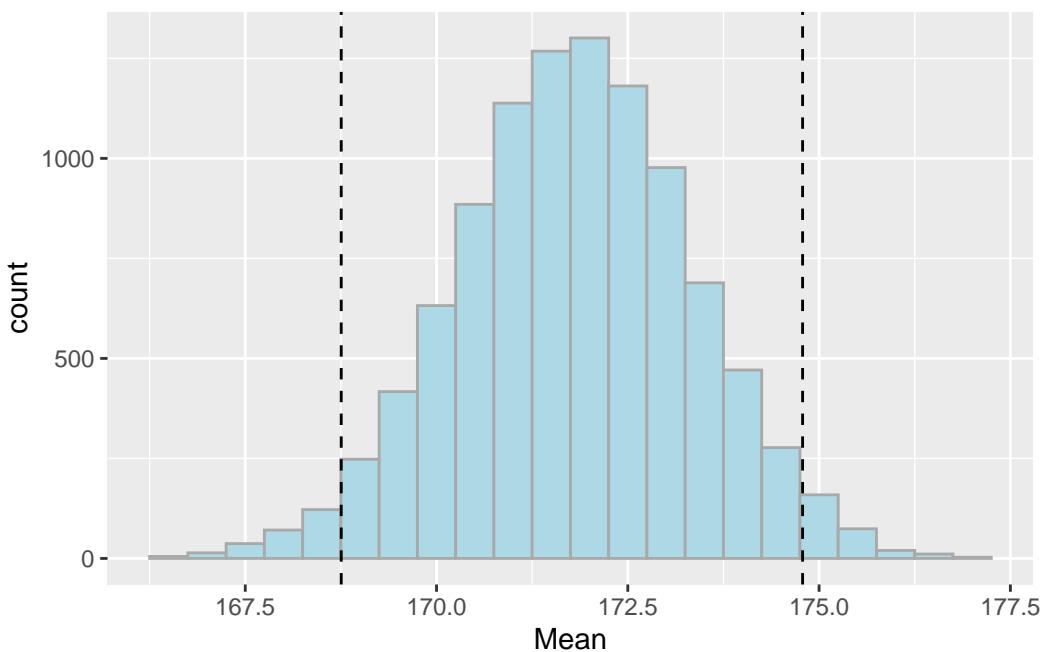
```
smean.cl.normal(experimental_data)
```

Mean	Lower	Upper
171.7957	168.6792	174.9121

```
smean.cl.boot(experimental_data, B = n_boot)
```

	Mean	Lower	Upper
171.7957	168.7258	174.8658	

```
#looped bootstrapping
boot_means <- tibble(Mean = NA_real_, .rows = 0)
for (run_i in seq_len(n_boot)){
  boot_means <-
    add_row(boot_means,
            Mean = mean(sample(x = experimental_data,
                                 size = n_subjects,
                                 replace = TRUE)))
}
cl_boot <- quantile(boot_means$Mean, probs = c(.025, .975))
ggplot(boot_means, aes(Mean))+
  geom_histogram(binwidth = .5, fill = "lightblue", color = "darkgrey")+
  geom_vline(xintercept = cl_boot, linetype = 2)
```



```
# package boot
# Custom function to calculate the mean for each resample
boot_mean <- function(d, i) {
  d[i] |> mean()
}
# Sample data
# Execute n_boot bootstrap replicates
```

```

boot_results <- boot(
  data = experimental_data,
  statistic = boot_mean,
  R = n_boot
)

# View the result
boot_results

```

#### ORDINARY NONPARAMETRIC BOOTSTRAP

Call:

```
boot(data = experimental_data, statistic = boot_mean, R = n_boot)
```

```

Bootstrap Statistics :
      original     bias    std. error
t1* 171.7957 -0.01456677   1.534077

```

```

# Get multiple types of CIs
boot_ci <- boot.ci(boot_results, type = "basic")

print(boot_ci)

```

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS  
Based on 10000 bootstrap replicates

```

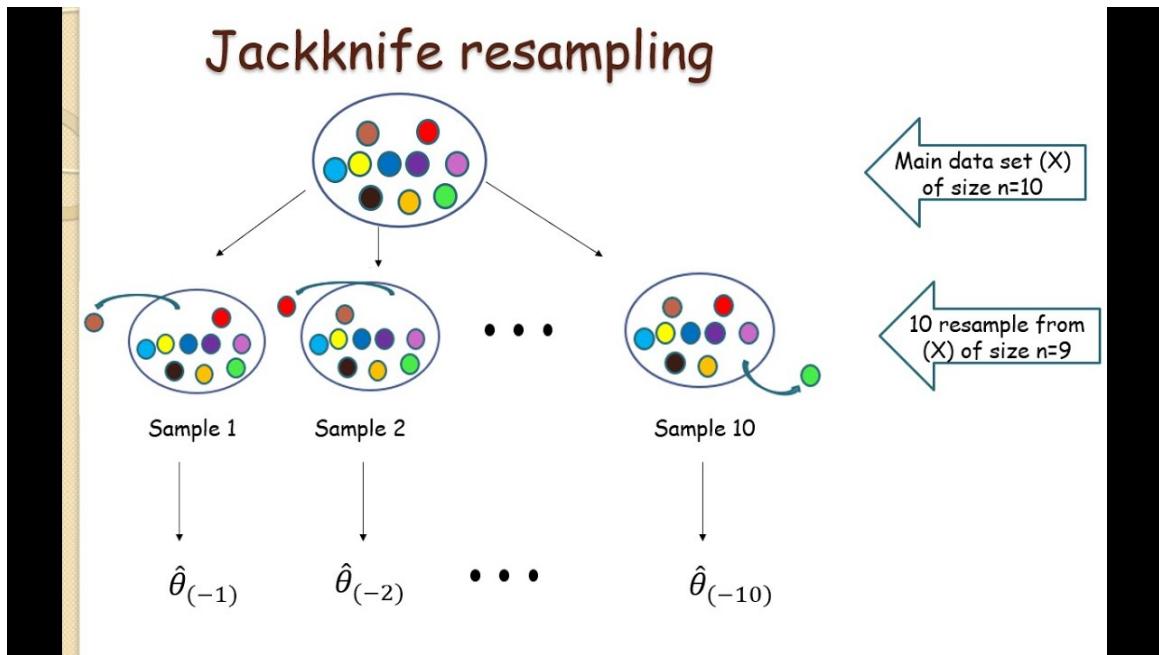
CALL :
boot.ci(boot.out = boot_results, type = "basic")

Intervals :
Level      Basic
95%   (168.8, 174.9 )
Calculations and Intervals on Original Scale

```

### 20.3.3 Jackknife

- Jackknife is resampling without replacement
- Creates n samples of size n-1

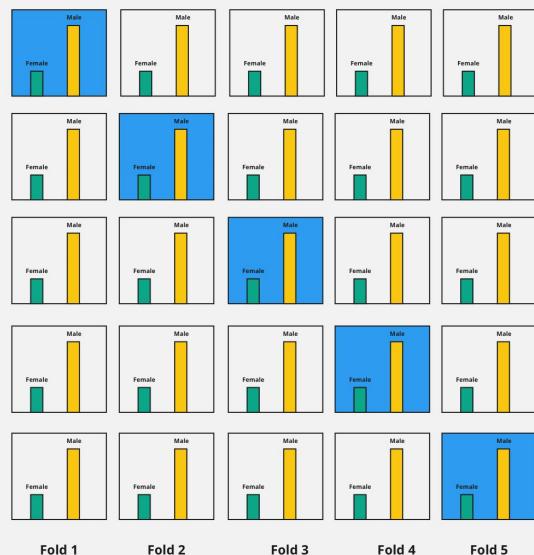
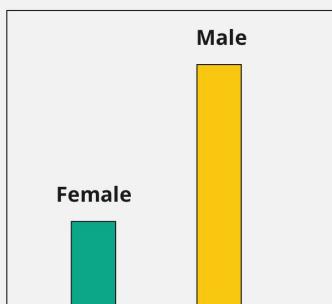


### 20.3.4 k-fold CV

- Splitting sample into  $k$  groups
- One group for testing,
- $k-1$  groups for training
- Repeated  $k$  times, no resampling

## Stratified K-Fold Cross Validation

Target Class Distribution



[dataaspirant.com](http://dataaspirant.com)

## 21 Markov Chain Monte Carlo: Metropolis algorithm simulation

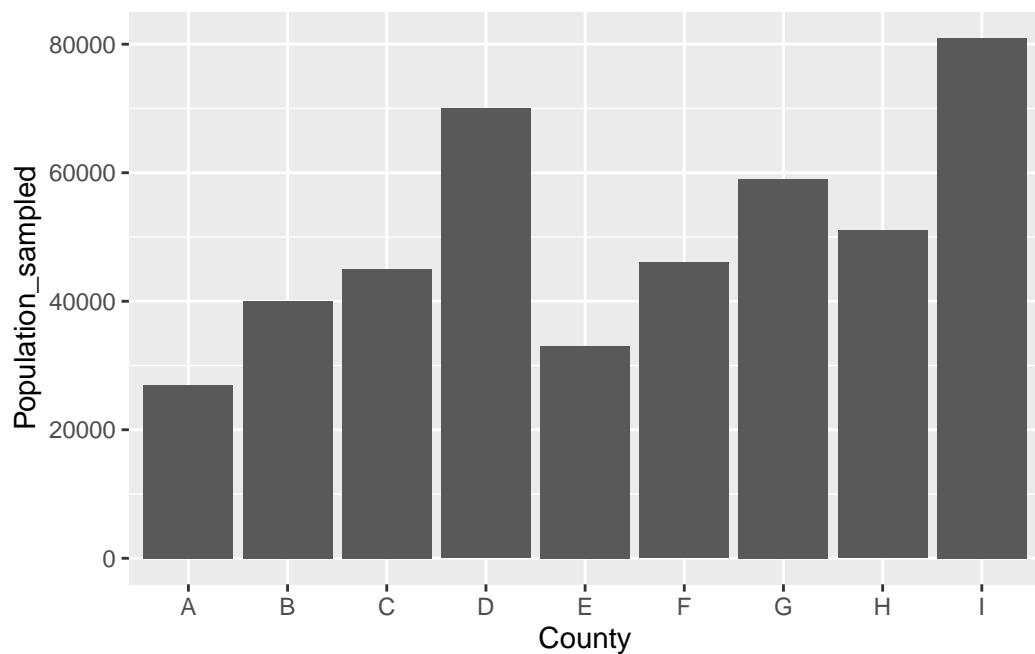
based on [Markov Chain Monte Carlo | Columbia Public Health](#) with slightly changed assumptions (10 counties, pre-defined populations either systematically chosen or randomized) and these rules:

1. Flip a coin. Heads to move east, tails to move west. If in borderline county, wrap around (alternative: move towards center?)
2. If the district indicated by the coin (east or west) has more voters than the present district, move there.
3. If the district indicated by the coin has fewer likely voters, make the decision based on a probability calculation:
4. calculate the probability of moving as the ratio of the number of likely voters in the proposed district, to the number of voters in the current district:
5.  $\text{Pr}[\text{move}] = \text{voters in indicated district}/\text{voters in present district}$
6. Take a random sample between 0 and 1.
7. If the value of the random sample is between 0 and the probability of moving, move. Otherwise, stay put.

```
pacman::p_load(wrappedtools, tidyverse, ggrepel, ggforce,
                 ggnewscale, ggtext, tictoc)

set.seed(1012)
counties <- tibble(County = LETTERS[1:9],
                    Population_defined=seq(from=10^4,
                                            to= 9*10^4,
                                            by=10^4),
                    Population_sampled = runif(n = 9,
                                                min = 10^4,
                                                max = 9*10^4) |>
                      roundR(level = 2,
                             textout = F,
                             smooth = T))
pop_selected <- 'Population_sampled'
ggplot(counties,aes(County,.data[[pop_selected]])) +
```

```
geom_col()
```



## 21.1 Rule definition

```
move_selection <- function(.counties=counties,
                           current_county,
                           which_population=pop_selected) {
  coinresult <- sample(x = c(1,-1),
                        size = 1)
  # if(current_county==1) {
  #   coinresult <- 1
  # }
  # if(current_county==nrow(.counties)) {
  #   coinresult <- -1
  # }
  next_county <- current_county+coinresult
  if(next_county==0) {next_county <- nrow(.counties)}
  if(next_county>nrow(counties)) {next_county <- 1}
  population_ratio <- .counties[[next_county,which_population]] /
    .counties[[current_county,which_population]]
  if(runif(n = 1,0,1)>population_ratio){
    next_county <- current_county
  }
}
```

```
    return(next_county)
}
```

```
n_moves <- 10^5
n_burnin <- 10^3
start_county <- 5
```

## 21.2 Data structures for simulation

```
moves <- tibble(move=seq_len(n_moves),
                 position=NA_integer_)
moves$position[1] <- start_county
```

## 21.3 Simulation

```
set.seed(1210)
tic toc::tic('here we go....')
for(step_i in 2:n_moves){
  moves$position[step_i] <-
    move_selection(current_county = moves$position[step_i-1])
}
tic toc::toc()
```

here we go....: 38.13 sec elapsed

## 21.4 Results

```
visits <- moves |>
  group_by(position) |>
  summarise(Visits=n()) |>
  ungroup() |>
  mutate(County = LETTERS[position]) |>
  select(-position) |>
  full_join(counties)
```

Joining with `by = join\_by(County)`

```

ggplot(visits,aes(.data[[pop_selected]],Visits))+  

  geom_smooth(method='lm')+  

  geom_abline(intercept = 0,  

              slope = n_moves/sum(counties[[pop_selected]]),  

              linetype=2)+  

  geom_point()+
  geom_label_repel(aes(label=County),nudge_x = 0, nudge_y = 100)+  

  scale_shape_manual(values=LETTERS, guide = NULL)+  

  scale_x_continuous(breaks=seq(0,10^5,10^4))+  

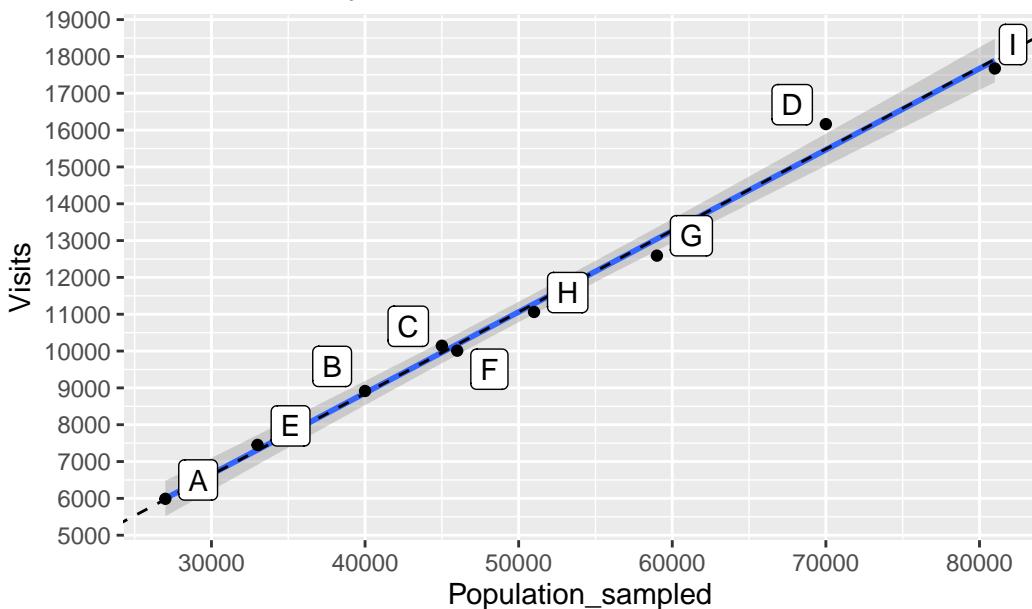
  scale_y_continuous(breaks=seq(0,10^5,10^3))+  

  ggtitle('All moves analyzed')

```

`geom\_smooth()` using formula = 'y ~ x'

All moves analyzed



```

visits <- moves |>  

  filter(move>n_burnin) |>  

  group_by(position) |>  

  summarise(Visits=n()) |>  

  ungroup() |>  

  mutate(County = LETTERS[position]) |>  

  select(-position) |>  

  full_join(counties)

```

Joining with `by = join\_by(County)`

```

moves_from_to <- expand.grid(1:9,1:9) |>
  as_tibble() |>
  rename(from=Var1,
         to=Var2) |>
  filter(abs(from-to)<2|abs(from-to)==8) |>
  mutate(count=0,
         start=LETTERS[from],
         stop=LETTERS[to])
for(move_i in seq_len(nrow(moves_from_to))){
  moves_from_to$count[move_i] <-
    sum(moves$position[-nrow(moves)]==moves_from_to$from[move_i] &
        moves$position[-1]==moves_from_to$to[move_i])
}
moves_to <- moves_from_to |>
  filter(to!=from) |> group_by(to) |> summarize(moves_to=sum(count)) |>
  mutate(County=LETTERS[to])
moves_from <- moves_from_to |>
  filter(to!=from) |> group_by(from) |> summarize(moves_from=sum(count)) |>
  mutate(County=LETTERS[from])
moves_stay <- moves_from_to |>
  filter(to==from) |> group_by(from) |> summarize(moves_stay=sum(count)) |>
  mutate(County=LETTERS[from])
moves_from_to_stay <-
  full_join(moves_from,moves_to) |>
  full_join(moves_stay) |>
  full_join(counties |> select(-Population_defined)) |>
  select(-from,-to) |>
  pivot_longer(cols=c(moves_from,moves_to,moves_stay),
               names_to='move',
               values_to='count') |>
  mutate(County=paste0(County,"\n",
                      round(Population_sampled/1000),
                      "k"
))

```

Joining with `by = join\_by(County)`  
 Joining with `by = join\_by(from, County)`  
 Joining with `by = join\_by(County)`

```

ggplot(moves_from_to_stay,aes(County,count, fill=move))+  

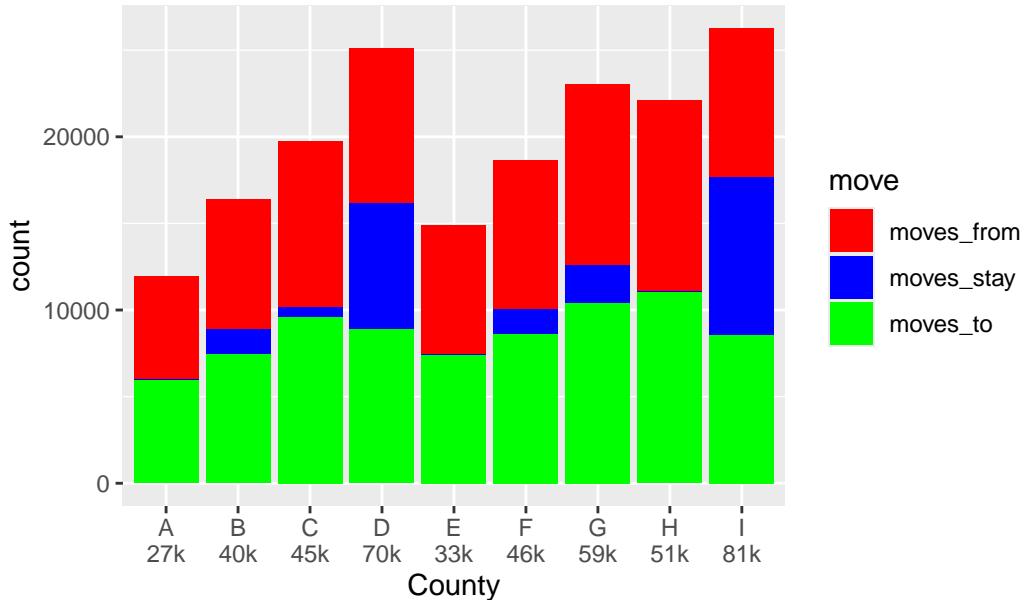
  geom_col()  

  scale_fill_manual(values=c('red','blue','green'))+  

  ggtitle('Moves from, to, and stay')

```

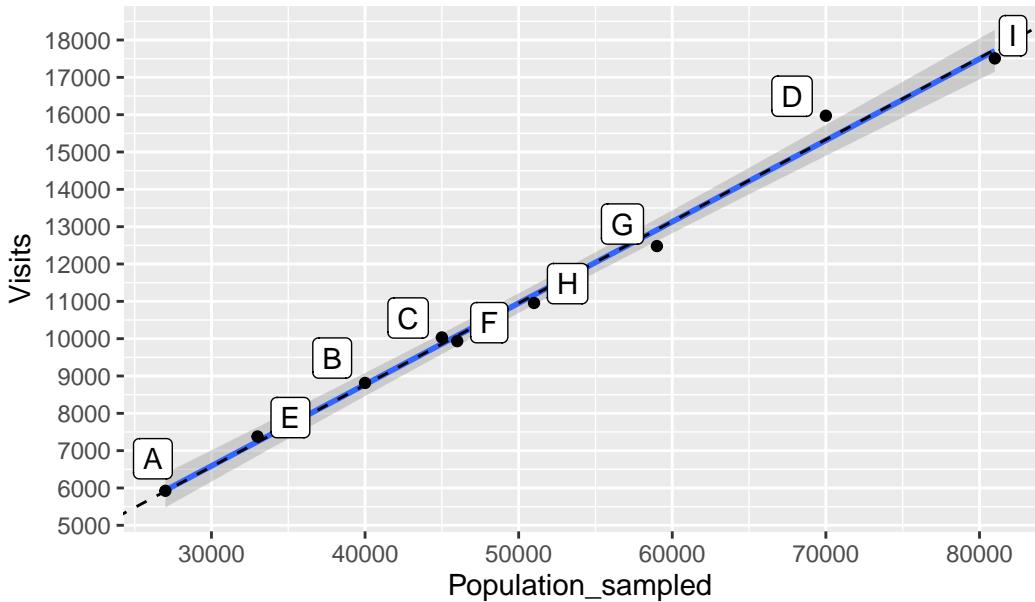
## Moves from, to, and stay



```
ggplot(visits,aes(.data[[pop_selected]],Visits))+  
  geom_smooth(method='lm')+  
  geom_abline(intercept = 0,  
             slope = (n_moves-n_burnin)/sum(counties[[pop_selected]]),  
             linetype=2)+  
  geom_point() +  
  geom_label_repel(aes(label=County),nudge_x = 0, nudge_y = 100)+  
  scale_shape_manual(values=LETTERS, guide = NULL)+  
  scale_x_continuous(breaks=seq(0,10^5,10^4))+  
  scale_y_continuous(breaks=seq(0,10^5,10^3))+  
  ggtitle('Only moves after burn-in analyzed')
```

`geom\_smooth()` using formula = 'y ~ x'

Only moves after burn-in analyzed



```
# circle for county plot
# Define circle aesthetics
theta <- seq(0, 2*pi, length.out = 10)[c(4:9,1:3)] |>
  rev()
# Create sequence for angles (0 to 2*pi) with 5 equally spaced points
radius <- 1 # Set radius of the circle

# Create data frame with circle coordinates and labels
circle_data <- tibble(
  angle=theta,
  x = radius * cos(theta),
  y = radius * sin(theta),
  label =LETTERS[1:9] # Assign letters A to E as labels
) |>
  full_join(visits, by=c('label'='County')) |>
  mutate(plotlabel=paste0(label,"\\n",
                         round(Population_sampled/1000),
                         "k"
                         ))
arrow_data <-
  moves_from_to |>
  full_join(circle_data |>
    select(x:label, angle), by=c('start'='label')) |>
  rename(from_x="x",from_y="y") |>
  full_join(circle_data |>
```

```

    select(x:label), by=c('stop'='label')) |>
  rename(to_x="x",to_y="y") |>
  mutate(x_end=from_x+count*10/n_moves*cos(angle),
         y_end=from_y+count*10/n_moves*sin(angle),
         count=case_when((from<to & !(from==1 & to==9)) | (from==9 & to==1)--count,
                         .default=count))

# Createggplot with circle and labels
# ggplot(circle_data, aes(x = x, y = y)) +
#   # geom_point(aes(size = Visits), shape=1) + # Increase point size for better visibility
#   geom_circle(aes(r=1,x0=0,y0=0),
#               color="darkorange2")+
#   geom_curve(data=arrow_data |>
#               filter(from<to),
#               aes(x=from_x,y=from_y,
#                   xend=to_x,yend=to_y,
#                   linewidth=count),
#               arrow=arrow(length=unit(0.1,"inches")),
#               curvature=-1.0,
#               color="darkolivegreen")+
#   geom_curve(data=arrow_data |>
#               filter(from>to),
#               aes(x=from_x,y=from_y,
#                   xend=to_x,yend=to_y,
#                   linewidth=count),
#               arrow=arrow(length=unit(0.1,"inches")),
#               angle=90,
#               curvature=-.75,
#               color="dodgerblue")+
#   geom_text(aes(label = plotlabel, size=.data[[pop_selected]]),
#             hjust = 0.5, vjust = 0.5) + # Adjust text position slightly
#   scale_size_continuous(range=c(3,7)) + # Set size of labels
#   scale_linewidth_continuous(range=c(.5,3)) + # Set size of labels
#   coord_fixed(xlim = c(-radius - radius/5, radius + radius/5), ylim = c(-radius - radius/5, radius + radius/5))
#   labs(title = "County population and move count", x = "", y = "",
#        caption = "inner arrows: moves to left neighbor,\nouter arrows: moves to right")
#   guides(size="none", linewidth="none")+
#   theme_void() # Remove background gridlines

# Create ggplot with circle and labels
ggplot(circle_data, aes(x = x, y = y)) +
  # geom_point(aes(size = Visits), shape=1) + # Increase point size for better visibility

```

```

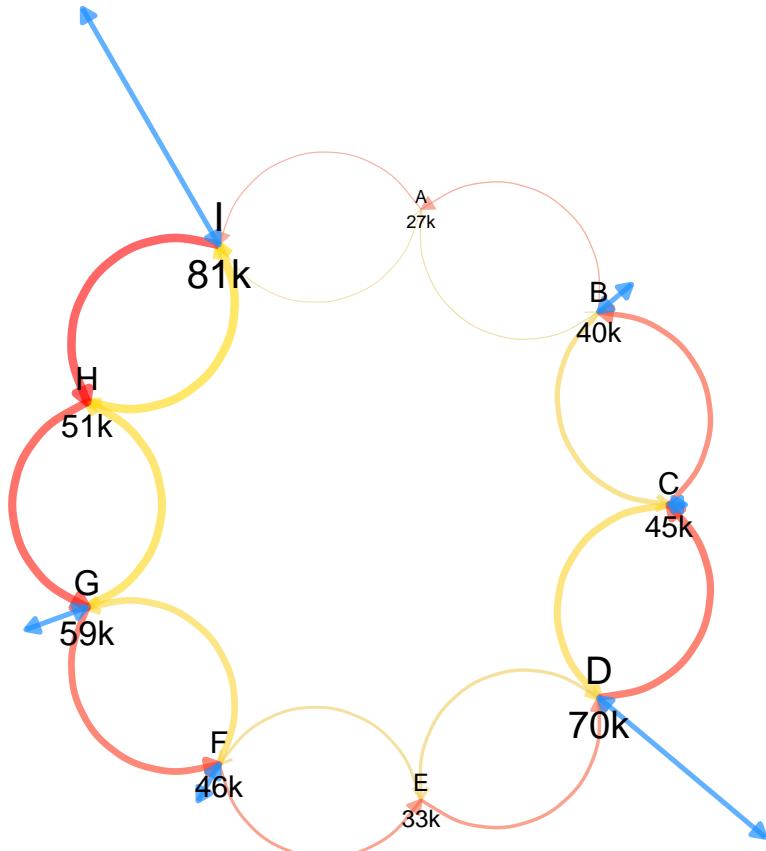
# geom_circle(aes(r=1,x0=0,y0=0),
#             color="darkorange2")+
geom_curve(data=arrow_data |>
    filter((from<to & !(to==9 & from==1)) | (to==1 & from==9)),
    aes(x=from_x,y=from_y,
        xend=to_x,yend=to_y,
        color=count, linewidth=abs(count)),
    arrow=arrow(length=unit(0.1,"inches")),
    curvature=.75,
    alpha=.6)+#, linewidth=1.5,)+
scale_colour_gradient2(low="gold", mid="grey",high="red")+
# scale_color_gradient("move count", low = "gold",high = "gold4") +
geom_curve(data=arrow_data |>
    filter(from>to &! (from==9 & to==1) | (to==9 & from==1)),
    aes(x=from_x,y=from_y,
        xend=to_x,yend=to_y,
        color=count, linewidth=abs(count)),
    arrow=arrow(length=unit(0.1,"inches"),
                type="closed"),
    # angle=90,#linewidth=1.5,
    curvature=.75,
    alpha=.6)+

geom_segment(data=arrow_data |>
    filter(from==to,count>0),
    aes(x=from_x,y=from_y,
        xend=x_end,
        yend=y_end),
    arrow=arrow(length=unit(0.1,"inches"),
                ends = "both",
                type="closed"),
    color='dodgerblue', linewidth=1.2, alpha=.7)+

geom_text(aes(label = plotlabel, size=.data[[pop_selected]]),
          hjust = 0.5, vjust = 0.5) + # Adjust text position slightly
scale_size_continuous(range=c(3,7)) + # Set size of labels
scale_linewidth_continuous(range = c(.25,2))+# Set size of labels
coord_fixed(xlim = c(-radius * 1.75, radius * 1.75),
            ylim = c(-radius * 1.75, radius * 1.75)) + # Set axis limits slightly bigger
labs(title = "County population and move count", x = "", y = "",
      caption = "inner golden arrows: moves to right neighbor (clockwise),\\nouter red dashed arrows: moves to left neighbor (counter-clockwise)"),
guides(size="none", linewidth="none", color="none")+
theme_void() # Remove background gridlines

```

County population and move count



inner golden arrows: moves to right neighbor (clockwise),  
outer reddish arrows: moves to left neighbor (counter-clockwise)  
straight blue arrows: stay put

```
visits <- moves |>
  filter(move<=n_burnin) |>
  group_by(position) |>
  summarise(Visits=n()) |>
```

```

ungroup() |>
mutate(County = LETTERS[position]) |>
select(-position) |>
full_join(counties)

```

Joining with `by = join\_by(County)`

```

ggplot(visits,aes(.data[[pop_selected]],Visits))+  

geom_smooth(method='lm')+  

geom_abline(intercept = 0,  

slope = n_burnin/sum(counties[[pop_selected]]),  

linetype=2)+  

geom_point() +  

geom_label_repel(aes(label=County),nudge_x = 0, nudge_y = 10)+  

scale_shape_manual(values=LETTERS, guide = NULL)+  

scale_x_continuous(breaks=seq(0,10^5,10^4))+  

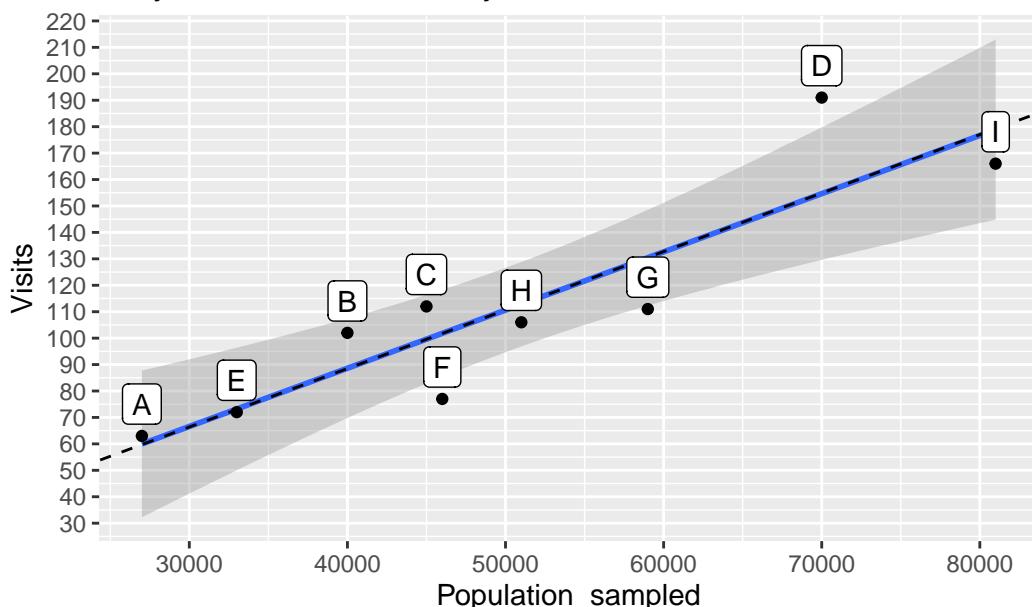
scale_y_continuous(breaks=seq(0,10^5,10^1))+  

ggtitle('Only burn-in moves analyzed')

```

`geom\_smooth()` using formula = 'y ~ x'

### Only burn-in moves analyzed



```

moves |>
  filter(move<=n_burnin) |>
  ggplot(aes(move,position))+  

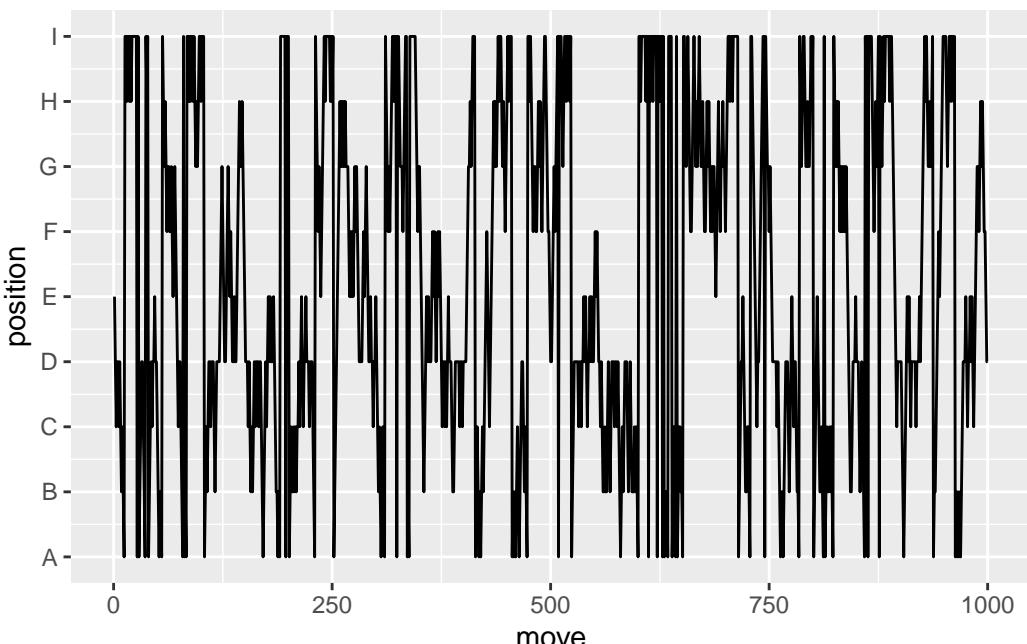
  # geom_point() +  

  geom_line() +  

  scale_y_continuous(breaks=1:9,  

                     labels = LETTERS[1:9])

```



```

moves |>
  filter(move<=100) |>
  ggplot(aes(move,position))+  

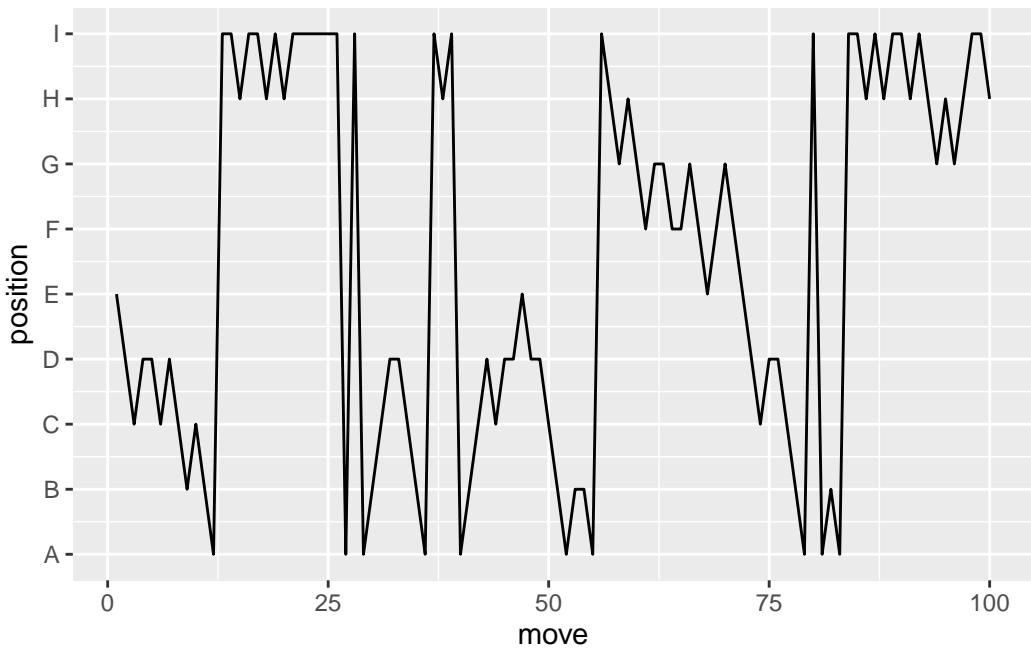
  # geom_point() +  

  geom_line() +  

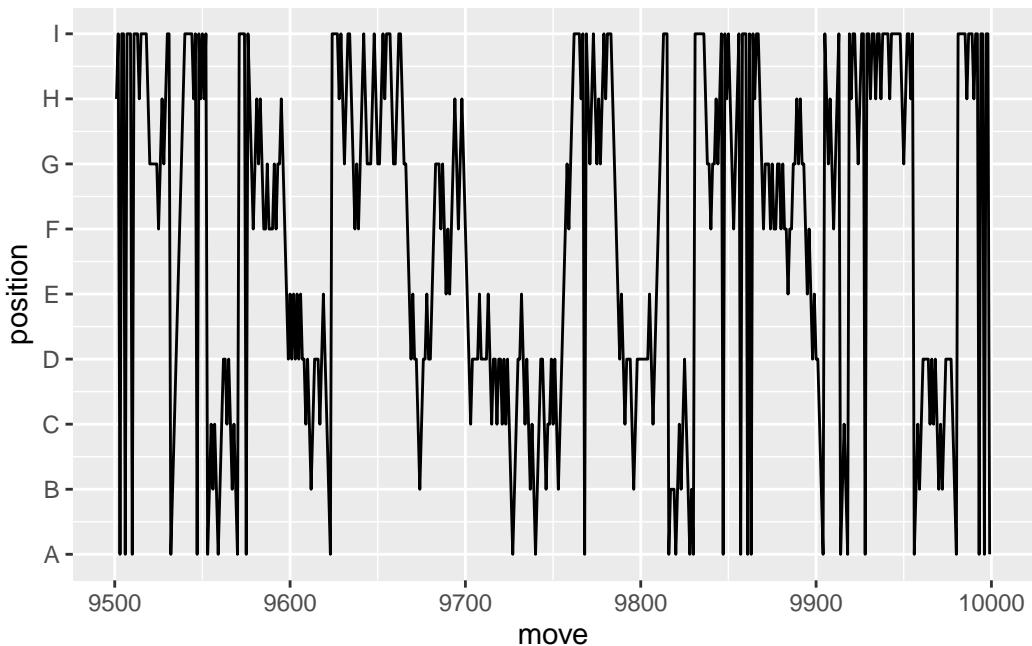
  scale_y_continuous(breaks=1:9,  

                     labels = LETTERS[1:9])

```

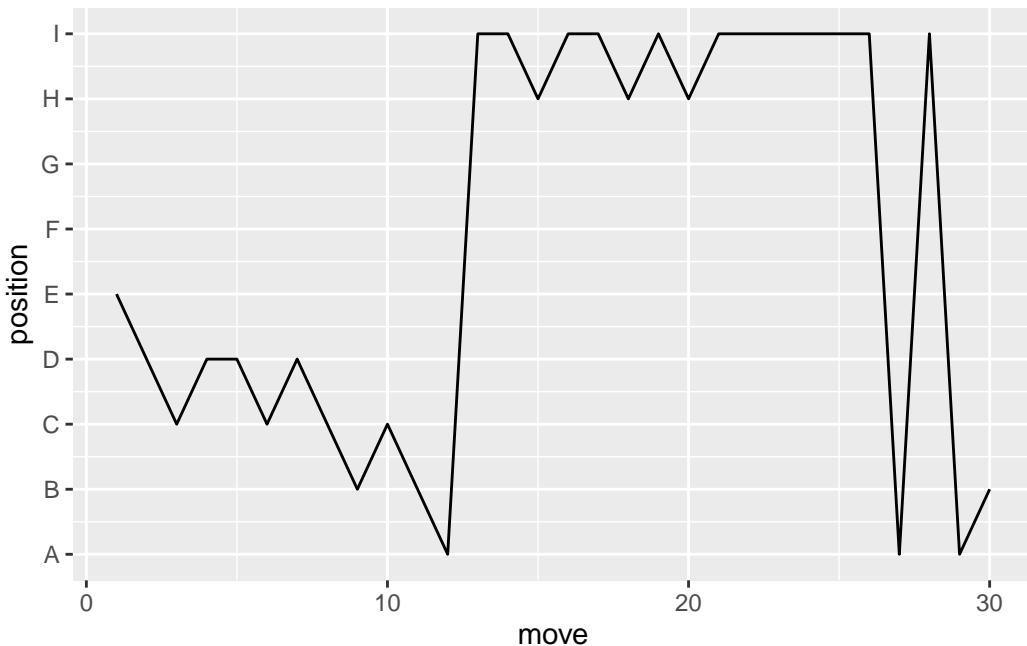


```
moves |>
  filter(move>9500, move<10000) |>
  ggplot(aes(move,position))+
  # geom_point()+
  geom_line()+
  scale_y_continuous(breaks=1:9,
                     labels = LETTERS[1:9])
```

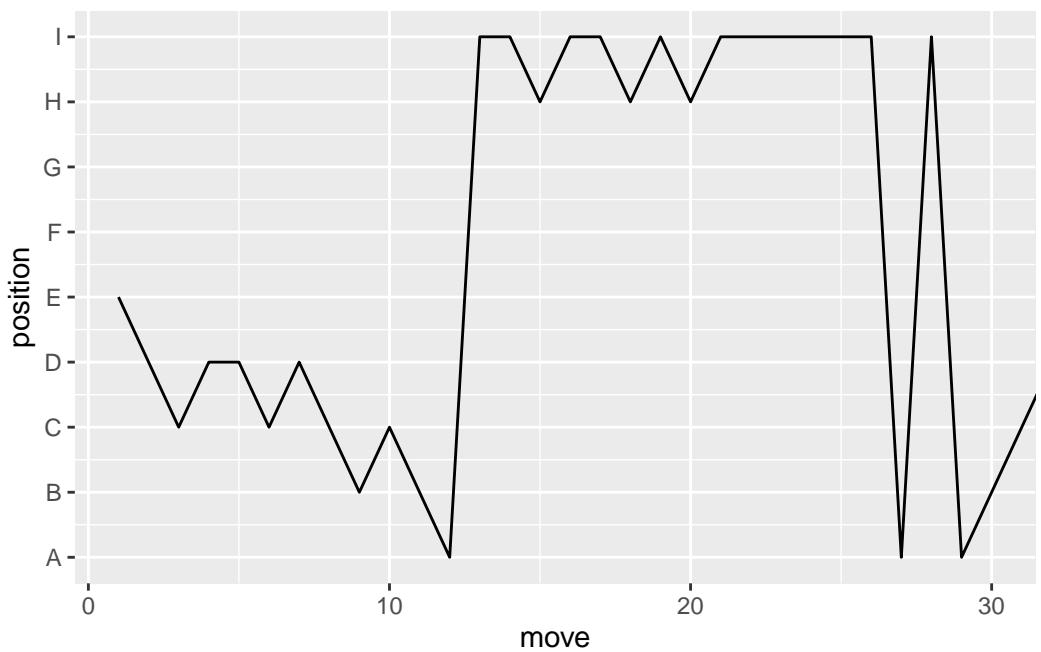


```
moves |>
  # filter(move<=n_burnin) |>
  ggplot(aes(move,position))+
  # geom_point()+
  geom_line()+
  scale_x_continuous(limits = c(1,30))+ 
  scale_y_continuous(breaks=1:9,
                     labels = LETTERS[1:9])
```

Warning: Removed 99970 rows containing missing values or values outside the scale range (`geom\_line()`).



```
moves |>
  # filter(move<=n_burnin) |>
  ggplot(aes(move,position))+
    # geom_point()+
    geom_line()+
    # scale_x_continuous(limits = c(1,30))+ 
    scale_y_continuous(breaks=1:9,
      labels = LETTERS[1:9])+ 
    coord_cartesian(xlim=c(1,30))
```



## 22 k nearest neighbors knn

```
# Bioconductor packages needed!
if(!requireNamespace("BiocManager", quietly = TRUE)) {
  install.packages("BiocManager")
  BiocManager::install(version=BiocManager::version())
}

if(!requireNamespace("preprocessCore", quietly = TRUE)) {
  BiocManager::install("preprocessCore")
}

pacman::p_load(conflicted,
                tidyverse,
                wrappedtools, # just tools
                palmerpenguins, # data
                ggforce, # for cluster plots, hulls, zoom etc
                ggbeeswarm,
                flextable,
                caret, # Classification and Regression Training Framework
                tidymodels, # alternative to caret
                preprocessCore, # pre-processing functions
                gmodels, # tools for model fitting
                easystats,
                yardstick) # model performance

# conflict_scout()
conflicts_prefer(dplyr::slice,
                  dplyr::filter,
                  palmerpenguins::penguins)
```

```
[conflicted] Will prefer dplyr::slice over any other package.
[conflicted] Will prefer dplyr::filter over any other package.
[conflicted] Will prefer palmerpenguins::penguins over any other package.
```

## 22.1 Data preparation

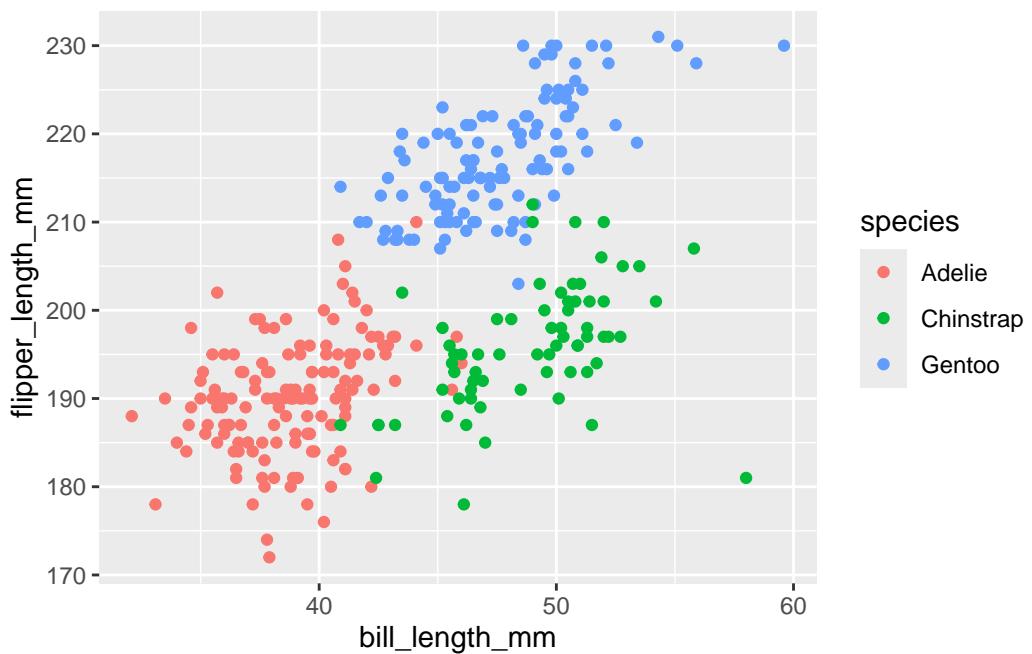
In the penguin data set, the four numerical variables related to body are `bill_length_mm`, `bill_depth_mm`, `flipper_length_mm`, and `body_mass_g`. We will use the first two variables to demonstrate the use knn.

```
rawdata <- penguins |>
  drop_na()
rawdata <- mutate(rawdata,
  ID=paste('P', 1:nrow(rawdata))) |>
  select(ID, everything())
predvars <- ColSeeker(rawdata,'length')
```

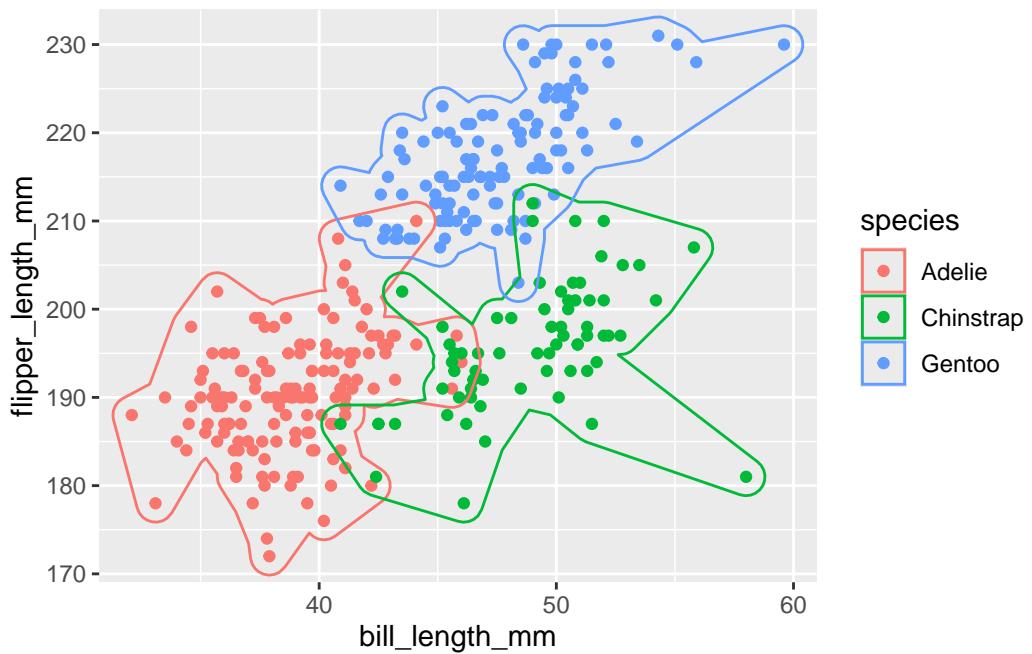
## 22.2 Exploratory plots

Package `ggforce` provides functions to plot convex hulls, ellipses, and zoomed facets. This is helpful in exploring group separation across 2 dimensions.

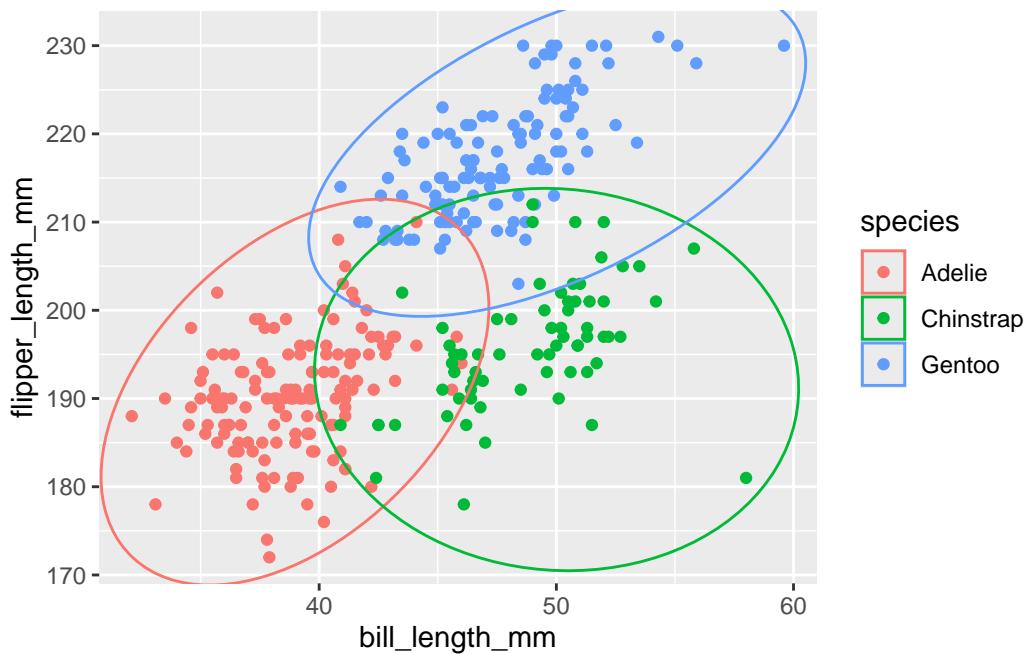
```
rawplot <-
  ggplot(rawdata,
    aes(.data[[predvars$names[1]]],
        .data[[predvars$names[2]]]),
    color=species))+
  geom_point()
rawplot
```



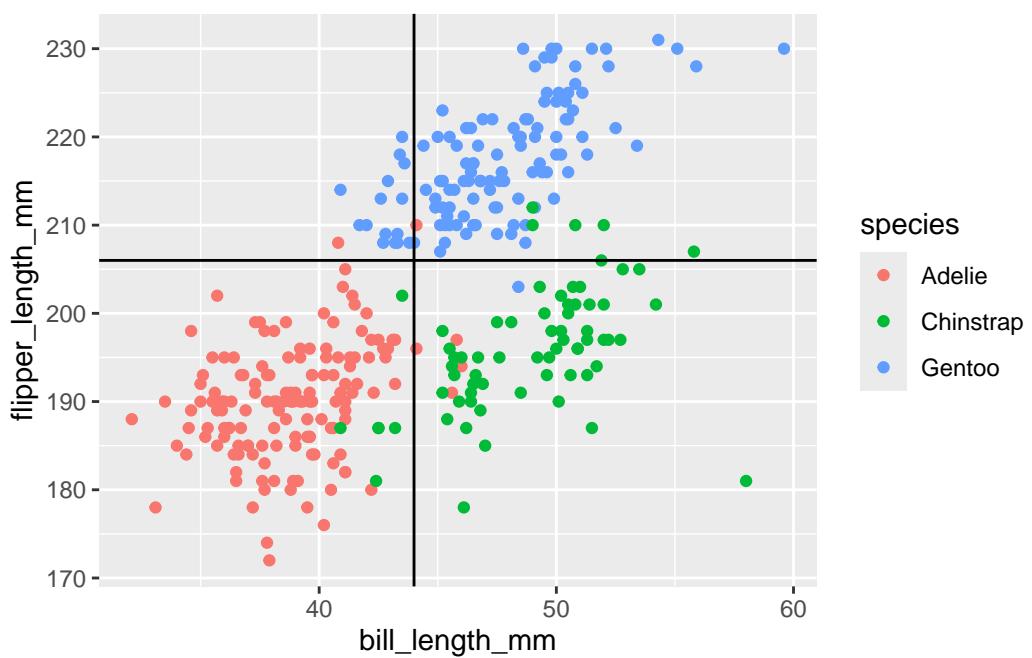
```
rawplot+
  geom_mark_hull(expand = unit(2.5, 'mm'))
```



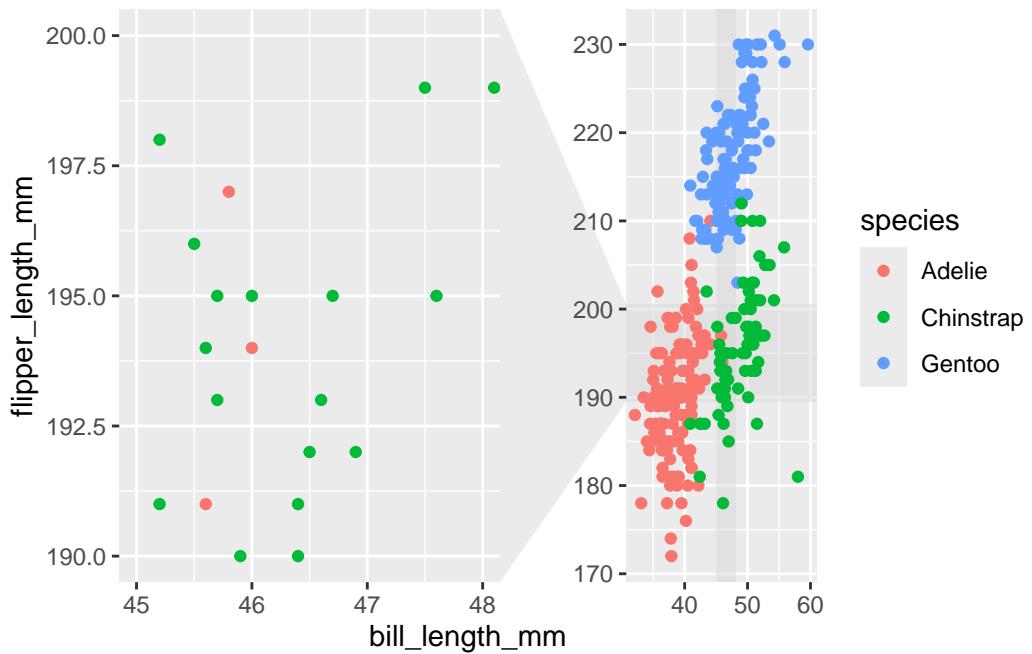
```
rawplot+
  geom_mark_ellipse(expand = unit(2.5, 'mm'))
```



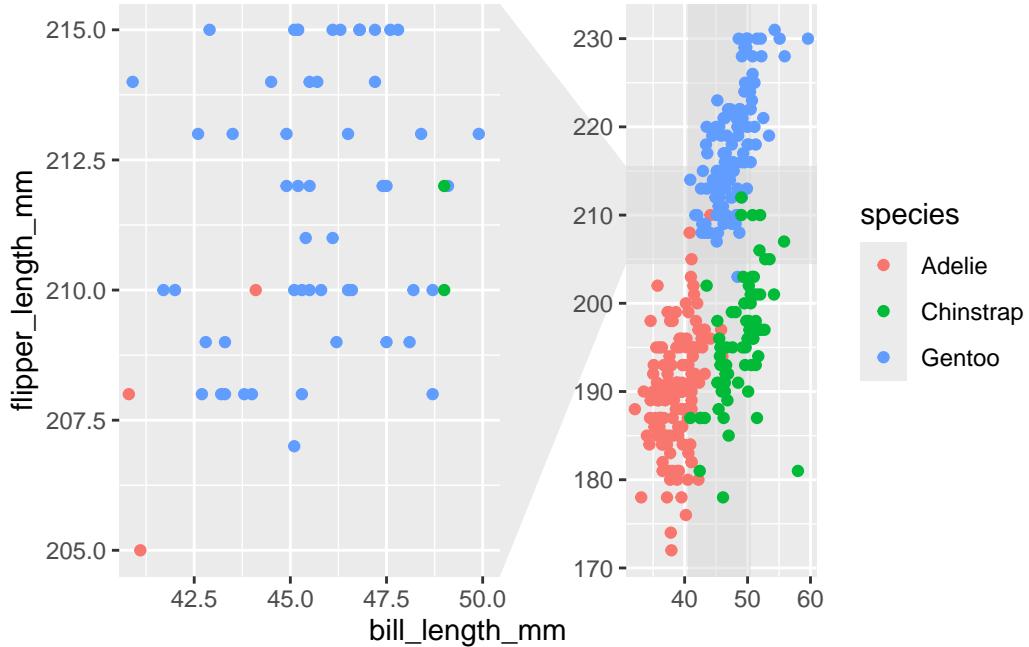
```
rawplot+
  geom_hline(yintercept = 206) +
  geom_vline(xintercept = 44)
```



```
rawplot+
  facet_zoom(xlim = c(45,48),
              ylim = c(190,200))
```



```
rawplot+
  facet_zoom(xlim = c(41,50),
              ylim = c(205,215))
```



## 22.3 Scaling

To avoid the effect of different scales of the variables, it is desirable to scale the variables. Depending on the distribution of the independent / predictor variables, there are a number of approaches to make center and spread/range of the data comparable. Here we first use the `preProcess` function from the `caret` package. To compare the scaled data to the original, we'll keep the original data in the data frame.

### 22.3.1 caret function `preProcess`

```
scaled <- rawdata |>
  select(predvars$names) |>
  caret::preProcess(method = c('center', "scale"))
rawdata <- predict(scaled, rawdata) |>
  select(ID, all_of(predvars$names)) |>
  rename_with(.cols=predvars$names,
              ~paste0(.x, "_std")) |>
  full_join(select(rawdata, -contains("_std")),
            by='ID')
```

```
scaled2 <- rawdata |>
  select(predvars$names) |>
```

```

caret::preProcess(method = c('range'))
rawdata <- predict(scaled2, rawdata) |>
  select(-ID, all_of(predvars$names)) |>
  rename_with(.cols = predvars$names,
              ~paste0(.x, "_rng")) |>
  full_join(select(rawdata, -contains("_rng")),
            by = 'ID')

```

### 22.3.2 tidymodel approach

```

# Build recipe for scaling
model_formula <-
  paste("~", paste(predvars$names, collapse = "+")) |>
  as.formula()
rec_normalized <- recipe(model_formula, data = rawdata) |>
  step_normalize(all_of(predvars$names))

# Fit recipe and apply to data
penguins_normalized <- rec_normalized |>
  prep() |>
  bake(new_data = rawdata)

rec_ranged <- recipe(model_formula, data = rawdata) |>
  step_range(all_of(predvars$names), min = 0, max = 1)

penguins_ranged_tidy <- rec_ranged |>
  prep() |>
  bake(new_data = rawdata)

```

### 22.3.3 preprocessCore function normalize.quantiles

For ...omics data, bioconductor provides functions for ML including preprocessing. Here we use the `normalize.quantiles` function from the `preprocessCore` package.

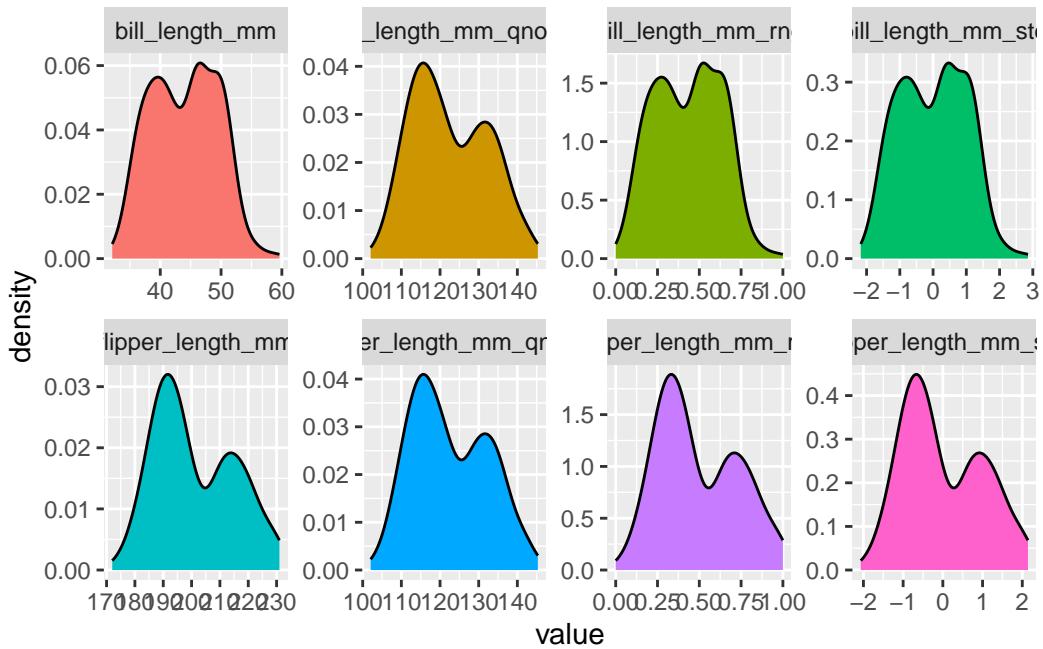
```

rawdata <- rawdata |>
  select(predvars$names) |>
  as.matrix() |>
  preprocessCore::normalize.quantiles(keep.names = TRUE) |>
  as_tibble() |>
  rename_with(~paste0(., '_qnorm')) |>
  bind_cols(rawdata)

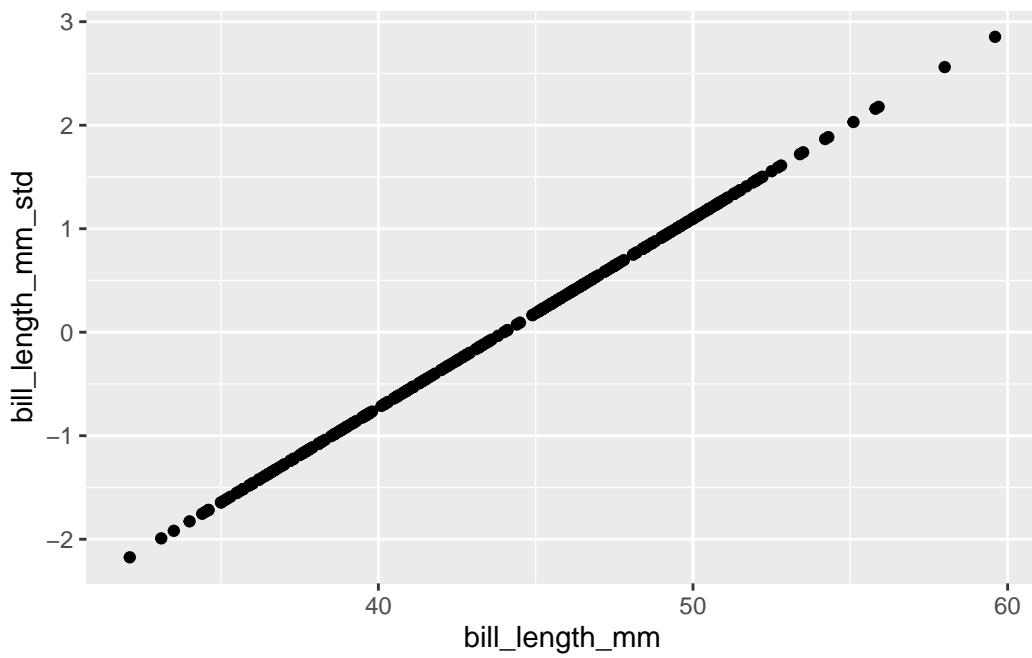
```

#### 22.3.4 Visual comparison of the scaled data

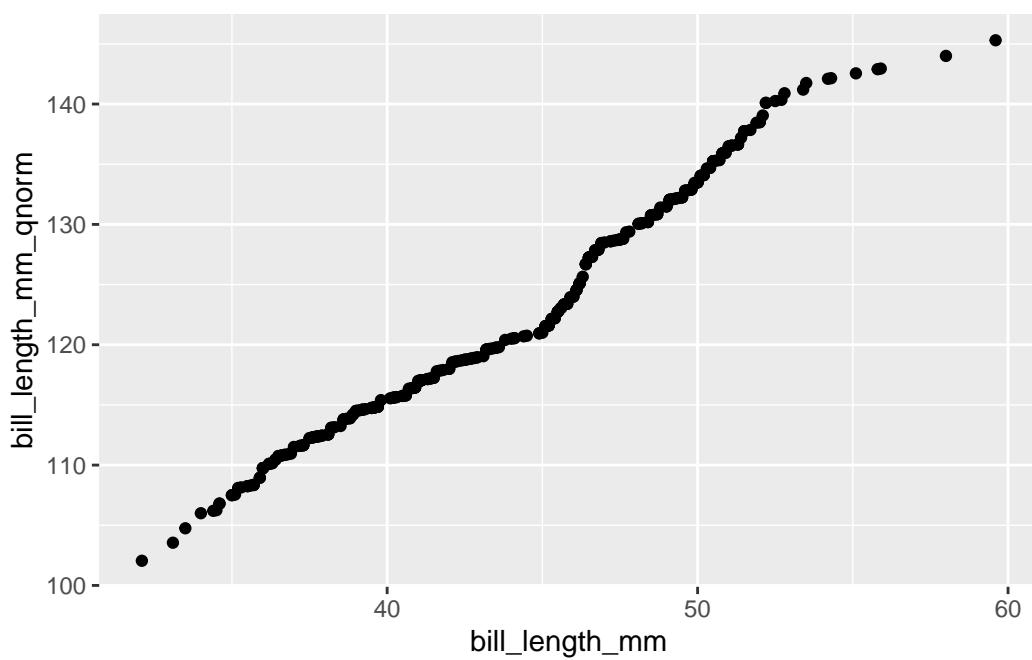
```
rawdata |>
  select(contains('length')) |>
  pivot_longer(everything()) |>
  ggplot(aes(value, fill=name)) +
  geom_density() +
  facet_wrap(facets = vars(name), scales='free', nrow = 2) +
  guides(fill="none")
```



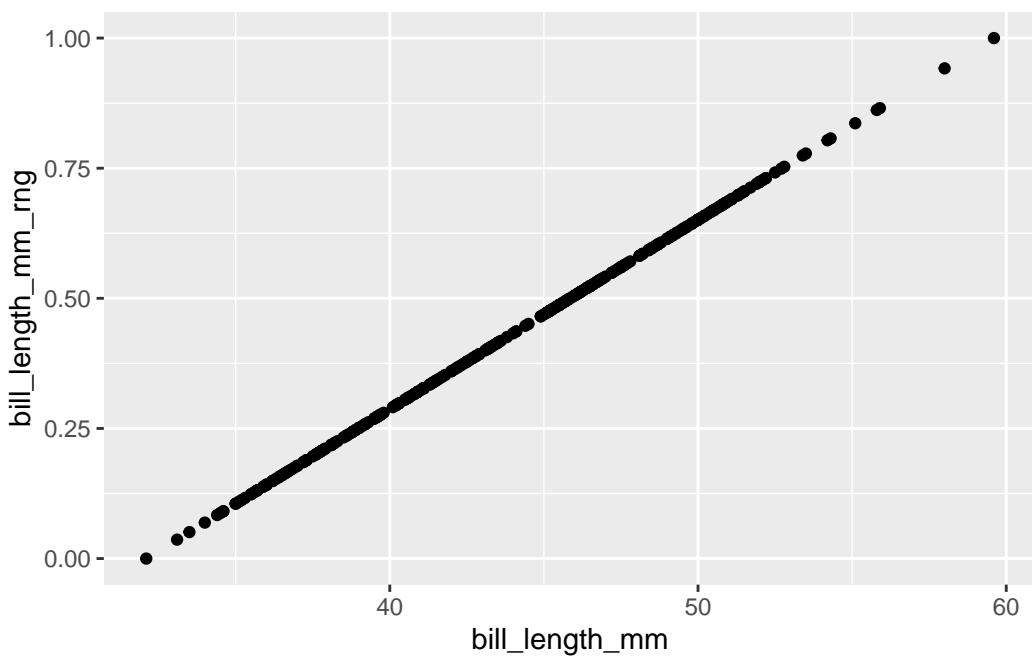
```
rawdata |>
  ggplot(aes(bill_length_mm,bill_length_mm_std)) +
  geom_point()
```



```
rawdata |>  
  ggplot(aes(bill_length_mm,bill_length_mm_qnorm))+  
  geom_point()
```



```
rawdata |>
  ggplot(aes(bill_length_mm,bill_length_mm_rng))+  
  geom_point()
```



## 22.4 Modelling

### 22.4.1 Definition of predictor variables (IV)

```
predvars_std <- ColSeeker(namepattern = '_std')
```

### 22.4.2 Data splitting

tidyverse approach:

```
set.seed(2026)  
traindata <- rawdata |>  
  select(ID,species,sex,predvars_std$names) |>  
  group_by(species, sex) |>  
  slice_sample(prop = 2/3) |>  
  ungroup() |>  
  select(-sex)
```

```

testdata <- filter(rawdata,
                     !ID %in% traindata$ID) |>
  select(ID, species, predvars_std$names)

```

tidymodels approach:

```

set.seed(2026)
data_split <- initial_split(
  rawdata,
  prop = 2/3,
  strata = species
)
traindata_tidy <- training(data_split)
testdata_tidy <- testing(data_split)

```

### 22.4.3 Model fitting

For the training data, a model is created and used to (re-)predict the species of the test data. The `knn3Train` function from the `caret` package is used to fit the model. The `k` parameter specifies the number of neighbors to consider. The `prob` attribute of the output is used to get the probabilities of the species.

```

train_out <-
  knn3Train(train = select(traindata, predvars_std$names),
            test = select(testdata, predvars_std$names),
            cl = traindata$species, k = 5)
str(train_out)

```

```

chr [1:115] "Adelie" "Adelie" "Chinstrap" "Adelie" "Adelie" "Adelie" ...
- attr(*, "prob")= num [1:115, 1:3] 1 1 0 1 1 1 1 1 1 1 ...
..- attr(*, "dimnames")=List of 2
.. ..$ : NULL
.. ..$ : chr [1:3] "Adelie" "Chinstrap" "Gentoo"

```

```
head(train_out)
```

```
[1] "Adelie"      "Adelie"      "Chinstrap"   "Adelie"      "Adelie"      "Adelie"
```

```

train_res <-
  attr(x = train_out, which = 'prob') |>
  as_tibble() |>

```

```

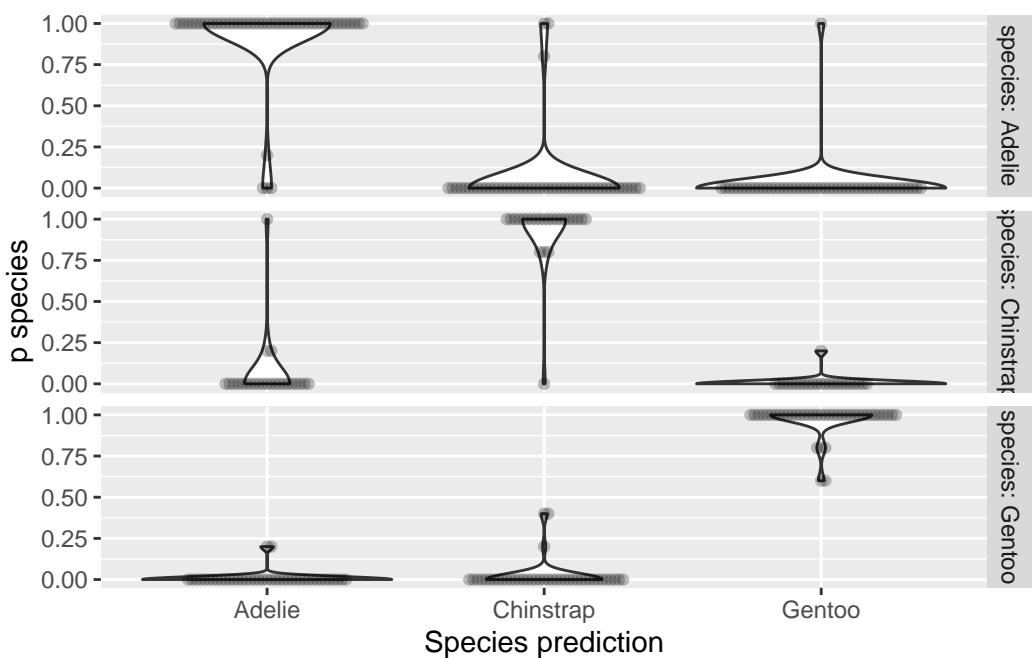
mutate(predicted=factor(train_out)) |>
bind_cols(testdata)

train_res |>
pivot_longer(c(Adelie,Chinstrap,Gentoo),
             values_to = 'p species',
             names_to = 'Species prediction') |>
ggplot(aes(`Species prediction`,`p species`))+  

geom_violin()+
geom_beeswarm(cex = .5, alpha=.25)+  

facet_grid(rows = vars(species), labeller='label_both')

```



```

model_formula <-
paste("species ~", paste(predvars_std$names, collapse = "+")) |>
as.formula()

knn_spec <- nearest_neighbor(neighbors = 5) |>
set_engine("kknn") |>
set_mode("classification")

knn_workflow <- workflow() |>
add_model(knn_spec)|>
add_formula(model_formula)

```

```
# Fit the model to the training data
knn_fit <- knn_workflow |>
  fit(data = traindata)
```

Attache Paket: 'kknn'

Das folgende Objekt ist maskiert 'package:caret':

contr.dummy

```
# Predict classes
predictions <- knn_fit |>
  predict(new_data = testdata)
prob_preds <- knn_fit |>
  predict(new_data = testdata, type = "prob")
# Combine with original data to check accuracy
results <- testdata |>
  select(species) |>
  bind_cols(predictions,prob_preds)
```

#### 22.4.4 Model evaluation

```
CrossTable(train_res$predicted,  
          train_res$species, prop.chisq = FALSE, prop.t = FALSE,  
          format = 'SAS')
```

Cell Contents										
	N									
	N / Row Total									
	N / Col Total									
-----										
Total Observations in Table: 115										
train_res\$species										
train_res\$predicted	Adelie	Chinstrap	Gentoo	Row Total						
----- ----- ----- ----- ----- -----										
Adelie	46	1	0	47						
	0.979	0.021	0.000	0.409						
	0.920	0.042	0.000							
----- ----- ----- ----- ----- -----										
Chinstrap	3	23	0	26						
	0.115	0.885	0.000	0.226						
	0.060	0.958	0.000							
----- ----- ----- ----- ----- -----										
Gentoo	1	0	41	42						
	0.024	0.000	0.976	0.365						
	0.020	0.000	1.000							
----- ----- ----- ----- ----- -----										
Column Total	50	24	41	115						
	0.435	0.209	0.357							
----- ----- ----- ----- ----- -----										

#### 22.4.5 Alternative approach to modelling

```
knn_formula <-
  paste0('species~',
         paste(predvars_std$names, collapse = '+')) |>
  as.formula()
knn_out <-
  knn3(knn_formula, data=rawdata,k = 5)
pred_all <- predict(knn_out,newdata = rawdata) |>
  as_tibble() |>
  bind_cols(rawdata) |>
  mutate(predicted= case_when(
    Adelie>=Chinstrap & Adelie>=Gentoo ~ 'Adelie',
    Chinstrap>Adelie & Chinstrap>Gentoo ~ 'Chinstrap',
    Gentoo>=Adelie & Gentoo>=Chinstrap ~ 'Gentoo') |>
    factor()))

# Alternative approach to prediction
predictions <-
  predict(knn_out,newdata = rawdata) |>
  as_tibble() |>
  rowwise() |>
  mutate(predicted=case_when(
    Adelie==max(Adelie,Chinstrap,Gentoo) ~ 'Adelie',
    Chinstrap==max(Adelie,Chinstrap,Gentoo) ~ 'Chinstrap',
    Gentoo==max(Adelie,Chinstrap,Gentoo) ~ 'Gentoo') |>
    factor()) |>
ungroup()
```

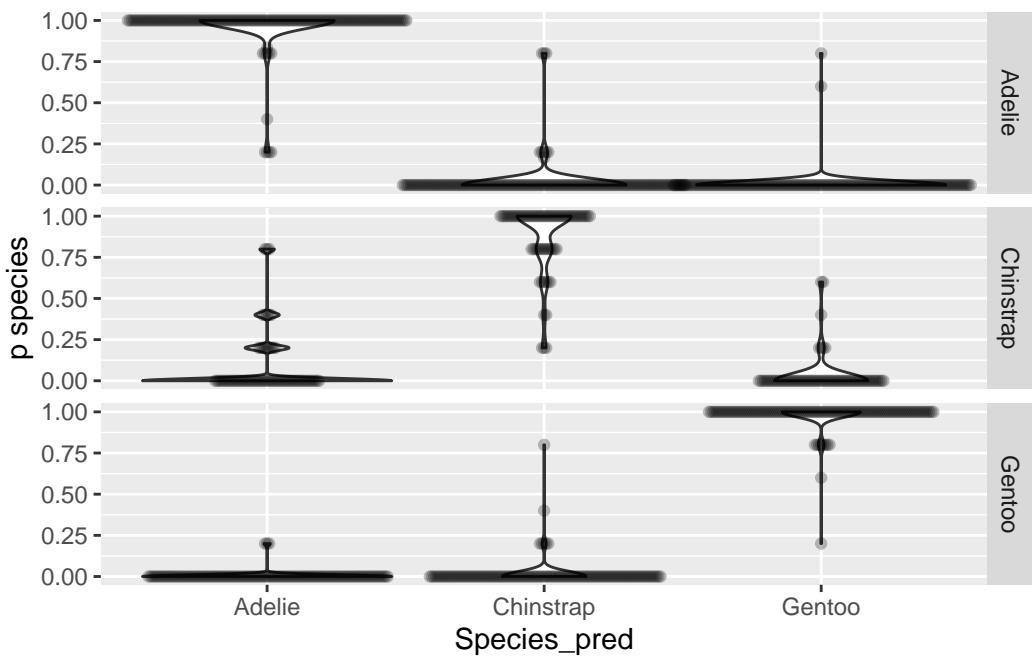
#### 22.4.6 Evaluation of the alternative approach

```
yardstick::accuracy(data = pred_all, truth=species, estimate=predicted)

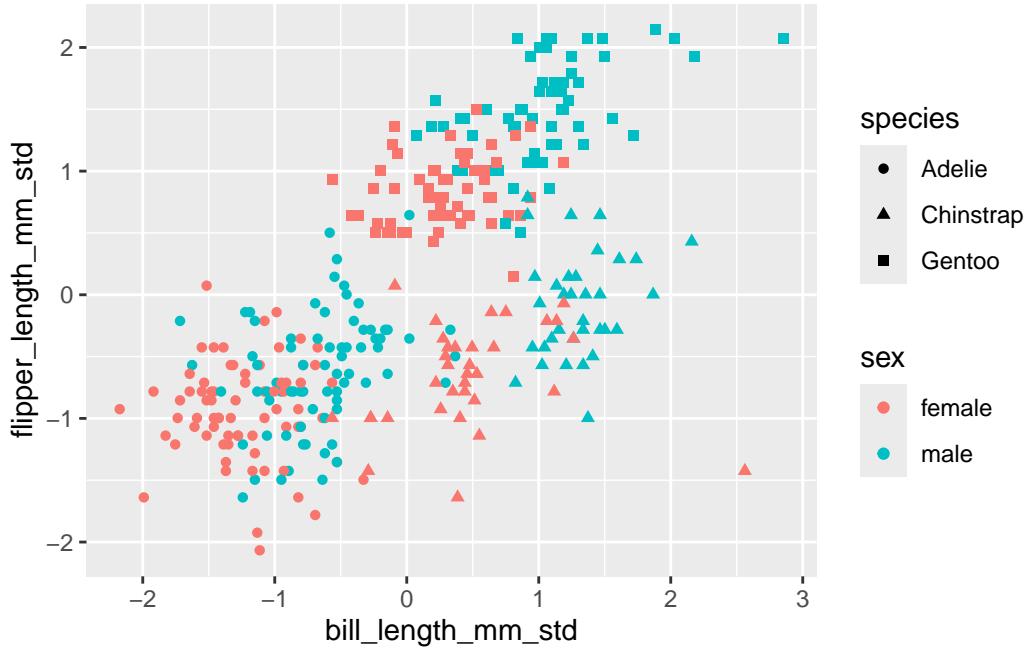
# A tibble: 1 x 3
  .metric   .estimator .estimate
  <chr>     <chr>        <dbl>
1 accuracy  multiclass  0.967

pred_all |>
  pivot_longer(c(Adelie,Chinstrap,Gentoo),
               values_to = 'p species',
               names_to = 'Species_pred') |>
```

```
ggplot(aes(Species_pred, `p species`))+  
  geom_violin() +  
  geom_beeswarm(cex = .25, alpha=.25) +  
  facet_grid(rows = vars(species))
```



```
ggplot(rawdata, aes(bill_length_mm_std, flipper_length_mm_std, color=sex, shape=species)) +  
  geom_point()
```



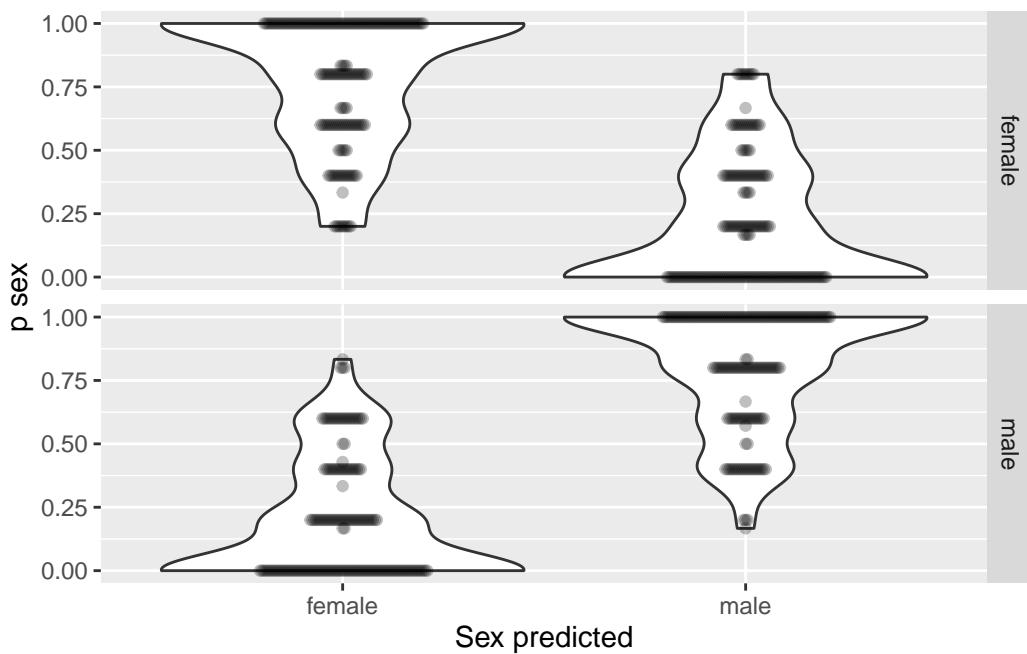
### 22.4.7 Adding predictor variables

Now we'll try to predict the sex based on body measures and species. Species will be automatically recoded using one-hot encoding.

```
knn_out <-
  knn3(sex ~ bill_length_mm_std + flipper_length_mm_std + species,
       data=rawdata,k = 5)

rawdata <-
  predict(knn_out,newdata = rawdata) |>
  as_tibble() |>
  cbind(rawdata)

rawdata |>
  pivot_longer(c(female,male),
               values_to = 'p sex',
               names_to = 'Sex predicted') |>
  ggplot(aes(`Sex predicted`,`p sex`))+
  geom_violin()+
  geom_beeswarm(cex = .25, alpha=.25)+
  facet_grid(rows = vars(sex))
```



```
rawdata <-
  mutate(rawdata,
    pred_sex=case_when( male>=.5 ~ "male",
                        .default = "female" ) |>
      as.factor())
yardstick::accuracy(data = rawdata, truth=sex,
                     estimate=pred_sex)
```

```
# A tibble: 1 x 3
  .metric   .estimator .estimate
  <chr>     <chr>        <dbl>
1 accuracy  binary       0.835
```

```
yardstick::sensitivity(data = rawdata, truth=sex,
                       estimate=pred_sex,
                       event_level="second")
```

```
# A tibble: 1 x 3
  .metric      .estimator .estimate
  <chr>        <chr>        <dbl>
1 sensitivity binary       0.851
```

```
yardstick::specificity(data = rawdata, truth=sex,
                      estimate=pred_sex,
                      event_level="second")
```

```
# A tibble: 1 x 3
  .metric      .estimator .estimate
  <chr>        <chr>          <dbl>
1 specificity  binary       0.818
```

```
yardstick::ppv(data = rawdata, truth=sex,
               estimate=pred_sex,
               event_level="second")
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 ppv     binary       0.827
```

```
CrossTable(rawdata$pred_sex,
           rawdata$sex, prop.chisq = F, prop.t = F,
           format = 'SPSS')
```

Cell Contents

	Count
	Row Percent
	Column Percent

Total Observations in Table: 333

	rawdata\$sex		Row Total
rawdata\$pred_sex	female	male	
female	135	25	160
	84.375%	15.625%	48.048%
	81.818%	14.881%	
male	30	143	173
	17.341%	82.659%	51.952%
	18.182%	85.119%	

Column Total	165	168	333	
	49.550%	50.450%		

```
confusionMatrix(rawdata$pred_sex, rawdata$sex,
                positive = "male")
```

#### Confusion Matrix and Statistics

		Reference
		Prediction
		female male
Prediction		female male
female	135	25
male	30	143
Accuracy : 0.8348		
95% CI : (0.7905, 0.8731)		
No Information Rate : 0.5045		
P-Value [Acc > NIR] : <2e-16		
Kappa : 0.6696		
McNemar's Test P-Value : 0.5896		
Sensitivity : 0.8512		
Specificity : 0.8182		
Pos Pred Value : 0.8266		
Neg Pred Value : 0.8438		
Prevalence : 0.5045		
Detection Rate : 0.4294		
Detection Prevalence : 0.5195		
Balanced Accuracy : 0.8347		
'Positive' Class : male		

# 23 Regression and classification trees / Random forrests

## 23.1 Regression trees

```
pacman::p_load(conflicted,
                 tidyverse,
                 wrappedtools,
                 palmerpenguins,
                 rpart, rpart.plot,
                 randomForest,
                 caret)

# conflict_scout()
conflicts_prefer(dplyr::slice,
                  dplyr::filter,
                  palmerpenguins::penguins)

[conflicted] Will prefer dplyr::slice over any other package.
[conflicted] Will prefer dplyr::filter over any other package.
[conflicted] Will prefer palmerpenguins::penguins over any other package.
```

```
rawdata <- penguins |>
  na.omit()
rawdata <- mutate(rawdata,
                  ID=paste('P', 1:nrow(rawdata))) |>
  select(ID, everything())
```

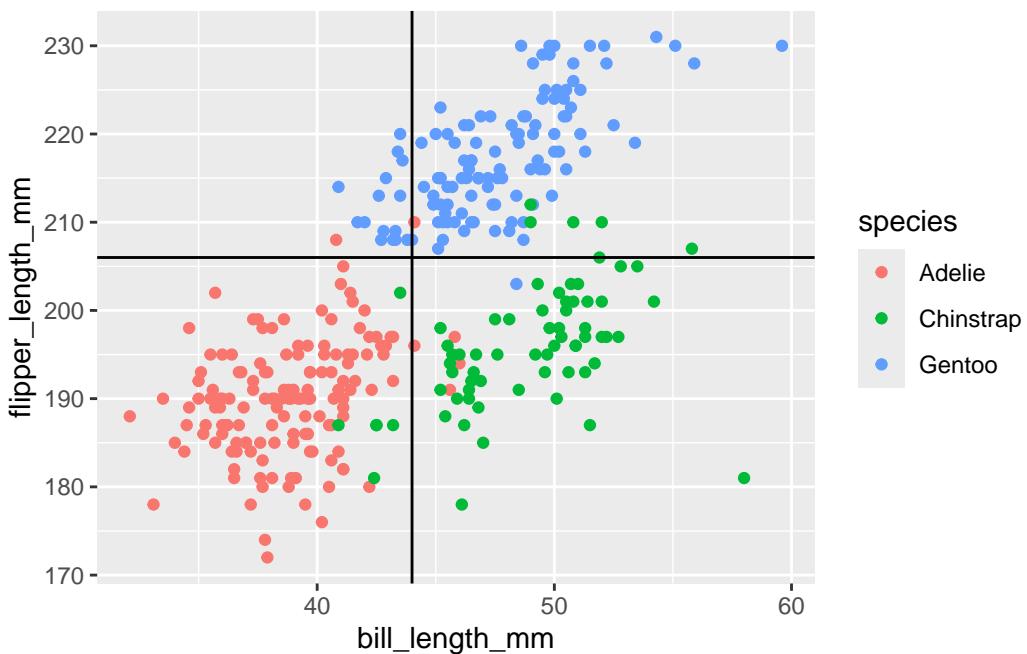
### 23.1.1 Graphical exploration

When looking at the 2 measures `bill_length_mm` and `flipper_length_mm`, we can see that the 3 species of penguins are separated to a large extend, with the Gentoos having longer flippers than Adelies and Chinstraps, and Adelies and Chinstraps separated by smaller / longer bills. We can define a rough classification rule based on that just by eyeballing the data.

```

predvars <- ColSeeker(namepattern = 'length')
rawplot <-
  ggplot(rawdata,
         aes(.data[[predvars$names[1]]],
              .data[[predvars$names[2]]], color=species))+
  geom_point()
rawplot+
  geom_hline(yintercept = 206)+
  geom_vline(xintercept = 44)

```



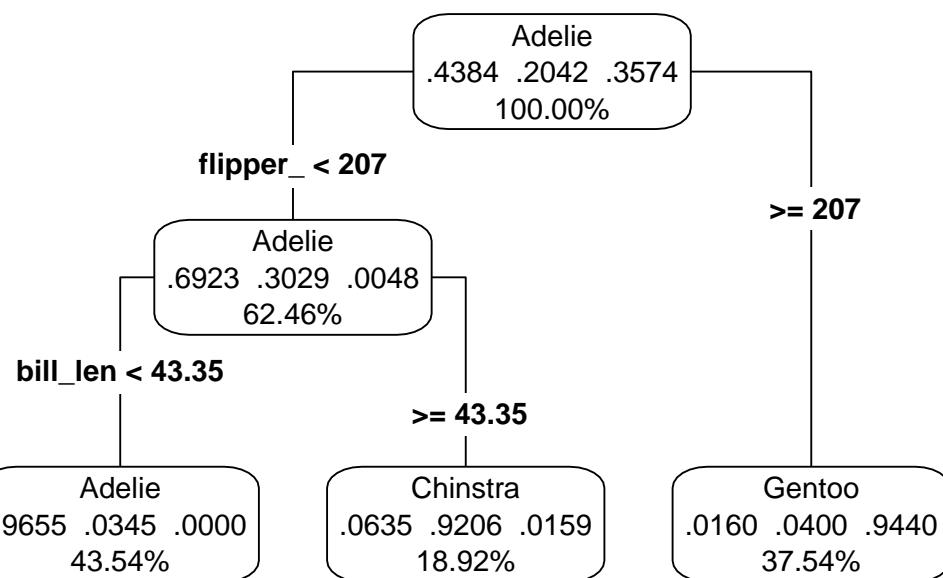
### 23.1.2 Modelling

A regression tree is following the same logic as the classification rule we defined above, but in a more systematic way. It is a recursive partitioning algorithm that splits the data into subsets that are as homogeneous as possible with respect to the target variable. The algorithm is greedy, meaning that it makes the best split at each step, without considering the impact of the split on future steps. This can lead to overfitting, so we need to be careful with the depth of the tree. As each split is testing each variable separately, there is no need for scaling. Accordingly, the results are easily comprehensible as they relate to the actual measurements. Beside the actual classification rules, the output contains information on variable importance as well.

```

rpart_formula <- paste('species',
                        paste(predvars$names,
                               collapse='+'),
                        sep='~') |>
  as.formula()
rpart_out <- rpart(formula = rpart_formula,
                     data = rawdata)
prp(rpart_out,
     type = 4,
     extra = 104,
     digits=4,
     fallen.leaves = TRUE)

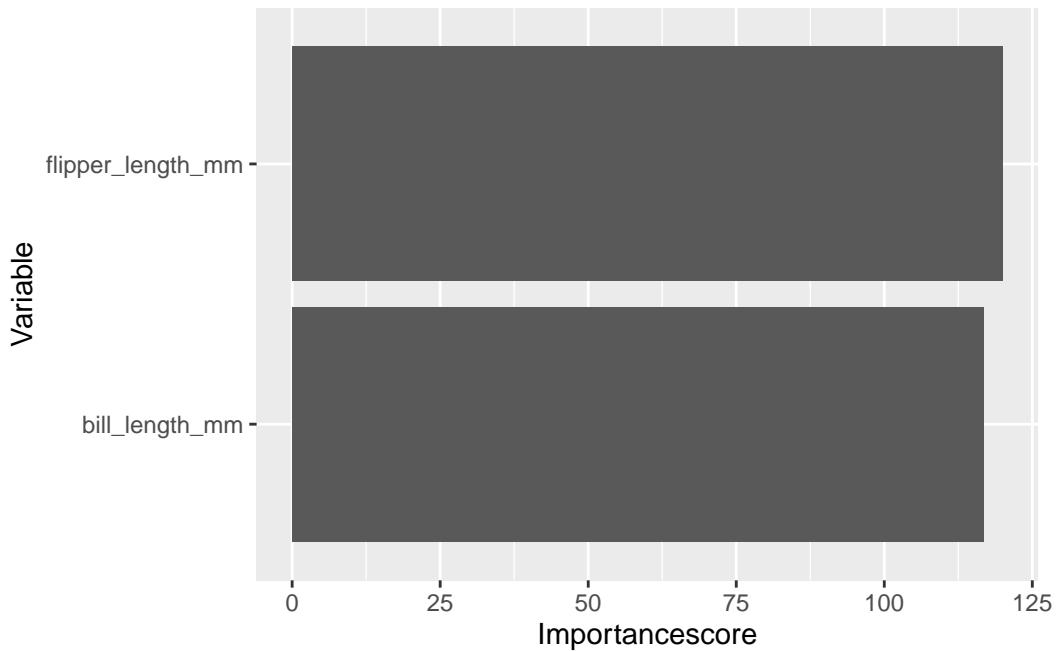
```



```

importance <- tibble(
  Variable=names(rpart_out$variable.importance),
  Importancescore=rpart_out$variable.importance)
ggplot(importance, aes(x=Variable,y=Importancescore))+ 
  geom_col()+
  coord_flip()

```



```
# rpart_out_c <- rpart(bill_depth_mm~bill_length_mm+flipper_length_mm+sex,
#                         data = rawdata,
#                         control = list(minsplit=2))
# prp(rpart_out_c, extra=100)
```

To make model more interesting, we add 2 more variables.

```
predvars <- ColSeeker(namepattern = c('_mm','_g',"sex"))
rpart_formula_4 <- paste('species',
                          paste(predvars$names,
                                collapse='+'),
                          sep='~') |>
  as.formula()

set.seed(2023)
traindata <- rawdata |>
  select(ID,species,sex,predvars$names) |>
  group_by(species, sex) |>
  slice_sample(prop = 2/3) |>
  ungroup() #|>
#  select(-sex)

testdata <- filter(rawdata,
                     !ID %in% traindata$ID) |>
  select(ID,species,sex,predvars$names)
```

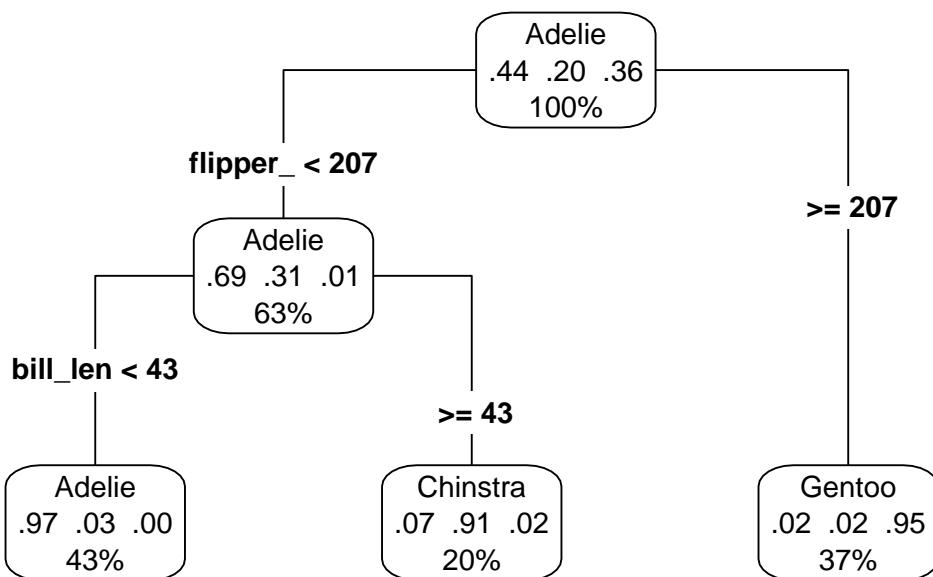
```
# tree for training sample
rpart_out_tr <- rpart(formula = rpart_formula_4,
                      data = traindata)
rpart_out$variable.importance
```

flipper_length_mm	bill_length_mm
120.0127	116.8639

```
rpart_out_tr$variable.importance
```

flipper_length_mm	bill_length_mm	bill_depth_mm	body_mass_g
79.16783	75.59169	56.58204	53.04566

```
prp(rpart_out_tr,
     type = 4,
     extra = 104,
     fallen.leaves = T)
```



### 23.1.3 Model evaluation

```
test_predicted <-
  bind_cols(testdata,
            as_tibble(
              predict(rpart_out_tr, testdata))) |>
  mutate(predicted =
    case_when(Adelie>Chinstrap &
                Adelie>Gentoo ~ 'Adelie',
              Chinstrap>Adelie &
                Chinstrap>Gentoo ~ 'Chinstrap',
              Gentoo>Adelie &
                Gentoo>Chinstrap ~ 'Gentoo') |>
    factor())  
  
gmodels::CrossTable(test_predicted$predicted,
                     test_predicted$species,
                     prop.chisq = F, prop.t = F,
                     format = 'SPSS')
```

Registered S3 method overwritten by 'gdata':

```
method      from
reorder.factor DescTools
```

Cell Contents					
	test_predicted\$species				
test_predicted\$predicted	Adelie	Chinstrap	Gentoo	Row Total	
Adelie	49	2	0	51	
	96.078%	3.922%	0.000%	44.348%	
	98.000%	8.333%	0.000%		
Chinstrap	1	19	0	20	
	5.000%	95.000%	0.000%	17.391%	
	2.000%	79.167%	0.000%		

	0	3	41	44	
Gentoo	0.000%	6.818%	93.182%	38.261%	
	0.000%	12.500%	100.000%		
Column Total	50	24	41	115	
	43.478%	20.870%	35.652%		

```
confusionMatrix(test_predicted$predicted,
                test_predicted$species)
```

### Confusion Matrix and Statistics

#### Reference

Prediction	Adelie	Chinstrap	Gentoo
Adelie	49	2	0
Chinstrap	1	19	0
Gentoo	0	3	41

### Overall Statistics

Accuracy : 0.9478

95% CI : (0.8899, 0.9806)

No Information Rate : 0.4348

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9178

Mcnemar's Test P-Value : NA

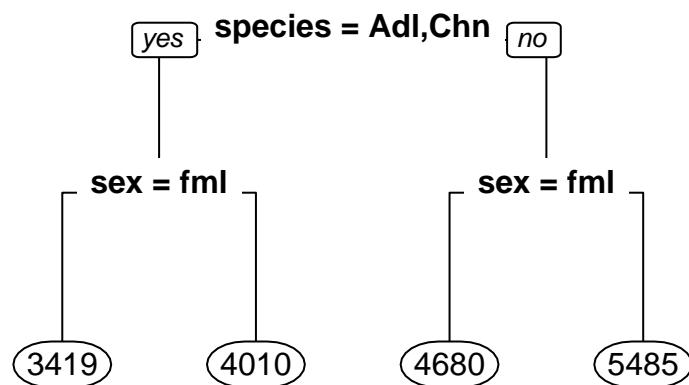
### Statistics by Class:

	Class: Adelie	Class: Chinstrap	Class: Gentoo
Sensitivity	0.9800	0.7917	1.0000
Specificity	0.9692	0.9890	0.9595
Pos Pred Value	0.9608	0.9500	0.9318
Neg Pred Value	0.9844	0.9474	1.0000
Prevalence	0.4348	0.2087	0.3565
Detection Rate	0.4261	0.1652	0.3565
Detection Prevalence	0.4435	0.1739	0.3826
Balanced Accuracy	0.9746	0.8903	0.9797

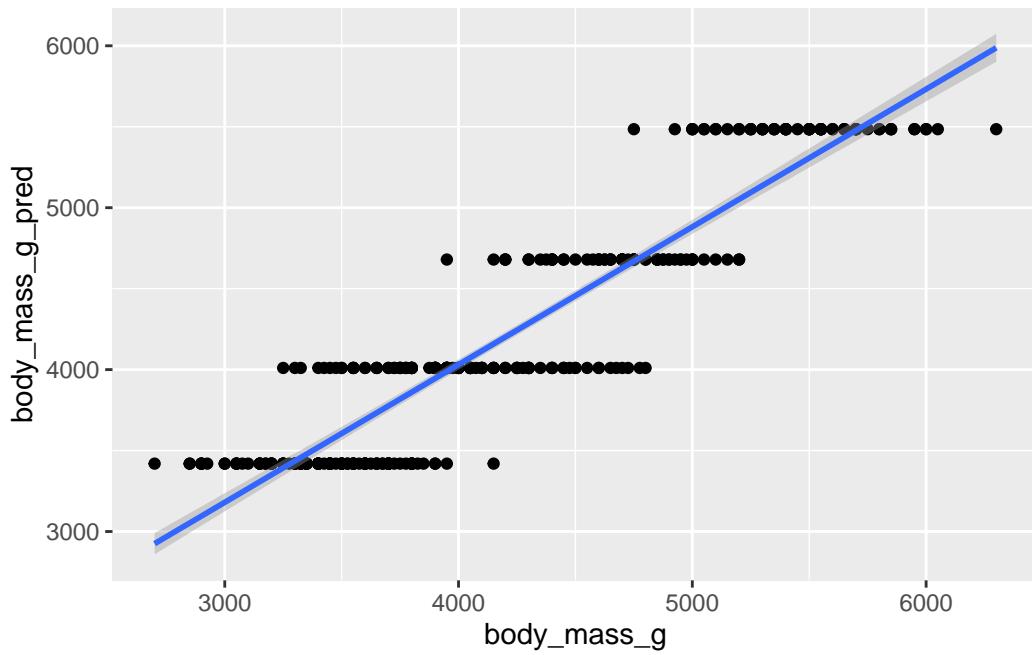
## 23.2 RT for continuous outcomes

```
rpart_out_cont <- rpart(body_mass_g~species+sex+
                           flipper_length_mm+bill_length_mm+
                           bill_depth_mm,
                           minsplit=3,
                           data=rawdata)

prp(rpart_out_cont,
     fallen.leaves = TRUE)
```



```
rawdata <-
  mutate(rawdata,
        body_mass_g_pred = predict(rpart_out_cont))
ggplot(rawdata,aes(body_mass_g, body_mass_g_pred))+  
  geom_point()+
  geom_smooth(method="lm")  
  
`geom_smooth()` using formula = 'y ~ x'
```



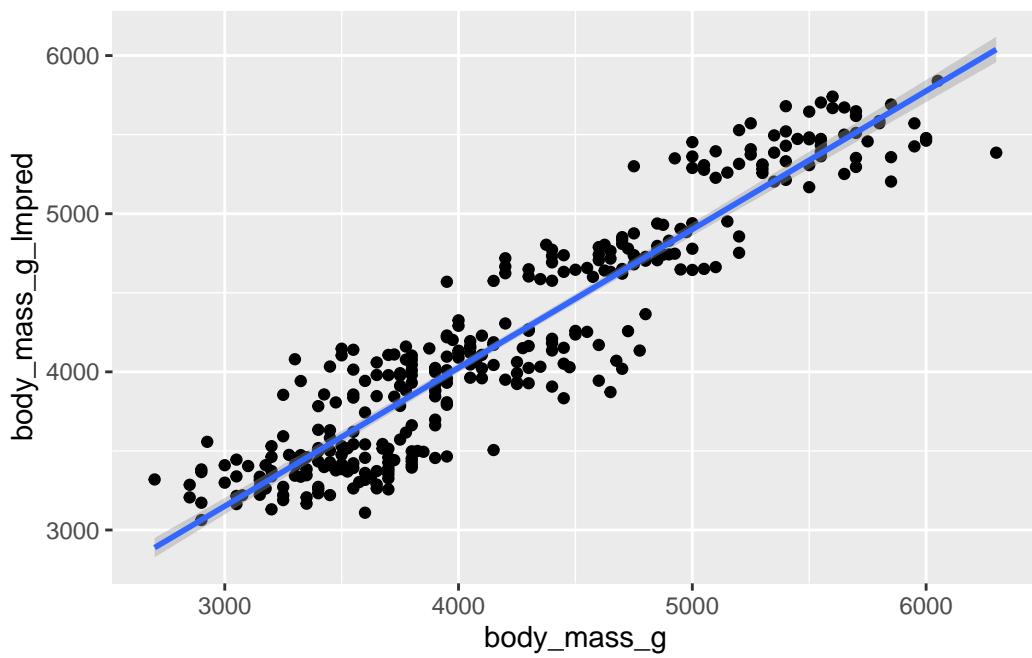
```

lm_out_cont <- lm(body_mass_g~species+sex+
                     flipper_length_mm+bill_length_mm+
                     bill_depth_mm,
                     data=rawdata)

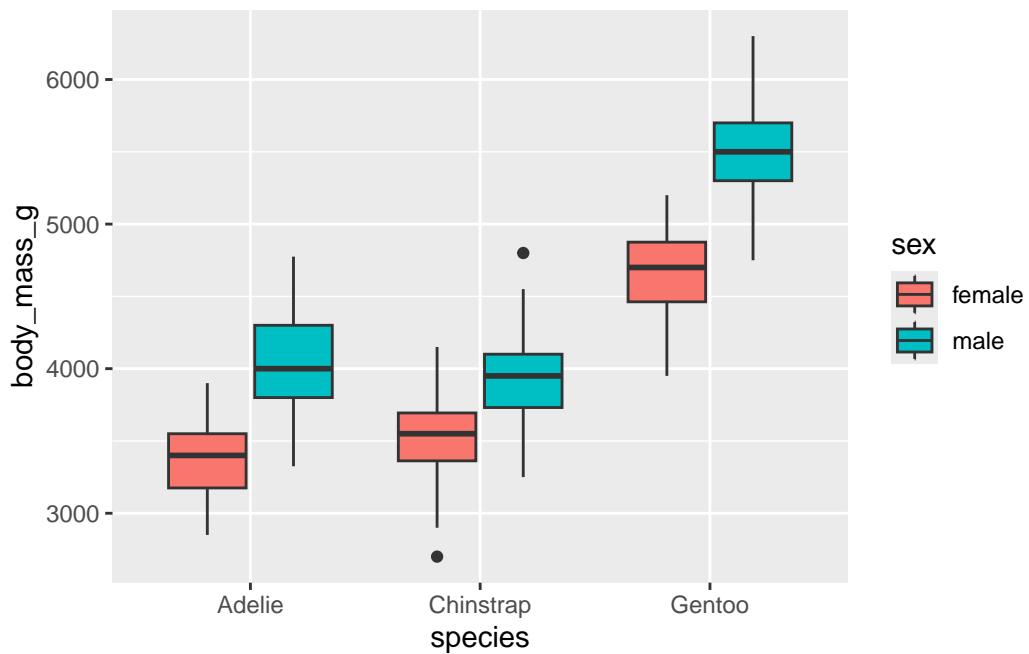
rawdata <-
  mutate(rawdata,
        body_mass_g_lmpred = predict(lm_out_cont))
ggplot(rawdata,aes(body_mass_g, body_mass_g_lmpred))+ 
  geom_point()+
  geom_smooth(method="lm")

`geom_smooth()` using formula = 'y ~ x'

```



```
ggplot(rawdata,aes(x=species,y=body_mass_g, fill=sex))+  
  geom_boxplot()
```



## 23.3 Random forest

Random forests are an ensemble method that builds multiple decision trees and averages their predictions. This reduces the risk of overfitting and increases the accuracy of the model. The random part comes from the fact that each tree is built on a random subset of the data and a random subset of the variables. The number of trees and the number of variables to consider at each split are hyper-parameters that need to be tuned. (more on tuning in chapter caret)

### 23.3.1 Modelling

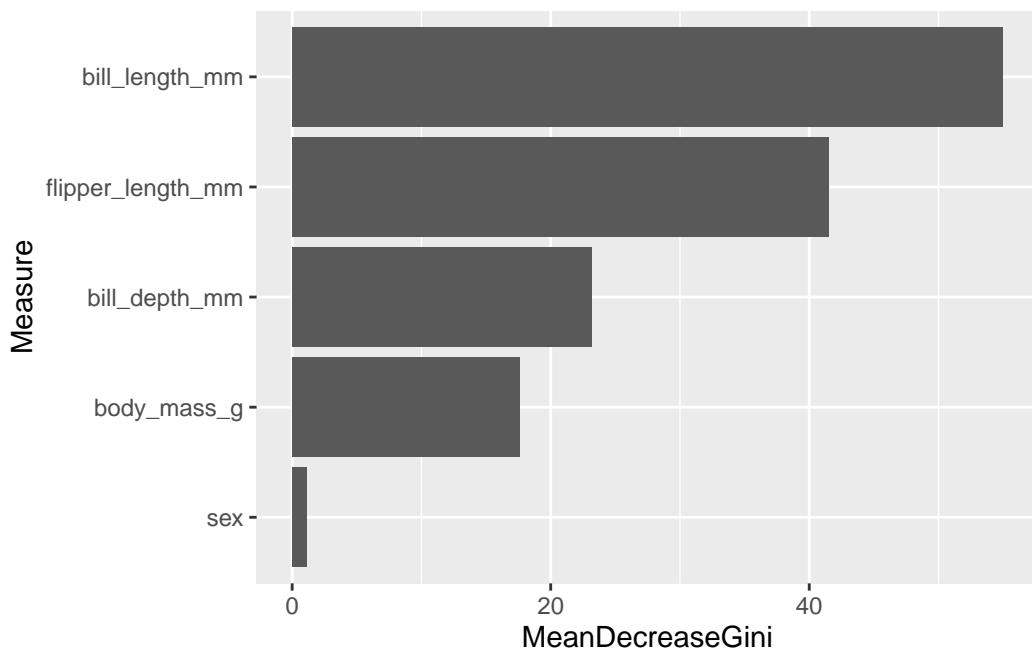
```
forrest_formula <-
  paste('species',
    paste(predvars$names, collapse='+'),
    sep='~') |>
  as.formula()

rf_out <- randomForest(forrest_formula,
                        data = traindata,
                        ntree=500,mtry=2)

importance(rf_out)
```

	MeanDecreaseGini
bill_length_mm	54.970253
bill_depth_mm	23.129604
flipper_length_mm	41.482374
body_mass_g	17.625764
sex	1.128116

```
importance(rf_out) |>
  as_tibble(rownames='Measure') |>
  arrange(#desc(
    MeanDecreaseGini) |> #) |>
  mutate(Measure=fct_inorder(Measure)) |>
  ggplot(aes(x=Measure,y=MeanDecreaseGini))+
  geom_col()+
  coord_flip()
```



### 23.3.2 Model evaluation

```
p1 <- predict(rf_out, traindata)
confusionMatrix(p1, traindata$species)
```

Confusion Matrix and Statistics

		Reference		
Prediction	Adelie	Chinstrap		Gentoo
		96	0	0
Chinstrap		0	44	0
Gentoo		0	0	78

Overall Statistics

```
Accuracy : 1
95% CI : (0.9832, 1)
No Information Rate : 0.4404
P-Value [Acc > NIR] : < 2.2e-16
```

Kappa : 1

Mcnemar's Test P-Value : NA

Statistics by Class:

	Class: Adelie	Class: Chinstrap	Class: Gentoo
Sensitivity	1.0000	1.0000	1.0000
Specificity	1.0000	1.0000	1.0000
Pos Pred Value	1.0000	1.0000	1.0000
Neg Pred Value	1.0000	1.0000	1.0000
Prevalence	0.4404	0.2018	0.3578
Detection Rate	0.4404	0.2018	0.3578
Detection Prevalence	0.4404	0.2018	0.3578
Balanced Accuracy	1.0000	1.0000	1.0000

```
p2 <- predict(rf_out, testdata)
confusionMatrix(p2, testdata$species)
```

Confusion Matrix and Statistics

		Reference		
		Adelie	Chinstrap	Gentoo
Prediction		Adelie	Chinstrap	Gentoo
Adelie	49	2	0	
Chinstrap	1	22	0	
Gentoo	0	0	41	

Overall Statistics

Accuracy : 0.9739  
95% CI : (0.9257, 0.9946)  
No Information Rate : 0.4348  
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9591

McNemar's Test P-Value : NA

Statistics by Class:

	Class: Adelie	Class: Chinstrap	Class: Gentoo
Sensitivity	0.9800	0.9167	1.0000
Specificity	0.9692	0.9890	1.0000
Pos Pred Value	0.9608	0.9565	1.0000
Neg Pred Value	0.9844	0.9783	1.0000
Prevalence	0.4348	0.2087	0.3565
Detection Rate	0.4261	0.1913	0.3565
Detection Prevalence	0.4435	0.2000	0.3565
Balanced Accuracy	0.9746	0.9528	1.0000

```
p2prob <- predict(rf_out, newdata = testdata, type = "prob")
```

## 24 Boosted regression trees

```
pacman::p_load(conflicted,
tidyverse,
ggbeeswarm,
wrappedtools, # just tools
palmerpenguins, # data
caret, # Classification and Regression Training
gmodels, # tools for model fitting
easystats,
yardstick, # model performance
xgboost) # XGBoost

conflicts_prefer(dplyr::slice,
dplyr::filter,
dplyr::lag,
palmerpenguins::penguins)
```

```
[conflicted] Will prefer dplyr::slice over any other package.
[conflicted] Will prefer dplyr::filter over any other package.
[conflicted] Will prefer dplyr::lag over any other package.
[conflicted] Will prefer palmerpenguins::penguins over any other package.
```

```
rawdata <- penguins |>
na.omit() |>
mutate(ID = paste('P', row_number()),
species_n = as.numeric(species)-1
) |>
select(ID, everything())
```

```
# one-hot-encoding sex + island
tmp <- dummyVars(~sex+island, rawdata,fullRank = F)
#predict(tmp,rawdata) |> view()
rawdata <- bind_cols(rawdata,
predict(tmp,rawdata))
cn()
```

```
[1] "ID"           "species"       "island"
```

```
[4] "bill_length_mm"      "bill_depth_mm"      "flipper_length_mm"  
[7] "body_mass_g"         "sex"                 "year"  
[10] "species_n"          "sex.female"        "sex.male"  
[13] "island.Biscoe"      "island.Dream"       "island.Torgersen"
```

```
predvars <- ColSeeker(rawdata,  
                      c('_._+', 'sex.+',  
                        "island.+", "year"))  
predvars$names
```

```
[1] "bill_length_mm"      "bill_depth_mm"      "flipper_length_mm"  
[4] "body_mass_g"         "year"                "sex.female"  
[7] "sex.male"            "island.Biscoe"       "island.Dream"  
[10] "island.Torgersen"
```

```
boostdata <- rawdata |>  
  select(predvars$names) |>  
  as.matrix()
```

```
xgb_out <- xgboost(  
  data = boostdata,  
  label = rawdata$species_n,  
  num_class=3,  
  nrounds = 100,  
  objective = "multi:softprob") # requests class probabilities
```

```
[1] train-mlogloss:0.720330  
[2] train-mlogloss:0.502322  
[3] train-mlogloss:0.361490  
[4] train-mlogloss:0.265603  
[5] train-mlogloss:0.198317  
[6] train-mlogloss:0.148850  
[7] train-mlogloss:0.113338  
[8] train-mlogloss:0.087211  
[9] train-mlogloss:0.067202  
[10]   train-mlogloss:0.052115  
[11]   train-mlogloss:0.041395  
[12]   train-mlogloss:0.033056  
[13]   train-mlogloss:0.027024  
[14]   train-mlogloss:0.022641  
[15]   train-mlogloss:0.018885  
[16]   train-mlogloss:0.016144
```

```
[17] train-mlogloss:0.013837
[18] train-mlogloss:0.012074
[19] train-mlogloss:0.010582
[20] train-mlogloss:0.009452
[21] train-mlogloss:0.008572
[22] train-mlogloss:0.007806
[23] train-mlogloss:0.007195
[24] train-mlogloss:0.006793
[25] train-mlogloss:0.006457
[26] train-mlogloss:0.006294
[27] train-mlogloss:0.006124
[28] train-mlogloss:0.005998
[29] train-mlogloss:0.005885
[30] train-mlogloss:0.005783
[31] train-mlogloss:0.005688
[32] train-mlogloss:0.005597
[33] train-mlogloss:0.005513
[34] train-mlogloss:0.005435
[35] train-mlogloss:0.005358
[36] train-mlogloss:0.005286
[37] train-mlogloss:0.005218
[38] train-mlogloss:0.005152
[39] train-mlogloss:0.005091
[40] train-mlogloss:0.005031
[41] train-mlogloss:0.004977
[42] train-mlogloss:0.004921
[43] train-mlogloss:0.004870
[44] train-mlogloss:0.004823
[45] train-mlogloss:0.004774
[46] train-mlogloss:0.004731
[47] train-mlogloss:0.004688
[48] train-mlogloss:0.004646
[49] train-mlogloss:0.004606
[50] train-mlogloss:0.004567
[51] train-mlogloss:0.004531
[52] train-mlogloss:0.004497
[53] train-mlogloss:0.004461
[54] train-mlogloss:0.004429
[55] train-mlogloss:0.004395
[56] train-mlogloss:0.004364
[57] train-mlogloss:0.004333
[58] train-mlogloss:0.004304
[59] train-mlogloss:0.004276
[60] train-mlogloss:0.004250
[61] train-mlogloss:0.004226
[62] train-mlogloss:0.004202
```

```
[63] train-mlogloss:0.004178
[64] train-mlogloss:0.004156
[65] train-mlogloss:0.004134
[66] train-mlogloss:0.004113
[67] train-mlogloss:0.004093
[68] train-mlogloss:0.004076
[69] train-mlogloss:0.004059
[70] train-mlogloss:0.004042
[71] train-mlogloss:0.004031
[72] train-mlogloss:0.004020
[73] train-mlogloss:0.004010
[74] train-mlogloss:0.004009
[75] train-mlogloss:0.004009
[76] train-mlogloss:0.004009
[77] train-mlogloss:0.004009
[78] train-mlogloss:0.004008
[79] train-mlogloss:0.004008
[80] train-mlogloss:0.004008
[81] train-mlogloss:0.004008
[82] train-mlogloss:0.004008
[83] train-mlogloss:0.004008
[84] train-mlogloss:0.004008
[85] train-mlogloss:0.004008
[86] train-mlogloss:0.004008
[87] train-mlogloss:0.004008
[88] train-mlogloss:0.004008
[89] train-mlogloss:0.004008
[90] train-mlogloss:0.004008
[91] train-mlogloss:0.004008
[92] train-mlogloss:0.004008
[93] train-mlogloss:0.004008
[94] train-mlogloss:0.004008
[95] train-mlogloss:0.004008
[96] train-mlogloss:0.004008
[97] train-mlogloss:0.004008
[98] train-mlogloss:0.004008
[99] train-mlogloss:0.004008
[100] train-mlogloss:0.004008
```

```
prediction <- predict(xgb_out,boostdata) |>
  as_tibble() |>
  mutate(
    ID=rep(rawdata$ID, each =3),
    species_pred=rep(levels(rawdata$species), # 1 row per species
```

```

            times=nrow(rawdata))) |>
pivot_wider(names_from=species_pred, # 1 col per species prob
            values_from=value) |>
mutate(predicted= # find highest prob
       case_when(Adelie>Chinstrap &
                  Adelie>Gentoo ~ 'Adelie',
                  Chinstrap>Adelie &
                  Chinstrap>Gentoo ~ 'Chinstrap',
                  Gentoo>Adelie &
                  Gentoo>Chinstrap ~ 'Gentoo') |>
factor())
slice_sample(prediction, n=20)

```

```

# A tibble: 20 x 5
  ID      Adelie  Chinstrap   Gentoo predicted
  <chr>    <dbl>     <dbl>     <dbl> <fct>
1 P 141  0.999  0.000333  0.000170 Adelie
2 P 117  0.998  0.00166   0.000672 Adelie
3 P 206  0.00107 0.00129   0.998   Gentoo
4 P 248  0.000557 0.00106   0.998   Gentoo
5 P 192  0.00107 0.00129   0.998   Gentoo
6 P 306  0.00185 0.994    0.00398  Chinstrap
7 P 45   0.999  0.000484  0.000211 Adelie
8 P 143  0.998  0.00134   0.000180 Adelie
9 P 103  0.997  0.00179   0.000848 Adelie
10 P 164  0.000557 0.00106   0.998   Gentoo
11 P 191  0.000984 0.000824  0.998   Gentoo
12 P 33   0.987  0.0124    0.000241 Adelie
13 P 87   0.995  0.00428   0.000723 Adelie
14 P 131  0.998  0.00173   0.000293 Adelie
15 P 184  0.00127 0.00106   0.998   Gentoo
16 P 82   0.998  0.00135   0.000180 Adelie
17 P 197  0.000557 0.00106   0.998   Gentoo
18 P 308  0.00109 0.999    0.000366 Chinstrap
19 P 47   0.999  0.000559  0.000211 Adelie
20 P 109  0.999  0.000304  0.000211 Adelie

```

```

train_res <- full_join(rawdata,prediction)

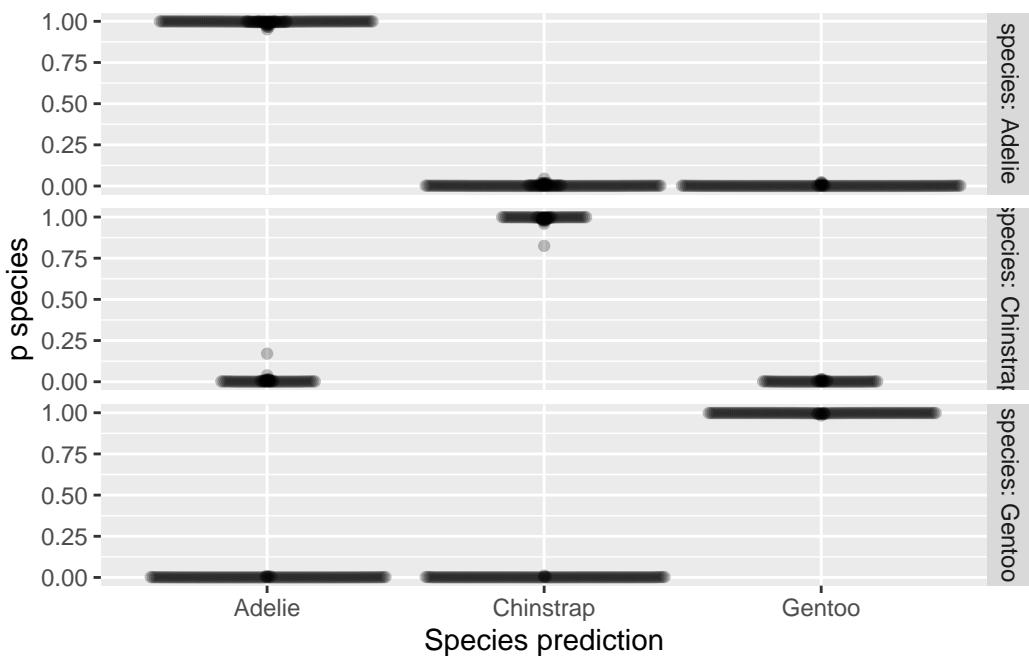
```

Joining with `by = join\_by(ID)`

```

train_res |>
  pivot_longer(c(Adelie,Chinstrap,Gentoo),
               values_to = 'p species',
               names_to = 'Species prediction') |>
  ggplot(aes(`Species prediction`,`p species`))+
```

# geom\_violin() +  
 geom\_beeswarm(cex = .25, alpha=.25) +  
 facet\_grid(rows = vars(species),  
 labeller='label\_both')



```

CrossTable(train_res$predicted,
           train_res$species,
           prop.chisq = F, prop.t = F,
           format = 'SPSS')
```

Cell Contents	
	Count
	Row Percent
	Column Percent

Total Observations in Table: 333

	train_res\$species				
train_res\$predicted	Adelie	Chinstrap	Gentoo	Row Total	
Adelie	146	0	0	146	
	100.000%	0.000%	0.000%	43.844%	
	100.000%	0.000%	0.000%		
Chinstrap	0	68	0	68	
	0.000%	100.000%	0.000%	20.420%	
	0.000%	100.000%	0.000%		
Gentoo	0	0	119	119	
	0.000%	0.000%	100.000%	35.736%	
	0.000%	0.000%	100.000%		
Column Total	146	68	119	333	
	43.844%	20.420%	35.736%		

```
yardstick::accuracy(data = train_res,
                     truth=species,
                     estimate=predicted)
```

```
# A tibble: 1 x 3
  .metric   .estimator .estimate
  <chr>     <chr>       <dbl>
1 accuracy  multiclass    1
```

```
yardstick::specificity(data = train_res,
                       truth=species,
                       estimate=predicted)
```

```
# A tibble: 1 x 3
  .metric   .estimator .estimate
  <chr>     <chr>       <dbl>
1 specificity macro      1
```

```
confusionMatrix(train_res$predicted,
                train_res$species)
```

## Confusion Matrix and Statistics

		Reference		
Prediction	Adelie	Chinstrap	Gentoo	
Adelie	146	0	0	
Chinstrap	0	68	0	
Gentoo	0	0	119	

## Overall Statistics

Accuracy : 1  
95% CI : (0.989, 1)  
No Information Rate : 0.4384  
P-Value [Acc > NIR] : < 2.2e-16

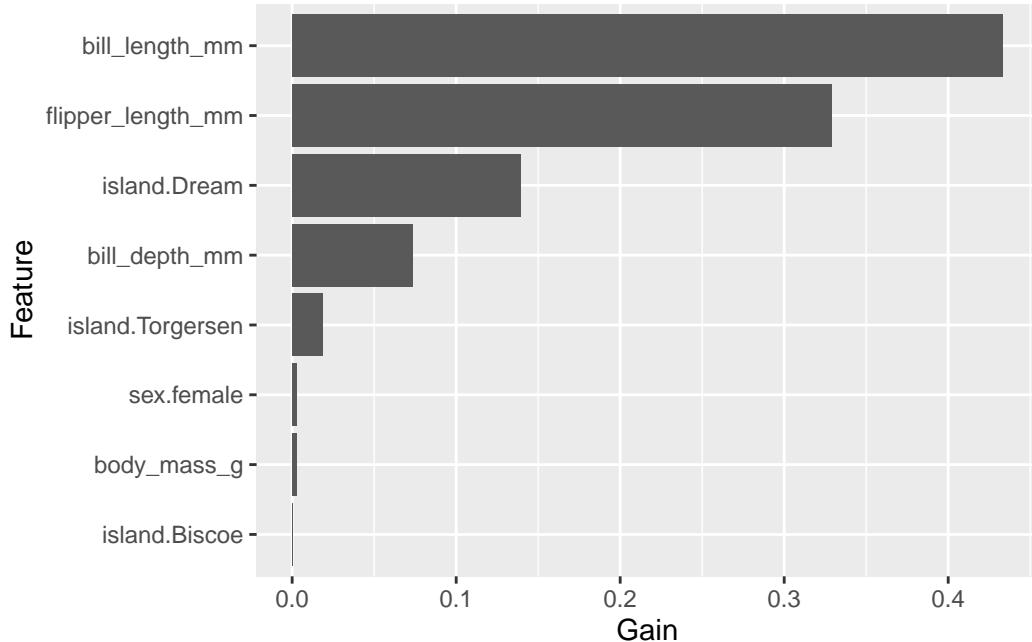
Kappa : 1

McNemar's Test P-Value : NA

## Statistics by Class:

	Class: Adelie	Class: Chinstrap	Class: Gentoo
Sensitivity	1.0000	1.0000	1.0000
Specificity	1.0000	1.0000	1.0000
Pos Pred Value	1.0000	1.0000	1.0000
Neg Pred Value	1.0000	1.0000	1.0000
Prevalence	0.4384	0.2042	0.3574
Detection Rate	0.4384	0.2042	0.3574
Detection Prevalence	0.4384	0.2042	0.3574
Balanced Accuracy	1.0000	1.0000	1.0000

```
xgb.importance(model = xgb_out) |>
  as_tibble() |>
  ggplot(aes(reorder(Feature, Gain), Gain))+
  geom_col()+
  coord_flip()+
  labs(x='Feature', y='Gain')
```



```

# splitting #####
set.seed(1958)
# splitted tibble
traindata <-
  rawdata |>
  select(ID, species, species_n,
         predvars$names) |>
  group_by(species_n) |>
  slice_sample(prop = 2/3) |>
  ungroup()
testdata <- filter(rawdata,
                     !ID %in% traindata$ID) |>
  select(ID, species, species_n,
         predvars$names)

# splitted matrix-objects
trainobject <-
  xgb.DMatrix(
    data=traindata |>
      select(-ID, -contains("species")) |>
      data.matrix(),
    label=traindata$species_n)
testobject <-
  xgb.DMatrix(
    data=testdata |>

```

```

    select(-ID,-contains("species")) |>
    data.matrix(),
  label=testdata$species_n)

watchlist = list(train=trainobject,
                 test=testobject)
# preliminary fitting to find nrounds
xgb.Train0 <- xgb.train(
  data = trainobject,
  params = list(max.depth = 5,
                num_class=3,
                objective = "multi:softprob"), # requests class probabilities
  watchlist = watchlist, #objects used in eval
  nrounds = 10^2) # set high, best will be checked afterwards

```

```

[1] train-mlogloss:0.725380 test-mlogloss:0.744892
[2] train-mlogloss:0.510681 test-mlogloss:0.541316
[3] train-mlogloss:0.370568 test-mlogloss:0.408343
[4] train-mlogloss:0.275208 test-mlogloss:0.319673
[5] train-mlogloss:0.207661 test-mlogloss:0.257748
[6] train-mlogloss:0.157250 test-mlogloss:0.212185
[7] train-mlogloss:0.120468 test-mlogloss:0.179662
[8] train-mlogloss:0.092550 test-mlogloss:0.148102
[9] train-mlogloss:0.072865 test-mlogloss:0.132556
[10]   train-mlogloss:0.057962 test-mlogloss:0.116226
[11]   train-mlogloss:0.046961 test-mlogloss:0.103999
[12]   train-mlogloss:0.038096 test-mlogloss:0.096813
[13]   train-mlogloss:0.031125 test-mlogloss:0.087630
[14]   train-mlogloss:0.025652 test-mlogloss:0.080146
[15]   train-mlogloss:0.021455 test-mlogloss:0.074210
[16]   train-mlogloss:0.018418 test-mlogloss:0.070350
[17]   train-mlogloss:0.016011 test-mlogloss:0.064644
[18]   train-mlogloss:0.014175 test-mlogloss:0.062493
[19]   train-mlogloss:0.012676 test-mlogloss:0.058542
[20]   train-mlogloss:0.011420 test-mlogloss:0.057008
[21]   train-mlogloss:0.010583 test-mlogloss:0.054179
[22]   train-mlogloss:0.010098 test-mlogloss:0.053776
[23]   train-mlogloss:0.009706 test-mlogloss:0.053982
[24]   train-mlogloss:0.009459 test-mlogloss:0.053720
[25]   train-mlogloss:0.009237 test-mlogloss:0.053842
[26]   train-mlogloss:0.009037 test-mlogloss:0.053649
[27]   train-mlogloss:0.008858 test-mlogloss:0.053790
[28]   train-mlogloss:0.008691 test-mlogloss:0.052490
[29]   train-mlogloss:0.008527 test-mlogloss:0.052625

```

```
[30] train-mlogloss:0.008378 test-mlogloss:0.051425
[31] train-mlogloss:0.008236 test-mlogloss:0.051586
[32] train-mlogloss:0.008103 test-mlogloss:0.050473
[33] train-mlogloss:0.007974 test-mlogloss:0.050379
[34] train-mlogloss:0.007860 test-mlogloss:0.050487
[35] train-mlogloss:0.007741 test-mlogloss:0.049457
[36] train-mlogloss:0.007636 test-mlogloss:0.049387
[37] train-mlogloss:0.007540 test-mlogloss:0.048508
[38] train-mlogloss:0.007438 test-mlogloss:0.048618
[39] train-mlogloss:0.007350 test-mlogloss:0.047760
[40] train-mlogloss:0.007260 test-mlogloss:0.047933
[41] train-mlogloss:0.007178 test-mlogloss:0.047124
[42] train-mlogloss:0.007098 test-mlogloss:0.047300
[43] train-mlogloss:0.007023 test-mlogloss:0.046533
[44] train-mlogloss:0.006951 test-mlogloss:0.046713
[45] train-mlogloss:0.006881 test-mlogloss:0.045985
[46] train-mlogloss:0.006817 test-mlogloss:0.046166
[47] train-mlogloss:0.006753 test-mlogloss:0.045506
[48] train-mlogloss:0.006692 test-mlogloss:0.045545
[49] train-mlogloss:0.006638 test-mlogloss:0.045712
[50] train-mlogloss:0.006580 test-mlogloss:0.045070
[51] train-mlogloss:0.006532 test-mlogloss:0.044508
[52] train-mlogloss:0.006487 test-mlogloss:0.044108
[53] train-mlogloss:0.006469 test-mlogloss:0.044056
[54] train-mlogloss:0.006452 test-mlogloss:0.043878
[55] train-mlogloss:0.006435 test-mlogloss:0.043834
[56] train-mlogloss:0.006420 test-mlogloss:0.043665
[57] train-mlogloss:0.006404 test-mlogloss:0.043628
[58] train-mlogloss:0.006389 test-mlogloss:0.043480
[59] train-mlogloss:0.006375 test-mlogloss:0.043448
[60] train-mlogloss:0.006362 test-mlogloss:0.043316
[61] train-mlogloss:0.006349 test-mlogloss:0.043290
[62] train-mlogloss:0.006337 test-mlogloss:0.043163
[63] train-mlogloss:0.006324 test-mlogloss:0.043141
[64] train-mlogloss:0.006314 test-mlogloss:0.043131
[65] train-mlogloss:0.006302 test-mlogloss:0.043013
[66] train-mlogloss:0.006292 test-mlogloss:0.043003
[67] train-mlogloss:0.006281 test-mlogloss:0.042887
[68] train-mlogloss:0.006271 test-mlogloss:0.042878
[69] train-mlogloss:0.006261 test-mlogloss:0.042766
[70] train-mlogloss:0.006252 test-mlogloss:0.042757
[71] train-mlogloss:0.006244 test-mlogloss:0.042756
[72] train-mlogloss:0.006237 test-mlogloss:0.042762
[73] train-mlogloss:0.006231 test-mlogloss:0.042774
[74] train-mlogloss:0.006226 test-mlogloss:0.042790
[75] train-mlogloss:0.006221 test-mlogloss:0.042810
```

```
[76] train-mlogloss:0.006221 test-mlogloss:0.042831
[77] train-mlogloss:0.006221 test-mlogloss:0.042849
[78] train-mlogloss:0.006221 test-mlogloss:0.042865
[79] train-mlogloss:0.006221 test-mlogloss:0.042878
[80] train-mlogloss:0.006221 test-mlogloss:0.042890
[81] train-mlogloss:0.006221 test-mlogloss:0.042901
[82] train-mlogloss:0.006221 test-mlogloss:0.042910
[83] train-mlogloss:0.006221 test-mlogloss:0.042917
[84] train-mlogloss:0.006221 test-mlogloss:0.042924
[85] train-mlogloss:0.006221 test-mlogloss:0.042930
[86] train-mlogloss:0.006217 test-mlogloss:0.042936
[87] train-mlogloss:0.006217 test-mlogloss:0.042945
[88] train-mlogloss:0.006217 test-mlogloss:0.042952
[89] train-mlogloss:0.006217 test-mlogloss:0.042959
[90] train-mlogloss:0.006217 test-mlogloss:0.042965
[91] train-mlogloss:0.006217 test-mlogloss:0.042969
[92] train-mlogloss:0.006217 test-mlogloss:0.042974
[93] train-mlogloss:0.006217 test-mlogloss:0.042978
[94] train-mlogloss:0.006217 test-mlogloss:0.042981
[95] train-mlogloss:0.006217 test-mlogloss:0.042984
[96] train-mlogloss:0.006217 test-mlogloss:0.042986
[97] train-mlogloss:0.006217 test-mlogloss:0.042989
[98] train-mlogloss:0.006217 test-mlogloss:0.042990
[99] train-mlogloss:0.006217 test-mlogloss:0.042992
[100] train-mlogloss:0.006217 test-mlogloss:0.042994
```

```
bestround <-
  xgb.Train0$evaluation_log |>
  as_tibble() |>
  filter(test_mlogloss==min(test_mlogloss)) |>
  pull(iter)
bestround
```

```
[1] 71
```

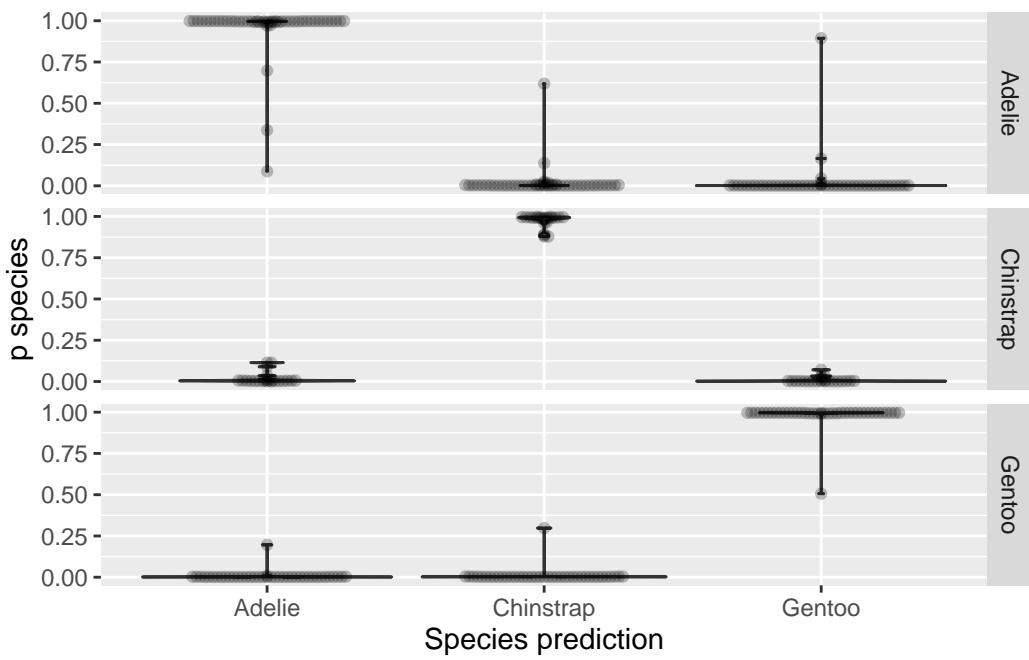
```
# v2 with threshold for improvement
bestround <-
  xgb.Train0$evaluation_log |>
  as_tibble() |>
  filter(test_mlogloss-lead(test_mlogloss)>.001) |>
  pull(iter)|>
  max()
bestround
```

[1] 34

```
# modelling
xgb.Train <- xgb.train(
  data = trainobject,
  params = list(objective = "multi:softprob",
                 max.depth = 5,
                 num_class=3),
  # watchlist=watchlist,
  nrounds = bestround)
prediction <- predict(xgb.Train,testobject) |>
  as_tibble() |>
  mutate(
    ID=rep(testdata$ID, each = 3),
    species_pred=rep(levels(rawdata$species),
                      times=nrow(testdata))) |>
  pivot_wider(names_from=species_pred,
              values_from=value) |>
  mutate(predicted=
    case_when(
      Adelie>Chinstrap &
        Adelie>Gentoo ~ 'Adelie',
      Chinstrap>Adelie &
        Chinstrap>Gentoo ~ 'Chinstrap',
      Gentoo>Adelie &
        Gentoo>Chinstrap ~ 'Gentoo') |>
    factor()))
train_res <- full_join(testdata,prediction)
```

Joining with `by = join\_by(ID)`

```
train_res |>
  pivot_longer(c(Adelie,Chinstrap,Gentoo),
               values_to = 'p species',
               names_to = 'Species prediction') |>
  ggplot(aes(`Species prediction`,`p species`))+  
  geom_violin()  
  geom_beeswarm(cex = .5, alpha=.25)+  
  facet_grid(rows = vars(species))
```



```
CrossTable(train_res$predicted,
          train_res$species,
          prop.chisq = F, prop.t = F,
          format = 'SPSS')
```

Cell Contents					
	train_res\$species				
train_res\$predicted	Adelie	Chinstrap	Gentoo	Row Total	
Adelie	47	0	0	47	
	100.000%	0.000%	0.000%	41.964%	
	95.918%	0.000%	0.000%		
Chinstrap	1	23	0	24	
	4.167%	95.833%	0.000%	21.429%	
	2.041%	100.000%	0.000%		

	1	0	40	41	
Gentoo	1	0	40	41	
	2.439%	0.000%	97.561%	36.607%	
	2.041%	0.000%	100.000%		
Column Total	49	23	40	112	
	43.750%	20.536%	35.714%		

```
yardstick::accuracy(data = train_res,
                     truth=species,
                     estimate=predicted)
```

```
# A tibble: 1 x 3
  .metric   .estimator .estimate
  <chr>     <chr>        <dbl>
1 accuracy  multiclass  0.982
```

```
confusionMatrix(train_res$predicted,
                 train_res$species)
```

#### Confusion Matrix and Statistics

		Reference		
Prediction	Adelie	Chinstrap	Gentoo	
Adelie	47	0	0	
Chinstrap	1	23	0	
Gentoo	1	0	40	

#### Overall Statistics

```
Accuracy : 0.9821
95% CI  : (0.937, 0.9978)
No Information Rate : 0.4375
P-Value [Acc > NIR] : < 2.2e-16
```

```
Kappa : 0.9722
```

```
McNemar's Test P-Value : NA
```

#### Statistics by Class:

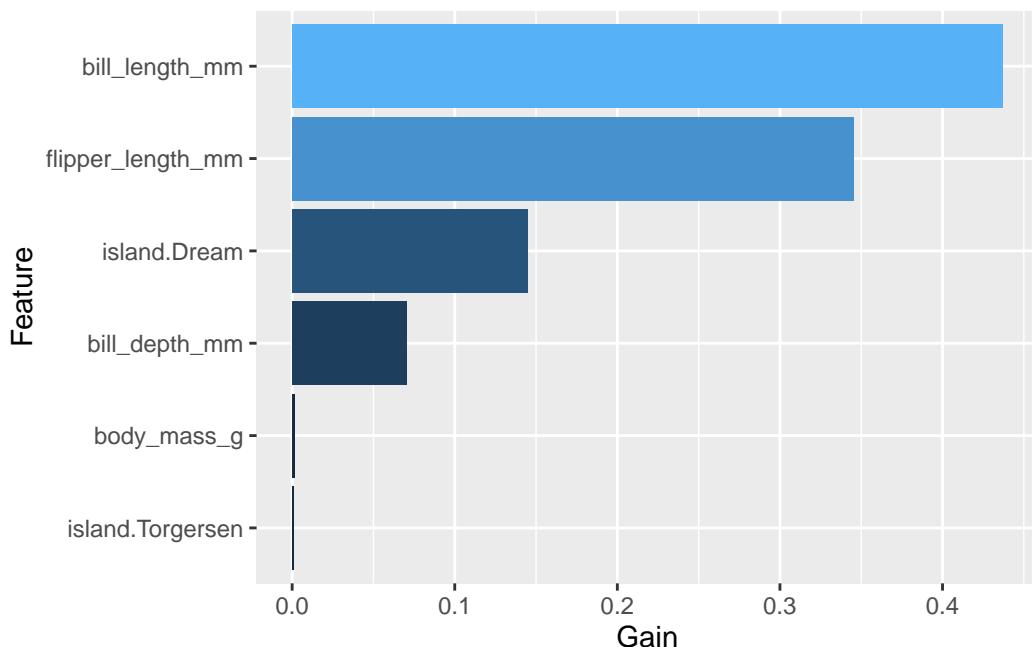
	Class: Adelie	Class: Chinstrap	Class: Gentoo
Sensitivity	0.9592	1.0000	1.0000
Specificity	1.0000	0.9888	0.9861
Pos Pred Value	1.0000	0.9583	0.9756
Neg Pred Value	0.9692	1.0000	1.0000
Prevalence	0.4375	0.2054	0.3571
Detection Rate	0.4196	0.2054	0.3571
Detection Prevalence	0.4196	0.2143	0.3661
Balanced Accuracy	0.9796	0.9944	0.9931

```

importance <-
  xgb.importance(model = xgb.Train) |>
  as_tibble() |>
  arrange(Gain) |>
  mutate(Feature=fct_inorder(Feature))

importance |>
  ggplot(aes(Feature, Gain)) +
  geom_col(aes(fill=Gain)) +
  coord_flip() +
  guides(fill="none")

```



```

plotfeatures <-
  slice_max(importance, Gain, n=2) |>
  pull(Feature) |>
  as.character()
# tail(importance$Feature, 2) |>
#  rev() |>
#  as.character()

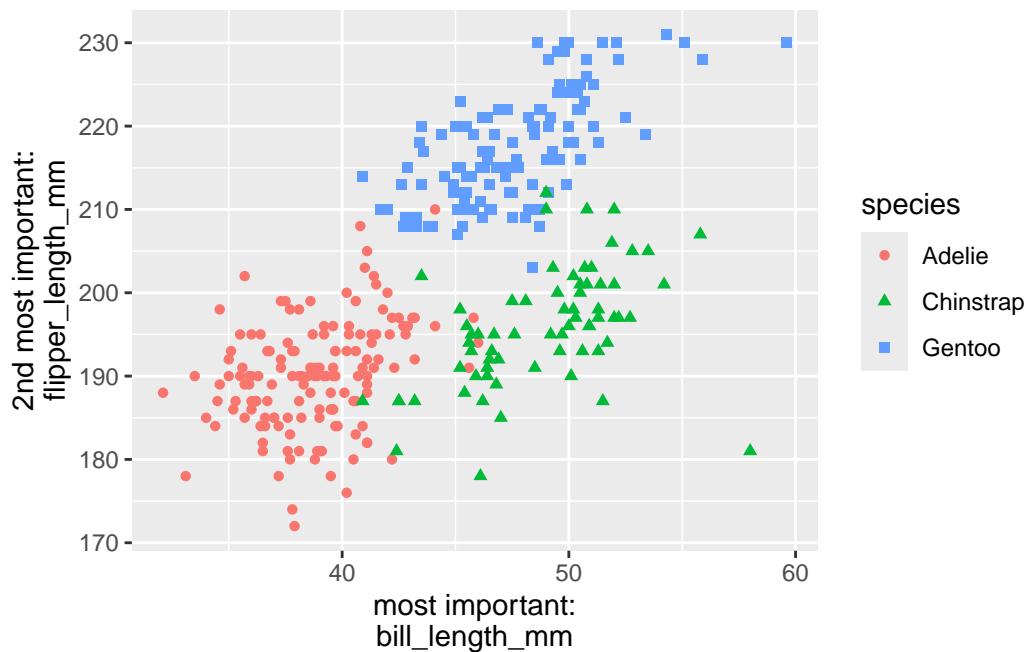
plotfeatures2 <-
  xgb.importance(model = xgb.Train)[[1]][1:2]

impo <- xgb.importance(model = xgb.Train)
plotfeatures_first <-
  impo$Feature[1]
plotfeatures_2nd <-
  impo$Feature[2]

ggplot(rawdata, aes(.data[[plotfeatures[1]]],
                    .data[[plotfeatures[2]]]),
       color=species, shape=species))+  

  geom_point()+
  xlab(paste("most important:",
             plotfeatures[1], sep = "\n"))+
  ylab(paste("2nd most important:",
             plotfeatures[2], sep = "\n"))

```



```
# importance per species
xgb.importance(model = xgb.Train,
               trees = seq(from=0, by=3,
                           length.out=bestround))
```

	Feature	Gain	Cover	Frequency
	<char>	<num>	<num>	<num>
1:	bill_length_mm	0.8704167214	0.50125029	0.52475248
2:	bill_depth_mm	0.1261444559	0.44086057	0.42574257
3:	island.Torgersen	0.0029074875	0.00817197	0.00990099
4:	flipper_length_mm	0.0005313351	0.04971718	0.03960396

```
xgb.importance(model = xgb.Train,
               trees = seq(from=1, by=3,
                           length.out=bestround))
```

	Feature	Gain	Cover	Frequency
	<char>	<num>	<num>	<num>
1:	island.Dream	0.535836145	0.53574611	0.2597403
2:	bill_length_mm	0.432707195	0.36586600	0.4415584
3:	bill_depth_mm	0.028554655	0.06305407	0.1298701
4:	body_mass_g	0.002902006	0.03533381	0.1688312

```
xgb.importance(model = xgb.Train,
               trees = seq(from=2, by=3,
                           length.out=bestround))
```

	Feature	Gain	Cover	Frequency
	<char>	<num>	<num>	<num>
1:	flipper_length_mm	9.523695e-01	0.675613282	0.48648649
2:	bill_depth_mm	4.524291e-02	0.286326266	0.35135135
3:	body_mass_g	2.292152e-03	0.035757548	0.13513514
4:	bill_length_mm	9.548313e-05	0.002302905	0.02702703

## 25 Principal Components Analysis

```
if(!requireNamespace("BiocManager", quietly = TRUE)) {
  install.packages("BiocManager")
}
if(!requireNamespace("PCAtools", quietly = TRUE)) {
  BiocManager::install("PCAtools")
}
# BiocManager::install("PCAtools")
pacman::p_load(conflicted,
  tidyverse,
  wrappedtools,
  palmerpenguins,
  ggfortify, GGally,
  PCAtools, # bioconductor
  FactoMineR,
  ggrepel,
  boot,
  caret)

# conflict_scout()
conflicts_prefer(dplyr::slice,
  dplyr::filter,
  stats::screeplot,
  stats::biplot,
  palmerpenguins::penguins)
```

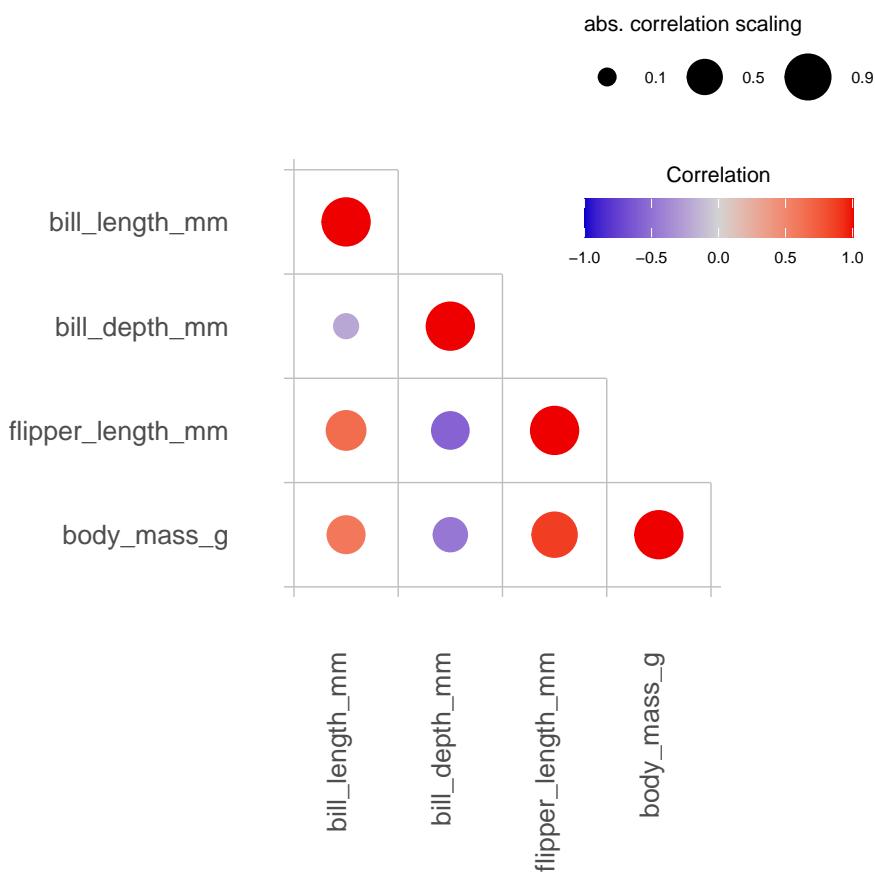
```
[conflicted] Will prefer dplyr::slice over any other package.
[conflicted] Will prefer dplyr::filter over any other package.
[conflicted] Will prefer stats::screeplot over any other package.
[conflicted] Will prefer stats::biplot over any other package.
[conflicted] Will prefer palmerpenguins::penguins over any other package.
```

```
rawdata <- penguins |>
  na.omit()
rawdata <- mutate(rawdata,
  ID=paste('P', 1:nrow(rawdata))) |>
  select(ID, everything())
```

```
predvars <- ColSeeker(namepattern = c('_mm', '_g'))
```

## 25.1 Exploration of correlations between predictor variables

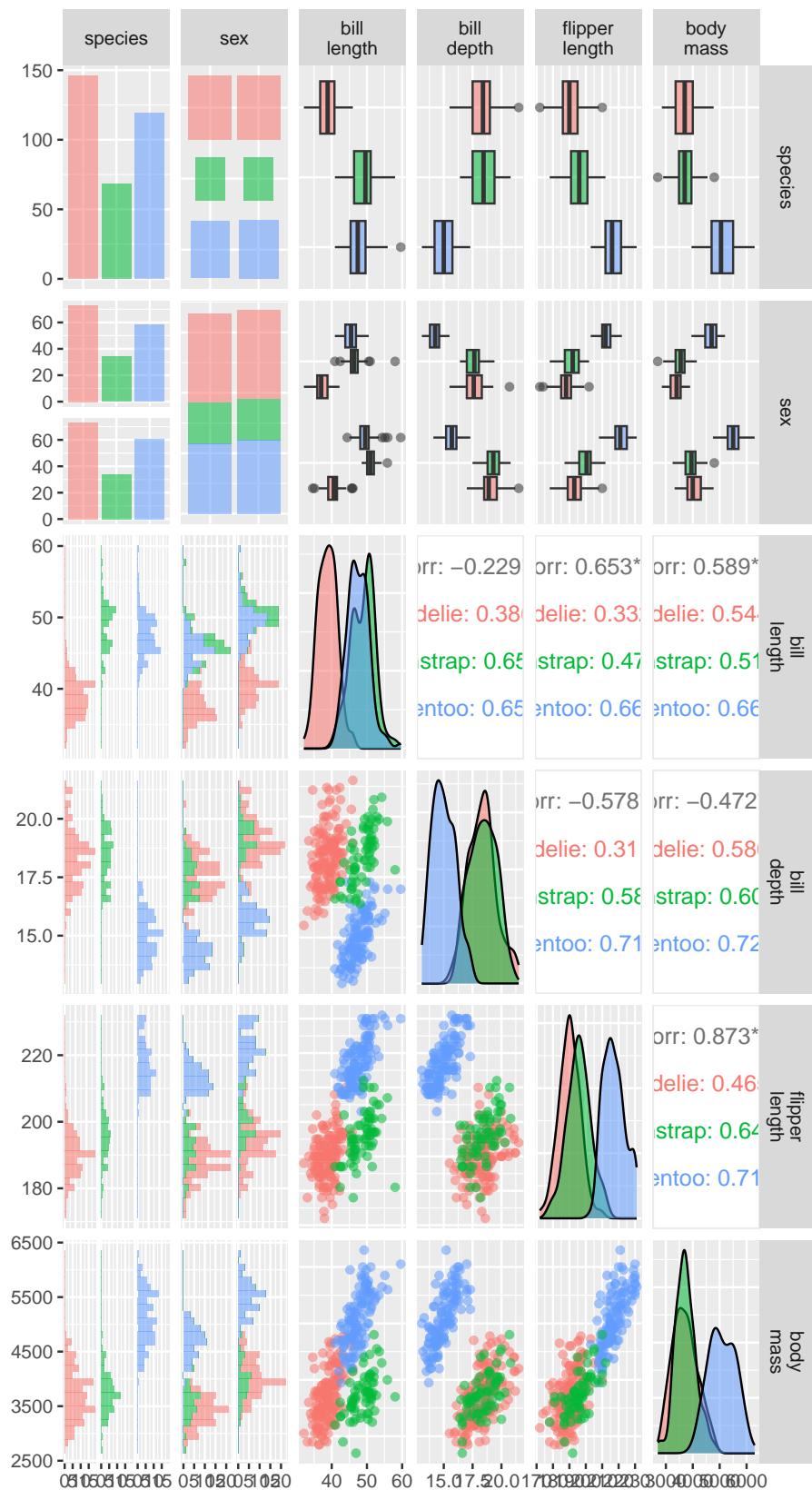
```
cortestR(rawdata |>
  # filter(species == "Adelie") |>
  select(predvars$names),
  split = T) |>
pluck('corout') |>
ggcormat(maxpoint = 8)
```



```
ggpairs(rawdata |> select(species, sex,
                           predvars$names) |>
  rename_with(~str_replace(.x, '(.+)_(.+)_+', '\\\1 \\\2')) |>
```

```
    str_wrap(8)),  
  aes(color=species, alpha=.5))
```

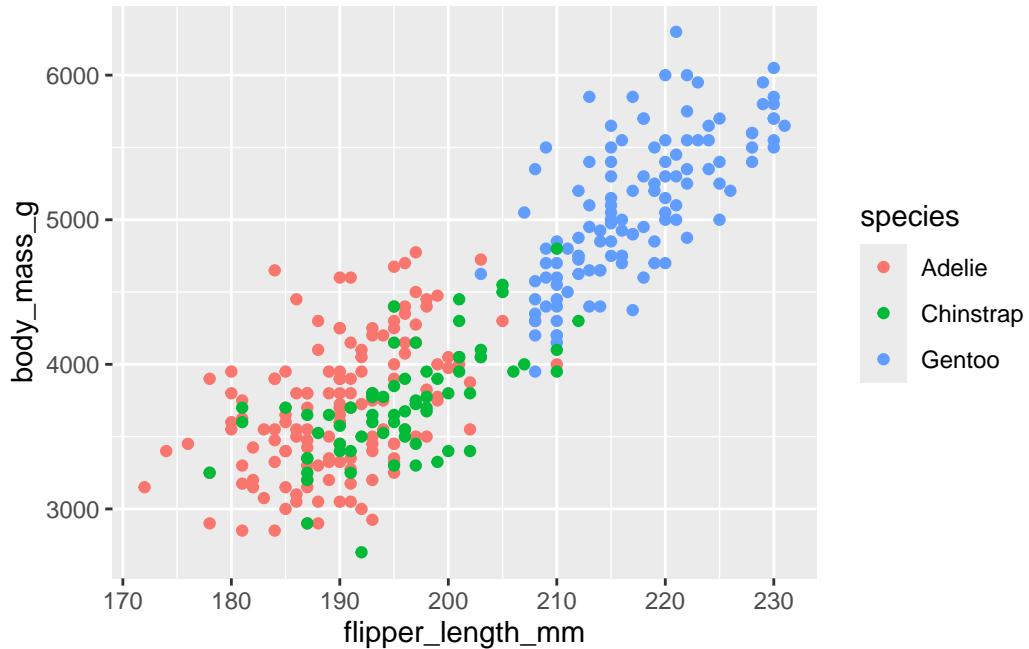
```
`stat_bin()` using `bins = 30`. Pick better value `binwidth`.  
`stat_bin()` using `bins = 30`. Pick better value `binwidth`.  
`stat_bin()` using `bins = 30`. Pick better value `binwidth`.  
`stat_bin()` using `bins = 30`. Pick better value `binwidth`.  
`stat_bin()` using `bins = 30`. Pick better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



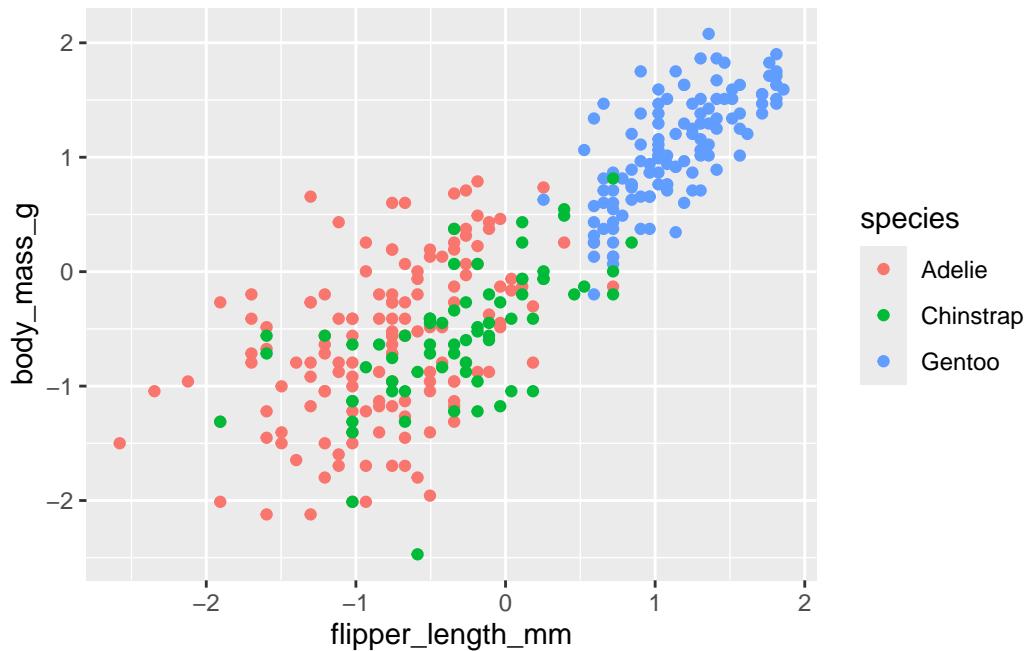
[Video on PCA](#)

## 25.2 Two variable example

```
predvars2 <- ColSeeker(namepattern = c('body','flipper'))
ggplot(rawdata, aes(.data[[predvars2$names[1]]], .data[[predvars2$names[2]]]))+
  geom_point(aes(color=species))
```

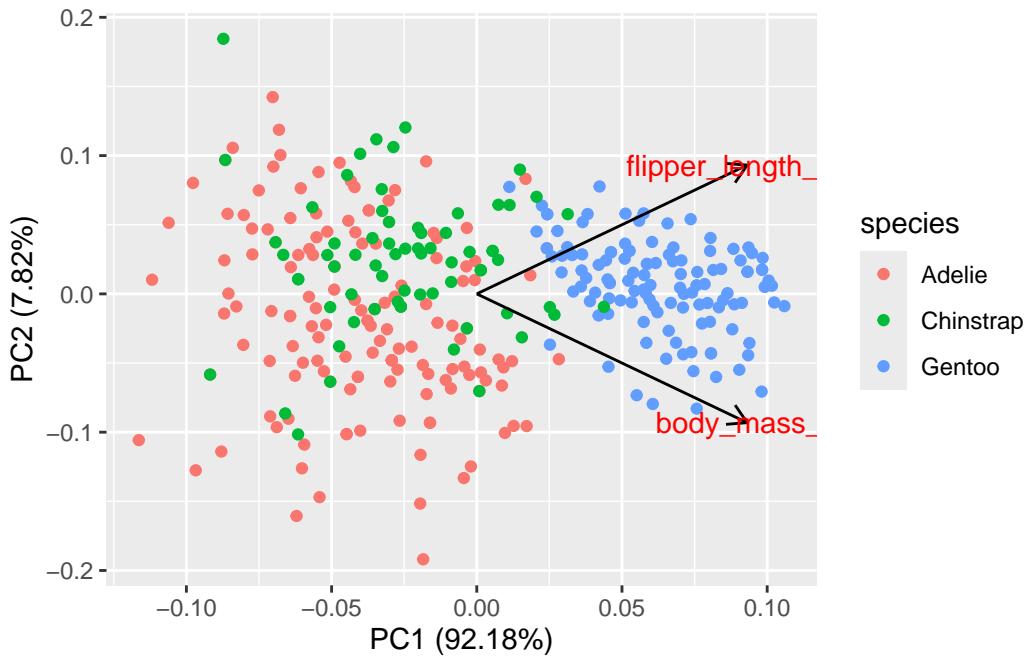


```
v2data <- predict(preProcess(rawdata |>
  select(predvars2$names),
  method = c("YeoJohnson","center", "scale")),
  rawdata) |>
  select(species,predvars2$names)
ggplot(v2data, aes(.data[[predvars2$names[1]]], .data[[predvars2$names[2]]]))+
  geom_point(aes(color=species))
```



```
pca2_out <- prcomp(v2data |>
  select(predvars2$names),
  center = F, scale. = F)
autoplot(pca2_out, data=v2data, colour='species',
  loadings = TRUE, loadings.colour = 'black',
  loadings.label = TRUE, loadings.label.size = 4)
```

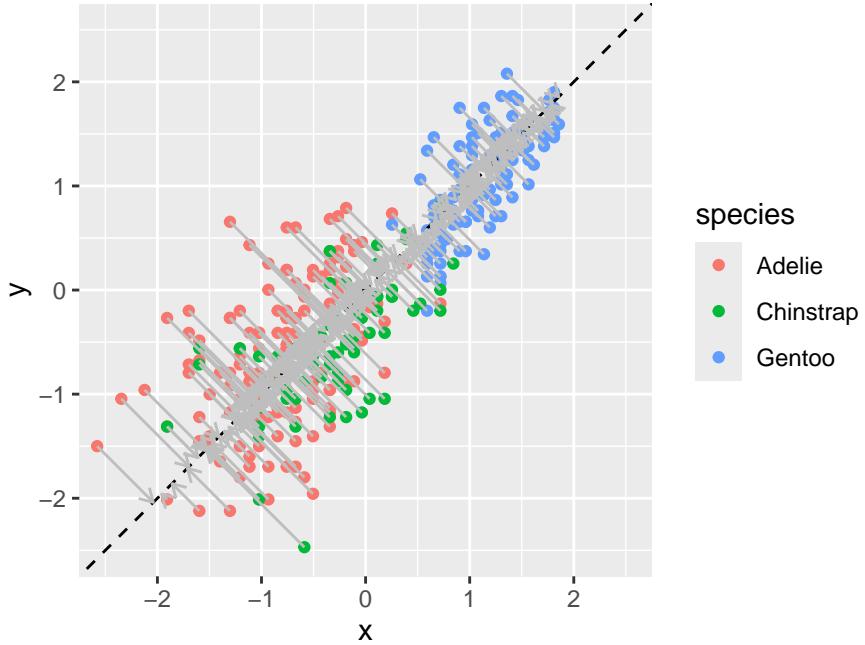
Warning: `aes\_string()` was deprecated in ggplot2 3.0.0.  
 i Please use tidy evaluation idioms with `aes()`.  
 i See also `vignette("ggplot2-in-packages")` for more information.  
 i The deprecated feature was likely used in the ggfortify package.  
 Please report the issue at <<https://github.com/sinhrks/ggfortify/issues>>.



```
# Extract PC1 loadings
pc1_loadings <- pca2_out$rotation[, 1]

# Calculate axis endpoint
axis_end <- c(pc1_loadings[1] * 2, pc1_loadings[2] * 2)
slope_pc1 <- pc1_loadings[2] / pc1_loadings[1]
# Create the plot
ggplot(v2data, aes(.data[[predvars2$names[1]]],
                    .data[[predvars2$names[2]]])) +
  geom_point(aes(0,0))+
  geom_point(aes(color = species)) +
  geom_abline(intercept = 0, slope = slope_pc1,
              color = "black", linetype = "dashed") +
  geom_segment(aes(xend = (.data[[predvars2$names[1]]] +
                        .data[[predvars2$names[2]]])/2,
                   yend = (.data[[predvars2$names[1]]] +
                        .data[[predvars2$names[2]]])/2),
               color = "grey",
               arrow = arrow(length = unit(0.2, "cm")))+
  coord_fixed(ratio = 1, xlim = c(-2.5, 2.5), ylim = c(-2.5, 2.5))
```

Warning in geom\_point(aes(0, 0)): All aesthetics have length 1, but the data has 333 rows.  
i Please consider using `annotate()` or provide this layer with data containing  
a single row.



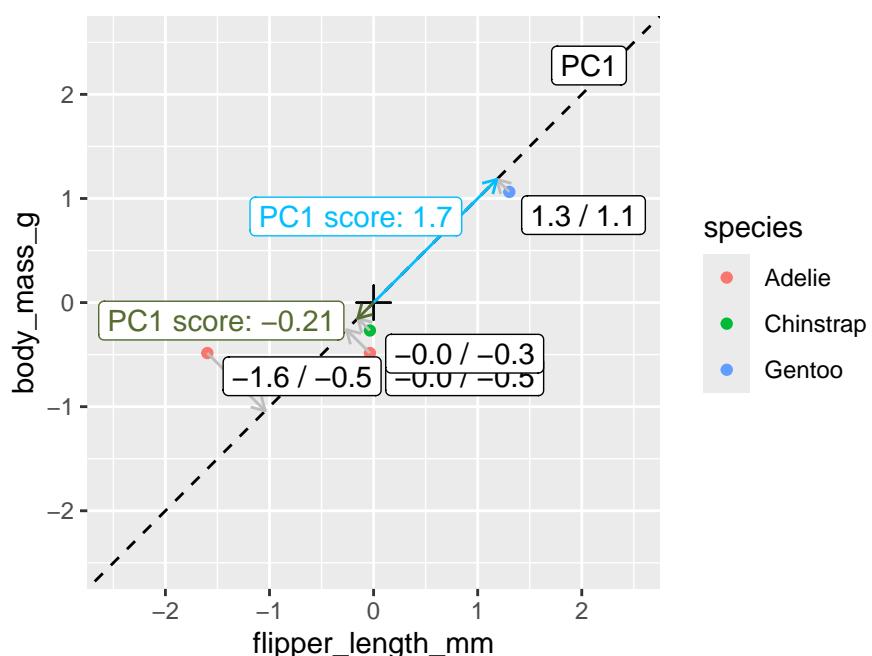
```

pc1plot <-
  ggplot(v2data |> slice(1,101, 201,301), aes(.data[[predvars2$names[1]]],
                                                 .data[[predvars2$names[2]]])) +
  geom_point(x=0,y=0, shape=3, size=4) +
  geom_point(aes(color = species)) +
  geom_abline(intercept = 0, slope = slope_pc1,
              color = "black", linetype = "dashed") +
  annotate("label",x=2.5,y=2.5,label = paste("PC1"),
           hjust = 1.1, vjust = 1.1, color="black")+
  geom_segment(aes(xend = (.data[[predvars2$names[1]]]+
                           .data[[predvars2$names[2]]])/2,
                   yend = (.data[[predvars2$names[1]]]+
                           .data[[predvars2$names[2]]])/2),
               color = "grey",
               arrow = arrow(length = unit(0.2, "cm")))+
  geom_label(aes(label = paste(roundR(.data[[predvars2$names[1]]]),"/",
                               roundR(.data[[predvars2$names[2]]]))),
             hjust = -.1, vjust = 1.1)+
  annotate("segment",x = 0, y = 0,
           xend = (v2data[[predvars2$names[1]]][201]+
                    v2data[[predvars2$names[2]]][201])/2,
           yend = (v2data[[predvars2$names[1]]][201]+
                    v2data[[predvars2$names[2]]][201])/2,
           arrow = arrow(length = unit(0.2, "cm")), color = "deepskyblue")+
  annotate("label",x=1.05,y=1.05,label = paste("PC1 score:",
```

```

    roundR(pca2_out$x[201,1])),
    hjust = 1.1, vjust = 1.1, color="deepskyblue")+
  annotate("segment",x = 0, y = 0,
           xend = (v2data[[predvars2$names[1]]][301]+
           v2data[[predvars2$names[2]]][301])/2,
           yend = (v2data[[predvars2$names[1]]][301]+
           v2data[[predvars2$names[2]]][301])/2,
           arrow = arrow(length = unit(0.2, "cm")), color = "darkolivegreen")+
  annotate("label",x=-0.05,y=0.05,label = paste("PC1 score:",
                                                roundR(pca2_out$x[301,1])),
           hjust = 1.1, vjust = 1.1, color="darkolivegreen")+
  coord_fixed(ratio = 1, xlim = c(-2.5, 2.5), ylim = c(-2.5, 2.5))
pc1plot

```



```

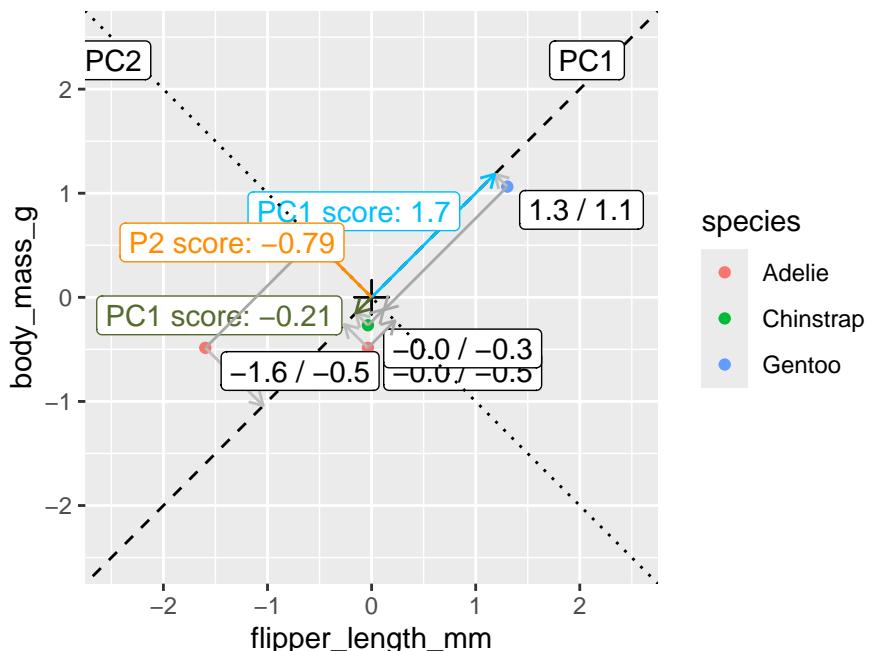
pc1plot+
  geom_abline(intercept = 0, slope = -slope_pc1,
              color = "black", linetype = "dotted")+
  annotate("label",x=-2.05,y=2.5,label = paste("PC2"),
           hjust = 1.1, vjust = 1.1, color="black")+
  geom_segment(aes(xend = (.data[[predvars2$names[1]]]-
           .data[[predvars2$names[2]]])/2,
                   yend = (.data[[predvars2$names[1]]]-
           .data[[predvars2$names[2]]])/-2),
               color = "darkgrey",

```

```

        arrow = arrow(length = unit(0.2, "cm"))+
annotate("segment",x = 0, y = 0,
         xend = (v2data[[predvars2$names[1]]][1]-
                  v2data[[predvars2$names[2]]][1])/2,
         yend = (v2data[[predvars2$names[1]]][1]-
                  v2data[[predvars2$names[2]]][1])/-2,
         arrow = arrow(length = unit(0.2, "cm")), color = "darkorange")+
annotate("label",x=-0.05,y=0.75,label = paste("P2 score:", roundR(pca2_out$x[1,2])),
         hjust = 1.1, vjust = 1.1, color="darkorange")

```



```
pca2_out$x[c(1,101, 201,301),]
```

	PC1	PC2
[1,]	-1.4719146	-0.7863321
[2,]	-0.3676829	0.3178996
[3,]	1.6738713	0.1701927
[4,]	-0.2143573	0.1645741

```
v2data[c(1,101, 201,301),]
```

```
# A tibble: 4 x 3
species   flipper_length_mm body_mass_g
```

	<fct>	<dbl>	<dbl>
1	Adelie	-1.60	-0.485
2	Adelie	-0.0352	-0.485
3	Gentoo	1.30	1.06
4	Chinstrap	-0.0352	-0.268

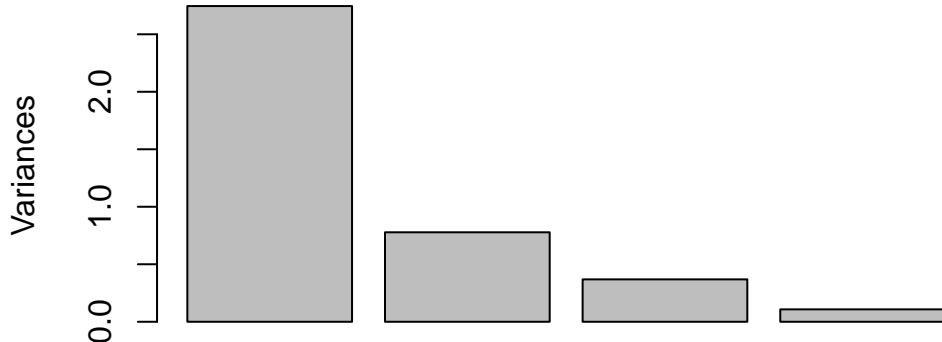
```
pca_out <- prcomp(rawdata |>
  select(predvars$names),
  center = T, scale. = T)
summary(pca_out)
```

Importance of components:

	PC1	PC2	PC3	PC4
Standard deviation	1.6569	0.8821	0.60716	0.32846
Proportion of Variance	0.6863	0.1945	0.09216	0.02697
Cumulative Proportion	0.6863	0.8809	0.97303	1.00000

```
screeplot(pca_out, npcs = 4)
```

## pca\_out

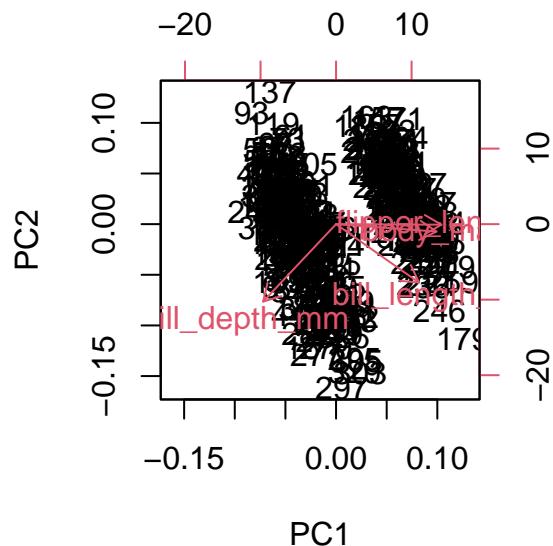


```
pca_out$rotation
```

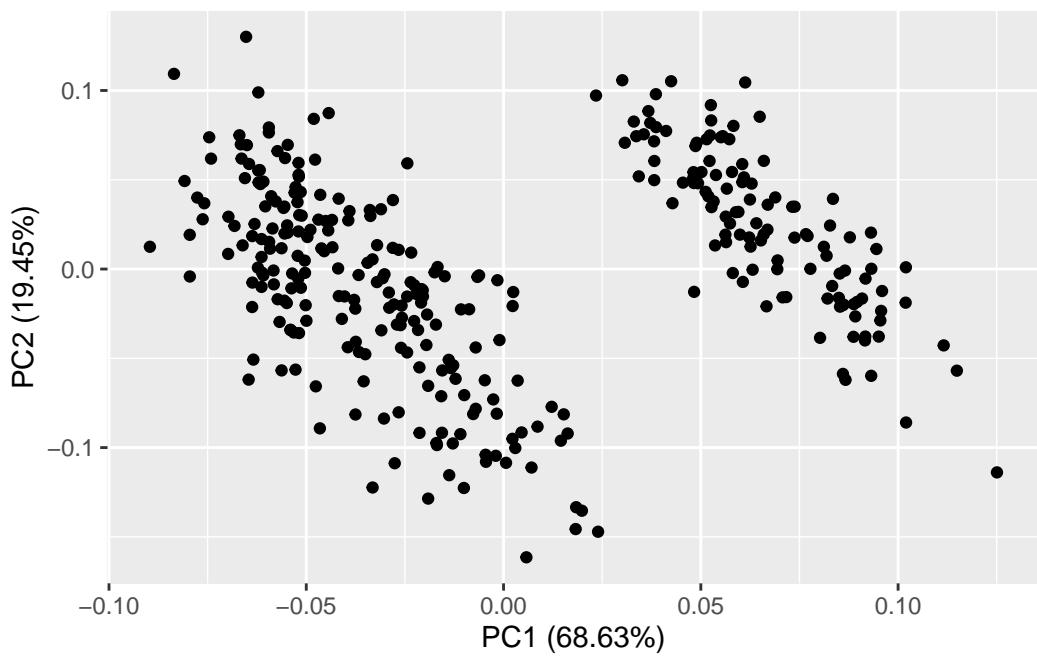
	PC1	PC2	PC3	PC4
bill_length_mm	0.4537532	-0.60019490	-0.6424951	0.1451695

```
bill_depth_mm      -0.3990472 -0.79616951  0.4258004 -0.1599044  
flipper_length_mm  0.5768250 -0.00578817  0.2360952 -0.7819837  
body_mass_g        0.5496747 -0.07646366  0.5917374  0.5846861
```

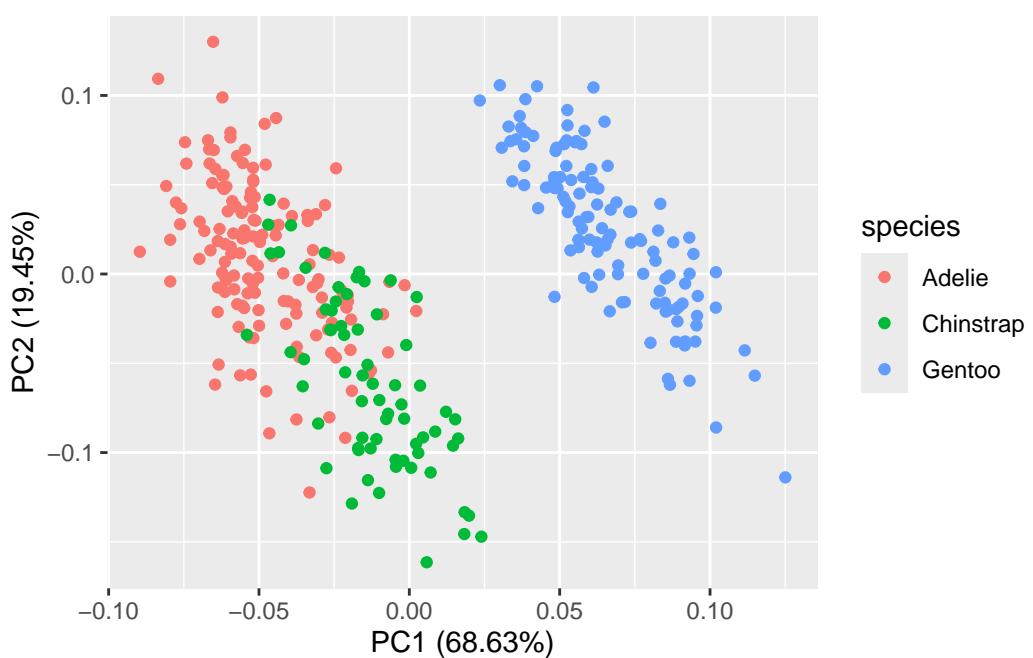
```
biplot(pca_out)
```



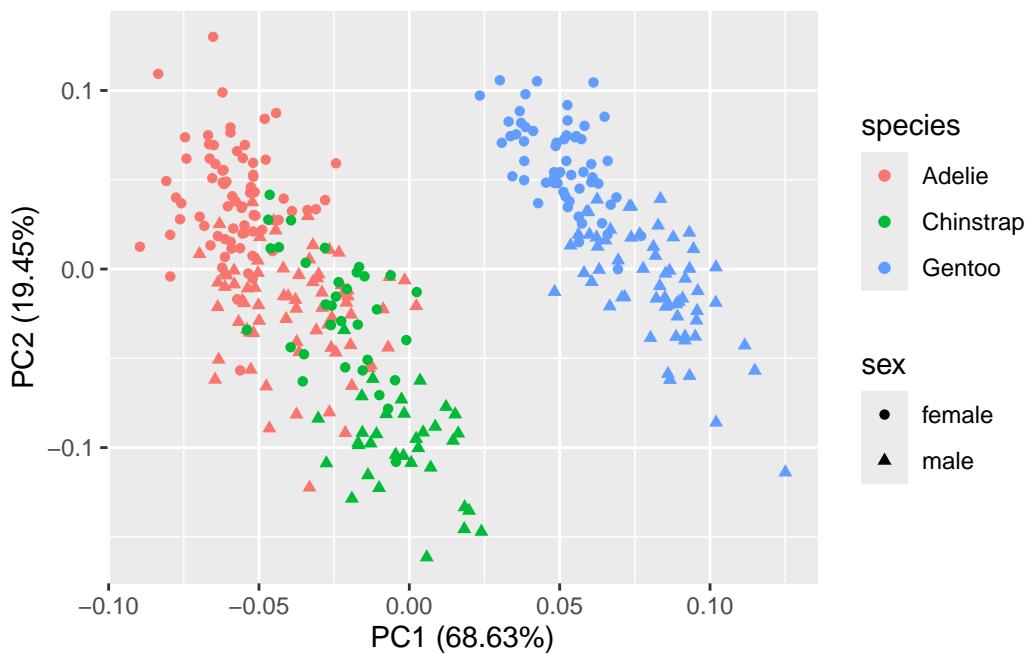
```
autoplot(pca_out)
```



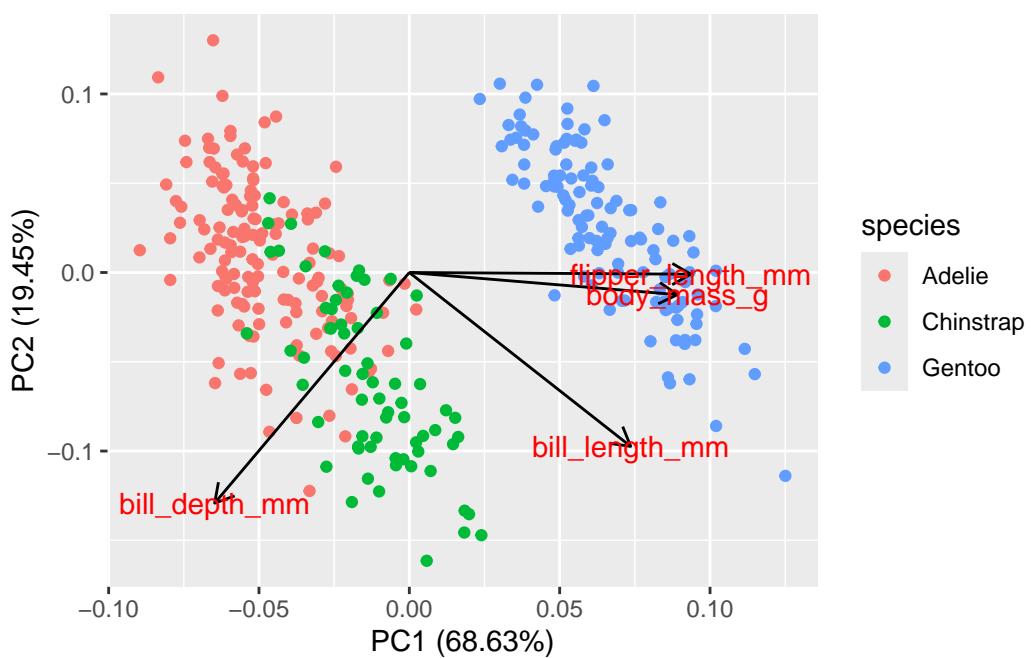
```
autoplot(pca_out, data=rawdata, colour='species')
```



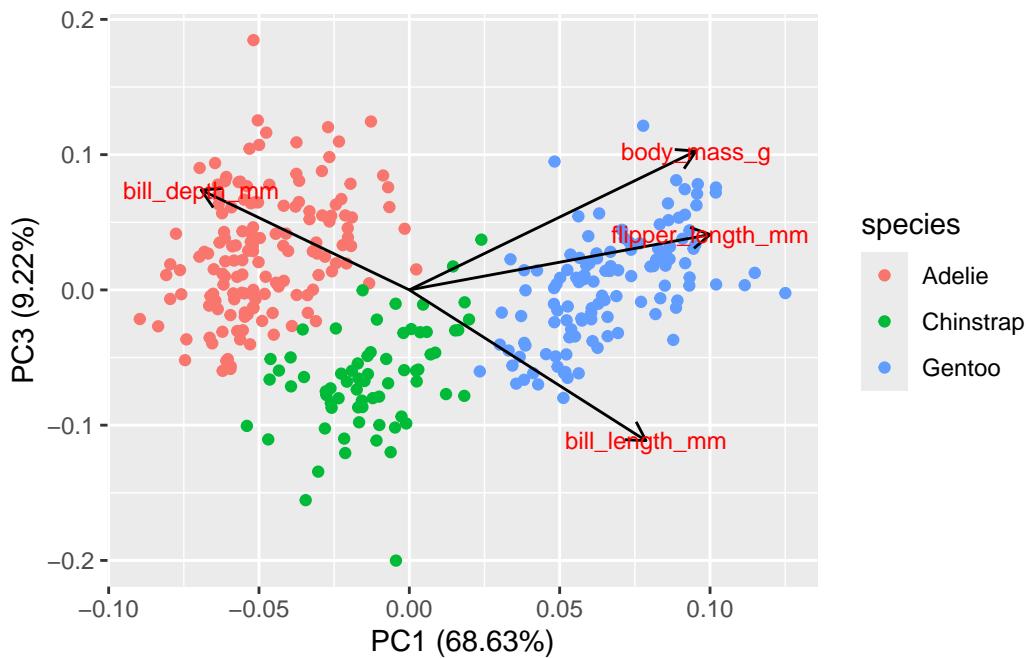
```
autoplot(pca_out, data=rawdata,
         colour='species', shape="sex")
```



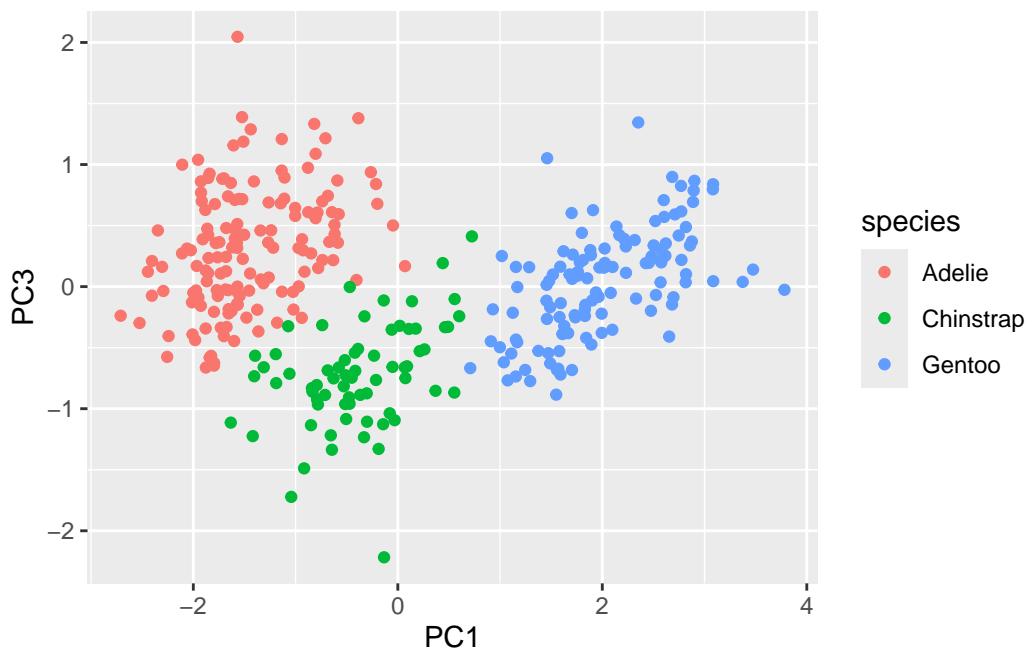
```
autoplot(pca_out, data=rawdata, colour='species',
         loadings = TRUE, loadings.colour = 'black',
         loadings.label = TRUE, loadings.label.size = 4)
```



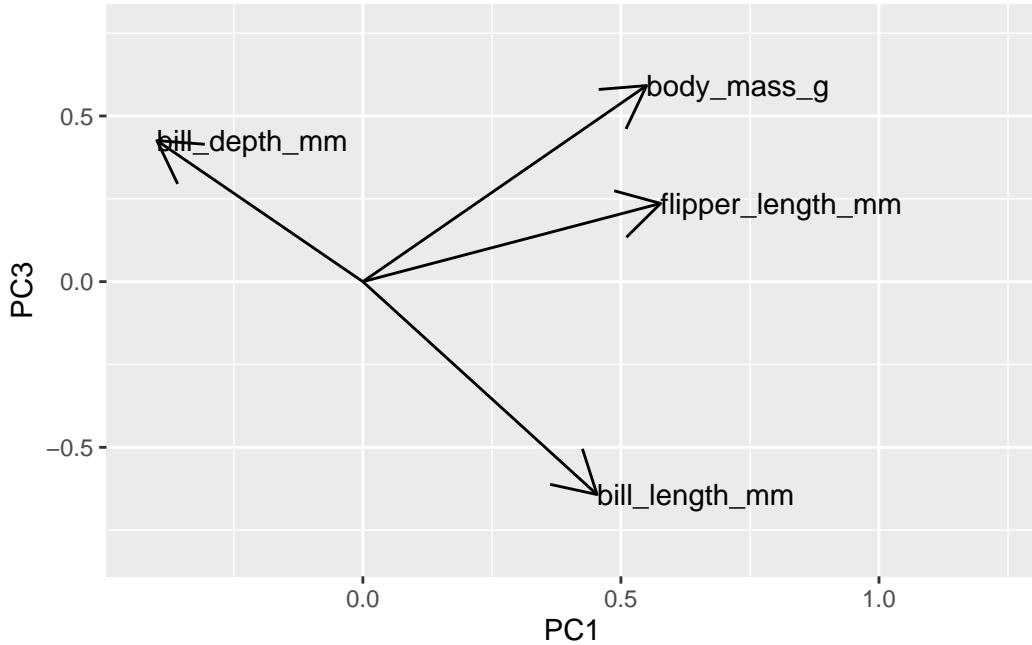
```
#contribution of variables to component
autoplot(pca_out, data=rawdata, colour='species',
         loadings = TRUE, loadings.colour = 'black',
         loadings.label = TRUE, loadings.label.size = 3,
         x=1,y=3)
```



```
pca_out$x |>
  as_tibble() |>
  bind_cols(rawdata) |>
  ggplot(aes(PC1,PC3,color=species))+
```



```
pca_out$rotation |>
  as_tibble(rownames = "Variable") |>
  ggplot(aes(PC1,PC3))+
  # geom_label_repel(aes(label=Variable))+ 
  geom_text(aes(label=Variable),
            hjust=0)+
  geom_segment(aes(xend=0,yend=0),
               arrow = arrow(end='first'))+
  scale_y_continuous(expand=expansion(.2))+ 
  scale_x_continuous(expand=expansion(
    mult = c(.1,.75)))
```

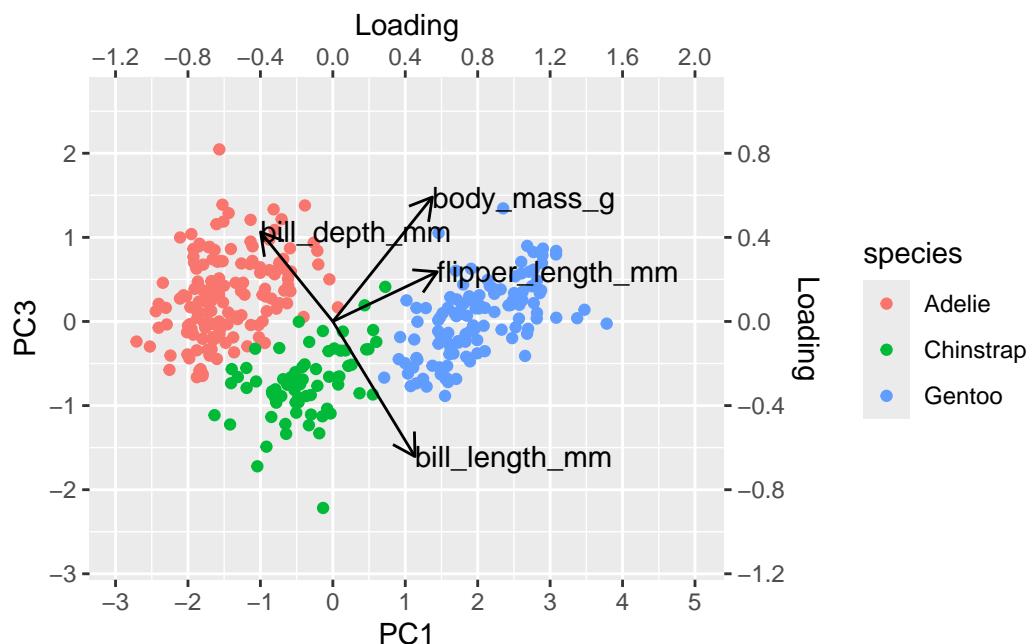


```
pca_loadings <- (pca_out$rotation*2.5) |>
  as_tibble(rownames = "Variable")
pca_out$x |>
  as_tibble() |>
  cbind(rawdata) |>
  ggplot(aes(PC1,PC3,color=species))+ 
  geom_point()+
  geom_text(
    data=pca_loadings,
    color="black",
    aes(label=Variable),
    hjust=0)+ 
  geom_segment(aes(xend=0,yend=0),
    data=pca_loadings,
    color="black",
    arrow = arrow(end='first',
                  length = unit(.05,
                                'npc')))+ 
  scale_y_continuous(expand=expansion(.2),
                     breaks=seq(-10,10,1),
                     sec.axis = sec_axis(
                       ~(. /2.5),
                       name = "Loading",
                       breaks = seq(-3,10,1)/2.5))+ 
  scale_x_continuous(expand=expansion(
```

```

mult = c(.1,.25)),
breaks=seq(-10,10,1),
sec.axis = sec_axis(
~(./2.5), name = "Loading",
breaks = seq(-10,10,1)/2.5))

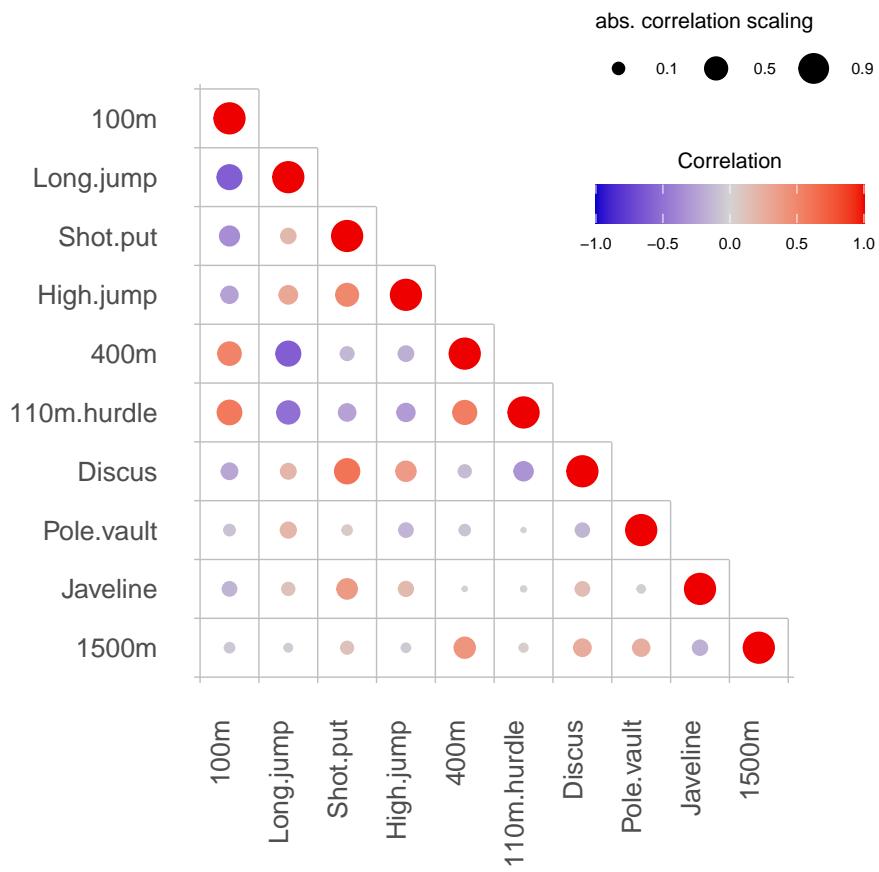
```



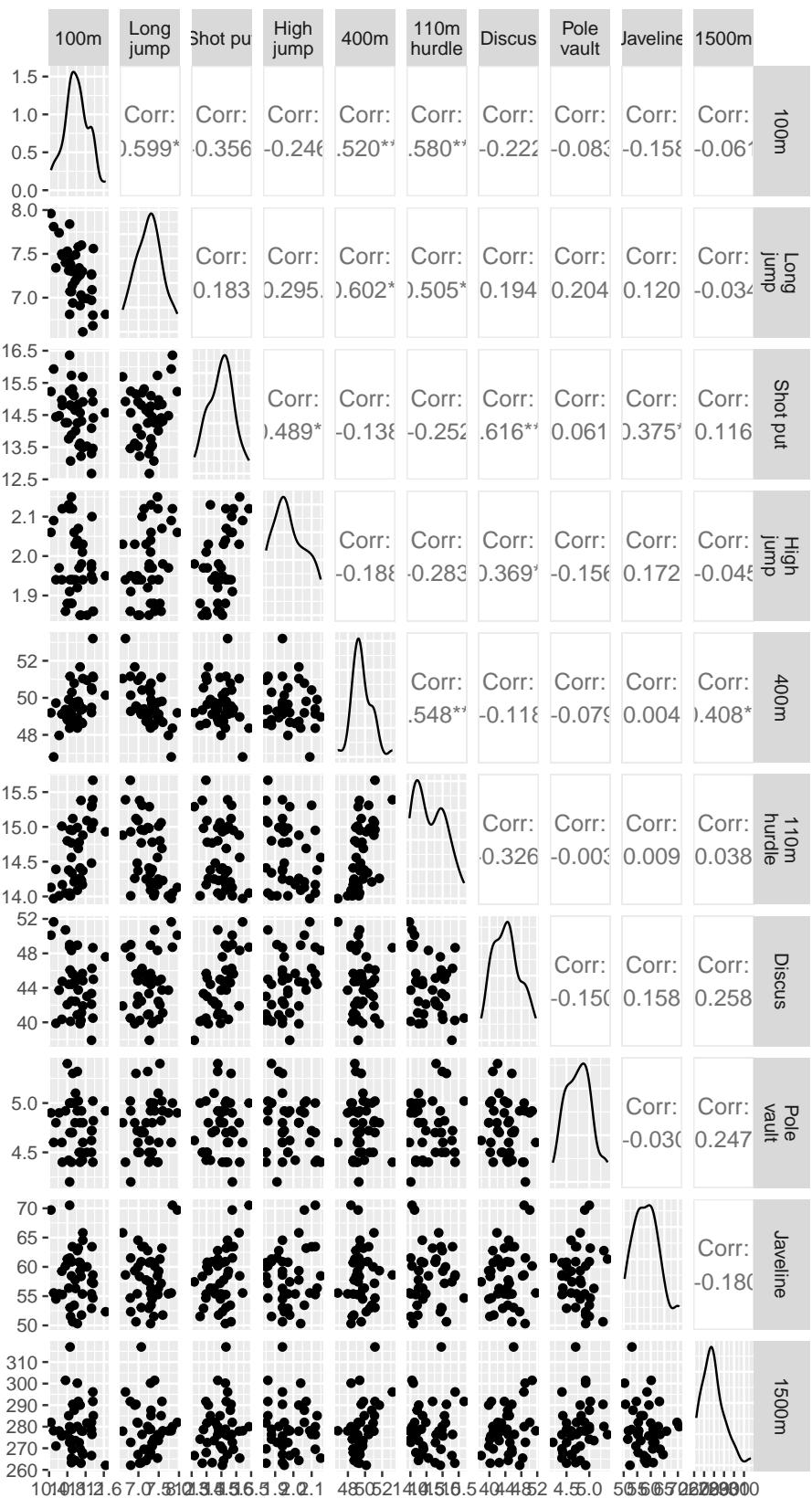
```

# decathlon
data("decathlon")
cortexR(decathlon |> select(1:10),
        split = T) |>
  pluck('corout') |>
  ggformat(maxpoint = 5)

```



```
ggpairs(decathlon |> select(1:10) |>
  rename_with(~str_replace(.x, '\\.', '_')) |>
  str_wrap(8)))
```



```
pca_out_deca <- prcomp(decathlon |> select(1:10),
                         center = T, scale. = T)
summary(pca_out_deca)
```

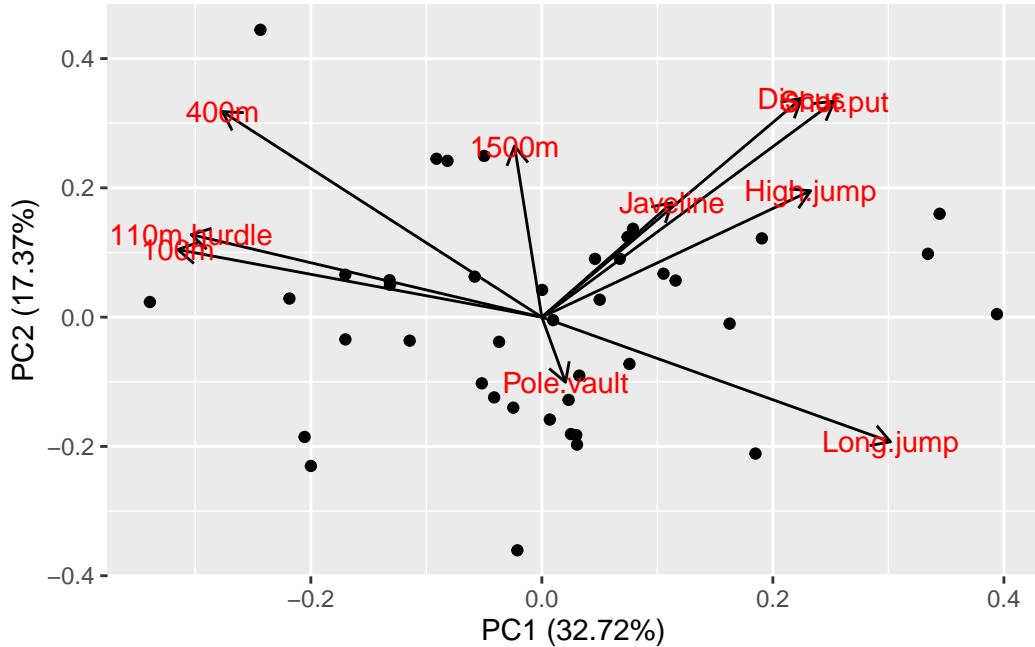
Importance of components:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7
Standard deviation	1.8088	1.3180	1.1853	1.0280	0.82751	0.77412	0.67174
Proportion of Variance	0.3272	0.1737	0.1405	0.1057	0.06848	0.05993	0.04512
Cumulative Proportion	0.3272	0.5009	0.6414	0.7471	0.81556	0.87548	0.92061
	PC8	PC9	PC10				
Standard deviation	0.62998	0.46348	0.42688				
Proportion of Variance	0.03969	0.02148	0.01822				
Cumulative Proportion	0.96030	0.98178	1.00000				

```
pca_out_deca$rotation |>
  as_tibble(rownames = 'Exercise') |>
  mutate(across(-Exercise,
                ~case_when(abs(.) < .25 ~ 0,
                           TRUE ~ .))) |>
  select(1:6)
```

# A tibble: 10 x 6	Exercise	PC1	PC2	PC3	PC4	PC5
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	100m	-0.428	0	0	0	0.365
2	Long.jump	0.410	-0.262	0	0	0
3	Shot.put	0.344	0.454	0	0	0
4	High.jump	0.316	0.266	0	0	0.671
5	400m	-0.376	0.432	0	0	0
6	110m.hurdle	-0.413	0	0	-0.283	0
7	Discus	0.305	0.460	0	0.253	0
8	Pole.vault	0	0	0.584	-0.536	0.399
9	Javeline	0	0	-0.329	-0.693	-0.369
10	1500m	0	0.360	0.660	0	0

```
autoplot(pca_out_deca, data=decathlon,
         loadings = TRUE, loadings.colour = 'black',
         loadings.label = TRUE, loadings.label.size = 4)
```



### 25.3 PCA bioconductor style

```
# PCA tools
pca_mat <- rawdata |> select(ID,predvars$names) |>
  column_to_rownames(var = 'ID') |>
  as.matrix() |>
  t()

pca_out3 <- pca(mat = pca_mat,
                  center = T,scale = T
)
getVars(pca_out3)
```

PC1	PC2	PC3	PC4
68.633893	19.452929	9.216063	2.697115

```
getLoadings(pca_out3)
```

	PC1	PC2	PC3	PC4
bill_length_mm	0.4537532	-0.60019490	-0.6424951	0.1451695
bill_depth_mm	-0.3990472	-0.79616951	0.4258004	-0.1599044
flipper_length_mm	0.5768250	-0.00578817	0.2360952	-0.7819837
body_mass_g	0.5496747	-0.07646366	0.5917374	0.5846861

```
PCAtools::screeplot(pca_out3)
```

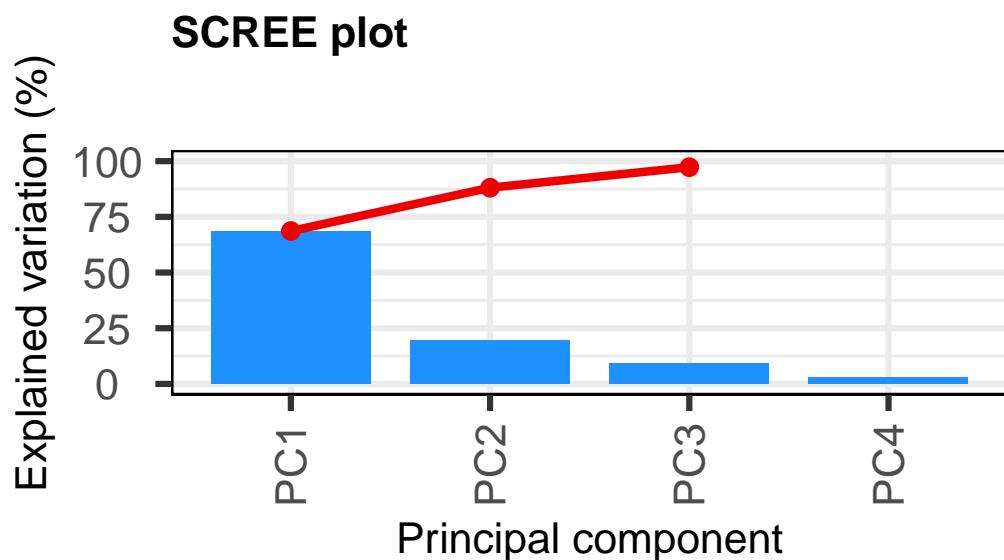
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.  
i Please use `linewidth` instead.

i The deprecated feature was likely used in the PCAtools package.  
Please report the issue to the authors.

Warning: The `size` argument of `element\_rect()` is deprecated as of ggplot2 3.4.0.  
i Please use the `linewidth` argument instead.  
i The deprecated feature was likely used in the PCAtools package.  
Please report the issue to the authors.

Warning: Removed 1 row containing missing values or values outside the scale range  
(`geom\_line()`).

Warning: Removed 1 row containing missing values or values outside the scale range  
(`geom\_point()`).



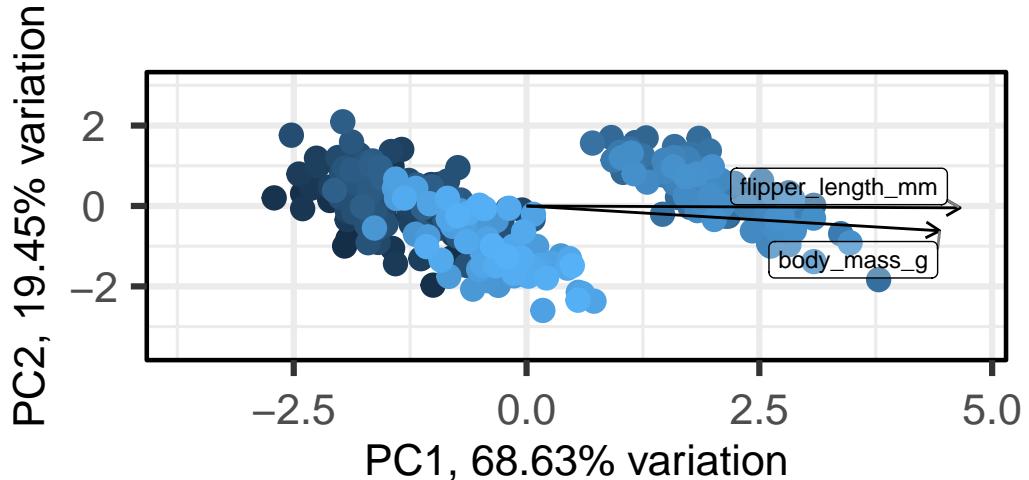
```
PCAtools::biplot(pca_out3,  
                  showLoadings = TRUE, ntopLoadings = 2,  
                  labSize = 5, pointSize = 3, sizeLoadingsNames = 3)
```

```
Ignoring unknown labels:
```

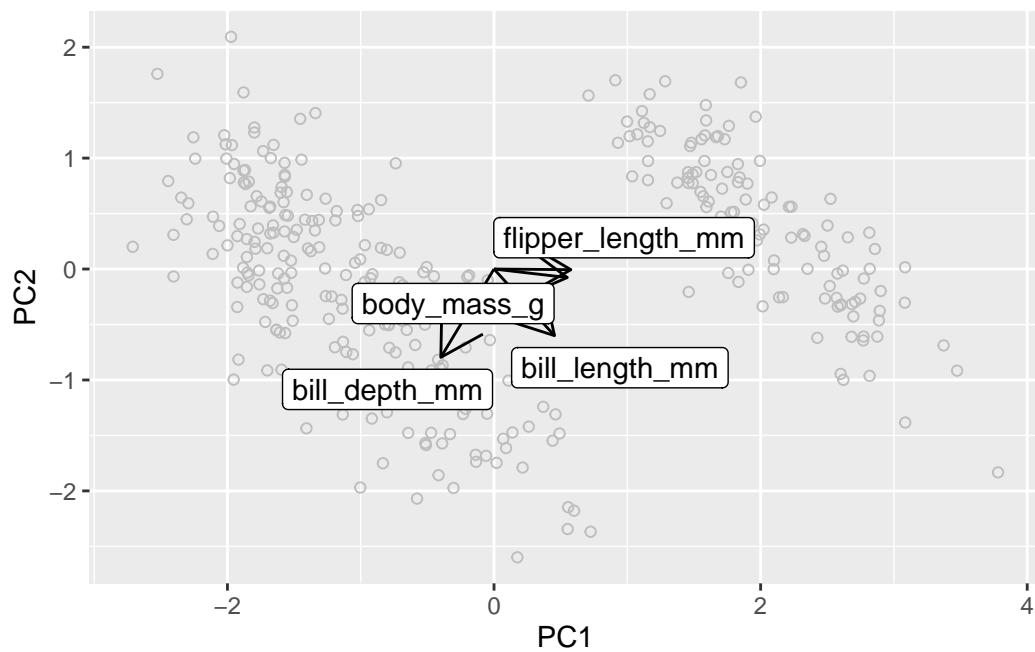
```
* fill : ""
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range  
(`geom_segment()`).
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range  
(`geom_label_repel()`).
```



```
# pairsplot(pca_out3)
# eigencorplot(pca_out3,
#               metavars=predvars$names)
pca_out3$loadings |>
  as_tibble(rownames='measure') |>
  ggplot(aes(PC1,PC2,shape=measure))+
  geom_point(data=pca_out3$rotated, color='grey', shape=1)+
  geom_segment(xend=0,yend=0,arrow=arrow(ends='first'))+
  ggrepel::geom_label_repel(aes(label=measure))
```



## 26 Linear discriminant analysis LDA

LDAs are used to find a linear combination of features that characterizes or separates two or more classes of objects or events. The resulting combination may be used as a linear classifier, or, more commonly, for dimensionality reduction before later classification. LDA is comparable to PCA, but instead of finding the component axes that maximize the variance of our data (PCA), we are interested in the axes that maximize the separation between multiple classes (LDA).

```
pacman::p_load(conflicted,
                 tidyverse,
                 wrappedtools,
                 here,
                 palmerpenguins,
                 ggfortify, GGally,
                 MASS,
                 caret
)

# conflict_scout()
conflicts_prefer(dplyr::select,
                  dplyr::filter,
                  palmerpenguins::penguins)
```

```
[conflicted] Will prefer dplyr::select over any other package.
[conflicted] Will prefer dplyr::filter over any other package.
[conflicted] Will prefer palmerpenguins::penguins over any other package.
```

```
rawdata <- penguins |>
  na.omit()
rawdata <- mutate(rawdata,
                  ID=paste('P', 1:nrow(rawdata))) |>
  select(ID, everything())
predvars <- ColSeeker(namepattern = c('_mm','_g'))
scaled <- rawdata |>
  select(predvars$names) |>
  caret::preProcess(method = c('center',"scale"))
rawdata <- predict(scaled,rawdata)
```

```

lda_formula <- paste('species',
                      paste(predvars$names, collapse='+'),
                      sep='~') |>
  as.formula()

lda_out <- lda(lda_formula, data=rawdata)
lda_out$prior

```

Adelie Chinstrap Gentoo  
0.4384384 0.2042042 0.3573574

```
lda_out$svd^2 / sum(lda_out$svd^2) # explained variance
```

[1] 0.8654754 0.1345246

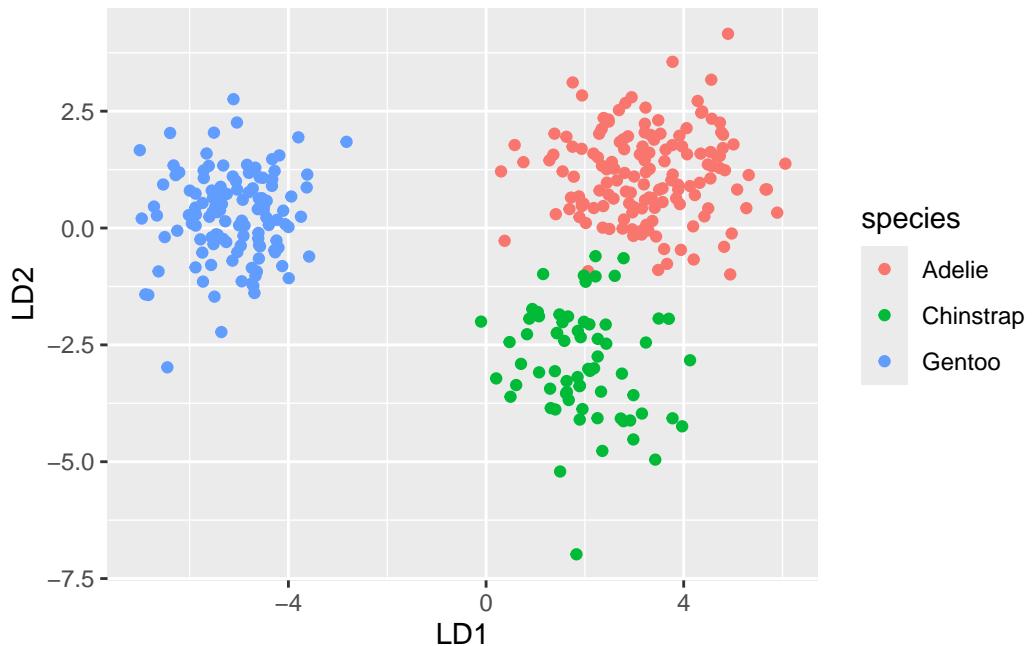
```
lda_out$scaling
```

	LD1	LD2
bill_length_mm	-0.4699047	-2.27825597
bill_depth_mm	2.0512477	-0.02052478
flipper_length_mm	-1.1850728	0.19966192
body_mass_g	-1.0849272	1.35726341

```

lda_pred <- predict(lda_out)
lda_plotdata <-
  lda_pred$x |>
  as_tibble() |>
  cbind(rawdata |> select(species))
lda_plotdata |>
  ggplot(aes(LD1, LD2, color=species))+
  geom_point()

```



```
confusionMatrix(lda_pred$class, rawdata$species)
```

#### Confusion Matrix and Statistics

		Reference		
Prediction	Adelie	Chinstrap	Gentoo	
Adelie	145	3	0	
Chinstrap	1	65	0	
Gentoo	0	0	119	

#### Overall Statistics

Accuracy : 0.988  
 95% CI : (0.9695, 0.9967)

No Information Rate : 0.4384

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9811

Mcnemar's Test P-Value : NA

#### Statistics by Class:

	Class: Adelie	Class: Chinstrap	Class: Gentoo
Sensitivity	0.9932	0.9559	1.0000

Specificity	0.9840	0.9962	1.0000
Pos Pred Value	0.9797	0.9848	1.0000
Neg Pred Value	0.9946	0.9888	1.0000
Prevalence	0.4384	0.2042	0.3574
Detection Rate	0.4354	0.1952	0.3574
Detection Prevalence	0.4444	0.1982	0.3574
Balanced Accuracy	0.9886	0.9761	1.0000

```

tdata <- readRDS(here('Data/cervical.RDS'))
predvars <- ColSeeker(tdata, namepattern = "-")
#preProcess
scale_rules <- tdata |>
  select(predvars$names) |>
  caret::preProcess(method = c("nzv",
                                "YeoJohnson",
                                "corr",
                                "scale", "center"))
tdata <- predict(scale_rules, tdata)
predvars <- ColSeeker(tdata, namepattern = "-")

lda_out2 <- lda(x = tdata[-(1:3)],
                 grouping=tdata$Tissuetype)

```

Warning in lda.default(x, grouping, ...): Variablen sind kollinear

```
lda_out2$prior
```

Control	Tumor
0.5	0.5

```
lda_out2$svd^2 / sum(lda_out2$svd^2) # explained var
```

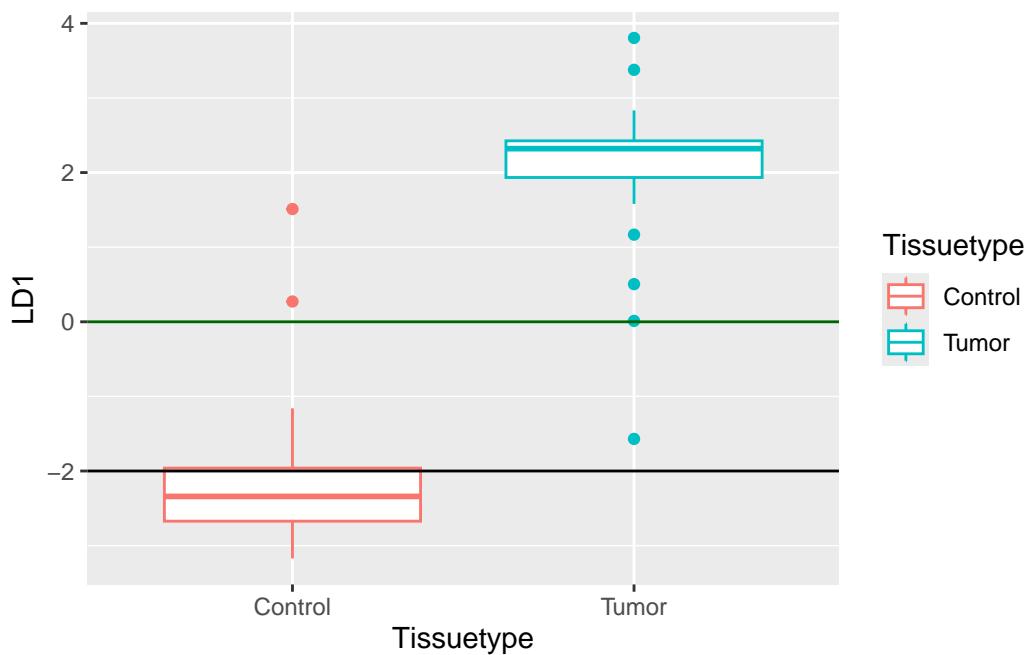
[1] 1

```
lda_out2$scaling |> head()
```

	LD1
let-7a	-0.020053423
let-7b	-0.003940093
let-7d	-0.005812444

```
let-7d* -0.028231729  
miR-1   -0.153359325  
miR-100 -0.015324858
```

```
lda_pred2 <- predict(lda_out2)  
lda_plotdata <-  
  lda_pred2$x |>  
  as_tibble() |>  
  cbind(tdata |> select(Tissuetype))  
lda_plotdata |>  
  ggplot(aes(Tissuetype, LD1, color=Tissuetype))+  
  geom_boxplot()  
  geom_hline(yintercept = 0, color="darkgreen") +  
  geom_hline(yintercept = -2)
```



```
confusionMatrix(lda_pred2$class, tdata$Tissuetype)
```

Confusion Matrix and Statistics

		Reference	
		Prediction	Tumor
Prediction	Control	27	1
	Tumor	2	28

```

Accuracy : 0.9483
95% CI : (0.8562, 0.9892)
No Information Rate : 0.5
P-Value [Acc > NIR] : 1.13e-13

Kappa : 0.8966

McNemar's Test P-Value : 1

Sensitivity : 0.9310
Specificity : 0.9655
Pos Pred Value : 0.9643
Neg Pred Value : 0.9333
Prevalence : 0.5000
Detection Rate : 0.4655
Detection Prevalence : 0.4828
Balanced Accuracy : 0.9483

'Positive' Class : Control

```

```

#data(bordeaux)
bordeaux <- readxl::read_excel(here('Data/bordeaux.xlsx')) |>
  mutate(quality=factor(quality,
                        levels=c('bad', 'medium', 'good')))
# import from excel!
lda_formula <- paste('quality',
                      paste(c('temperature', 'sun',
                             'heat', 'rain'), collapse='+'),
                      sep='~') #|>
# as.formula()

lda_out <- lda(as.formula(lda_formula), data=bordeaux)
lda_out$prior

```

```

bad      medium      good
0.3529412 0.3235294 0.3235294

```

```

lda_out$svd^2 / sum(lda_out$svd^2) # explained var

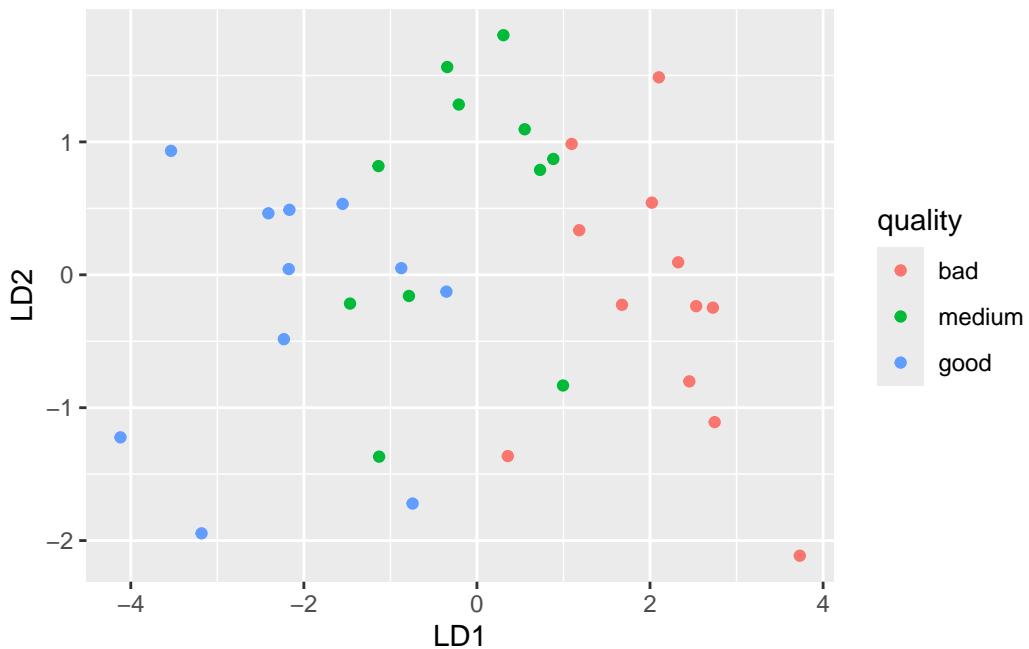
```

```
[1] 0.95945086 0.04054914
```

```
lda_out$scaling
```

	LD1	LD2
temperature	-0.008566046	4.625059e-05
sun	-0.006773869	5.329293e-03
heat	0.027054492	-1.276362e-01
rain	0.005865665	-6.174556e-03

```
lda_pred <- predict(lda_out)
lda_plotdata <-
  lda_pred$x |>
  as_tibble() |>
  cbind(bordeaux |> select(quality))
lda_plotdata |>
  ggplot(aes(LD1, LD2, color=quality)) +
  geom_point()
```



```
#  
confusionMatrix(lda_pred$class, bordeaux$quality)
```

Confusion Matrix and Statistics

Reference

Prediction	bad	medium	good
bad	10	1	0
medium	2	8	2
good	0	2	9

#### Overall Statistics

Accuracy : 0.7941  
 95% CI : (0.621, 0.913)  
 No Information Rate : 0.3529  
 P-Value [Acc > NIR] : 1.808e-07

Kappa : 0.6913

McNemar's Test P-Value : NA

#### Statistics by Class:

	Class: bad	Class: medium	Class: good
Sensitivity	0.8333	0.7273	0.8182
Specificity	0.9545	0.8261	0.9130
Pos Pred Value	0.9091	0.6667	0.8182
Neg Pred Value	0.9130	0.8636	0.9130
Prevalence	0.3529	0.3235	0.3235
Detection Rate	0.2941	0.2353	0.2647
Detection Prevalence	0.3235	0.3529	0.3235
Balanced Accuracy	0.8939	0.7767	0.8656

# 27 Cluster analysis

## 27.1 kmeans

```
pacman::p_load(conflicted, tidyverse,
                 wrappedtools,
                 palmerpenguins,
                 ggfortify, GGally,
                 factoextra,
                 caret, clue,
                 dendextend, circlize,
                 easystats, NbClust, mclust
)
```

Installiere Paket nach 'C:/Users/abusj/AppData/Local/R/win-library/4.5'  
(da 'lib' nicht spezifiziert)

installiere auch Abhängigkeit 'GlobalOptions'

Warning: kann nicht auf den Index für das Repository <http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/4.5/PACKAGES> nicht öffnen  
kann URL '<http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/4.5/PACKAGES>' nicht öffnen

Paket 'GlobalOptions' erfolgreich ausgepackt und MD5 Summen abgeglichen  
Paket 'circlize' erfolgreich ausgepackt und MD5 Summen abgeglichen

Die heruntergeladenen Binärpakete sind in  
C:\Users\abusj\AppData\Local\Temp\RtmpAB6miP\downloaded\_packages

circlize installed

Installiere Paket nach 'C:/Users/abusj/AppData/Local/R/win-library/4.5'  
(da 'lib' nicht spezifiziert)

Warning: kann nicht auf den Index für das Repository <http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/4.5/PACKAGES> nicht öffnen  
kann URL '<http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/4.5/PACKAGES>' nicht öffnen

```
Paket 'NbClust' erfolgreich ausgepackt und MD5 Summen abgeglichen
```

```
Die heruntergeladenen Binärpakete sind in
```

```
C:\Users\abusj\AppData\Local\Temp\RtmpAB6miP\downloaded_packages
```

```
NbClust installed
```

```
Installiere Paket nach 'C:/Users/abusj/AppData/Local/R/win-library/4.5'  
(da 'lib' nicht spezifiziert)
```

```
Warning: kann nicht auf den Index für das Repository http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/4.5/PACKAGES' nicht öffnen  
kann URL 'http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/4.5/PACKAGES' nicht öffnen
```

```
Paket 'mclust' erfolgreich ausgepackt und MD5 Summen abgeglichen
```

```
Die heruntergeladenen Binärpakete sind in
```

```
C:\Users\abusj\AppData\Local\Temp\RtmpAB6miP\downloaded_packages
```

```
mclust installed
```

```
# conflict_scout()  
conflicts_prefer(dplyr::slice,  
                  dplyr::filter,  
                  palmerpenguins::penguins)
```

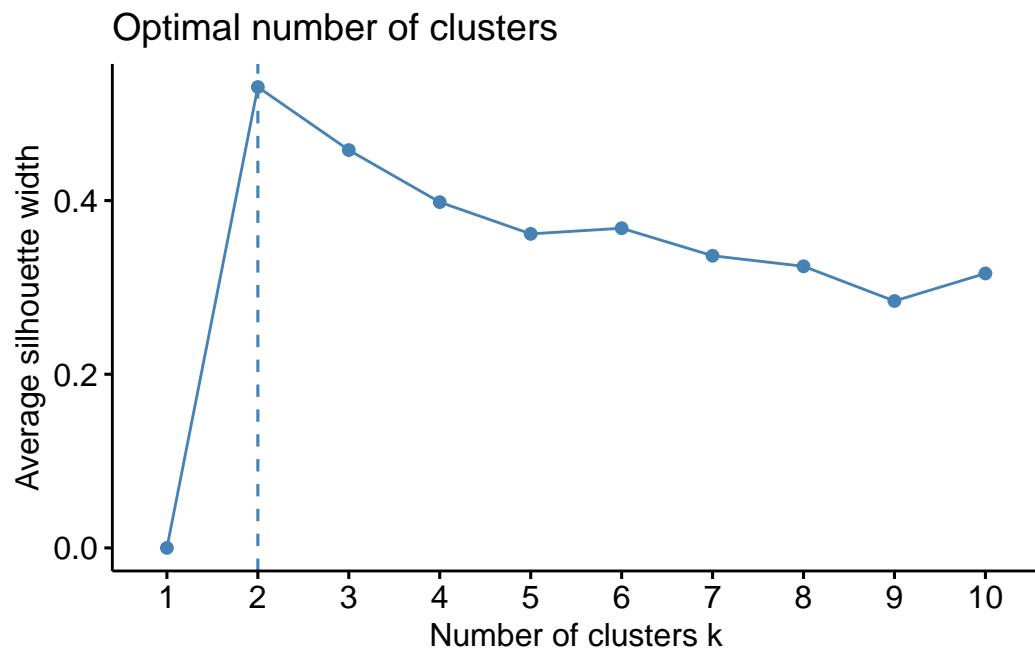
```
[conflicted] Will prefer dplyr::slice over any other package.
```

```
[conflicted] Will prefer dplyr::filter over any other package.
```

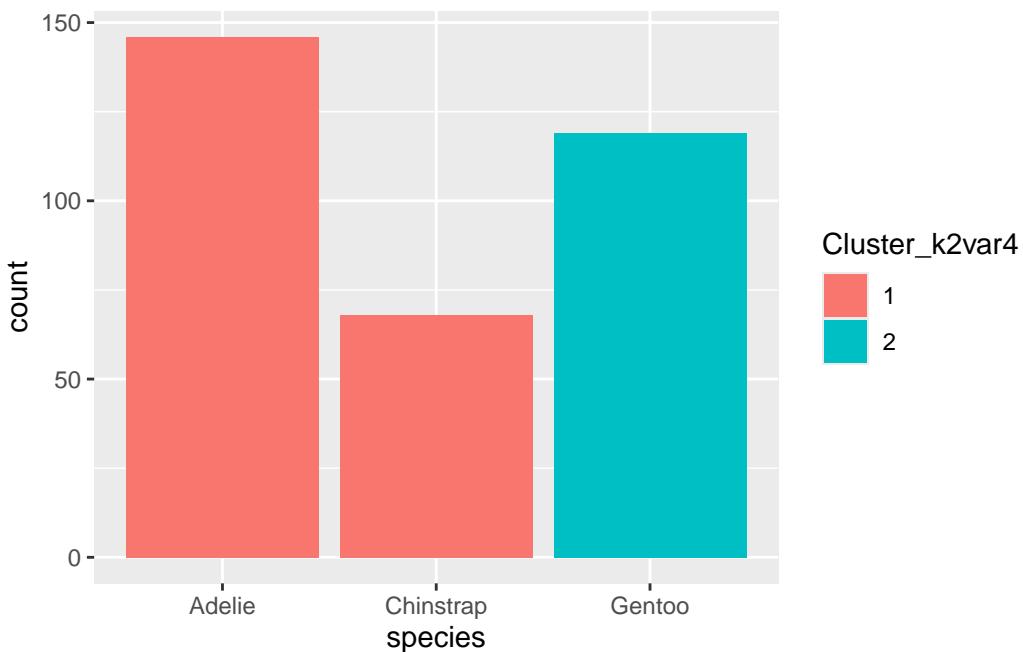
```
[conflicted] Will prefer palmerpenguins::penguins over any other package.
```

```
rawdata <- penguins |>  
na.omit()  
rawdata <- mutate(rawdata,  
                  ID=paste('P', 1:nrow(rawdata))) |>  
select(ID, everything())  
predvars <- ColSeeker(namepattern = c('_mm','_g'))  
  
rawdata <- predict(preProcess(rawdata |>  
                               select(predvars$names),  
                               method = c("center", "scale")),  
                               rawdata)
```

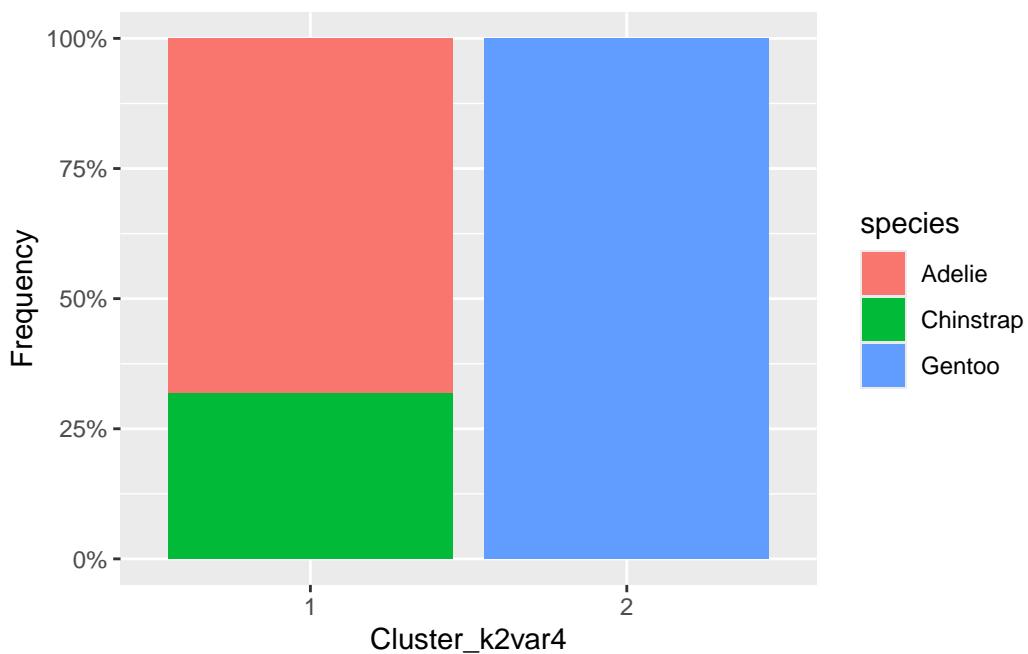
```
fviz_nbclust(rawdata |> select(predvars$names),  
             FUNcluster = kmeans)
```



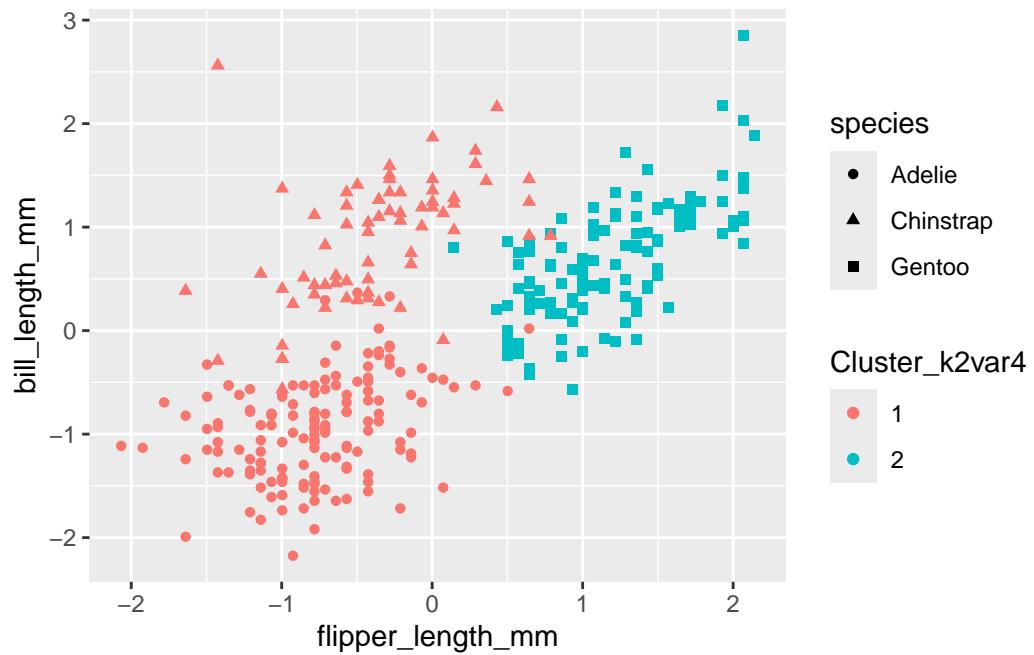
```
kmeans_out <- kmeans(rawdata |> select(predvars$names),  
                      centers = 2)  
  
rawdata <-  
  mutate(rawdata, Cluster_k2var4=kmeans_out$cluster |> as.factor())  
rawdata |>  
  ggplot(aes(species, fill=Cluster_k2var4)) +  
  geom_bar()
```



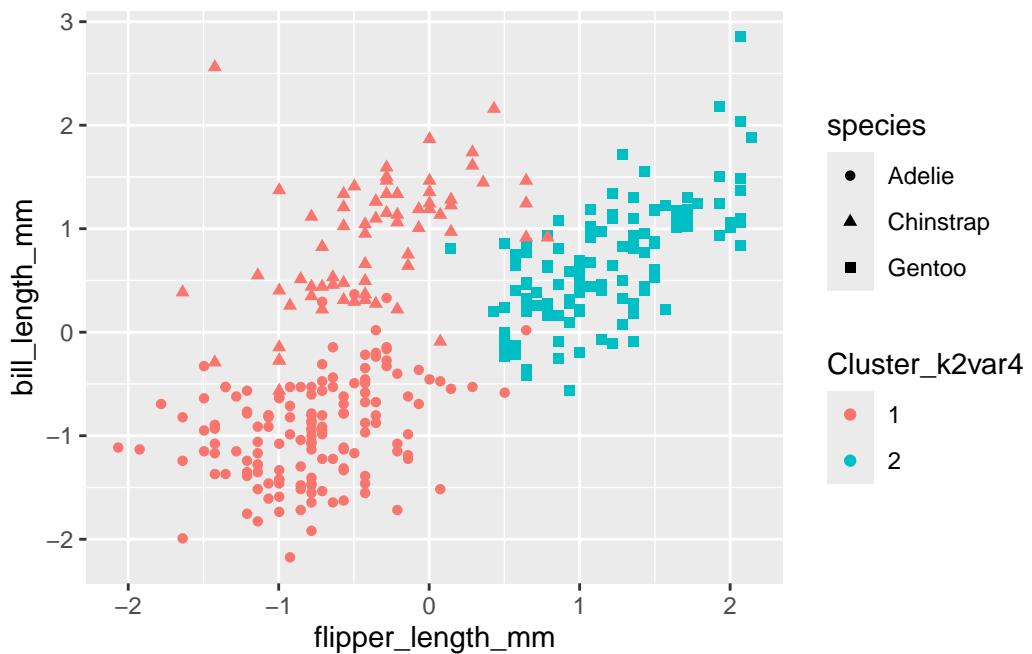
```
rawdata |>
  ggplot(aes(fill=species,x=Cluster_k2var4))+
  geom_bar(position = 'fill')+
  scale_y_continuous(name = 'Frequency', labels=scales::percent)
```



```
rawdata |>
  ggplot(aes(flipper_length_mm,bill_length_mm,
             shape=species,color=Cluster_k2var4))+  
  geom_point()
```



```
rawdata |>
  ggplot(aes(flipper_length_mm,bill_length_mm,
             shape=species,color=Cluster_k2var4))+  
  geom_point()
```



```
fviz_cluster(kmeans_out, rawdata |> select(predvars$names))
```

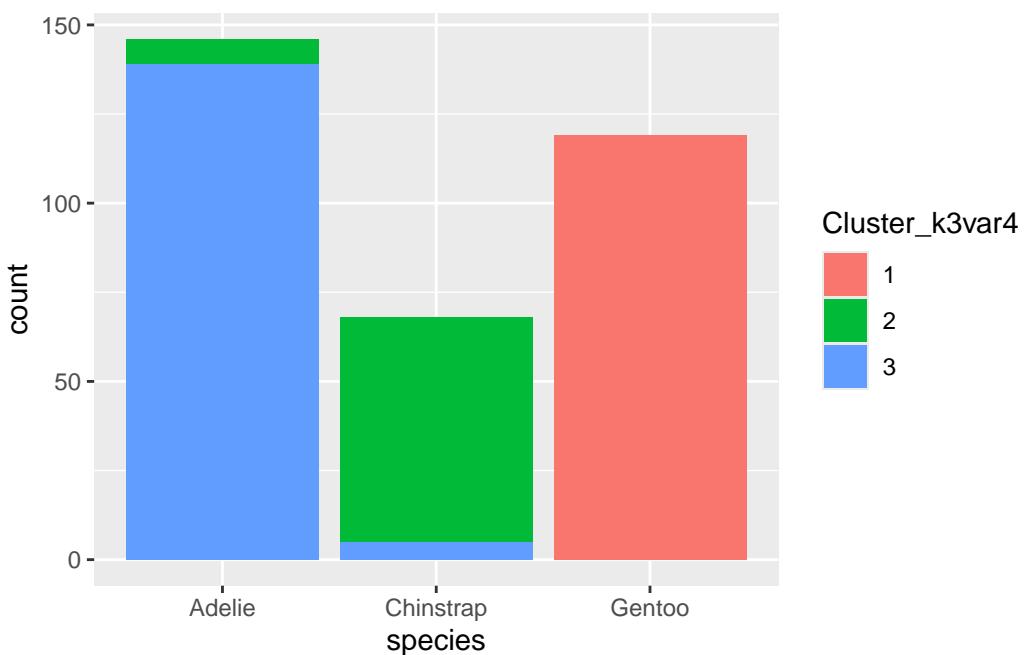


```
# predict(kmeans_out)
sample(clue::cl_predict(kmeans_out), 10)
```

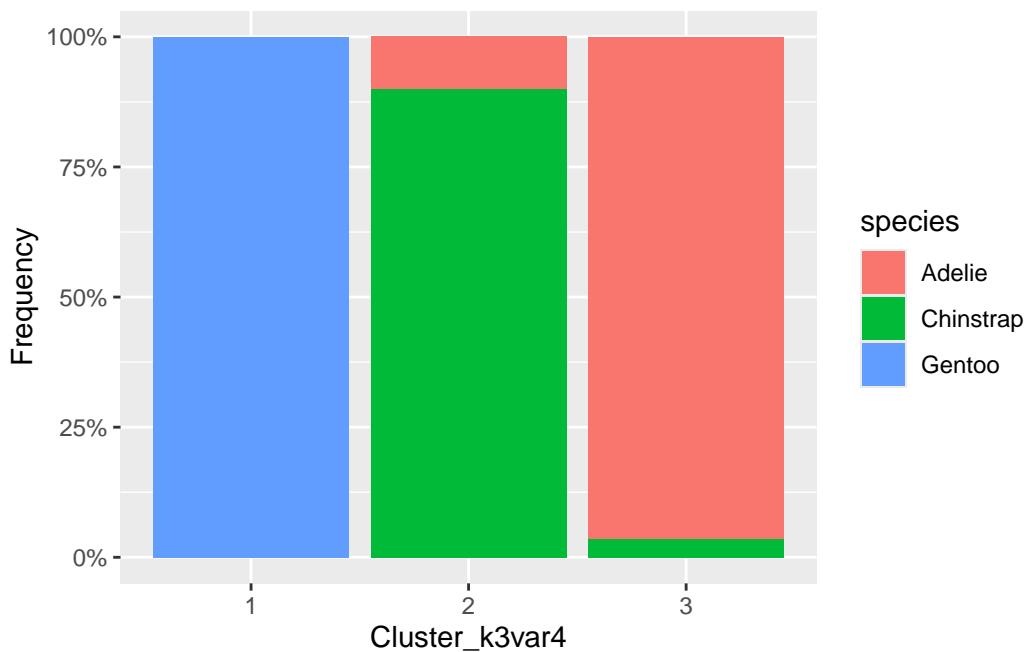
```
[1] 1 1 2 2 2 1 1 1 1 1
```

```
# more clusters
kmeans_out3 <- kmeans(rawdata |> select(predvars$names),
                        centers = 3)

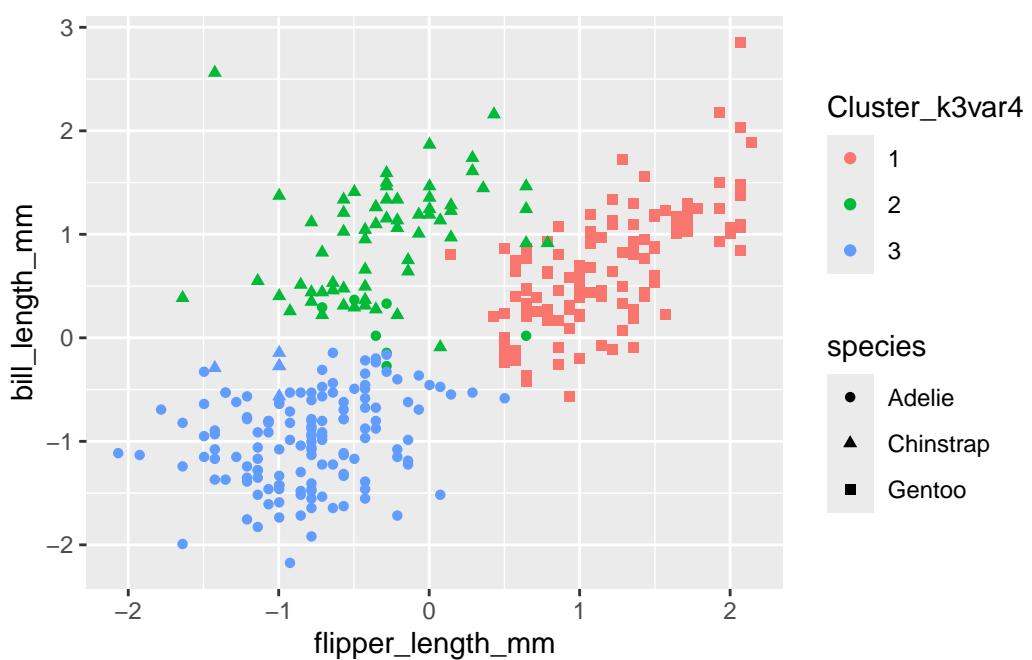
rawdata <-
  mutate(rawdata, Cluster_k3var4=kmeans_out3$cluster |> as.factor())
rawdata |>
  ggplot(aes(species, fill=Cluster_k3var4)) +
  geom_bar()
```



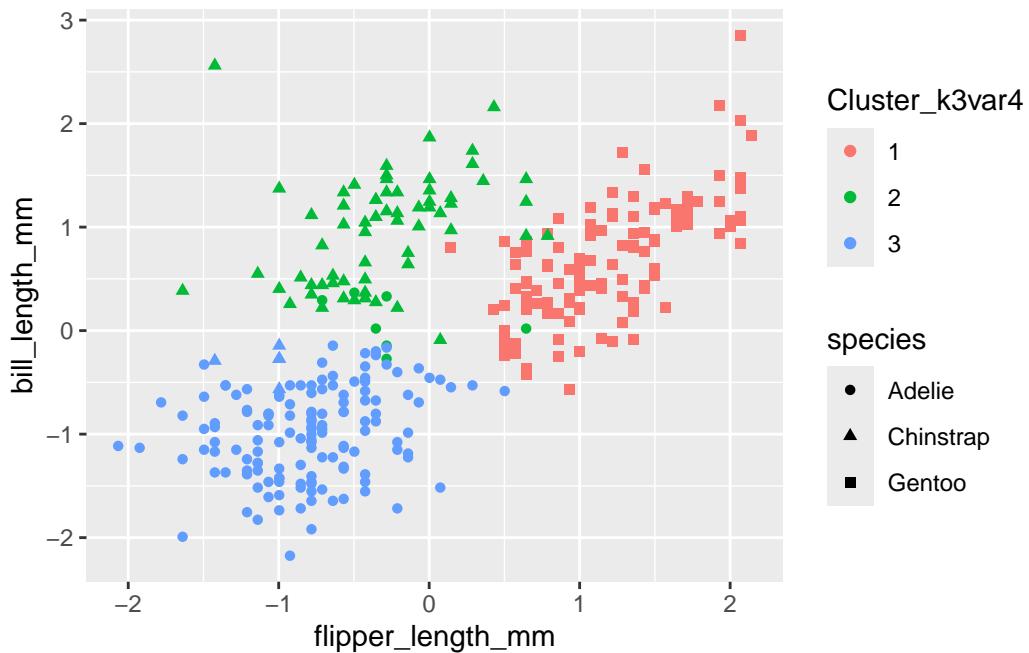
```
rawdata |>
  ggplot(aes(fill=species, x=Cluster_k3var4)) +
  geom_bar(position = 'fill') +
  scale_y_continuous(name = 'Frequency', labels=scales::percent)
```



```
rawdata |>
  ggplot(aes(flipper_length_mm,bill_length_mm,
             shape=species,color=Cluster_k3var4))+  
  geom_point()
```

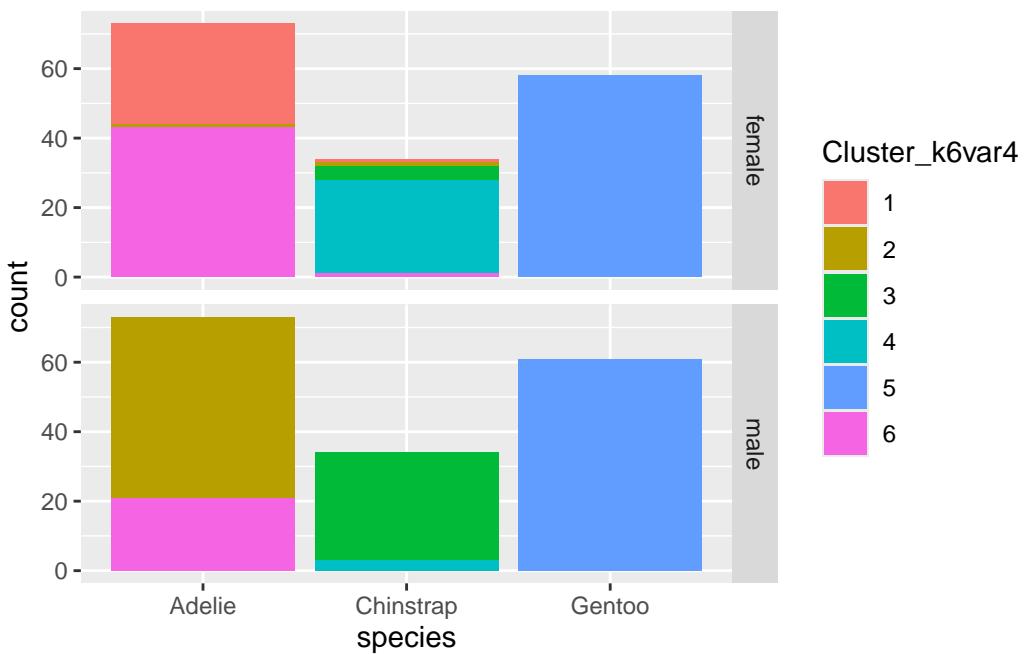


```
rawdata |>
  ggplot(aes(flipper_length_mm,bill_length_mm,
             shape=species,color=Cluster_k3var4))+  
  geom_point()
```

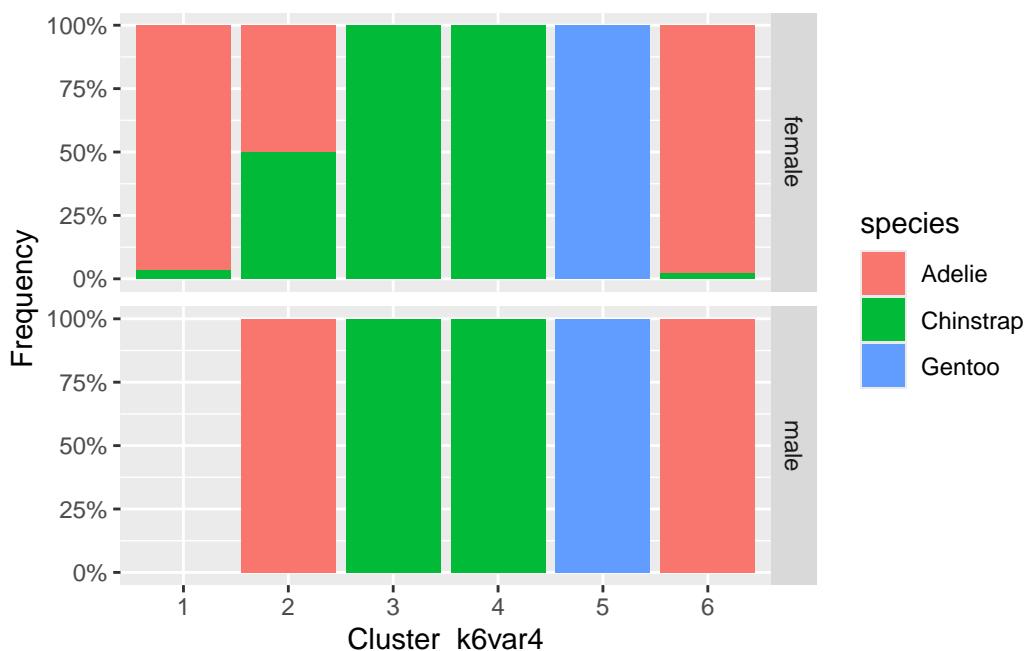


```
kmeans_out6 <- kmeans(rawdata |> select(predvars$names),
                        centers = 6)

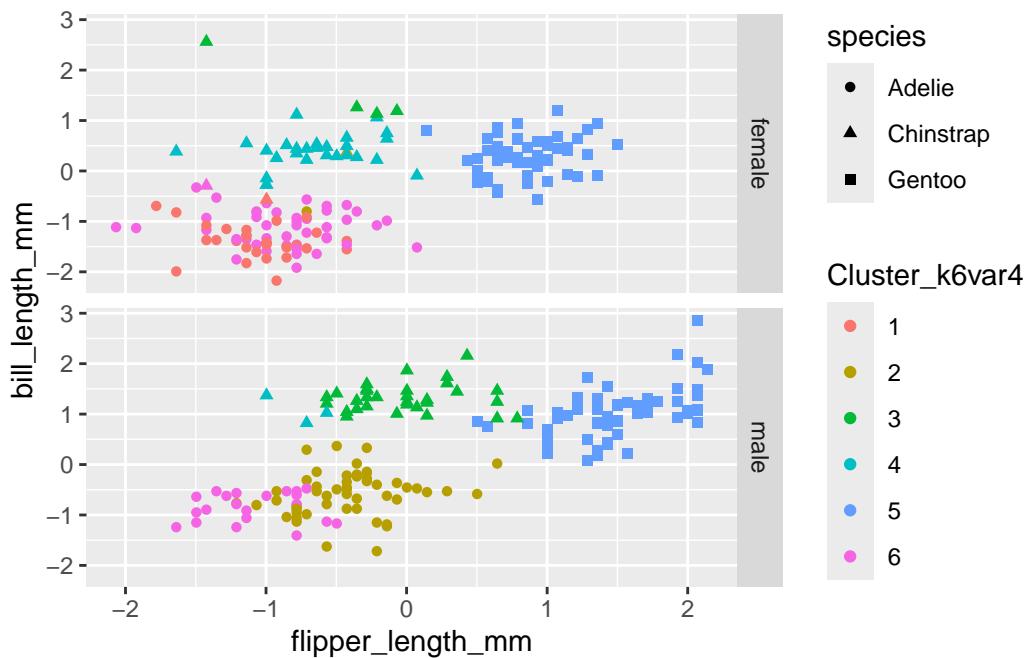
rawdata <-
  mutate(rawdata,Cluster_k6var4=kmeans_out6$cluster |> as.factor())
rawdata |>
  ggplot(aes(species,fill=Cluster_k6var4))+  
  geom_bar()+
  facet_grid(rows=vars(sex))
```



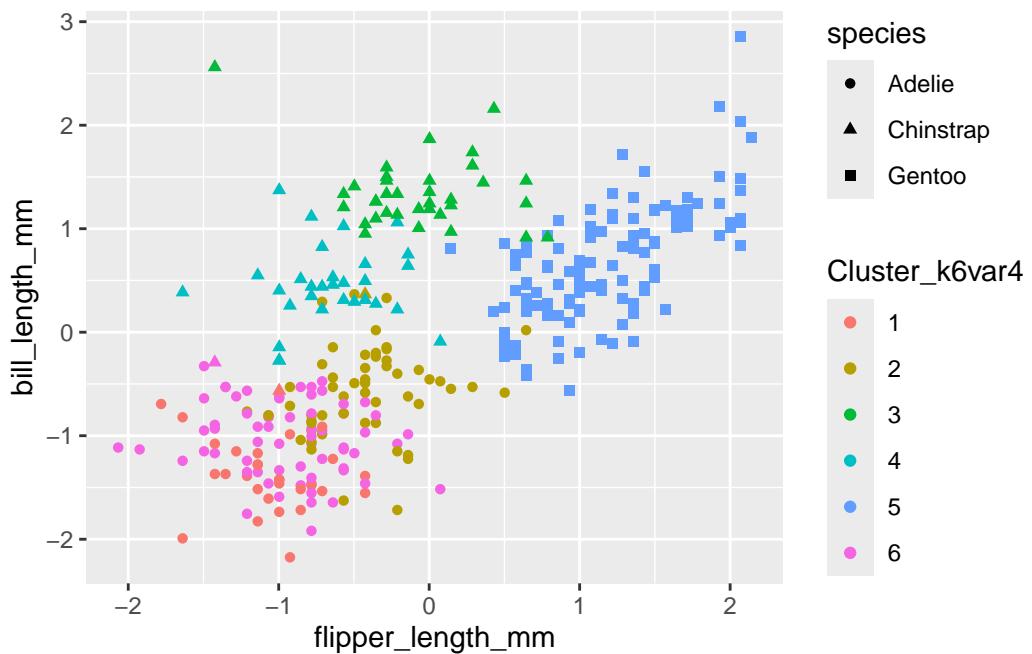
```
rawdata |>
  ggplot(aes(fill=species,x=Cluster_k6var4))+
  geom_bar(position = 'fill')+
  scale_y_continuous(name = 'Frequency', labels=scales::percent)+
```



```
rawdata |>
  ggplot(aes(flipper_length_mm,bill_length_mm,
             shape=species,color=Cluster_k6var4))+  
  geom_point() +  
  facet_grid(rows=vars(sex))
```



```
rawdata |>
  ggplot(aes(flipper_length_mm,bill_length_mm,
             shape=species,color=Cluster_k6var4))+  
  geom_point()
```



## 27.2 HClust

```
penguins_scaled <-
  rawdata |>
  select(predvars$names)
# Compute Distance Matrix
# Hierarchical clustering starts by calculating the distance between every
# pair of observations. The most common distance metric is Euclidean distance.

distance_matrix <- dist(penguins_scaled, method = "euclidean")

as.matrix(distance_matrix)[1:5, 1:5] # small portion of the matrix
```

	1	2	3	4	5
1	0.0000000	0.7564881	1.2481347	1.0757725	1.1661972
2	0.7564881	0.0000000	0.9965556	1.2772797	1.6607532
3	1.2481347	0.9965556	0.0000000	0.9753012	1.4665233
4	1.0757725	1.2772797	0.9753012	0.0000000	0.8771267
5	1.1661972	1.6607532	1.4665233	0.8771267	0.0000000

```
summary(distance_matrix)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.1089	1.5057	2.5986	2.5571	3.4774	7.2635

```
# Perform Hierarchical Clustering
#   `hclust()` from base R.
#   `method` parameter specifies the agglomeration method (linkage method).
#   Common methods: "ward.D2", "complete", "average", "single".
#   "ward.D2" is often preferred as it tends to produce more balanced clusters.

hc_result <- hclust(distance_matrix, method = "ward.D2")

hc_result
```

Call:

```
hclust(d = distance_matrix, method = "ward.D2")
```

```
Cluster method : ward.D2
Distance       : euclidean
Number of objects: 333
```

```
# Visualize the Dendrogram using factoextra
#   The dendrogram is the primary output of hierarchical clustering.
#   It shows how observations are grouped together.
#   `fviz_dend()` from `factoextra` provides a beautiful and easy way to plot it.

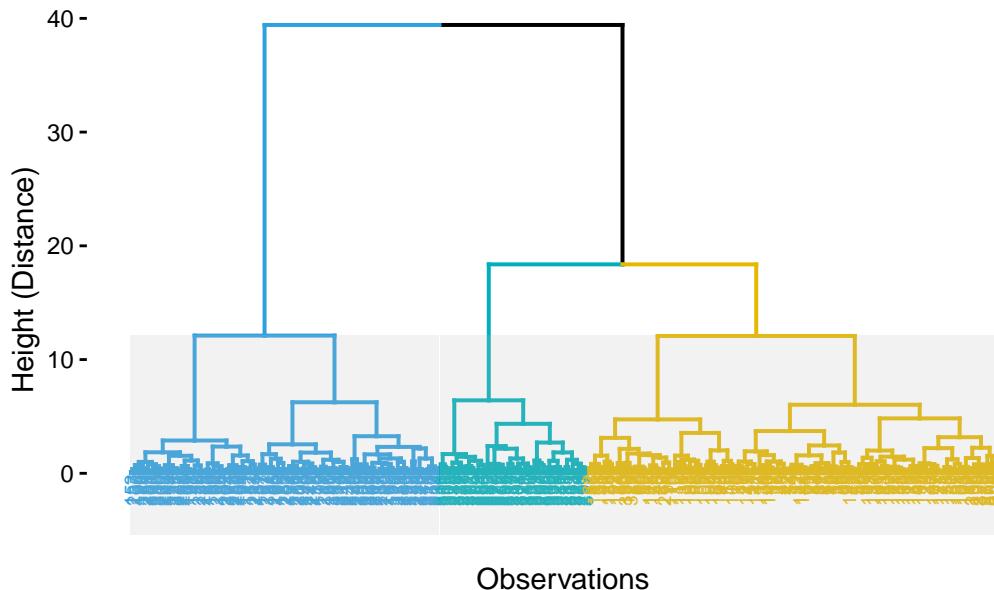
fviz_dend(hc_result,
          k = 3, # Cut the dendrogram into 3 groups (you can try other numbers)
          cex = 0.5, # Adjust label size
          k_colors = c("#2E9FDF", "#00AFBB", "#E7B800"), # Custom colors for clusters
          # color_labels_by_k = TRUE, # Color labels by group
          rect = TRUE, # Draw a rectangle around clusters
          # rect_border = c("#2E9FDF", "#00AFBB", "#E7B800"),
          rect_fill = TRUE,
          main = "Hierarchical Clustering Dendrogram (Ward's Method)",
          sub = "Penguins Data",
          xlab = "Observations",
          ylab = "Height (Distance)")
```

Warning: `aes\_string()` was deprecated in ggplot2 3.0.0.  
 i Please use tidy evaluation idioms with `aes()`.  
 i See also `vignette("ggplot2-in-packages")` for more information.  
 i The deprecated feature was likely used in the factoextra package.  
 Please report the issue at <<https://github.com/kassambara/factoextra/issues>>.

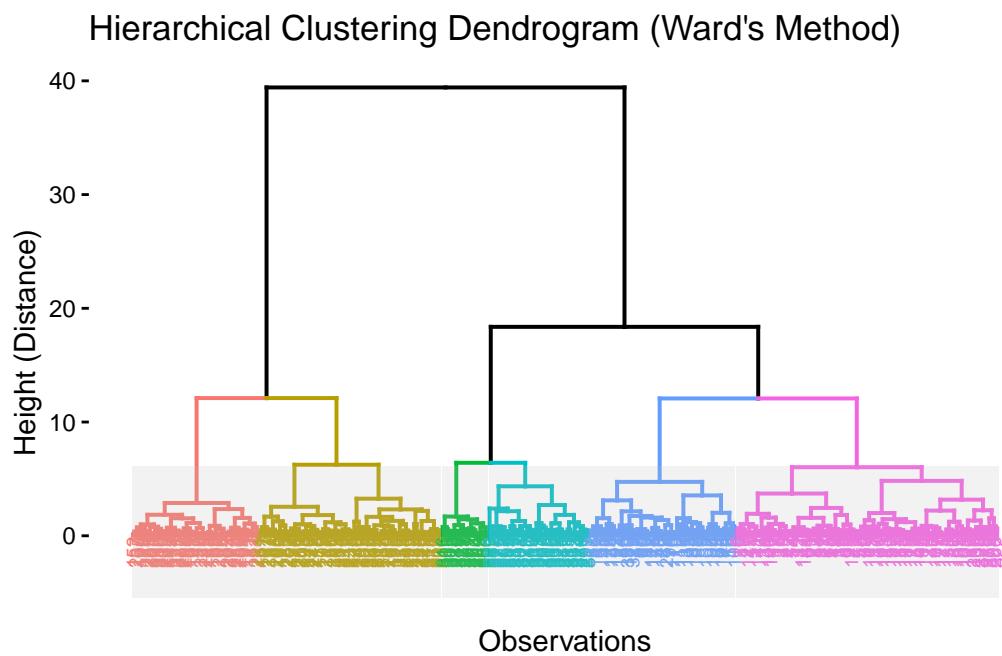
```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.  
i Please use `linewidth` instead.  
i The deprecated feature was likely used in the factoextra package.  
Please report the issue at <https://github.com/kassambara/factoextra/issues>.
```

```
Warning: The `<scale>` argument of `guides()` cannot be `FALSE`. Use "none" instead as  
of ggplot2 3.3.4.  
i The deprecated feature was likely used in the factoextra package.  
Please report the issue at <https://github.com/kassambara/factoextra/issues>.
```

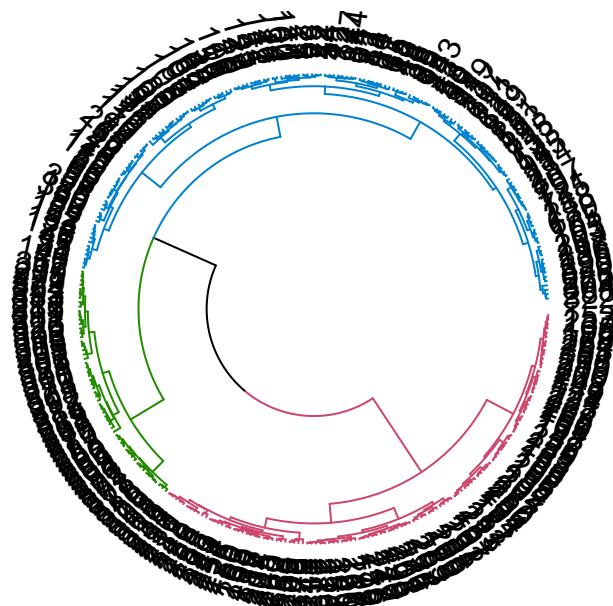
### Hierarchical Clustering Dendrogram (Ward's Method)



```
fviz_dend(hc_result,  
          k = 6, cex = 0.5,  
          rect = TRUE,  
          rect_fill = TRUE,  
          main = "Hierarchical Clustering Dendrogram (Ward's Method)",  
          sub = "Penguins Data", #ignored??  
          xlab = "Observations",  
          ylab = "Height (Distance)")
```



```
dend <- as.dendrogram(hc_result)
dend <- color_branches(dend, k=3)
dendextend::circlize_dendrogram(dend,
                                facing = "outside")
```



```

# Cut the Dendrogram and Extract Clusters
#   To form distinct clusters, you "cut" the dendrogram at a certain height
#   or specify the desired number of clusters (k).

num_clusters <- 3 # Let's aim for 3 clusters, similar to the 3 penguin species

# Cut tree into k groups
clusters <- cutree(hc_result, k = num_clusters)

sample(clusters,10)

```

[1] 1 2 1 2 1 2 1 2 1 1

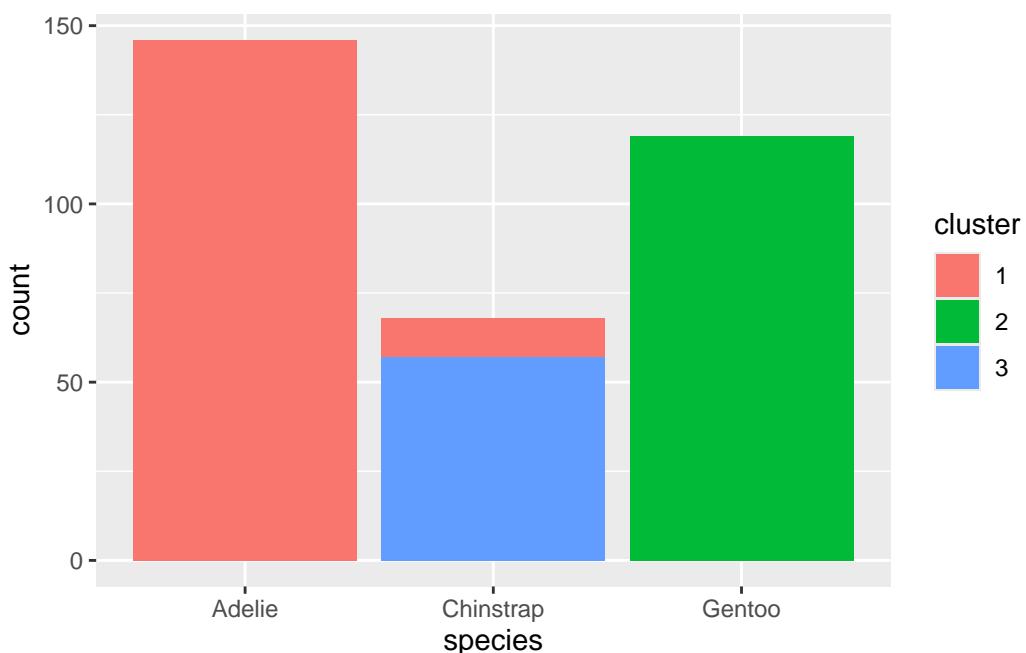
```

# Add cluster assignments back to the original (non-scaled) data for easier interpretation
rawdata <- rawdata |>
  mutate(cluster = as.factor(clusters))

rawdata |>
  ggplot(aes(species,fill=cluster))+  

  geom_bar()

```



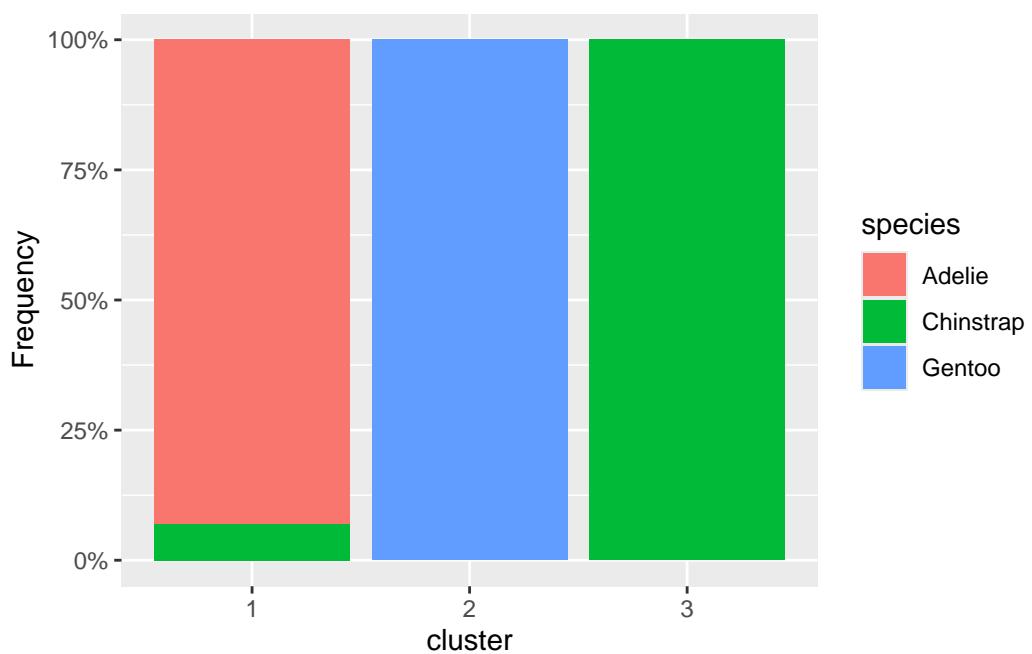
```

rawdata |>
  ggplot(aes(fill=species,x=cluster))+  

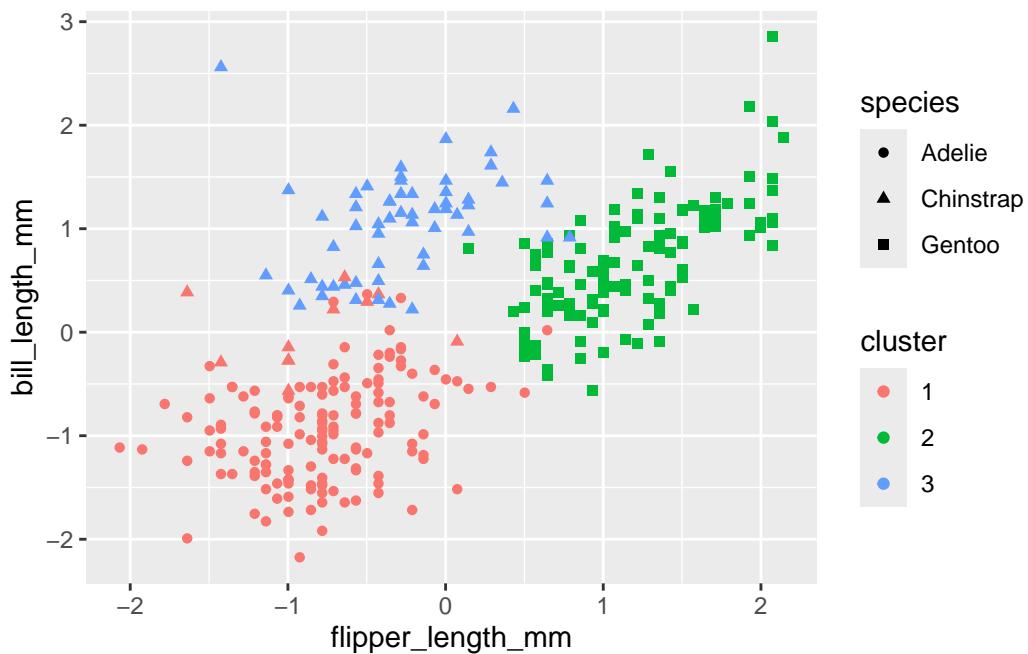
  geom_bar(position = 'fill')+

```

```
scale_y_continuous(name = 'Frequency', labels=scales::percent)
```



```
rawdata |>  
  ggplot(aes(flipper_length_mm,bill_length_mm,  
             shape=species,color=cluster))+  
  geom_point()
```



```
confusionMatrix(rawdata$cluster, reference = rawdata$species |>
  as.numeric() |> factor(levels=c(1,3,2),
  labels=c(1,2,3)))
```

#### Confusion Matrix and Statistics

		Reference		
Prediction	1	2	3	
1	146	0	11	
2	0	119	0	
3	0	0	57	

#### Overall Statistics

Accuracy : 0.967  
 95% CI : (0.9417, 0.9834)  
 No Information Rate : 0.4384  
 P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9476

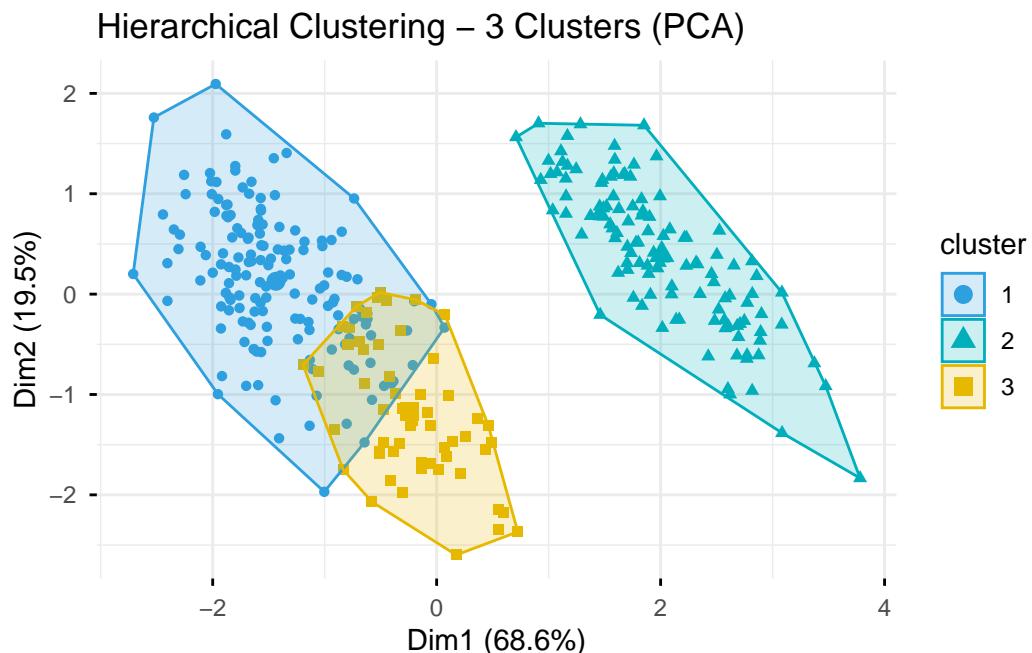
McNemar's Test P-Value : NA

#### Statistics by Class:

	Class: 1	Class: 2	Class: 3
Sensitivity	1.0000	1.0000	0.8382
Specificity	0.9412	1.0000	1.0000
Pos Pred Value	0.9299	1.0000	1.0000
Neg Pred Value	1.0000	1.0000	0.9601
Prevalence	0.4384	0.3574	0.2042
Detection Rate	0.4384	0.3574	0.1712
Detection Prevalence	0.4715	0.3574	0.1712
Balanced Accuracy	0.9706	1.0000	0.9191

```
# Visualize the Clusters on a Scatter Plot using factoextra
#   `fviz_cluster()` helps to visualize the clusters formed on a scatter plot
#   (using PCA for dimensionality reduction if data has > 2 dimensions).

fviz_cluster(list(data = penguins_scaled, cluster = clusters),
             geom = "point",
             ellipse.type = "convex", # Draw convex hulls around clusters
             palette = c("#2E9FDF", "#00AFBB", "#E7B800"), # Use same colors as dendrograms
             main = paste0("Hierarchical Clustering - ", num_clusters, " Clusters (PCA)",
             # xlab = "Principal Component 1",
             # ylab = "Principal Component 2",
             ggtheme = theme_minimal())
```



## 27.3 Unified from easystats

```
easycluster1 <- cluster_analysis(penguins_scaled,  
                                  standardize = FALSE)
```

Using solution with 2 clusters, supported by 10 out of 28 methods.

```
print(easycluster1)
```

# Clustering Solution

The 2 clusters accounted for 58.51% of the total variance of the original data.

Cluster	n_Obs	Sum_Squares	bill_length_mm	bill_depth_mm
1	119	139.47	0.65	-1.10
2	214	411.54	-0.36	0.61

Cluster	flipper_length_mm	body_mass_g
1	1.16	1.10
2	-0.65	-0.61

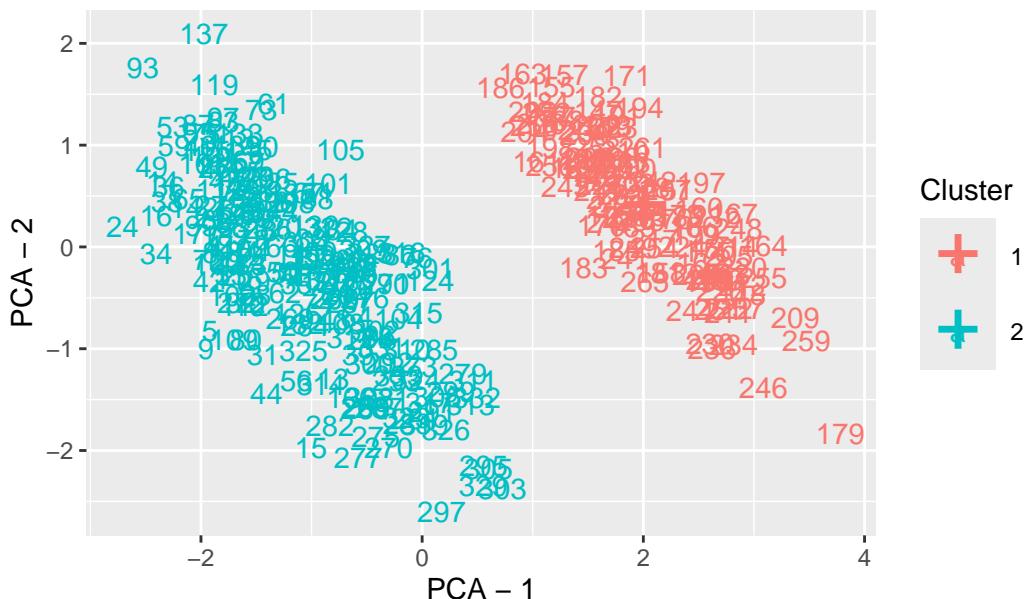
# Indices of model performance

Sum_Squares_Total	Sum_Squares_Between	Sum_Squares_Within	R2
1328	776.989	551.011	0.585

# You can access the predicted clusters via `predict()`.

```
plot(easycluster1)
```

## Clustering Solution



```
easycluster2 <- cluster_analysis(penguins_scaled,
                                  standardize = FALSE,
                                  method = "hclust",
                                  n = 6)
print(easycluster2)
```

# Clustering Solution

The 6 clusters accounted for 83.82% of the total variance of the original data.

Cluster	n_Obs	Sum_Squares	bill_length_mm	bill_depth_mm
1	97	65.06	-1.11	0.29
2	54	39.63	-0.58	1.10
3	51	39.55	0.87	0.55
4	49	16.73	0.21	-1.56
5	70	49.43	0.96	-0.78
6	12	4.50	1.40	1.36

Cluster | flipper\_length\_mm | body\_mass\_g

Cluster	flipper_length_mm	body_mass_g
1	-0.99	-0.96
2	-0.43	-0.05
3	-0.47	-0.69
4	0.77	0.53

```

5      |      1.43 |      1.50
6      |      0.37 |      0.01

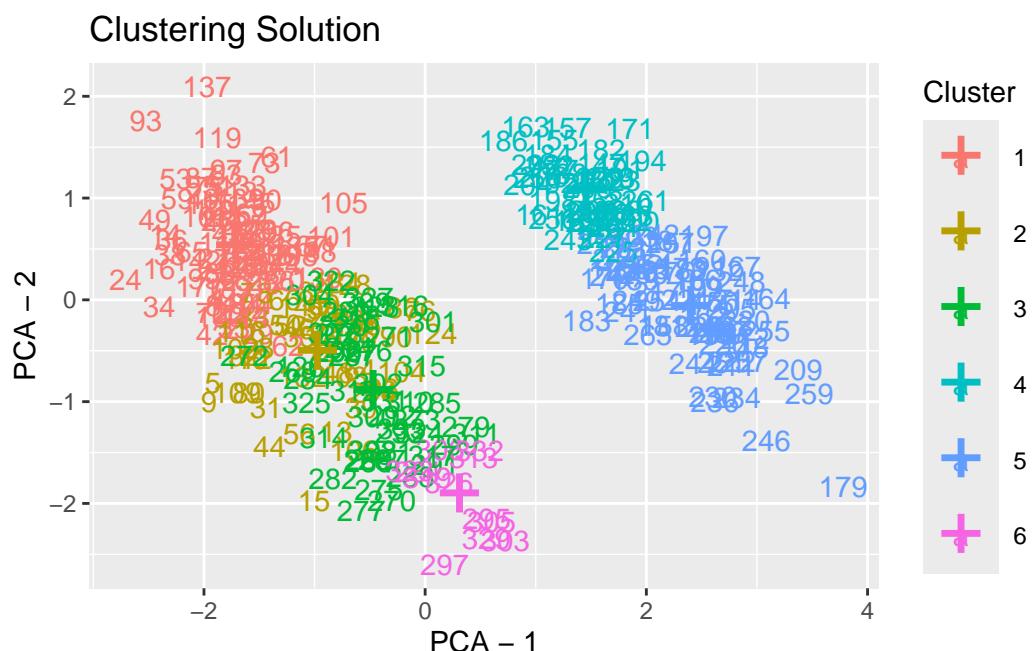
# Indices of model performance

Sum_Squares_Total | Sum_Squares_Between | Sum_Squares_Within |      R2
-----
1328             |      1113.100 |      214.900 |  0.838

# You can access the predicted clusters via `predict()`.


```

```
plot(easycluster2)
```



## 28 caret package for machine learning

The caret (**C**lassification **A**nd **R**Egression **T**raining) package is a suite of functions designed to streamline the process of model training and tuning for classification and regression problems. It provides a standardized interface that abstracts away the complexities of using numerous individual R packages for machine learning, making it easier to compare and select optimal models.

Key Functionality:

- **Unified Interface:** Offers a consistent workflow using the primary function `train()` to fit, tune, and evaluate over 250 different machine learning models, regardless of the underlying R package.
- **Model Preprocessing:** Includes tools for common data preprocessing steps like centering, scaling, principal component analysis (PCA), and feature selection.
- **Resampling Methods:** Facilitates various resampling techniques such as cross-validation, bootstrap, and subsampling for robust model performance estimation.
- **Hyperparameter Tuning:** Automates the tuning of model hyperparameters over a grid of possible values to find the combination that maximizes performance.
- **Performance Evaluation:** Provides functions for calculating and visualizing performance metrics (e.g., accuracy, RMSE, ROC curves) and comparing multiple models.

```
# remove.packages("xgboost")
#
# install.packages(
#   "https://cran.r-project.org/src/contrib/Archive/xgboost/xgboost_1.7.11.1.tar.gz",
#   repos = NULL,
#   type = "source"
# )

pacman::p_load(conflicted,
  tidyverse, broom,
  wrappedtools, flextable,
  palmerpenguins, tictoc,
  ggfortify, GGally, nnet,
  caret, randomForest, kernlab, naivebayes,
  mlbench,
  doParallel, future, doFuture, future.mirai)
```

```

# conflict_scout()
conflicts_prefer(
  dplyr::filter,
  ggplot2::alpha,
  dplyr::combine,
  dplyr::slice,
  palmerpenguins::penguins,
  caret::lift,
  .quiet = TRUE)
rawdata <- penguins |>
  na.omit()
predvars <- ColSeeker(namepattern = c('_mm','_g'))
rawdata <- select(rawdata,
                  species, predvars$names)

```

## 28.1 Parallel computing setup

```

# Determine the number of cores to use (it's common to leave one core free for the operating system)
cores <- parallel::detectCores() - 1
if(cores < 1) {
  cores <- 1 # Ensure at least one core is used
}
# Create the cluster
# makePSOCKcluster works on all operating systems (Windows, macOS, Linux)
cl <- makePSOCKcluster(cores)

# Register the cluster as the parallel backend for the 'foreach' package (which caret uses)
registerDoParallel(cl)

# Optional: Check how many workers are registered
getDoParWorkers()

```

[1] 21

## 28.2 Define global modelling options

This will be applied to all models. Here we use 5-fold cross-validation repeated 20 times. Thus 100 models will be trained for each tuning parameter combination.

```
ctrl <- trainControl(method = "repeatedcv",
                      number=5,
                      repeats = 20)
```

Method-specific options will be defined below for each model (**tune**).

## 28.3 Model tuning and training

### 28.3.1 K-Nearest Neighbors (KNN)

```
tune <- expand.grid(k=seq(1,9,2))
knnfit <- train(form = species~.,
                 data = rawdata,
                 preProcess = c('center','scale'),
                 method='knn',
                 metric='Accuracy',
                 trControl = ctrl,
                 tuneGrid=tune)
```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-1') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-2') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-3') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-4') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-5') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-6') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-7') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-8') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-9') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-10') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead

of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-11') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-12') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-13') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-14') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-15') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-16') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-17') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-18') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-19') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-20') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-21') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead

of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

### 28.3.2 Extreme Gradient Boosting (XGBoost)

```
tune <- expand.grid(nrounds=c(25,50,75),
                      max_depth=seq(2,10,2),
                      eta=1,
                      gamma=c(.01,.005),
                      colsample_bytree=1,
                      min_child_weight=1,
                      subsample=1)
tic toc::tic("xgb")
xgbfit <- train(form = species~.,
                  data = rawdata,
                  # preProcess = c('center','scale'),
                  method='xgbTree',
                  # objective = "multi:softprob", # set by train
                  metric='Accuracy',
                  trControl = ctrl,
                  tuneGrid=tune,
                  verbosity = 0)
```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-1') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-2') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-3') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results

might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-4') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-5') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-6') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-7') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-8') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-9') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-10') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-11') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-12') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-13') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-14') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead

of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-15') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-16') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-17') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-18') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-19') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-20') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-21') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```
tictoc::toc()
```

```
xgb: 43.66 sec elapsed
```

### 28.3.3 Random Forest (RF)

```
tune <- expand.grid(mtry=seq(2,3,1))
rffit <- train(form = species~.,
               data = rawdata,
               # preProcess = c('center','scale'),
               method='rf',
               metric='Accuracy',
               trControl = ctrl,
               tuneGrid=tune)
```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-1') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-2') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-3') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-4') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-5') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-6') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-7') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-8') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-9') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-10') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead

of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-11') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-12') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-13') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-14') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-15') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-16') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-17') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-18') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-19') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-20') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-21') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead

of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

#### 28.3.4 Linear Discriminant Analysis (LDA)

```
ldafit <- train(form = species~.,
                 data = rawdata,
                 preProcess = c('center','scale'),
                 method='lda',
                 metric='Accuracy',
                 trControl = ctrl)
```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-1') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-2') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-3') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-4') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-5') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-6') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-7') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-8') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-9') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-10') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead

of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-11') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-12') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-13') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-14') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-15') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-16') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-17') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-18') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-19') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-20') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-21') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead

of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

### 28.3.5 Support Vector Machine (SVM)

```
svmfit <- train(form = species~.,
                 data = rawdata,
                 preProcess = c('center','scale'),
                 method='svmLinear',
                 metric='Accuracy',
                 trControl = ctrl)
```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-1') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-2') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-3') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-4') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-5') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-6') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-7') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-8') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-9') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-10') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead

of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-11') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-12') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-13') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-14') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-15') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-16') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-17') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-18') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-19') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-20') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-21') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead

of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

### 28.3.6 Naive Bayes

```
bayesfit <- train(form = species~.,
                    data = rawdata,
                    preProcess = c('center','scale'),
                    method='naive_bayes',
                    metric='Accuracy',
                    trControl = ctrl)
```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-1') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-2') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-3') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-4') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-5') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-6') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-7') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-8') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-9') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-10') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead

of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-11') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-12') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-13') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-14') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-15') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-16') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-17') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-18') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-19') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-20') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-21') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead

of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

### 28.3.7 Neural Network (NN)

```
nnfit <- train(form=species~.,
  data=rawdata,
  method="nnet",
  preProcess=c("center","scale"),
  metric="Accuracy",
  trControl=ctrl)

# weights:  11
initial  value 305.132124
iter   10 value 114.145886
iter   20 value 45.541772
iter   30 value 26.581353
iter   40 value 20.572057
iter   50 value 20.139763
iter   60 value 19.648224
iter   70 value 19.108721
iter   80 value 18.792423
iter   90 value 18.643616
iter  100 value 18.597919
final  value 18.597919
stopped after 100 iterations
# weights:  27
initial  value 313.042776
iter   10 value 7.721130
iter   20 value 0.262286
iter   30 value 0.007601
iter   40 value 0.000448
iter   50 value 0.000270
iter   60 value 0.000171
final  value 0.000074
converged
# weights:  43
initial  value 284.561345
iter   10 value 2.884019
iter   20 value 0.041822
iter   30 value 0.000117
iter   30 value 0.000063
iter   30 value 0.000062
final  value 0.000062
converged
# weights:  11
initial  value 310.069286
```

```

iter 10 value 122.920670
iter 20 value 81.561150
iter 30 value 77.363512
final value 77.089708
converged
# weights: 27
initial value 324.524681
iter 10 value 27.964630
iter 20 value 22.783226
iter 30 value 22.147708
iter 40 value 22.129802
iter 50 value 22.128010
final value 22.128010
converged
# weights: 43
initial value 288.799369
iter 10 value 51.863905
iter 20 value 18.373155
iter 30 value 17.324394
iter 40 value 17.272987
iter 50 value 17.268533
iter 60 value 17.268169
iter 70 value 17.267981
final value 17.267981
converged
# weights: 11
initial value 292.358267
iter 10 value 113.608940
iter 20 value 106.415187
iter 30 value 99.730755
iter 40 value 88.974846
iter 50 value 58.782962
iter 60 value 23.443072
iter 70 value 20.834564
iter 80 value 19.866617
iter 90 value 19.839856
iter 100 value 19.765648
final value 19.765648
stopped after 100 iterations
# weights: 27
initial value 311.517712
iter 10 value 13.840731
iter 20 value 0.512587
iter 30 value 0.409904
iter 40 value 0.381643
iter 50 value 0.339435

```

```

iter 60 value 0.308446
iter 70 value 0.268872
iter 80 value 0.252715
iter 90 value 0.224319
iter 100 value 0.209697
final value 0.209697
stopped after 100 iterations
# weights: 43
initial value 321.326155
iter 10 value 15.430039
iter 20 value 0.795509
iter 30 value 0.259395
iter 40 value 0.248959
iter 50 value 0.223615
iter 60 value 0.202378
iter 70 value 0.190648
iter 80 value 0.177968
iter 90 value 0.168068
iter 100 value 0.165505
final value 0.165505
stopped after 100 iterations
# weights: 11
initial value 291.710991
iter 10 value 106.875285
iter 20 value 85.754943
iter 30 value 30.321701
iter 40 value 25.738198
iter 50 value 25.513893
iter 60 value 25.282357
iter 70 value 25.116663
iter 80 value 25.107095
iter 90 value 25.001942
iter 100 value 24.970545
final value 24.970545
stopped after 100 iterations
# weights: 27
initial value 303.802486
iter 10 value 1.144187
iter 20 value 0.009168
iter 30 value 0.001647
iter 40 value 0.000802
final value 0.000098
converged
# weights: 43
initial value 276.098821
iter 10 value 9.239369

```

```

iter 20 value 0.101791
iter 30 value 0.006727
iter 40 value 0.000906
iter 50 value 0.000165
final value 0.000065
converged
# weights: 11
initial value 276.086039
iter 10 value 114.964701
iter 20 value 81.555931
iter 30 value 78.036271
final value 77.882065
converged
# weights: 27
initial value 382.400585
iter 10 value 35.962525
iter 20 value 18.633619
iter 30 value 18.210040
iter 40 value 18.203881
iter 50 value 18.202507
iter 50 value 18.202507
iter 50 value 18.202507
final value 18.202507
converged
# weights: 43
initial value 335.446538
iter 10 value 50.850949
iter 20 value 18.900917
iter 30 value 17.071625
iter 40 value 16.703454
iter 50 value 16.677793
iter 60 value 16.466133
iter 70 value 16.382296
iter 80 value 16.376820
iter 90 value 16.376596
iter 90 value 16.376596
iter 90 value 16.376596
final value 16.376596
converged
# weights: 11
initial value 298.598433
iter 10 value 112.949788
iter 20 value 101.591620
iter 30 value 100.367965
iter 40 value 33.897956
iter 50 value 26.685973

```

```

iter 60 value 26.059217
iter 70 value 25.711393
iter 80 value 25.705095
iter 90 value 25.688019
final value 25.687844
converged
# weights: 27
initial value 274.613661
iter 10 value 8.455395
iter 20 value 0.603461
iter 30 value 0.208570
iter 40 value 0.165805
iter 50 value 0.157378
iter 60 value 0.146243
iter 70 value 0.140907
iter 80 value 0.125808
iter 90 value 0.108790
iter 100 value 0.098505
final value 0.098505
stopped after 100 iterations
# weights: 43
initial value 338.867107
iter 10 value 1.391672
iter 20 value 0.175890
iter 30 value 0.119445
iter 40 value 0.110310
iter 50 value 0.104368
iter 60 value 0.099462
iter 70 value 0.097281
iter 80 value 0.095972
iter 90 value 0.094713
iter 100 value 0.094025
final value 0.094025
stopped after 100 iterations
# weights: 11
initial value 320.194579
iter 10 value 110.898621
iter 20 value 106.649410
final value 106.645257
converged
# weights: 27
initial value 345.536660
iter 10 value 13.280399
iter 20 value 1.184414
iter 30 value 0.027434
iter 40 value 0.001239

```

```
iter 50 value 0.000250
iter 60 value 0.000105
final value 0.000098
converged
# weights: 43
initial value 305.389422
iter 10 value 3.228955
iter 20 value 0.309085
iter 30 value 0.007873
iter 40 value 0.002950
iter 50 value 0.000958
iter 60 value 0.000541
iter 70 value 0.000154
iter 80 value 0.000143
final value 0.000093
converged
# weights: 11
initial value 314.206025
iter 10 value 111.705898
iter 20 value 76.792805
iter 30 value 74.394453
final value 74.247799
converged
# weights: 27
initial value 297.159774
iter 10 value 44.298891
iter 20 value 23.299405
iter 30 value 21.219294
iter 40 value 20.962578
iter 50 value 20.851198
iter 60 value 20.578052
iter 70 value 20.412526
final value 20.412001
converged
# weights: 43
initial value 276.612057
iter 10 value 28.345256
iter 20 value 17.959476
iter 30 value 16.542483
iter 40 value 16.512186
iter 50 value 16.511961
final value 16.511916
converged
# weights: 11
initial value 302.789448
iter 10 value 111.968744
```

```

iter 20 value 106.396860
iter 30 value 105.633674
iter 40 value 104.092451
iter 50 value 101.627309
iter 60 value 99.505872
iter 70 value 91.881969
iter 80 value 71.425527
iter 90 value 40.010252
iter 100 value 38.720911
final value 38.720911
stopped after 100 iterations
# weights: 27
initial value 307.081725
iter 10 value 10.466059
iter 20 value 0.888278
iter 30 value 0.275806
iter 40 value 0.250074
iter 50 value 0.228331
iter 60 value 0.209709
iter 70 value 0.191959
iter 80 value 0.182536
iter 90 value 0.174730
iter 100 value 0.164762
final value 0.164762
stopped after 100 iterations
# weights: 43
initial value 327.908569
iter 10 value 4.875740
iter 20 value 0.472449
iter 30 value 0.345287
iter 40 value 0.324253
iter 50 value 0.273331
iter 60 value 0.223515
iter 70 value 0.178359
iter 80 value 0.154835
iter 90 value 0.144172
iter 100 value 0.136404
final value 0.136404
stopped after 100 iterations
# weights: 11
initial value 304.303682
iter 10 value 123.793324
iter 20 value 109.576628
iter 30 value 107.186776
iter 40 value 101.529867
iter 50 value 98.176788

```

```
iter 60 value 97.045565
iter 70 value 96.842594
iter 80 value 96.808853
iter 90 value 96.794563
iter 100 value 96.715917
final value 96.715917
stopped after 100 iterations
# weights: 27
initial value 313.495506
iter 10 value 8.880498
iter 20 value 0.081302
iter 30 value 0.000138
iter 30 value 0.000077
iter 30 value 0.000077
final value 0.000077
converged
# weights: 43
initial value 291.820582
iter 10 value 1.887341
iter 20 value 0.293458
iter 30 value 0.010955
iter 40 value 0.002793
iter 50 value 0.000845
final value 0.000093
converged
# weights: 11
initial value 317.152983
iter 10 value 172.752992
iter 20 value 104.721997
iter 30 value 102.284037
final value 102.284004
converged
# weights: 27
initial value 330.094109
iter 10 value 46.289510
iter 20 value 21.295497
iter 30 value 19.914237
iter 40 value 19.532910
iter 50 value 19.322390
iter 60 value 19.300039
iter 70 value 19.298994
iter 70 value 19.298994
iter 70 value 19.298994
final value 19.298994
converged
# weights: 43
```

```

initial value 386.578108
iter 10 value 42.472560
iter 20 value 20.096715
iter 30 value 18.303604
iter 40 value 17.673754
iter 50 value 17.443777
iter 60 value 17.363013
iter 70 value 17.346544
iter 80 value 17.343647
iter 90 value 17.342688
iter 100 value 17.342607
final value 17.342607
stopped after 100 iterations
# weights: 11
initial value 325.301266
iter 10 value 109.081864
iter 20 value 107.583921
iter 30 value 107.363975
iter 40 value 107.291879
iter 50 value 107.278849
iter 60 value 106.859950
iter 70 value 106.282139
iter 80 value 105.936830
iter 90 value 105.764284
iter 100 value 104.979530
final value 104.979530
stopped after 100 iterations
# weights: 27
initial value 273.762081
iter 10 value 20.914858
iter 20 value 1.074560
iter 30 value 0.432530
iter 40 value 0.287841
iter 50 value 0.260621
iter 60 value 0.237981
iter 70 value 0.227059
iter 80 value 0.219934
iter 90 value 0.213408
iter 100 value 0.204265
final value 0.204265
stopped after 100 iterations
# weights: 43
initial value 366.653313
iter 10 value 20.573246
iter 20 value 0.750358
iter 30 value 0.596959

```

```
iter 40 value 0.501118
iter 50 value 0.427585
iter 60 value 0.365062
iter 70 value 0.271066
iter 80 value 0.240873
iter 90 value 0.224381
iter 100 value 0.209447
final value 0.209447
stopped after 100 iterations
# weights: 11
initial value 295.455560
iter 10 value 49.974321
iter 20 value 30.574192
iter 30 value 28.913699
iter 40 value 28.167411
iter 50 value 28.078476
iter 60 value 27.914358
iter 70 value 27.884195
iter 80 value 27.864305
iter 90 value 27.841950
iter 100 value 27.792257
final value 27.792257
stopped after 100 iterations
# weights: 27
initial value 335.353589
iter 10 value 6.969237
iter 20 value 0.137701
iter 30 value 0.000488
final value 0.000053
converged
# weights: 43
initial value 321.961631
iter 10 value 6.641759
iter 20 value 0.029818
iter 30 value 0.004318
iter 40 value 0.001090
iter 50 value 0.000281
iter 60 value 0.000115
iter 60 value 0.000079
iter 60 value 0.000079
final value 0.000079
converged
# weights: 11
initial value 272.692482
iter 10 value 122.131164
iter 20 value 118.007387
```

```

iter 30 value 105.379456
iter 40 value 101.669068
final value 101.669067
converged
# weights: 27
initial value 333.585394
iter 10 value 33.137580
iter 20 value 20.189866
iter 30 value 19.558152
iter 40 value 19.457675
iter 50 value 19.448544
final value 19.448544
converged
# weights: 43
initial value 310.091035
iter 10 value 32.524923
iter 20 value 17.082255
iter 30 value 16.829524
iter 40 value 16.808245
iter 50 value 16.807360
iter 50 value 16.807360
iter 50 value 16.807360
final value 16.807360
converged

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-1') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```

# weights: 11
initial value 311.017120
iter 10 value 91.872919
iter 20 value 34.125636
iter 30 value 29.193452
iter 40 value 28.625189
iter 50 value 28.571376
iter 60 value 28.549426
iter 70 value 28.499020
final value 28.499019
converged
# weights: 27

```

```
initial value 313.529159
iter 10 value 85.151334
iter 20 value 12.929137
iter 30 value 7.148466
iter 40 value 6.620483
iter 50 value 6.179888
iter 60 value 5.998793
iter 70 value 5.876916
iter 80 value 5.271240
iter 90 value 0.885592
iter 100 value 0.799869
final value 0.799869
stopped after 100 iterations
# weights: 43
initial value 268.175254
iter 10 value 2.919546
iter 20 value 0.124405
iter 30 value 0.115279
iter 40 value 0.104497
iter 50 value 0.096990
iter 60 value 0.091071
iter 70 value 0.086121
iter 80 value 0.083858
iter 90 value 0.082345
iter 100 value 0.082039
final value 0.082039
stopped after 100 iterations
# weights: 11
initial value 309.366018
iter 10 value 161.399058
iter 20 value 112.611047
iter 30 value 108.256201
iter 40 value 108.191626
iter 50 value 108.138840
iter 60 value 107.975473
iter 70 value 107.779290
iter 80 value 107.703283
iter 90 value 107.464559
iter 100 value 106.722147
final value 106.722147
stopped after 100 iterations
# weights: 27
initial value 338.859081
iter 10 value 3.019996
iter 20 value 0.012383
iter 30 value 0.000556
```

```
final value 0.000061
converged
# weights: 43
initial value 356.198627
iter 10 value 77.861379
iter 20 value 14.726131
iter 30 value 8.715566
iter 40 value 4.189605
iter 50 value 4.159047
iter 60 value 4.154594
iter 70 value 4.120858
iter 80 value 4.105867
iter 90 value 4.055503
iter 100 value 4.022165
final value 4.022165
stopped after 100 iterations
# weights: 11
initial value 326.149525
iter 10 value 129.922177
iter 20 value 107.162959
iter 30 value 78.440636
final value 78.229093
converged
# weights: 27
initial value 339.819980
iter 10 value 37.137700
iter 20 value 21.732516
iter 30 value 21.590809
iter 40 value 21.589492
iter 50 value 21.589377
iter 60 value 21.586307
iter 70 value 21.395792
iter 80 value 21.112093
final value 21.111999
converged
# weights: 43
initial value 322.893328
iter 10 value 27.914523
iter 20 value 19.590089
iter 30 value 17.762553
iter 40 value 17.599153
iter 50 value 17.589237
iter 60 value 17.561169
iter 70 value 17.224473
iter 80 value 17.206366
iter 90 value 17.199539
```

```
iter 100 value 17.197615
final  value 17.197615
stopped after 100 iterations
# weights: 11
initial  value 268.945214
iter   10 value 105.962175
iter   20 value 98.855571
iter   30 value 93.679784
iter   40 value 34.241480
iter   50 value 27.425014
iter   60 value 26.796711
iter   70 value 26.447637
iter   80 value 26.342627
iter   90 value 26.341859
final  value 26.341513
converged
# weights: 27
initial  value 337.241727
iter   10 value 40.991452
iter   20 value 1.536344
iter   30 value 0.292361
iter   40 value 0.258121
iter   50 value 0.232877
iter   60 value 0.225889
iter   70 value 0.216995
iter   80 value 0.212656
iter   90 value 0.208169
iter 100 value 0.202035
final  value 0.202035
stopped after 100 iterations
# weights: 43
initial  value 293.430112
iter   10 value 2.577552
iter   20 value 0.337673
iter   30 value 0.282133
iter   40 value 0.255365
iter   50 value 0.240071
iter   60 value 0.218463
iter   70 value 0.200782
iter   80 value 0.183535
iter   90 value 0.172383
iter 100 value 0.166332
final  value 0.166332
stopped after 100 iterations
# weights: 11
initial  value 307.473740
```

```

iter 10 value 136.547434
iter 20 value 83.440169
iter 30 value 46.120444
iter 40 value 28.261787
iter 50 value 27.678610
iter 60 value 27.522217
iter 70 value 27.330353
iter 80 value 27.322847
iter 90 value 27.246823
iter 100 value 27.209534
final value 27.209534
stopped after 100 iterations
# weights: 27
initial value 326.456669
iter 10 value 1.770229
iter 20 value 0.033245
iter 30 value 0.000244
final value 0.000077
converged
# weights: 43
initial value 378.740286
iter 10 value 3.109695
iter 20 value 0.029555
iter 30 value 0.002579
iter 40 value 0.000352
iter 50 value 0.000191
iter 50 value 0.000091
iter 50 value 0.000091
final value 0.000091
converged
# weights: 11
initial value 289.218561
iter 10 value 84.593635
iter 20 value 77.259069
iter 30 value 77.219776
final value 77.218850
converged
# weights: 27
initial value 431.531653
iter 10 value 76.127526
iter 20 value 23.338519
iter 30 value 18.911866
iter 40 value 18.646638
iter 50 value 18.620592
final value 18.619983
converged

```

```
# weights: 43
initial value 261.214268
iter 10 value 30.205062
iter 20 value 17.937725
iter 30 value 17.503979
iter 40 value 17.265841
iter 50 value 16.883790
iter 60 value 16.850709
iter 70 value 16.850264
final value 16.850260
converged
# weights: 11
initial value 318.174441
iter 10 value 126.804064
iter 20 value 35.678495
iter 30 value 30.029404
iter 40 value 28.001314
iter 50 value 27.741512
iter 60 value 27.714553
iter 70 value 27.709379
iter 70 value 27.709379
final value 27.709379
converged
# weights: 27
initial value 326.304056
iter 10 value 4.228152
iter 20 value 0.284563
iter 30 value 0.262899
iter 40 value 0.230931
iter 50 value 0.188617
iter 60 value 0.160641
iter 70 value 0.131198
iter 80 value 0.121337
iter 90 value 0.107686
iter 100 value 0.105419
final value 0.105419
stopped after 100 iterations
# weights: 43
initial value 382.656492
iter 10 value 11.639077
iter 20 value 0.607609
iter 30 value 0.298247
iter 40 value 0.266387
iter 50 value 0.210785
iter 60 value 0.158525
iter 70 value 0.141243
```

```

iter 80 value 0.124965
iter 90 value 0.118705
iter 100 value 0.114963
final value 0.114963
stopped after 100 iterations
# weights: 11
initial value 293.943528
iter 10 value 109.874841
iter 20 value 100.096746
iter 30 value 60.900758
iter 40 value 42.052485
iter 50 value 40.711892
iter 60 value 40.567652
iter 70 value 40.456950
iter 80 value 40.438053
iter 90 value 40.427694
iter 100 value 40.415859
final value 40.415859
stopped after 100 iterations
# weights: 27
initial value 403.781112
iter 10 value 12.541729
iter 20 value 0.030538
final value 0.000062
converged
# weights: 43
initial value 302.275294
iter 10 value 0.294419
iter 20 value 0.000867
iter 30 value 0.000260
final value 0.000067
converged
# weights: 11
initial value 336.372071
iter 10 value 128.425613
iter 20 value 105.682989
iter 30 value 74.635376
final value 74.307145
converged
# weights: 27
initial value 340.426845
iter 10 value 35.604294
iter 20 value 19.037074
iter 30 value 17.314992
iter 40 value 17.264075
iter 50 value 17.255338

```

```
iter 60 value 17.255139
final value 17.255137
converged
# weights: 43
initial value 362.392273
iter 10 value 47.459581
iter 20 value 15.707629
iter 30 value 15.405939
iter 40 value 15.360252
iter 50 value 15.344950
iter 60 value 15.332874
final value 15.331749
converged
# weights: 11
initial value 315.241448
iter 10 value 117.193721
iter 20 value 106.857552
iter 30 value 106.839778
iter 40 value 106.812350
iter 50 value 106.652904
iter 60 value 95.141705
iter 70 value 57.856437
iter 80 value 19.721408
iter 90 value 16.394371
iter 100 value 16.178794
final value 16.178794
stopped after 100 iterations
# weights: 27
initial value 306.750437
iter 10 value 1.143154
iter 20 value 0.154317
iter 30 value 0.139812
iter 40 value 0.128529
iter 50 value 0.111041
iter 60 value 0.104807
iter 70 value 0.101907
iter 80 value 0.097767
iter 90 value 0.092894
iter 100 value 0.090000
final value 0.090000
stopped after 100 iterations
# weights: 43
initial value 303.713491
iter 10 value 0.602716
iter 20 value 0.107837
iter 30 value 0.102516
```

```

iter 40 value 0.099333
iter 50 value 0.094389
iter 60 value 0.087151
iter 70 value 0.080392
iter 80 value 0.072327
iter 90 value 0.071100
iter 100 value 0.068732
final value 0.068732
stopped after 100 iterations
# weights: 11
initial value 339.448200
iter 10 value 147.752167
iter 20 value 107.359665
iter 30 value 101.297436
iter 40 value 96.627740
iter 50 value 48.482942
iter 60 value 28.406504
iter 70 value 26.157419
iter 80 value 25.109104
iter 90 value 24.543186
iter 100 value 24.414883
final value 24.414883
stopped after 100 iterations
# weights: 27
initial value 367.941116
iter 10 value 7.185971
iter 20 value 0.580100
iter 30 value 0.019829
iter 40 value 0.004934
iter 50 value 0.000838
iter 60 value 0.000295
final value 0.000093
converged
# weights: 43
initial value 333.908164
iter 10 value 2.437425
iter 20 value 0.467499
iter 30 value 0.021468
iter 40 value 0.004777
iter 50 value 0.002479
iter 60 value 0.000984
iter 70 value 0.000619
iter 80 value 0.000399
iter 90 value 0.000145
final value 0.000100
converged

```

```

# weights: 11
initial value 415.294864
iter 10 value 95.618186
iter 20 value 77.336636
iter 30 value 77.330371
final value 77.330343
converged
# weights: 27
initial value 307.849784
iter 10 value 32.799737
iter 20 value 21.528156
iter 30 value 21.118701
iter 40 value 20.909144
iter 50 value 20.893641
iter 50 value 20.893641
iter 50 value 20.893641
final value 20.893641
converged
# weights: 43
initial value 308.196723
iter 10 value 29.601239
iter 20 value 19.564641
iter 30 value 17.508262
iter 40 value 17.154163
iter 50 value 17.009296
iter 60 value 16.985502
iter 70 value 16.984677
final value 16.984558
converged
# weights: 11
initial value 322.765300
iter 10 value 81.478044
iter 20 value 28.693916
iter 30 value 26.310636
iter 40 value 25.294878
iter 50 value 25.212333
iter 60 value 25.202357
iter 70 value 25.187077
iter 80 value 25.184067
iter 90 value 25.183286
iter 100 value 25.181752
final value 25.181752
stopped after 100 iterations
# weights: 27
initial value 299.126266
iter 10 value 6.033717

```

```
iter 20 value 0.805135
iter 30 value 0.656232
iter 40 value 0.575037
iter 50 value 0.389693
iter 60 value 0.313354
iter 70 value 0.268954
iter 80 value 0.230183
iter 90 value 0.222769
iter 100 value 0.211361
final value 0.211361
stopped after 100 iterations
# weights: 43
initial value 305.711833
iter 10 value 3.214462
iter 20 value 0.208245
iter 30 value 0.184197
iter 40 value 0.174899
iter 50 value 0.169150
iter 60 value 0.166500
iter 70 value 0.160264
iter 80 value 0.157573
iter 90 value 0.156103
iter 100 value 0.153939
final value 0.153939
stopped after 100 iterations
# weights: 11
initial value 299.312883
iter 10 value 117.839872
iter 20 value 114.020065
iter 30 value 113.947371
iter 40 value 109.599754
iter 50 value 107.798663
iter 60 value 107.792040
final value 107.791520
converged
# weights: 27
initial value 319.812025
iter 10 value 2.958407
iter 20 value 0.069480
iter 30 value 0.015058
iter 40 value 0.003001
final value 0.000076
converged
# weights: 43
initial value 272.242348
iter 10 value 21.253311
```

```

iter 20 value 0.322650
iter 30 value 0.000552
final value 0.000057
converged
# weights: 11
initial value 285.059041
iter 10 value 118.132835
iter 20 value 90.842704
iter 30 value 78.885250
iter 40 value 78.884654
final value 78.884649
converged

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-2') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```

# weights: 27
initial value 290.287991
iter 10 value 68.936353
iter 20 value 22.176601
iter 30 value 21.753356
iter 40 value 21.742775
iter 50 value 21.696019
iter 60 value 21.516922
final value 21.515512
converged
# weights: 43
initial value 365.064125
iter 10 value 20.494980
iter 20 value 18.287705
iter 30 value 18.181209
iter 40 value 18.171325
iter 50 value 18.165938
final value 18.165748
converged
# weights: 11
initial value 313.472142
iter 10 value 114.541971
iter 20 value 101.851541
iter 30 value 75.049617

```

```

iter 40 value 49.859564
iter 50 value 45.144585
iter 60 value 43.919684
iter 70 value 43.853346
iter 80 value 43.849138
iter 90 value 43.832860
final value 43.832857
converged
# weights: 27
initial value 388.704944
iter 10 value 9.234671
iter 20 value 1.320652
iter 30 value 0.469608
iter 40 value 0.421953
iter 50 value 0.366494
iter 60 value 0.328079
iter 70 value 0.317506
iter 80 value 0.220561
iter 90 value 0.195140
iter 100 value 0.180566
final value 0.180566
stopped after 100 iterations
# weights: 43
initial value 294.731100
iter 10 value 4.900313
iter 20 value 0.254820
iter 30 value 0.220824
iter 40 value 0.197224
iter 50 value 0.193012
iter 60 value 0.184240
iter 70 value 0.178100
iter 80 value 0.176114
iter 90 value 0.173034
iter 100 value 0.170475
final value 0.170475
stopped after 100 iterations
# weights: 11
initial value 286.092477
iter 10 value 108.861103
iter 20 value 102.201287
iter 30 value 69.794439
iter 40 value 44.840356
iter 50 value 43.855362
iter 60 value 38.966946
iter 70 value 38.266820
iter 80 value 37.952443

```

```
iter 90 value 37.937962
iter 100 value 37.912657
final value 37.912657
stopped after 100 iterations
# weights: 27
initial value 320.698069
iter 10 value 6.714914
iter 20 value 0.399666
iter 30 value 0.004356
iter 40 value 0.000823
final value 0.000086
converged
# weights: 43
initial value 269.372969
iter 10 value 4.822603
iter 20 value 0.743834
iter 30 value 0.027273
iter 40 value 0.002647
iter 50 value 0.000380
iter 60 value 0.000110
final value 0.000094
converged
# weights: 11
initial value 282.247389
iter 10 value 99.510191
iter 20 value 78.626219
iter 30 value 78.569586
final value 78.569522
converged
# weights: 27
initial value 332.673626
iter 10 value 32.047338
iter 20 value 20.845931
iter 30 value 19.436577
iter 40 value 19.113526
iter 50 value 18.983607
iter 60 value 18.980755
iter 70 value 18.980225
iter 70 value 18.980225
iter 70 value 18.980225
final value 18.980225
converged
# weights: 43
initial value 326.533534
iter 10 value 38.400401
iter 20 value 20.468811
```

```
iter 30 value 17.645302
iter 40 value 17.098994
iter 50 value 17.040893
iter 60 value 17.005785
iter 70 value 17.003388
final value 17.002511
converged
# weights: 11
initial value 286.644472
iter 10 value 109.654847
iter 20 value 95.246932
iter 30 value 52.168863
iter 40 value 45.018249
iter 50 value 43.307774
iter 60 value 41.042631
iter 70 value 40.929730
iter 80 value 40.888015
final value 40.884567
converged
# weights: 27
initial value 315.256498
iter 10 value 33.760153
iter 20 value 28.153658
iter 30 value 23.874279
iter 40 value 18.834886
iter 50 value 16.062466
iter 60 value 10.015601
iter 70 value 6.579007
iter 80 value 4.946667
iter 90 value 3.663206
iter 100 value 3.050357
final value 3.050357
stopped after 100 iterations
# weights: 43
initial value 280.526149
iter 10 value 32.715121
iter 20 value 8.893817
iter 30 value 7.764815
iter 40 value 2.365116
iter 50 value 1.175449
iter 60 value 0.960918
iter 70 value 0.744704
iter 80 value 0.601756
iter 90 value 0.487309
iter 100 value 0.419140
final value 0.419140
```

```

stopped after 100 iterations
# weights: 11
initial value 311.725516
iter 10 value 107.244618
iter 20 value 96.083808
iter 30 value 57.443260
iter 40 value 46.715939
iter 50 value 46.081346
iter 60 value 46.054361
iter 70 value 46.011236
iter 80 value 46.006543
iter 90 value 46.000000
final value 45.999913
converged
# weights: 27
initial value 365.651314
iter 10 value 5.839552
iter 20 value 0.247126
iter 30 value 0.004614
final value 0.000078
converged
# weights: 43
initial value 247.977751
iter 10 value 30.244925
iter 20 value 8.322274
iter 30 value 0.088797
iter 40 value 0.003568
iter 50 value 0.001160
iter 60 value 0.000139
iter 70 value 0.000113
final value 0.000093
converged
# weights: 11
initial value 311.073052
iter 10 value 120.350840
iter 20 value 115.188032
iter 30 value 103.249057
final value 102.861600
converged
# weights: 27
initial value 298.624834
iter 10 value 29.443893
iter 20 value 19.850498
iter 30 value 18.844095
iter 40 value 18.684287
iter 50 value 18.680805

```

```
final  value 18.680363
converged
# weights: 43
initial  value 338.987255
iter   10 value 21.126286
iter   20 value 17.037809
iter   30 value 16.981255
iter   40 value 16.954316
iter   50 value 16.951775
final  value 16.951720
converged
# weights: 11
initial  value 315.417680
iter   10 value 129.897030
iter   20 value 105.290638
iter   30 value 100.039120
iter   40 value 54.155210
iter   50 value 28.516873
iter   60 value 26.770564
iter   70 value 26.587359
iter   80 value 26.502341
final  value 26.502222
converged
# weights: 27
initial  value 333.601501
iter   10 value 37.718083
iter   20 value 11.554569
iter   30 value 6.475432
iter   40 value 0.890811
iter   50 value 0.798795
iter   60 value 0.546422
iter   70 value 0.387161
iter   80 value 0.303920
iter   90 value 0.216214
iter  100 value 0.197614
final  value 0.197614
stopped after 100 iterations
# weights: 43
initial  value 319.057407
iter   10 value 4.040068
iter   20 value 0.408590
iter   30 value 0.356188
iter   40 value 0.285627
iter   50 value 0.230113
iter   60 value 0.204385
iter   70 value 0.176129
```

```

iter 80 value 0.167025
iter 90 value 0.163262
iter 100 value 0.157326
final value 0.157326
stopped after 100 iterations
# weights: 11
initial value 271.206747
iter 10 value 106.558702
iter 20 value 88.401370
iter 30 value 45.752251
iter 40 value 42.143585
iter 50 value 41.894139
iter 60 value 41.686952
iter 70 value 41.676954
iter 80 value 41.665986
iter 90 value 41.642620
iter 100 value 41.636921
final value 41.636921
stopped after 100 iterations
# weights: 27
initial value 286.617364
iter 10 value 8.020054
iter 20 value 0.033818
iter 30 value 0.000125
final value 0.000093
converged
# weights: 43
initial value 309.710266
iter 10 value 8.527029
iter 20 value 0.327074
iter 30 value 0.000535
final value 0.000075
converged
# weights: 11
initial value 296.469934
iter 10 value 118.907858
iter 20 value 90.972630
iter 30 value 78.716651
final value 78.709960
converged
# weights: 27
initial value 360.849908
iter 10 value 30.565275
iter 20 value 22.056593
iter 30 value 21.534171
iter 40 value 21.463379

```

```
iter 50 value 21.450052
final value 21.449533
converged
# weights: 43
initial value 294.040397
iter 10 value 36.608184
iter 20 value 19.532669
iter 30 value 18.196664
iter 40 value 17.847323
iter 50 value 17.812619
iter 60 value 17.806543
iter 70 value 17.797779
final value 17.797772
converged
# weights: 11
initial value 283.073835
iter 10 value 106.834354
iter 20 value 106.799846
iter 30 value 106.734793
iter 40 value 106.707015
iter 50 value 106.093354
iter 60 value 92.631498
iter 70 value 45.343590
iter 80 value 30.058894
iter 90 value 29.275735
iter 100 value 28.923133
final value 28.923133
stopped after 100 iterations
# weights: 27
initial value 368.409099
iter 10 value 2.205453
iter 20 value 0.205111
iter 30 value 0.198624
iter 40 value 0.192076
iter 50 value 0.184075
iter 60 value 0.181454
iter 70 value 0.179783
iter 80 value 0.177388
iter 90 value 0.175311
iter 100 value 0.174915
final value 0.174915
stopped after 100 iterations
# weights: 43
initial value 290.321984
iter 10 value 1.698621
iter 20 value 0.316317
```

```

iter 30 value 0.217868
iter 40 value 0.199690
iter 50 value 0.191858
iter 60 value 0.183054
iter 70 value 0.173129
iter 80 value 0.168388
iter 90 value 0.166469
iter 100 value 0.164475
final value 0.164475
stopped after 100 iterations
# weights: 11
initial value 302.479013
iter 10 value 112.267478
iter 20 value 106.651327
final value 106.645390
converged
# weights: 27
initial value 288.468519
iter 10 value 11.225273
iter 20 value 0.352269
iter 30 value 0.019073
final value 0.000072
converged
# weights: 43
initial value 358.457937
iter 10 value 2.434293
iter 20 value 0.113419
iter 30 value 0.003643
iter 40 value 0.000216
final value 0.000084
converged
# weights: 11
initial value 294.212944
iter 10 value 131.639721
iter 20 value 89.138004
iter 30 value 74.892710
final value 74.887940
converged
# weights: 27
initial value 271.295727
iter 10 value 33.614768
iter 20 value 20.912232
iter 30 value 18.896463
iter 40 value 18.648100
iter 50 value 18.632807
final value 18.632716

```

```

converged
# weights: 43
initial value 350.377503
iter 10 value 23.499009
iter 20 value 17.216515
iter 30 value 17.058298
iter 40 value 16.792661
iter 50 value 16.725156
iter 60 value 16.690543
iter 70 value 16.689012
final value 16.688999
converged
# weights: 11
initial value 289.865546
iter 10 value 105.570227
iter 20 value 97.448591
iter 30 value 91.848057
iter 40 value 54.343340
iter 50 value 28.197657
iter 60 value 24.243865
iter 70 value 23.985183
iter 80 value 23.486577
iter 90 value 23.412154
iter 100 value 23.208628
final value 23.208628
stopped after 100 iterations
# weights: 27
initial value 260.035929
iter 10 value 6.369759
iter 20 value 0.399576
iter 30 value 0.373652
iter 40 value 0.359566
iter 50 value 0.328798
iter 60 value 0.290420
iter 70 value 0.273482
iter 80 value 0.242471
iter 90 value 0.225355
iter 100 value 0.198768
final value 0.198768
stopped after 100 iterations
# weights: 43
initial value 311.451484
iter 10 value 4.589203
iter 20 value 0.291746
iter 30 value 0.223012
iter 40 value 0.208364

```

```

iter 50 value 0.190458
iter 60 value 0.184844
iter 70 value 0.175938
iter 80 value 0.171640
iter 90 value 0.167886
iter 100 value 0.162506
final value 0.162506
stopped after 100 iterations
# weights: 11
initial value 297.636641
iter 10 value 116.673355
iter 20 value 107.420408
final value 107.405872
converged
# weights: 27
initial value 297.191152
iter 10 value 0.425021
iter 20 value 0.034259
iter 30 value 0.005624
final value 0.000098
converged

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-3') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```

# weights: 43
initial value 261.493721
iter 10 value 0.793726
iter 20 value 0.015880
iter 30 value 0.000568
final value 0.000085
converged
# weights: 11
initial value 275.439852
iter 10 value 86.017758
iter 20 value 75.429499
iter 30 value 75.354098
final value 75.354059
converged
# weights: 27

```

```
initial value 303.657109
iter 10 value 35.205607
iter 20 value 20.200806
iter 30 value 18.901191
iter 40 value 18.653082
iter 50 value 17.945176
iter 60 value 17.835247
iter 70 value 17.834921
iter 70 value 17.834921
iter 70 value 17.834921
final value 17.834921
converged
# weights: 43
initial value 301.834183
iter 10 value 25.123438
iter 20 value 16.836756
iter 30 value 15.946332
iter 40 value 15.790000
iter 50 value 15.755214
iter 60 value 15.718248
iter 70 value 15.707766
iter 80 value 15.704931
iter 80 value 15.704931
iter 80 value 15.704931
final value 15.704931
converged
# weights: 11
initial value 345.540483
iter 10 value 96.160533
iter 20 value 49.230109
iter 30 value 46.634310
iter 40 value 46.171993
iter 50 value 46.137321
iter 60 value 46.108003
iter 70 value 46.078631
iter 80 value 46.077026
iter 90 value 46.073085
iter 100 value 46.071941
final value 46.071941
stopped after 100 iterations
# weights: 27
initial value 351.482406
iter 10 value 5.730027
iter 20 value 0.212056
iter 30 value 0.178290
iter 40 value 0.163732
```

```
iter 50 value 0.158508
iter 60 value 0.152134
iter 70 value 0.138935
iter 80 value 0.131103
iter 90 value 0.116552
iter 100 value 0.104587
final value 0.104587
stopped after 100 iterations
# weights: 43
initial value 357.659091
iter 10 value 1.451918
iter 20 value 0.104329
iter 30 value 0.101551
iter 40 value 0.097920
iter 50 value 0.093699
iter 60 value 0.087002
iter 70 value 0.084653
iter 80 value 0.077858
iter 90 value 0.076069
iter 100 value 0.074463
final value 0.074463
stopped after 100 iterations
# weights: 11
initial value 310.411317
iter 10 value 178.931880
iter 20 value 160.684184
iter 30 value 155.552175
iter 40 value 100.295928
iter 50 value 70.191415
iter 60 value 46.262520
iter 70 value 42.023581
iter 80 value 40.757545
iter 90 value 40.576988
iter 100 value 39.844847
final value 39.844847
stopped after 100 iterations
# weights: 27
initial value 388.025559
iter 10 value 18.285423
iter 20 value 2.065058
iter 30 value 0.592316
iter 40 value 0.046009
iter 50 value 0.001924
iter 60 value 0.001160
iter 70 value 0.000856
iter 80 value 0.000128
```

```
final  value 0.000093
converged
# weights: 43
initial  value 315.479268
iter   10 value 3.326957
iter   20 value 0.044679
iter   30 value 0.000749
final  value 0.000070
converged
# weights: 11
initial  value 312.773836
iter   10 value 113.010671
iter   20 value 79.756162
iter   30 value 77.714994
final  value 77.528234
converged
# weights: 27
initial  value 304.313507
iter   10 value 34.345553
iter   20 value 20.490810
iter   30 value 19.297734
iter   40 value 19.129923
iter   50 value 19.080255
final  value 19.080238
converged
# weights: 43
initial  value 383.818883
iter   10 value 34.645311
iter   20 value 20.408244
iter   30 value 19.420751
iter   40 value 19.367536
iter   50 value 19.309346
iter   60 value 19.305127
iter   70 value 19.304886
iter   80 value 19.304858
iter   80 value 19.304858
iter   80 value 19.304858
final  value 19.304858
converged
# weights: 11
initial  value 307.292227
iter   10 value 100.459544
iter   20 value 49.061605
iter   30 value 26.280315
iter   40 value 25.275380
iter   50 value 24.954037
```

```

iter 60 value 24.933215
iter 70 value 24.926514
iter 80 value 24.913332
final value 24.913319
converged
# weights: 27
initial value 363.999685
iter 10 value 41.755153
iter 20 value 0.553948
iter 30 value 0.286205
iter 40 value 0.235655
iter 50 value 0.224274
iter 60 value 0.212498
iter 70 value 0.207414
iter 80 value 0.200645
iter 90 value 0.196436
iter 100 value 0.194549
final value 0.194549
stopped after 100 iterations
# weights: 43
initial value 311.281593
iter 10 value 1.523751
iter 20 value 0.268585
iter 30 value 0.218150
iter 40 value 0.211630
iter 50 value 0.198723
iter 60 value 0.189154
iter 70 value 0.182068
iter 80 value 0.175266
iter 90 value 0.168160
iter 100 value 0.167005
final value 0.167005
stopped after 100 iterations
# weights: 11
initial value 275.454216
iter 10 value 106.798879
iter 20 value 85.536090
iter 30 value 38.457404
iter 40 value 31.959789
iter 50 value 27.204549
iter 60 value 27.019391
iter 70 value 26.608060
iter 80 value 26.251012
iter 90 value 26.167017
iter 100 value 26.130658
final value 26.130658

```

```

stopped after 100 iterations
# weights: 27
initial value 338.095112
iter 10 value 5.392730
iter 20 value 0.332196
iter 30 value 0.001458
final value 0.000094
converged
# weights: 43
initial value 297.245415
iter 10 value 14.276046
iter 20 value 0.534983
iter 30 value 0.041752
iter 40 value 0.005486
iter 50 value 0.001244
iter 60 value 0.000834
final value 0.000093
converged
# weights: 11
initial value 278.591374
iter 10 value 121.773089
iter 20 value 78.643190
iter 30 value 77.327536
iter 30 value 77.327536
final value 77.327536
converged
# weights: 27
initial value 331.206004
iter 10 value 37.011551
iter 20 value 20.942973
iter 30 value 20.040578
iter 40 value 19.855458
iter 50 value 19.573009
iter 60 value 19.441065
iter 70 value 19.440775
iter 70 value 19.440774
iter 70 value 19.440774
final value 19.440774
converged
# weights: 43
initial value 297.310825
iter 10 value 31.162969
iter 20 value 19.984054
iter 30 value 17.975464
iter 40 value 17.671227
iter 50 value 17.635120

```

```
iter 60 value 17.628314
iter 70 value 17.628237
iter 80 value 17.628233
final value 17.628231
converged
# weights: 11
initial value 277.960819
iter 10 value 95.187175
iter 20 value 29.684934
iter 30 value 27.446794
iter 40 value 26.850839
iter 50 value 26.623650
iter 60 value 26.600653
iter 70 value 26.592480
final value 26.592465
converged
# weights: 27
initial value 309.929194
iter 10 value 27.861324
iter 20 value 19.500663
iter 30 value 11.492985
iter 40 value 1.147121
iter 50 value 0.735465
iter 60 value 0.618353
iter 70 value 0.340143
iter 80 value 0.238620
iter 90 value 0.221896
iter 100 value 0.210948
final value 0.210948
stopped after 100 iterations
# weights: 43
initial value 299.170495
iter 10 value 5.536159
iter 20 value 0.536690
iter 30 value 0.234646
iter 40 value 0.218493
iter 50 value 0.199744
iter 60 value 0.193657
iter 70 value 0.190860
iter 80 value 0.185527
iter 90 value 0.181094
iter 100 value 0.178547
final value 0.178547
stopped after 100 iterations
# weights: 11
initial value 331.406386
```

```
iter 10 value 106.623836
iter 20 value 95.944427
iter 30 value 62.651406
iter 40 value 42.999851
iter 50 value 42.356433
iter 60 value 42.086071
iter 70 value 41.944097
iter 80 value 41.939000
iter 90 value 41.914090
iter 100 value 41.913719
final value 41.913719
stopped after 100 iterations
# weights: 27
initial value 297.203272
iter 10 value 4.602026
iter 20 value 0.059194
iter 30 value 0.000432
iter 40 value 0.000118
final value 0.000097
converged
# weights: 43
initial value 283.806183
iter 10 value 4.266324
iter 20 value 0.068181
iter 30 value 0.008926
iter 40 value 0.002898
final value 0.000083
converged
# weights: 11
initial value 297.008390
iter 10 value 131.340083
iter 20 value 108.786068
iter 30 value 79.927494
final value 78.970385
converged
# weights: 27
initial value 272.574767
iter 10 value 21.127360
iter 20 value 19.281759
iter 30 value 19.119313
iter 40 value 19.110874
final value 19.108980
converged
# weights: 43
initial value 350.064634
iter 10 value 27.768746
```

```

iter 20 value 19.263217
iter 30 value 17.920587
iter 40 value 17.686505
iter 50 value 17.625627
iter 60 value 17.621970
iter 70 value 17.621666
final value 17.621648
converged
# weights: 11
initial value 282.748818
iter 10 value 149.573259
iter 20 value 104.528412
iter 30 value 103.436881
iter 40 value 102.089907
iter 50 value 101.490888
iter 60 value 101.260337
iter 70 value 100.779602
iter 80 value 100.481484
iter 90 value 100.066570
iter 100 value 100.065536
final value 100.065536
stopped after 100 iterations
# weights: 27
initial value 300.271280
iter 10 value 2.948371
iter 20 value 0.305833
iter 30 value 0.190244
iter 40 value 0.182701
iter 50 value 0.174703
iter 60 value 0.159012
iter 70 value 0.147581
iter 80 value 0.133453
iter 90 value 0.125463
iter 100 value 0.122264
final value 0.122264
stopped after 100 iterations
# weights: 43
initial value 316.848001
iter 10 value 3.642532
iter 20 value 0.210583
iter 30 value 0.159918
iter 40 value 0.151754
iter 50 value 0.144754
iter 60 value 0.141122
iter 70 value 0.128490
iter 80 value 0.119867

```

```

iter 90 value 0.116486
iter 100 value 0.114278
final value 0.114278
stopped after 100 iterations
# weights: 11
initial value 308.843928
iter 10 value 99.400122
iter 20 value 24.581434
iter 30 value 23.558813
iter 40 value 23.033504
iter 50 value 22.885967
iter 60 value 22.834451
iter 70 value 22.810193
iter 80 value 22.800181
iter 90 value 22.774605
iter 100 value 22.751444
final value 22.751444
stopped after 100 iterations
# weights: 27
initial value 291.070687
iter 10 value 1.459322
iter 20 value 0.009180
iter 30 value 0.000155
final value 0.000097
converged
# weights: 43
initial value 324.803395
iter 10 value 0.205759
iter 20 value 0.018875
iter 30 value 0.003148
iter 40 value 0.001966
final value 0.000058
converged
# weights: 11
initial value 308.178216
iter 10 value 125.450714
iter 20 value 105.981716
iter 30 value 101.967411
final value 101.144270
converged
# weights: 27
initial value 313.694077
iter 10 value 30.508661
iter 20 value 19.474310
iter 30 value 19.267782
iter 40 value 19.196162

```

```
iter 50 value 19.194028
iter 50 value 19.194028
iter 50 value 19.194028
final value 19.194028
converged
# weights: 43
initial value 302.170221
iter 10 value 21.864460
iter 20 value 16.182354
iter 30 value 15.587904
iter 40 value 15.522263
iter 50 value 15.498979
iter 60 value 15.491148
final value 15.491145
converged
# weights: 11
initial value 369.624653
iter 10 value 82.148801
iter 20 value 25.806527
iter 30 value 23.743811
iter 40 value 23.498514
iter 50 value 23.411284
final value 23.411174
converged
# weights: 27
initial value 318.839346
iter 10 value 5.898609
iter 20 value 0.299908
iter 30 value 0.181012
iter 40 value 0.177631
iter 50 value 0.160873
iter 60 value 0.147225
iter 70 value 0.136123
iter 80 value 0.127400
iter 90 value 0.116311
iter 100 value 0.098240
final value 0.098240
stopped after 100 iterations
# weights: 43
initial value 299.323691
iter 10 value 0.773273
iter 20 value 0.219295
iter 30 value 0.199931
iter 40 value 0.180302
iter 50 value 0.143348
iter 60 value 0.105703
```

```

iter 70 value 0.090479
iter 80 value 0.082270
iter 90 value 0.078826
iter 100 value 0.075947
final value 0.075947
stopped after 100 iterations

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-4') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```

# weights: 11
initial value 279.043804
iter 10 value 91.216142
iter 20 value 27.657563
iter 30 value 18.722645
iter 40 value 16.448326
iter 50 value 16.162843
iter 60 value 16.035376
iter 70 value 15.599061
iter 80 value 15.506204
iter 90 value 15.484525
iter 100 value 15.432069
final value 15.432069
stopped after 100 iterations
# weights: 27
initial value 284.598186
iter 10 value 7.550266
iter 20 value 0.212358
iter 30 value 0.009571
iter 40 value 0.000252
final value 0.000090
converged
# weights: 43
initial value 450.157683
iter 10 value 6.678958
iter 20 value 1.155257
iter 30 value 0.003085
iter 40 value 0.001262
final value 0.000078
converged

```

```

# weights: 11
initial value 295.347956
iter 10 value 137.569065
iter 20 value 119.783837
iter 30 value 113.206776
iter 40 value 102.049064
final value 102.032951
converged
# weights: 27
initial value 357.325962
iter 10 value 47.446923
iter 20 value 24.037947
iter 30 value 21.813468
iter 40 value 21.278346
iter 50 value 20.957431
iter 60 value 20.910599
iter 70 value 20.901041
final value 20.901040
converged
# weights: 43
initial value 318.753420
iter 10 value 33.399800
iter 20 value 18.322065
iter 30 value 17.488313
iter 40 value 17.351260
iter 50 value 17.339274
iter 60 value 17.336620
iter 70 value 17.336562
final value 17.336561
converged
# weights: 11
initial value 329.952019
iter 10 value 118.550582
iter 20 value 54.335770
iter 30 value 47.128469
iter 40 value 45.325117
iter 50 value 45.268007
iter 60 value 45.257390
iter 70 value 45.210568
final value 45.210559
converged
# weights: 27
initial value 274.108084
iter 10 value 2.445176
iter 20 value 0.353609
iter 30 value 0.240012

```

```
iter 40 value 0.226537
iter 50 value 0.217866
iter 60 value 0.214610
iter 70 value 0.210699
iter 80 value 0.206704
iter 90 value 0.192739
iter 100 value 0.185166
final value 0.185166
stopped after 100 iterations
# weights: 43
initial value 408.355188
iter 10 value 1.262057
iter 20 value 1.055574
iter 30 value 0.857341
iter 40 value 0.691743
iter 50 value 0.468972
iter 60 value 0.321945
iter 70 value 0.258345
iter 80 value 0.202647
iter 90 value 0.184868
iter 100 value 0.178541
final value 0.178541
stopped after 100 iterations
# weights: 11
initial value 325.266912
iter 10 value 107.900314
iter 20 value 51.461814
iter 30 value 43.641951
iter 40 value 41.918675
iter 50 value 41.700379
iter 60 value 41.680577
iter 70 value 41.665750
iter 80 value 41.654636
iter 90 value 41.649599
iter 100 value 41.628067
final value 41.628067
stopped after 100 iterations
# weights: 27
initial value 342.328764
iter 10 value 4.435579
iter 20 value 0.004205
final value 0.000066
converged
# weights: 43
initial value 357.279627
iter 10 value 26.033894
```

```
iter 20 value 0.200620
iter 30 value 0.023271
iter 40 value 0.003917
iter 50 value 0.001674
iter 60 value 0.000216
final value 0.000099
converged
# weights: 11
initial value 290.076748
iter 10 value 185.167251
iter 20 value 121.406786
iter 30 value 115.840608
iter 40 value 83.262370
iter 50 value 79.026503
iter 60 value 79.024557
iter 60 value 79.024557
iter 60 value 79.024557
final value 79.024557
converged
# weights: 27
initial value 353.208969
iter 10 value 44.971866
iter 20 value 19.652198
iter 30 value 18.365394
iter 40 value 17.952754
iter 50 value 17.562337
iter 60 value 17.461028
iter 70 value 17.458874
final value 17.458873
converged
# weights: 43
initial value 308.451421
iter 10 value 44.983613
iter 20 value 18.725790
iter 30 value 16.115812
iter 40 value 16.013563
iter 50 value 15.997562
iter 60 value 15.994952
iter 70 value 15.994864
iter 70 value 15.994864
iter 70 value 15.994864
final value 15.994864
converged
# weights: 11
initial value 333.751642
iter 10 value 84.184418
```

```
iter 20 value 32.642634
iter 30 value 28.526408
iter 40 value 27.359110
iter 50 value 26.891052
iter 60 value 26.861069
iter 70 value 26.850369
final value 26.850354
converged
# weights: 27
initial value 281.711731
iter 10 value 0.679250
iter 20 value 0.164934
iter 30 value 0.157723
iter 40 value 0.144698
iter 50 value 0.134655
iter 60 value 0.115091
iter 70 value 0.109168
iter 80 value 0.103489
iter 90 value 0.093851
iter 100 value 0.088822
final value 0.088822
stopped after 100 iterations
# weights: 43
initial value 325.354617
iter 10 value 3.348455
iter 20 value 0.166915
iter 30 value 0.135288
iter 40 value 0.125014
iter 50 value 0.122741
iter 60 value 0.119480
iter 70 value 0.115771
iter 80 value 0.110060
iter 90 value 0.105946
iter 100 value 0.103673
final value 0.103673
stopped after 100 iterations
# weights: 11
initial value 354.852962
iter 10 value 112.791549
iter 20 value 78.296243
iter 30 value 34.972767
iter 40 value 28.241800
iter 50 value 27.769711
iter 60 value 27.632571
iter 70 value 27.438126
iter 80 value 27.363747
```

```
iter 90 value 27.329580
iter 100 value 27.284042
final value 27.284042
stopped after 100 iterations
# weights: 27
initial value 287.265502
iter 10 value 5.260263
iter 20 value 0.133211
iter 30 value 0.002434
final value 0.000078
converged
# weights: 43
initial value 298.384980
iter 10 value 2.499290
iter 20 value 0.031888
iter 30 value 0.000641
final value 0.000096
converged
# weights: 11
initial value 337.513072
iter 10 value 115.462032
iter 20 value 82.901153
iter 30 value 77.879335
final value 77.798196
converged
# weights: 27
initial value 339.051645
iter 10 value 28.910792
iter 20 value 20.637711
iter 30 value 20.178462
iter 40 value 20.138553
iter 50 value 20.138098
final value 20.138094
converged
# weights: 43
initial value 276.383626
iter 10 value 19.828780
iter 20 value 17.279058
iter 30 value 16.979826
iter 40 value 16.969571
iter 50 value 16.968730
final value 16.968721
converged
# weights: 11
initial value 304.908112
iter 10 value 107.155830
```

```

iter 20 value 42.785449
iter 30 value 32.256889
iter 40 value 28.218470
iter 50 value 28.051365
iter 60 value 28.003013
iter 70 value 27.859600
iter 80 value 27.859164
iter 90 value 27.858917
final value 27.858915
converged
# weights: 27
initial value 357.234210
iter 10 value 8.399221
iter 20 value 0.517619
iter 30 value 0.435819
iter 40 value 0.399414
iter 50 value 0.349809
iter 60 value 0.288834
iter 70 value 0.261618
iter 80 value 0.243610
iter 90 value 0.211577
iter 100 value 0.196237
final value 0.196237
stopped after 100 iterations
# weights: 43
initial value 313.660475
iter 10 value 3.813389
iter 20 value 0.300058
iter 30 value 0.210613
iter 40 value 0.192622
iter 50 value 0.190327
iter 60 value 0.172028
iter 70 value 0.165121
iter 80 value 0.161914
iter 90 value 0.158190
iter 100 value 0.154015
final value 0.154015
stopped after 100 iterations
# weights: 11
initial value 270.959770
iter 10 value 101.265763
iter 20 value 97.712485
iter 30 value 96.139305
iter 40 value 41.527965
iter 50 value 28.736374
iter 60 value 26.911880

```

```

iter 70 value 26.530463
iter 80 value 26.348710
iter 90 value 26.303071
iter 100 value 26.200534
final value 26.200534
stopped after 100 iterations
# weights: 27
initial value 350.832914
iter 10 value 13.626788
iter 20 value 0.701446
iter 30 value 0.008341
iter 40 value 0.003362
iter 50 value 0.001334
iter 60 value 0.001149
iter 70 value 0.000881
iter 80 value 0.000755
iter 90 value 0.000684
iter 100 value 0.000388
final value 0.000388
stopped after 100 iterations
# weights: 43
initial value 311.944197
iter 10 value 5.356567
iter 20 value 0.362750
iter 30 value 0.002377
iter 40 value 0.000728
final value 0.000073
converged
# weights: 11
initial value 301.321141
iter 10 value 111.948924
iter 20 value 79.524757
final value 77.753853
converged
# weights: 27
initial value 320.881214
iter 10 value 43.793940
iter 20 value 20.620339
iter 30 value 19.354006
iter 40 value 19.273120
iter 50 value 19.257693
final value 19.257662
converged
# weights: 43
initial value 330.696073
iter 10 value 20.086137

```

```
iter 20 value 17.663800
iter 30 value 17.559405
iter 40 value 17.518599
iter 50 value 17.514775
final value 17.514751
converged
# weights: 11
initial value 320.952804
iter 10 value 123.441334
iter 20 value 105.872561
iter 30 value 105.008945
iter 40 value 104.619953
iter 50 value 104.611266
iter 60 value 104.606874
iter 70 value 104.605892
iter 70 value 104.605891
final value 104.605891
converged
# weights: 27
initial value 316.203762
iter 10 value 14.826951
iter 20 value 2.251003
iter 30 value 0.378532
iter 40 value 0.296250
iter 50 value 0.274120
iter 60 value 0.252682
iter 70 value 0.243026
iter 80 value 0.234991
iter 90 value 0.225689
iter 100 value 0.214128
final value 0.214128
stopped after 100 iterations
# weights: 43
initial value 317.858027
iter 10 value 2.389444
iter 20 value 0.250294
iter 30 value 0.240988
iter 40 value 0.228919
iter 50 value 0.219026
iter 60 value 0.201540
iter 70 value 0.193819
iter 80 value 0.184599
iter 90 value 0.172499
iter 100 value 0.166425
final value 0.166425
stopped after 100 iterations
```

```

# weights: 11
initial value 321.380837
iter 10 value 124.183308
iter 20 value 120.299302
iter 30 value 119.763426
iter 40 value 119.755077
iter 50 value 119.750355
iter 60 value 119.749746
final value 119.749241
converged
# weights: 27
initial value 293.074468
iter 10 value 2.561636
iter 20 value 0.301562
iter 30 value 0.004556
final value 0.000059
converged
# weights: 43
initial value 310.329352
iter 10 value 2.445244
iter 20 value 0.090874
iter 30 value 0.000530
iter 40 value 0.000101
final value 0.000100
converged
# weights: 11
initial value 348.969278
iter 10 value 151.980149
iter 20 value 112.909733
iter 30 value 103.947816
iter 40 value 102.309603
iter 40 value 102.309603
iter 40 value 102.309603
final value 102.309603
converged
# weights: 27
initial value 300.196136
iter 10 value 44.565286
iter 20 value 21.790107
iter 30 value 20.258995
iter 40 value 19.495897
iter 50 value 19.406098
iter 60 value 19.404888
final value 19.404864
converged
# weights: 43

```

```

initial  value 334.681213
iter   10 value 47.676242
iter   20 value 18.697414
iter   30 value 17.918985
iter   40 value 17.802844
iter   50 value 17.766645
iter   60 value 17.765590
iter   70 value 17.765355
final   value 17.765355
converged
# weights: 11
initial  value 304.295228
iter   10 value 123.206593
iter   20 value 100.174506
iter   30 value 97.046045
iter   40 value 96.472478
iter   50 value 96.395812
iter   60 value 96.299800
iter   70 value 96.247147
iter   80 value 96.243463
iter   90 value 96.233430
final   value 96.232843
converged

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-5') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```

# weights: 27
initial  value 303.419060
iter   10 value 2.002207
iter   20 value 0.347296
iter   30 value 0.236080
iter   40 value 0.217189
iter   50 value 0.213506
iter   60 value 0.212518
iter   70 value 0.207633
iter   80 value 0.201541
iter   90 value 0.198204
iter  100 value 0.193102
final   value 0.193102

```

```
stopped after 100 iterations
# weights: 43
initial value 287.882829
iter 10 value 22.776825
iter 20 value 0.345781
iter 30 value 0.301125
iter 40 value 0.279733
iter 50 value 0.253170
iter 60 value 0.228939
iter 70 value 0.212785
iter 80 value 0.197506
iter 90 value 0.192466
iter 100 value 0.187848
final value 0.187848
stopped after 100 iterations
# weights: 11
initial value 337.075118
iter 10 value 115.614653
iter 20 value 106.655248
final value 106.645623
converged
# weights: 27
initial value 303.078048
iter 10 value 4.146151
iter 20 value 0.054673
final value 0.000051
converged
# weights: 43
initial value 292.472022
iter 10 value 2.391737
iter 20 value 0.269749
iter 30 value 0.004612
iter 40 value 0.000295
final value 0.000082
converged
# weights: 11
initial value 322.159807
iter 10 value 140.469642
iter 20 value 118.514904
iter 30 value 116.964390
iter 40 value 103.840109
iter 50 value 101.292331
final value 101.292219
converged
# weights: 27
initial value 301.122251
```

```

iter 10 value 51.204624
iter 20 value 24.478330
iter 30 value 21.999887
iter 40 value 21.360739
iter 50 value 21.281401
final value 21.281372
converged
# weights: 43
initial value 324.897727
iter 10 value 75.343826
iter 20 value 19.668286
iter 30 value 17.799600
iter 40 value 17.521579
iter 50 value 17.335212
iter 60 value 17.250909
iter 70 value 17.223096
iter 80 value 17.186317
iter 90 value 17.184199
iter 100 value 17.183949
final value 17.183949
stopped after 100 iterations
# weights: 11
initial value 330.170768
iter 10 value 107.512294
iter 20 value 106.752076
iter 30 value 103.215477
iter 40 value 72.287871
iter 50 value 45.189022
iter 60 value 41.814938
iter 70 value 38.999877
iter 80 value 38.101419
iter 90 value 38.078090
iter 100 value 38.072522
final value 38.072522
stopped after 100 iterations
# weights: 27
initial value 322.875275
iter 10 value 12.630716
iter 20 value 0.829408
iter 30 value 0.309995
iter 40 value 0.295111
iter 50 value 0.275457
iter 60 value 0.204095
iter 70 value 0.191208
iter 80 value 0.188641
iter 90 value 0.185546

```

```

iter 100 value 0.183357
final  value 0.183357
stopped after 100 iterations
# weights: 43
initial  value 356.817529
iter   10 value 18.250631
iter   20 value 5.412225
iter   30 value 2.286146
iter   40 value 0.577236
iter   50 value 0.454275
iter   60 value 0.428064
iter   70 value 0.383600
iter   80 value 0.301213
iter   90 value 0.206594
iter 100 value 0.181550
final  value 0.181550
stopped after 100 iterations
# weights: 11
initial  value 309.885260
iter   10 value 116.525825
iter   20 value 98.058159
iter   30 value 97.124505
iter   40 value 93.731951
iter   50 value 91.309379
iter   60 value 91.221611
iter   70 value 91.209655
iter   80 value 91.159315
iter   90 value 90.621410
iter 100 value 88.821771
final  value 88.821771
stopped after 100 iterations
# weights: 27
initial  value 324.812289
iter   10 value 9.443748
iter   20 value 0.487615
iter   30 value 0.106715
iter   40 value 0.005987
final  value 0.000064
converged
# weights: 43
initial  value 279.944051
iter   10 value 7.048273
iter   20 value 0.195230
iter   30 value 0.031182
iter   40 value 0.000213
final  value 0.000098

```

```

converged
# weights: 11
initial value 332.270124
iter 10 value 144.658450
iter 20 value 84.912995
iter 30 value 77.410890
final value 76.964895
converged
# weights: 27
initial value 303.422052
iter 10 value 32.152651
iter 20 value 22.677347
iter 30 value 22.619261
iter 40 value 22.617945
iter 50 value 22.617848
final value 22.617848
converged
# weights: 43
initial value 297.191269
iter 10 value 25.919542
iter 20 value 17.877576
iter 30 value 17.719355
iter 40 value 17.701866
iter 50 value 17.684290
iter 60 value 17.672502
iter 70 value 17.594366
final value 17.594195
converged
# weights: 11
initial value 285.142597
iter 10 value 96.374713
iter 20 value 41.180131
iter 30 value 24.987792
iter 40 value 24.452560
iter 50 value 23.893729
iter 60 value 23.254891
iter 70 value 23.222254
iter 80 value 23.138837
iter 90 value 23.136217
iter 100 value 23.133449
final value 23.133449
stopped after 100 iterations
# weights: 27
initial value 491.890686
iter 10 value 9.533482
iter 20 value 0.380021

```

```
iter 30 value 0.303463
iter 40 value 0.287338
iter 50 value 0.256855
iter 60 value 0.244414
iter 70 value 0.234792
iter 80 value 0.214567
iter 90 value 0.195357
iter 100 value 0.185773
final value 0.185773
stopped after 100 iterations
# weights: 43
initial value 296.842597
iter 10 value 2.479203
iter 20 value 0.294360
iter 30 value 0.258872
iter 40 value 0.246563
iter 50 value 0.234929
iter 60 value 0.217051
iter 70 value 0.208920
iter 80 value 0.192624
iter 90 value 0.189775
iter 100 value 0.180409
final value 0.180409
stopped after 100 iterations
# weights: 11
initial value 286.683137
iter 10 value 106.666131
iter 20 value 79.312986
iter 30 value 35.486703
iter 40 value 19.876362
iter 50 value 19.349946
iter 60 value 18.759078
iter 70 value 18.615154
iter 80 value 18.588056
iter 90 value 18.432520
iter 100 value 18.420618
final value 18.420618
stopped after 100 iterations
# weights: 27
initial value 339.943195
iter 10 value 3.770642
iter 20 value 0.016168
iter 30 value 0.000820
final value 0.000080
converged
# weights: 43
```

```
initial value 401.094258
iter 10 value 1.758668
iter 20 value 0.028182
iter 30 value 0.005692
iter 40 value 0.001564
iter 50 value 0.000184
iter 60 value 0.000119
final value 0.000088
converged
# weights: 11
initial value 299.265077
iter 10 value 140.918211
iter 20 value 120.530813
iter 30 value 109.102840
iter 40 value 99.523528
final value 99.522953
converged
# weights: 27
initial value 355.292683
iter 10 value 43.162467
iter 20 value 24.954412
iter 30 value 18.591216
iter 40 value 17.876278
iter 50 value 17.264385
iter 60 value 16.924773
iter 70 value 16.647779
final value 16.646854
converged
# weights: 43
initial value 314.451015
iter 10 value 25.112246
iter 20 value 16.010574
iter 30 value 15.040065
iter 40 value 14.924990
iter 50 value 14.921740
iter 60 value 14.921579
final value 14.921552
converged
# weights: 11
initial value 280.724140
iter 10 value 106.061767
iter 20 value 39.902746
iter 30 value 19.880056
iter 40 value 19.347434
iter 50 value 19.246777
iter 60 value 19.139796
```

```

iter 70 value 19.122416
iter 80 value 19.068651
iter 90 value 19.067714
iter 100 value 19.063110
final value 19.063110
stopped after 100 iterations
# weights: 27
initial value 319.569874
iter 10 value 0.884548
iter 20 value 0.137366
iter 30 value 0.130594
iter 40 value 0.115189
iter 50 value 0.098888
iter 60 value 0.087713
iter 70 value 0.083272
iter 80 value 0.076918
iter 90 value 0.073753
iter 100 value 0.071258
final value 0.071258
stopped after 100 iterations
# weights: 43
initial value 394.294311
iter 10 value 1.169476
iter 20 value 0.278702
iter 30 value 0.260588
iter 40 value 0.224829
iter 50 value 0.143626
iter 60 value 0.120210
iter 70 value 0.095989
iter 80 value 0.086032
iter 90 value 0.070503
iter 100 value 0.067337
final value 0.067337
stopped after 100 iterations
# weights: 11
initial value 304.289942
iter 10 value 90.036986
iter 20 value 36.998157
iter 30 value 29.552556
iter 40 value 29.024448
iter 50 value 28.609087
iter 60 value 28.502524
iter 70 value 28.459090
iter 80 value 28.385084
iter 90 value 28.376019
iter 100 value 28.310160

```

```

final  value 28.310160
stopped after 100 iterations
# weights: 27
initial  value 285.428991
iter  10 value 22.953679
iter  20 value 0.500925
iter  30 value 0.013325
iter  40 value 0.005722
iter  50 value 0.003780
iter  60 value 0.000858
final  value 0.000065
converged
# weights: 43
initial  value 381.547454
iter  10 value 6.334663
iter  20 value 3.644834
iter  30 value 0.113669
iter  40 value 0.010850
iter  50 value 0.003170
iter  60 value 0.001009
iter  70 value 0.000620
final  value 0.000073
converged
# weights: 11
initial  value 294.606215
iter  10 value 150.566709
iter  20 value 87.654443
iter  30 value 79.156610
final  value 79.053459
converged
# weights: 27
initial  value 312.261653
iter  10 value 68.042673
iter  20 value 20.858818
iter  30 value 19.387483
iter  40 value 18.956136
iter  50 value 18.928785
final  value 18.928471
converged
# weights: 43
initial  value 335.833039
iter  10 value 25.685604
iter  20 value 17.339487
iter  30 value 17.071719
iter  40 value 17.005208
iter  50 value 17.000293

```

```
iter 60 value 16.970551
iter 70 value 16.969325
final value 16.969324
converged
# weights: 11
initial value 296.318621
iter 10 value 106.757352
iter 20 value 106.739012
iter 30 value 106.722187
iter 40 value 106.695037
iter 50 value 106.681695
iter 60 value 106.633051
iter 70 value 104.328274
iter 80 value 90.690191
iter 90 value 35.222993
iter 100 value 30.292911
final value 30.292911
stopped after 100 iterations
# weights: 27
initial value 301.684484
iter 10 value 2.971304
iter 20 value 0.288377
iter 30 value 0.240483
iter 40 value 0.216405
iter 50 value 0.201512
iter 60 value 0.190812
iter 70 value 0.180330
iter 80 value 0.172959
iter 90 value 0.165247
iter 100 value 0.155952
final value 0.155952
stopped after 100 iterations
# weights: 43
initial value 283.120214
iter 10 value 3.684585
iter 20 value 0.206479
iter 30 value 0.175828
iter 40 value 0.170594
iter 50 value 0.157152
iter 60 value 0.147442
iter 70 value 0.140568
iter 80 value 0.139250
iter 90 value 0.137396
iter 100 value 0.136824
final value 0.136824
stopped after 100 iterations
```

```

# weights: 11
initial value 281.422289
iter 10 value 110.864492
iter 20 value 107.892662
iter 30 value 107.793574
final value 107.792022
converged
# weights: 27
initial value 294.004814
iter 10 value 7.152681
iter 20 value 0.570344
iter 30 value 0.009365
final value 0.000078
converged
# weights: 43
initial value 372.854155
iter 10 value 16.951078
iter 20 value 1.170466
iter 30 value 0.094481
iter 40 value 0.001152
iter 50 value 0.000454
final value 0.000099
converged
# weights: 11
initial value 323.807293
iter 10 value 127.929079
iter 20 value 114.384472
iter 30 value 78.969184
iter 40 value 78.330489
iter 40 value 78.330489
final value 78.330489
converged
# weights: 27
initial value 304.462784
iter 10 value 70.473107
iter 20 value 24.749712
iter 30 value 23.123079
iter 40 value 22.960870
iter 50 value 22.929463
iter 60 value 22.901297
final value 22.901213
converged

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-6') unexpectedly generated random numbers without declaring so. There is a risk

that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```
# weights: 43
initial value 257.315959
iter 10 value 21.506243
iter 20 value 17.680555
iter 30 value 17.597338
iter 40 value 17.591603
final value 17.591422
converged
# weights: 11
initial value 292.836280
iter 10 value 85.267825
iter 20 value 42.121485
iter 30 value 28.750922
iter 40 value 26.930856
iter 50 value 26.743358
iter 60 value 26.740016
iter 70 value 26.727577
final value 26.727575
converged
# weights: 27
initial value 272.181288
iter 10 value 5.936592
iter 20 value 0.323960
iter 30 value 0.308244
iter 40 value 0.270032
iter 50 value 0.258502
iter 60 value 0.246359
iter 70 value 0.233169
iter 80 value 0.223426
iter 90 value 0.216960
iter 100 value 0.201660
final value 0.201660
stopped after 100 iterations
# weights: 43
initial value 386.121106
iter 10 value 5.933454
iter 20 value 0.417044
iter 30 value 0.283832
iter 40 value 0.247364
```

```

iter 50 value 0.225575
iter 60 value 0.209345
iter 70 value 0.200617
iter 80 value 0.189771
iter 90 value 0.183746
iter 100 value 0.176189
final value 0.176189
stopped after 100 iterations
# weights: 11
initial value 315.509218
iter 10 value 122.856031
iter 20 value 117.421919
iter 30 value 101.930617
iter 40 value 87.830893
iter 50 value 44.589306
iter 60 value 26.636211
iter 70 value 24.940361
iter 80 value 24.377979
iter 90 value 24.328062
iter 100 value 24.264775
final value 24.264775
stopped after 100 iterations
# weights: 27
initial value 314.983093
iter 10 value 29.690514
iter 20 value 1.496611
iter 30 value 0.110667
iter 40 value 0.004114
final value 0.000069
converged
# weights: 43
initial value 271.454505
iter 10 value 3.388963
iter 20 value 0.692067
iter 30 value 0.098483
iter 40 value 0.003811
iter 50 value 0.000859
iter 60 value 0.000202
final value 0.000087
converged
# weights: 11
initial value 297.452963
iter 10 value 137.008971
iter 20 value 119.852571
iter 30 value 86.133765
iter 40 value 76.958362

```

```
final  value 76.957595
converged
# weights: 27
initial  value 307.332860
iter   10 value 26.307013
iter   20 value 19.467084
iter   30 value 18.990646
iter   40 value 18.987559
final   value 18.987240
converged
# weights: 43
initial  value 281.225996
iter   10 value 43.138664
iter   20 value 19.756888
iter   30 value 18.267563
iter   40 value 17.547038
iter   50 value 17.437237
iter   60 value 17.354721
iter   70 value 17.150674
iter   80 value 17.138506
iter   90 value 17.137609
iter  100 value 17.137359
final   value 17.137359
stopped after 100 iterations
# weights: 11
initial  value 275.006698
iter   10 value 102.563739
iter   20 value 99.204887
iter   30 value 63.899867
iter   40 value 29.534443
iter   50 value 25.537049
iter   60 value 25.173728
iter   70 value 24.864366
iter   80 value 24.860998
iter   90 value 24.855394
final   value 24.854963
converged
# weights: 27
initial  value 341.185090
iter   10 value 4.265430
iter   20 value 0.311384
iter   30 value 0.281954
iter   40 value 0.266479
iter   50 value 0.243775
iter   60 value 0.237321
iter   70 value 0.222193
```

```
iter 80 value 0.198080
iter 90 value 0.182908
iter 100 value 0.179147
final value 0.179147
stopped after 100 iterations
# weights: 43
initial value 289.653422
iter 10 value 2.942434
iter 20 value 0.332172
iter 30 value 0.240508
iter 40 value 0.221273
iter 50 value 0.211264
iter 60 value 0.196515
iter 70 value 0.184329
iter 80 value 0.181464
iter 90 value 0.173919
iter 100 value 0.166115
final value 0.166115
stopped after 100 iterations
# weights: 11
initial value 288.529327
iter 10 value 106.015664
iter 20 value 105.238980
iter 30 value 98.760280
iter 40 value 62.751366
iter 50 value 47.669222
iter 60 value 45.488529
iter 70 value 44.276047
iter 80 value 44.238967
iter 90 value 44.231437
iter 100 value 44.175827
final value 44.175827
stopped after 100 iterations
# weights: 27
initial value 282.604530
iter 10 value 2.125677
iter 20 value 0.026488
iter 30 value 0.002207
iter 40 value 0.000822
iter 50 value 0.000560
iter 60 value 0.000490
iter 70 value 0.000107
final value 0.000099
converged
# weights: 43
initial value 296.442487
```

```

iter 10 value 7.208149
iter 20 value 0.077893
iter 30 value 0.003728
iter 40 value 0.000125
iter 40 value 0.000066
iter 40 value 0.000066
final value 0.000066
converged
# weights: 11
initial value 333.773986
iter 10 value 126.255836
iter 20 value 104.510827
iter 30 value 82.535378
iter 40 value 77.539794
iter 40 value 77.539794
final value 77.539794
converged
# weights: 27
initial value 303.883323
iter 10 value 53.640274
iter 20 value 24.127034
iter 30 value 22.906815
iter 40 value 22.782814
iter 50 value 22.757647
iter 60 value 22.756644
iter 70 value 22.755900
final value 22.755772
converged
# weights: 43
initial value 346.774449
iter 10 value 53.023539
iter 20 value 19.254602
iter 30 value 17.859075
iter 40 value 17.690398
iter 50 value 17.680928
iter 60 value 17.680686
iter 70 value 17.659830
iter 80 value 17.618702
iter 90 value 17.618341
final value 17.618325
converged
# weights: 11
initial value 319.227751
iter 10 value 123.694069
iter 20 value 106.288541
iter 30 value 102.893688

```

```
iter 40 value 102.620845
iter 50 value 102.422727
iter 60 value 101.085735
iter 70 value 99.801541
iter 80 value 96.691970
iter 90 value 50.082025
iter 100 value 27.771148
final value 27.771148
stopped after 100 iterations
# weights: 27
initial value 281.703895
iter 10 value 9.121412
iter 20 value 4.115824
iter 30 value 1.339053
iter 40 value 0.827759
iter 50 value 0.547221
iter 60 value 0.431069
iter 70 value 0.398761
iter 80 value 0.357993
iter 90 value 0.303484
iter 100 value 0.286888
final value 0.286888
stopped after 100 iterations
# weights: 43
initial value 331.347275
iter 10 value 2.980114
iter 20 value 0.295682
iter 30 value 0.277098
iter 40 value 0.252848
iter 50 value 0.232673
iter 60 value 0.215879
iter 70 value 0.176745
iter 80 value 0.170803
iter 90 value 0.167363
iter 100 value 0.165209
final value 0.165209
stopped after 100 iterations
# weights: 11
initial value 294.575960
iter 10 value 55.576283
iter 20 value 30.886960
iter 30 value 29.998338
iter 40 value 29.311045
iter 50 value 28.992506
iter 60 value 28.940783
iter 70 value 28.875615
```

```
iter 80 value 28.852096
iter 90 value 28.827342
iter 100 value 28.775875
final value 28.775875
stopped after 100 iterations
# weights: 27
initial value 303.329665
iter 10 value 3.897349
iter 20 value 0.089654
iter 30 value 0.000988
final value 0.000063
converged
# weights: 43
initial value 257.986743
iter 10 value 5.934914
iter 20 value 0.030152
final value 0.000084
converged
# weights: 11
initial value 279.530202
iter 10 value 146.430431
iter 20 value 108.259624
iter 30 value 103.848955
iter 40 value 103.100370
iter 40 value 103.100369
iter 40 value 103.100369
final value 103.100369
converged
# weights: 27
initial value 337.700116
iter 10 value 49.853279
iter 20 value 22.698243
iter 30 value 20.553244
iter 40 value 20.028240
iter 50 value 19.779189
iter 60 value 19.744164
final value 19.744038
converged
# weights: 43
initial value 400.333438
iter 10 value 24.603153
iter 20 value 20.162232
iter 30 value 18.177940
iter 40 value 17.913777
iter 50 value 17.890707
iter 60 value 17.879091
```

```
iter 70 value 17.877741
final value 17.877621
converged
# weights: 11
initial value 338.174763
iter 10 value 109.762148
iter 20 value 69.403542
iter 30 value 47.657879
iter 40 value 46.907227
iter 50 value 46.632845
iter 60 value 46.580451
iter 70 value 46.528049
iter 80 value 46.468330
iter 90 value 46.458114
iter 100 value 46.440573
final value 46.440573
stopped after 100 iterations
# weights: 27
initial value 304.774967
iter 10 value 2.490848
iter 20 value 0.692835
iter 30 value 0.615283
iter 40 value 0.460571
iter 50 value 0.356876
iter 60 value 0.289192
iter 70 value 0.273821
iter 80 value 0.246913
iter 90 value 0.221628
iter 100 value 0.197423
final value 0.197423
stopped after 100 iterations
# weights: 43
initial value 321.369930
iter 10 value 24.959005
iter 20 value 11.561977
iter 30 value 8.402564
iter 40 value 5.484829
iter 50 value 4.785946
iter 60 value 3.813876
iter 70 value 3.445040
iter 80 value 2.970477
iter 90 value 1.527461
iter 100 value 1.331493
final value 1.331493
stopped after 100 iterations
# weights: 11
```

```

initial value 299.480094
iter 10 value 101.015607
iter 20 value 58.347552
iter 30 value 13.830229
iter 40 value 13.459572
iter 50 value 12.700140
iter 60 value 12.684495
iter 70 value 12.583518
iter 80 value 12.321923
iter 90 value 12.245433
iter 100 value 12.118920
final value 12.118920
stopped after 100 iterations
# weights: 27
initial value 277.282244
iter 10 value 4.840801
iter 20 value 0.142730
iter 30 value 0.013406
iter 40 value 0.001508
iter 50 value 0.000188
iter 60 value 0.000137
final value 0.000096
converged
# weights: 43
initial value 287.723411
iter 10 value 0.831478
iter 20 value 0.041723
iter 30 value 0.014169
iter 40 value 0.000884
final value 0.000086
converged
# weights: 11
initial value 278.079186
iter 10 value 139.604594
iter 20 value 119.509232
iter 30 value 105.221647
iter 40 value 73.266626
final value 72.876963
converged
# weights: 27
initial value 365.215805
iter 10 value 34.677584
iter 20 value 18.729894
iter 30 value 17.997272
iter 40 value 17.899606
iter 50 value 17.890037

```

```
iter 60 value 17.889866
final value 17.889839
converged
# weights: 43
initial value 331.149095
iter 10 value 29.442909
iter 20 value 14.982021
iter 30 value 14.310767
iter 40 value 14.281172
iter 50 value 14.277901
final value 14.277880
converged
# weights: 11
initial value 375.715911
iter 10 value 109.096727
iter 20 value 94.338843
iter 30 value 87.981779
iter 40 value 58.739209
iter 50 value 18.713672
iter 60 value 14.896292
iter 70 value 13.903847
iter 80 value 13.734342
iter 90 value 13.722535
iter 100 value 13.701630
final value 13.701630
stopped after 100 iterations
# weights: 27
initial value 282.557001
iter 10 value 3.141693
iter 20 value 0.450288
iter 30 value 0.334144
iter 40 value 0.291430
iter 50 value 0.250390
iter 60 value 0.195359
iter 70 value 0.161061
iter 80 value 0.149677
iter 90 value 0.134087
iter 100 value 0.115182
final value 0.115182
stopped after 100 iterations
# weights: 43
initial value 293.570958
iter 10 value 4.453705
iter 20 value 0.114294
iter 30 value 0.103166
iter 40 value 0.083405
```

```

iter 50 value 0.078813
iter 60 value 0.070920
iter 70 value 0.068392
iter 80 value 0.066052
iter 90 value 0.064018
iter 100 value 0.062841
final value 0.062841
stopped after 100 iterations
# weights: 11
initial value 327.733069
iter 10 value 90.080008
iter 20 value 41.357874
iter 30 value 27.626502
iter 40 value 26.628653
iter 50 value 26.271846
iter 60 value 25.712213
iter 70 value 25.686610
iter 80 value 25.596833
iter 90 value 25.550467
iter 100 value 25.540683
final value 25.540683
stopped after 100 iterations
# weights: 27
initial value 436.293664
iter 10 value 0.827055
iter 20 value 0.039194
iter 30 value 0.001890
iter 40 value 0.000373
iter 50 value 0.000107
final value 0.000096
converged
# weights: 43
initial value 268.569505
iter 10 value 3.832771
iter 20 value 0.034619
iter 30 value 0.001093
iter 40 value 0.000152
iter 50 value 0.000112
final value 0.000099
converged

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-7') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead

of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```
# weights: 11
initial value 326.458712
iter 10 value 138.610932
iter 20 value 79.060756
iter 30 value 77.947425
final value 77.919416
converged
# weights: 27
initial value 335.689436
iter 10 value 36.348019
iter 20 value 20.062751
iter 30 value 18.796397
iter 40 value 18.669269
iter 50 value 18.646902
final value 18.646180
converged
# weights: 43
initial value 299.732583
iter 10 value 44.819969
iter 20 value 17.923418
iter 30 value 16.986955
iter 40 value 16.619029
iter 50 value 16.601187
iter 60 value 16.564008
iter 70 value 16.563714
final value 16.563712
converged
# weights: 11
initial value 354.880072
iter 10 value 109.751651
iter 20 value 108.012375
iter 30 value 107.977802
iter 40 value 107.660701
iter 50 value 106.244512
iter 60 value 105.727167
iter 70 value 105.652352
iter 80 value 104.670998
iter 90 value 96.785560
iter 100 value 33.541075
final value 33.541075
stopped after 100 iterations
```

```

# weights: 27
initial value 322.865551
iter 10 value 1.189190
iter 20 value 0.133816
iter 30 value 0.121645
iter 40 value 0.118619
iter 50 value 0.114955
iter 60 value 0.112324
iter 70 value 0.109967
iter 80 value 0.106637
iter 90 value 0.100991
iter 100 value 0.096775
final value 0.096775
stopped after 100 iterations
# weights: 43
initial value 331.065061
iter 10 value 1.635133
iter 20 value 0.109153
iter 30 value 0.090438
iter 40 value 0.087894
iter 50 value 0.084993
iter 60 value 0.083314
iter 70 value 0.082763
iter 80 value 0.081472
iter 90 value 0.080325
iter 100 value 0.078831
final value 0.078831
stopped after 100 iterations
# weights: 11
initial value 310.795199
iter 10 value 109.469805
iter 20 value 106.648406
final value 106.645226
converged
# weights: 27
initial value 346.456850
iter 10 value 6.078004
iter 20 value 0.205993
iter 30 value 0.016229
iter 40 value 0.001478
final value 0.000066
converged
# weights: 43
initial value 294.592273
iter 10 value 5.370949
iter 20 value 0.213633

```

```

iter 30 value 0.000830
final value 0.000054
converged
# weights: 11
initial value 323.579218
iter 10 value 119.142135
iter 20 value 96.923283
iter 30 value 78.193347
final value 77.990661
converged
# weights: 27
initial value 314.293469
iter 10 value 26.649576
iter 20 value 19.354688
iter 30 value 19.278675
iter 40 value 19.268732
final value 19.268712
converged
# weights: 43
initial value 287.836513
iter 10 value 24.392473
iter 20 value 18.628125
iter 30 value 17.738117
iter 40 value 17.629334
iter 50 value 17.584281
iter 60 value 17.561473
iter 70 value 17.557395
iter 80 value 17.557028
final value 17.557022
converged
# weights: 11
initial value 332.551277
iter 10 value 103.535057
iter 20 value 100.621575
iter 30 value 100.183314
iter 40 value 52.061690
iter 50 value 30.426460
iter 60 value 28.928670
iter 70 value 27.489516
iter 80 value 27.200540
iter 90 value 27.143511
iter 100 value 27.114996
final value 27.114996
stopped after 100 iterations
# weights: 27
initial value 311.908526

```

```

iter 10 value 22.389766
iter 20 value 1.230930
iter 30 value 0.295508
iter 40 value 0.262217
iter 50 value 0.254498
iter 60 value 0.210151
iter 70 value 0.197361
iter 80 value 0.189185
iter 90 value 0.186124
iter 100 value 0.183307
final value 0.183307
stopped after 100 iterations
# weights: 43
initial value 311.224636
iter 10 value 3.638302
iter 20 value 0.441538
iter 30 value 0.251428
iter 40 value 0.229275
iter 50 value 0.224388
iter 60 value 0.213638
iter 70 value 0.198449
iter 80 value 0.191524
iter 90 value 0.185394
iter 100 value 0.182627
final value 0.182627
stopped after 100 iterations
# weights: 11
initial value 307.126747
iter 10 value 108.898033
iter 20 value 107.799554
final value 107.791934
converged
# weights: 27
initial value 296.194083
iter 10 value 14.387518
iter 20 value 1.759381
iter 30 value 0.117803
iter 40 value 0.001157
final value 0.000092
converged
# weights: 43
initial value 415.055644
iter 10 value 8.995038
iter 20 value 0.785111
iter 30 value 0.027403
iter 40 value 0.003373

```

```
iter 50 value 0.000428
iter 60 value 0.000257
iter 70 value 0.000130
iter 70 value 0.000075
iter 70 value 0.000075
final value 0.000075
converged
# weights: 11
initial value 282.276749
iter 10 value 129.905445
iter 20 value 110.135998
iter 30 value 80.901396
iter 40 value 79.346901
final value 79.346897
converged
# weights: 27
initial value 409.958398
iter 10 value 52.900073
iter 20 value 23.210363
iter 30 value 20.904238
iter 40 value 20.613154
iter 50 value 19.982423
iter 60 value 19.710074
iter 70 value 19.642048
final value 19.642003
converged
# weights: 43
initial value 304.584161
iter 10 value 44.114866
iter 20 value 18.677661
iter 30 value 18.486842
iter 40 value 18.462421
iter 50 value 18.400012
iter 60 value 18.095560
iter 70 value 17.989697
iter 80 value 17.989487
iter 80 value 17.989487
iter 80 value 17.989487
final value 17.989487
converged
# weights: 11
initial value 312.988230
iter 10 value 92.804005
iter 20 value 40.313549
iter 30 value 28.340542
iter 40 value 27.619313
```

```

iter 50 value 26.889221
iter 60 value 26.759653
iter 70 value 26.679125
iter 80 value 26.646761
iter 90 value 26.645537
final value 26.645192
converged
# weights: 27
initial value 351.785618
iter 10 value 4.501361
iter 20 value 0.878673
iter 30 value 0.512805
iter 40 value 0.454486
iter 50 value 0.369984
iter 60 value 0.307396
iter 70 value 0.292561
iter 80 value 0.272104
iter 90 value 0.227844
iter 100 value 0.214957
final value 0.214957
stopped after 100 iterations
# weights: 43
initial value 324.850992
iter 10 value 6.508858
iter 20 value 0.343558
iter 30 value 0.235294
iter 40 value 0.221084
iter 50 value 0.212018
iter 60 value 0.203195
iter 70 value 0.198359
iter 80 value 0.191784
iter 90 value 0.187566
iter 100 value 0.181946
final value 0.181946
stopped after 100 iterations
# weights: 11
initial value 320.523263
iter 10 value 157.155130
iter 20 value 143.497314
iter 30 value 101.622383
iter 40 value 47.655422
iter 50 value 46.618042
iter 60 value 46.202378
iter 70 value 45.893321
iter 80 value 45.879275
iter 90 value 45.860057

```

```
iter 100 value 45.852040
final  value 45.852040
stopped after 100 iterations
# weights: 27
initial  value 328.212430
iter   10 value 3.839921
iter   20 value 0.152308
iter   30 value 0.008922
iter   40 value 0.000745
iter   50 value 0.000494
final  value 0.000085
converged
# weights: 43
initial  value 301.575561
iter   10 value 0.335532
iter   20 value 0.006517
iter   30 value 0.003326
iter   40 value 0.001631
iter   50 value 0.000559
iter   60 value 0.000458
final  value 0.000087
converged
# weights: 11
initial  value 310.166212
iter   10 value 121.032945
iter   20 value 116.090013
iter   30 value 103.153170
final  value 102.144185
converged
# weights: 27
initial  value 329.774869
iter   10 value 36.002871
iter   20 value 23.915464
iter   30 value 18.921771
iter   40 value 18.390093
iter   50 value 18.325950
iter   60 value 18.319342
iter   70 value 18.318430
final  value 18.318429
converged
# weights: 43
initial  value 318.952840
iter   10 value 36.256469
iter   20 value 18.068442
iter   30 value 17.100204
iter   40 value 16.840456
```

```

iter 50 value 16.559290
iter 60 value 16.488379
iter 70 value 16.476007
iter 80 value 16.474233
final value 16.473981
converged
# weights: 11
initial value 293.742366
iter 10 value 108.110443
iter 20 value 108.072131
iter 30 value 107.901373
iter 40 value 107.893783
iter 50 value 107.858130
iter 60 value 107.021852
iter 70 value 99.527756
iter 80 value 42.871972
iter 90 value 24.085736
iter 100 value 20.896387
final value 20.896387
stopped after 100 iterations
# weights: 27
initial value 344.503364
iter 10 value 1.412060
iter 20 value 0.158767
iter 30 value 0.139325
iter 40 value 0.127041
iter 50 value 0.121220
iter 60 value 0.116720
iter 70 value 0.112853
iter 80 value 0.108787
iter 90 value 0.104060
iter 100 value 0.102239
final value 0.102239
stopped after 100 iterations
# weights: 43
initial value 397.730589
iter 10 value 4.058229
iter 20 value 0.145368
iter 30 value 0.137121
iter 40 value 0.119258
iter 50 value 0.106247
iter 60 value 0.094638
iter 70 value 0.088931
iter 80 value 0.084093
iter 90 value 0.083105
iter 100 value 0.082648

```

```

final value 0.082648
stopped after 100 iterations
# weights: 11
initial value 305.001450
iter 10 value 107.506574
iter 20 value 106.255827
iter 30 value 106.204400
iter 40 value 105.384665
iter 50 value 103.352874
iter 60 value 96.092090
iter 70 value 57.635899
iter 80 value 46.491833
iter 90 value 44.746202
iter 100 value 44.199578
final value 44.199578
stopped after 100 iterations
# weights: 27
initial value 306.606256
iter 10 value 1.828698
iter 20 value 0.039825
iter 30 value 0.002083
final value 0.000057
converged
# weights: 43
initial value 321.956592
iter 10 value 3.584171
iter 20 value 0.104881
iter 30 value 0.001897
iter 40 value 0.000649
final value 0.000075
converged
# weights: 11
initial value 315.826407
iter 10 value 136.131770
iter 20 value 117.615421
iter 30 value 105.498221
iter 40 value 103.016700
final value 103.016606
converged
# weights: 27
initial value 264.666910
iter 10 value 27.456994
iter 20 value 20.089709
iter 30 value 19.261931
iter 40 value 19.250181
final value 19.249682

```

```

converged
# weights: 43
initial value 301.580443
iter 10 value 25.011337
iter 20 value 17.409692
iter 30 value 16.456266
iter 40 value 16.402484
iter 50 value 16.376619
iter 60 value 16.374066
final value 16.373233
converged
# weights: 11
initial value 282.549261
iter 10 value 108.648288
iter 20 value 104.085047
iter 30 value 103.643606
iter 40 value 103.624077
iter 50 value 103.583748
iter 60 value 103.566843
iter 70 value 103.546017
iter 80 value 103.533748
iter 90 value 103.532857
iter 100 value 103.532412
final value 103.532412
stopped after 100 iterations
# weights: 27
initial value 302.613673
iter 10 value 5.520238
iter 20 value 1.252874
iter 30 value 0.483212
iter 40 value 0.265911
iter 50 value 0.243019
iter 60 value 0.232583
iter 70 value 0.200103
iter 80 value 0.178702
iter 90 value 0.169410
iter 100 value 0.155513
final value 0.155513
stopped after 100 iterations
# weights: 43
initial value 337.290129
iter 10 value 2.704863
iter 20 value 0.240629
iter 30 value 0.188165
iter 40 value 0.174355
iter 50 value 0.169063

```

```

iter 60 value 0.160928
iter 70 value 0.154507
iter 80 value 0.148375
iter 90 value 0.144067
iter 100 value 0.142056
final value 0.142056
stopped after 100 iterations
# weights: 11
initial value 367.438053
iter 10 value 152.585615
iter 20 value 104.909722
iter 30 value 103.575723
iter 40 value 103.472990
iter 50 value 103.390186
iter 60 value 103.314934
iter 70 value 103.146047
iter 80 value 103.145182
final value 103.144942
converged

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-8') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```

# weights: 27
initial value 263.060269
iter 10 value 11.654575
iter 20 value 0.691993
iter 30 value 0.011297
iter 40 value 0.000736
iter 50 value 0.000464
final value 0.000087
converged
# weights: 43
initial value 314.124494
iter 10 value 17.332869
iter 20 value 0.000274
final value 0.000086
converged
# weights: 11
initial value 270.757320

```

```

iter 10 value 120.046449
iter 20 value 117.072128
iter 30 value 103.382432
iter 40 value 100.206089
final value 100.206086
converged
# weights: 27
initial value 354.184435
iter 10 value 36.359366
iter 20 value 27.353269
iter 30 value 23.769282
iter 40 value 20.819686
iter 50 value 20.447910
iter 60 value 20.433949
iter 70 value 20.433413
iter 70 value 20.433413
iter 70 value 20.433413
final value 20.433413
converged
# weights: 43
initial value 354.208022
iter 10 value 26.610346
iter 20 value 17.302071
iter 30 value 16.667557
iter 40 value 16.515323
iter 50 value 16.469485
iter 60 value 16.462604
final value 16.462585
converged
# weights: 11
initial value 287.794180
iter 10 value 52.306884
iter 20 value 25.784407
iter 30 value 24.491297
iter 40 value 24.083218
iter 50 value 24.050640
iter 60 value 24.033551
iter 70 value 24.026152
iter 70 value 24.026152
final value 24.026152
converged
# weights: 27
initial value 285.175515
iter 10 value 6.998953
iter 20 value 0.588123
iter 30 value 0.306919

```

```

iter 40 value 0.218787
iter 50 value 0.174946
iter 60 value 0.168192
iter 70 value 0.158896
iter 80 value 0.149971
iter 90 value 0.144086
iter 100 value 0.136923
final value 0.136923
stopped after 100 iterations
# weights: 43
initial value 367.511072
iter 10 value 4.152061
iter 20 value 0.189889
iter 30 value 0.174786
iter 40 value 0.162409
iter 50 value 0.129455
iter 60 value 0.116614
iter 70 value 0.104144
iter 80 value 0.101378
iter 90 value 0.099115
iter 100 value 0.095474
final value 0.095474
stopped after 100 iterations
# weights: 11
initial value 295.765349
iter 10 value 160.856483
iter 20 value 150.795523
iter 30 value 106.279439
iter 40 value 101.510885
iter 50 value 100.947555
iter 60 value 100.225510
iter 70 value 100.180917
iter 80 value 100.179635
final value 100.178984
converged
# weights: 27
initial value 340.050898
iter 10 value 3.840556
iter 20 value 0.195038
iter 30 value 0.011652
iter 40 value 0.000481
final value 0.000071
converged
# weights: 43
initial value 330.167443
iter 10 value 4.274540

```

```

iter 20 value 0.609652
iter 30 value 0.013371
iter 40 value 0.001800
iter 50 value 0.000619
iter 60 value 0.000260
iter 70 value 0.000228
iter 80 value 0.000185
iter 90 value 0.000148
final value 0.000090
converged
# weights: 11
initial value 280.972981
iter 10 value 150.099833
iter 20 value 117.186637
iter 30 value 96.995538
iter 40 value 78.622655
final value 78.580634
converged
# weights: 27
initial value 360.392345
iter 10 value 49.610486
iter 20 value 26.080806
iter 30 value 21.139867
iter 40 value 20.373210
iter 50 value 19.493266
iter 60 value 19.322710
iter 70 value 19.207008
final value 19.207004
converged
# weights: 43
initial value 295.443195
iter 10 value 24.296215
iter 20 value 17.711744
iter 30 value 17.217567
iter 40 value 17.111411
iter 50 value 17.085778
iter 60 value 17.084616
iter 70 value 17.084373
final value 17.084370
converged
# weights: 11
initial value 296.701618
iter 10 value 101.408114
iter 20 value 93.737382
iter 30 value 52.355702
iter 40 value 31.027555

```

```

iter 50 value 29.724710
iter 60 value 28.739650
iter 70 value 28.531327
iter 80 value 28.515910
iter 90 value 28.481150
iter 100 value 28.480931
final value 28.480931
stopped after 100 iterations
# weights: 27
initial value 355.597159
iter 10 value 9.314752
iter 20 value 0.881473
iter 30 value 0.333668
iter 40 value 0.305670
iter 50 value 0.281566
iter 60 value 0.258334
iter 70 value 0.250207
iter 80 value 0.239539
iter 90 value 0.226408
iter 100 value 0.222628
final value 0.222628
stopped after 100 iterations
# weights: 43
initial value 440.578217
iter 10 value 5.525666
iter 20 value 0.244235
iter 30 value 0.204430
iter 40 value 0.190062
iter 50 value 0.180871
iter 60 value 0.178663
iter 70 value 0.177418
iter 80 value 0.174154
iter 90 value 0.172496
iter 100 value 0.170767
final value 0.170767
stopped after 100 iterations
# weights: 11
initial value 344.583997
iter 10 value 109.414413
iter 20 value 107.794241
final value 107.791554
converged
# weights: 27
initial value 329.721708
iter 10 value 1.664972
iter 20 value 0.423721

```

```
iter 30 value 0.006960
final value 0.000081
converged
# weights: 43
initial value 339.623382
iter 10 value 2.504656
iter 20 value 0.231892
iter 30 value 0.018287
iter 40 value 0.006162
iter 50 value 0.001476
iter 60 value 0.000188
final value 0.000097
converged
# weights: 11
initial value 307.346742
iter 10 value 136.078469
iter 20 value 117.664333
iter 30 value 78.812763
iter 40 value 77.181072
iter 40 value 77.181072
iter 40 value 77.181072
final value 77.181072
converged
# weights: 27
initial value 306.828681
iter 10 value 38.269115
iter 20 value 19.459720
iter 30 value 18.913847
iter 40 value 18.803329
iter 50 value 18.801683
final value 18.801678
converged
# weights: 43
initial value 380.134555
iter 10 value 33.164705
iter 20 value 18.337175
iter 30 value 17.270709
iter 40 value 16.945681
iter 50 value 16.898348
iter 60 value 16.891624
iter 70 value 16.889942
final value 16.889935
converged
# weights: 11
initial value 313.587169
iter 10 value 120.574440
```

```

iter 20 value 101.472243
iter 30 value 97.691113
iter 40 value 60.929672
iter 50 value 27.269147
iter 60 value 24.304890
iter 70 value 24.283174
iter 80 value 24.250593
iter 90 value 24.250123
iter 100 value 24.247130
final value 24.247130
stopped after 100 iterations
# weights: 27
initial value 275.862079
iter 10 value 2.175684
iter 20 value 0.283600
iter 30 value 0.229805
iter 40 value 0.226189
iter 50 value 0.223441
iter 60 value 0.220661
iter 70 value 0.207815
iter 80 value 0.201094
iter 90 value 0.197419
iter 100 value 0.186653
final value 0.186653
stopped after 100 iterations
# weights: 43
initial value 296.608984
iter 10 value 1.936322
iter 20 value 0.211435
iter 30 value 0.195478
iter 40 value 0.175245
iter 50 value 0.169602
iter 60 value 0.165052
iter 70 value 0.162243
iter 80 value 0.158878
iter 90 value 0.156171
iter 100 value 0.155358
final value 0.155358
stopped after 100 iterations
# weights: 11
initial value 316.538208
iter 10 value 139.676612
iter 20 value 108.234264
iter 30 value 107.853391
iter 40 value 107.834740
iter 50 value 107.377766

```

```

iter 60 value 106.411608
iter 70 value 104.173472
iter 80 value 103.858644
iter 90 value 103.373731
iter 100 value 103.068498
final value 103.068498
stopped after 100 iterations
# weights: 27
initial value 290.197652
iter 10 value 11.627612
iter 20 value 0.765299
iter 30 value 0.080360
iter 40 value 0.003029
iter 50 value 0.001547
final value 0.000063
converged
# weights: 43
initial value 336.501505
iter 10 value 4.319904
iter 20 value 0.116092
iter 30 value 0.003870
iter 40 value 0.000635
iter 50 value 0.000473
iter 60 value 0.000169
final value 0.000077
converged
# weights: 11
initial value 370.915756
iter 10 value 188.546433
iter 20 value 167.560280
iter 30 value 105.445753
iter 40 value 100.713053
final value 100.712964
converged
# weights: 27
initial value 299.664115
iter 10 value 27.183968
iter 20 value 20.952004
iter 30 value 19.660246
iter 40 value 19.651273
iter 50 value 19.650841
final value 19.650826
converged
# weights: 43
initial value 288.966143
iter 10 value 21.047419

```

```

iter 20 value 18.370843
iter 30 value 17.797466
iter 40 value 17.724948
iter 50 value 17.706835
iter 60 value 17.706654
final value 17.706653
converged
# weights: 11
initial value 305.539641
iter 10 value 114.884504
iter 20 value 58.694178
iter 30 value 43.445194
iter 40 value 41.609316
iter 50 value 41.572305
iter 60 value 41.550936
iter 70 value 41.505632
final value 41.505626
converged
# weights: 27
initial value 388.608553
iter 10 value 1.978515
iter 20 value 0.286334
iter 30 value 0.248373
iter 40 value 0.237197
iter 50 value 0.228027
iter 60 value 0.221704
iter 70 value 0.218969
iter 80 value 0.212479
iter 90 value 0.196678
iter 100 value 0.189369
final value 0.189369
stopped after 100 iterations
# weights: 43
initial value 338.317129
iter 10 value 1.974666
iter 20 value 0.213858
iter 30 value 0.185690
iter 40 value 0.179136
iter 50 value 0.172391
iter 60 value 0.168362
iter 70 value 0.167131
iter 80 value 0.166181
iter 90 value 0.165883
iter 100 value 0.165387
final value 0.165387
stopped after 100 iterations

```

```

# weights: 11
initial value 283.753634
iter 10 value 58.395746
iter 20 value 27.619002
iter 30 value 26.201209
iter 40 value 25.982493
iter 50 value 25.586700
iter 60 value 25.463962
iter 70 value 25.429065
iter 80 value 25.376041
iter 90 value 25.357474
iter 100 value 25.315026
final value 25.315026
stopped after 100 iterations
# weights: 27
initial value 347.453513
iter 10 value 0.488603
iter 20 value 0.062665
iter 30 value 0.007603
iter 40 value 0.001034
final value 0.000075
converged
# weights: 43
initial value 299.375764
iter 10 value 1.361789
iter 20 value 0.069428
iter 30 value 0.012561
iter 40 value 0.000829
iter 50 value 0.000230
final value 0.000093
converged
# weights: 11
initial value 294.339918
iter 10 value 109.814465
iter 20 value 100.741371
iter 30 value 100.662272
final value 100.661039
converged
# weights: 27
initial value 324.885653
iter 10 value 44.739454
iter 20 value 22.989858
iter 30 value 21.579666
iter 40 value 21.522823
iter 50 value 21.509074
final value 21.506871

```

```

converged
# weights: 43
initial value 329.101472
iter 10 value 38.843254
iter 20 value 18.627334
iter 30 value 17.575720
iter 40 value 17.545647
iter 50 value 17.538092
iter 60 value 17.520350
iter 70 value 17.516630
final value 17.515895
converged
# weights: 11
initial value 254.408504
iter 10 value 83.301355
iter 20 value 32.399100
iter 30 value 27.277053
iter 40 value 26.745525
iter 50 value 26.210473
iter 60 value 26.162043
iter 70 value 26.156541
final value 26.151620
converged
# weights: 27
initial value 326.422457
iter 10 value 8.873838
iter 20 value 0.905084
iter 30 value 0.396778
iter 40 value 0.354436
iter 50 value 0.313345
iter 60 value 0.289992
iter 70 value 0.264598
iter 80 value 0.248761
iter 90 value 0.237036
iter 100 value 0.226844
final value 0.226844
stopped after 100 iterations

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-9') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```

# weights: 43
initial value 335.773826
iter 10 value 8.930973
iter 20 value 0.525916
iter 30 value 0.379426
iter 40 value 0.362142
iter 50 value 0.285095
iter 60 value 0.251433
iter 70 value 0.239135
iter 80 value 0.198224
iter 90 value 0.193797
iter 100 value 0.186678
final value 0.186678
stopped after 100 iterations
# weights: 11
initial value 302.844632
iter 10 value 66.107343
iter 20 value 19.539776
iter 30 value 17.455113
iter 40 value 16.807247
iter 50 value 16.574339
iter 60 value 16.546524
iter 70 value 16.500367
iter 80 value 16.432889
iter 90 value 16.404781
iter 100 value 16.359462
final value 16.359462
stopped after 100 iterations
# weights: 27
initial value 280.723708
iter 10 value 3.236619
iter 20 value 0.059268
iter 30 value 0.010897
iter 40 value 0.004234
iter 50 value 0.001397
final value 0.000066
converged
# weights: 43
initial value 262.791536
iter 10 value 0.378623
iter 20 value 0.010093
iter 30 value 0.003127
iter 40 value 0.000736
iter 50 value 0.000132
final value 0.000096
converged

```

```

# weights: 11
initial value 307.396600
iter 10 value 152.069445
iter 20 value 83.393222
iter 30 value 77.331006
final value 76.858644
converged
# weights: 27
initial value 366.900772
iter 10 value 70.634613
iter 20 value 23.022831
iter 30 value 20.107737
iter 40 value 19.426080
iter 50 value 18.697747
iter 60 value 18.525422
iter 70 value 18.524874
final value 18.524874
converged
# weights: 43
initial value 306.014433
iter 10 value 25.521272
iter 20 value 18.006565
iter 30 value 16.575184
iter 40 value 16.263755
iter 50 value 16.246351
iter 60 value 16.244975
iter 70 value 16.244774
iter 80 value 16.244680
iter 90 value 16.244468
iter 100 value 16.243239
final value 16.243239
stopped after 100 iterations
# weights: 11
initial value 304.248504
iter 10 value 172.213617
iter 20 value 111.989780
iter 30 value 107.202181
iter 40 value 106.726697
iter 50 value 105.474487
iter 60 value 104.977102
iter 70 value 104.663257
iter 80 value 104.400948
iter 90 value 82.015021
iter 100 value 50.137731
final value 50.137731
stopped after 100 iterations

```

```

# weights: 27
initial value 357.920603
iter 10 value 1.587770
iter 20 value 0.307681
iter 30 value 0.247599
iter 40 value 0.179776
iter 50 value 0.163005
iter 60 value 0.132568
iter 70 value 0.123644
iter 80 value 0.118700
iter 90 value 0.106064
iter 100 value 0.100219
final value 0.100219
stopped after 100 iterations
# weights: 43
initial value 332.367486
iter 10 value 5.069094
iter 20 value 0.267059
iter 30 value 0.254979
iter 40 value 0.214856
iter 50 value 0.173208
iter 60 value 0.135601
iter 70 value 0.123789
iter 80 value 0.112098
iter 90 value 0.105142
iter 100 value 0.102302
final value 0.102302
stopped after 100 iterations
# weights: 11
initial value 294.938692
iter 10 value 106.041452
iter 20 value 99.266790
iter 30 value 77.071263
iter 40 value 43.741435
iter 50 value 37.778419
iter 60 value 36.742030
iter 70 value 35.057703
iter 80 value 33.950256
iter 90 value 33.894833
iter 100 value 33.744692
final value 33.744692
stopped after 100 iterations
# weights: 27
initial value 323.180715
iter 10 value 2.160571
iter 20 value 0.185055

```

```
iter 30 value 0.002213
iter 40 value 0.000230
iter 50 value 0.000112
final value 0.000098
converged
# weights: 43
initial value 252.439263
iter 10 value 2.010781
iter 20 value 0.062443
iter 30 value 0.000977
final value 0.000082
converged
# weights: 11
initial value 294.034202
iter 10 value 119.338126
iter 20 value 106.116901
iter 30 value 76.909525
final value 76.809652
converged
# weights: 27
initial value 264.623608
iter 10 value 36.293158
iter 20 value 20.646366
iter 30 value 19.858183
iter 40 value 19.812006
iter 50 value 19.775903
iter 50 value 19.775903
iter 50 value 19.775903
final value 19.775903
converged
# weights: 43
initial value 366.827432
iter 10 value 41.170713
iter 20 value 16.462891
iter 30 value 16.138821
iter 40 value 16.072132
iter 50 value 16.059725
iter 60 value 16.056921
final value 16.056692
converged
# weights: 11
initial value 273.523573
iter 10 value 107.231220
iter 20 value 101.421251
iter 30 value 96.194380
iter 40 value 88.715403
```

```
iter 50 value 51.865926
iter 60 value 33.417508
iter 70 value 26.684193
iter 80 value 25.952411
iter 90 value 25.873795
iter 100 value 25.792614
final value 25.792614
stopped after 100 iterations
# weights: 27
initial value 293.635902
iter 10 value 1.183159
iter 20 value 0.226349
iter 30 value 0.192849
iter 40 value 0.185660
iter 50 value 0.177194
iter 60 value 0.175670
iter 70 value 0.170693
iter 80 value 0.154074
iter 90 value 0.149282
iter 100 value 0.144934
final value 0.144934
stopped after 100 iterations
# weights: 43
initial value 313.564384
iter 10 value 1.282807
iter 20 value 0.306880
iter 30 value 0.186129
iter 40 value 0.174151
iter 50 value 0.161960
iter 60 value 0.158374
iter 70 value 0.152511
iter 80 value 0.146492
iter 90 value 0.140270
iter 100 value 0.133351
final value 0.133351
stopped after 100 iterations
# weights: 11
initial value 324.597315
iter 10 value 108.094336
iter 20 value 107.638790
iter 30 value 107.048223
iter 40 value 106.269353
iter 50 value 106.198331
iter 60 value 105.931381
iter 70 value 105.712486
iter 80 value 104.496886
```

```
iter 90 value 102.677868
iter 100 value 93.433416
final value 93.433416
stopped after 100 iterations
# weights: 27
initial value 302.365857
iter 10 value 3.630712
iter 20 value 0.250559
iter 30 value 0.000891
iter 40 value 0.000250
final value 0.000077
converged
# weights: 43
initial value 349.818243
iter 10 value 1.575202
iter 20 value 0.137140
iter 30 value 0.004769
iter 40 value 0.000234
iter 50 value 0.000123
final value 0.000095
converged
# weights: 11
initial value 308.989372
iter 10 value 134.142905
iter 20 value 120.900282
iter 30 value 115.949730
iter 40 value 103.226914
final value 103.162489
converged
# weights: 27
initial value 425.357589
iter 10 value 33.066827
iter 20 value 18.627847
iter 30 value 18.182035
iter 40 value 17.937048
iter 50 value 17.921912
final value 17.921871
converged
# weights: 43
initial value 385.457935
iter 10 value 31.662847
iter 20 value 17.640535
iter 30 value 16.250623
iter 40 value 16.060623
iter 50 value 16.051062
iter 60 value 16.049851
```

```
iter 70 value 16.049829
final value 16.049828
converged
# weights: 11
initial value 305.569245
iter 10 value 108.161379
iter 20 value 103.993324
iter 30 value 99.636314
iter 40 value 88.205960
iter 50 value 48.958406
iter 60 value 28.845180
iter 70 value 26.774368
iter 80 value 26.036339
iter 90 value 26.005498
iter 100 value 25.992697
final value 25.992697
stopped after 100 iterations
# weights: 27
initial value 294.553785
iter 10 value 3.381727
iter 20 value 0.319415
iter 30 value 0.304127
iter 40 value 0.291446
iter 50 value 0.278693
iter 60 value 0.241122
iter 70 value 0.228524
iter 80 value 0.215535
iter 90 value 0.195465
iter 100 value 0.185819
final value 0.185819
stopped after 100 iterations
# weights: 43
initial value 318.336848
iter 10 value 2.164418
iter 20 value 0.653980
iter 30 value 0.276656
iter 40 value 0.252817
iter 50 value 0.238520
iter 60 value 0.220666
iter 70 value 0.211282
iter 80 value 0.203784
iter 90 value 0.195554
iter 100 value 0.191662
final value 0.191662
stopped after 100 iterations
# weights: 11
```

```
initial value 289.567716
iter 10 value 108.983502
iter 20 value 104.099382
iter 30 value 103.171737
iter 40 value 103.129923
final value 103.128066
converged
# weights: 27
initial value 281.632196
iter 10 value 1.870810
iter 20 value 0.037819
iter 30 value 0.002751
final value 0.000076
converged
# weights: 43
initial value 276.443815
iter 10 value 1.933538
iter 20 value 0.031278
iter 30 value 0.000183
final value 0.000076
converged
# weights: 11
initial value 357.861952
iter 10 value 117.384529
iter 20 value 100.654381
iter 30 value 78.855452
final value 78.278421
converged
# weights: 27
initial value 334.321320
iter 10 value 44.089525
iter 20 value 20.543402
iter 30 value 20.527716
final value 20.527716
converged
# weights: 43
initial value 308.593217
iter 10 value 58.544522
iter 20 value 17.841543
iter 30 value 16.181031
iter 40 value 16.127065
iter 50 value 16.113466
iter 60 value 16.110181
iter 70 value 16.109565
iter 80 value 16.109273
iter 90 value 16.109250
```

```
final  value 16.109247
converged
# weights: 11
initial  value 326.333484
iter   10 value 113.769262
iter   20 value 103.174785
iter   30 value 64.078099
iter   40 value 30.643653
iter   50 value 29.209267
iter   60 value 28.255238
iter   70 value 28.093638
iter   80 value 28.089719
iter   90 value 28.085281
final  value 28.085265
converged
# weights: 27
initial  value 342.051413
iter   10 value 2.687219
iter   20 value 0.177764
iter   30 value 0.171387
iter   40 value 0.165404
iter   50 value 0.144193
iter   60 value 0.134019
iter   70 value 0.127145
iter   80 value 0.120006
iter   90 value 0.111261
iter  100 value 0.098144
final  value 0.098144
stopped after 100 iterations
# weights: 43
initial  value 470.641643
iter   10 value 16.479406
iter   20 value 3.391813
iter   30 value 0.523555
iter   40 value 0.425504
iter   50 value 0.307519
iter   60 value 0.258069
iter   70 value 0.237048
iter   80 value 0.195559
iter   90 value 0.164843
iter  100 value 0.137971
final  value 0.137971
stopped after 100 iterations
# weights: 11
initial  value 284.808532
iter   10 value 89.688375
```

```
iter 20 value 29.265940
iter 30 value 22.167474
iter 40 value 21.905341
iter 50 value 21.819854
iter 60 value 21.748135
iter 70 value 21.661241
iter 80 value 21.563180
iter 90 value 21.487684
iter 100 value 21.475251
final value 21.475251
stopped after 100 iterations
# weights: 27
initial value 378.627286
iter 10 value 4.616471
iter 20 value 0.097339
iter 30 value 0.005239
iter 40 value 0.000479
iter 50 value 0.000304
final value 0.000099
converged
# weights: 43
initial value 308.483220
iter 10 value 5.566472
iter 20 value 0.010587
iter 30 value 0.000540
iter 40 value 0.000203
final value 0.000094
converged
# weights: 11
initial value 327.154534
iter 10 value 146.811371
iter 20 value 90.765198
iter 30 value 76.743438
iter 40 value 76.737498
final value 76.737463
converged
# weights: 27
initial value 306.387636
iter 10 value 29.595055
iter 20 value 20.099884
iter 30 value 18.980526
iter 40 value 18.833064
iter 50 value 18.815328
iter 60 value 18.810463
final value 18.810449
converged
```

```

# weights: 43
initial value 269.508760
iter 10 value 35.823738
iter 20 value 20.054294
iter 30 value 17.645462
iter 40 value 17.383527
iter 50 value 17.069304
iter 60 value 17.051326
iter 70 value 17.050567
final value 17.050438
converged

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-10') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```

# weights: 11
initial value 305.802934
iter 10 value 104.247219
iter 20 value 70.337664
iter 30 value 51.304753
iter 40 value 48.550849
iter 50 value 48.322206
iter 60 value 48.311385
final value 48.297152
converged
# weights: 27
initial value 283.330003
iter 10 value 5.093628
iter 20 value 0.249361
iter 30 value 0.187176
iter 40 value 0.169147
iter 50 value 0.158273
iter 60 value 0.148767
iter 70 value 0.142296
iter 80 value 0.134757
iter 90 value 0.122490
iter 100 value 0.114107
final value 0.114107
stopped after 100 iterations
# weights: 43

```

```
initial value 304.146632
iter 10 value 1.756075
iter 20 value 0.172236
iter 30 value 0.121462
iter 40 value 0.116950
iter 50 value 0.113497
iter 60 value 0.111361
iter 70 value 0.110110
iter 80 value 0.105367
iter 90 value 0.100438
iter 100 value 0.098386
final value 0.098386
stopped after 100 iterations
# weights: 11
initial value 354.724204
iter 10 value 111.224405
iter 20 value 103.797787
iter 30 value 103.566206
iter 40 value 102.673288
iter 50 value 96.265445
iter 60 value 95.900564
iter 70 value 95.040152
iter 80 value 94.505308
iter 90 value 94.484282
iter 100 value 94.481073
final value 94.481073
stopped after 100 iterations
# weights: 27
initial value 340.530597
iter 10 value 2.169106
iter 20 value 0.206914
iter 30 value 0.023357
final value 0.000089
converged
# weights: 43
initial value 304.450313
iter 10 value 3.823344
iter 20 value 0.050616
iter 30 value 0.000236
final value 0.000082
converged
# weights: 11
initial value 350.047163
iter 10 value 127.430025
iter 20 value 104.068990
iter 30 value 75.598920
```

```
final  value 75.033195
converged
# weights: 27
initial  value 293.728531
iter   10 value 45.993149
iter   20 value 22.755351
iter   30 value 18.929313
iter   40 value 18.824572
iter   50 value 18.821484
final  value 18.821464
converged
# weights: 43
initial  value 310.692505
iter   10 value 26.997628
iter   20 value 19.397870
iter   30 value 17.614848
iter   40 value 17.214132
iter   50 value 17.061255
iter   60 value 16.993587
iter   70 value 16.989767
final  value 16.989763
converged
# weights: 11
initial  value 288.708296
iter   10 value 116.232524
iter   20 value 31.062074
iter   30 value 21.734003
iter   40 value 21.400677
iter   50 value 21.152509
iter   60 value 21.148278
iter   70 value 21.139581
iter   80 value 21.138048
final  value 21.138048
converged
# weights: 27
initial  value 302.463706
iter   10 value 20.387984
iter   20 value 0.780093
iter   30 value 0.457239
iter   40 value 0.382713
iter   50 value 0.310458
iter   60 value 0.264912
iter   70 value 0.247558
iter   80 value 0.228876
iter   90 value 0.206292
iter  100 value 0.175942
```

```

final  value 0.175942
stopped after 100 iterations
# weights: 43
initial  value 425.540078
iter  10 value 9.393580
iter  20 value 2.834147
iter  30 value 0.755127
iter  40 value 0.612649
iter  50 value 0.517616
iter  60 value 0.459726
iter  70 value 0.403178
iter  80 value 0.329106
iter  90 value 0.246791
iter 100 value 0.227341
final  value 0.227341
stopped after 100 iterations
# weights: 11
initial  value 293.822941
iter  10 value 104.083914
iter  20 value 35.034888
iter  30 value 28.108400
iter  40 value 26.328749
iter  50 value 25.644107
iter  60 value 25.585224
iter  70 value 25.338460
iter  80 value 25.321206
iter  90 value 25.274569
iter 100 value 25.230177
final  value 25.230177
stopped after 100 iterations
# weights: 27
initial  value 281.170383
iter  10 value 8.538247
iter  20 value 0.322451
iter  30 value 0.000776
final  value 0.000070
converged
# weights: 43
initial  value 369.888222
iter  10 value 3.717960
iter  20 value 0.114396
iter  30 value 0.009118
iter  40 value 0.000119
final  value 0.000085
converged
# weights: 11

```

```
initial value 312.007445
iter 10 value 158.459329
iter 20 value 116.568545
iter 30 value 102.715446
iter 40 value 100.676205
final value 100.676197
converged
# weights: 27
initial value 361.544346
iter 10 value 26.523720
iter 20 value 20.884003
iter 30 value 20.593879
iter 40 value 19.779705
iter 50 value 19.558644
iter 60 value 19.547367
final value 19.547176
converged
# weights: 43
initial value 339.850059
iter 10 value 30.277363
iter 20 value 18.743357
iter 30 value 18.489365
iter 40 value 18.217073
iter 50 value 17.863352
iter 60 value 17.759964
iter 70 value 17.726800
iter 80 value 17.725701
iter 80 value 17.725701
iter 80 value 17.725701
final value 17.725701
converged
# weights: 11
initial value 290.405559
iter 10 value 85.352298
iter 20 value 31.322831
iter 30 value 27.706134
iter 40 value 26.182509
iter 50 value 26.036565
iter 60 value 26.007504
final value 25.992829
converged
# weights: 27
initial value 278.199224
iter 10 value 16.203211
iter 20 value 0.685568
iter 30 value 0.346848
```

```

iter 40 value 0.334780
iter 50 value 0.317750
iter 60 value 0.293490
iter 70 value 0.282082
iter 80 value 0.259450
iter 90 value 0.237641
iter 100 value 0.228520
final value 0.228520
stopped after 100 iterations
# weights: 43
initial value 330.696110
iter 10 value 8.339990
iter 20 value 0.728177
iter 30 value 0.222136
iter 40 value 0.197815
iter 50 value 0.191075
iter 60 value 0.172602
iter 70 value 0.164307
iter 80 value 0.159419
iter 90 value 0.157512
iter 100 value 0.154587
final value 0.154587
stopped after 100 iterations
# weights: 11
initial value 287.361462
iter 10 value 107.823800
iter 20 value 105.186509
iter 30 value 101.169299
iter 40 value 99.895939
iter 50 value 94.260303
iter 60 value 77.998758
iter 70 value 37.961601
iter 80 value 26.646033
iter 90 value 25.761756
iter 100 value 25.395124
final value 25.395124
stopped after 100 iterations
# weights: 27
initial value 370.886917
iter 10 value 4.328237
iter 20 value 0.196525
iter 30 value 0.006269
iter 40 value 0.000848
final value 0.000093
converged
# weights: 43

```

```

initial value 333.275204
iter 10 value 30.049441
iter 20 value 4.431049
iter 30 value 3.993592
iter 40 value 0.353621
iter 50 value 0.040950
iter 60 value 0.000475
final value 0.000096
converged
# weights: 11
initial value 307.673088
iter 10 value 126.742546
iter 20 value 95.455592
iter 30 value 79.070352
iter 40 value 77.136677
final value 77.136495
converged
# weights: 27
initial value 332.170548
iter 10 value 34.155995
iter 20 value 23.649518
iter 30 value 21.885322
iter 40 value 21.789862
iter 50 value 21.604768
iter 60 value 21.584085
iter 70 value 21.583074
final value 21.583074
converged
# weights: 43
initial value 338.715909
iter 10 value 34.774308
iter 20 value 19.609030
iter 30 value 17.859022
iter 40 value 17.599106
iter 50 value 17.526806
iter 60 value 17.516129
iter 70 value 17.505920
final value 17.505615
converged
# weights: 11
initial value 303.657170
iter 10 value 101.363306
iter 20 value 41.773279
iter 30 value 27.061900
iter 40 value 26.291700
iter 50 value 25.772966

```

```

iter 60 value 25.620381
iter 70 value 25.613555
iter 80 value 25.610589
iter 80 value 25.610589
final value 25.610589
converged
# weights: 27
initial value 277.827024
iter 10 value 2.548544
iter 20 value 0.402780
iter 30 value 0.252278
iter 40 value 0.233478
iter 50 value 0.225768
iter 60 value 0.221323
iter 70 value 0.212955
iter 80 value 0.211445
iter 90 value 0.207499
iter 100 value 0.203459
final value 0.203459
stopped after 100 iterations
# weights: 43
initial value 327.829044
iter 10 value 6.316272
iter 20 value 0.349360
iter 30 value 0.268849
iter 40 value 0.259893
iter 50 value 0.243282
iter 60 value 0.230734
iter 70 value 0.223063
iter 80 value 0.205145
iter 90 value 0.191486
iter 100 value 0.186048
final value 0.186048
stopped after 100 iterations
# weights: 11
initial value 277.115404
iter 10 value 113.695795
iter 20 value 95.608171
iter 30 value 28.154040
iter 40 value 23.112452
iter 50 value 22.431601
iter 60 value 22.322301
iter 70 value 22.274625
iter 80 value 22.207383
iter 90 value 22.112930
iter 100 value 22.099790

```

```
final  value 22.099790
stopped after 100 iterations
# weights: 27
initial  value 333.397248
iter  10 value 2.192448
iter  20 value 0.008607
iter  30 value 0.000343
final  value 0.000077
converged
# weights: 43
initial  value 283.952948
iter  10 value 2.057694
iter  20 value 0.222541
iter  30 value 0.008625
iter  40 value 0.001718
iter  50 value 0.000414
iter  60 value 0.000165
iter  60 value 0.000096
iter  60 value 0.000095
final  value 0.000095
converged
# weights: 11
initial  value 295.325783
iter  10 value 199.882308
iter  20 value 128.206123
iter  30 value 117.790701
iter  40 value 82.167543
iter  50 value 75.267173
iter  60 value 75.195199
final  value 75.195196
converged
# weights: 27
initial  value 287.594097
iter  10 value 42.281120
iter  20 value 24.765586
iter  30 value 21.582254
iter  40 value 21.305273
iter  50 value 21.240125
iter  60 value 21.238047
final  value 21.238040
converged
# weights: 43
initial  value 324.185154
iter  10 value 31.914192
iter  20 value 17.844994
iter  30 value 17.045256
```

```

iter 40 value 16.850930
iter 50 value 16.820676
iter 60 value 16.808748
final value 16.808715
converged
# weights: 11
initial value 342.649726
iter 10 value 107.123100
iter 20 value 102.942994
iter 30 value 85.620784
iter 40 value 52.641049
iter 50 value 46.366623
iter 60 value 44.764991
iter 70 value 42.680131
iter 80 value 42.656900
iter 90 value 42.638474
final value 42.635136
converged
# weights: 27
initial value 298.072496
iter 10 value 2.383000
iter 20 value 0.378003
iter 30 value 0.254959
iter 40 value 0.246178
iter 50 value 0.242443
iter 60 value 0.234618
iter 70 value 0.228279
iter 80 value 0.222182
iter 90 value 0.217249
iter 100 value 0.211319
final value 0.211319
stopped after 100 iterations
# weights: 43
initial value 326.464421
iter 10 value 8.876118
iter 20 value 3.744117
iter 30 value 1.826163
iter 40 value 0.571278
iter 50 value 0.486189
iter 60 value 0.438203
iter 70 value 0.413342
iter 80 value 0.354470
iter 90 value 0.330488
iter 100 value 0.296364
final value 0.296364
stopped after 100 iterations

```

```

# weights: 11
initial value 303.320233
iter 10 value 106.762783
iter 20 value 105.816321
iter 30 value 105.552890
iter 40 value 105.536357
iter 50 value 105.498998
iter 60 value 105.433505
iter 70 value 105.131091
iter 80 value 105.112403
iter 90 value 105.111781
iter 100 value 105.109983
final value 105.109983
stopped after 100 iterations
# weights: 27
initial value 288.443076
iter 10 value 15.933276
iter 20 value 4.027993
iter 30 value 0.051736
iter 40 value 0.000973
iter 50 value 0.000248
iter 60 value 0.000185
final value 0.000088
converged
# weights: 43
initial value 293.834085
iter 10 value 4.963321
iter 20 value 0.040161
iter 30 value 0.000288
final value 0.000096
converged

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-11') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```

# weights: 11
initial value 314.169956
iter 10 value 140.522313
iter 20 value 123.756888
iter 30 value 82.851877

```

```
iter 40 value 78.557089
final value 78.557077
converged
# weights: 27
initial value 336.957251
iter 10 value 32.685813
iter 20 value 20.086949
iter 30 value 18.787435
iter 40 value 18.686564
iter 50 value 18.022612
iter 60 value 17.994486
final value 17.994483
converged
# weights: 43
initial value 329.673332
iter 10 value 24.367698
iter 20 value 16.684613
iter 30 value 16.331872
iter 40 value 16.191633
iter 50 value 16.175728
iter 60 value 16.173745
final value 16.173718
converged
# weights: 11
initial value 294.314083
iter 10 value 115.958565
iter 20 value 108.063045
iter 30 value 107.908001
iter 40 value 104.320040
iter 50 value 102.444835
iter 60 value 99.338676
iter 70 value 99.051001
iter 80 value 65.754410
iter 90 value 31.268995
iter 100 value 27.376420
final value 27.376420
stopped after 100 iterations
# weights: 27
initial value 377.100842
iter 10 value 11.682339
iter 20 value 5.688264
iter 30 value 5.347736
iter 40 value 5.296459
iter 50 value 5.105633
iter 60 value 0.407887
iter 70 value 0.236570
```

```
iter 80 value 0.207611
iter 90 value 0.163760
iter 100 value 0.117259
final value 0.117259
stopped after 100 iterations
# weights: 43
initial value 280.372843
iter 10 value 2.210043
iter 20 value 0.146623
iter 30 value 0.113127
iter 40 value 0.109537
iter 50 value 0.105277
iter 60 value 0.102415
iter 70 value 0.099591
iter 80 value 0.097296
iter 90 value 0.095143
iter 100 value 0.092211
final value 0.092211
stopped after 100 iterations
# weights: 11
initial value 302.196699
iter 10 value 108.454149
iter 20 value 105.923158
iter 30 value 104.934176
iter 40 value 104.419447
iter 50 value 104.014334
iter 60 value 103.698978
iter 70 value 103.603577
iter 80 value 101.740178
iter 90 value 54.941981
iter 100 value 43.751209
final value 43.751209
stopped after 100 iterations
# weights: 27
initial value 424.816640
iter 10 value 0.947527
iter 20 value 0.080142
iter 30 value 0.006200
iter 40 value 0.002294
iter 50 value 0.001802
iter 60 value 0.000375
iter 70 value 0.000321
iter 80 value 0.000161
final value 0.000099
converged
# weights: 43
```

```
initial value 444.437482
iter 10 value 3.477053
iter 20 value 0.073601
iter 30 value 0.002133
iter 40 value 0.001171
iter 50 value 0.000371
final value 0.000071
converged
# weights: 11
initial value 298.471520
iter 10 value 143.086556
iter 20 value 84.431835
iter 30 value 76.089894
final value 75.921102
converged
# weights: 27
initial value 286.414708
iter 10 value 45.396565
iter 20 value 19.336759
iter 30 value 18.546496
iter 40 value 18.510839
iter 50 value 18.504752
iter 60 value 18.504569
final value 18.504523
converged
# weights: 43
initial value 286.564091
iter 10 value 49.260202
iter 20 value 17.674677
iter 30 value 16.691100
iter 40 value 16.602976
iter 50 value 16.594360
iter 60 value 16.591027
iter 70 value 16.590945
final value 16.590942
converged
# weights: 11
initial value 315.161346
iter 10 value 140.657899
iter 20 value 110.824969
iter 30 value 105.997363
iter 40 value 84.414730
iter 50 value 40.577185
iter 60 value 23.281984
iter 70 value 22.572862
iter 80 value 21.986562
```

```
iter 90 value 21.976602
iter 100 value 21.967804
final value 21.967804
stopped after 100 iterations
# weights: 27
initial value 380.627302
iter 10 value 1.828956
iter 20 value 0.335106
iter 30 value 0.175323
iter 40 value 0.171062
iter 50 value 0.166065
iter 60 value 0.159481
iter 70 value 0.153682
iter 80 value 0.152436
iter 90 value 0.148732
iter 100 value 0.144749
final value 0.144749
stopped after 100 iterations
# weights: 43
initial value 351.228563
iter 10 value 7.940213
iter 20 value 0.415054
iter 30 value 0.248989
iter 40 value 0.230693
iter 50 value 0.217388
iter 60 value 0.205616
iter 70 value 0.202083
iter 80 value 0.185957
iter 90 value 0.181195
iter 100 value 0.173536
final value 0.173536
stopped after 100 iterations
# weights: 11
initial value 327.960410
iter 10 value 111.619084
iter 20 value 107.792000
final value 107.791429
converged
# weights: 27
initial value 336.659386
iter 10 value 47.084910
iter 20 value 1.205279
iter 30 value 0.072804
iter 40 value 0.011482
iter 50 value 0.003522
iter 60 value 0.001284
```

```
iter 70 value 0.000885
iter 80 value 0.000346
iter 90 value 0.000238
iter 100 value 0.000192
final value 0.000192
stopped after 100 iterations
# weights: 43
initial value 284.585415
iter 10 value 0.952626
iter 20 value 0.062559
iter 30 value 0.016514
iter 40 value 0.000411
final value 0.000097
converged
# weights: 11
initial value 275.032210
iter 10 value 90.917061
iter 20 value 79.397288
iter 30 value 79.330510
final value 79.330282
converged
# weights: 27
initial value 287.427362
iter 10 value 26.152030
iter 20 value 18.963257
iter 30 value 18.425227
iter 40 value 18.391016
iter 50 value 18.388872
final value 18.388871
converged
# weights: 43
initial value 353.292013
iter 10 value 34.399453
iter 20 value 18.114217
iter 30 value 16.556991
iter 40 value 16.385396
iter 50 value 16.351786
iter 60 value 16.349822
iter 70 value 16.349500
final value 16.349497
converged
# weights: 11
initial value 320.144835
iter 10 value 108.061508
iter 20 value 78.369687
iter 30 value 33.675440
```

```

iter 40 value 27.223872
iter 50 value 26.887083
iter 60 value 26.326418
iter 70 value 26.266128
iter 80 value 26.239286
iter 90 value 26.229415
iter 100 value 26.222128
final value 26.222128
stopped after 100 iterations
# weights: 27
initial value 382.960941
iter 10 value 25.029193
iter 20 value 16.286969
iter 30 value 5.183324
iter 40 value 0.275000
iter 50 value 0.218639
iter 60 value 0.204434
iter 70 value 0.171452
iter 80 value 0.149589
iter 90 value 0.134384
iter 100 value 0.128116
final value 0.128116
stopped after 100 iterations
# weights: 43
initial value 372.925640
iter 10 value 2.440875
iter 20 value 0.150469
iter 30 value 0.126319
iter 40 value 0.123328
iter 50 value 0.118650
iter 60 value 0.114366
iter 70 value 0.109756
iter 80 value 0.099268
iter 90 value 0.092753
iter 100 value 0.090638
final value 0.090638
stopped after 100 iterations
# weights: 11
initial value 366.638868
iter 10 value 114.879890
iter 20 value 27.874188
iter 30 value 25.890931
iter 40 value 24.345895
iter 50 value 24.097756
iter 60 value 24.087121
iter 70 value 23.949602

```

```

iter 80 value 23.937066
iter 90 value 23.903021
iter 100 value 23.900022
final value 23.900022
stopped after 100 iterations
# weights: 27
initial value 307.438603
iter 10 value 44.493866
iter 20 value 0.182592
iter 30 value 0.003011
final value 0.000046
converged
# weights: 43
initial value 300.653052
iter 10 value 34.153932
iter 20 value 4.262960
iter 30 value 0.191451
iter 40 value 0.011170
iter 50 value 0.005909
iter 60 value 0.000856
iter 70 value 0.000690
final value 0.000089
converged
# weights: 11
initial value 344.261041
iter 10 value 144.585977
iter 20 value 133.132635
iter 30 value 99.600386
iter 40 value 76.958411
final value 76.913817
converged
# weights: 27
initial value 286.997690
iter 10 value 50.719537
iter 20 value 21.784375
iter 30 value 20.206665
iter 40 value 20.122229
iter 50 value 20.097806
final value 20.095380
converged
# weights: 43
initial value 342.579443
iter 10 value 52.347868
iter 20 value 23.320813
iter 30 value 16.882529
iter 40 value 16.589354

```

```

iter 50 value 16.390790
iter 60 value 16.264375
iter 70 value 16.222772
iter 80 value 16.211649
iter 90 value 16.208662
iter 100 value 16.206141
final value 16.206141
stopped after 100 iterations
# weights: 11
initial value 295.787376
iter 10 value 108.892357
iter 20 value 67.643625
iter 30 value 37.499610
iter 40 value 25.107086
iter 50 value 24.667754
iter 60 value 24.488377
iter 70 value 24.429298
iter 80 value 24.426425
iter 90 value 24.412718
final value 24.412678
converged
# weights: 27
initial value 316.259074
iter 10 value 100.504306
iter 20 value 0.833931
iter 30 value 0.193247
iter 40 value 0.174561
iter 50 value 0.148560
iter 60 value 0.134324
iter 70 value 0.121952
iter 80 value 0.114864
iter 90 value 0.109556
iter 100 value 0.105307
final value 0.105307
stopped after 100 iterations
# weights: 43
initial value 403.676161
iter 10 value 2.432968
iter 20 value 0.180154
iter 30 value 0.113066
iter 40 value 0.104205
iter 50 value 0.101483
iter 60 value 0.099953
iter 70 value 0.098965
iter 80 value 0.097154
iter 90 value 0.096448

```

```
iter 100 value 0.095446
final  value 0.095446
stopped after 100 iterations
# weights: 11
initial  value 323.977010
iter   10 value 123.984480
iter   20 value 110.050016
iter   30 value 105.159108
iter   40 value 104.482956
iter   50 value 104.439082
iter   60 value 104.408972
iter   70 value 103.473090
iter   80 value 103.443635
iter   90 value 103.432947
iter 100 value 103.432605
final  value 103.432605
stopped after 100 iterations
# weights: 27
initial  value 336.570996
iter   10 value 22.839877
iter   20 value 0.892358
iter   30 value 0.004951
iter   40 value 0.000616
final  value 0.000083
converged
# weights: 43
initial  value 296.449864
iter   10 value 1.992574
iter   20 value 0.015984
iter   30 value 0.002231
final  value 0.000059
converged
# weights: 11
initial  value 294.880826
iter   10 value 135.929518
iter   20 value 92.147644
iter   30 value 77.827352
final  value 77.707218
converged
# weights: 27
initial  value 309.754547
iter   10 value 38.153768
iter   20 value 23.008362
iter   30 value 21.791629
iter   40 value 21.568538
iter   50 value 21.568069
```

```

final  value 21.568067
converged
# weights: 43
initial  value 315.868762
iter   10 value 45.070384
iter   20 value 17.828296
iter   30 value 17.559734
iter   40 value 17.545809
final  value 17.545769
converged
# weights: 11
initial  value 289.977034
iter   10 value 111.897392
iter   20 value 106.666377
iter   30 value 106.486441
iter   40 value 106.129332
iter   50 value 105.371979
iter   60 value 95.834901
iter   70 value 60.660318
iter   80 value 46.733416
iter   90 value 46.169200
iter  100 value 46.068677
final  value 46.068677
stopped after 100 iterations
# weights: 27
initial  value 334.357427
iter   10 value 10.958369
iter   20 value 2.217491
iter   30 value 0.333275
iter   40 value 0.261402
iter   50 value 0.245371
iter   60 value 0.235708
iter   70 value 0.225732
iter   80 value 0.216458
iter   90 value 0.209624
iter  100 value 0.205263
final  value 0.205263
stopped after 100 iterations
# weights: 43
initial  value 325.261977
iter   10 value 7.123908
iter   20 value 0.650912
iter   30 value 0.310115
iter   40 value 0.298449
iter   50 value 0.240619
iter   60 value 0.217462

```

```

iter 70 value 0.205308
iter 80 value 0.202083
iter 90 value 0.196346
iter 100 value 0.192129
final value 0.192129
stopped after 100 iterations
# weights: 11
initial value 293.546837
iter 10 value 124.441800
iter 20 value 105.472700
iter 30 value 104.253432
iter 40 value 104.029539
iter 50 value 102.283052
iter 60 value 98.230914
iter 70 value 93.674190
iter 80 value 35.573676
iter 90 value 23.480866
iter 100 value 22.991290
final value 22.991290
stopped after 100 iterations

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-12') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```

# weights: 27
initial value 299.383793
iter 10 value 20.120034
iter 20 value 0.418479
iter 30 value 0.001401
final value 0.000070
converged
# weights: 43
initial value 341.412347
iter 10 value 1.156705
iter 20 value 0.095412
iter 30 value 0.010921
iter 40 value 0.000179
final value 0.000089
converged
# weights: 11

```

```
initial value 284.353073
iter 10 value 119.058433
iter 20 value 96.462574
iter 30 value 76.877320
final value 76.830871
converged
# weights: 27
initial value 359.929165
iter 10 value 46.174122
iter 20 value 23.637810
iter 30 value 20.912629
iter 40 value 19.826224
iter 50 value 19.739835
iter 60 value 19.738889
final value 19.738842
converged
# weights: 43
initial value 298.551755
iter 10 value 41.787420
iter 20 value 16.599821
iter 30 value 15.704826
iter 40 value 15.644590
iter 50 value 15.632282
iter 60 value 15.631574
final value 15.631557
converged
# weights: 11
initial value 286.513556
iter 10 value 99.635608
iter 20 value 28.751250
iter 30 value 23.780219
iter 40 value 22.944365
iter 50 value 22.813762
iter 60 value 22.811440
iter 70 value 22.797372
final value 22.797235
converged
# weights: 27
initial value 296.864150
iter 10 value 0.853863
iter 20 value 0.252068
iter 30 value 0.243738
iter 40 value 0.200777
iter 50 value 0.193553
iter 60 value 0.183698
iter 70 value 0.174604
```

```
iter 80 value 0.168858
iter 90 value 0.159691
iter 100 value 0.155453
final value 0.155453
stopped after 100 iterations
# weights: 43
initial value 298.877233
iter 10 value 2.454734
iter 20 value 0.262662
iter 30 value 0.239977
iter 40 value 0.221233
iter 50 value 0.215671
iter 60 value 0.200896
iter 70 value 0.188106
iter 80 value 0.170616
iter 90 value 0.159236
iter 100 value 0.153373
final value 0.153373
stopped after 100 iterations
# weights: 11
initial value 286.714045
iter 10 value 109.135002
iter 20 value 106.272073
iter 30 value 106.264290
final value 106.264148
converged
# weights: 27
initial value 313.189397
iter 10 value 20.114325
iter 20 value 1.321487
iter 30 value 0.055987
iter 40 value 0.023778
iter 50 value 0.008061
iter 60 value 0.005543
iter 70 value 0.002973
iter 80 value 0.001558
iter 90 value 0.000434
iter 100 value 0.000156
final value 0.000156
stopped after 100 iterations
# weights: 43
initial value 301.839477
iter 10 value 2.855067
iter 20 value 0.014515
iter 30 value 0.000484
final value 0.000094
```

```

converged
# weights: 11
initial value 278.883030
iter 10 value 131.923013
iter 20 value 85.595465
iter 30 value 78.612925
iter 40 value 78.594063
final value 78.594043
converged
# weights: 27
initial value 353.603366
iter 10 value 28.945090
iter 20 value 22.096371
iter 30 value 21.828827
iter 40 value 21.811996
final value 21.811562
converged
# weights: 43
initial value 491.527266
iter 10 value 42.296543
iter 20 value 19.963819
iter 30 value 18.198054
iter 40 value 17.396643
iter 50 value 17.200794
iter 60 value 17.183100
iter 70 value 17.180982
final value 17.180776
converged
# weights: 11
initial value 296.856452
iter 10 value 108.865745
iter 20 value 104.337452
iter 30 value 103.864383
iter 40 value 98.231274
iter 50 value 65.242197
iter 60 value 43.047731
iter 70 value 40.382846
iter 80 value 39.595139
iter 90 value 38.425978
iter 100 value 38.322395
final value 38.322395
stopped after 100 iterations
# weights: 27
initial value 362.419031
iter 10 value 3.302969
iter 20 value 0.209348

```

```
iter 30 value 0.188317
iter 40 value 0.183429
iter 50 value 0.181411
iter 60 value 0.178036
iter 70 value 0.168735
iter 80 value 0.163947
iter 90 value 0.153688
iter 100 value 0.150759
final value 0.150759
stopped after 100 iterations
# weights: 43
initial value 400.536057
iter 10 value 27.253103
iter 20 value 0.779103
iter 30 value 0.563415
iter 40 value 0.465852
iter 50 value 0.317258
iter 60 value 0.214332
iter 70 value 0.187787
iter 80 value 0.161346
iter 90 value 0.156526
iter 100 value 0.151566
final value 0.151566
stopped after 100 iterations
# weights: 11
initial value 298.306927
iter 10 value 143.063096
iter 20 value 32.780373
iter 30 value 25.299043
iter 40 value 23.135724
iter 50 value 22.682873
iter 60 value 22.646604
iter 70 value 22.384187
iter 80 value 22.372321
iter 90 value 22.355087
iter 100 value 22.318241
final value 22.318241
stopped after 100 iterations
# weights: 27
initial value 352.707248
iter 10 value 21.909644
iter 20 value 0.244991
iter 30 value 0.003827
iter 40 value 0.002439
iter 50 value 0.000441
final value 0.000057
```

```

converged
# weights: 43
initial value 389.734968
iter 10 value 12.809362
iter 20 value 0.171190
iter 30 value 0.016665
iter 40 value 0.000274
iter 50 value 0.000111
final value 0.000100
converged
# weights: 11
initial value 332.198806
iter 10 value 124.054482
iter 20 value 89.750989
iter 30 value 76.135491
final value 75.852752
converged
# weights: 27
initial value 309.151029
iter 10 value 52.336223
iter 20 value 22.475071
iter 30 value 20.481548
iter 40 value 20.230337
iter 50 value 19.392737
iter 60 value 19.126419
iter 70 value 19.095282
final value 19.095273
converged
# weights: 43
initial value 307.371309
iter 10 value 42.629166
iter 20 value 19.345202
iter 30 value 17.620800
iter 40 value 16.858507
iter 50 value 16.763608
iter 60 value 16.740236
iter 70 value 16.739833
iter 80 value 16.739797
final value 16.739792
converged
# weights: 11
initial value 393.539941
iter 10 value 176.631436
iter 20 value 102.762687
iter 30 value 100.398927
iter 40 value 98.040713

```

```

iter 50 value 97.616633
iter 60 value 97.420562
iter 70 value 97.166438
iter 80 value 97.138775
iter 90 value 97.103511
iter 100 value 97.081602
final value 97.081602
stopped after 100 iterations
# weights: 27
initial value 292.329657
iter 10 value 23.370960
iter 20 value 2.559908
iter 30 value 0.769997
iter 40 value 0.715236
iter 50 value 0.579757
iter 60 value 0.456460
iter 70 value 0.376661
iter 80 value 0.304524
iter 90 value 0.260113
iter 100 value 0.183112
final value 0.183112
stopped after 100 iterations
# weights: 43
initial value 359.971610
iter 10 value 12.656707
iter 20 value 0.555790
iter 30 value 0.445907
iter 40 value 0.400321
iter 50 value 0.254866
iter 60 value 0.199225
iter 70 value 0.162211
iter 80 value 0.137542
iter 90 value 0.131420
iter 100 value 0.121988
final value 0.121988
stopped after 100 iterations
# weights: 11
initial value 330.103970
iter 10 value 127.965736
iter 20 value 114.018112
iter 30 value 110.457541
iter 40 value 106.390986
iter 50 value 106.059882
iter 60 value 106.026913
iter 70 value 106.002198
iter 70 value 106.002198

```

```
final value 106.002198
converged
# weights: 27
initial value 284.656443
iter 10 value 9.552190
iter 20 value 1.619460
iter 30 value 0.145802
iter 40 value 0.000906
final value 0.000049
converged
# weights: 43
initial value 427.062894
iter 10 value 1.132076
iter 20 value 0.103511
iter 30 value 0.004981
iter 40 value 0.000496
iter 50 value 0.000151
final value 0.000093
converged
# weights: 11
initial value 340.008485
iter 10 value 155.853669
iter 20 value 120.714388
iter 30 value 81.311265
iter 40 value 78.165148
final value 78.165130
converged
# weights: 27
initial value 373.558386
iter 10 value 33.395134
iter 20 value 21.245100
iter 30 value 21.062029
iter 40 value 21.056170
final value 21.055553
converged
# weights: 43
initial value 251.762783
iter 10 value 32.140705
iter 20 value 18.319130
iter 30 value 17.646886
iter 40 value 17.375520
iter 50 value 17.342677
iter 60 value 17.333557
final value 17.333532
converged
# weights: 11
```

```

initial value 341.313659
iter 10 value 107.709685
iter 20 value 107.624501
iter 30 value 107.423654
iter 40 value 106.564934
iter 50 value 106.100753
iter 60 value 105.855388
iter 70 value 105.562749
iter 80 value 105.486758
iter 90 value 95.458932
iter 100 value 52.485352
final value 52.485352
stopped after 100 iterations
# weights: 27
initial value 325.413642
iter 10 value 30.569845
iter 20 value 21.638709
iter 30 value 16.514250
iter 40 value 5.819312
iter 50 value 4.990106
iter 60 value 4.251001
iter 70 value 3.595546
iter 80 value 3.334735
iter 90 value 1.808632
iter 100 value 1.590466
final value 1.590466
stopped after 100 iterations
# weights: 43
initial value 299.836515
iter 10 value 6.144913
iter 20 value 0.246214
iter 30 value 0.203820
iter 40 value 0.199039
iter 50 value 0.188523
iter 60 value 0.168733
iter 70 value 0.162161
iter 80 value 0.149385
iter 90 value 0.147249
iter 100 value 0.145996
final value 0.145996
stopped after 100 iterations
# weights: 11
initial value 323.625562
iter 10 value 107.810233
iter 20 value 107.791663
final value 107.791428

```

```

converged
# weights: 27
initial value 320.984681
iter 10 value 11.235361
iter 20 value 0.075319
iter 30 value 0.008270
iter 40 value 0.003888
iter 50 value 0.002307
iter 60 value 0.000273
iter 70 value 0.000136
final value 0.000094
converged
# weights: 43
initial value 538.612569
iter 10 value 10.284085
iter 20 value 0.363906
iter 30 value 0.042626
iter 40 value 0.018054
iter 50 value 0.002112
iter 60 value 0.000552
final value 0.000092
converged
# weights: 11
initial value 310.216715
iter 10 value 120.962652
iter 20 value 92.905677
iter 30 value 81.204593
final value 80.942233
converged
# weights: 27
initial value 385.869196
iter 10 value 32.017081
iter 20 value 22.587428
iter 30 value 20.194398
iter 40 value 19.650893
iter 50 value 19.633054
iter 60 value 19.628018
iter 60 value 19.628018
iter 60 value 19.628018
final value 19.628018
converged
# weights: 43
initial value 305.718122
iter 10 value 22.549623
iter 20 value 18.109386
iter 30 value 17.741219

```

```

iter 40 value 17.728927
iter 50 value 17.726560
final value 17.726558
converged
# weights: 11
initial value 313.772397
iter 10 value 108.710369
iter 20 value 107.636797
iter 30 value 107.393553
iter 40 value 107.090413
iter 50 value 106.797278
iter 60 value 106.507544
iter 70 value 106.482000
iter 80 value 105.996710
iter 90 value 105.385180
iter 100 value 104.454737
final value 104.454737
stopped after 100 iterations
# weights: 27
initial value 299.265331
iter 10 value 5.403335
iter 20 value 0.451097
iter 30 value 0.308906
iter 40 value 0.300892
iter 50 value 0.282905
iter 60 value 0.259378
iter 70 value 0.238551
iter 80 value 0.220507
iter 90 value 0.203427
iter 100 value 0.189050
final value 0.189050
stopped after 100 iterations

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-13') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```

# weights: 43
initial value 285.989211
iter 10 value 4.245606
iter 20 value 0.487852

```

```

iter 30 value 0.446792
iter 40 value 0.380202
iter 50 value 0.324499
iter 60 value 0.284974
iter 70 value 0.243884
iter 80 value 0.233249
iter 90 value 0.207465
iter 100 value 0.199104
final value 0.199104
stopped after 100 iterations
# weights: 11
initial value 294.986282
iter 10 value 107.133661
iter 20 value 106.650669
iter 30 value 104.936716
iter 40 value 94.605345
iter 50 value 65.766896
iter 60 value 30.239315
iter 70 value 27.683079
iter 80 value 27.475810
iter 90 value 26.913208
iter 100 value 26.710176
final value 26.710176
stopped after 100 iterations
# weights: 27
initial value 270.775555
iter 10 value 4.694058
iter 20 value 0.052912
final value 0.000082
converged
# weights: 43
initial value 397.585329
iter 10 value 0.619546
iter 20 value 0.040845
iter 30 value 0.019076
iter 40 value 0.003014
iter 50 value 0.000962
iter 60 value 0.000506
iter 70 value 0.000116
final value 0.000082
converged
# weights: 11
initial value 315.653737
iter 10 value 182.859393
iter 20 value 121.522023
iter 30 value 105.012876

```

```

final  value 103.348031
converged
# weights: 27
initial  value 376.613627
iter   10 value 34.802007
iter   20 value 20.261712
iter   30 value 20.016471
iter   40 value 19.955021
iter   50 value 19.933193
final  value 19.931805
converged
# weights: 43
initial  value 436.222467
iter   10 value 114.545746
iter   20 value 25.290720
iter   30 value 17.874889
iter   40 value 16.750586
iter   50 value 16.467253
iter   60 value 16.418333
iter   70 value 16.411549
iter   80 value 16.409896
iter   90 value 16.407023
iter  100 value 16.406952
final  value 16.406952
stopped after 100 iterations
# weights: 11
initial  value 300.091965
iter   10 value 123.972820
iter   20 value 107.471529
iter   30 value 107.009537
iter   40 value 106.904191
iter   50 value 106.866086
iter   60 value 106.863158
iter   70 value 106.859213
iter   70 value 106.859213
final  value 106.859213
converged
# weights: 27
initial  value 286.362713
iter   10 value 7.156350
iter   20 value 1.485441
iter   30 value 0.262652
iter   40 value 0.240392
iter   50 value 0.217338
iter   60 value 0.203238
iter   70 value 0.185655

```

```

iter 80 value 0.161798
iter 90 value 0.147756
iter 100 value 0.123142
final value 0.123142
stopped after 100 iterations
# weights: 43
initial value 316.090528
iter 10 value 0.656709
iter 20 value 0.126736
iter 30 value 0.113914
iter 40 value 0.108048
iter 50 value 0.105531
iter 60 value 0.100588
iter 70 value 0.094557
iter 80 value 0.090963
iter 90 value 0.088161
iter 100 value 0.086004
final value 0.086004
stopped after 100 iterations
# weights: 11
initial value 341.218434
iter 10 value 113.581882
iter 20 value 106.659354
iter 30 value 106.645832
final value 106.645103
converged
# weights: 27
initial value 333.932066
iter 10 value 1.147695
iter 20 value 0.009246
iter 30 value 0.000779
iter 40 value 0.000307
final value 0.000090
converged
# weights: 43
initial value 340.124937
iter 10 value 4.969593
iter 20 value 0.062316
iter 30 value 0.006924
iter 40 value 0.000235
final value 0.000094
converged
# weights: 11
initial value 346.700679
iter 10 value 118.665578
iter 20 value 100.742026

```

```

iter 30 value 74.788483
final value 74.597478
converged
# weights: 27
initial value 291.545655
iter 10 value 34.573040
iter 20 value 19.052681
iter 30 value 18.095339
iter 40 value 17.959775
iter 50 value 17.879263
iter 60 value 17.876977
final value 17.876769
converged
# weights: 43
initial value 298.707665
iter 10 value 44.343080
iter 20 value 21.582953
iter 30 value 16.807733
iter 40 value 16.251993
iter 50 value 16.065293
iter 60 value 15.998022
iter 70 value 15.966724
iter 80 value 15.948645
iter 90 value 15.945719
iter 100 value 15.944202
final value 15.944202
stopped after 100 iterations
# weights: 11
initial value 303.203151
iter 10 value 107.647278
iter 20 value 104.991216
iter 30 value 80.869413
iter 40 value 42.972486
iter 50 value 40.271802
iter 60 value 39.285376
iter 70 value 39.154940
iter 80 value 39.137015
iter 90 value 39.101048
final value 39.100932
converged
# weights: 27
initial value 320.429131
iter 10 value 8.001351
iter 20 value 0.509730
iter 30 value 0.380593
iter 40 value 0.319856

```

```

iter 50 value 0.261392
iter 60 value 0.185292
iter 70 value 0.173944
iter 80 value 0.163592
iter 90 value 0.157994
iter 100 value 0.149777
final value 0.149777
stopped after 100 iterations
# weights: 43
initial value 366.337147
iter 10 value 1.752088
iter 20 value 0.175150
iter 30 value 0.145604
iter 40 value 0.139423
iter 50 value 0.126099
iter 60 value 0.121890
iter 70 value 0.116412
iter 80 value 0.112822
iter 90 value 0.109481
iter 100 value 0.106699
final value 0.106699
stopped after 100 iterations
# weights: 11
initial value 364.270780
iter 10 value 110.465579
iter 20 value 70.610021
iter 30 value 16.270890
iter 40 value 14.119783
iter 50 value 13.544589
iter 60 value 13.071666
iter 70 value 12.825199
iter 80 value 12.807339
iter 90 value 12.788324
iter 100 value 12.744367
final value 12.744367
stopped after 100 iterations
# weights: 27
initial value 306.657988
iter 10 value 1.182532
iter 20 value 0.042476
iter 30 value 0.016464
iter 40 value 0.005504
iter 50 value 0.000525
iter 60 value 0.000137
final value 0.000093
converged

```

```

# weights: 43
initial value 294.371406
iter 10 value 1.815027
iter 20 value 0.017934
iter 30 value 0.003949
iter 40 value 0.001867
iter 50 value 0.000667
final value 0.000051
converged
# weights: 11
initial value 290.301531
iter 10 value 93.222910
iter 20 value 73.840855
iter 30 value 73.623475
final value 73.596521
converged
# weights: 27
initial value 344.657654
iter 10 value 32.710115
iter 20 value 19.553031
iter 30 value 18.778595
iter 40 value 18.718434
iter 50 value 18.713243
final value 18.713222
converged
# weights: 43
initial value 347.516626
iter 10 value 37.204534
iter 20 value 18.457358
iter 30 value 17.055903
iter 40 value 16.956357
iter 50 value 16.916293
iter 60 value 16.907806
iter 70 value 16.903845
iter 80 value 16.874761
iter 90 value 16.827491
iter 100 value 16.824285
final value 16.824285
stopped after 100 iterations
# weights: 11
initial value 303.989952
iter 10 value 106.192414
iter 20 value 58.096809
iter 30 value 21.276998
iter 40 value 14.205946
iter 50 value 13.830209

```

```
iter 60 value 13.807584
iter 70 value 13.800577
iter 80 value 13.799000
iter 90 value 13.797678
final value 13.797670
converged
# weights: 27
initial value 302.511328
iter 10 value 6.786609
iter 20 value 0.224254
iter 30 value 0.208158
iter 40 value 0.201944
iter 50 value 0.193769
iter 60 value 0.186804
iter 70 value 0.180510
iter 80 value 0.171114
iter 90 value 0.161286
iter 100 value 0.154460
final value 0.154460
stopped after 100 iterations
# weights: 43
initial value 379.575701
iter 10 value 16.109259
iter 20 value 1.317048
iter 30 value 0.450007
iter 40 value 0.394728
iter 50 value 0.385491
iter 60 value 0.344539
iter 70 value 0.317559
iter 80 value 0.300184
iter 90 value 0.278199
iter 100 value 0.251130
final value 0.251130
stopped after 100 iterations
# weights: 11
initial value 354.860242
iter 10 value 108.754185
iter 20 value 106.002496
iter 30 value 105.448300
iter 40 value 104.441366
iter 50 value 103.142869
iter 60 value 102.935091
iter 70 value 101.874503
iter 80 value 101.835274
iter 90 value 100.894186
iter 100 value 100.717787
```

```
final  value 100.717787
stopped after 100 iterations
# weights: 27
initial  value 333.762046
iter   10 value 20.564737
iter   20 value 0.216700
iter   30 value 0.026352
iter   40 value 0.000972
iter   50 value 0.000273
final  value 0.000062
converged
# weights: 43
initial  value 361.936254
iter   10 value 9.109126
iter   20 value 0.051578
iter   30 value 0.016103
iter   40 value 0.003410
iter   50 value 0.002331
iter   60 value 0.000484
iter   70 value 0.000418
final  value 0.000078
converged
# weights: 11
initial  value 265.310027
iter   10 value 93.779284
iter   20 value 77.943288
iter   30 value 77.914313
final  value 77.914304
converged
# weights: 27
initial  value 373.234680
iter   10 value 25.501349
iter   20 value 18.862587
iter   30 value 18.464867
iter   40 value 18.444710
iter   50 value 18.439670
final  value 18.439668
converged
# weights: 43
initial  value 255.080090
iter   10 value 27.604473
iter   20 value 17.523826
iter   30 value 16.903449
iter   40 value 16.794961
iter   50 value 16.426986
iter   60 value 16.400005
```

```
iter 70 value 16.398050
final value 16.398045
converged
# weights: 11
initial value 293.784104
iter 10 value 110.306121
iter 20 value 106.804403
iter 30 value 106.791266
iter 40 value 106.671411
iter 50 value 48.694228
iter 60 value 29.359248
iter 70 value 26.069680
iter 80 value 25.844171
iter 90 value 25.521852
iter 100 value 25.509505
final value 25.509505
stopped after 100 iterations
# weights: 27
initial value 308.690010
iter 10 value 11.020206
iter 20 value 0.333879
iter 30 value 0.183159
iter 40 value 0.164260
iter 50 value 0.158600
iter 60 value 0.150923
iter 70 value 0.145609
iter 80 value 0.130860
iter 90 value 0.120604
iter 100 value 0.115361
final value 0.115361
stopped after 100 iterations
# weights: 43
initial value 310.100206
iter 10 value 1.162122
iter 20 value 0.300546
iter 30 value 0.254951
iter 40 value 0.226449
iter 50 value 0.199923
iter 60 value 0.154329
iter 70 value 0.135618
iter 80 value 0.113006
iter 90 value 0.097758
iter 100 value 0.095135
final value 0.095135
stopped after 100 iterations
# weights: 11
```

```

initial value 311.377886
iter 10 value 94.786931
iter 20 value 54.536010
iter 30 value 27.072842
iter 40 value 22.996491
iter 50 value 22.933725
iter 60 value 22.901842
iter 70 value 22.858295
iter 80 value 22.836490
iter 90 value 22.815455
iter 100 value 22.759172
final value 22.759172
stopped after 100 iterations
# weights: 27
initial value 323.037710
iter 10 value 33.178580
iter 20 value 5.908598
iter 30 value 4.811066
iter 40 value 4.420746
iter 50 value 4.294314
iter 60 value 4.082911
iter 70 value 3.864859
iter 80 value 3.626665
iter 90 value 3.076865
iter 100 value 2.994430
final value 2.994430
stopped after 100 iterations
# weights: 43
initial value 314.809780
iter 10 value 2.518680
iter 20 value 0.040899
iter 30 value 0.003432
iter 40 value 0.000559
iter 50 value 0.000129
final value 0.000061
converged
# weights: 11
initial value 405.807035
iter 10 value 130.694334
iter 20 value 80.943164
iter 30 value 76.260795
final value 76.260491
converged
# weights: 27
initial value 346.246118
iter 10 value 28.445063

```

```

iter 20 value 19.200287
iter 30 value 18.985034
iter 40 value 18.306418
iter 50 value 17.897471
iter 60 value 17.895954
iter 70 value 17.895722
iter 70 value 17.895722
iter 70 value 17.895722
final value 17.895722
converged
# weights: 43
initial value 275.576664
iter 10 value 22.008011
iter 20 value 16.214003
iter 30 value 16.071634
iter 40 value 16.058569
iter 50 value 16.058229
final value 16.058228
converged

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-14') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```

# weights: 11
initial value 350.764850
iter 10 value 107.628176
iter 20 value 94.117398
iter 30 value 62.076069
iter 40 value 31.572353
iter 50 value 25.093288
iter 60 value 23.337989
iter 70 value 23.305976
iter 80 value 23.277350
iter 90 value 23.266793
iter 90 value 23.266793
final value 23.266792
converged
# weights: 27
initial value 278.881120
iter 10 value 4.125520

```

```

iter 20 value 0.362948
iter 30 value 0.254874
iter 40 value 0.224509
iter 50 value 0.211277
iter 60 value 0.204327
iter 70 value 0.189941
iter 80 value 0.177480
iter 90 value 0.163346
iter 100 value 0.152780
final value 0.152780
stopped after 100 iterations
# weights: 43
initial value 361.513373
iter 10 value 5.940013
iter 20 value 0.431528
iter 30 value 0.287343
iter 40 value 0.268025
iter 50 value 0.256740
iter 60 value 0.200243
iter 70 value 0.178552
iter 80 value 0.133438
iter 90 value 0.125173
iter 100 value 0.119549
final value 0.119549
stopped after 100 iterations
# weights: 11
initial value 296.457973
iter 10 value 117.828601
iter 20 value 106.667900
iter 30 value 106.646242
iter 40 value 106.644531
iter 50 value 106.642914
iter 60 value 106.578223
iter 70 value 105.544269
iter 80 value 105.450879
iter 90 value 103.190922
iter 100 value 80.525234
final value 80.525234
stopped after 100 iterations
# weights: 27
initial value 326.480019
iter 10 value 1.336682
iter 20 value 0.039401
iter 30 value 0.001686
iter 40 value 0.000172
final value 0.000085

```

```

converged
# weights: 43
initial value 368.348356
iter 10 value 4.771324
iter 20 value 0.018725
final value 0.000082
converged
# weights: 11
initial value 342.164742
iter 10 value 143.655589
iter 20 value 105.924641
iter 30 value 77.180129
iter 40 value 74.892644
final value 74.892642
converged
# weights: 27
initial value 319.236640
iter 10 value 59.032097
iter 20 value 21.639995
iter 30 value 20.564191
iter 40 value 20.540164
iter 50 value 20.538998
iter 60 value 20.538815
final value 20.538815
converged
# weights: 43
initial value 283.853391
iter 10 value 28.843606
iter 20 value 19.566480
iter 30 value 17.303164
iter 40 value 17.087827
iter 50 value 17.007170
iter 60 value 17.000302
iter 70 value 16.998636
final value 16.998635
converged
# weights: 11
initial value 325.261831
iter 10 value 117.320638
iter 20 value 106.823714
iter 30 value 106.800737
iter 40 value 106.783567
final value 106.708496
converged
# weights: 27
initial value 282.346662

```

```
iter 10 value 21.872828
iter 20 value 7.397479
iter 30 value 1.095728
iter 40 value 0.464699
iter 50 value 0.401837
iter 60 value 0.377520
iter 70 value 0.340172
iter 80 value 0.299875
iter 90 value 0.265001
iter 100 value 0.236729
final value 0.236729
stopped after 100 iterations
# weights: 43
initial value 289.866225
iter 10 value 1.882651
iter 20 value 0.189629
iter 30 value 0.175136
iter 40 value 0.168240
iter 50 value 0.154106
iter 60 value 0.149377
iter 70 value 0.145486
iter 80 value 0.141636
iter 90 value 0.139676
iter 100 value 0.138385
final value 0.138385
stopped after 100 iterations
# weights: 11
initial value 380.606222
iter 10 value 106.988356
iter 20 value 106.268672
final value 106.264382
converged
# weights: 27
initial value 284.719304
iter 10 value 9.030198
iter 20 value 0.254287
iter 30 value 0.000230
final value 0.000053
converged
# weights: 43
initial value 265.267050
iter 10 value 1.936531
iter 20 value 0.058628
iter 30 value 0.002280
final value 0.000085
converged
```

```

# weights: 11
initial value 278.332662
iter 10 value 149.844364
iter 20 value 115.372350
iter 30 value 95.318043
iter 40 value 79.651988
final value 79.647588
converged
# weights: 27
initial value 312.651909
iter 10 value 49.722027
iter 20 value 23.033085
iter 30 value 20.869512
iter 40 value 20.531962
iter 50 value 20.065057
iter 60 value 19.808817
iter 70 value 19.665839
final value 19.665726
converged
# weights: 43
initial value 323.781059
iter 10 value 38.559897
iter 20 value 18.783578
iter 30 value 17.845680
iter 40 value 17.735499
iter 50 value 17.708561
iter 60 value 17.706379
iter 70 value 17.704480
final value 17.704476
converged
# weights: 11
initial value 282.141434
iter 10 value 109.426480
iter 20 value 106.163444
iter 30 value 105.408375
iter 40 value 105.049983
iter 50 value 104.985500
iter 60 value 104.754479
iter 70 value 104.512666
iter 80 value 104.411779
iter 90 value 101.811603
iter 100 value 49.040733
final value 49.040733
stopped after 100 iterations
# weights: 27
initial value 344.317357

```

```

iter 10 value 19.748487
iter 20 value 2.453845
iter 30 value 0.381426
iter 40 value 0.331800
iter 50 value 0.300880
iter 60 value 0.268131
iter 70 value 0.219824
iter 80 value 0.194908
iter 90 value 0.191458
iter 100 value 0.185694
final value 0.185694
stopped after 100 iterations
# weights: 43
initial value 365.135159
iter 10 value 11.378700
iter 20 value 0.360051
iter 30 value 0.252402
iter 40 value 0.237332
iter 50 value 0.227447
iter 60 value 0.212535
iter 70 value 0.191501
iter 80 value 0.173972
iter 90 value 0.167758
iter 100 value 0.163050
final value 0.163050
stopped after 100 iterations
# weights: 11
initial value 315.633403
iter 10 value 116.359345
iter 20 value 97.877986
iter 30 value 96.322104
iter 40 value 71.469625
iter 50 value 32.722448
iter 60 value 26.280355
iter 70 value 23.934582
iter 80 value 23.376467
iter 90 value 23.346395
iter 100 value 23.247848
final value 23.247848
stopped after 100 iterations
# weights: 27
initial value 313.703121
iter 10 value 8.168650
iter 20 value 0.097324
iter 30 value 0.000780
final value 0.000088

```

```

converged
# weights: 43
initial value 325.396369
iter 10 value 7.644367
iter 20 value 0.138351
iter 30 value 0.007304
final value 0.000093
converged
# weights: 11
initial value 315.741174
iter 10 value 124.377254
iter 20 value 108.928103
iter 30 value 102.314533
iter 40 value 101.403493
iter 40 value 101.403493
iter 40 value 101.403493
final value 101.403493
converged
# weights: 27
initial value 297.566836
iter 10 value 45.046057
iter 20 value 20.057223
iter 30 value 19.408178
iter 40 value 19.166401
iter 50 value 18.853793
iter 60 value 18.814805
iter 70 value 18.811436
final value 18.811432
converged
# weights: 43
initial value 315.421250
iter 10 value 25.773428
iter 20 value 18.679814
iter 30 value 17.224278
iter 40 value 17.023451
iter 50 value 16.932090
iter 60 value 16.912274
iter 70 value 16.910426
iter 80 value 16.910245
final value 16.910113
converged
# weights: 11
initial value 343.616824
iter 10 value 139.215896
iter 20 value 128.528476
iter 30 value 123.891132

```

```

iter 40 value 123.656659
iter 50 value 123.548925
iter 60 value 122.782108
iter 70 value 122.514978
iter 80 value 112.763905
iter 90 value 112.330949
iter 100 value 106.611326
final value 106.611326
stopped after 100 iterations
# weights: 27
initial value 300.014212
iter 10 value 9.122212
iter 20 value 0.314082
iter 30 value 0.260939
iter 40 value 0.253723
iter 50 value 0.242891
iter 60 value 0.223563
iter 70 value 0.212254
iter 80 value 0.200303
iter 90 value 0.186859
iter 100 value 0.180647
final value 0.180647
stopped after 100 iterations
# weights: 43
initial value 403.131750
iter 10 value 10.785747
iter 20 value 0.323652
iter 30 value 0.273512
iter 40 value 0.250419
iter 50 value 0.233088
iter 60 value 0.213402
iter 70 value 0.193090
iter 80 value 0.182316
iter 90 value 0.180879
iter 100 value 0.175705
final value 0.175705
stopped after 100 iterations
# weights: 11
initial value 287.878947
iter 10 value 106.677362
iter 20 value 106.645008
final value 106.644971
converged
# weights: 27
initial value 278.243400
iter 10 value 11.605371

```

```

iter 20 value 0.116712
iter 30 value 0.002590
final value 0.000074
converged
# weights: 43
initial value 276.194544
iter 10 value 2.441461
iter 20 value 0.034003
iter 30 value 0.000823
iter 40 value 0.000191
final value 0.000083
converged
# weights: 11
initial value 323.473352
iter 10 value 124.028278
iter 20 value 104.587002
iter 30 value 100.356405
final value 100.256750
converged
# weights: 27
initial value 323.879034
iter 10 value 48.115086
iter 20 value 20.151856
iter 30 value 19.331687
iter 40 value 19.150872
iter 50 value 18.730982
iter 60 value 18.651371
iter 70 value 18.650938
iter 70 value 18.650938
iter 70 value 18.650938
final value 18.650938
converged
# weights: 43
initial value 350.412591
iter 10 value 66.485608
iter 20 value 17.708364
iter 30 value 16.999594
iter 40 value 16.861567
iter 50 value 16.824241
iter 60 value 16.808645
iter 70 value 16.807735
iter 80 value 16.807707
final value 16.807701
converged
# weights: 11
initial value 322.992545

```

```

iter 10 value 70.967588
iter 20 value 35.416498
iter 30 value 30.042670
iter 40 value 29.655101
iter 50 value 29.419031
iter 60 value 29.366134
iter 70 value 29.351154
final value 29.351153
converged
# weights: 27
initial value 323.720778
iter 10 value 12.747702
iter 20 value 0.435633
iter 30 value 0.275900
iter 40 value 0.262879
iter 50 value 0.250260
iter 60 value 0.191584
iter 70 value 0.175534
iter 80 value 0.164938
iter 90 value 0.147269
iter 100 value 0.139650
final value 0.139650
stopped after 100 iterations
# weights: 43
initial value 460.584864
iter 10 value 7.506081
iter 20 value 0.267539
iter 30 value 0.158187
iter 40 value 0.145153
iter 50 value 0.138929
iter 60 value 0.132630
iter 70 value 0.129942
iter 80 value 0.126784
iter 90 value 0.118260
iter 100 value 0.112359
final value 0.112359
stopped after 100 iterations
# weights: 11
initial value 327.142936
iter 10 value 104.446931
iter 20 value 90.670544
iter 30 value 27.804759
iter 40 value 22.428014
iter 50 value 19.085159
iter 60 value 18.808715
iter 70 value 18.745232

```

```

iter 80 value 18.696912
iter 90 value 18.675637
iter 100 value 18.668047
final value 18.668047
stopped after 100 iterations
# weights: 27
initial value 334.414249
iter 10 value 7.576314
iter 20 value 0.168303
iter 30 value 0.001076
final value 0.000077
converged
# weights: 43
initial value 264.936792
iter 10 value 6.754692
iter 20 value 0.295450
iter 30 value 0.023550
iter 40 value 0.010497
iter 50 value 0.002349
iter 60 value 0.000817
iter 70 value 0.000144
final value 0.000090
converged
# weights: 11
initial value 344.372658
iter 10 value 166.509808
iter 20 value 139.103761
iter 30 value 128.868951
iter 40 value 95.848302
iter 50 value 74.464253
final value 74.459967
converged

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-15') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```

# weights: 27
initial value 350.767741
iter 10 value 29.940160
iter 20 value 21.257250

```

```

iter 30 value 21.139141
iter 40 value 21.010889
iter 50 value 20.952772
final value 20.952767
converged
# weights: 43
initial value 348.778597
iter 10 value 27.778711
iter 20 value 18.051247
iter 30 value 17.474432
iter 40 value 17.297172
iter 50 value 17.273453
iter 60 value 17.267537
final value 17.266328
converged
# weights: 11
initial value 291.664048
iter 10 value 106.967774
iter 20 value 100.724162
iter 30 value 57.059376
iter 40 value 46.865176
iter 50 value 46.195117
iter 60 value 45.701539
iter 70 value 45.698442
iter 80 value 45.664620
final value 45.663507
converged
# weights: 27
initial value 280.135489
iter 10 value 13.431381
iter 20 value 0.594079
iter 30 value 0.494588
iter 40 value 0.425470
iter 50 value 0.394917
iter 60 value 0.357075
iter 70 value 0.339241
iter 80 value 0.251228
iter 90 value 0.234823
iter 100 value 0.208880
final value 0.208880
stopped after 100 iterations
# weights: 43
initial value 291.126387
iter 10 value 4.357043
iter 20 value 0.370510
iter 30 value 0.291883

```

```

iter 40 value 0.287391
iter 50 value 0.249224
iter 60 value 0.238994
iter 70 value 0.206812
iter 80 value 0.191662
iter 90 value 0.187970
iter 100 value 0.181045
final value 0.181045
stopped after 100 iterations
# weights: 11
initial value 288.096141
iter 10 value 98.379605
iter 20 value 35.459259
iter 30 value 28.979204
iter 40 value 27.551829
iter 50 value 27.039711
iter 60 value 27.020203
iter 70 value 26.935847
iter 80 value 26.898353
iter 90 value 26.876187
iter 100 value 26.826122
final value 26.826122
stopped after 100 iterations
# weights: 27
initial value 301.339325
iter 10 value 6.874327
iter 20 value 0.196958
iter 30 value 0.002657
iter 40 value 0.001186
final value 0.000089
converged
# weights: 43
initial value 285.006722
iter 10 value 0.733794
iter 20 value 0.013352
iter 30 value 0.000542
iter 40 value 0.000106
iter 40 value 0.000067
iter 40 value 0.000065
final value 0.000065
converged
# weights: 11
initial value 311.746223
iter 10 value 120.492223
iter 20 value 116.345669
iter 30 value 104.354022

```

```
iter 40 value 101.482038
final value 101.482031
converged
# weights: 27
initial value 324.729235
iter 10 value 40.044211
iter 20 value 20.532366
iter 30 value 19.198713
iter 40 value 18.775289
iter 50 value 18.733706
iter 60 value 18.732824
final value 18.732740
converged
# weights: 43
initial value 372.067915
iter 10 value 19.035514
iter 20 value 17.396166
iter 30 value 16.909982
iter 40 value 16.783606
iter 50 value 16.770351
iter 60 value 16.769945
final value 16.769907
converged
# weights: 11
initial value 295.483715
iter 10 value 114.833470
iter 20 value 58.859016
iter 30 value 29.917448
iter 40 value 28.239242
iter 50 value 27.586371
iter 60 value 27.561082
iter 70 value 27.522824
final value 27.522823
converged
# weights: 27
initial value 313.369619
iter 10 value 17.841633
iter 20 value 8.042563
iter 30 value 3.296027
iter 40 value 2.822788
iter 50 value 2.159220
iter 60 value 1.694184
iter 70 value 1.324223
iter 80 value 1.091026
iter 90 value 0.684032
iter 100 value 0.476425
```

```

final value 0.476425
stopped after 100 iterations
# weights: 43
initial value 317.613137
iter 10 value 3.945256
iter 20 value 0.249623
iter 30 value 0.174243
iter 40 value 0.159649
iter 50 value 0.156148
iter 60 value 0.150442
iter 70 value 0.147622
iter 80 value 0.143764
iter 90 value 0.143089
iter 100 value 0.138737
final value 0.138737
stopped after 100 iterations
# weights: 11
initial value 321.711475
iter 10 value 106.787445
iter 20 value 105.863882
iter 30 value 104.297782
iter 40 value 101.180898
iter 50 value 98.600306
iter 60 value 54.438264
iter 70 value 24.912915
iter 80 value 22.886634
iter 90 value 22.653035
iter 100 value 21.189134
final value 21.189134
stopped after 100 iterations
# weights: 27
initial value 297.872372
iter 10 value 3.313762
iter 20 value 0.072945
iter 30 value 0.005375
iter 40 value 0.000221
final value 0.000068
converged
# weights: 43
initial value 298.521735
iter 10 value 2.671229
iter 20 value 0.010784
iter 30 value 0.000127
final value 0.000037
converged
# weights: 11

```

```
initial value 319.025278
iter 10 value 142.424868
iter 20 value 119.935553
iter 30 value 102.829230
iter 40 value 101.092513
iter 40 value 101.092513
final value 101.092513
converged
# weights: 27
initial value 297.017554
iter 10 value 34.114177
iter 20 value 20.930764
iter 30 value 18.878992
iter 40 value 18.382029
iter 50 value 18.204910
final value 18.204756
converged
# weights: 43
initial value 293.861209
iter 10 value 21.272198
iter 20 value 16.082938
iter 30 value 15.943815
iter 40 value 15.940657
iter 50 value 15.939445
final value 15.939382
converged
# weights: 11
initial value 341.548423
iter 10 value 106.083418
iter 20 value 101.886968
iter 30 value 77.186256
iter 40 value 46.451853
iter 50 value 43.994940
iter 60 value 43.730813
iter 70 value 43.591164
iter 80 value 43.582267
iter 90 value 43.555324
iter 100 value 43.552403
final value 43.552403
stopped after 100 iterations
# weights: 27
initial value 339.256223
iter 10 value 20.694924
iter 20 value 0.486502
iter 30 value 0.228924
iter 40 value 0.195640
```

```

iter 50 value 0.161375
iter 60 value 0.142649
iter 70 value 0.133915
iter 80 value 0.126567
iter 90 value 0.118063
iter 100 value 0.102578
final value 0.102578
stopped after 100 iterations
# weights: 43
initial value 330.719857
iter 10 value 0.931942
iter 20 value 0.128551
iter 30 value 0.122789
iter 40 value 0.117121
iter 50 value 0.107921
iter 60 value 0.101031
iter 70 value 0.094360
iter 80 value 0.084646
iter 90 value 0.082321
iter 100 value 0.080230
final value 0.080230
stopped after 100 iterations
# weights: 11
initial value 279.422000
iter 10 value 64.802331
iter 20 value 27.946403
iter 30 value 26.236302
iter 40 value 25.475810
iter 50 value 25.299751
iter 60 value 25.284476
iter 70 value 25.207281
iter 80 value 25.184629
iter 90 value 25.162136
iter 100 value 25.124891
final value 25.124891
stopped after 100 iterations
# weights: 27
initial value 314.712004
iter 10 value 5.809146
iter 20 value 0.065983
iter 30 value 0.000235
iter 30 value 0.000073
iter 30 value 0.000073
final value 0.000073
converged
# weights: 43

```

```
initial value 343.840189
iter 10 value 3.357197
iter 20 value 0.202932
iter 30 value 0.006395
iter 40 value 0.000589
final value 0.000085
converged
# weights: 11
initial value 352.064032
iter 10 value 122.576109
iter 20 value 101.529355
iter 30 value 99.982053
final value 99.980388
converged
# weights: 27
initial value 333.354950
iter 10 value 30.865750
iter 20 value 23.083419
iter 30 value 22.711712
iter 40 value 22.706491
iter 50 value 22.706059
final value 22.706047
converged
# weights: 43
initial value 299.296419
iter 10 value 29.581524
iter 20 value 18.987895
iter 30 value 17.689937
iter 40 value 17.452644
iter 50 value 17.382867
iter 60 value 17.358193
final value 17.357716
converged
# weights: 11
initial value 297.732437
iter 10 value 110.471483
iter 20 value 100.398829
iter 30 value 99.575666
iter 40 value 90.049095
iter 50 value 42.695627
iter 60 value 27.997643
iter 70 value 26.415894
iter 80 value 26.004040
iter 90 value 25.919171
iter 100 value 25.867900
final value 25.867900
```

```

stopped after 100 iterations
# weights: 27
initial value 330.376969
iter 10 value 6.023687
iter 20 value 0.654618
iter 30 value 0.267471
iter 40 value 0.237677
iter 50 value 0.222480
iter 60 value 0.219798
iter 70 value 0.211778
iter 80 value 0.208943
iter 90 value 0.199705
iter 100 value 0.195579
final value 0.195579
stopped after 100 iterations
# weights: 43
initial value 347.831838
iter 10 value 3.066632
iter 20 value 0.312657
iter 30 value 0.239603
iter 40 value 0.210073
iter 50 value 0.190833
iter 60 value 0.183693
iter 70 value 0.179277
iter 80 value 0.176550
iter 90 value 0.173536
iter 100 value 0.170419
final value 0.170419
stopped after 100 iterations
# weights: 11
initial value 289.757611
iter 10 value 109.533196
iter 20 value 99.334668
iter 30 value 85.098953
iter 40 value 37.107124
iter 50 value 25.255289
iter 60 value 24.779091
iter 70 value 24.569470
iter 80 value 24.281514
iter 90 value 24.249343
iter 100 value 24.232671
final value 24.232671
stopped after 100 iterations
# weights: 27
initial value 327.349143
iter 10 value 5.295570

```

```

iter 20 value 0.181597
iter 30 value 0.007687
iter 40 value 0.000359
final value 0.000055
converged
# weights: 43
initial value 347.321793
iter 10 value 0.989233
iter 20 value 0.070753
iter 30 value 0.003645
iter 40 value 0.000347
iter 50 value 0.000118
iter 50 value 0.000079
iter 50 value 0.000078
final value 0.000078
converged
# weights: 11
initial value 336.279604
iter 10 value 120.635809
iter 20 value 109.527522
iter 30 value 102.395823
final value 102.268783
converged
# weights: 27
initial value 347.303166
iter 10 value 40.099207
iter 20 value 21.901094
iter 30 value 20.885242
iter 40 value 20.084801
iter 50 value 19.577686
iter 60 value 19.036507
iter 70 value 18.783240
final value 18.783059
converged
# weights: 43
initial value 312.149984
iter 10 value 20.313511
iter 20 value 17.767633
iter 30 value 17.428835
iter 40 value 17.378916
iter 50 value 17.359397
iter 60 value 17.358997
final value 17.358978
converged
# weights: 11
initial value 317.536232

```

```
iter 10 value 108.595922
iter 20 value 105.478594
iter 30 value 102.338043
iter 40 value 99.507609
iter 50 value 83.230489
iter 60 value 32.835341
iter 70 value 25.901446
iter 80 value 25.280314
iter 90 value 24.917159
iter 100 value 24.907615
final value 24.907615
stopped after 100 iterations
# weights: 27
initial value 269.740054
iter 10 value 34.967159
iter 20 value 0.824536
iter 30 value 0.596129
iter 40 value 0.472947
iter 50 value 0.413316
iter 60 value 0.350473
iter 70 value 0.335913
iter 80 value 0.302650
iter 90 value 0.264286
iter 100 value 0.235720
final value 0.235720
stopped after 100 iterations
# weights: 43
initial value 300.908858
iter 10 value 2.927369
iter 20 value 0.395035
iter 30 value 0.201021
iter 40 value 0.192006
iter 50 value 0.180614
iter 60 value 0.169275
iter 70 value 0.161822
iter 80 value 0.153872
iter 90 value 0.151605
iter 100 value 0.148250
final value 0.148250
stopped after 100 iterations
# weights: 11
initial value 305.880931
iter 10 value 110.871519
iter 20 value 106.393742
iter 30 value 105.485757
iter 40 value 103.971469
```

```

iter 50 value 99.083948
iter 60 value 64.882536
iter 70 value 45.792125
iter 80 value 45.320136
iter 90 value 44.843289
iter 100 value 44.835792
final value 44.835792
stopped after 100 iterations
# weights: 27
initial value 337.433180
iter 10 value 3.586809
iter 20 value 0.162249
iter 30 value 0.002176
iter 40 value 0.000801
iter 50 value 0.000108
final value 0.000082
converged

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-16') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```

# weights: 43
initial value 267.847752
iter 10 value 3.772908
iter 20 value 0.068072
iter 30 value 0.003339
iter 40 value 0.001007
final value 0.000098
converged
# weights: 11
initial value 295.964677
iter 10 value 119.526495
iter 20 value 102.379584
iter 30 value 100.067722
final value 100.059690
converged
# weights: 27
initial value 306.913983
iter 10 value 29.875414
iter 20 value 20.304832

```

```

iter 30 value 19.988043
iter 40 value 19.656258
iter 50 value 19.608676
final value 19.608674
converged
# weights: 43
initial value 307.633035
iter 10 value 33.311488
iter 20 value 18.595028
iter 30 value 17.685016
iter 40 value 17.657942
iter 50 value 17.654574
iter 60 value 17.654207
final value 17.654203
converged
# weights: 11
initial value 311.404429
iter 10 value 105.020288
iter 20 value 99.761418
iter 30 value 50.619524
iter 40 value 30.439174
iter 50 value 27.371216
iter 60 value 26.192194
iter 70 value 26.172500
iter 80 value 26.172080
iter 90 value 26.171034
final value 26.171034
converged
# weights: 27
initial value 302.799706
iter 10 value 4.436763
iter 20 value 0.257625
iter 30 value 0.241475
iter 40 value 0.237069
iter 50 value 0.222989
iter 60 value 0.214822
iter 70 value 0.209771
iter 80 value 0.200758
iter 90 value 0.194303
iter 100 value 0.188430
final value 0.188430
stopped after 100 iterations
# weights: 43
initial value 354.805931
iter 10 value 3.555975
iter 20 value 0.368303

```

```

iter 30 value 0.329114
iter 40 value 0.300086
iter 50 value 0.277280
iter 60 value 0.256535
iter 70 value 0.237988
iter 80 value 0.217327
iter 90 value 0.211007
iter 100 value 0.203793
final value 0.203793
stopped after 100 iterations
# weights: 11
initial value 283.275019
iter 10 value 116.661928
iter 20 value 101.061124
iter 30 value 97.883324
iter 40 value 96.201930
iter 50 value 95.599665
iter 60 value 95.241797
iter 70 value 94.862369
iter 80 value 94.844969
iter 90 value 94.787481
iter 100 value 94.697431
final value 94.697431
stopped after 100 iterations
# weights: 27
initial value 350.783464
iter 10 value 15.306736
iter 20 value 1.226822
iter 30 value 0.140516
iter 40 value 0.000905
final value 0.000056
converged
# weights: 43
initial value 343.694415
iter 10 value 2.097069
iter 20 value 0.170689
iter 30 value 0.013099
iter 40 value 0.003240
iter 50 value 0.000711
final value 0.000088
converged
# weights: 11
initial value 278.936965
iter 10 value 120.058922
iter 20 value 90.987295
iter 30 value 76.269537

```

```
iter 40 value 76.258978
final value 76.258976
converged
# weights: 27
initial value 286.599687
iter 10 value 31.275323
iter 20 value 21.251700
iter 30 value 20.868755
iter 40 value 20.820261
iter 50 value 20.817965
final value 20.817957
converged
# weights: 43
initial value 283.365690
iter 10 value 24.501898
iter 20 value 19.925802
iter 30 value 18.785164
iter 40 value 17.561774
iter 50 value 17.419069
iter 60 value 17.417563
final value 17.417562
converged
# weights: 11
initial value 323.830703
iter 10 value 78.991016
iter 20 value 32.410140
iter 30 value 24.548488
iter 40 value 23.497690
iter 50 value 23.448891
iter 60 value 23.300937
iter 70 value 23.285327
iter 80 value 23.282765
iter 90 value 23.274830
iter 100 value 23.266093
final value 23.266093
stopped after 100 iterations
# weights: 27
initial value 320.344567
iter 10 value 6.159707
iter 20 value 0.238463
iter 30 value 0.211328
iter 40 value 0.202544
iter 50 value 0.193376
iter 60 value 0.190139
iter 70 value 0.182224
iter 80 value 0.164919
```

```

iter 90 value 0.156845
iter 100 value 0.154176
final value 0.154176
stopped after 100 iterations
# weights: 43
initial value 323.473910
iter 10 value 2.102237
iter 20 value 0.362438
iter 30 value 0.295754
iter 40 value 0.283881
iter 50 value 0.271154
iter 60 value 0.242809
iter 70 value 0.210431
iter 80 value 0.178519
iter 90 value 0.168283
iter 100 value 0.161430
final value 0.161430
stopped after 100 iterations
# weights: 11
initial value 290.977599
iter 10 value 116.998193
iter 20 value 109.197456
iter 30 value 109.009390
iter 40 value 106.613751
iter 50 value 101.086754
iter 60 value 92.747249
iter 70 value 33.547468
iter 80 value 28.650206
iter 90 value 28.228936
iter 100 value 27.865208
final value 27.865208
stopped after 100 iterations
# weights: 27
initial value 318.606119
iter 10 value 17.761583
iter 20 value 0.221247
iter 30 value 0.000262
final value 0.000054
converged
# weights: 43
initial value 337.306063
iter 10 value 14.404516
iter 20 value 0.426554
iter 30 value 0.001689
iter 40 value 0.000860
iter 50 value 0.000678

```

```
final value 0.000056
converged
# weights: 11
initial value 276.723653
iter 10 value 102.607573
iter 20 value 79.550229
iter 30 value 79.040510
final value 79.040016
converged
# weights: 27
initial value 300.214044
iter 10 value 69.758243
iter 20 value 21.956782
iter 30 value 21.578730
iter 40 value 21.557855
iter 50 value 21.496973
iter 60 value 20.910770
final value 20.908367
converged
# weights: 43
initial value 323.576854
iter 10 value 27.910676
iter 20 value 17.274750
iter 30 value 17.022504
iter 40 value 17.012066
iter 50 value 17.011217
final value 17.011203
converged
# weights: 11
initial value 283.911769
iter 10 value 128.816082
iter 20 value 108.395097
iter 30 value 108.165221
iter 40 value 108.040900
iter 50 value 107.959256
iter 60 value 107.893441
iter 70 value 107.847110
final value 107.847109
converged
# weights: 27
initial value 398.120428
iter 10 value 4.193121
iter 20 value 0.396713
iter 30 value 0.289352
iter 40 value 0.271305
iter 50 value 0.264756
```

```

iter 60 value 0.224065
iter 70 value 0.204829
iter 80 value 0.199310
iter 90 value 0.178129
iter 100 value 0.166868
final value 0.166868
stopped after 100 iterations
# weights: 43
initial value 303.617968
iter 10 value 3.838289
iter 20 value 0.190433
iter 30 value 0.168660
iter 40 value 0.163801
iter 50 value 0.157585
iter 60 value 0.153931
iter 70 value 0.152021
iter 80 value 0.150419
iter 90 value 0.148155
iter 100 value 0.147603
final value 0.147603
stopped after 100 iterations
# weights: 11
initial value 289.897291
iter 10 value 102.667097
iter 20 value 97.906791
iter 30 value 97.588843
iter 40 value 97.571133
final value 97.570689
converged
# weights: 27
initial value 363.837862
iter 10 value 1.504524
iter 20 value 0.016070
iter 30 value 0.000352
final value 0.000058
converged
# weights: 43
initial value 338.589094
iter 10 value 0.279356
iter 20 value 0.038695
iter 30 value 0.003190
iter 40 value 0.000177
iter 50 value 0.000120
iter 50 value 0.000092
iter 50 value 0.000092
final value 0.000092

```

```

converged
# weights: 11
initial value 315.704466
iter 10 value 122.971896
iter 20 value 100.801386
iter 30 value 76.757087
iter 40 value 76.316263
iter 40 value 76.316263
iter 40 value 76.316263
final value 76.316263
converged
# weights: 27
initial value 316.841853
iter 10 value 40.219186
iter 20 value 18.912700
iter 30 value 17.592063
iter 40 value 17.165840
iter 50 value 16.911871
iter 60 value 16.889110
iter 70 value 16.886043
final value 16.886041
converged
# weights: 43
initial value 292.237083
iter 10 value 21.493530
iter 20 value 16.060406
iter 30 value 15.495569
iter 40 value 15.467113
iter 50 value 15.460041
iter 60 value 15.456238
iter 70 value 15.456163
final value 15.456153
converged
# weights: 11
initial value 306.108801
iter 10 value 106.661741
iter 20 value 106.471507
iter 30 value 106.438277
iter 40 value 106.363232
iter 50 value 106.343417
iter 60 value 106.333115
iter 70 value 106.253964
iter 80 value 71.860877
iter 90 value 31.391563
iter 100 value 20.732818
final value 20.732818

```

```

stopped after 100 iterations
# weights: 27
initial value 289.126792
iter 10 value 5.312101
iter 20 value 0.454477
iter 30 value 0.282954
iter 40 value 0.253066
iter 50 value 0.205347
iter 60 value 0.187871
iter 70 value 0.175052
iter 80 value 0.159381
iter 90 value 0.123449
iter 100 value 0.105212
final value 0.105212
stopped after 100 iterations
# weights: 43
initial value 362.671484
iter 10 value 3.117441
iter 20 value 0.181494
iter 30 value 0.163761
iter 40 value 0.149269
iter 50 value 0.127031
iter 60 value 0.112053
iter 70 value 0.103372
iter 80 value 0.095389
iter 90 value 0.088623
iter 100 value 0.083950
final value 0.083950
stopped after 100 iterations
# weights: 11
initial value 314.877715
iter 10 value 108.181502
iter 20 value 93.028002
iter 30 value 92.469108
iter 40 value 88.467285
iter 50 value 26.433701
iter 60 value 23.071670
iter 70 value 22.668206
iter 80 value 22.640393
iter 90 value 22.538386
iter 100 value 22.533276
final value 22.533276
stopped after 100 iterations
# weights: 27
initial value 313.434553
iter 10 value 0.404873

```

```

iter 20 value 0.015960
iter 30 value 0.001478
iter 40 value 0.000155
final value 0.000082
converged
# weights: 43
initial value 367.077425
iter 10 value 1.132177
iter 20 value 0.030029
iter 30 value 0.000539
final value 0.000078
converged
# weights: 11
initial value 303.811269
iter 10 value 127.839188
iter 20 value 118.076369
iter 30 value 103.432346
iter 40 value 101.122495
iter 40 value 101.122495
iter 40 value 101.122495
final value 101.122495
converged
# weights: 27
initial value 361.038178
iter 10 value 43.203229
iter 20 value 18.326164
iter 30 value 17.931069
iter 40 value 17.847620
iter 50 value 17.829960
final value 17.829956
converged
# weights: 43
initial value 316.259407
iter 10 value 33.929536
iter 20 value 17.665270
iter 30 value 16.781225
iter 40 value 16.551916
iter 50 value 16.513396
iter 60 value 16.509896
iter 70 value 16.509539
final value 16.509515
converged
# weights: 11
initial value 344.972949
iter 10 value 108.653549
iter 20 value 108.018825

```

```

iter 30 value 107.981049
iter 40 value 107.968485
iter 50 value 107.840158
iter 60 value 107.821350
iter 70 value 107.585414
iter 80 value 91.339642
iter 90 value 48.234771
iter 100 value 45.664840
final value 45.664840
stopped after 100 iterations
# weights: 27
initial value 289.880639
iter 10 value 0.967125
iter 20 value 0.294539
iter 30 value 0.253586
iter 40 value 0.186883
iter 50 value 0.158261
iter 60 value 0.148588
iter 70 value 0.138677
iter 80 value 0.127564
iter 90 value 0.114256
iter 100 value 0.094974
final value 0.094974
stopped after 100 iterations
# weights: 43
initial value 252.340834
iter 10 value 1.452946
iter 20 value 0.100320
iter 30 value 0.094196
iter 40 value 0.092027
iter 50 value 0.089866
iter 60 value 0.087083
iter 70 value 0.084012
iter 80 value 0.082719
iter 90 value 0.081712
iter 100 value 0.080940
final value 0.080940
stopped after 100 iterations

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-17') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```

# weights: 11
initial value 388.611300
iter 10 value 72.013872
iter 20 value 28.026959
iter 30 value 22.143735
iter 40 value 20.292817
iter 50 value 20.187762
iter 60 value 20.133690
iter 70 value 20.004232
iter 80 value 19.997233
iter 90 value 19.982661
iter 100 value 19.926156
final value 19.926156
stopped after 100 iterations
# weights: 27
initial value 309.252514
iter 10 value 5.532968
iter 20 value 0.914381
iter 30 value 0.020388
iter 40 value 0.000301
final value 0.000069
converged
# weights: 43
initial value 339.024900
iter 10 value 3.954062
iter 20 value 0.122611
final value 0.000097
converged
# weights: 11
initial value 280.291638
iter 10 value 125.757753
iter 20 value 105.221227
iter 30 value 76.492482
iter 40 value 75.313079
final value 75.313078
converged
# weights: 27
initial value 303.203702
iter 10 value 40.551175
iter 20 value 21.655372
iter 30 value 19.110408
iter 40 value 18.871267
iter 50 value 18.796819
iter 60 value 18.771863
final value 18.771831
converged

```

```

# weights: 43
initial value 336.012878
iter 10 value 28.944969
iter 20 value 17.278923
iter 30 value 16.926393
iter 40 value 16.822692
iter 50 value 16.741169
iter 60 value 16.734143
iter 70 value 16.734026
final value 16.734026
converged
# weights: 11
initial value 298.610277
iter 10 value 126.170381
iter 20 value 107.500563
iter 30 value 107.468177
iter 40 value 107.462265
iter 50 value 107.014500
iter 60 value 78.704641
iter 70 value 35.383843
iter 80 value 22.221249
iter 90 value 21.142197
iter 100 value 20.833755
final value 20.833755
stopped after 100 iterations
# weights: 27
initial value 325.583355
iter 10 value 0.703754
iter 20 value 0.432265
iter 30 value 0.395454
iter 40 value 0.301301
iter 50 value 0.271044
iter 60 value 0.218775
iter 70 value 0.195969
iter 80 value 0.183732
iter 90 value 0.179420
iter 100 value 0.173492
final value 0.173492
stopped after 100 iterations
# weights: 43
initial value 319.567532
iter 10 value 11.039054
iter 20 value 0.875256
iter 30 value 0.506518
iter 40 value 0.407981
iter 50 value 0.322403

```

```

iter 60 value 0.285239
iter 70 value 0.249609
iter 80 value 0.202071
iter 90 value 0.192628
iter 100 value 0.187853
final value 0.187853
stopped after 100 iterations
# weights: 11
initial value 316.271973
iter 10 value 115.193171
iter 20 value 97.222797
iter 30 value 96.898819
iter 40 value 96.740691
iter 50 value 96.684827
iter 60 value 96.680769
iter 70 value 96.660859
iter 70 value 96.660859
final value 96.660859
converged
# weights: 27
initial value 337.247906
iter 10 value 6.838494
iter 20 value 0.039082
iter 30 value 0.002082
final value 0.000079
converged
# weights: 43
initial value 330.703475
iter 10 value 0.754760
iter 20 value 0.006040
iter 30 value 0.002117
iter 40 value 0.000647
iter 50 value 0.000369
iter 60 value 0.000334
final value 0.000065
converged
# weights: 11
initial value 319.548229
iter 10 value 168.384168
iter 20 value 117.413568
iter 30 value 117.115350
iter 40 value 117.048884
iter 50 value 111.605085
iter 60 value 90.148820
iter 70 value 75.567746
final value 75.536961

```

```

converged
# weights: 27
initial value 316.630359
iter 10 value 37.609008
iter 20 value 20.041497
iter 30 value 19.708390
iter 40 value 19.671522
final value 19.671234
converged
# weights: 43
initial value 363.555841
iter 10 value 42.486007
iter 20 value 17.490138
iter 30 value 15.873669
iter 40 value 15.846110
iter 50 value 15.837196
iter 60 value 15.835112
iter 70 value 15.834809
iter 70 value 15.834808
iter 70 value 15.834808
final value 15.834808
converged
# weights: 11
initial value 326.603262
iter 10 value 105.017266
iter 20 value 93.998585
iter 30 value 91.264548
iter 40 value 85.076595
iter 50 value 29.755329
iter 60 value 24.489362
iter 70 value 23.263291
iter 80 value 22.984305
iter 90 value 22.764460
iter 100 value 22.533068
final value 22.533068
stopped after 100 iterations
# weights: 27
initial value 285.762170
iter 10 value 8.742057
iter 20 value 0.226610
iter 30 value 0.206399
iter 40 value 0.188178
iter 50 value 0.168516
iter 60 value 0.158279
iter 70 value 0.150909
iter 80 value 0.134793

```

```

iter 90 value 0.122991
iter 100 value 0.114206
final value 0.114206
stopped after 100 iterations
# weights: 43
initial value 318.073411
iter 10 value 1.507030
iter 20 value 0.131600
iter 30 value 0.125032
iter 40 value 0.122659
iter 50 value 0.117068
iter 60 value 0.104620
iter 70 value 0.099980
iter 80 value 0.095452
iter 90 value 0.089832
iter 100 value 0.088183
final value 0.088183
stopped after 100 iterations
# weights: 11
initial value 291.564772
iter 10 value 108.455243
iter 20 value 94.889668
iter 30 value 36.558541
iter 40 value 28.796010
iter 50 value 28.318549
iter 60 value 27.969747
iter 70 value 27.922720
iter 80 value 27.891109
iter 90 value 27.829624
iter 100 value 27.820071
final value 27.820071
stopped after 100 iterations
# weights: 27
initial value 336.927683
iter 10 value 24.785627
iter 20 value 1.947319
iter 30 value 0.162362
iter 40 value 0.005721
iter 50 value 0.001118
final value 0.000093
converged
# weights: 43
initial value 261.742984
iter 10 value 3.705536
iter 20 value 0.272190
iter 30 value 0.015300

```

```

iter 40 value 0.004281
iter 50 value 0.001594
iter 60 value 0.000243
final value 0.000075
converged
# weights: 11
initial value 318.230882
iter 10 value 118.479524
iter 20 value 109.140475
iter 30 value 104.384192
final value 104.019460
converged
# weights: 27
initial value 401.108513
iter 10 value 31.021397
iter 20 value 21.004694
iter 30 value 20.161578
iter 40 value 19.950039
iter 50 value 19.913945
final value 19.913919
converged
# weights: 43
initial value 309.865204
iter 10 value 51.804924
iter 20 value 19.758998
iter 30 value 17.948128
iter 40 value 17.918288
iter 50 value 17.913084
iter 60 value 17.911174
iter 70 value 17.905471
iter 80 value 17.901118
final value 17.901117
converged
# weights: 11
initial value 289.747705
iter 10 value 109.431147
iter 20 value 106.900828
iter 30 value 106.861423
iter 40 value 106.787906
iter 50 value 106.731845
iter 60 value 100.755250
iter 70 value 90.123900
iter 80 value 42.042185
iter 90 value 29.406354
iter 100 value 28.570249
final value 28.570249

```

```

stopped after 100 iterations
# weights: 27
initial value 306.392677
iter 10 value 4.751042
iter 20 value 0.315879
iter 30 value 0.274682
iter 40 value 0.249699
iter 50 value 0.232961
iter 60 value 0.227482
iter 70 value 0.216577
iter 80 value 0.196703
iter 90 value 0.182110
iter 100 value 0.174164
final value 0.174164
stopped after 100 iterations
# weights: 43
initial value 316.038009
iter 10 value 4.545831
iter 20 value 0.445403
iter 30 value 0.265683
iter 40 value 0.247321
iter 50 value 0.243187
iter 60 value 0.231493
iter 70 value 0.217387
iter 80 value 0.202258
iter 90 value 0.188430
iter 100 value 0.184210
final value 0.184210
stopped after 100 iterations
# weights: 11
initial value 283.565780
iter 10 value 107.398016
iter 20 value 106.645235
final value 106.644789
converged
# weights: 27
initial value 280.547425
iter 10 value 4.191715
iter 20 value 0.145701
iter 30 value 0.003187
final value 0.000098
converged
# weights: 43
initial value 299.207315
iter 10 value 6.996728
iter 20 value 0.311939

```

```

iter 30 value 0.010544
iter 40 value 0.003059
iter 50 value 0.001091
final value 0.000088
converged
# weights: 11
initial value 274.173786
iter 10 value 121.014567
iter 20 value 79.605945
iter 30 value 79.497297
final value 79.497062
converged
# weights: 27
initial value 296.759211
iter 10 value 32.451605
iter 20 value 21.290390
iter 30 value 20.173438
iter 40 value 19.531461
iter 50 value 19.508690
final value 19.508677
converged
# weights: 43
initial value 318.484169
iter 10 value 43.000541
iter 20 value 18.935419
iter 30 value 17.659963
iter 40 value 17.500928
iter 50 value 17.481971
iter 60 value 17.474140
final value 17.474133
converged
# weights: 11
initial value 366.863066
iter 10 value 113.755634
iter 20 value 58.726644
iter 30 value 37.281074
iter 40 value 29.977309
iter 50 value 29.415620
iter 60 value 29.327778
iter 70 value 29.129766
iter 80 value 29.128608
iter 90 value 29.125141
final value 29.125108
converged
# weights: 27
initial value 401.910897

```

```
iter 10 value 18.053265
iter 20 value 0.526894
iter 30 value 0.432008
iter 40 value 0.388424
iter 50 value 0.293481
iter 60 value 0.265157
iter 70 value 0.252798
iter 80 value 0.226240
iter 90 value 0.205097
iter 100 value 0.188704
final value 0.188704
stopped after 100 iterations
# weights: 43
initial value 360.895732
iter 10 value 4.010178
iter 20 value 0.460165
iter 30 value 0.337821
iter 40 value 0.295755
iter 50 value 0.279655
iter 60 value 0.268037
iter 70 value 0.254469
iter 80 value 0.213258
iter 90 value 0.204366
iter 100 value 0.193642
final value 0.193642
stopped after 100 iterations
# weights: 11
initial value 327.362036
iter 10 value 47.489514
iter 20 value 30.794310
iter 30 value 27.902031
iter 40 value 26.829000
iter 50 value 26.077572
iter 60 value 26.059479
iter 70 value 25.952863
iter 80 value 25.914690
iter 90 value 25.893591
iter 100 value 25.862190
final value 25.862190
stopped after 100 iterations
# weights: 27
initial value 305.724340
iter 10 value 30.516958
iter 20 value 0.557945
iter 30 value 0.027816
iter 40 value 0.000385
```

```

iter 50 value 0.000281
iter 60 value 0.000125
iter 70 value 0.000102
final value 0.000099
converged
# weights: 43
initial value 271.164427
iter 10 value 3.041698
iter 20 value 0.069572
iter 30 value 0.006355
iter 40 value 0.000643
iter 50 value 0.000426
final value 0.000077
converged
# weights: 11
initial value 293.279735
iter 10 value 123.733904
iter 20 value 115.366987
iter 30 value 104.264279
iter 40 value 103.021000
final value 103.020998
converged
# weights: 27
initial value 361.689632
iter 10 value 81.514058
iter 20 value 24.834341
iter 30 value 20.907740
iter 40 value 20.383707
final value 20.379730
converged
# weights: 43
initial value 306.969948
iter 10 value 22.600761
iter 20 value 17.674299
iter 30 value 17.444192
iter 40 value 17.373262
iter 50 value 17.348841
iter 60 value 17.346312
final value 17.346302
converged
# weights: 11
initial value 294.228207
iter 10 value 153.620257
iter 20 value 120.408306
iter 30 value 110.021149
iter 40 value 108.213237

```

```
iter 50 value 108.113267
iter 60 value 108.047194
iter 70 value 107.757050
iter 80 value 107.742063
iter 90 value 107.741027
iter 90 value 107.741027
final value 107.741027
converged
```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-18') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```
# weights: 27
initial value 333.162171
iter 10 value 14.193546
iter 20 value 0.570136
iter 30 value 0.324892
iter 40 value 0.295967
iter 50 value 0.263928
iter 60 value 0.245162
iter 70 value 0.234763
iter 80 value 0.220778
iter 90 value 0.206844
iter 100 value 0.197848
final value 0.197848
stopped after 100 iterations
# weights: 43
initial value 290.469698
iter 10 value 1.724725
iter 20 value 0.407189
iter 30 value 0.229451
iter 40 value 0.215046
iter 50 value 0.210652
iter 60 value 0.199031
iter 70 value 0.187126
iter 80 value 0.178167
iter 90 value 0.170886
iter 100 value 0.165527
final value 0.165527
stopped after 100 iterations
```

```

# weights: 11
initial value 301.623745
iter 10 value 170.648120
iter 20 value 162.163452
iter 30 value 153.300729
iter 40 value 114.632688
iter 50 value 55.724282
iter 60 value 46.698048
iter 70 value 46.491050
iter 80 value 46.234794
final value 46.214972
converged
# weights: 27
initial value 323.915096
iter 10 value 5.137818
iter 20 value 0.133981
iter 30 value 0.007919
iter 40 value 0.001469
iter 50 value 0.000280
final value 0.000075
converged
# weights: 43
initial value 320.226972
iter 10 value 6.909700
iter 20 value 0.291156
iter 30 value 0.006835
iter 40 value 0.002824
final value 0.000070
converged
# weights: 11
initial value 292.859191
iter 10 value 120.981129
iter 20 value 88.109098
iter 30 value 79.424736
iter 40 value 78.937199
iter 40 value 78.937199
iter 40 value 78.937199
final value 78.937199
converged
# weights: 27
initial value 288.320168
iter 10 value 40.319732
iter 20 value 22.727041
iter 30 value 22.006969
iter 40 value 21.979708
iter 50 value 21.977693

```

```

iter 60 value 21.972637
final value 21.972598
converged
# weights: 43
initial value 332.533971
iter 10 value 27.609978
iter 20 value 18.243219
iter 30 value 17.725408
iter 40 value 17.656311
iter 50 value 17.649388
iter 60 value 17.648906
final value 17.648862
converged
# weights: 11
initial value 298.728783
iter 10 value 106.909626
iter 20 value 106.534313
iter 30 value 104.795187
iter 40 value 78.881240
iter 50 value 50.864869
iter 60 value 47.294449
iter 70 value 46.660773
iter 80 value 46.641405
iter 90 value 46.631743
iter 100 value 46.618533
final value 46.618533
stopped after 100 iterations
# weights: 27
initial value 375.211047
iter 10 value 23.383912
iter 20 value 4.936889
iter 30 value 0.392222
iter 40 value 0.294421
iter 50 value 0.269862
iter 60 value 0.232496
iter 70 value 0.215877
iter 80 value 0.206426
iter 90 value 0.201001
iter 100 value 0.192530
final value 0.192530
stopped after 100 iterations
# weights: 43
initial value 316.676590
iter 10 value 23.507083
iter 20 value 5.321259
iter 30 value 2.188480

```

```

iter 40 value 0.473504
iter 50 value 0.387461
iter 60 value 0.357489
iter 70 value 0.275966
iter 80 value 0.220218
iter 90 value 0.189803
iter 100 value 0.176680
final value 0.176680
stopped after 100 iterations
# weights: 11
initial value 300.143482
iter 10 value 107.368606
iter 20 value 102.659174
iter 30 value 40.283857
iter 40 value 18.905632
iter 50 value 17.693450
iter 60 value 16.978277
iter 70 value 16.091189
iter 80 value 15.672873
iter 90 value 15.501946
iter 100 value 15.488100
final value 15.488100
stopped after 100 iterations
# weights: 27
initial value 293.435833
iter 10 value 12.549609
iter 20 value 0.017742
iter 30 value 0.000668
iter 40 value 0.000247
final value 0.000098
converged
# weights: 43
initial value 507.193686
iter 10 value 29.609939
iter 20 value 11.220492
iter 30 value 0.045804
iter 40 value 0.005974
iter 50 value 0.000377
final value 0.000065
converged
# weights: 11
initial value 317.289529
iter 10 value 117.691758
iter 20 value 88.491489
iter 30 value 76.270479
final value 76.174260

```

```

converged
# weights: 27
initial value 296.662053
iter 10 value 33.411861
iter 20 value 20.226344
iter 30 value 19.781665
iter 40 value 19.722503
iter 50 value 19.577724
iter 60 value 19.529963
final value 19.529858
converged
# weights: 43
initial value 325.175719
iter 10 value 27.542001
iter 20 value 16.907581
iter 30 value 16.101727
iter 40 value 16.066250
iter 50 value 16.064028
iter 60 value 16.062260
iter 70 value 16.062202
final value 16.062180
converged
# weights: 11
initial value 357.224320
iter 10 value 109.172838
iter 20 value 106.036695
iter 30 value 105.505962
iter 40 value 104.974647
iter 50 value 96.661727
iter 60 value 52.123714
iter 70 value 46.880803
iter 80 value 46.405012
iter 90 value 45.972535
iter 100 value 45.965770
final value 45.965770
stopped after 100 iterations
# weights: 27
initial value 270.012610
iter 10 value 0.343801
iter 20 value 0.150146
iter 30 value 0.140120
iter 40 value 0.125571
iter 50 value 0.113800
iter 60 value 0.107448
iter 70 value 0.102618
iter 80 value 0.098396

```

```

iter 90 value 0.089557
iter 100 value 0.086006
final value 0.086006
stopped after 100 iterations
# weights: 43
initial value 279.246147
iter 10 value 3.636359
iter 20 value 0.277955
iter 30 value 0.261207
iter 40 value 0.239752
iter 50 value 0.187628
iter 60 value 0.167745
iter 70 value 0.149335
iter 80 value 0.134598
iter 90 value 0.115088
iter 100 value 0.109701
final value 0.109701
stopped after 100 iterations
# weights: 11
initial value 307.005084
iter 10 value 165.563015
iter 20 value 150.074094
iter 30 value 114.325645
iter 40 value 47.691589
iter 50 value 43.858709
iter 60 value 43.327559
iter 70 value 42.994646
iter 80 value 42.990856
iter 90 value 42.943605
iter 100 value 42.931223
final value 42.931223
stopped after 100 iterations
# weights: 27
initial value 344.860376
iter 10 value 12.779218
iter 20 value 0.027155
iter 30 value 0.005349
iter 40 value 0.001326
iter 50 value 0.000189
final value 0.000075
converged
# weights: 43
initial value 284.750281
iter 10 value 0.902507
iter 20 value 0.007927
iter 30 value 0.002418

```

```
iter 40 value 0.000539
iter 50 value 0.000493
final value 0.000095
converged
# weights: 11
initial value 316.212799
iter 10 value 204.702933
iter 20 value 123.524093
iter 30 value 106.140895
iter 40 value 101.078699
final value 101.078537
converged
# weights: 27
initial value 305.715481
iter 10 value 31.388350
iter 20 value 18.423419
iter 30 value 17.616597
iter 40 value 17.516402
iter 50 value 17.509579
final value 17.509577
converged
# weights: 43
initial value 269.611982
iter 10 value 22.095604
iter 20 value 15.854812
iter 30 value 15.730314
iter 40 value 15.709482
iter 50 value 15.707935
iter 60 value 15.706645
final value 15.706643
converged
# weights: 11
initial value 317.224392
iter 10 value 110.943475
iter 20 value 106.249975
iter 30 value 105.704519
iter 40 value 105.543917
iter 50 value 105.140413
iter 60 value 104.386816
iter 70 value 103.933176
iter 80 value 100.423102
iter 90 value 66.815238
iter 100 value 46.003674
final value 46.003674
stopped after 100 iterations
# weights: 27
```

```
initial value 294.872909
iter 10 value 1.060168
iter 20 value 0.211720
iter 30 value 0.190995
iter 40 value 0.173100
iter 50 value 0.148884
iter 60 value 0.138531
iter 70 value 0.128342
iter 80 value 0.117486
iter 90 value 0.105036
iter 100 value 0.098139
final value 0.098139
stopped after 100 iterations
# weights: 43
initial value 339.269712
iter 10 value 1.677518
iter 20 value 0.105495
iter 30 value 0.088914
iter 40 value 0.086515
iter 50 value 0.083762
iter 60 value 0.081629
iter 70 value 0.080399
iter 80 value 0.077646
iter 90 value 0.075019
iter 100 value 0.073483
final value 0.073483
stopped after 100 iterations
# weights: 11
initial value 286.311768
iter 10 value 115.415718
iter 20 value 84.020100
iter 30 value 38.716465
iter 40 value 30.066844
iter 50 value 29.621951
iter 60 value 29.084613
iter 70 value 28.979954
iter 80 value 28.943349
iter 90 value 28.897924
iter 100 value 28.856849
final value 28.856849
stopped after 100 iterations
# weights: 27
initial value 267.877971
iter 10 value 5.872783
iter 20 value 0.054527
iter 30 value 0.015753
```

```
iter 40 value 0.003983
iter 50 value 0.000419
final value 0.000081
converged
# weights: 43
initial value 276.373845
iter 10 value 5.187889
iter 20 value 0.377904
iter 30 value 0.026019
iter 40 value 0.005736
iter 50 value 0.002653
iter 60 value 0.001165
iter 70 value 0.000625
iter 80 value 0.000362
final value 0.000089
converged
# weights: 11
initial value 286.838066
iter 10 value 90.755791
iter 20 value 80.293298
iter 30 value 80.292148
final value 80.292140
converged
# weights: 27
initial value 282.899427
iter 10 value 51.005169
iter 20 value 22.273888
iter 30 value 20.766020
iter 40 value 20.519434
iter 50 value 20.157558
iter 60 value 19.796412
iter 70 value 19.275163
iter 80 value 19.229367
final value 19.229365
converged
# weights: 43
initial value 289.366265
iter 10 value 38.346343
iter 20 value 18.977232
iter 30 value 17.998055
iter 40 value 17.890359
iter 50 value 17.340585
iter 60 value 17.301236
iter 70 value 17.299209
iter 80 value 17.296819
iter 90 value 17.296583
```

```
final value 17.296558
converged
# weights: 11
initial value 299.887250
iter 10 value 107.988992
iter 20 value 105.341017
iter 30 value 79.655804
iter 40 value 44.068842
iter 50 value 35.620621
iter 60 value 30.236339
iter 70 value 29.958325
iter 80 value 29.784071
iter 90 value 29.650590
iter 100 value 29.648352
final value 29.648352
stopped after 100 iterations
# weights: 27
initial value 352.480977
iter 10 value 1.462515
iter 20 value 0.386290
iter 30 value 0.347371
iter 40 value 0.242565
iter 50 value 0.203751
iter 60 value 0.190581
iter 70 value 0.180494
iter 80 value 0.165276
iter 90 value 0.160225
iter 100 value 0.153282
final value 0.153282
stopped after 100 iterations
# weights: 43
initial value 280.698741
iter 10 value 7.000019
iter 20 value 0.347247
iter 30 value 0.198230
iter 40 value 0.186098
iter 50 value 0.176328
iter 60 value 0.169051
iter 70 value 0.163708
iter 80 value 0.159608
iter 90 value 0.158702
iter 100 value 0.154305
final value 0.154305
stopped after 100 iterations
# weights: 11
initial value 310.668988
```

```
iter 10 value 111.931613
iter 20 value 107.414046
iter 30 value 107.405529
final value 107.405500
converged
# weights: 27
initial value 297.101207
iter 10 value 0.641749
iter 20 value 0.079614
iter 30 value 0.020347
iter 40 value 0.002275
iter 50 value 0.000834
iter 60 value 0.000299
iter 70 value 0.000239
iter 80 value 0.000123
iter 90 value 0.000114
iter 90 value 0.000084
iter 90 value 0.000084
final value 0.000084
converged
# weights: 43
initial value 292.578855
iter 10 value 3.189598
iter 20 value 0.012349
iter 30 value 0.002121
iter 40 value 0.000843
iter 50 value 0.000314
iter 60 value 0.000188
final value 0.000075
converged
# weights: 11
initial value 305.795127
iter 10 value 143.138923
iter 20 value 83.951738
iter 30 value 77.455236
final value 77.324855
converged
# weights: 27
initial value 306.124330
iter 10 value 31.294972
iter 20 value 19.515347
iter 30 value 18.139434
iter 40 value 17.954294
iter 50 value 17.951807
final value 17.951122
converged
```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-19') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```
# weights: 43
initial value 338.076088
iter 10 value 24.268982
iter 20 value 16.711471
iter 30 value 16.262951
iter 40 value 16.242935
iter 50 value 16.240954
iter 60 value 16.240430
final value 16.240414
converged
# weights: 11
initial value 285.457751
iter 10 value 97.810865
iter 20 value 74.112875
iter 30 value 30.130161
iter 40 value 26.533748
iter 50 value 26.134644
iter 60 value 25.929047
iter 70 value 25.920875
iter 80 value 25.910253
final value 25.910063
converged
# weights: 27
initial value 288.806889
iter 10 value 3.178877
iter 20 value 0.204166
iter 30 value 0.127517
iter 40 value 0.116965
iter 50 value 0.113572
iter 60 value 0.112693
iter 70 value 0.110862
iter 80 value 0.107825
iter 90 value 0.106732
iter 100 value 0.105294
final value 0.105294
stopped after 100 iterations
# weights: 43
```

```

initial value 310.074333
iter 10 value 1.597044
iter 20 value 0.122182
iter 30 value 0.108029
iter 40 value 0.107260
iter 50 value 0.103937
iter 60 value 0.102219
iter 70 value 0.100998
iter 80 value 0.098786
iter 90 value 0.094563
iter 100 value 0.091544
final value 0.091544
stopped after 100 iterations
# weights: 11
initial value 303.992911
iter 10 value 91.418357
iter 20 value 37.370045
iter 30 value 27.723450
iter 40 value 26.343460
iter 50 value 26.135157
iter 60 value 26.037301
iter 70 value 25.838175
iter 80 value 25.831686
iter 90 value 25.802194
iter 100 value 25.777248
final value 25.777248
stopped after 100 iterations
# weights: 27
initial value 311.000988
iter 10 value 2.211538
iter 20 value 0.038781
iter 30 value 0.005679
iter 40 value 0.000313
iter 50 value 0.000106
iter 50 value 0.000085
iter 50 value 0.000084
final value 0.000084
converged
# weights: 43
initial value 377.134031
iter 10 value 11.399835
iter 20 value 0.070354
iter 30 value 0.008999
iter 40 value 0.002599
iter 50 value 0.001279
iter 60 value 0.000883

```

```

iter 70 value 0.000169
iter 80 value 0.000159
final value 0.000076
converged
# weights: 11
initial value 290.407716
iter 10 value 122.019011
iter 20 value 93.450180
iter 30 value 78.200497
iter 40 value 77.471597
iter 40 value 77.471597
iter 40 value 77.471597
final value 77.471597
converged
# weights: 27
initial value 306.185318
iter 10 value 50.795070
iter 20 value 21.021322
iter 30 value 20.508022
iter 40 value 20.490388
iter 50 value 20.480966
iter 50 value 20.480966
iter 50 value 20.480966
final value 20.480966
converged
# weights: 43
initial value 273.570197
iter 10 value 53.195578
iter 20 value 19.144489
iter 30 value 17.284862
iter 40 value 17.036369
iter 50 value 16.811375
iter 60 value 16.771611
iter 70 value 16.750069
iter 80 value 16.749223
final value 16.748898
converged
# weights: 11
initial value 295.421201
iter 10 value 128.530819
iter 20 value 112.427519
iter 30 value 107.855261
iter 40 value 100.522972
iter 50 value 85.139339
iter 60 value 37.245501
iter 70 value 27.384130

```

```

iter 80 value 26.776044
iter 90 value 26.390104
iter 100 value 26.377457
final value 26.377457
stopped after 100 iterations
# weights: 27
initial value 316.049771
iter 10 value 34.303484
iter 20 value 5.182298
iter 30 value 0.375108
iter 40 value 0.268261
iter 50 value 0.250878
iter 60 value 0.206567
iter 70 value 0.191956
iter 80 value 0.170463
iter 90 value 0.155875
iter 100 value 0.135728
final value 0.135728
stopped after 100 iterations
# weights: 43
initial value 399.525517
iter 10 value 4.544415
iter 20 value 0.386647
iter 30 value 0.304616
iter 40 value 0.246634
iter 50 value 0.208378
iter 60 value 0.158324
iter 70 value 0.122642
iter 80 value 0.103977
iter 90 value 0.096830
iter 100 value 0.094210
final value 0.094210
stopped after 100 iterations
# weights: 11
initial value 312.693595
iter 10 value 105.372555
iter 20 value 90.239248
iter 30 value 60.495197
iter 40 value 46.115245
iter 50 value 45.114215
iter 60 value 44.973144
iter 70 value 44.959941
iter 80 value 44.918872
iter 90 value 44.902785
iter 100 value 44.898332
final value 44.898332

```

```

stopped after 100 iterations
# weights: 27
initial value 286.925387
iter 10 value 4.587271
iter 20 value 0.183101
iter 30 value 0.003072
iter 40 value 0.000378
final value 0.000093
converged
# weights: 43
initial value 315.273022
iter 10 value 2.765426
iter 20 value 0.025207
iter 30 value 0.000971
iter 40 value 0.000290
iter 50 value 0.000102
iter 50 value 0.000095
iter 50 value 0.000095
final value 0.000095
converged
# weights: 11
initial value 334.715783
iter 10 value 112.164946
iter 20 value 77.214549
final value 76.740347
converged
# weights: 27
initial value 298.430169
iter 10 value 49.601872
iter 20 value 23.444610
iter 30 value 21.870703
iter 40 value 21.782204
iter 50 value 21.771735
iter 60 value 21.769662
iter 70 value 21.769281
final value 21.769277
converged
# weights: 43
initial value 280.370036
iter 10 value 32.279006
iter 20 value 18.128566
iter 30 value 17.296149
iter 40 value 17.242415
iter 50 value 17.234590
iter 60 value 17.232087
final value 17.231928

```

```

converged
# weights: 11
initial value 289.910396
iter 10 value 106.919873
iter 20 value 106.836968
iter 30 value 106.815106
iter 40 value 106.159337
iter 50 value 70.412104
iter 60 value 49.674253
iter 70 value 41.076294
iter 80 value 33.982204
iter 90 value 28.406307
iter 100 value 27.548394
final value 27.548394
stopped after 100 iterations
# weights: 27
initial value 294.420616
iter 10 value 5.889083
iter 20 value 0.388339
iter 30 value 0.311851
iter 40 value 0.277683
iter 50 value 0.268228
iter 60 value 0.246174
iter 70 value 0.238504
iter 80 value 0.228303
iter 90 value 0.211653
iter 100 value 0.206981
final value 0.206981
stopped after 100 iterations
# weights: 43
initial value 274.361737
iter 10 value 2.629064
iter 20 value 0.266487
iter 30 value 0.186791
iter 40 value 0.182697
iter 50 value 0.179314
iter 60 value 0.169919
iter 70 value 0.165191
iter 80 value 0.157725
iter 90 value 0.154197
iter 100 value 0.151235
final value 0.151235
stopped after 100 iterations
# weights: 11
initial value 356.130868
iter 10 value 108.052063

```

```
iter 20 value 106.647970
final value 106.645162
converged
# weights: 27
initial value 303.268350
iter 10 value 5.506785
iter 20 value 0.067751
iter 30 value 0.000260
final value 0.000079
converged
# weights: 43
initial value 300.558309
iter 10 value 6.657646
iter 20 value 0.709013
iter 30 value 0.008746
iter 40 value 0.001142
iter 50 value 0.000452
final value 0.000063
converged
# weights: 11
initial value 350.451495
iter 10 value 121.177298
iter 20 value 106.026812
iter 30 value 77.060604
final value 76.774547
converged
# weights: 27
initial value 331.481025
iter 10 value 30.698868
iter 20 value 19.759682
iter 30 value 19.068463
iter 40 value 18.937106
iter 50 value 18.913386
iter 60 value 18.912153
final value 18.912137
converged
# weights: 43
initial value 407.962309
iter 10 value 24.595711
iter 20 value 18.178334
iter 30 value 17.568612
iter 40 value 17.465519
iter 50 value 17.464028
iter 60 value 17.463988
final value 17.463984
converged
```

```

# weights: 11
initial value 282.981286
iter 10 value 139.377003
iter 20 value 107.066840
iter 30 value 106.986407
iter 40 value 106.824449
iter 50 value 105.460750
iter 60 value 104.423648
iter 70 value 99.263182
iter 80 value 63.910086
iter 90 value 27.657767
iter 100 value 21.198167
final value 21.198167
stopped after 100 iterations
# weights: 27
initial value 426.453784
iter 10 value 1.561949
iter 20 value 0.499081
iter 30 value 0.476756
iter 40 value 0.334579
iter 50 value 0.308151
iter 60 value 0.214858
iter 70 value 0.195630
iter 80 value 0.182056
iter 90 value 0.171175
iter 100 value 0.157281
final value 0.157281
stopped after 100 iterations
# weights: 43
initial value 303.366957
iter 10 value 2.362463
iter 20 value 0.227959
iter 30 value 0.177360
iter 40 value 0.169695
iter 50 value 0.159877
iter 60 value 0.149808
iter 70 value 0.144519
iter 80 value 0.143235
iter 90 value 0.142283
iter 100 value 0.138880
final value 0.138880
stopped after 100 iterations
# weights: 11
initial value 293.722625
iter 10 value 112.601976
iter 20 value 66.068913

```

```

iter 30 value 31.068494
iter 40 value 26.877421
iter 50 value 25.953529
iter 60 value 25.821036
iter 70 value 25.447405
iter 80 value 25.402062
iter 90 value 25.326916
iter 100 value 25.286486
final value 25.286486
stopped after 100 iterations
# weights: 27
initial value 390.463329
iter 10 value 5.975063
iter 20 value 0.036108
iter 30 value 0.014735
iter 40 value 0.001786
iter 50 value 0.001395
final value 0.000065
converged
# weights: 43
initial value 344.713332
iter 10 value 4.349724
iter 20 value 0.131115
iter 30 value 0.001943
iter 40 value 0.000278
iter 50 value 0.000100
iter 50 value 0.000085
iter 50 value 0.000084
final value 0.000084
converged
# weights: 11
initial value 286.726036
iter 10 value 122.153075
iter 20 value 99.829741
iter 30 value 79.305288
iter 40 value 78.419235
iter 40 value 78.419235
iter 40 value 78.419235
final value 78.419235
converged
# weights: 27
initial value 380.844456
iter 10 value 27.694032
iter 20 value 21.310519
iter 30 value 21.164526
iter 40 value 21.131020

```

```

final  value 21.131011
converged
# weights: 43
initial  value 309.186527
iter   10 value 33.782019
iter   20 value 18.778218
iter   30 value 17.684150
iter   40 value 17.372413
iter   50 value 17.334111
iter   60 value 17.332620
final  value 17.332504
converged
# weights: 11
initial  value 288.409438
iter   10 value 155.696903
iter   20 value 134.956397
iter   30 value 58.831835
iter   40 value 46.092846
iter   50 value 44.785664
iter   60 value 44.747017
iter   70 value 44.590445
iter   80 value 44.571037
iter   90 value 44.541530
iter  100 value 44.533036
final  value 44.533036
stopped after 100 iterations
# weights: 27
initial  value 286.834331
iter   10 value 4.417813
iter   20 value 0.437678
iter   30 value 0.243529
iter   40 value 0.214139
iter   50 value 0.205154
iter   60 value 0.197450
iter   70 value 0.192842
iter   80 value 0.178885
iter   90 value 0.174468
iter  100 value 0.172055
final  value 0.172055
stopped after 100 iterations
# weights: 43
initial  value 275.800628
iter   10 value 1.840124
iter   20 value 0.346854
iter   30 value 0.290212
iter   40 value 0.278699

```

```

iter 50 value 0.253851
iter 60 value 0.229057
iter 70 value 0.218349
iter 80 value 0.204994
iter 90 value 0.197325
iter 100 value 0.188515
final value 0.188515
stopped after 100 iterations
# weights: 11
initial value 291.146481
iter 10 value 135.724680
iter 20 value 108.884734
iter 30 value 108.173783
iter 40 value 104.384400
iter 50 value 104.064319
iter 60 value 102.995216
iter 70 value 100.613599
iter 80 value 88.673687
iter 90 value 34.254236
iter 100 value 27.217452
final value 27.217452
stopped after 100 iterations
# weights: 27
initial value 255.184945
iter 10 value 7.155956
iter 20 value 0.216897
iter 30 value 0.001200
final value 0.000074
converged
# weights: 43
initial value 310.370811
iter 10 value 2.095731
iter 20 value 0.218495
iter 30 value 0.021090
iter 40 value 0.009023
iter 50 value 0.002423
iter 60 value 0.000932
iter 70 value 0.000666
final value 0.000083
converged

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-20') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead

of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```
# weights: 11
initial value 300.360670
iter 10 value 137.294601
iter 20 value 117.569501
iter 30 value 91.167577
iter 40 value 78.899067
final value 78.898095
converged
# weights: 27
initial value 362.269587
iter 10 value 44.836650
iter 20 value 20.996217
iter 30 value 19.553254
iter 40 value 19.161145
iter 50 value 18.927754
iter 60 value 18.903183
final value 18.903167
converged
# weights: 43
initial value 308.973713
iter 10 value 45.775409
iter 20 value 19.326641
iter 30 value 17.882882
iter 40 value 17.401981
iter 50 value 17.259938
iter 60 value 17.246039
iter 70 value 17.244809
iter 80 value 17.244565
final value 17.244565
converged
# weights: 11
initial value 315.468079
iter 10 value 109.842423
iter 20 value 107.954062
iter 30 value 100.092884
iter 40 value 88.310061
iter 50 value 40.278550
iter 60 value 28.035304
iter 70 value 27.480847
iter 80 value 27.057420
iter 90 value 26.973661
```

```

iter 100 value 26.973496
final  value 26.973496
stopped after 100 iterations
# weights: 27
initial  value 305.774096
iter  10 value 7.277303
iter  20 value 2.677930
iter  30 value 0.791391
iter  40 value 0.520087
iter  50 value 0.420250
iter  60 value 0.365273
iter  70 value 0.314615
iter  80 value 0.294904
iter  90 value 0.285448
iter 100 value 0.225955
final  value 0.225955
stopped after 100 iterations
# weights: 43
initial  value 272.014454
iter  10 value 4.031430
iter  20 value 0.412472
iter  30 value 0.271851
iter  40 value 0.240783
iter  50 value 0.231264
iter  60 value 0.211262
iter  70 value 0.198021
iter  80 value 0.179978
iter  90 value 0.176317
iter 100 value 0.171633
final  value 0.171633
stopped after 100 iterations
# weights: 11
initial  value 297.600530
iter  10 value 127.393563
iter  20 value 121.674852
iter  30 value 100.456883
iter  40 value 74.597005
iter  50 value 20.482687
iter  60 value 17.680973
iter  70 value 17.299742
iter  80 value 16.852957
iter  90 value 16.836028
iter 100 value 16.707271
final  value 16.707271
stopped after 100 iterations
# weights: 27

```

```
initial value 337.183395
iter 10 value 103.634871
iter 20 value 95.069997
iter 30 value 0.009495
iter 40 value 0.001212
iter 50 value 0.000154
iter 60 value 0.000106
iter 60 value 0.000087
iter 60 value 0.000087
final value 0.000087
converged
# weights: 43
initial value 315.090764
iter 10 value 1.192469
iter 20 value 0.020421
iter 30 value 0.005954
iter 40 value 0.001462
iter 50 value 0.001097
iter 60 value 0.000333
final value 0.000056
converged
# weights: 11
initial value 332.667984
iter 10 value 199.029294
iter 20 value 113.264504
iter 30 value 81.526765
iter 40 value 75.511180
final value 75.511044
converged
# weights: 27
initial value 312.734293
iter 10 value 36.833678
iter 20 value 18.681541
iter 30 value 17.974123
iter 40 value 17.866587
iter 50 value 17.859537
iter 60 value 17.859321
final value 17.859292
converged
# weights: 43
initial value 295.775961
iter 10 value 53.997122
iter 20 value 19.332851
iter 30 value 16.392008
iter 40 value 16.260696
iter 50 value 16.248620
```

```
iter 60 value 16.242763
iter 70 value 16.241944
iter 80 value 16.241912
final value 16.241903
converged
# weights: 11
initial value 286.237774
iter 10 value 104.711040
iter 20 value 69.536981
iter 30 value 25.459282
iter 40 value 18.377335
iter 50 value 18.135413
iter 60 value 17.679874
iter 70 value 17.671356
iter 80 value 17.657612
final value 17.654873
converged
# weights: 27
initial value 336.056403
iter 10 value 0.643952
iter 20 value 0.167912
iter 30 value 0.154220
iter 40 value 0.145738
iter 50 value 0.141586
iter 60 value 0.126655
iter 70 value 0.119883
iter 80 value 0.114913
iter 90 value 0.096962
iter 100 value 0.087844
final value 0.087844
stopped after 100 iterations
# weights: 43
initial value 378.295265
iter 10 value 1.019639
iter 20 value 0.122643
iter 30 value 0.116629
iter 40 value 0.112418
iter 50 value 0.096899
iter 60 value 0.087592
iter 70 value 0.082156
iter 80 value 0.071911
iter 90 value 0.068332
iter 100 value 0.066802
final value 0.066802
stopped after 100 iterations
# weights: 11
```

```
initial value 298.077859
iter 10 value 130.083566
iter 20 value 29.410743
iter 30 value 26.664753
iter 40 value 25.405071
iter 50 value 24.986476
iter 60 value 24.942728
iter 70 value 24.837913
iter 80 value 24.819005
iter 90 value 24.795737
iter 100 value 24.757594
final value 24.757594
stopped after 100 iterations
# weights: 27
initial value 387.050177
iter 10 value 4.513445
iter 20 value 0.071004
iter 30 value 0.003732
iter 40 value 0.000445
iter 50 value 0.000161
iter 50 value 0.000091
iter 50 value 0.000090
final value 0.000090
converged
# weights: 43
initial value 274.254509
iter 10 value 3.357789
iter 20 value 0.146507
iter 30 value 0.001294
iter 40 value 0.000432
final value 0.000080
converged
# weights: 11
initial value 379.474767
iter 10 value 196.750813
iter 20 value 149.593956
iter 30 value 127.010879
iter 40 value 81.622965
iter 50 value 77.050461
final value 77.050143
converged
# weights: 27
initial value 411.028637
iter 10 value 32.635422
iter 20 value 21.574384
iter 30 value 20.999819
```

```

iter 40 value 20.274790
iter 50 value 19.857624
iter 60 value 19.857027
final value 19.856983
converged
# weights: 43
initial value 420.412004
iter 10 value 48.399471
iter 20 value 21.201685
iter 30 value 20.053702
iter 40 value 19.913649
iter 50 value 19.904111
iter 60 value 19.898563
iter 70 value 19.898178
final value 19.898048
converged
# weights: 11
initial value 350.237977
iter 10 value 104.922133
iter 20 value 100.741376
iter 30 value 100.014558
iter 40 value 97.951121
iter 50 value 96.737526
iter 60 value 91.734988
iter 70 value 43.655726
iter 80 value 28.509507
iter 90 value 25.916760
iter 100 value 25.563088
final value 25.563088
stopped after 100 iterations
# weights: 27
initial value 281.446073
iter 10 value 3.659745
iter 20 value 0.374692
iter 30 value 0.340444
iter 40 value 0.326697
iter 50 value 0.302106
iter 60 value 0.275296
iter 70 value 0.264919
iter 80 value 0.243846
iter 90 value 0.236846
iter 100 value 0.213639
final value 0.213639
stopped after 100 iterations
# weights: 43
initial value 348.542969

```

```

iter 10 value 5.714495
iter 20 value 0.595875
iter 30 value 0.341199
iter 40 value 0.319946
iter 50 value 0.306116
iter 60 value 0.258491
iter 70 value 0.233757
iter 80 value 0.221282
iter 90 value 0.208241
iter 100 value 0.201530
final value 0.201530
stopped after 100 iterations
# weights: 11
initial value 278.207336
iter 10 value 80.436035
iter 20 value 31.802600
iter 30 value 27.851316
iter 40 value 26.615577
iter 50 value 26.431657
iter 60 value 26.400164
iter 70 value 26.297331
iter 80 value 26.292760
iter 90 value 26.263156
iter 100 value 26.213053
final value 26.213053
stopped after 100 iterations
# weights: 27
initial value 313.222635
iter 10 value 3.039674
iter 20 value 0.019703
iter 30 value 0.001901
iter 40 value 0.000311
iter 50 value 0.000134
final value 0.000054
converged
# weights: 43
initial value 325.561500
iter 10 value 2.885618
iter 20 value 0.016140
iter 30 value 0.002353
final value 0.000054
converged
# weights: 11
initial value 307.633845
iter 10 value 119.592045
iter 20 value 111.856711

```

```
iter 30 value 102.034574
final value 101.425219
converged
# weights: 27
initial value 306.737074
iter 10 value 28.805489
iter 20 value 19.443184
iter 30 value 18.213421
iter 40 value 18.197893
iter 50 value 18.196818
final value 18.196812
converged
# weights: 43
initial value 351.518148
iter 10 value 24.332257
iter 20 value 17.482397
iter 30 value 17.213382
iter 40 value 16.927173
iter 50 value 16.723300
iter 60 value 16.467298
iter 70 value 16.457497
final value 16.457455
converged
# weights: 11
initial value 318.605308
iter 10 value 64.268254
iter 20 value 29.163651
iter 30 value 27.620722
iter 40 value 27.061876
iter 50 value 26.882788
iter 60 value 26.865316
iter 70 value 26.829064
final value 26.829053
converged
# weights: 27
initial value 283.217030
iter 10 value 2.455619
iter 20 value 0.201424
iter 30 value 0.182276
iter 40 value 0.172550
iter 50 value 0.169548
iter 60 value 0.159413
iter 70 value 0.155737
iter 80 value 0.144546
iter 90 value 0.133680
iter 100 value 0.125687
```

```

final value 0.125687
stopped after 100 iterations
# weights: 43
initial value 307.013556
iter 10 value 2.837261
iter 20 value 1.172323
iter 30 value 1.022601
iter 40 value 0.800452
iter 50 value 0.290725
iter 60 value 0.239873
iter 70 value 0.189466
iter 80 value 0.166874
iter 90 value 0.145784
iter 100 value 0.136395
final value 0.136395
stopped after 100 iterations
# weights: 11
initial value 303.371226
iter 10 value 109.080687
iter 20 value 104.625356
iter 30 value 103.163407
iter 40 value 103.134225
iter 50 value 103.125682
iter 60 value 103.122999
iter 70 value 103.116927
iter 80 value 103.115214
iter 90 value 103.089671
iter 100 value 102.776563
final value 102.776563
stopped after 100 iterations
# weights: 27
initial value 314.178096
iter 10 value 3.886490
iter 20 value 0.104562
iter 30 value 0.004322
iter 40 value 0.000370
iter 50 value 0.000169
final value 0.000067
converged
# weights: 43
initial value 312.148693
iter 10 value 1.811841
iter 20 value 0.077026
iter 30 value 0.009446
iter 40 value 0.000772
final value 0.000054

```

```

converged
# weights: 11
initial value 281.477193
iter 10 value 123.253395
iter 20 value 111.434650
iter 30 value 86.956226
iter 40 value 78.744576
final value 78.744520
converged
# weights: 27
initial value 337.109286
iter 10 value 25.171887
iter 20 value 19.407660
iter 30 value 19.078009
iter 40 value 19.063592
final value 19.063521
converged
# weights: 43
initial value 312.866272
iter 10 value 31.146596
iter 20 value 18.561387
iter 30 value 17.485400
iter 40 value 17.403375
iter 50 value 17.392927
iter 60 value 17.392722
iter 70 value 17.392605
iter 70 value 17.392605
iter 70 value 17.392605
final value 17.392605
converged
# weights: 11
initial value 299.607728
iter 10 value 87.677714
iter 20 value 35.021408
iter 30 value 28.706535
iter 40 value 27.335489
iter 50 value 26.985537
iter 60 value 26.957597
iter 70 value 26.911874
final value 26.911853
converged
# weights: 27
initial value 377.321427
iter 10 value 17.678669
iter 20 value 0.886517
iter 30 value 0.325857

```

```

iter 40 value 0.278144
iter 50 value 0.256844
iter 60 value 0.228021
iter 70 value 0.219488
iter 80 value 0.203413
iter 90 value 0.190720
iter 100 value 0.176921
final value 0.176921
stopped after 100 iterations
# weights: 43
initial value 341.597420
iter 10 value 6.689296
iter 20 value 0.244372
iter 30 value 0.214287
iter 40 value 0.198241
iter 50 value 0.189551
iter 60 value 0.178604
iter 70 value 0.172961
iter 80 value 0.168394
iter 90 value 0.165085
iter 100 value 0.161976
final value 0.161976
stopped after 100 iterations

```

Warning: UNRELIABLE VALUE: One of the foreach() iterations ('doFuture-21') unexpectedly generated random numbers without declaring so. There is a risk that those random numbers are not statistically sound and the overall results might be invalid. To fix this, use '%dorng%' from the 'doRNG' package instead of '%dopar%'. This ensures that proper, parallel-safe random numbers are produced. To disable this check, set option 'doFuture.rng.onMisuse' to "ignore".

```

# weights: 27
initial value 346.545098
iter 10 value 47.055146
iter 20 value 23.916453
iter 30 value 22.535497
iter 40 value 22.502251
iter 50 value 22.491487
iter 60 value 22.487199
iter 70 value 22.486987
final value 22.486985
converged

```

## 28.4 Stopping parallel computing

```
stopCluster(cl)
registerDoSEQ() # Optional: reverts to sequential operation
# plan(sequential) # for mirai
```

## 28.5 Model comparison

The 100 models trained for the selected optimal tuning parameter combination are compared below.

```
resamps <- resamples(list(knn = knnfit,
                           lda = ldafit,
                           rf=rffit,
                           xgb=xgbfit,
                           svm=svmfit,
                           bayes=bayesfit,
                           nn=nnfit))

summary(resamps) |>
  pluck("statistics") |>
  pluck("Accuracy") |>
  as_tibble(rownames = "Model") |>
  mutate(across(-Model,
                .fns = ~round(.x, 3))) |>
  flextable() |>
  set_table_properties(width=1, layout="autofit")
```

Model	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
knn	0.955	0.970	0.985	0.987	1.000	1	0
lda	0.939	0.985	0.985	0.986	1.000	1	0
rf	0.910	0.970	0.970	0.974	0.985	1	0
xgb	0.924	0.970	0.985	0.977	0.985	1	0
svm	0.955	0.970	0.985	0.985	1.000	1	0
bayes	0.925	0.955	0.970	0.970	0.985	1	0
nn	0.940	0.985	1.000	0.990	1.000	1	0

```

resamps$values |>
  head() |>
  select(1:7) |>
  mutate(across(-Resample, round, 3)) |>
  flextable() |>
  set_table_properties(width=1, layout="autofit")

```

Warning: There was 1 warning in `mutate()`.  
 i In argument: `across(-Resample, round, 3)`.  
 Caused by warning:  
 ! The `...` argument of `across()` is deprecated as of dplyr 1.1.0.  
 Supply arguments directly to `~.fns` through an anonymous function instead.

```

# Previously
across(a:b, mean, na.rm = TRUE)

# Now
across(a:b, ~mean(x, na.rm = TRUE))

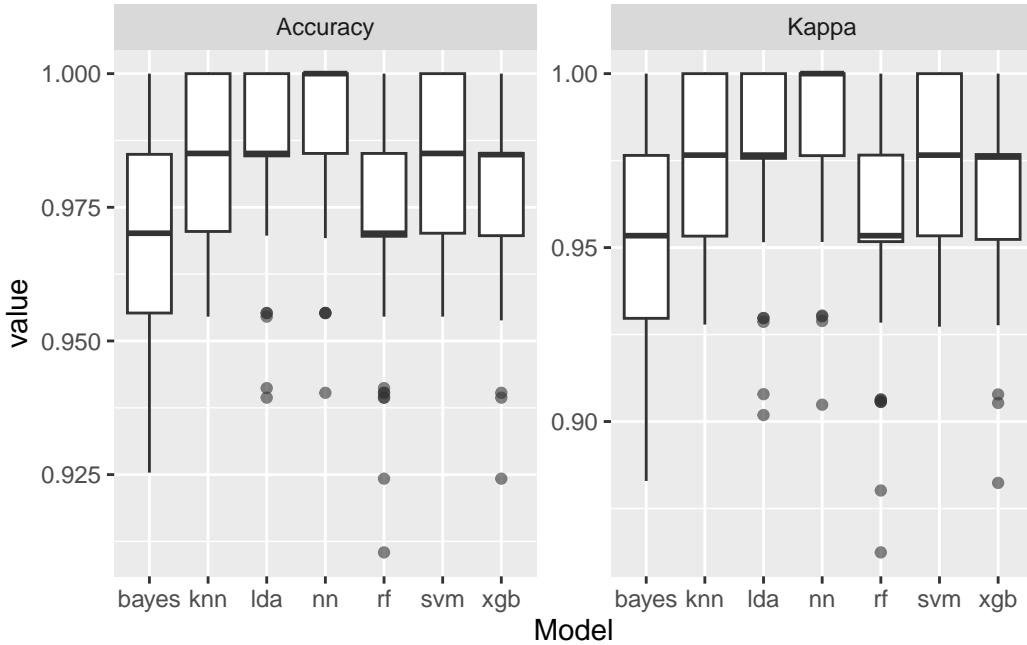
```

Resample	knn~Accuracy	knn~Kappa	lda~Accuracy	lda~Kappa	rf~Accuracy	rf~Kappa
Fold1.Rep01	0.970	0.953	1.000	1.000	0.985	0.976
Fold1.Rep02	1.000	1.000	0.985	0.977	0.970	0.953
Fold1.Rep03	0.955	0.929	0.985	0.977	0.910	0.862
Fold1.Rep04	0.985	0.977	0.985	0.977	0.970	0.952
Fold1.Rep05	0.985	0.976	0.985	0.977	0.955	0.930
Fold1.Rep06	1.000	1.000	0.985	0.976	0.985	0.977

```

resamps$values |>
  pivot_longer(contains('~'),
               names_to = c('Model', 'Measure'),
               names_sep = '~') |>
  ggplot(aes(Model, value)) +
  geom_boxplot(outlier.alpha = .6) +
  facet_wrap(facets = vars(Measure), scales = 'free')

```



If really deemed necessary, the differences between models can be tested statistically.

```
diffs <- diff(resamps, adjustment = "fdr")
diffs$statistics <-
  diffs$statistics |>
  map_depth(.depth = 2, \x) {
    if (is.list(x) && "p.value" %in% names(x)) {
      x$p.value <- formatP(x$p.value, ndigits = 2, textout = F)
    }
    return(x)
  })
diffs$statistics <-
  diffs$statistics |>
  map_depth(.depth = 2, function(x) {
    if (is.list(x) && "estimate" %in% names(x)) {
      x$estimate <- round(x$estimate, 4)
    }
    return(x)
  })
summary(diffs)
```

```
Call:
summary.diff.resamples(object = diffs)

p-value adjustment: fdr
```

Upper diagonal: estimates of the difference  
Lower diagonal: p-value for H0: difference = 0

#### Accuracy

	knn	lda	rf	xgb	svm	bayes	nn
knn		0.0007	0.0132	0.0098	0.0021	0.0165	-0.0036
lda	0.7200		0.0125	0.0090	0.0013	0.0158	-0.0044
rf	0.0150	0.0150		-0.0034	-0.0111	0.0033	-0.0168
xgb	0.0150	0.0150	0.1976		-0.0077	0.0067	-0.0134
svm	0.2874	0.5565	0.0150	0.0150		0.0144	-0.0057
bayes	0.0150	0.0150	0.2800	0.0150	0.0150		-0.0201
nn	0.0525	0.0280	0.0150	0.0150	0.0150	0.0150	

#### Kappa

	knn	lda	rf	xgb	svm	bayes	nn
knn		0.0011	0.0205	0.0151	0.0032	0.0257	-0.0058
lda	0.7400		0.0194	0.0140	0.0021	0.0246	-0.0069
rf	0.0150	0.0150		-0.0054	-0.0173	0.0052	-0.0263
xgb	0.0150	0.0150	0.1976		-0.0119	0.0106	-0.0210
svm	0.2984	0.5460	0.0150	0.0150		0.0225	-0.0090
bayes	0.0150	0.0150	0.2800	0.0150	0.0150		-0.0315
nn	0.0525	0.0280	0.0150	0.0150	0.0150	0.0150	

## 28.6 Using the best model for prediction

While it may be advisable to rebuild the model(s) of choice with optimal tuning parameters, the results from train() can be used directly. Here we use the XGBoost model as an example.

```
xgbfit
```

```
eXtreme Gradient Boosting
```

```
333 samples
 4 predictor
 3 classes: 'Adelie', 'Chinstrap', 'Gentoo'
```

```
No pre-processing
Resampling: Cross-Validated (5 fold, repeated 20 times)
Summary of sample sizes: 266, 266, 266, 268, 266, 266, ...
Resampling results across tuning parameters:
```

max_depth	gamma	nrounds	Accuracy	Kappa
2	0.005	25	0.9748876	0.9606440

2	0.005	50	0.9756386	0.9618385
2	0.005	75	0.9760864	0.9625498
2	0.010	25	0.9747384	0.9604069
2	0.010	50	0.9751885	0.9611288
2	0.010	75	0.9754916	0.9616060
4	0.005	25	0.9768460	0.9637414
4	0.005	50	0.9766945	0.9635104
4	0.005	75	0.9766945	0.9635104
4	0.010	25	0.9769975	0.9639816
4	0.010	50	0.9769953	0.9639793
4	0.010	75	0.9769953	0.9639793
6	0.005	25	0.9756451	0.9618557
6	0.005	50	0.9760974	0.9625630
6	0.005	75	0.9760974	0.9625630
6	0.010	25	0.9760930	0.9625543
6	0.010	50	0.9760974	0.9625621
6	0.010	75	0.9760974	0.9625630
8	0.005	25	0.9756451	0.9618557
8	0.005	50	0.9760974	0.9625630
8	0.005	75	0.9760974	0.9625630
8	0.010	25	0.9760930	0.9625543
8	0.010	50	0.9760974	0.9625621
8	0.010	75	0.9760974	0.9625630
10	0.005	25	0.9756451	0.9618557
10	0.005	50	0.9760974	0.9625630
10	0.005	75	0.9760974	0.9625630
10	0.010	25	0.9760930	0.9625543
10	0.010	50	0.9760974	0.9625621
10	0.010	75	0.9760974	0.9625630

Tuning parameter 'eta' was held constant at a value of 1

Tuning

Tuning parameter 'min\_child\_weight' was held constant at a value of 1

Tuning parameter 'subsample' 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 nrounds = 25, max\_depth = 4, eta = 1, gamma = 0.01, colsample\_bytree = 1, min\_child\_weight = 1 and subsample = 1.

```
xgbfit[["bestTune"]]
```

nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
10	25	4	1	0.01	1	1

```
xgbpred <- predict(xgbfit, newdata = rawdata)
confusionMatrix(xgbpred, rawdata$species)
```

#### Confusion Matrix and Statistics

	Reference		
Prediction	Adelie	Chinstrap	Gentoo
Adelie	146	0	0
Chinstrap	0	68	0
Gentoo	0	0	119

#### Overall Statistics

Accuracy : 1  
95% CI : (0.989, 1)  
No Information Rate : 0.4384  
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 1

McNemar's Test P-Value : NA

#### Statistics by Class:

	Class: Adelie	Class: Chinstrap	Class: Gentoo
Sensitivity	1.0000	1.0000	1.0000
Specificity	1.0000	1.0000	1.0000
Pos Pred Value	1.0000	1.0000	1.0000
Neg Pred Value	1.0000	1.0000	1.0000
Prevalence	0.4384	0.2042	0.3574
Detection Rate	0.4384	0.2042	0.3574
Detection Prevalence	0.4384	0.2042	0.3574
Balanced Accuracy	1.0000	1.0000	1.0000

```
# Importance
importance_xgb <-
  xgboost::xgb.importance(model=xgbfit[["finalModel"]]) |>
  arrange(Gain) |>
  mutate(Feature=fct_inorder(Feature))
importance_xgb
```

Feature	Gain	Cover	Frequency
<fctr>	<num>	<num>	<num>

```
1:      body_mass_g 0.02968646 0.09740191 0.22307692
2:      bill_depth_mm 0.10885540 0.32390674 0.31538462
3:      bill_length_mm 0.41020662 0.42590446 0.38461538
4: flipper_length_mm 0.45125152 0.15278688 0.07692308
```

```
importance_xgb |>
  ggplot(aes(Feature, Gain)) +
  geom_col(aes(fill=Gain)) +
  coord_flip() +
  guides(fill="none")
```

