

RStatsbook

Andreas Busjahn

2025-07-21

Table of contents

Preface	7
1 Introduction	8
1.1 Installation of R	8
1.2 Installation of RStudio Desktop (UI)	8
1.3 Installation of Quarto (Reportgenerator)	8
1.4 Installation of git:	8
1.5 Setup RStudio	9
1.6 Projects on your computer	10
1.6.1 Option 1: Create manually	10
1.6.2 Option 2: Create from a github repository	11
2 Some useful resources	12
2.1 Books	12
2.2 Ressources	12
3 Syntax rules / basic things to know about R	13
3.1 <i>Script preparation / basic setup</i>	13
3.2 <i>Numeric operations</i>	14
3.3 <i>Variables</i>	15
3.3.1 <i>Variable names</i>	15
3.3.2 <i>Basic classes of data</i>	16
3.3.3 <i>Exercise: variables / Assignments</i>	18
3.3.4 <i>Exercise: Factors</i>	20
3.3.5 <i>Indexing variables</i>	24
3.3.6 <i>Usage of variables</i>	25
3.4 <i>Functions</i>	26
3.4.1 <i>Function usage</i>	26
3.4.2 <i>Functions combined</i>	28
3.4.3 <i>Writing functions</i>	30
3.5 <i>More complex data types, created by functions</i>	33
3.5.1 <i>Matrix</i>	33
3.5.2 <i>Data frame</i>	35
3.5.3 <i>Tibble</i>	38
3.5.4 <i>Exercise: tibble</i>	49
3.5.5 <i>List</i>	50
3.6 <i>Control structures</i>	53
3.6.1 <i>Loops</i>	53

3.6.2	<i>Conditions</i>	56
3.6.3	<i>Exercise: for and while loops, and if/else/case_xxx</i>	60
4	Regular expressions	62
4.1	Intro	62
4.1.1	Some basic examples:	62
4.1.2	An example for their use in renaming variables:	64
4.2	Exercise	65
5	Importing data	66
5.1	Import from text files (.txt, .csv)	66
5.2	Import from Excel	67
5.2.1	Tidy Excel files	67
5.2.2	Excel file with units row	67
5.3	ODS files	69
5.3.1	Import from SPSS/SAS	69
6	Changing structure wide <-> long	72
6.1	Example 1: single repeated measure	73
6.2	Example 2: several repeated measures	73
6.3	Example 3: long to wide	75
6.4	More examples	76
6.4.1	Step 1: Create example data:	76
6.4.2	Step 2: Transform that data to a long form:	77
6.4.3	Step 3 Transform long to wide	78
6.5	Exercise: Reshaping Data	79
6.5.1	Part 1: From Long to Wide (<code>pivot_wider()</code>)	79
7	Visualize data with ggplot	81
7.1	Example data	81
7.2	Basic structure of a ggplot call	82
7.3	fill vs. color	86
7.4	Color systems	88
7.4.1	External color definitions from ggsci	91
7.5	Exporting ggplots	94
7.6	Other geoms	94
7.7	Exercise 1	104
7.8	Combining and finetuning aesthetics	105
7.9	Positioning elements	112
7.10	Order of layers	117
7.11	Local aesthetics for layers	121
7.12	Faceting (splitting) plots	123
7.12.1	<code>facet_grid</code>	123
7.12.2	<code>facet_wrap</code>	126
7.12.3	Controlling scales in facets (default: <code>scales="fixed"</code>)	129
7.13	Showing summaries	131

7.14	Indicating significances	137
7.15	Theme definitions / changes	142
7.16	Combining figures with patchwork	148
7.17	ggplots in loops	150
8	Grouping of variables by type / distribution / use	154
8.1	Test for Normal distribution	154
8.1.1	Testing a single variable	154
8.1.2	Testing several variables	157
8.2	Exercise: Distribution of penguin measures	162
8.3	Picking column names and positions	162
9	Descriptive statistics	164
9.1	Typical descriptives	164
9.2	Reading in data	164
9.3	Graphical exploration should start before descriptive statistics	164
9.4	Gaussian variables	165
9.5	A little theory	165
9.5.1	Simple function calls	166
9.5.2	Combined reporting	167
9.6	Ordinal variables	168
9.7	Categorical variables	170
9.8	Summarize data	172
10	Exercises	177
11	Summarize / across	178
12	Simple test statistics	186
12.1	Tests require hypotheses	187
12.1.1	Null hypothesis ?	188
12.2	Quantitative measures with Gaussian distribution	188
12.3	Ordinal data	196
12.4	Categorial data	199
13	Covariance / Correlation	203
13.1	Covariance	203
13.2	Correlation	203
13.3	Vizualizations	205
14	Intro to linear models	208
14.1	Setup	208
14.2	Import / Preparation	208
14.3	Graphical exploration	209
14.4	Linear Models	212
14.4.1	Linear regression	212
14.4.2	ANOVA	217

14.4.3 LM with continuous AND categorical IV	222
14.4.4 Model exploration with package performance	228
15 Interaction in linear models	231
15.1 No age effect, no treatment effect, interaction treatment*agegroup	231
15.2 Age effect, no treatment effect, interaction treatment*agegroup	235
15.3 Age effect, treatment effect, interaction treatment*agegroup	240
15.4 How to specify interaction in multivariable models	247
16 Logistic regression	251
16.1 Odds vs. probability	251
16.2 Data preparation	252
16.3 Build model	255
16.4 Create structure for ggplot	257
16.5 create forest plot	257
16.6 Create predictions	259
16.7 Regression tree as alternative to glm	263
16.8 Jackknife	268
17 Machine Learning with R: Basic concepts	271
17.1 structured vs. unstructured data	271
17.2 supervised vs. unsupervised methods	271
17.3 Resampling methods	271
17.3.1 Permutation tests	271
17.3.2 Bootstrapping	272
17.3.3 Jackknife	273
17.3.4 k-fold CV	274
18 Markov Chain Monte Carlo: Metropolis algorithm simulation	275
18.1 Rule definition	276
18.2 Data structures for simulation	277
18.3 Simulation	277
18.4 Results	277
19 k nearest neighbors knn	291
19.1 Data preparation	292
19.2 Exploratory plots	292
19.3 Scaling	296
19.3.1 caret function preProcess	296
19.3.2 preprocessCore function normalize.quantiles	297
19.3.3 Visual comparison of the scaled data	297
19.4 Modelling	299
19.4.1 Definition of predictor variables (IV)	299
19.4.2 Data splitting	299
19.4.3 Model fitting	300
19.4.4 Model evaluation	302

19.4.5 Alternative approach to modelling	303
19.4.6 Evaluation of the alternative approach	303
19.4.7 Adding predictor variables	305
20 Regression and classification trees / Random forests	309
20.1 Regression trees	309
20.1.1 Graphical exploration	309
20.1.2 Modelling	310
20.1.3 Model evaluation	314
20.2 RT for continuous outcomes	316
20.3 Random forest	319
20.3.1 Modelling	319
20.3.2 Model evaluation	320
21 Boosted regression trees	323
22 Model tuning with caret	361
22.1 Create training/test data	362
22.2 Define global modelling options	363
22.3 Define method-specific options and tune models	363
22.4 Combine models for comparison	682
22.5 Access to individual models	686
23 Principal Components Analysis	690
23.1 Exploration of correlations between predictor variables	691
23.2 Two variable example	694
23.3 PCA bioconductor style	710
24 Linear discriminant analysis LDA	714
25 Cluster analysis	722
25.1 kmeans	722
25.2 HClust	732
25.3 Unified from easystats	740

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)

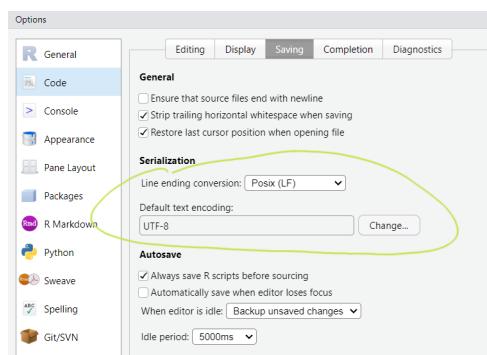
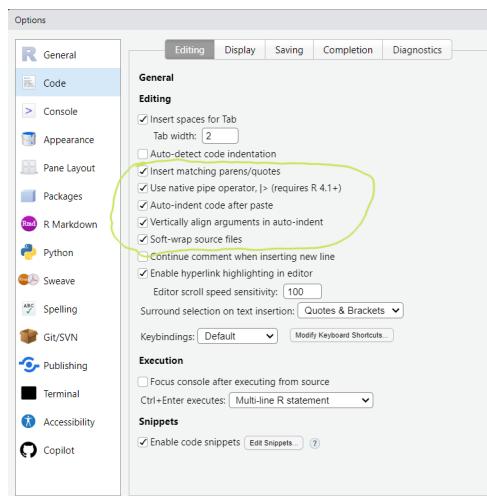
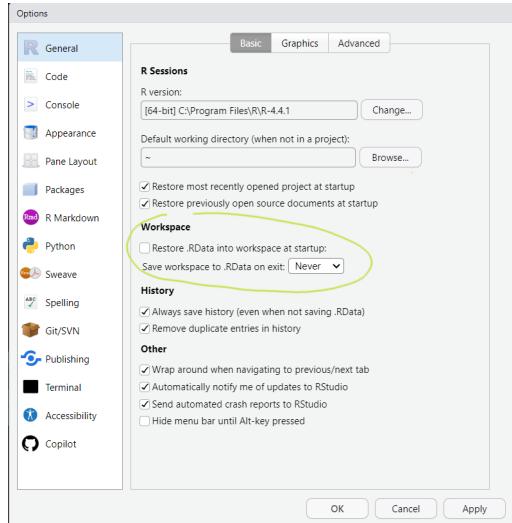
[Quarto - Get Started](#)

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**

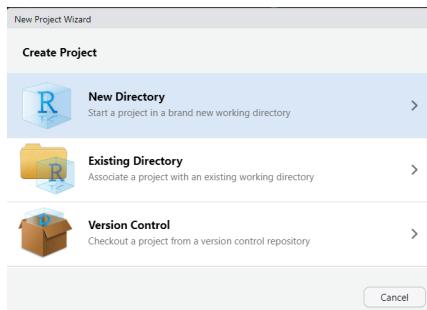
1.6.1 Option 1: Create manually

- 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!!

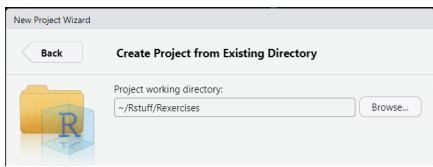
You should have a folder structure something like this:

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

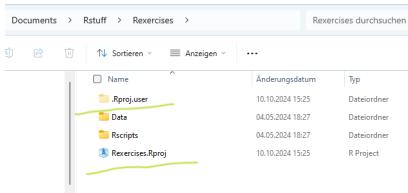
Now you are ready to create a project in RStudio:



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 and CamelCaseStyle.

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] "2025-07-21"
```

```
class(date_var)
```

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

```
typeof(date_var)
```

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

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

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

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")
```

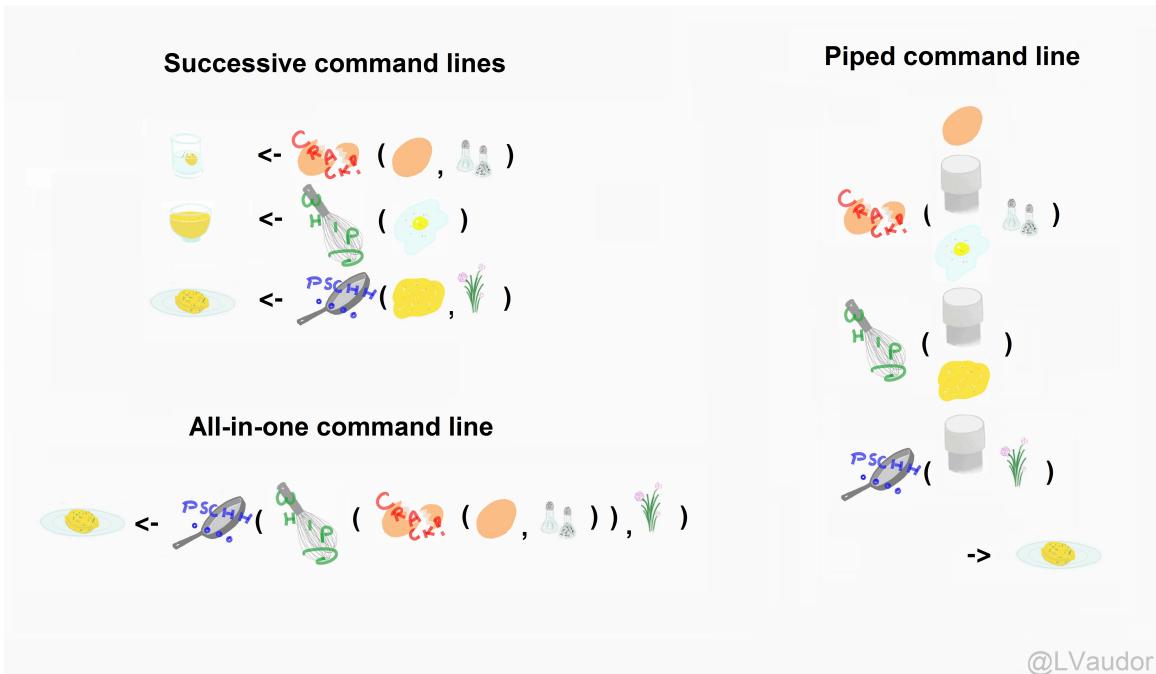
```
some text: 1  
some text: 2  
some text: 3
```

```
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 tabulate 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 |>



@LVaudor

Figure 3.1: *Piping functions*

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

[1] 39

```
# 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] 66

```
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
```

3.5.2 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).

3.5.2.1 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

3.5.2.2 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)
```

3.5.3 *Tibble*

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

3.5.3.1 *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)

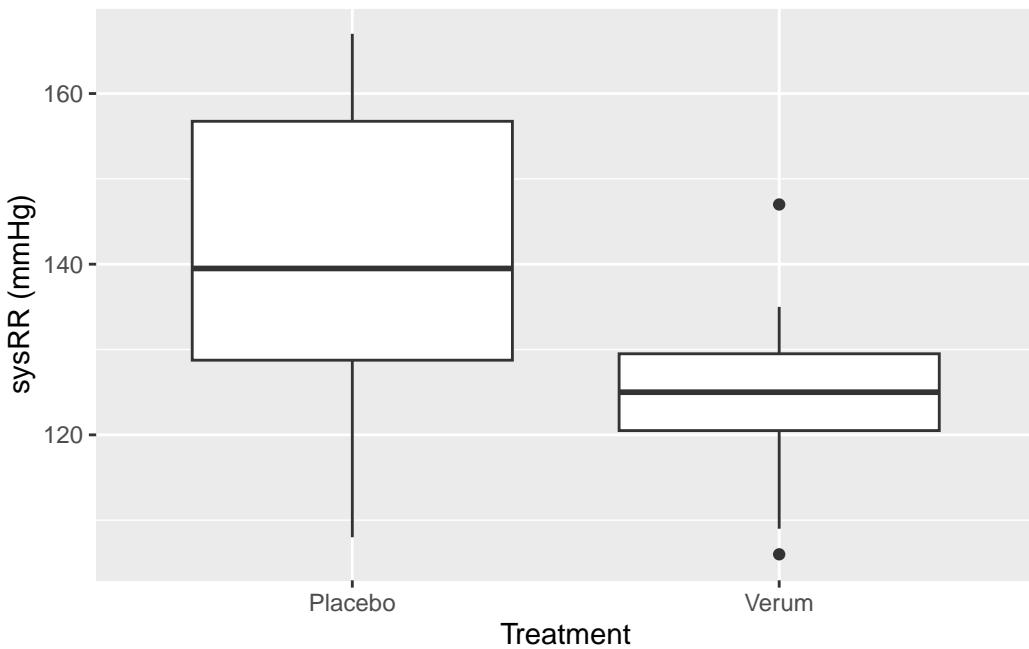
[1] "PatID"          "Sex"           "Ethnicity"       "Given name"     "Family name"
[6] "Treatment"      "sysRR (mmHg)"    "diaRR (mmHg)"   "HR"

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()

```



3.5.3.2 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 Al~ 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
```

3.5.4 *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



3.5.5 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.

3.5.5.1 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"

$menge
[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

```

3.5.5.2 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
```

3.6 Control structures

3.6.1 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.

3.6.1.1 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(col_i)
}

```

```

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

```

```

[1] "Treatment"
[1] "sysRR (mmHg)"
[1] "diaRR (mmHg)"
[1] "HR"

## 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"

# edge-case of 0 elements -> 0 runs
for (col_i in seq_len(0)) {

```

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

3.6.1.2 *while-loops*

If not number of repetitions is known, 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
```

3.6.2 *Conditions*

3.6.2.1 *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("still growing\n")  
}
```

```
still growing
```

3.6.2.2 *ifelse*

We can get text conditionally:

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

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

```
p <- .012  
paste0(  
  "That is ",
```

```

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

```

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

```

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

```

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

3.6.2.3 *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"
```

3.6.3 Exercise: for and while loops, and if/ifelse/case_xxx

3.6.3.1 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). * **Identify Differentially Expressed Genes:** * Using a **for** loop, iterate through each gene's expression level. * Inside the loop, use an **if/else 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.

3.6.3.2 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: After the loop, print a message indicating how many generations it took to reach the carrying capacity.

3.6.3.3 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.
- `[]` (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 = "p.+a\\?\\$")
```

[1] TRUE

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

[1] TRUE

```
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"      "Cupsize [cm]"    "Cupheight [cm]"
```

4.2 Exercise

```

testset1 <- c(
  "Meier", "Mayer", "Maier", "Meyer", "Mayr",
  "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 |> slice(2) |> pull(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)
```

```
'data.frame': 360 obs. of 5 variables:  
 $ bead_nr : num 1 2 3 4 5 6 7 8 9 10 ...  
 $ ZahlZellen: num NA ...  
 $ Wachstum : num 135 111 101 115 135 ...  
 $ Passage : num 1 2 3 4 5 6 1 2 3 4 ...  
 $ Bedingung : Factor w/ 3 levels "Kontrolle","AngII",...: 1 1 1 1 1 1 1 1 1 1 ...  
 - attr(*, "variable.labels")= Named chr(0)  
 ..- attr(*, "names")= chr(0)  
 - attr(*, "codepage")= int 1252
```

```
as_tibble(import1)
```

```
# A tibble: 360 x 5
  bead_nr ZahlZellen Wachstum Passage Bedingung
     <dbl>      <dbl>    <dbl>    <dbl>   <fct>
1       1          NA     135.      1 Kontrolle
2       2          NA     111.      2 Kontrolle
3       3          NA     101.      3 Kontrolle
4       4          NA     115.      4 Kontrolle
5       5          NA     135.      5 Kontrolle
6       6          NA     114.      6 Kontrolle
7       7          NA     134.      1 Kontrolle
8       8          NA     119.      2 Kontrolle
9       9          NA     127.      3 Kontrolle
10      10         NA     119.      4 Kontrolle
# i 350 more rows
```

```
import2 <- read_sav(file = here("data/Zellbeads.sav"))
str(import2$Bedingung)
```

```
attr(import2$Bedingung, "labels")
```

Kontrolle AngII Whatever
 1 2 3

```
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         82.9       88.9
#> 2 P 2    B         90.5      106.
#> 3 P 3    C         77.4      82.9
```

```
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  82.9
#> 2 P 1    A    Var2Time2  88.9
#> 3 P 2    B    Var2Time1  90.5
#> 4 P 2    B    Var2Time2 106.
#> 5 P 3    C    Var2Time1  77.4
#> 6 P 3    C    Var2Time2  82.9
```

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 column names. 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)
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 datasets, but with different column names!

7.2 Basic structure of a ggplot call

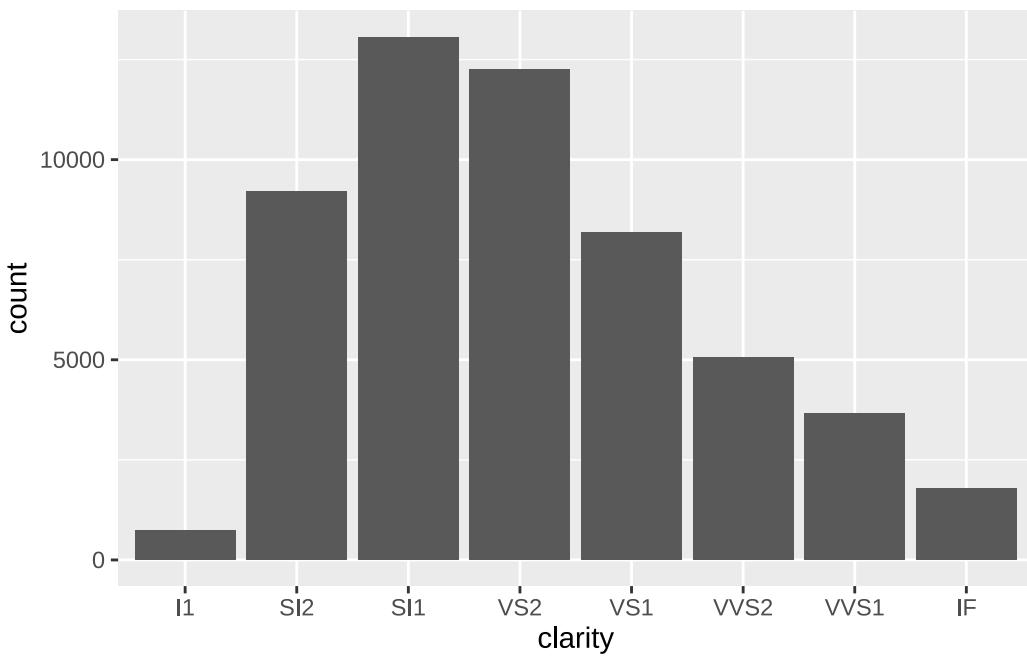
To create a figure, we at least use 2 function calls:

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.

Minimal example:

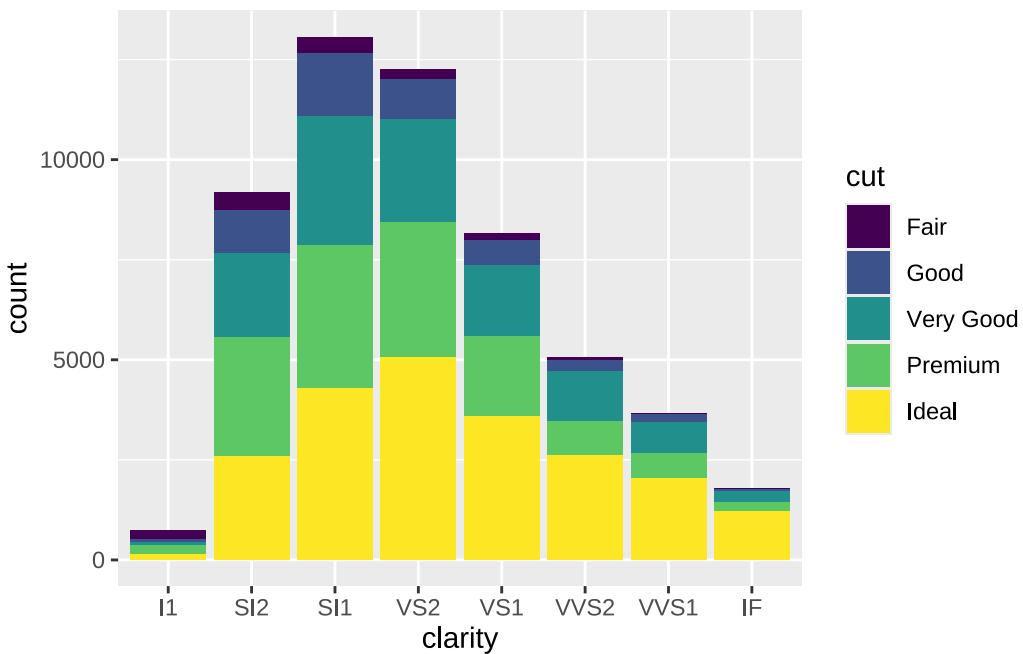
```
ggplot(data=diamonds, mapping = aes(x=clarity))+  
  geom_bar()
```



`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()
```



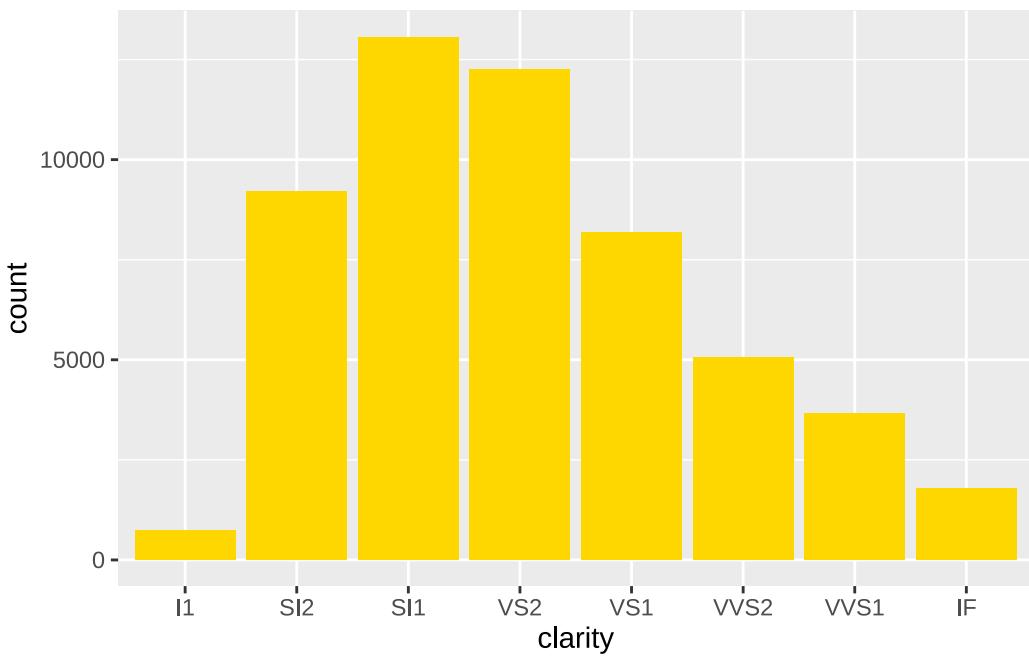
This graph represents this count table:

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

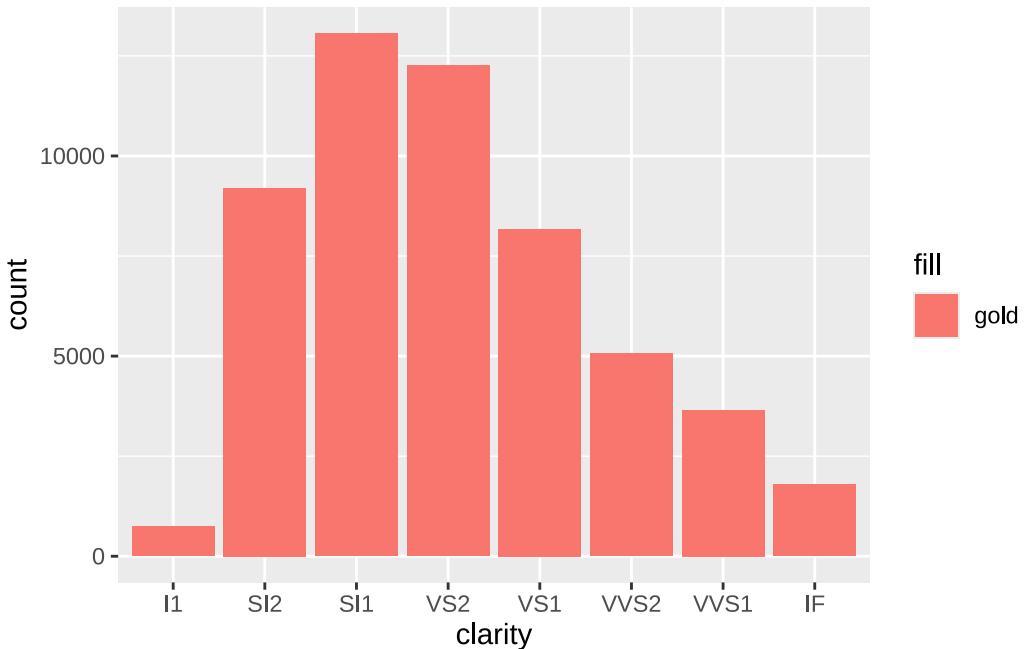
```
# A tibble: 5 x 9
# Groups:   cut [5]
  cut      I1    SI2    SI1    VS2    VS1    VVS2   VVS1    IF
  <ord>  <int> <int> <int> <int> <int> <int> <int>
1 Fair     210    466    408    261    170     69     17     9
2 Good     96    1081   1560    978    648    286    186    71
3 Very Good  84    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:

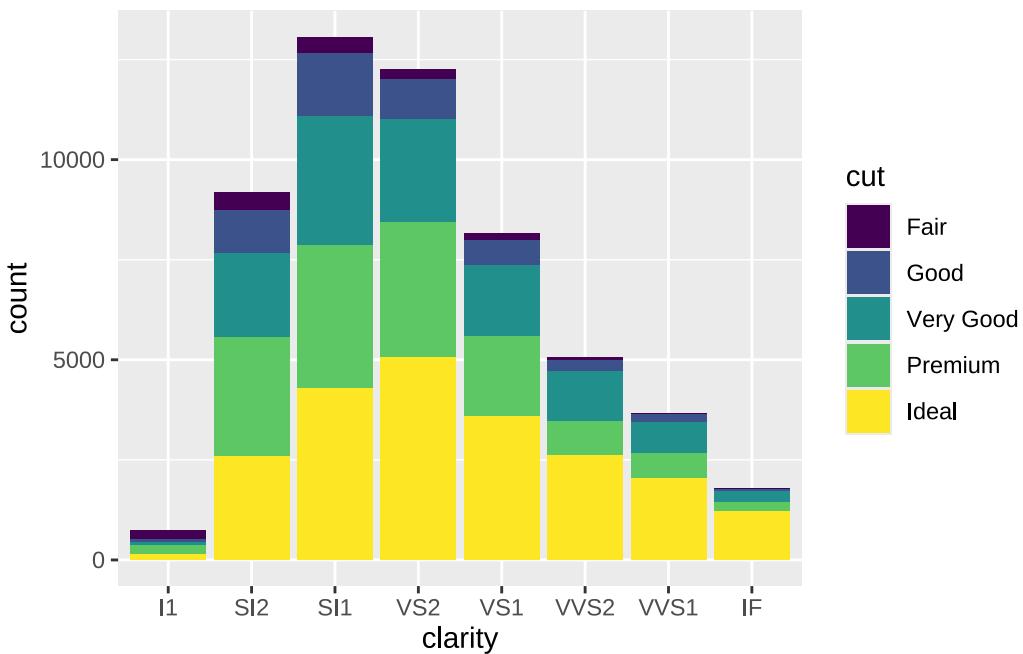
```
#aesthetics outside aes
ggplot(data=diamonds,aes(x=clarity))+  
  geom_bar(fill="gold")
```



```
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
```



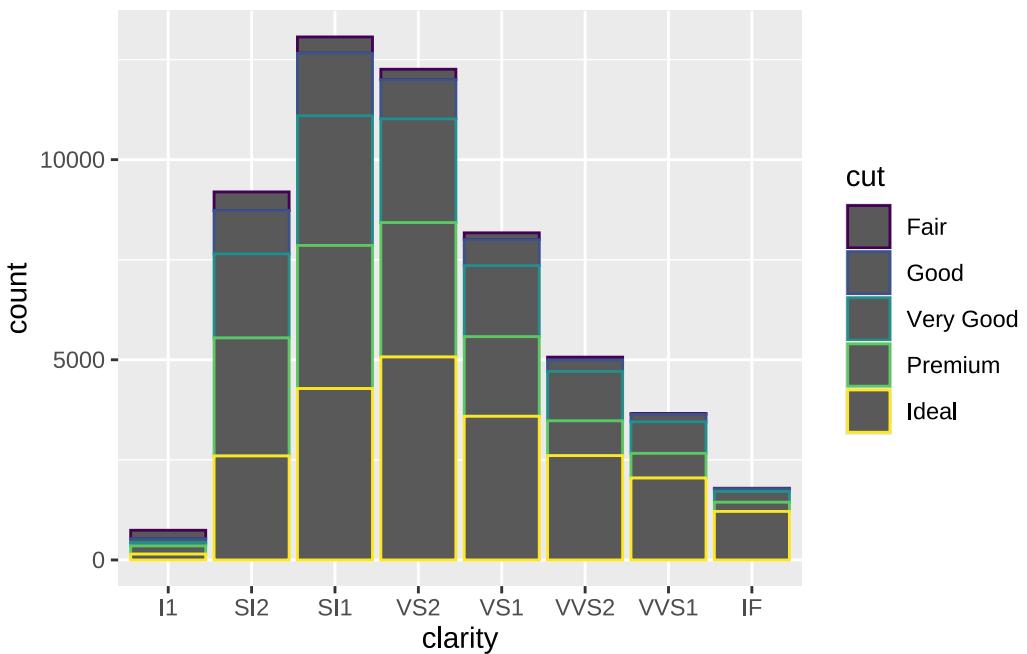
7.3 fill vs. color

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

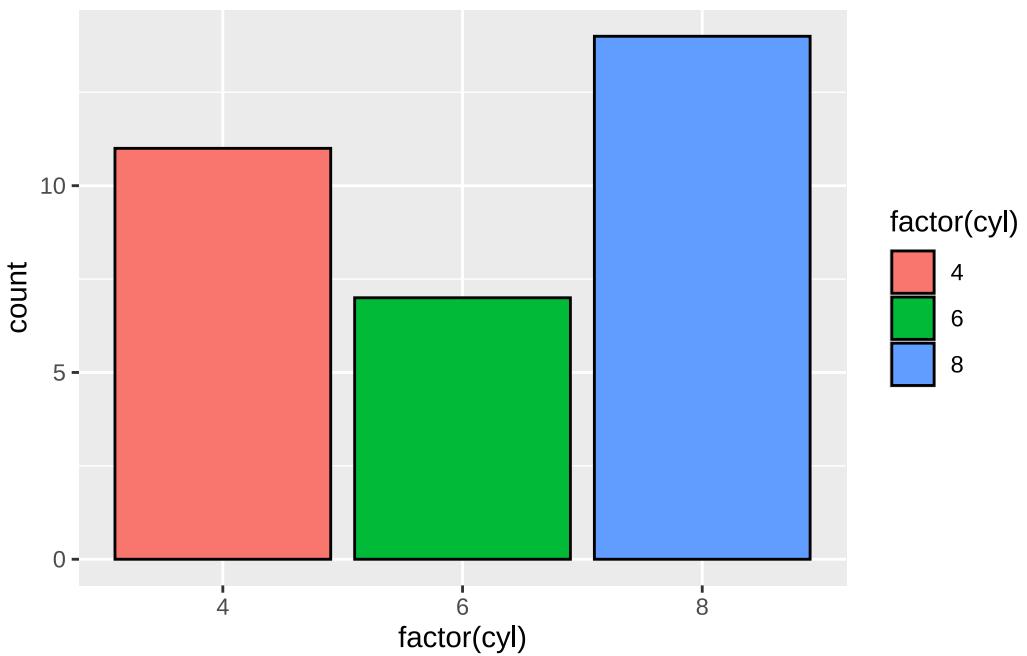
- inner color, defined by `fill`
- outer frame color, defined by `color`

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.

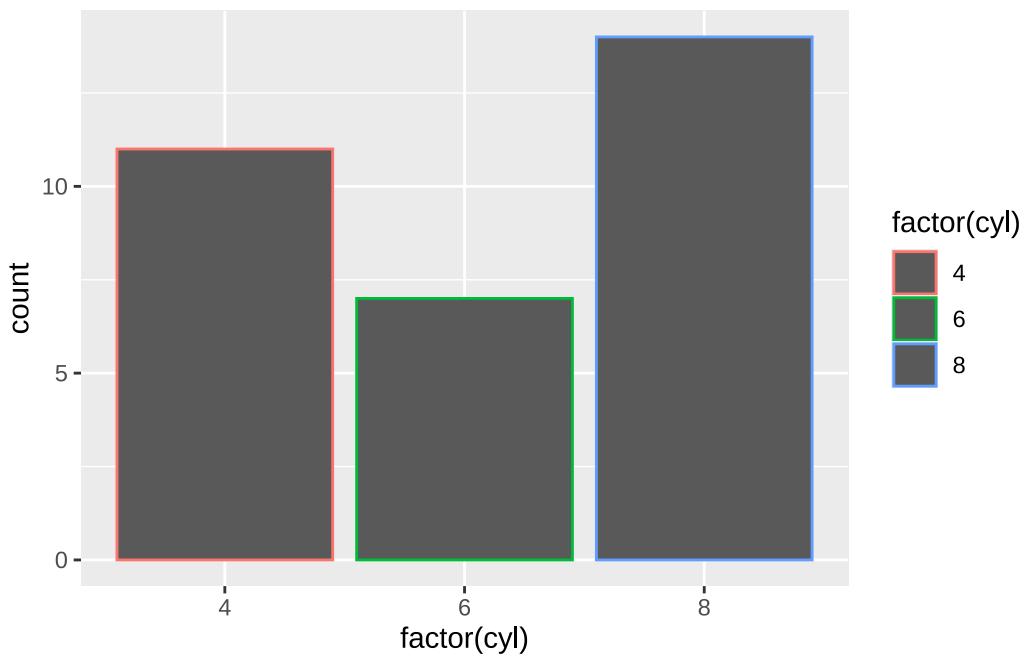
```
ggplot(data=diamonds,aes(x=clarity,color=cut))+  
  geom_bar()
```



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



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

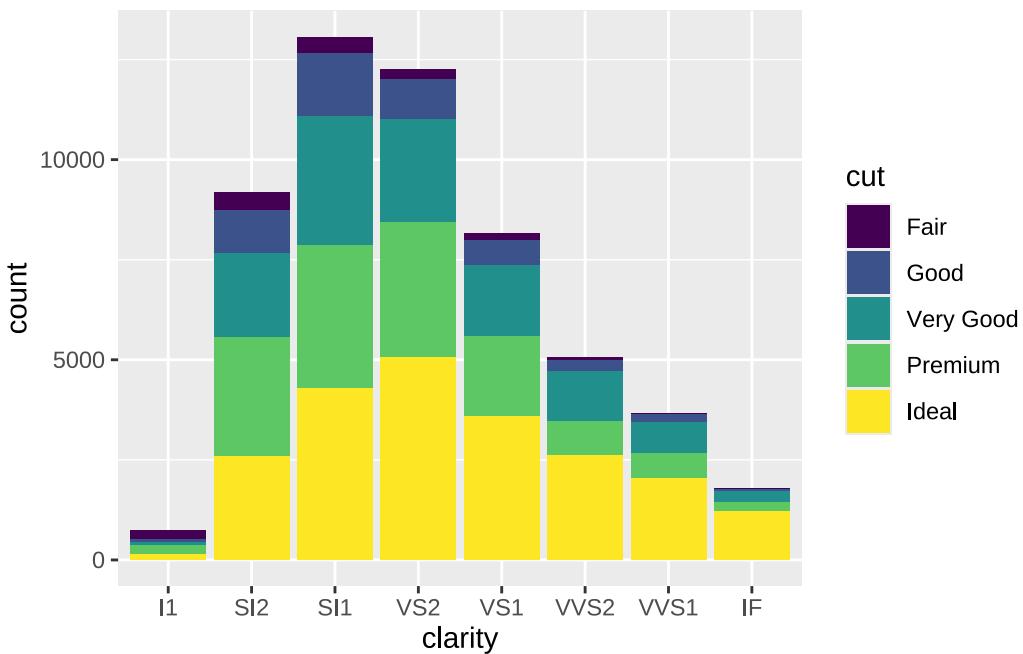


7.4 Color systems

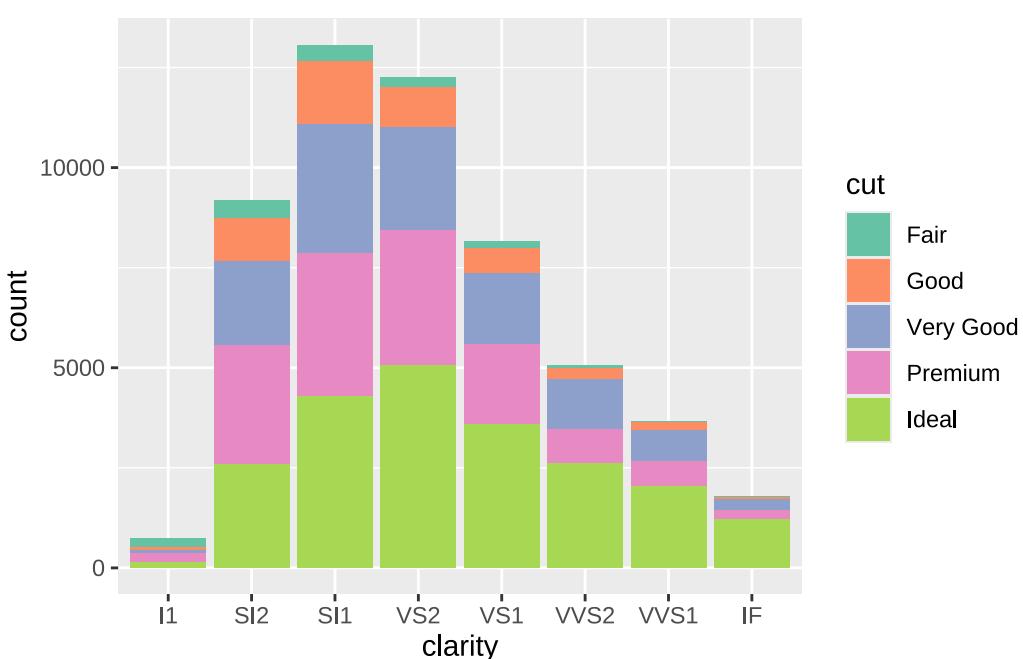
ggplot2 comes with various color definitions, many external packages extend that. Manual definition of colors is possible as well. Redefining the mapping between data and aesthetics can be done with scale_... functions

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

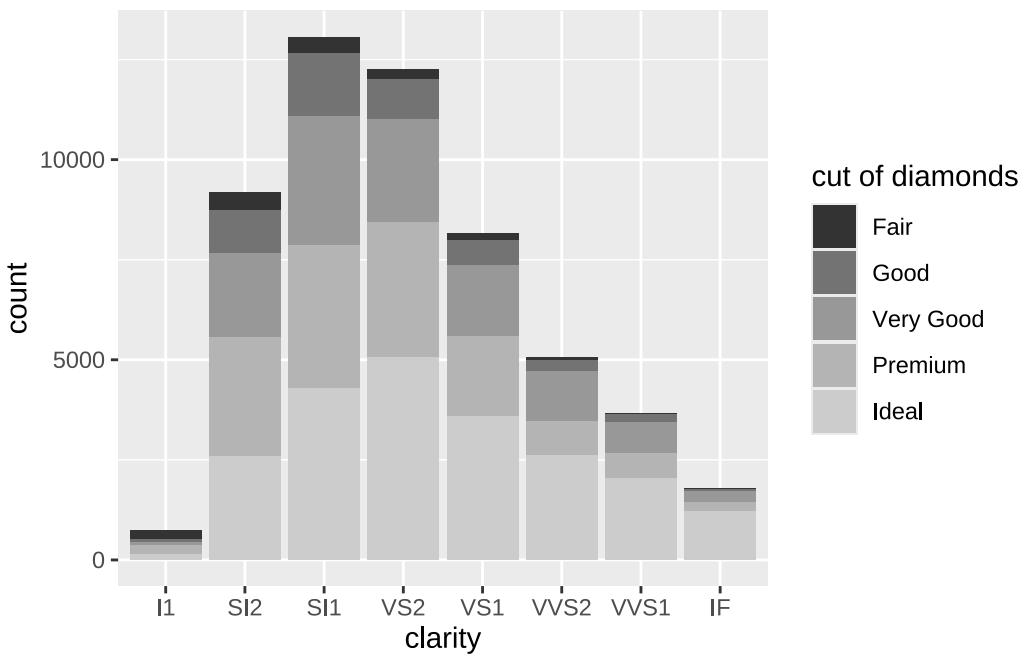
```
(plottemp <- ggplot(data=diamonds, aes(x=clarity, fill=cut))+
  geom_bar())
```



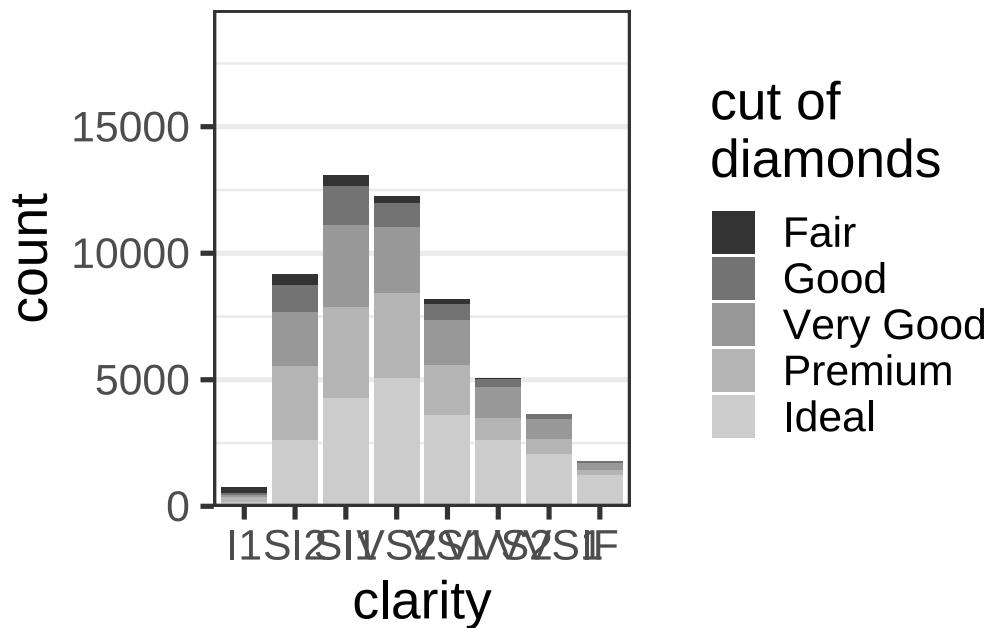
```
plottemp + scale_fill_brewer(palette="Set2") #in-built "scale" for fill
```



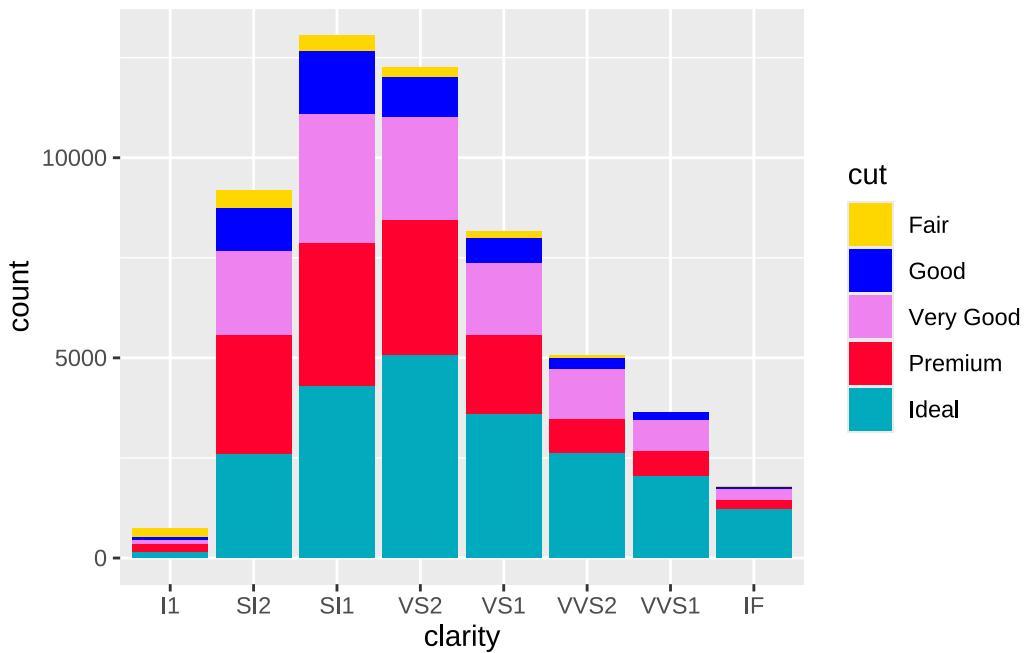
```
plottemp + scale_fill_grey(name = "cut of diamonds")
```



```
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())
```

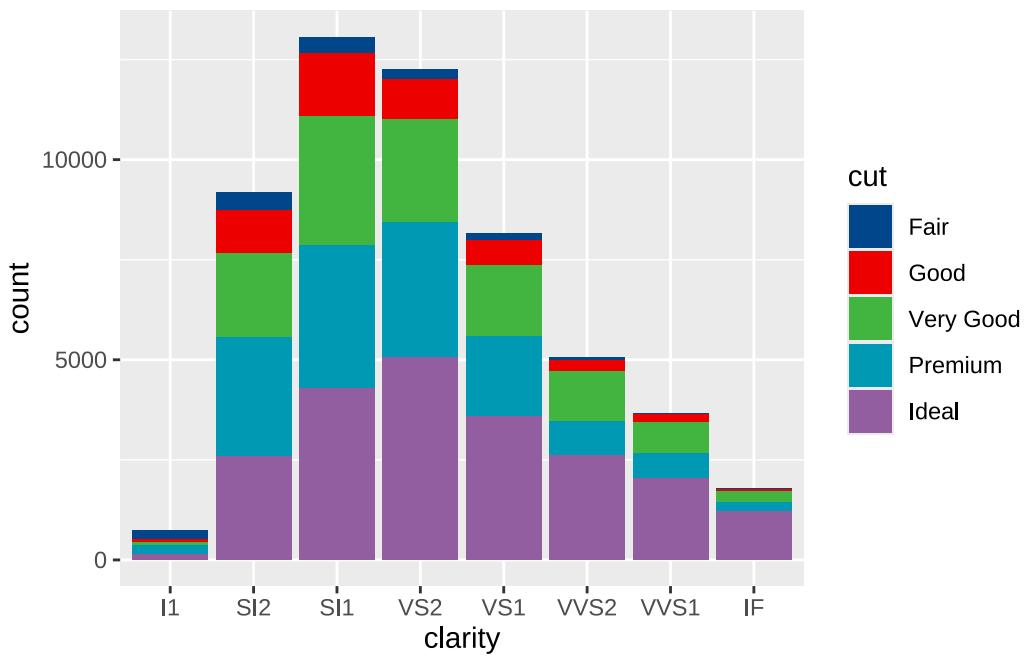


```
my_colors <- c("gold",'blue','violet',"#FF012F","#01AABC")
plottemp+ scale_fill_manual(values=my_colors)
```

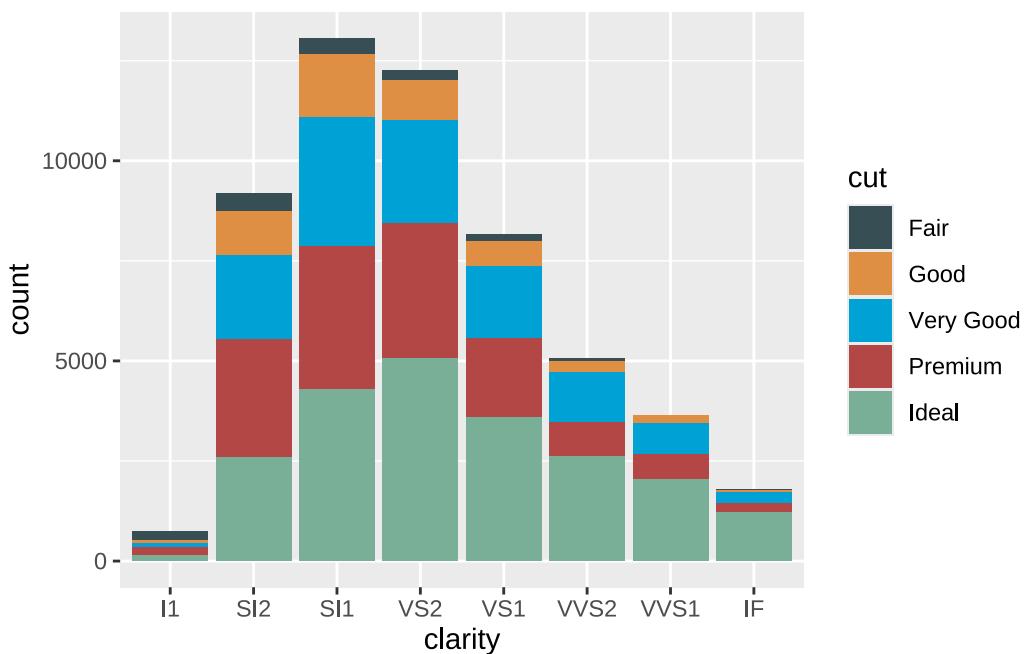


7.4.1 External color definitions from ggsci

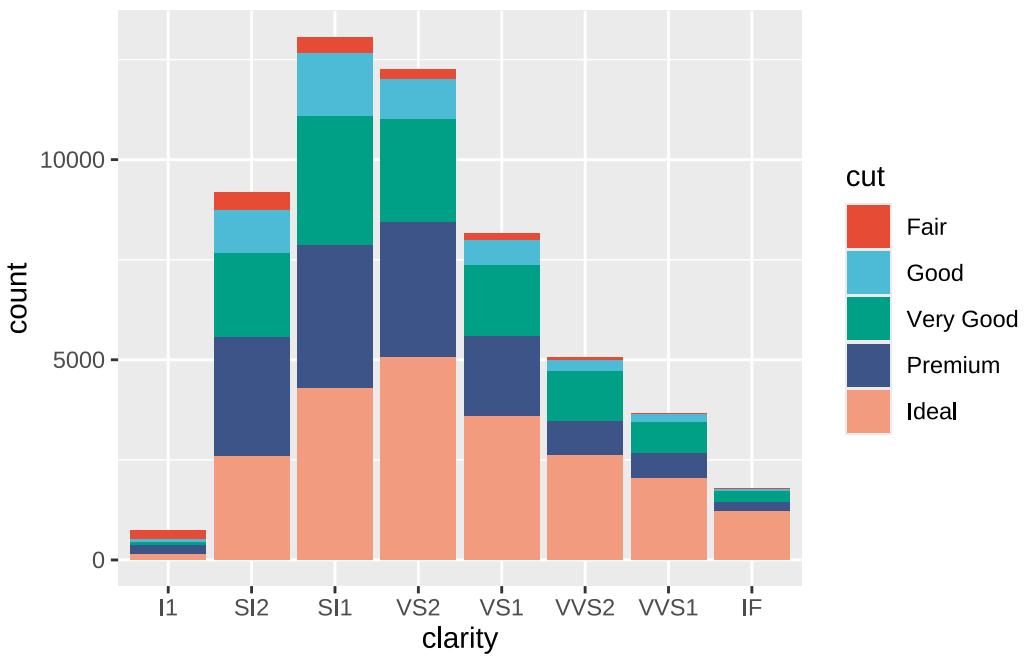
```
plottemp+scale_fill_lancet()
```



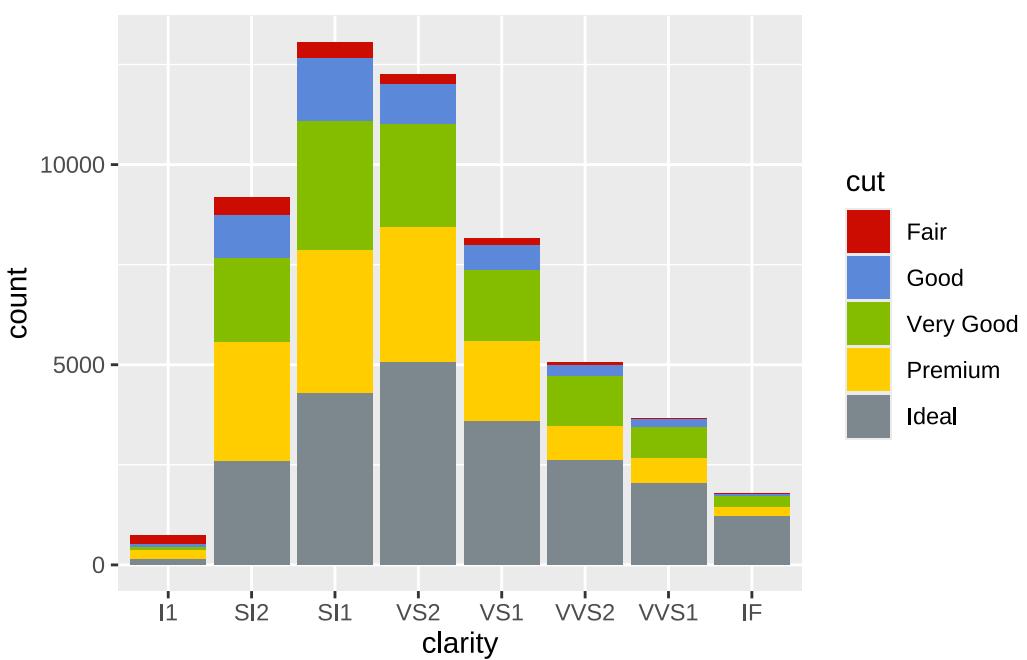
```
plottemp+scale_fill_jama()
```



```
plottemp+scale_fill_npg()
```



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



7.5 Exporting ggplots

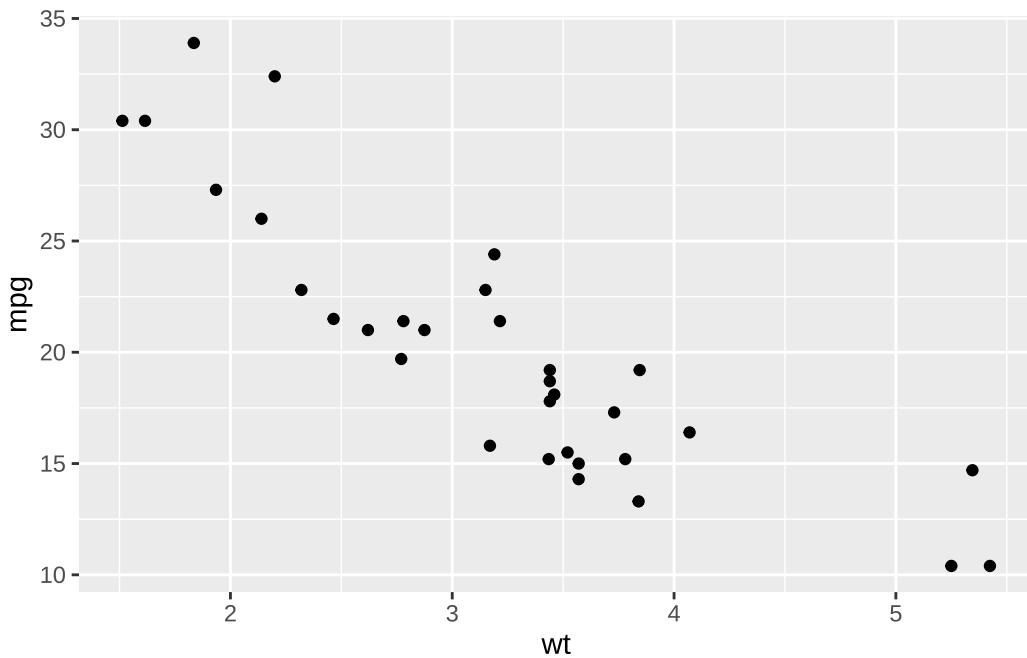
There are two distinct ways, using `ggsave()` or more generally creating an external graphic device with e.g. `png()` / `tiff()` / `pdf()`:

```
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

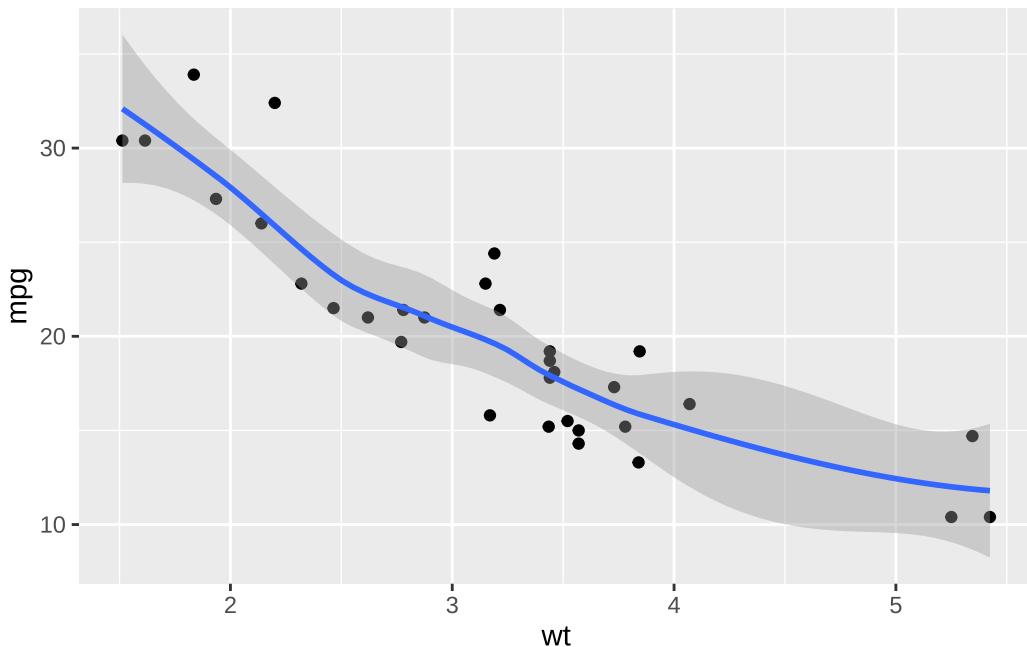
Common forms of plots are barplots, boxplots, scatterplots (possibly with regression line), and density-plots. For plotting dots for groups, there are various options to avoid overplotting of repeated data, with the `beeswarm` as my preference.

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



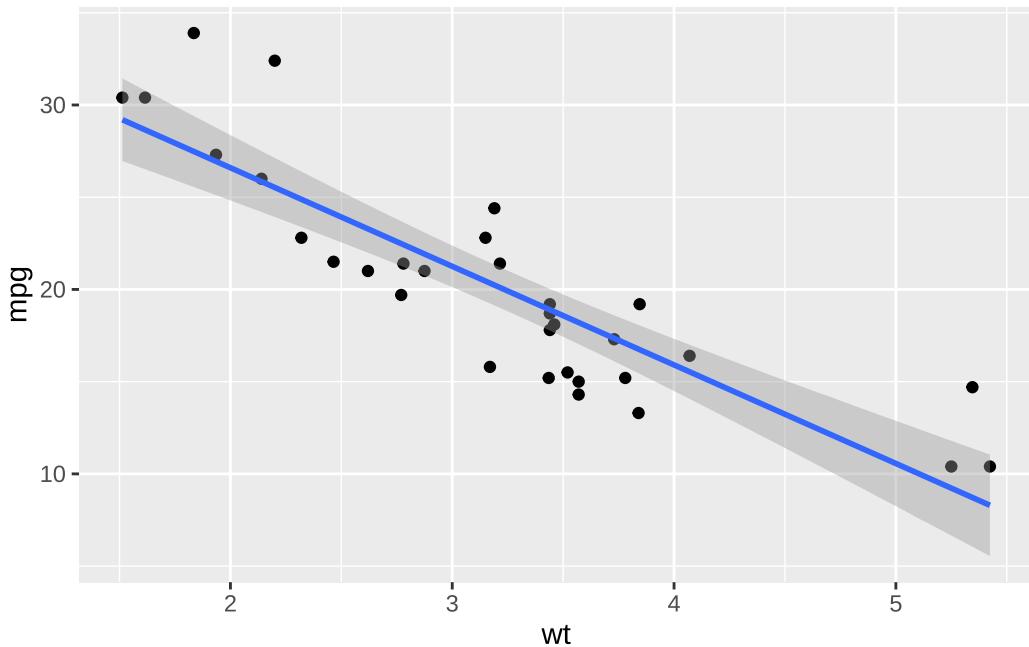
```
ggplot(data=mtcars,aes(x = wt,y = mpg))+  
  geom_point() +  
  geom_smooth()
```

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

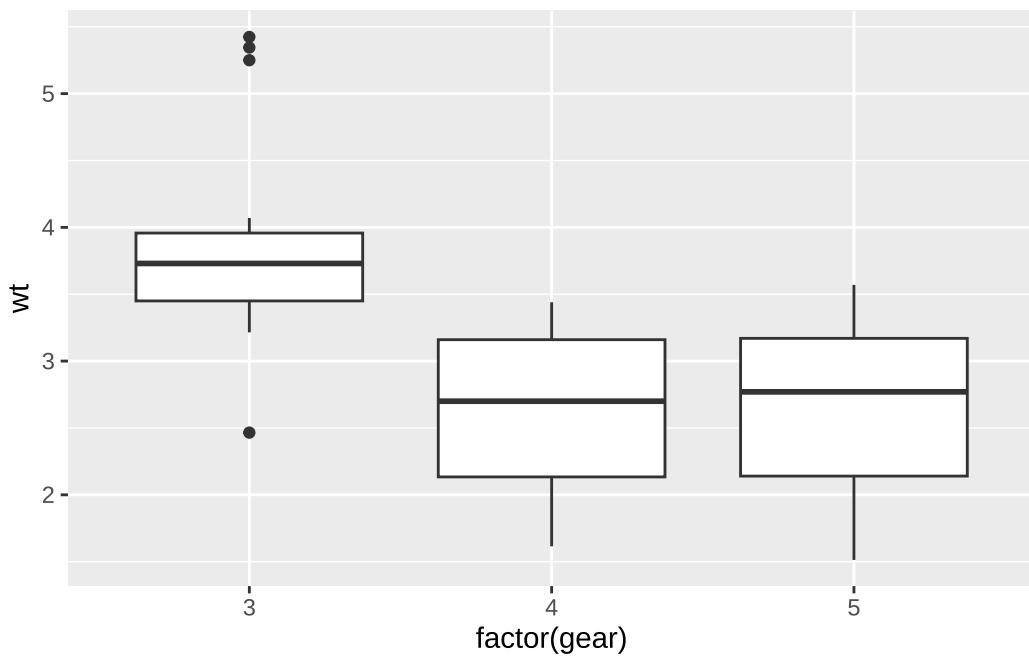


```
ggplot(data=mtcars,aes(x = wt,y = mpg))+  
  geom_point() +  
  geom_smooth(method="lm")
```

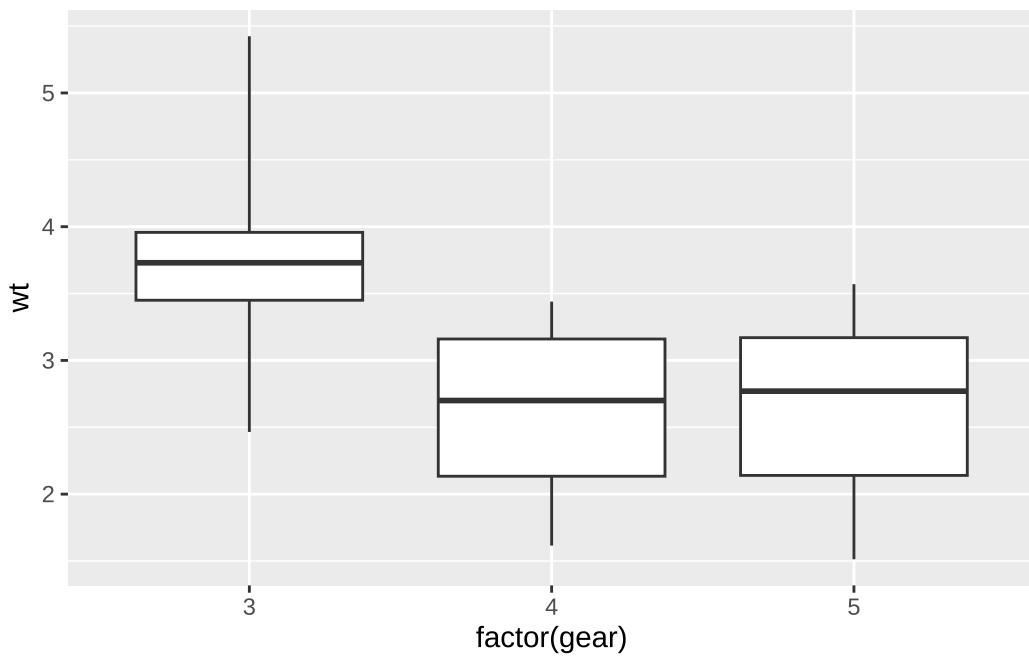
`geom_smooth()` using formula = 'y ~ x'



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

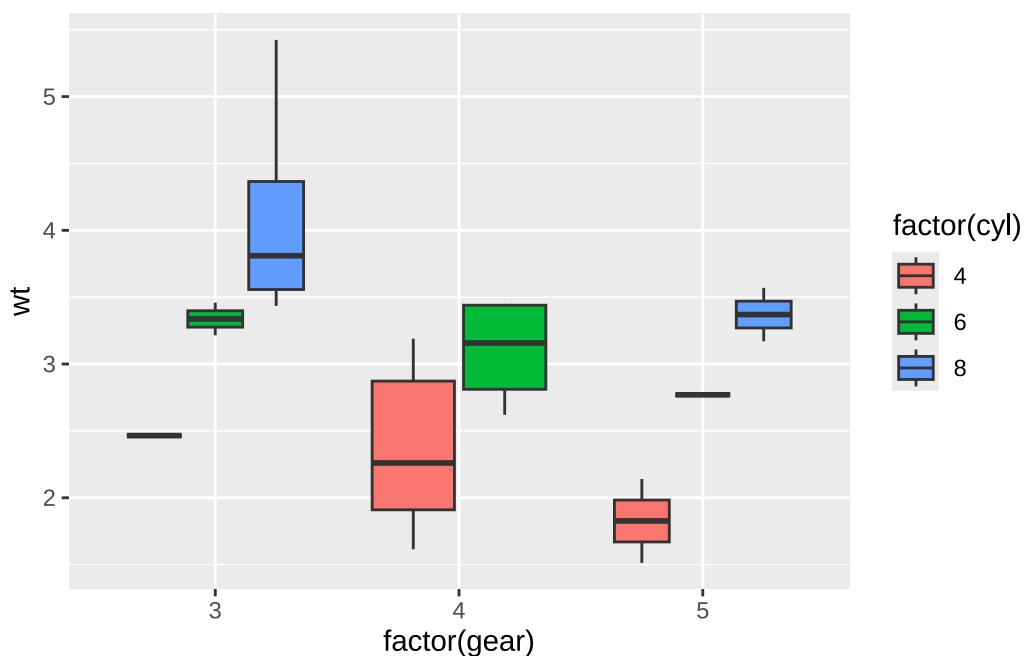


```
ggplot(mtcars,aes(x = factor(gear),y = wt))+  
  geom_boxplot(coef=3) # this extends range of expected values
```

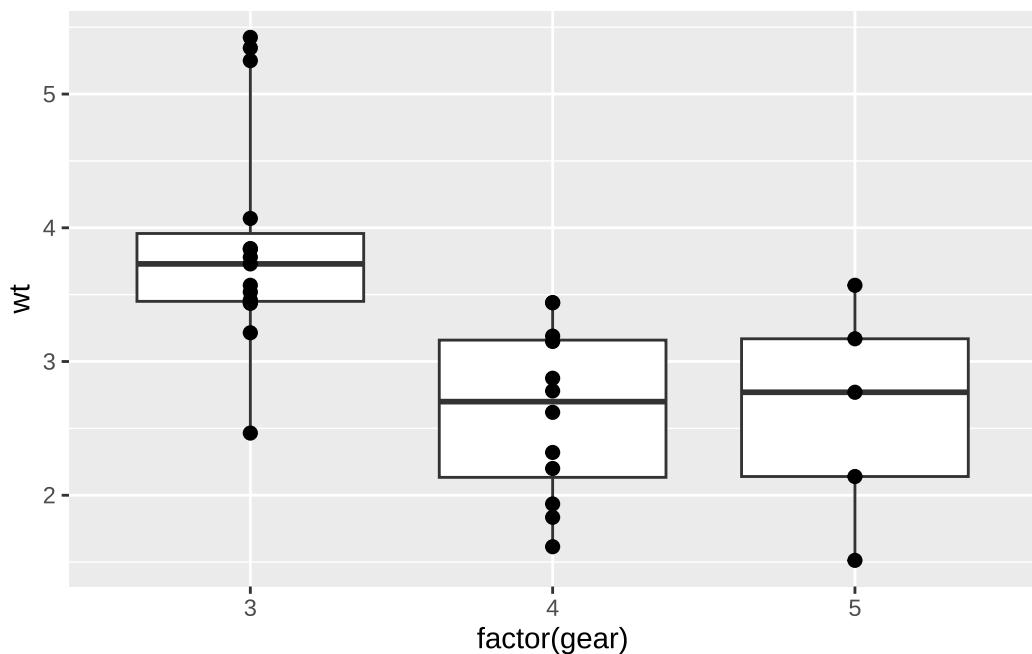


```
ggplot(mtcars,aes(x = factor(gear),y = wt,  
  fill=factor(cyl)))+
```

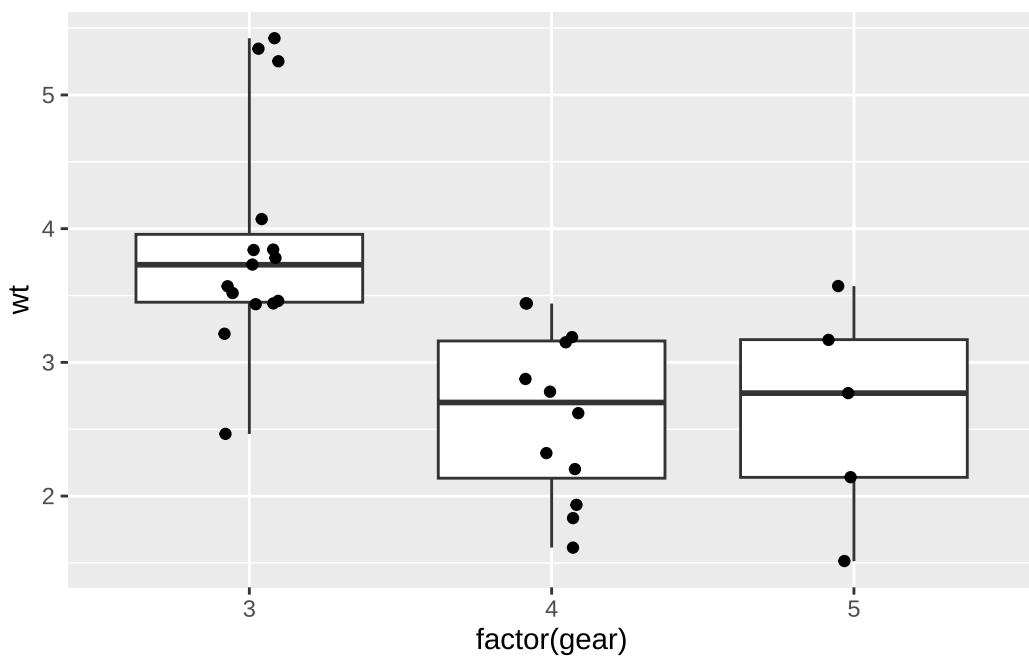
```
geom_boxplot(coef=3) # group by cyl, as it is mapped to fill
```



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

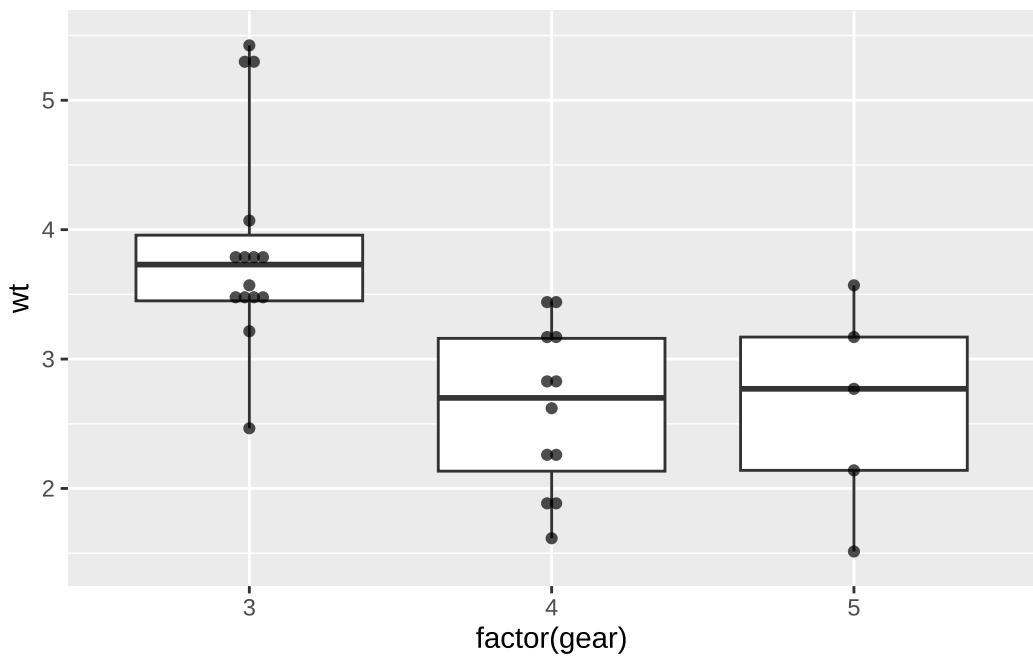


```
ggplot(mtcars,aes(x = factor(gear),y = wt))+  
  geom_boxplot(coef=3)+  
  geom_point(position = position_jitter(width = .1))
```

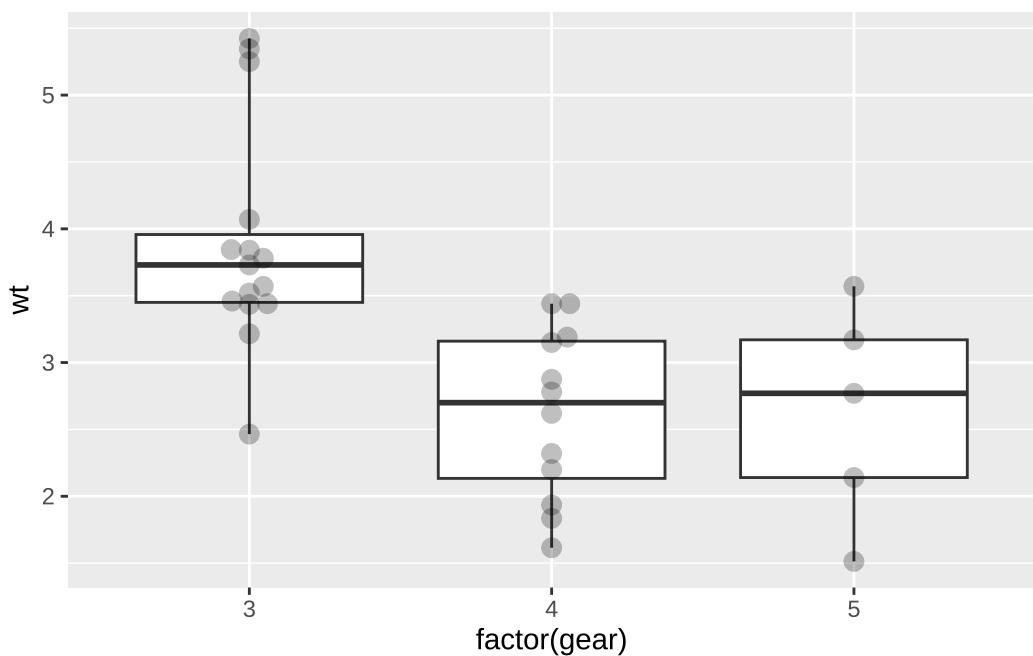


```
ggplot(mtcars,aes(x = factor(gear),y = wt))+  
  geom_boxplot(coef=3)+  
  geom_dotplot(alpha=.7, # group similar(ish) data on a line  
               binaxis = "y",stackdir = "center",  
               stackratio = .9,dotsize = .6)
```

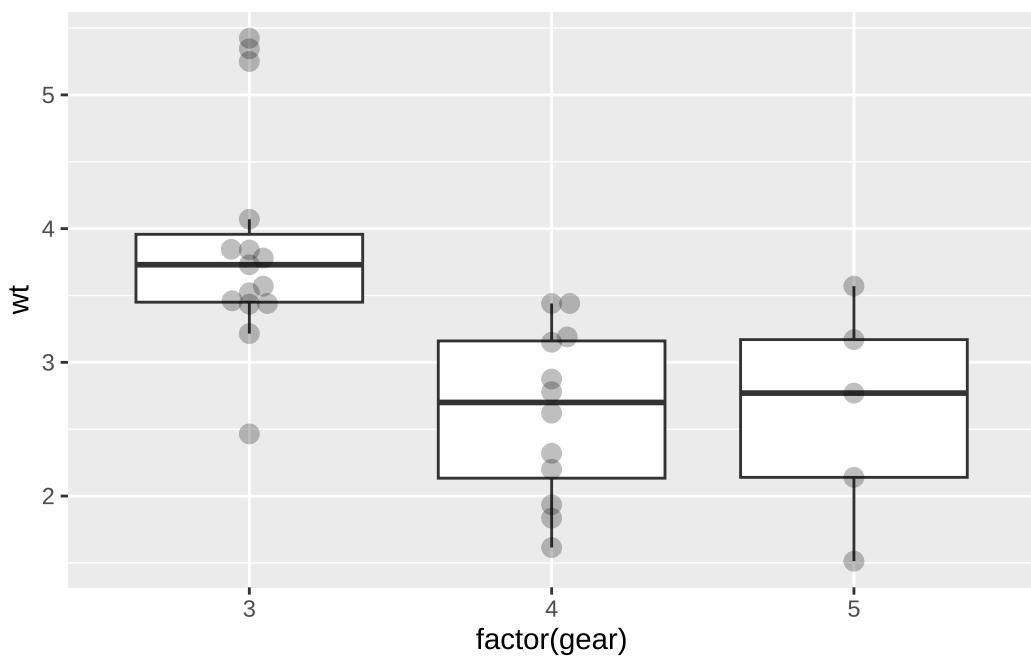
Bin width defaults to 1/30 of the range of the data. Pick better value with `binwidth`.



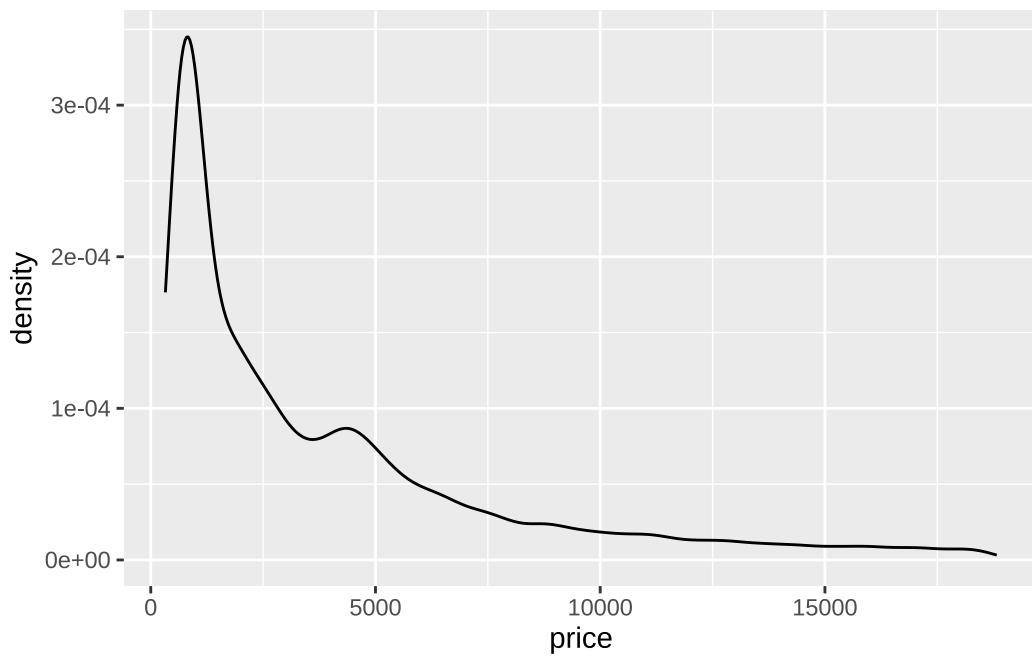
```
ggplot(mtcars,aes(x = factor(gear),y = wt))+  
  geom_boxplot(coef=3)+  
  ggbeeswarm::geom_beeswarm(cex = 2,size=3,alpha=.25)
```



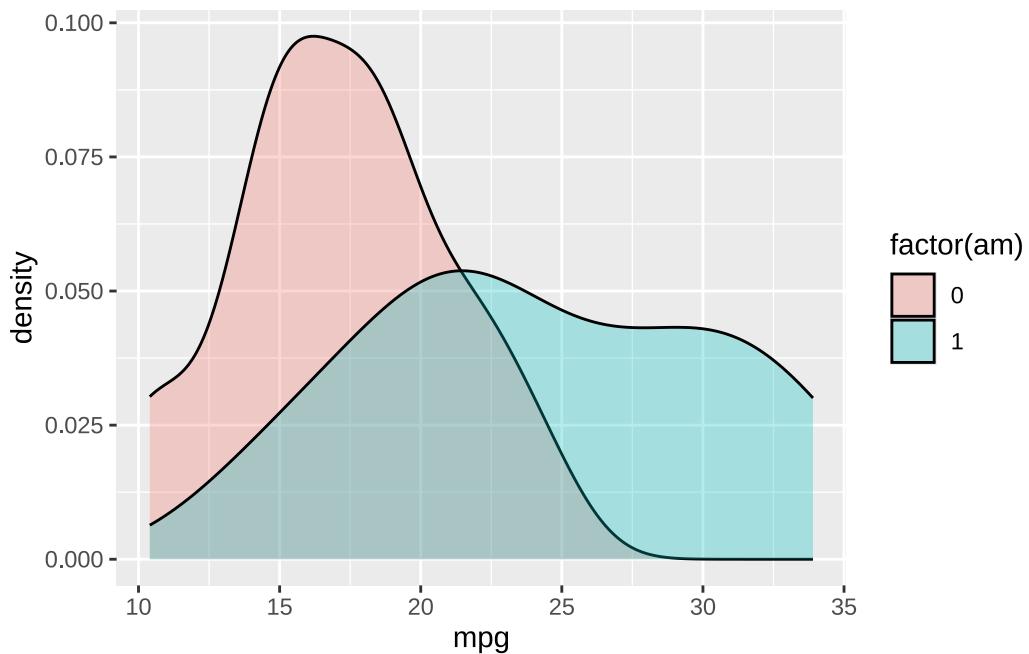
```
ggplot(mtcars,aes(x = factor(gear),y = wt))+  
  geom_boxplot(outlier.alpha = 0)+ # to avoid plotting outliers twice  
  geom_beeswarm(cex = 2,size=3,alpha=.25)
```



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

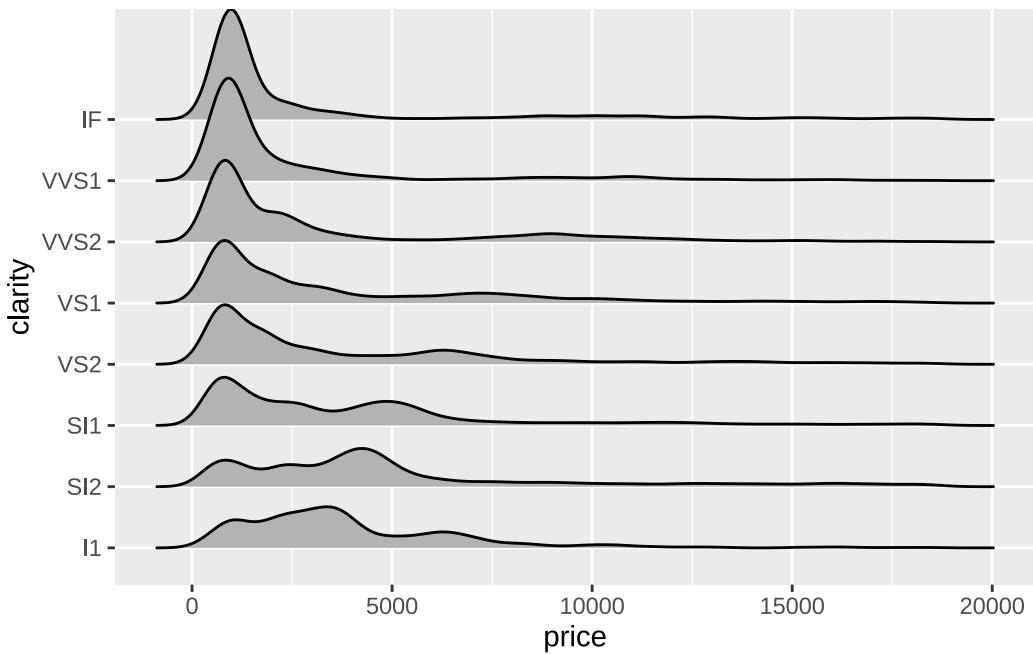


```
ggplot(mtcars,aes(mpg, fill=factor(am)))+
  geom_density(alpha=.3)
```

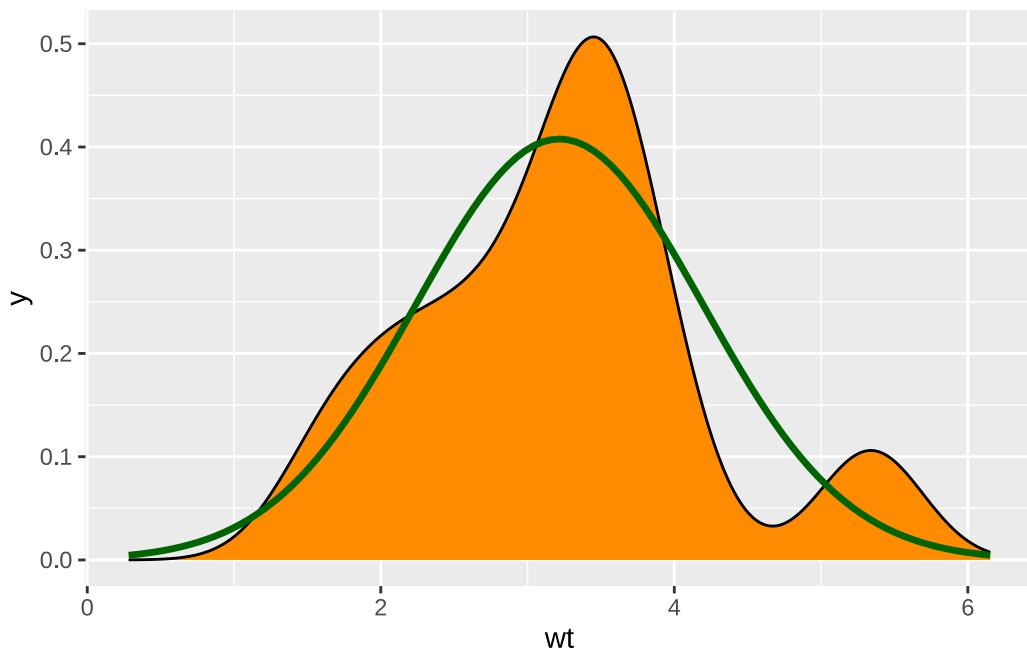


```
ggplot(diamonds,aes(price,y=clarity))+
  geom_density_ridges()
```

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)))
```



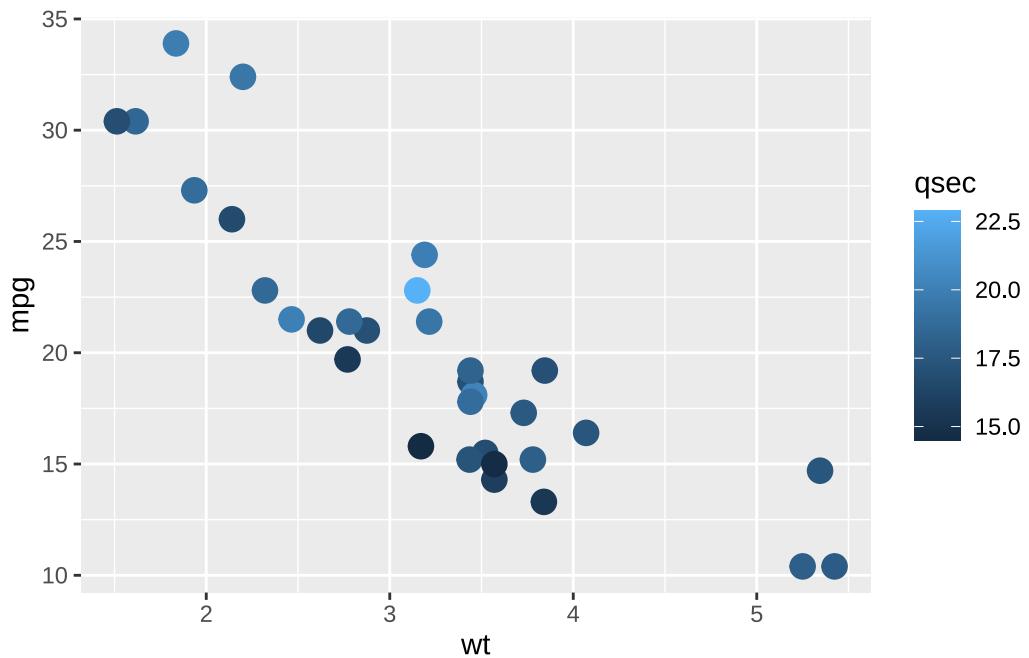
7.7 Exercise 1

Vizualize the following:

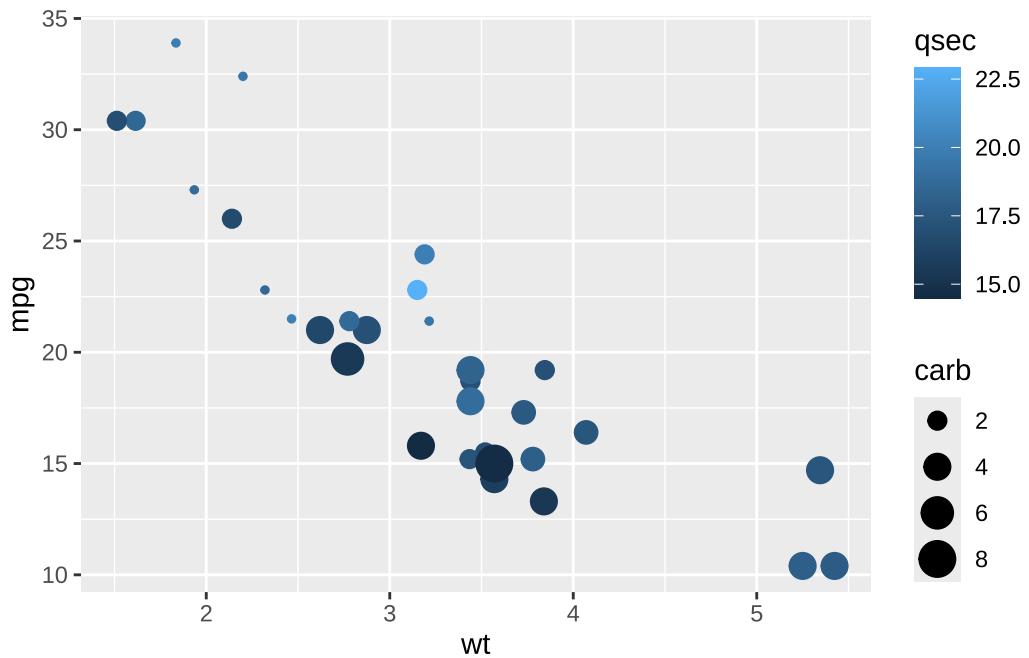
- count of sex within species
- boxplot+beeswarm for weight and species
- boxplot+beeswarm weight for species AND sex
- scatterplot for flipper length vs. body mass
- add a regression line to that plot
- do the same scatterplot and regression grouped by species and sex
- optionally, define your own colors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

7.8 Combining and finetuning aesthetics

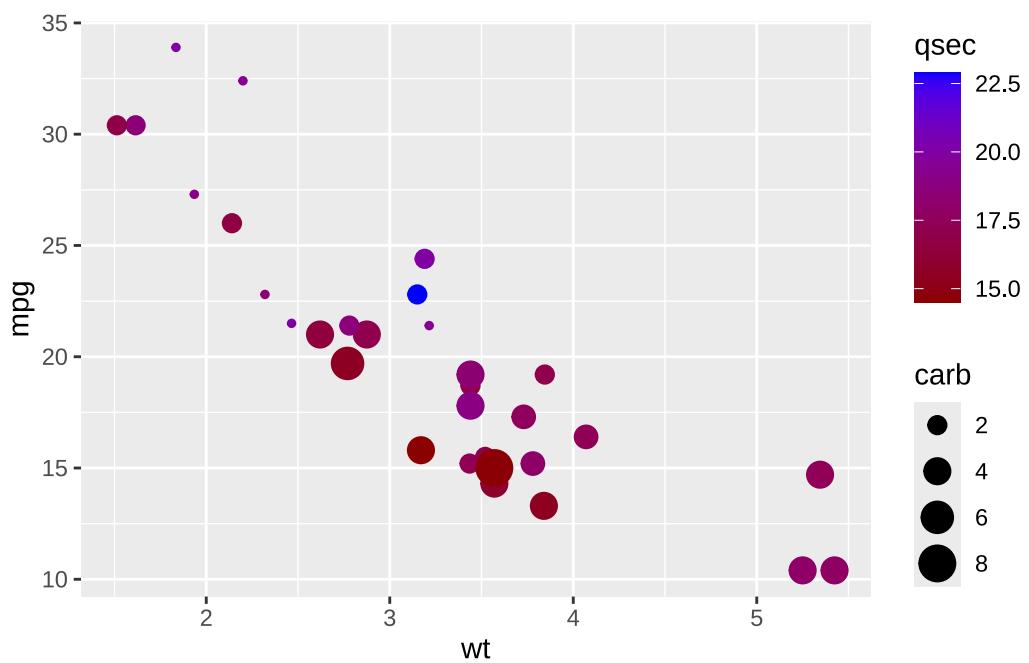
```
ggplot(data=mtcars,aes(wt, mpg,color=qsec))+  
  geom_point(size=4) #outside aes!
```



```
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,),  

                     minor_breaks=seq(-10^3,10^3,1),  

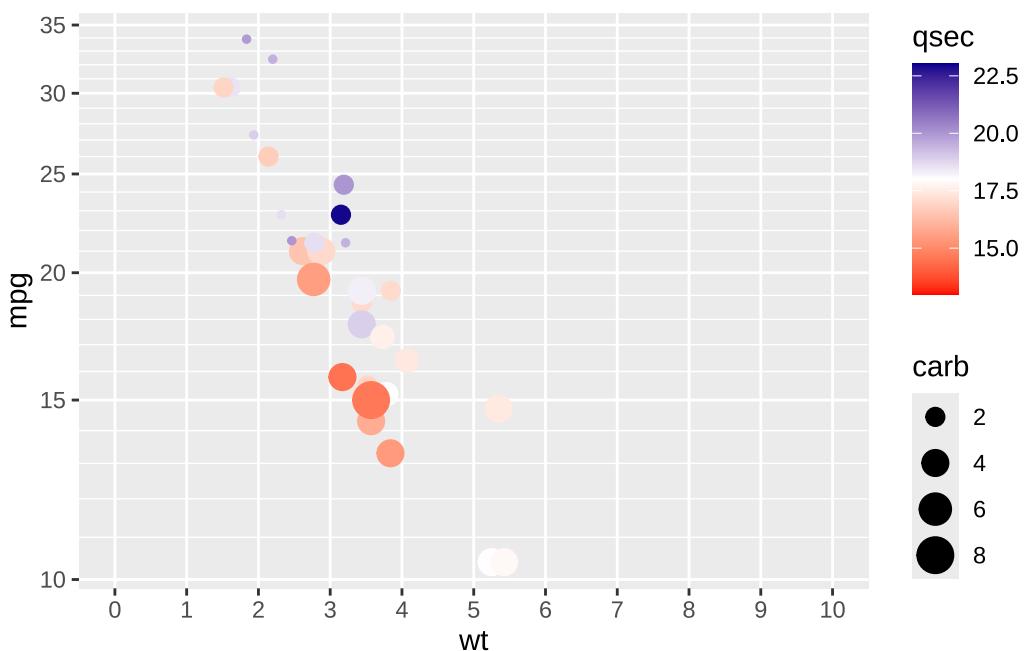
                     limits = c(0,10))+  

  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)+  

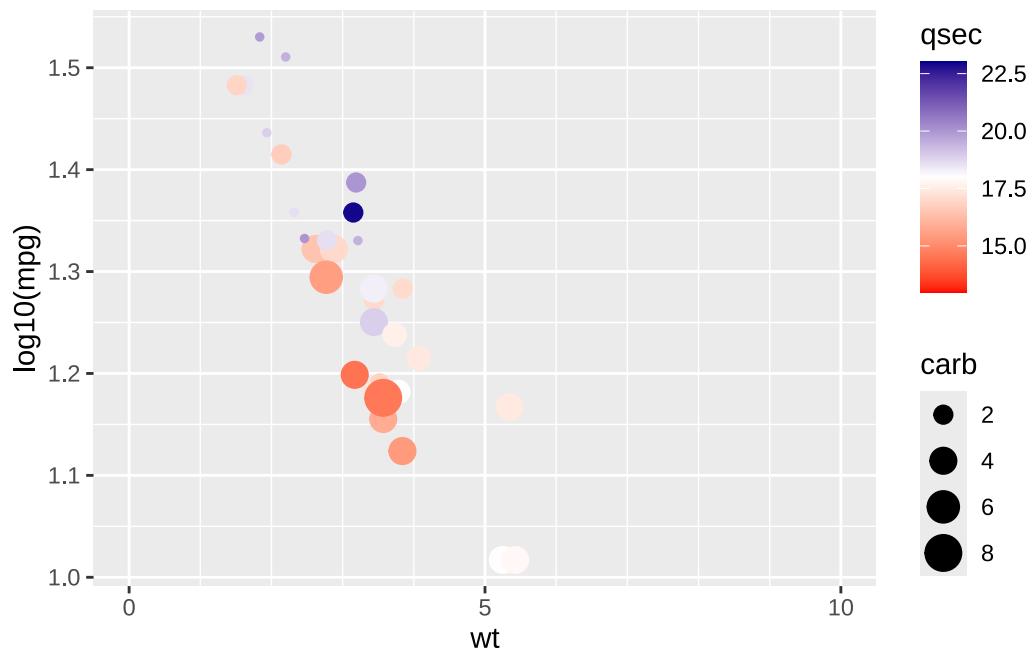
  geom_point() +  

  scale_x_continuous(breaks = seq(0,100,5),  

                     minor_breaks=seq(0,100,1),  

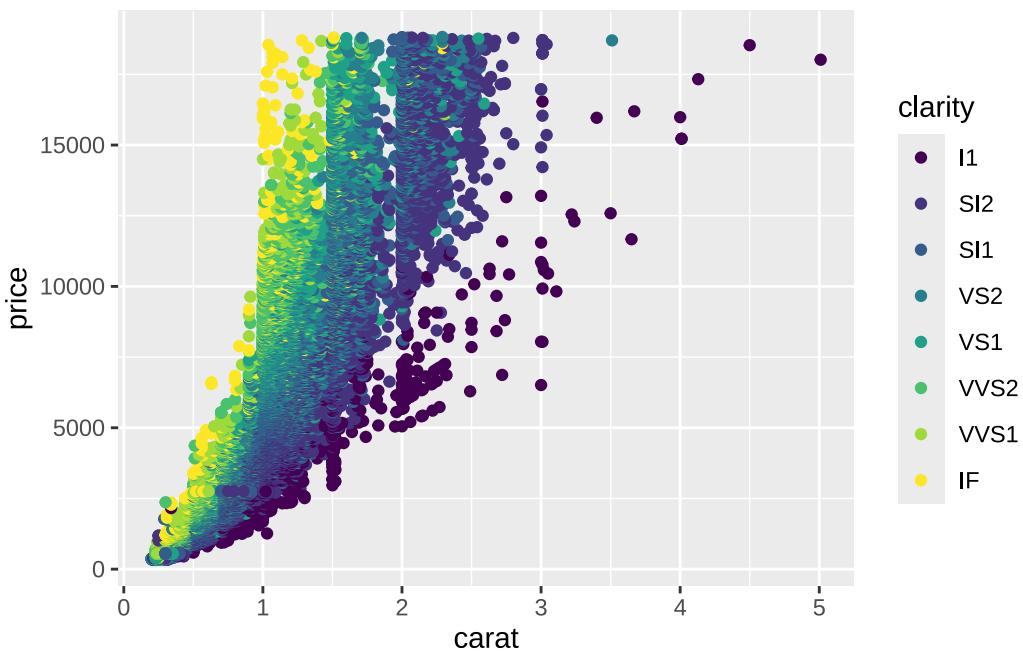
                     limits = c(0,10))#+

```

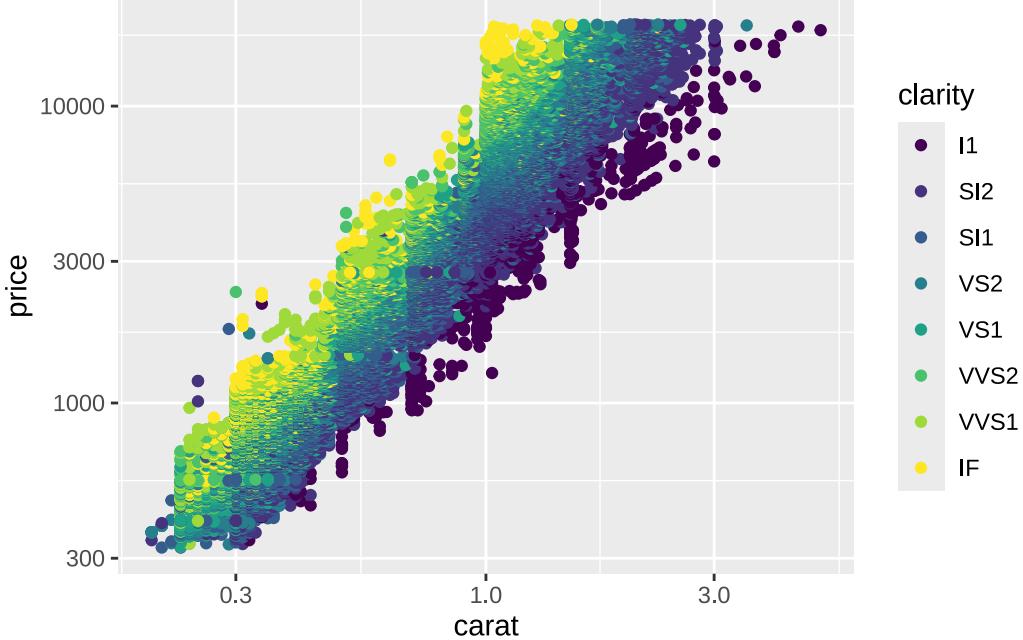


```
# scale_y_log10(minor_breaks=seq(0,100,1))

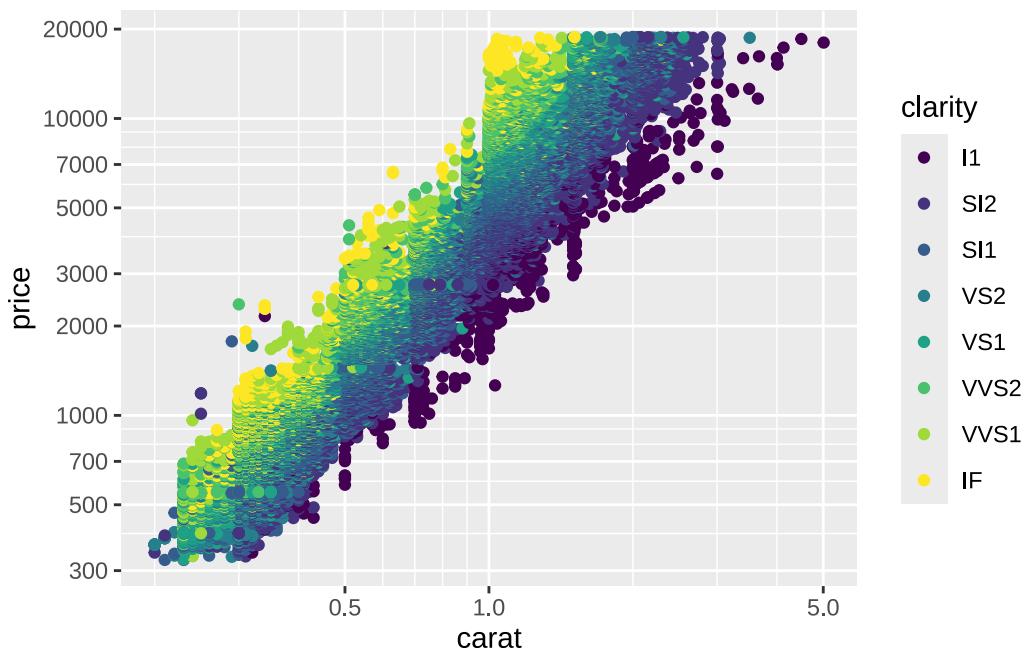
ggplot(diamonds,aes(carat,price,color=clarity))+
  geom_point()
```



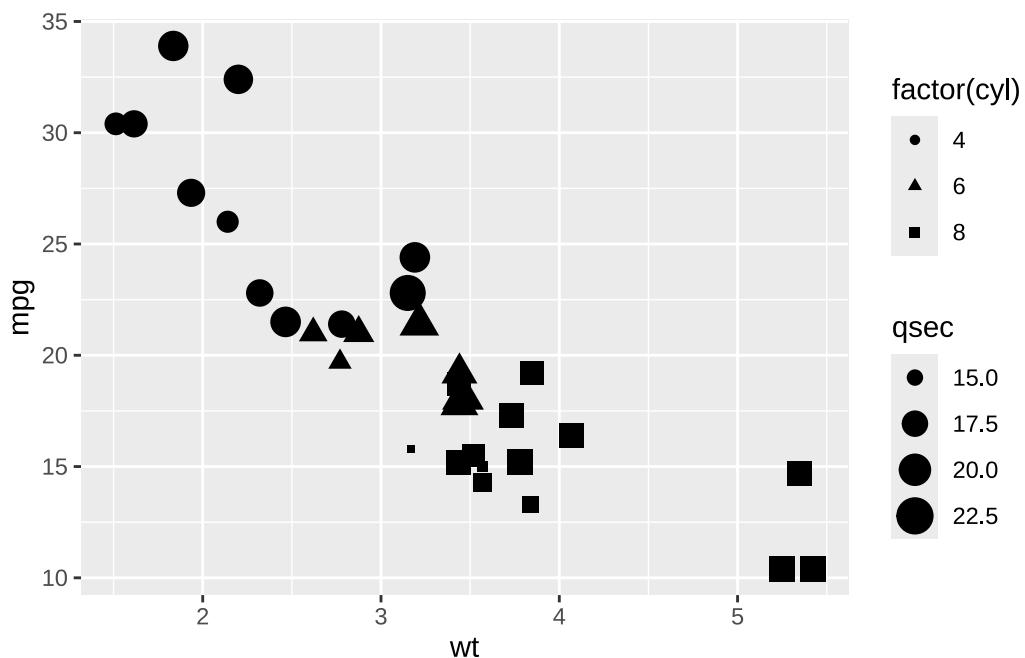
```
ggplot(diamonds,aes(carat,price,color=clarity))+  
  geom_point() +  
  scale_x_log10() +  
  scale_y_log10()
```



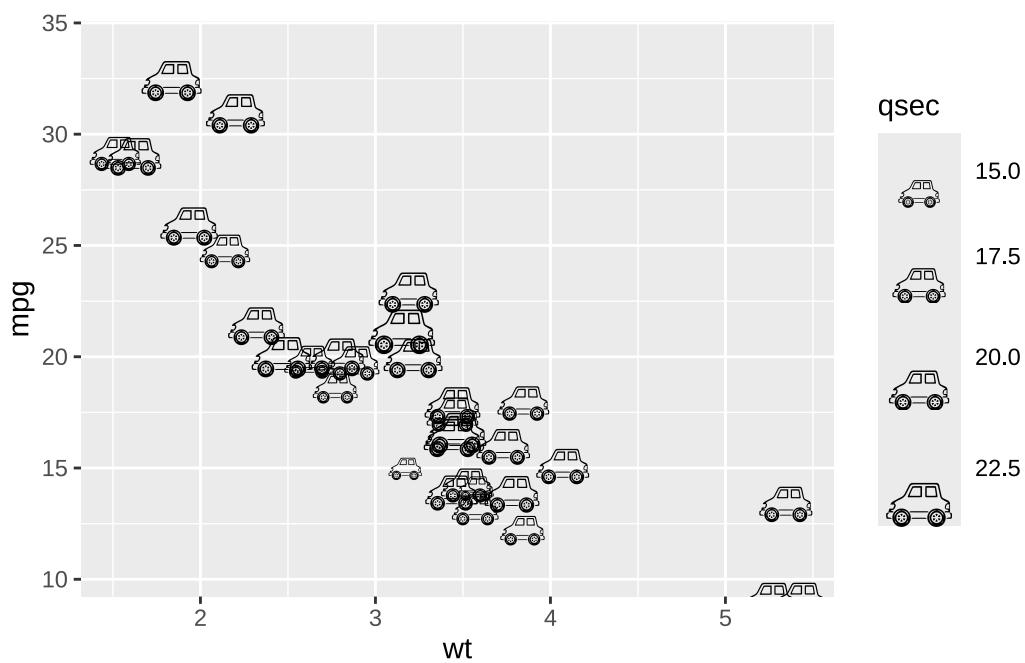
```
ggplot(diamonds,aes(carat,price,color=clarity))+  
  geom_point() +  
  scale_x_log10(  
    breaks=logrange_15,  
    minor_breaks=logrange_123456789) +  
  scale_y_log10(  
    breaks=logrange_12357,  
    minor_breaks=logrange_123456789)
```



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



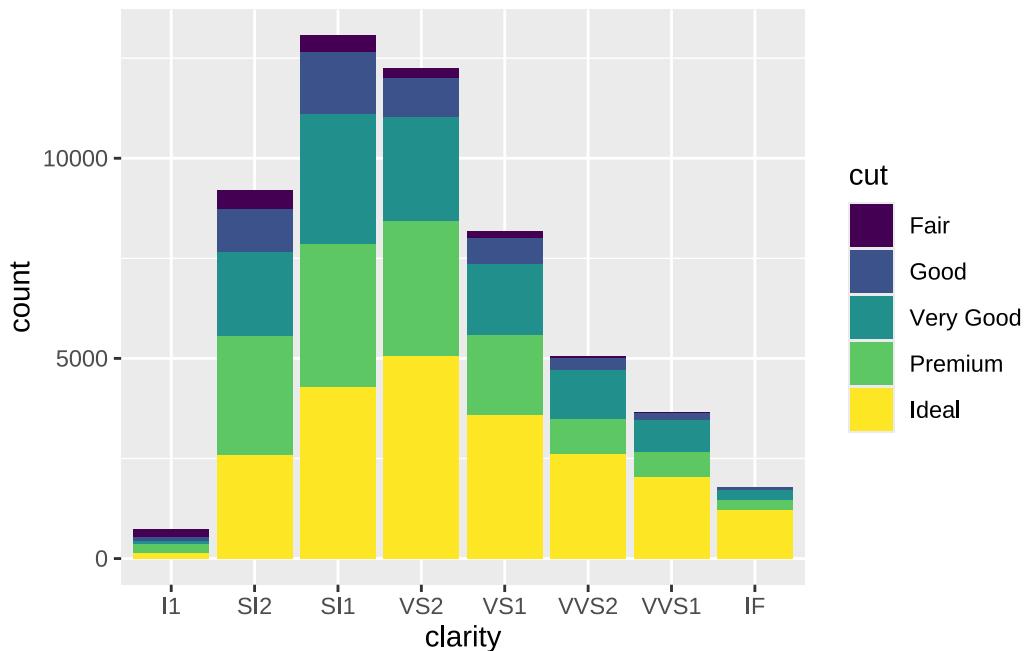
```
ggplot(data=mtcars,aes(wt, mpg, size=qsec))+  
  geom_text(family="EmojiOne",label="\U1F697") +  
  scale_size_continuous(range = c(5,10))
```



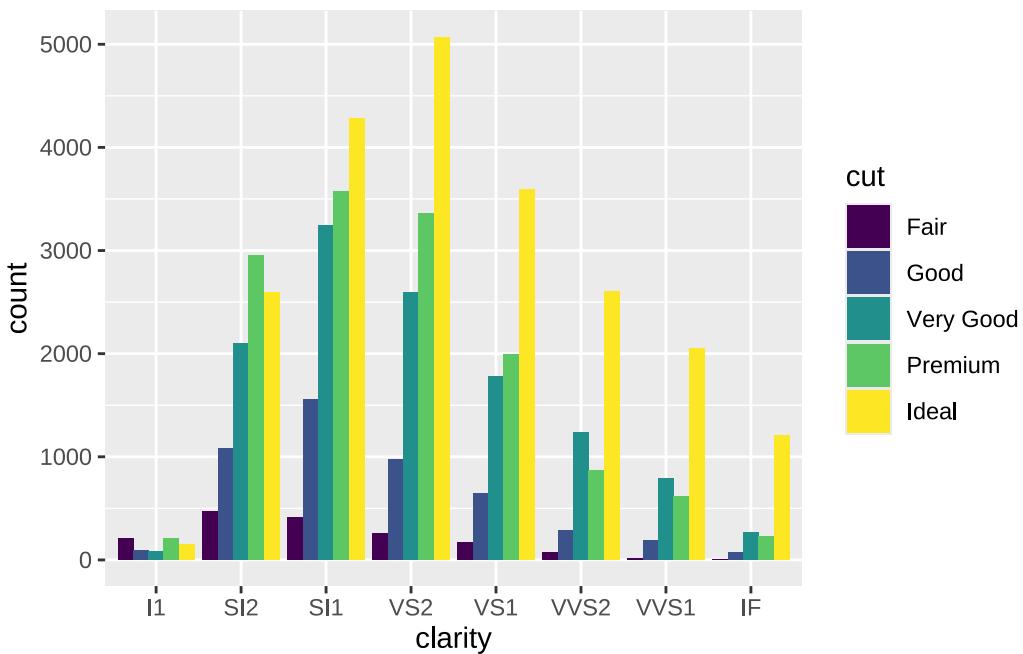
7.9 Positioning elements

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.

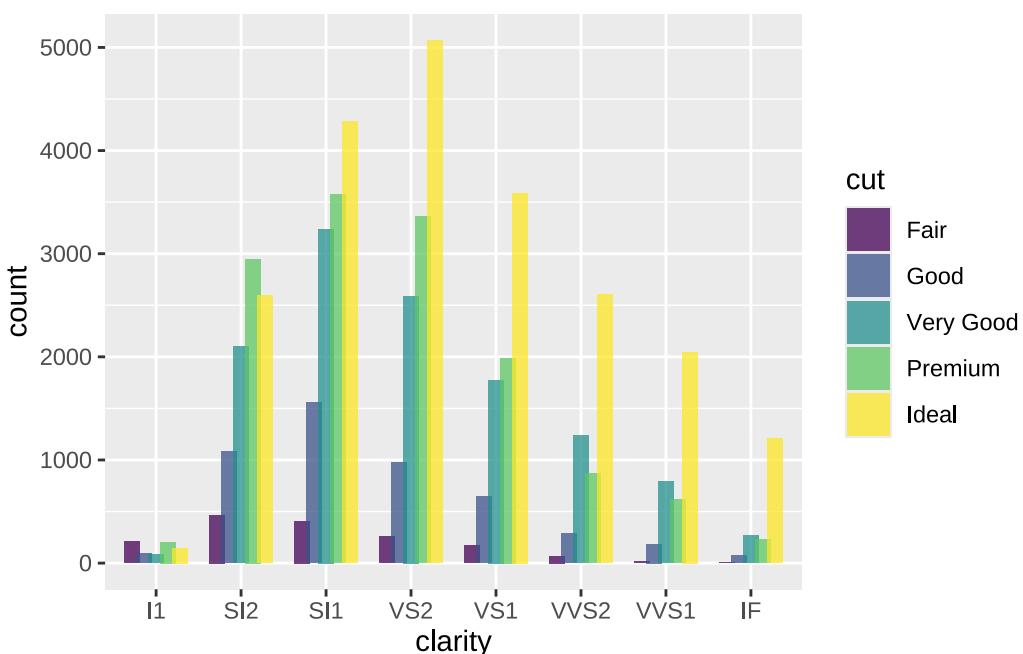
```
p<-ggplot(data=diamonds,aes(clarity,fill=cut))  
p+geom_bar(position="stack") # default for bar
```



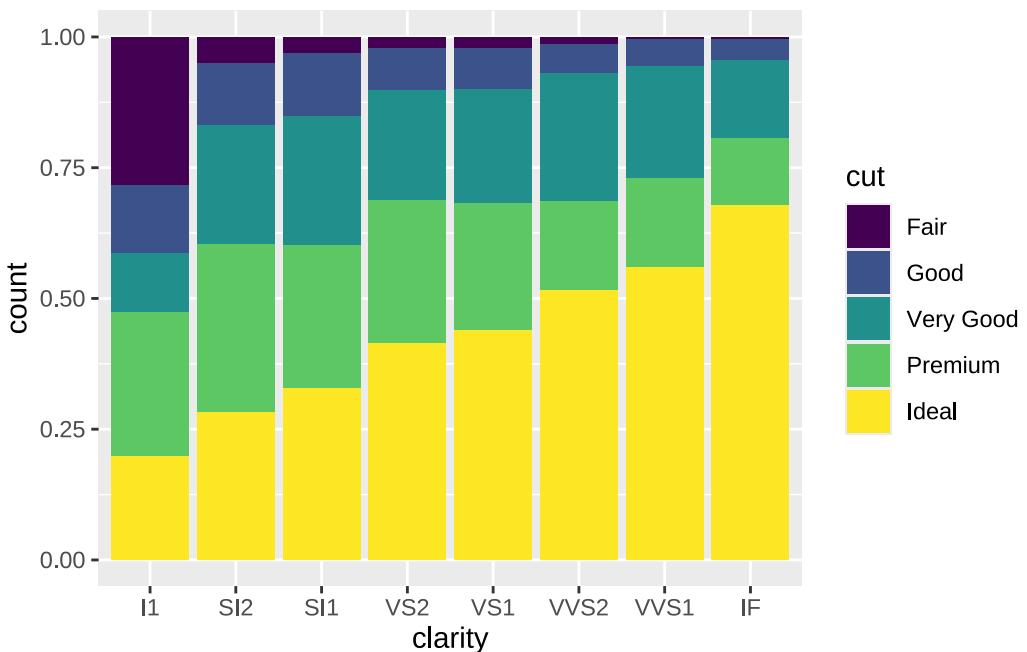
```
p+geom_bar(position="dodge")
```



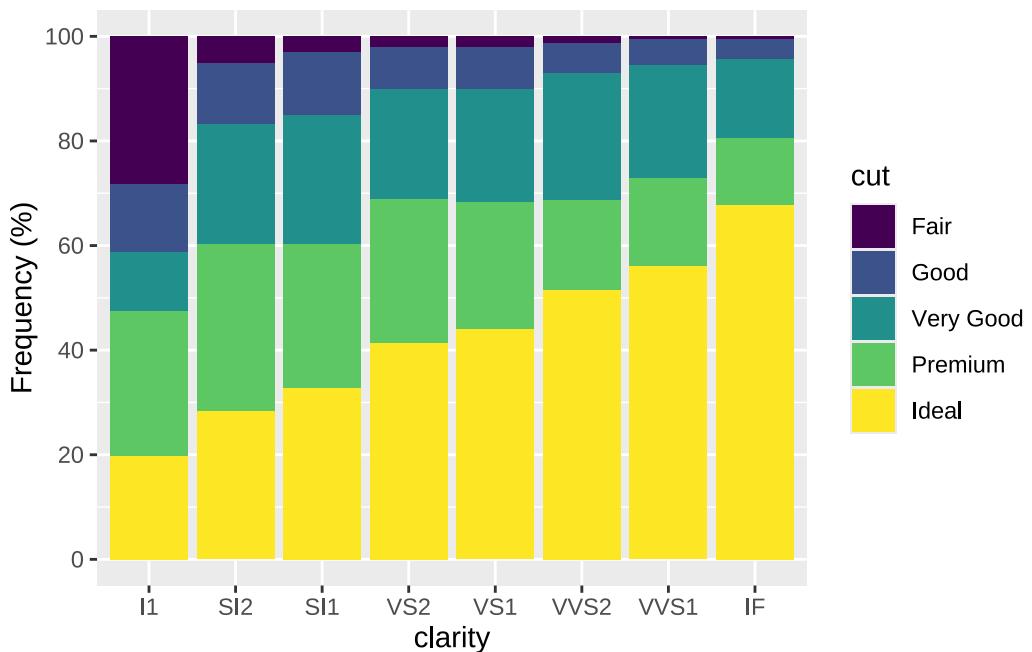
```
p+geom_bar(position=position_dodge(width = 0.7), alpha=.75)
```



```
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))
```

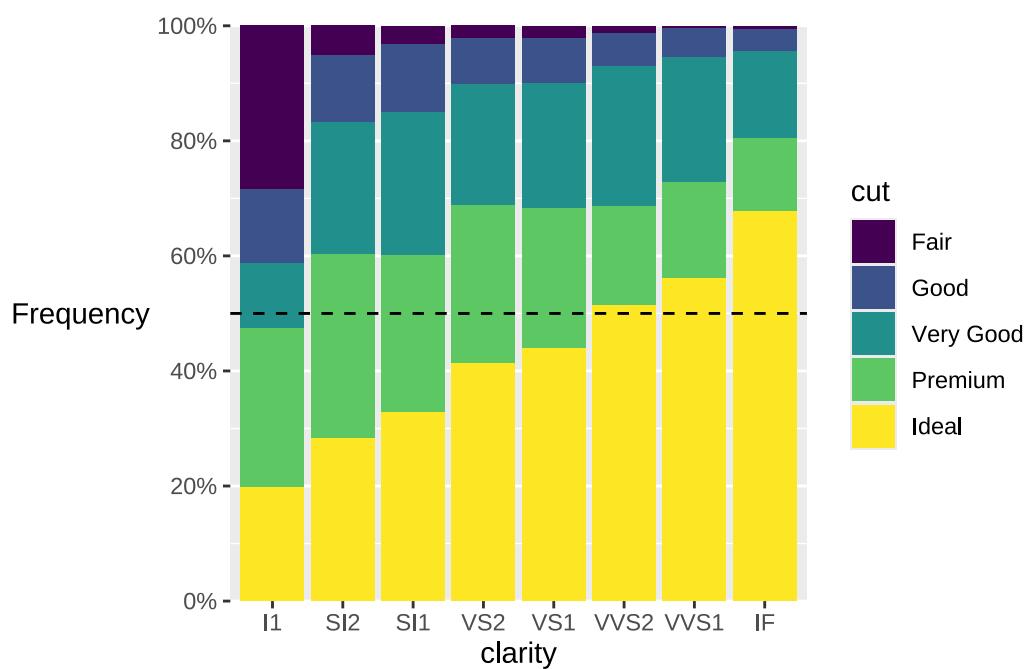


```

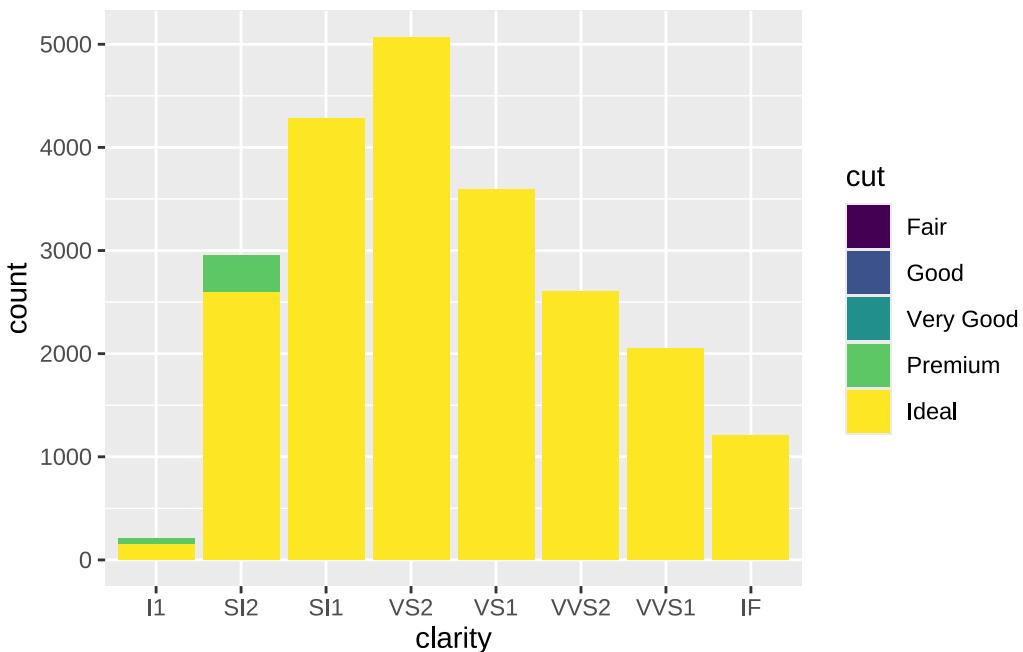
p+geom_bar(position="fill")+
  scale_y_continuous("Frequency",
                     breaks=seq(0,1,.2),
                     labels=scales::percent,
                     expand=expansion(mult = c(0,0)))+
  theme(axis.title.y = element_text(angle = 0,
                                    vjust = .5))+  

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

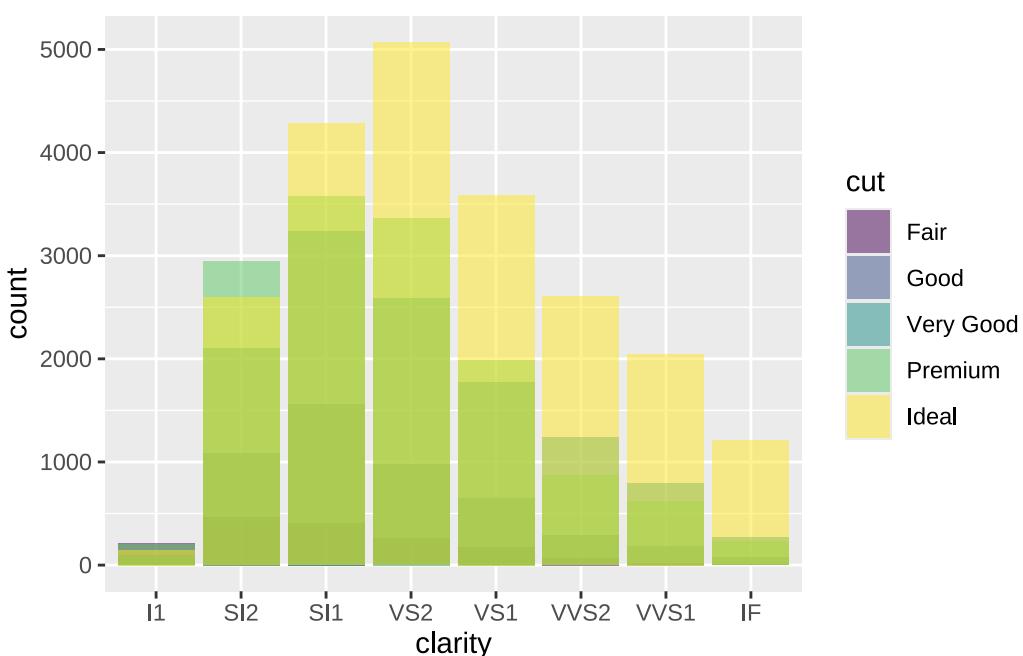
```



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

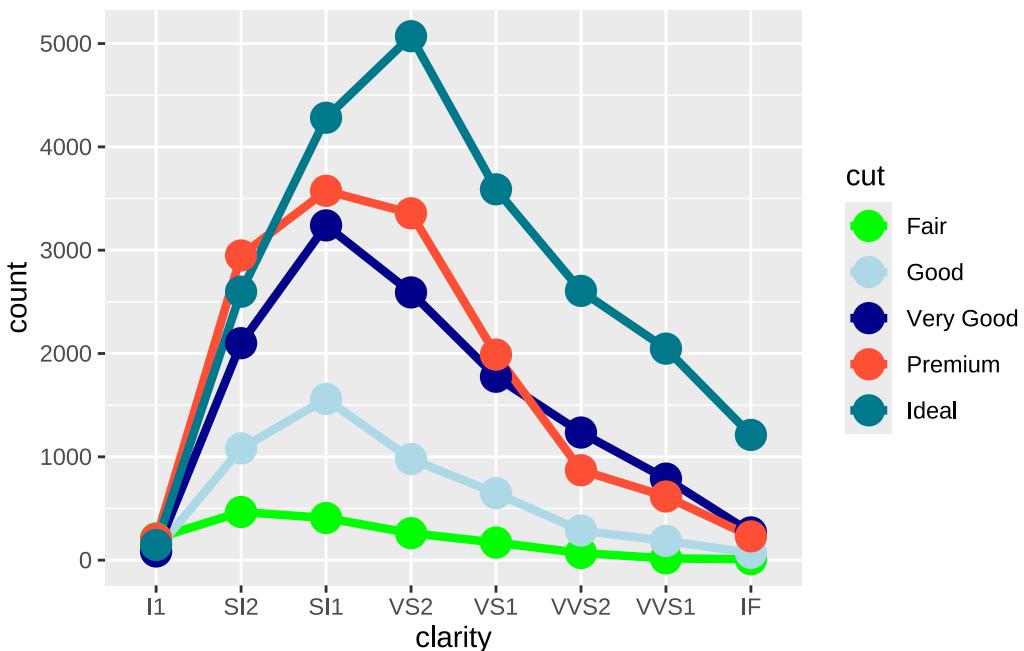


```
p+geom_bar(position="identity",alpha=.5) # even worse!!
```



```
ggplot(data=diamonds,aes(clarity,color=cut, group=cut))+  
  geom_freqpoly(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"))
```

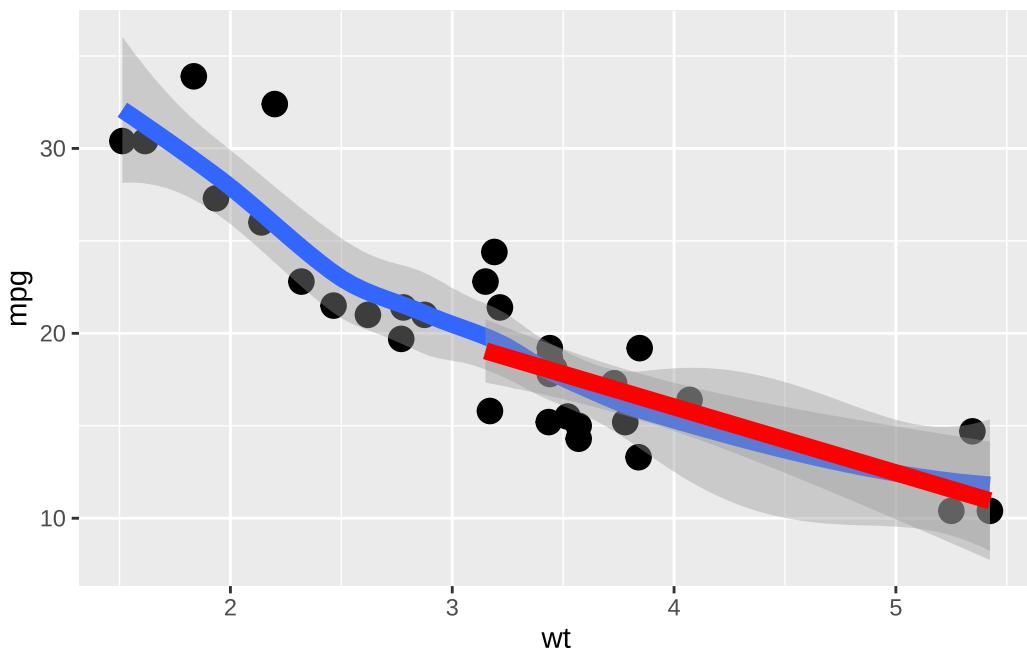


7.10 Order of layers

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

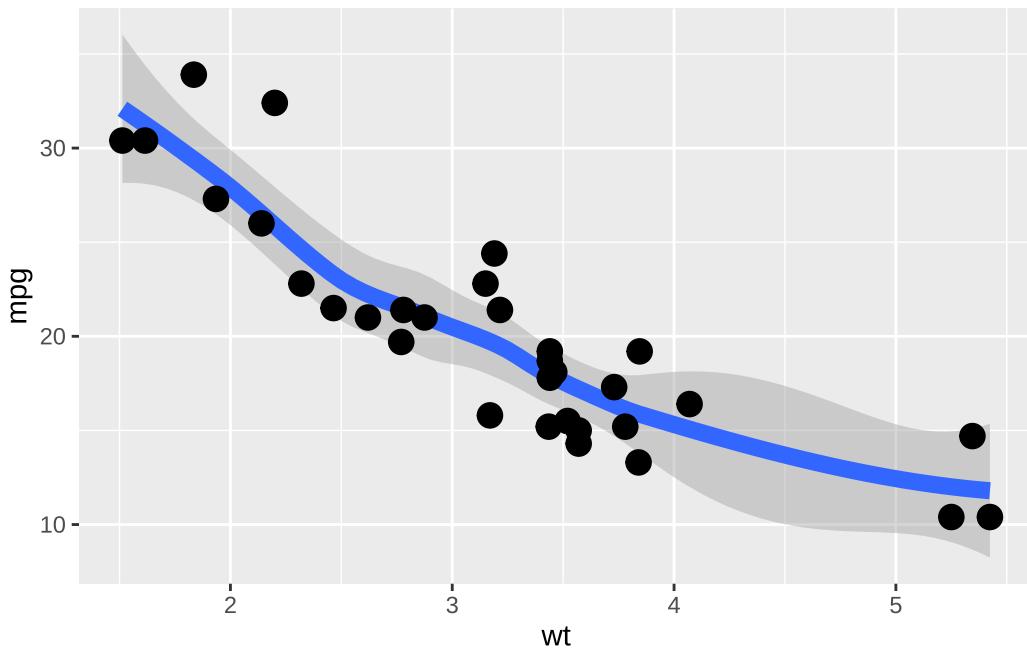
```
ggplot(data=mtcars,aes(wt, mpg))+  
  geom_point(size=4)+  
  geom_smooth(linewidth=3)+ # line overlaps points  
  geom_smooth(data=mtcars |> filter(wt>3), #picks a sub-sample  
              method="lm", linewidth=3, color="red")
```

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



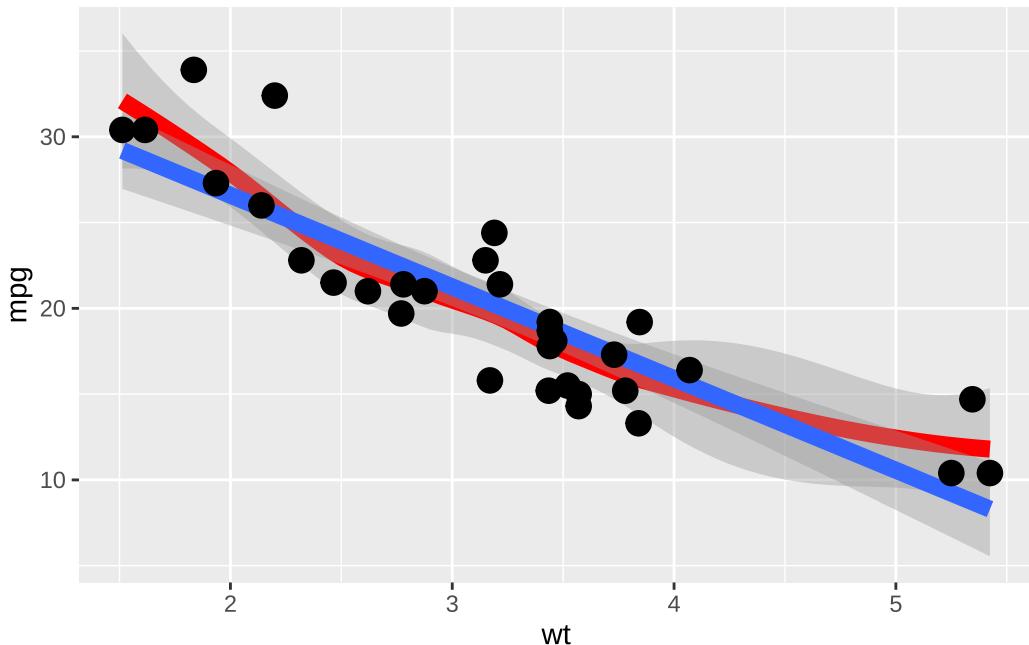
```
ggplot(data=mtcars,aes(wt, mpg))+  
  geom_smooth(linewidth=3)+  
  geom_point(size=4)
```

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



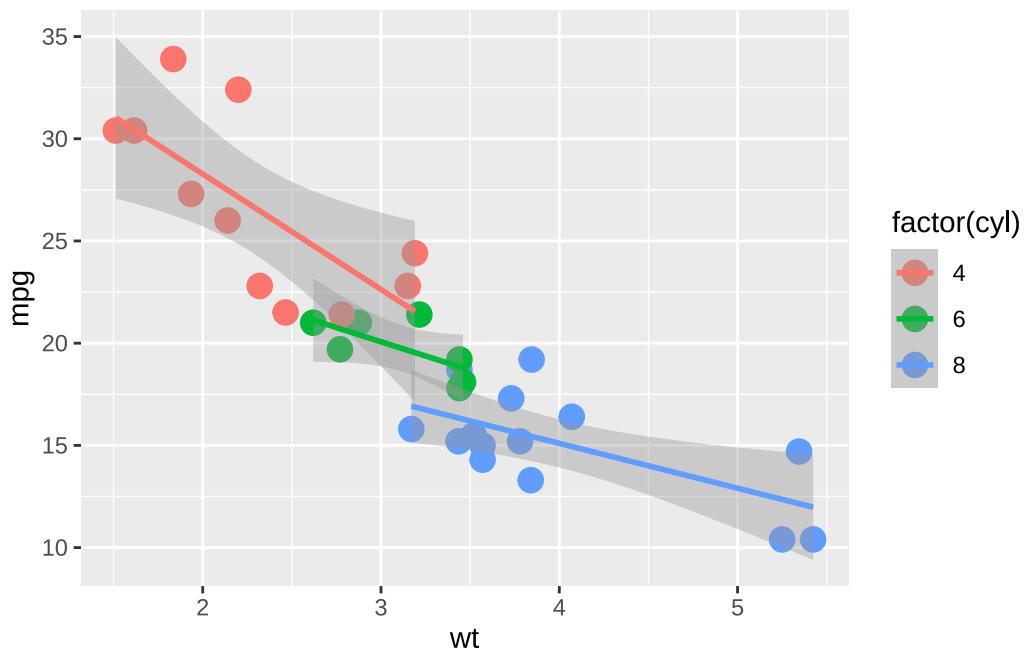
```
ggplot(data=mtcars,aes(wt, mpg))+  
  geom_smooth(linewidth=3,color="red") +  
  geom_smooth(method="lm", linewidth=3) +  
  geom_point(size=4)
```

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

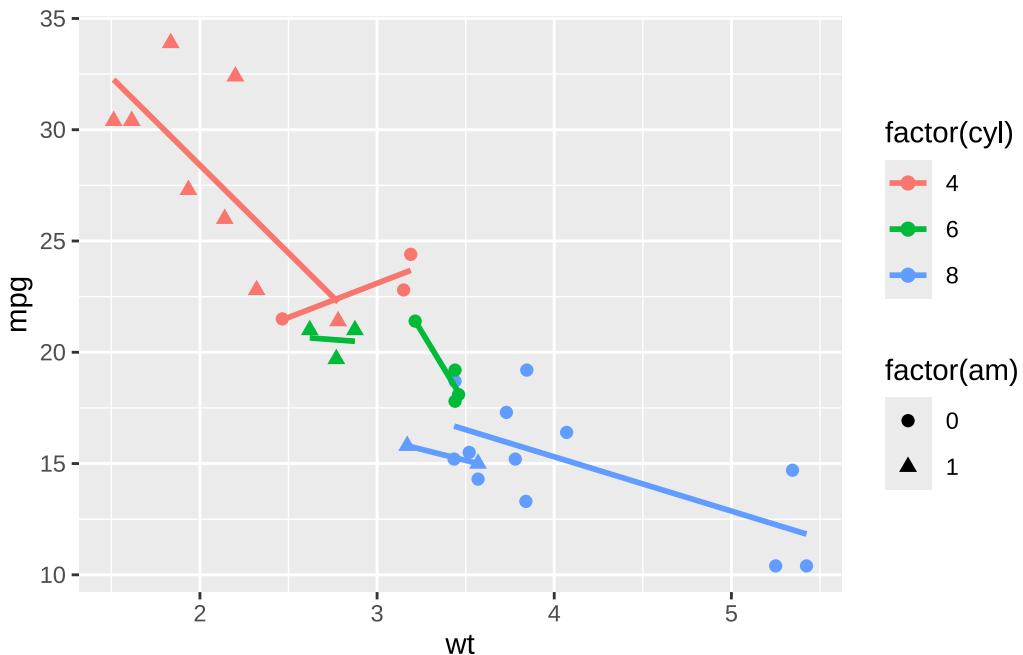


```
ggplot(data=mtcars,aes(wt, mpg,  
                      color=factor(cyl)))+  
  geom_point(size=4)+  
  geom_smooth(method="lm", linewidth=1)
```

```
`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'
```

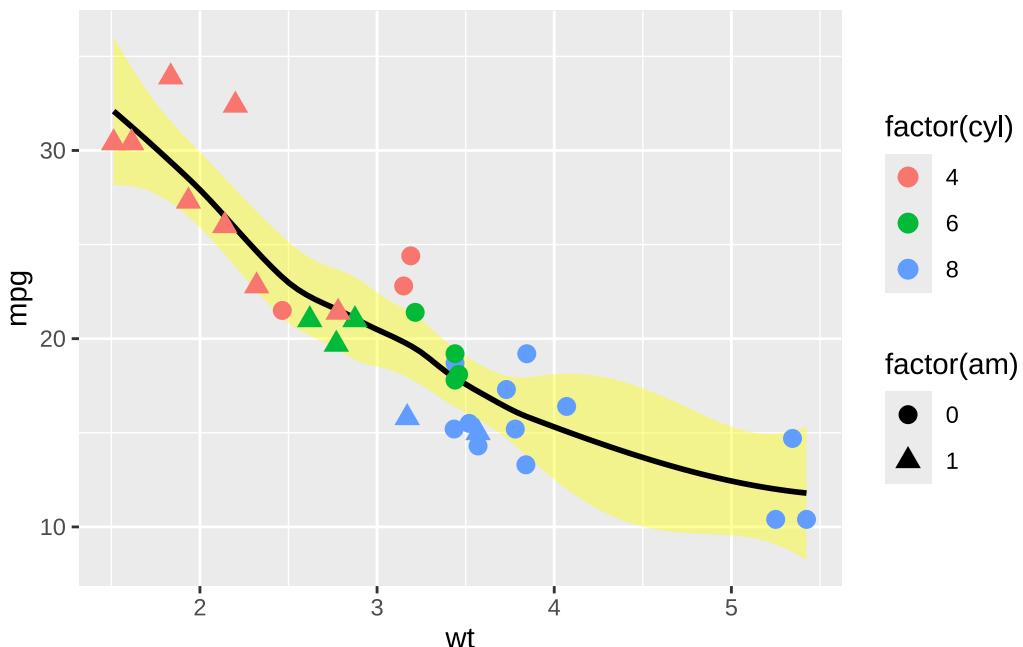


7.11 Local aesthetics for layers

```
#? 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
```

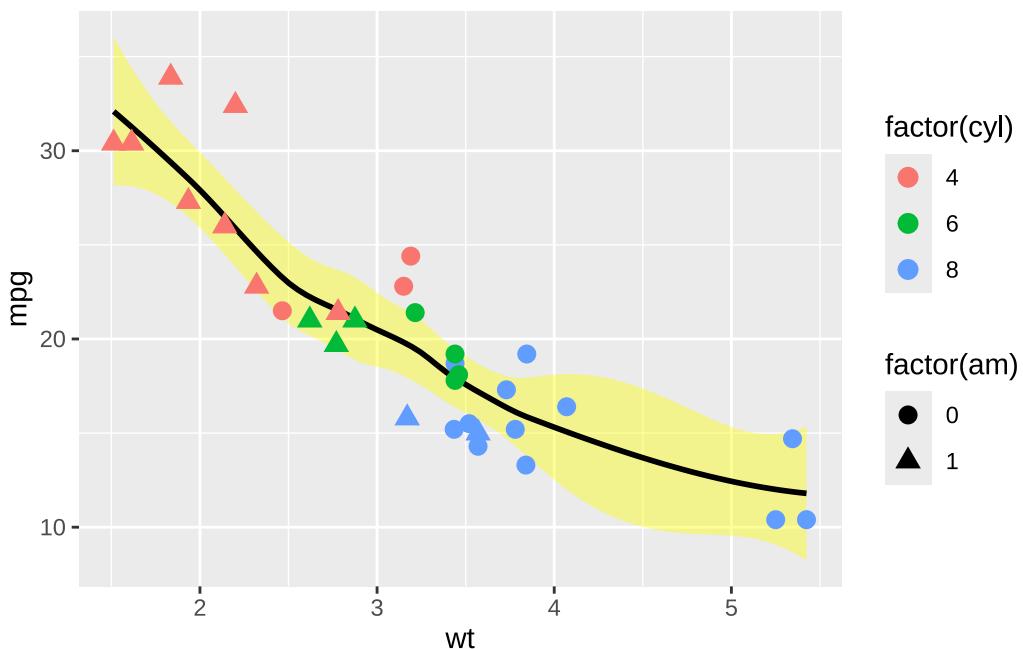
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.

`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).

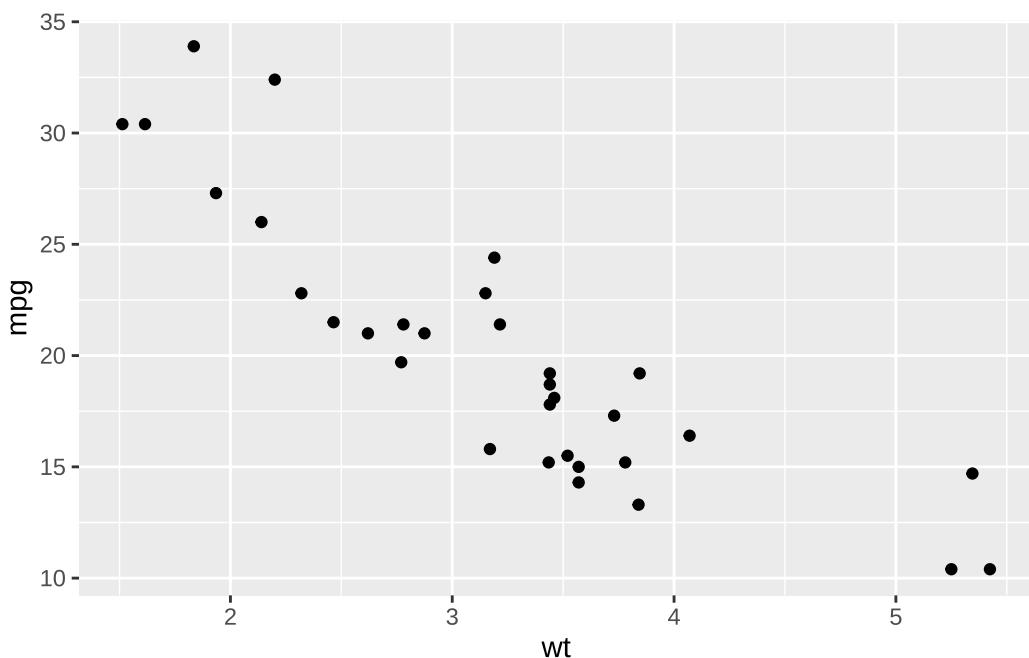
7.12.1 facet_grid

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

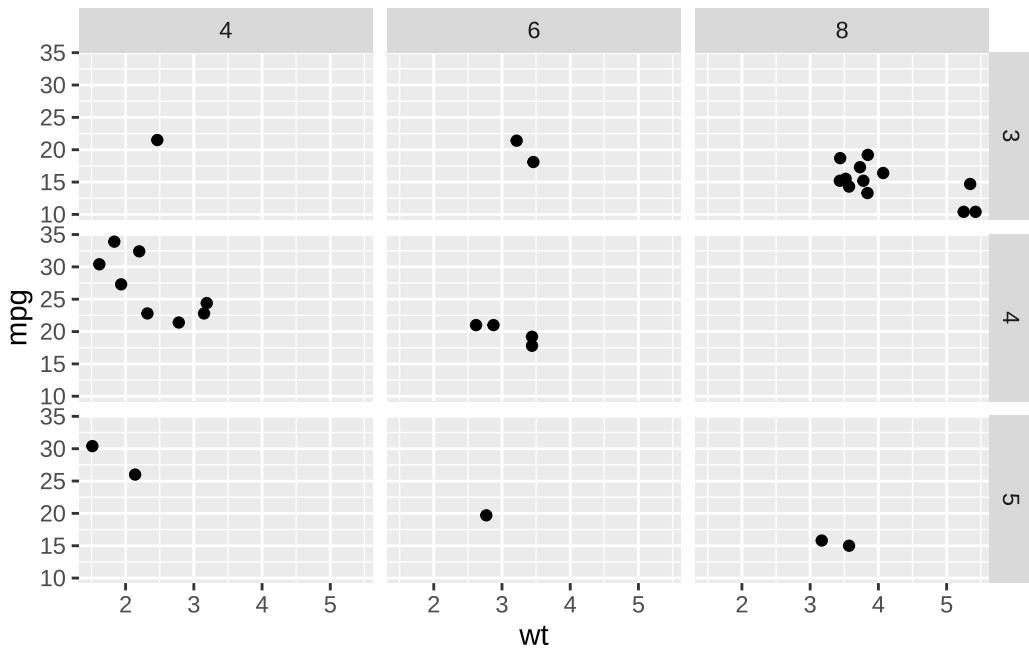
Labeling of facets often requires name and content to be informative.

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

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



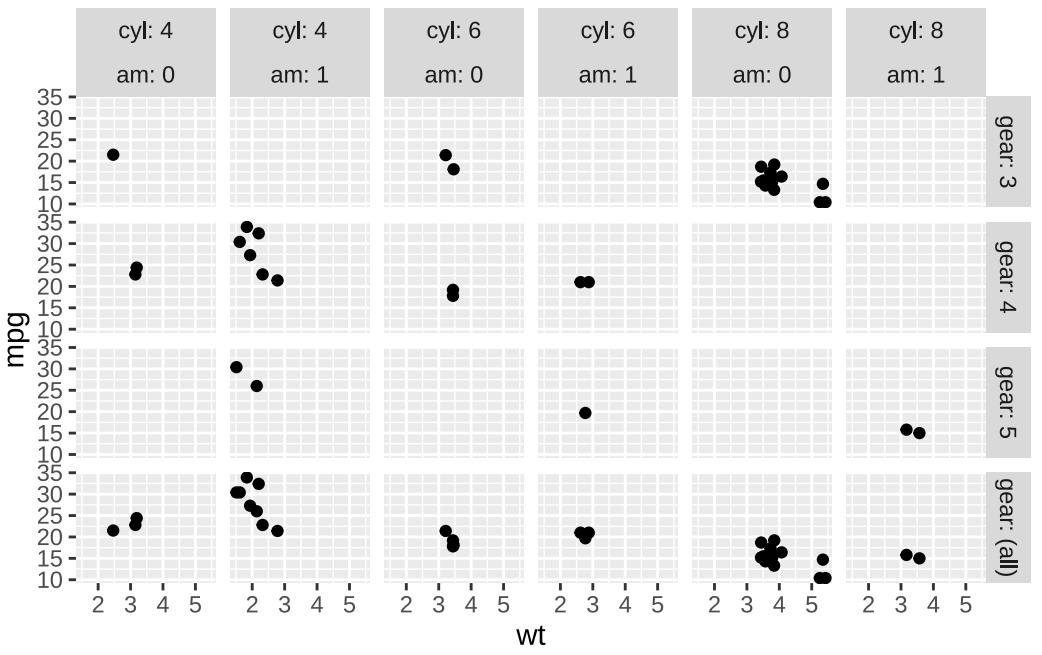
```
plot_tmp + facet_grid(rows = vars(gear),  
                      cols = vars(cyl))
```



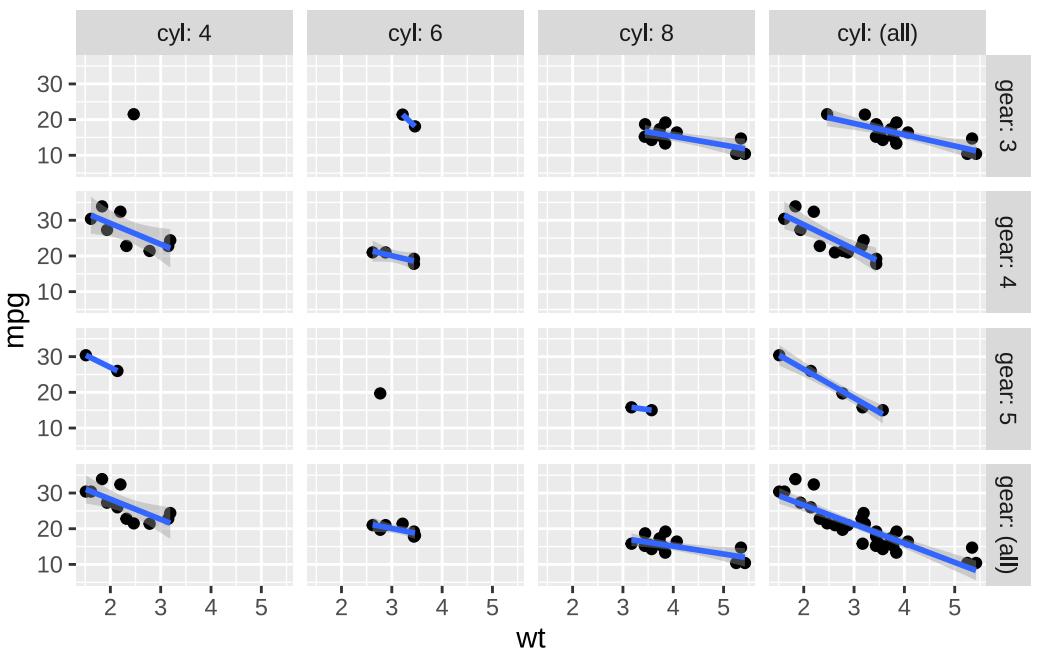
```
cat("facet labeling improved:\n")
```

facet labeling improved:

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

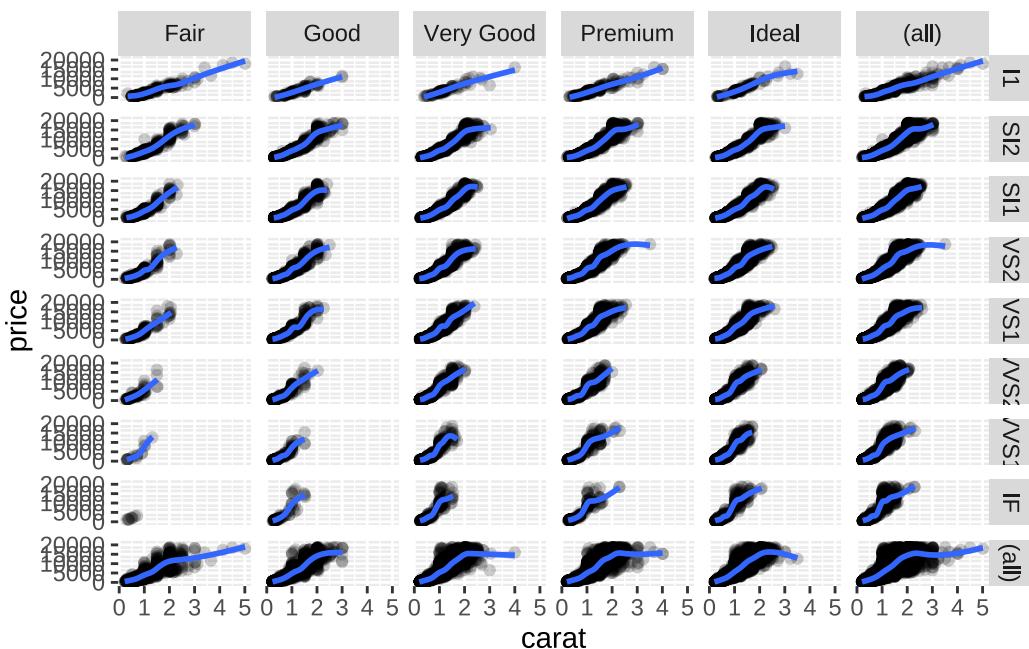


```
# options(warn=-1)
plot_tmp + geom_smooth(method="lm")+
  facet_grid(rows = vars(gear),
             cols = vars(cyl),
             labeller=label_both, margins=TRUE)
```



```
# options(warn=0)

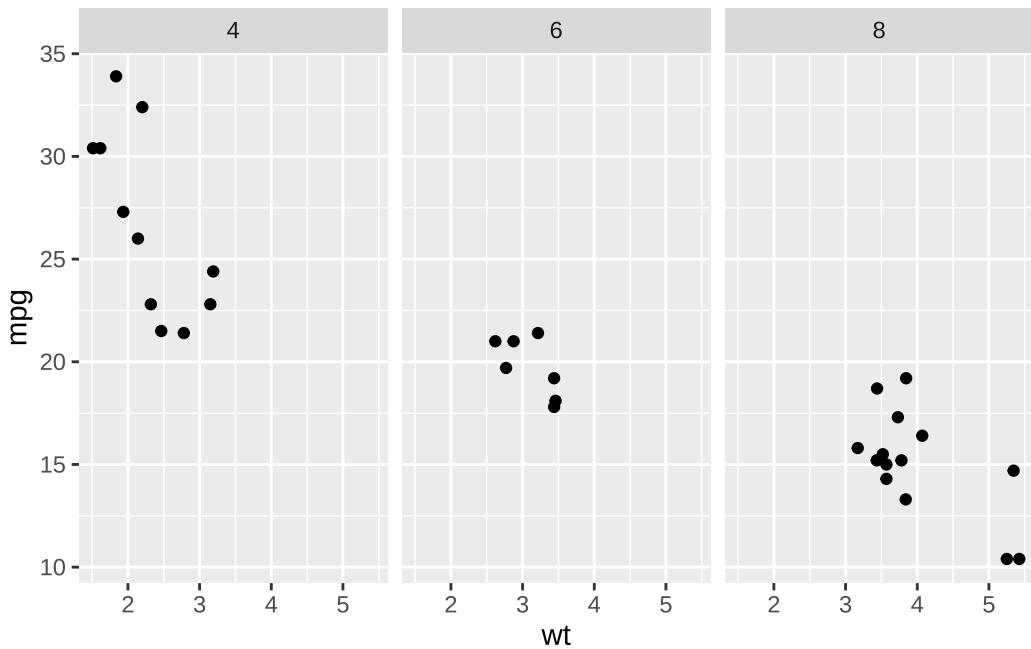
ggplot(diamonds,aes(carat,price))+
  geom_point(alpha=.2)+
  geom_smooth()+
  facet_grid(rows = vars(clarity),
             cols = vars(cut),
             margins=TRUE)
```



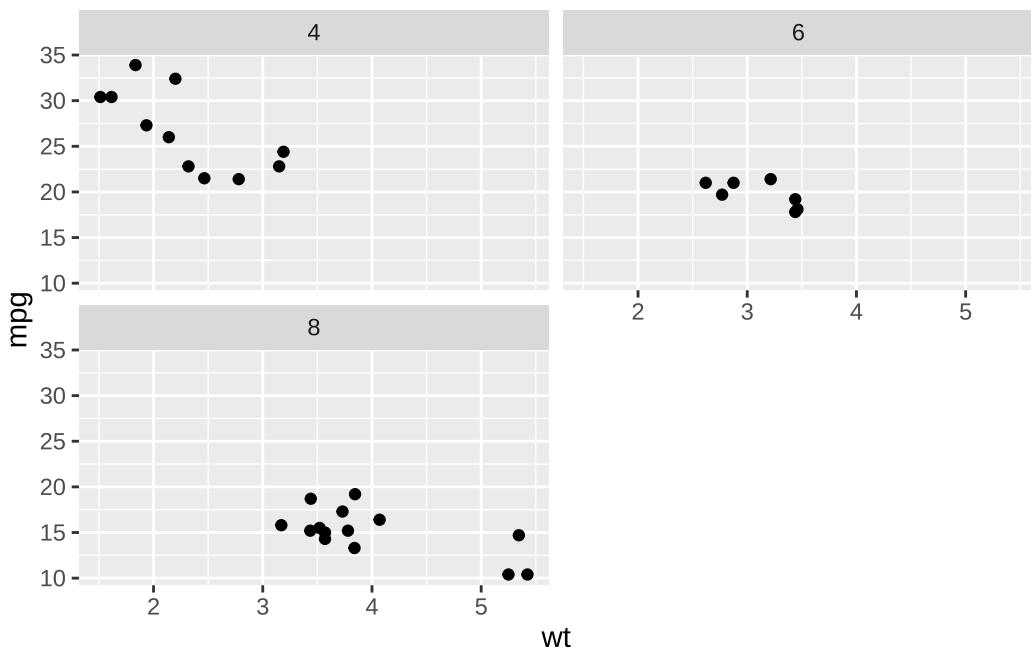
7.12.2 `facet_wrap`

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

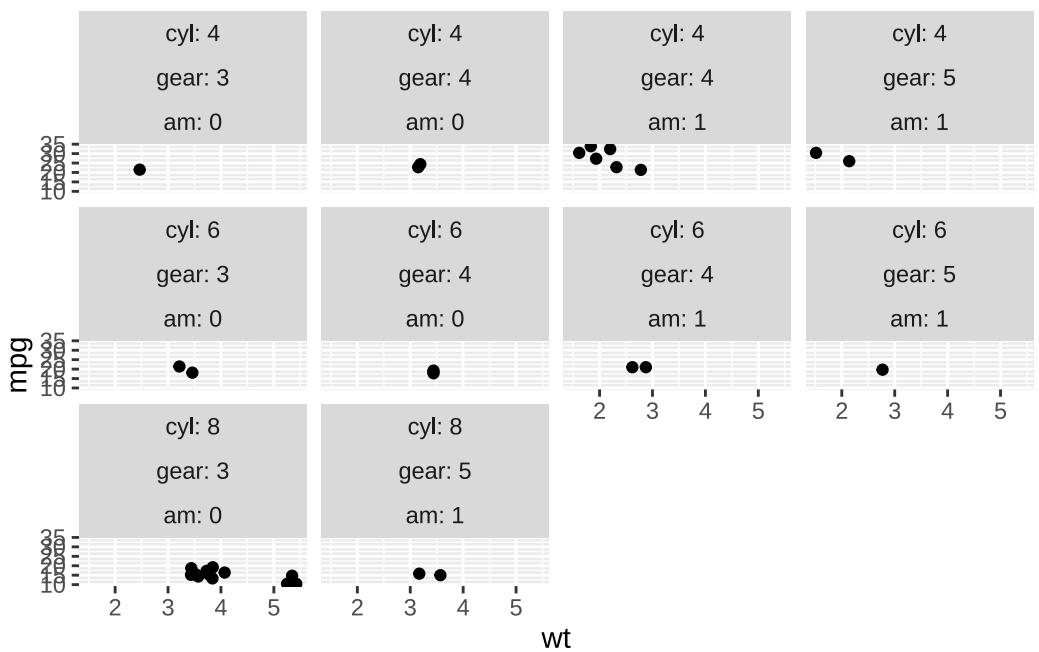
```
plot_tmp + facet_wrap(facets = vars(cyl))
```



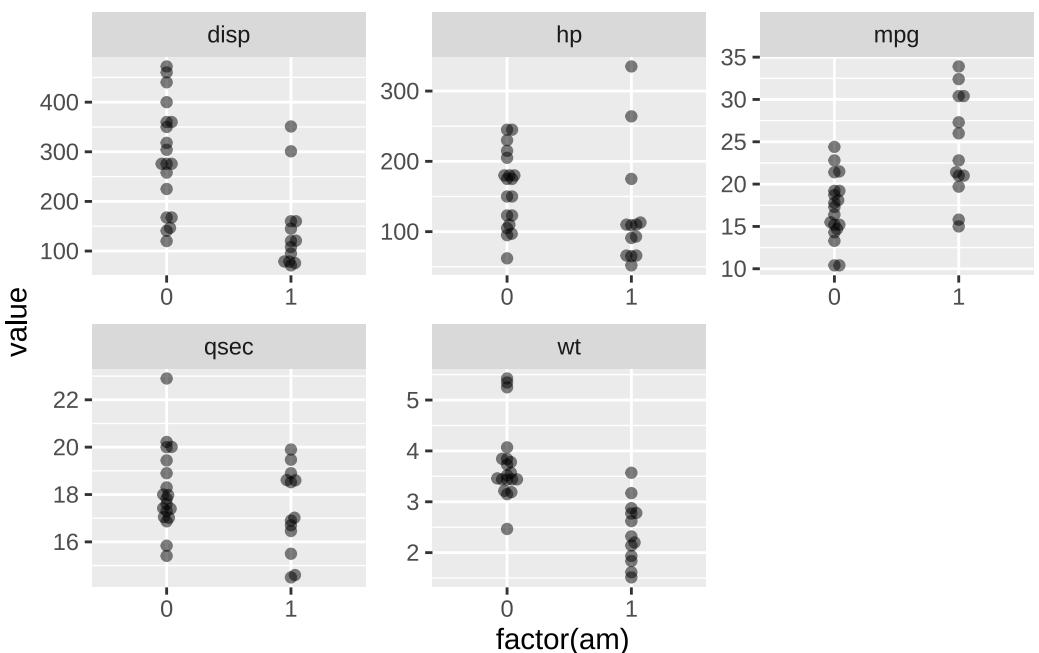
```
plot_tmp + facet_wrap(facets = vars(cyl), ncol=2)
```



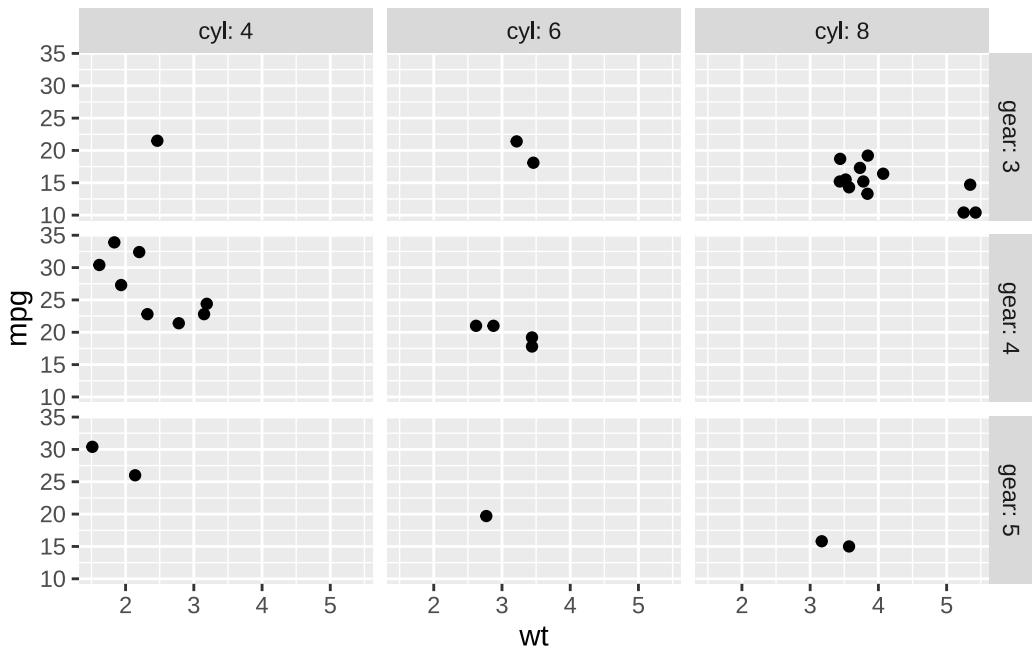
```
# empty combination is dropped  
plot_tmp + facet_wrap(facets=vars(cyl,gear,am),labeler=label_both)
```

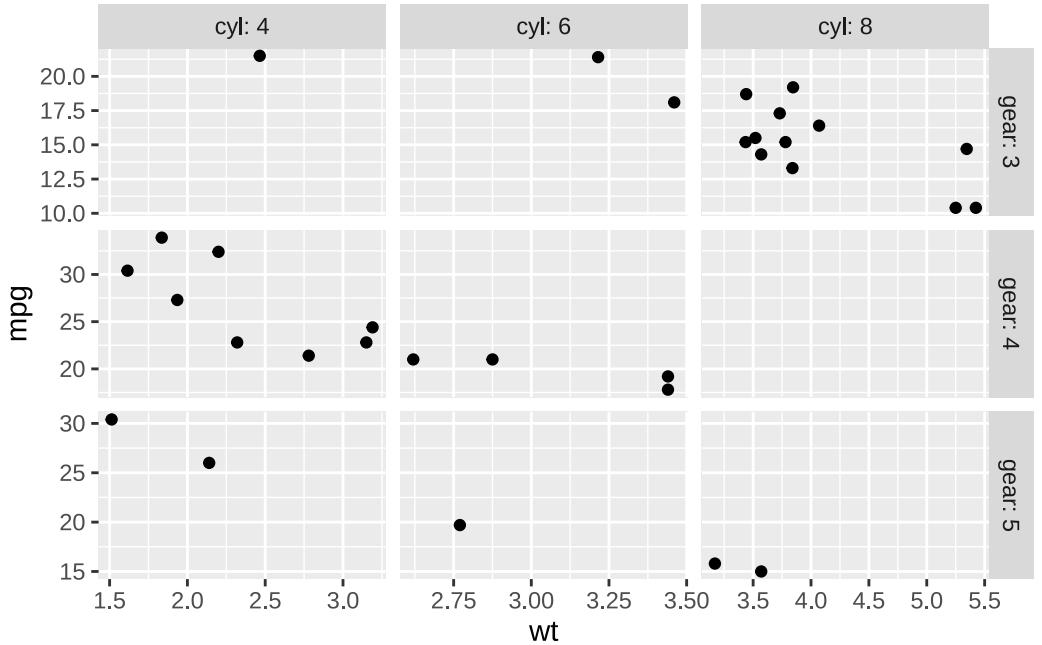


```
#combining variables
mtcars |>
  pivot_longer(cols = c(wt, mpg, hp, disp, qsec)) |> #view()
  ggplot(aes(x=factor(am), y=value))+
  geom_beeswarm(alpha=.5, cex=2)+
  facet_wrap(facets = vars(name), scales="free")
```

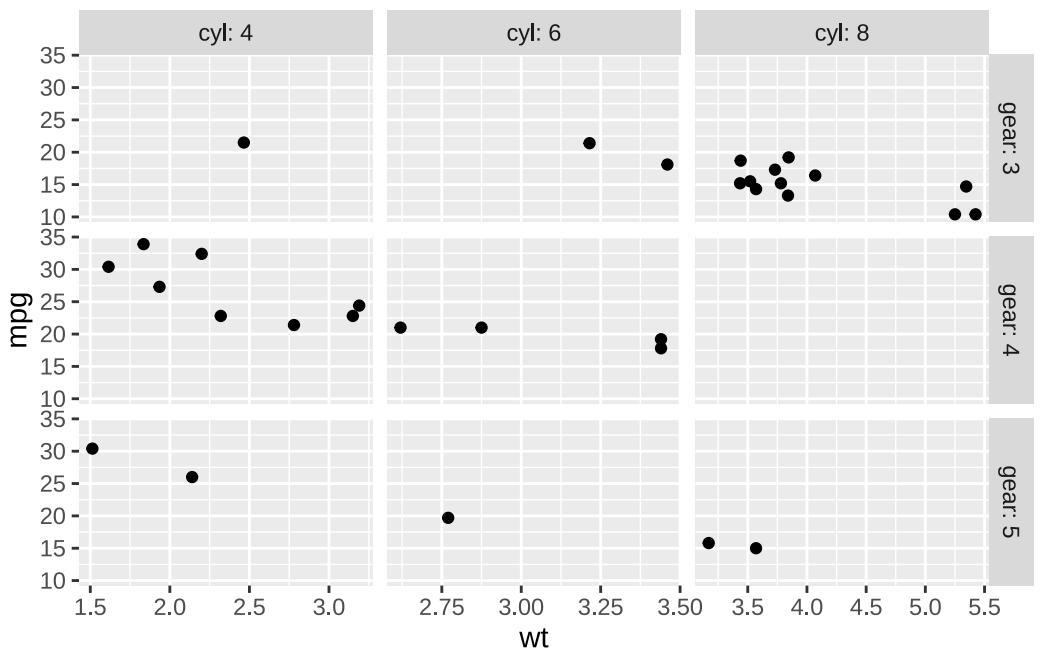


7.12.3 Controlling scales in facets (default: `scales="fixed"`)

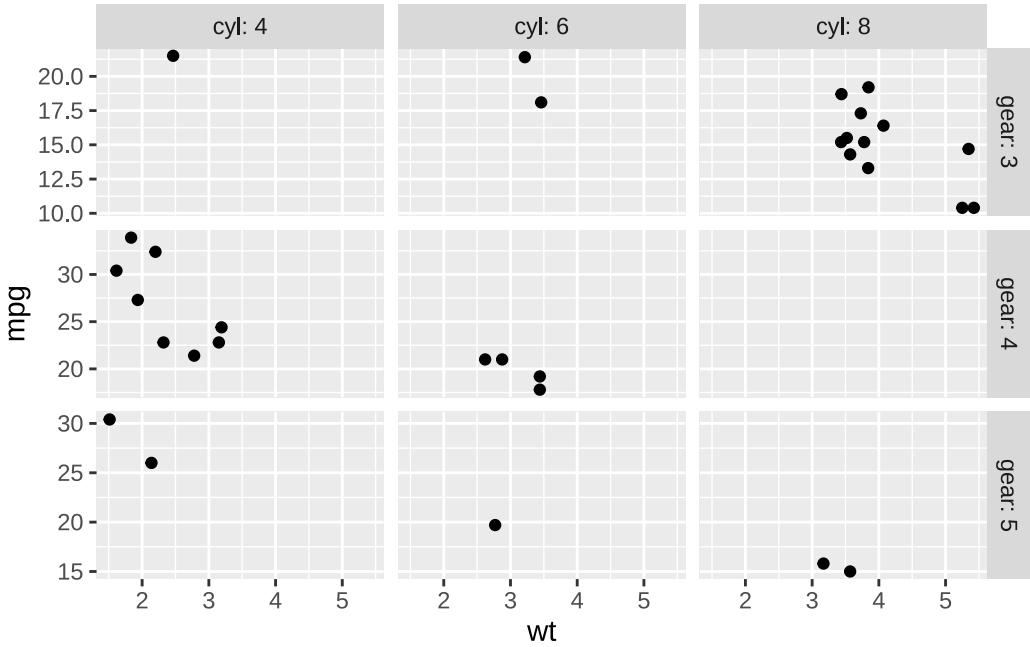




```
plot_tmp + facet_grid(rows=vars(gear),cols=vars(cyl),
                      labeller=label_both, scales="free_x")
```



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

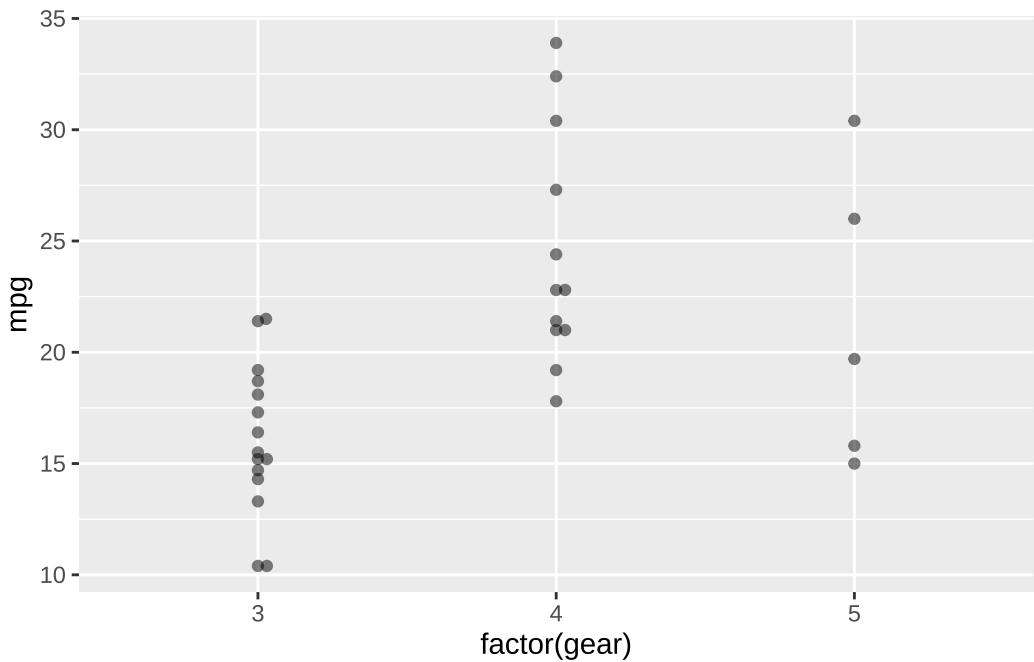


7.13 Showing summaries

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!!

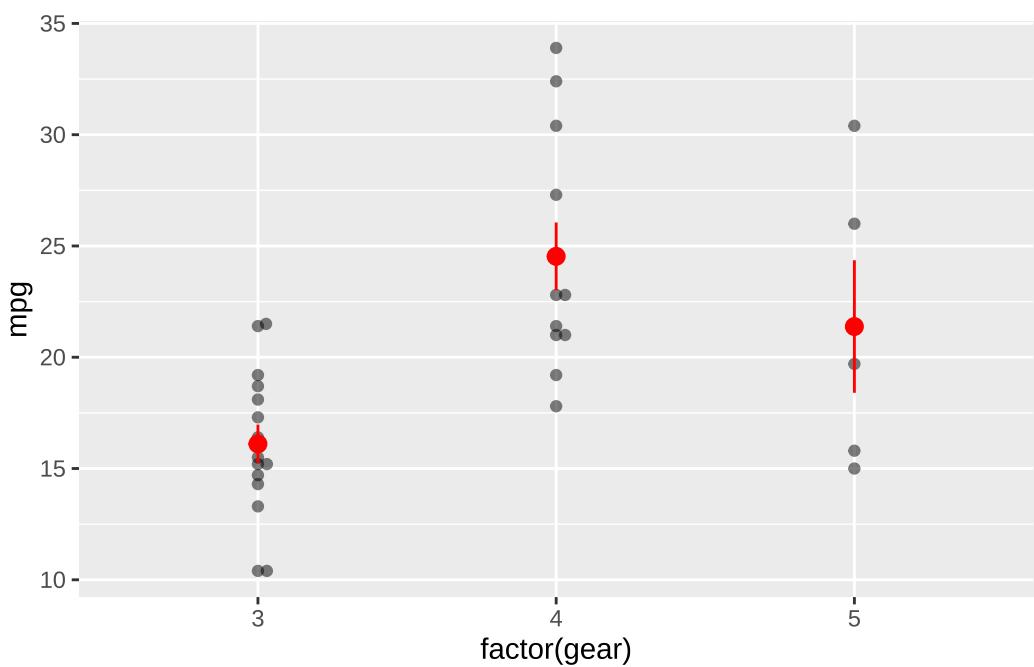
Functions for summary statistics (mean_se, mean_cl_normal, mean_cl_boot etc.) are build on top of Hmisc functions. So this package is needed but not automatically installed with ggplot2.

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

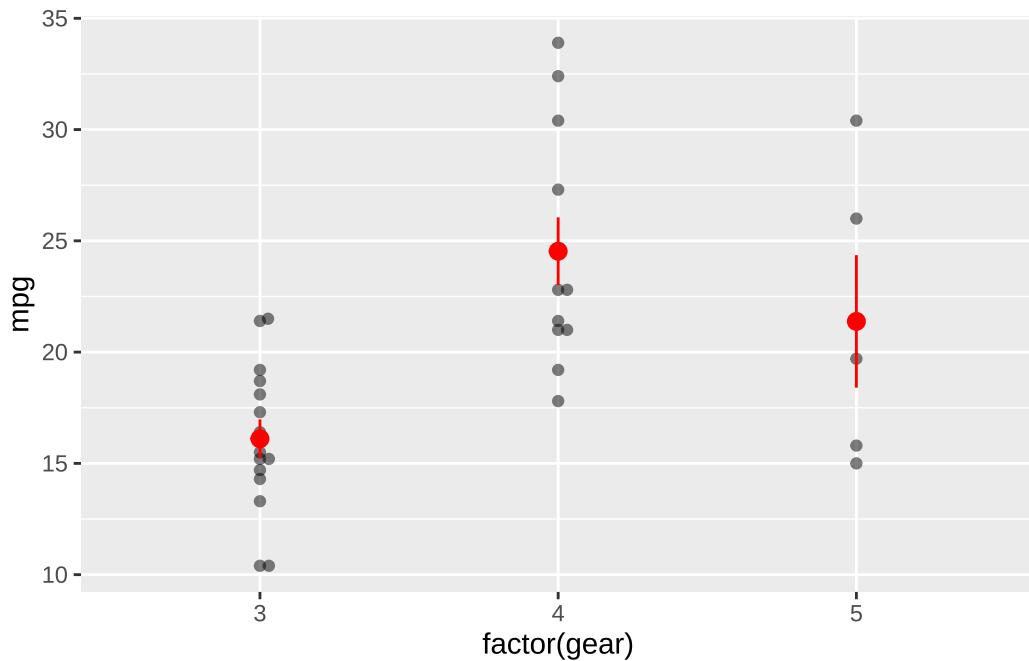


```
plottemp+stat_summary(color="red")
```

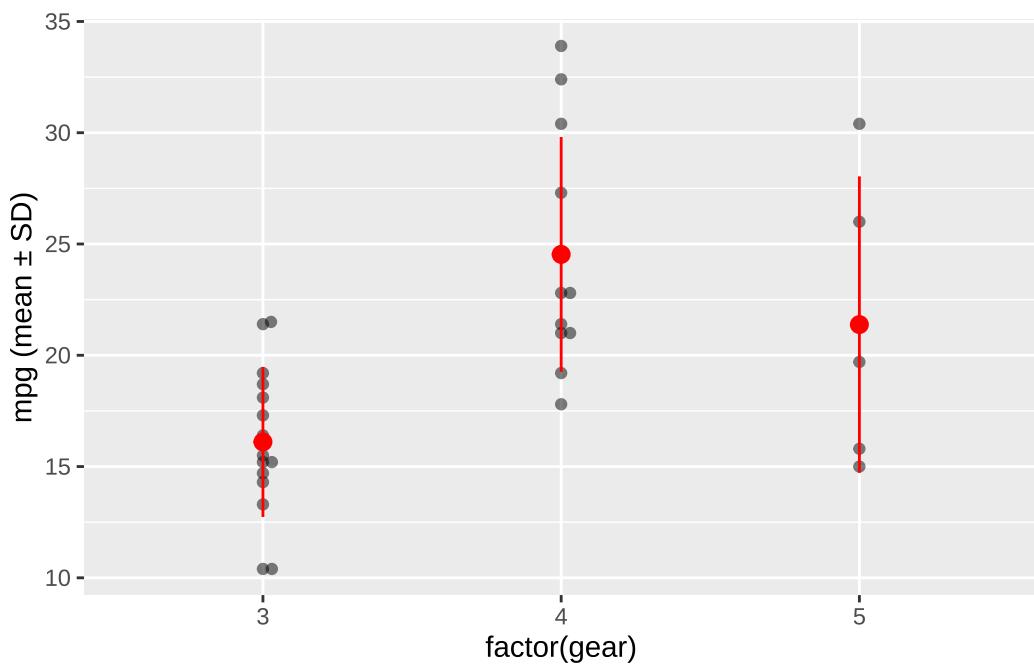
No summary function supplied, defaulting to `mean_se()`



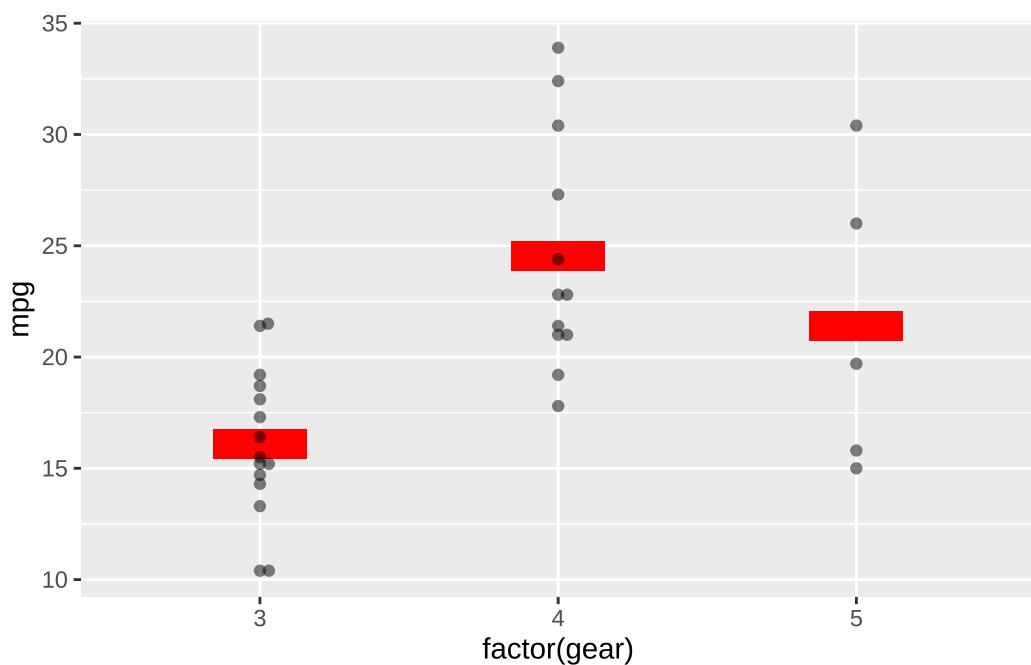
```
plottemp+stat_summary(fun.data="mean_se",
                      color="red")
```



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



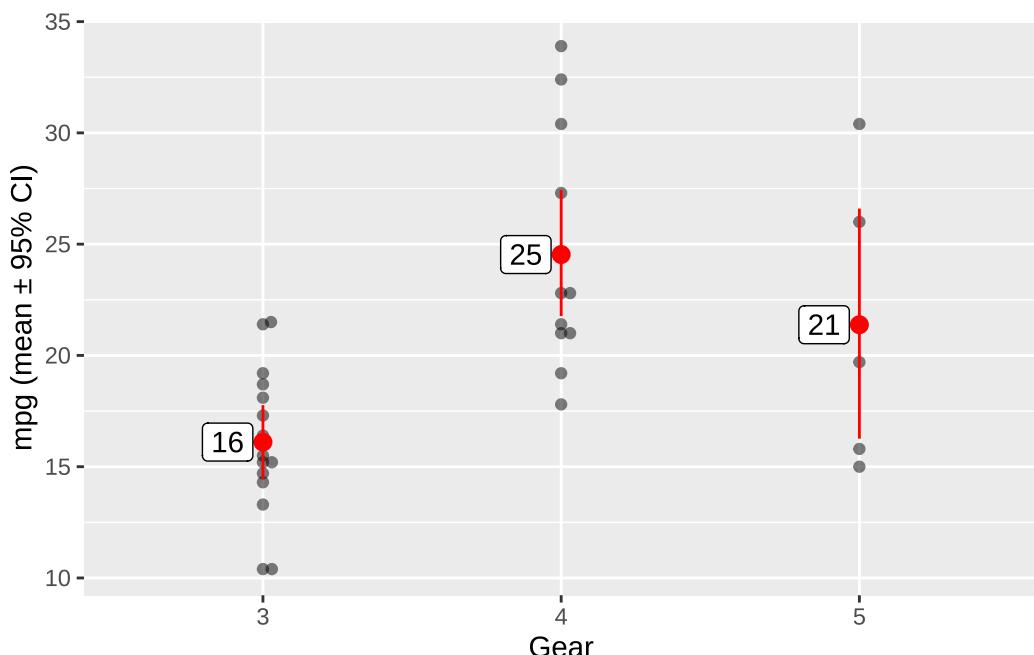
```
ggplot(mtcars,aes(factor(gear),mpg))+  
  stat_summary(geom = "point", shape="-",  
               size=50,  
               fun = "mean",color="red") +  
  geom_beeswarm(alpha=.5)
```



```

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")

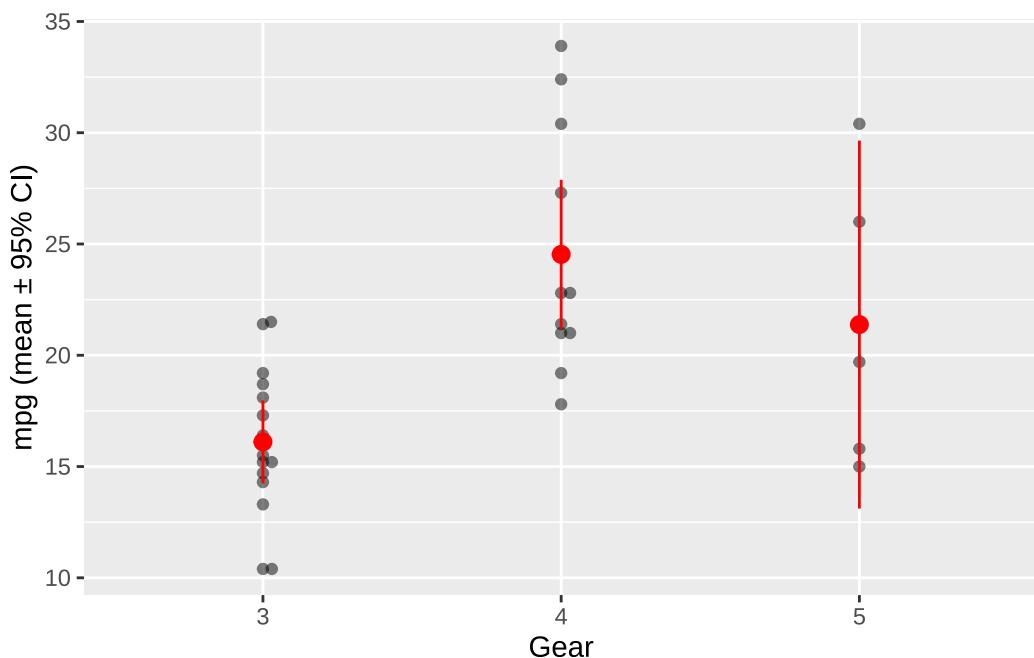
```



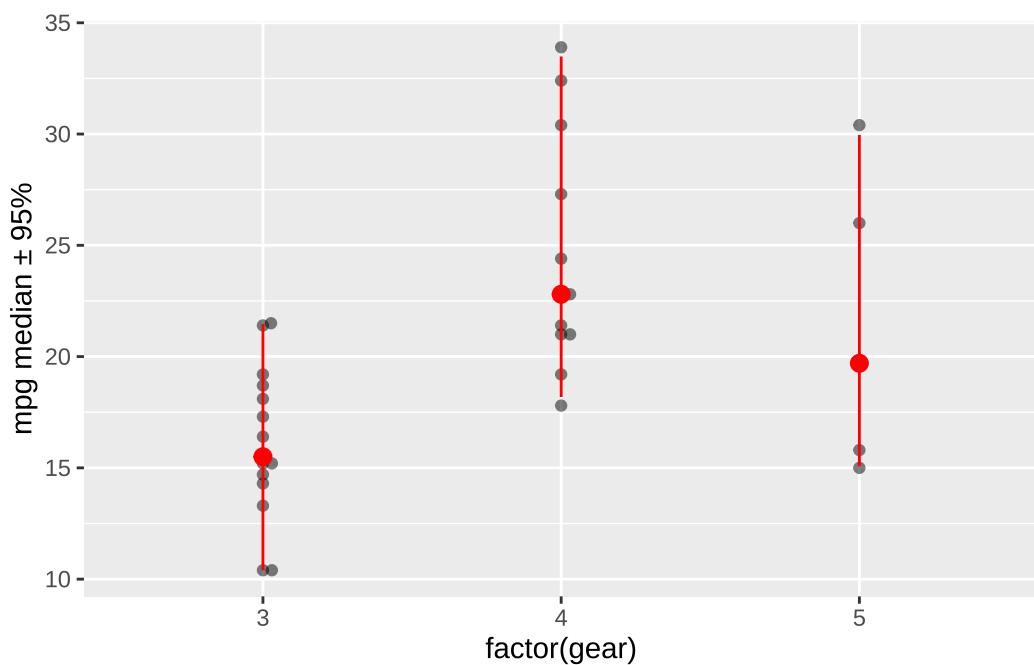
```

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

```

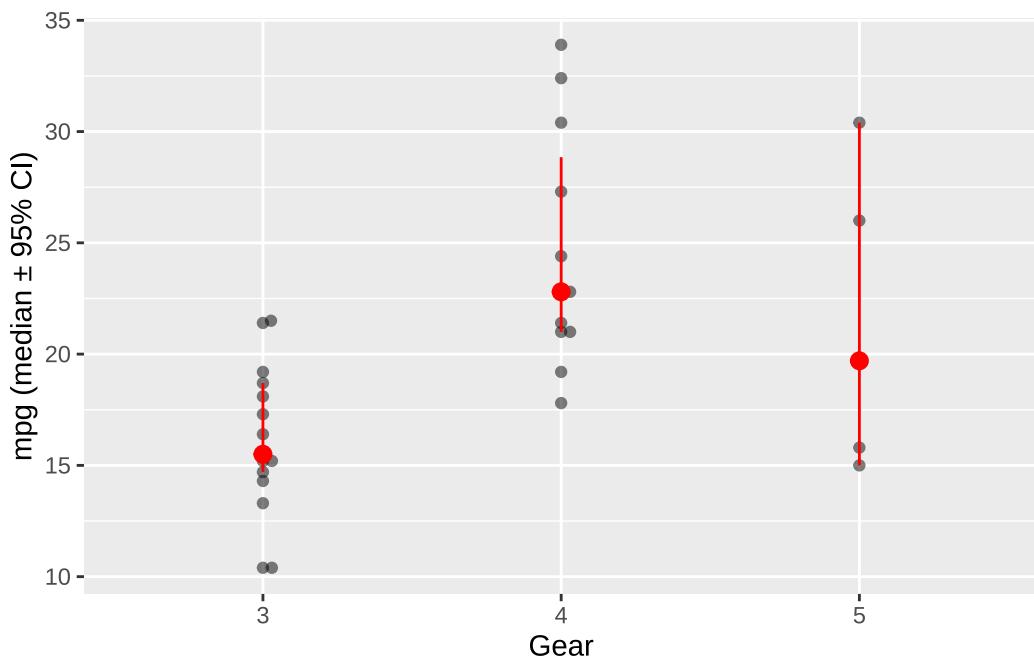


```
plottemp+stat_summary(fun.data="median_hilow",color="red")+
  ylab("mpg median \u00b1 95%")
```



```
# geom_pointrange()
plottemp+stat_summary(fun.data="median_cl_boot_gg",color="red")+
```

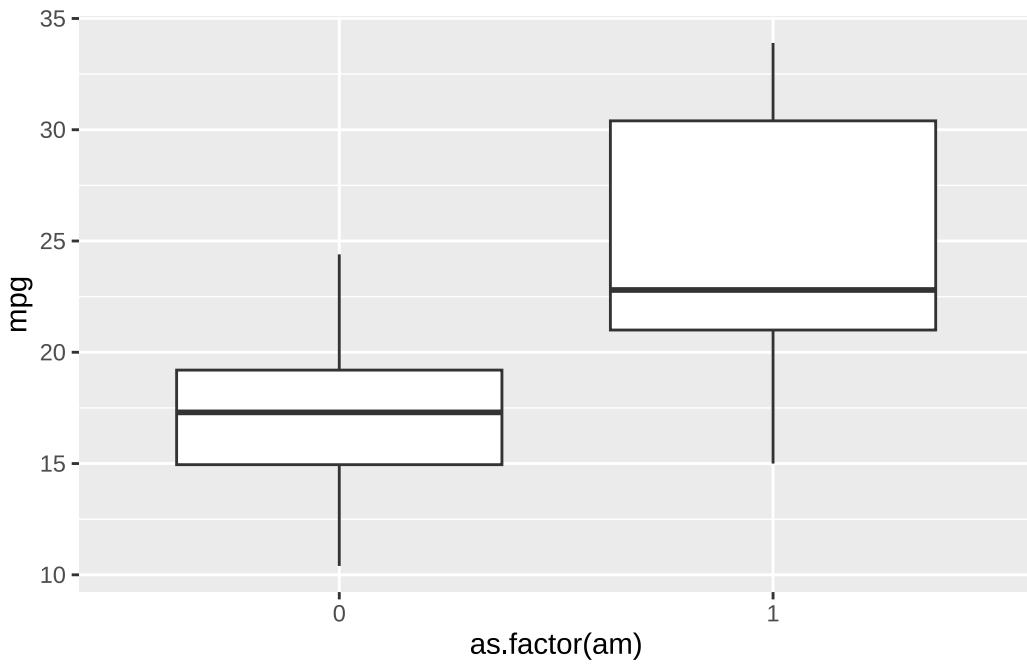
```
ylab("mpg (median \u00b1 95% CI)")+
xlab("Gear")
```



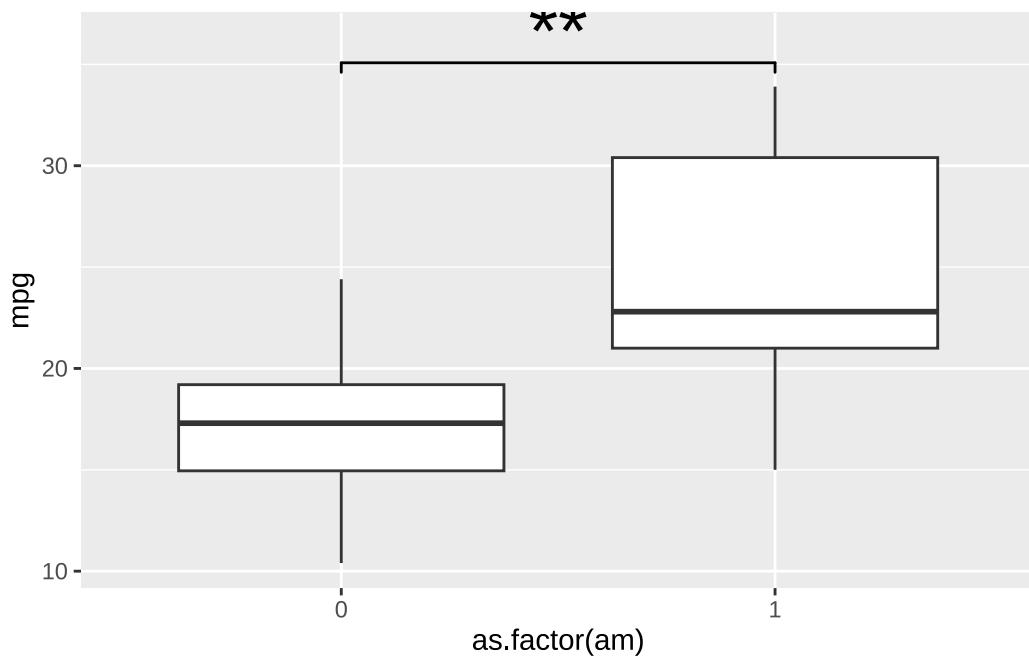
7.14 Indicating significances

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())
```

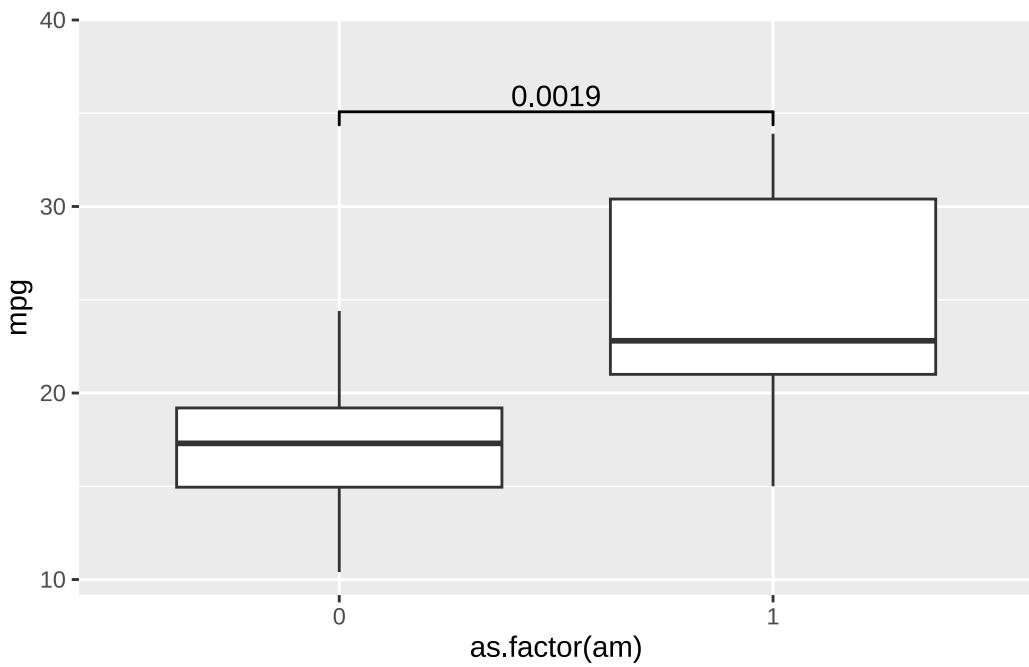


```
plottemp+geom_signif(  
  comparisons=list(c(1,2)),  
  # aes(y=0),  
  textsize = rel(10), vjust = .0,  
  #y_position=max(mtcars$mpg+3),  
  # annotations=paste0("p = ", p),  
  annotations=markSign(p),  
  # annotations=p,  
  tip_length=.02)+  
  scale_y_continuous(expand = expansion(mult=c(0.05,.1)))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

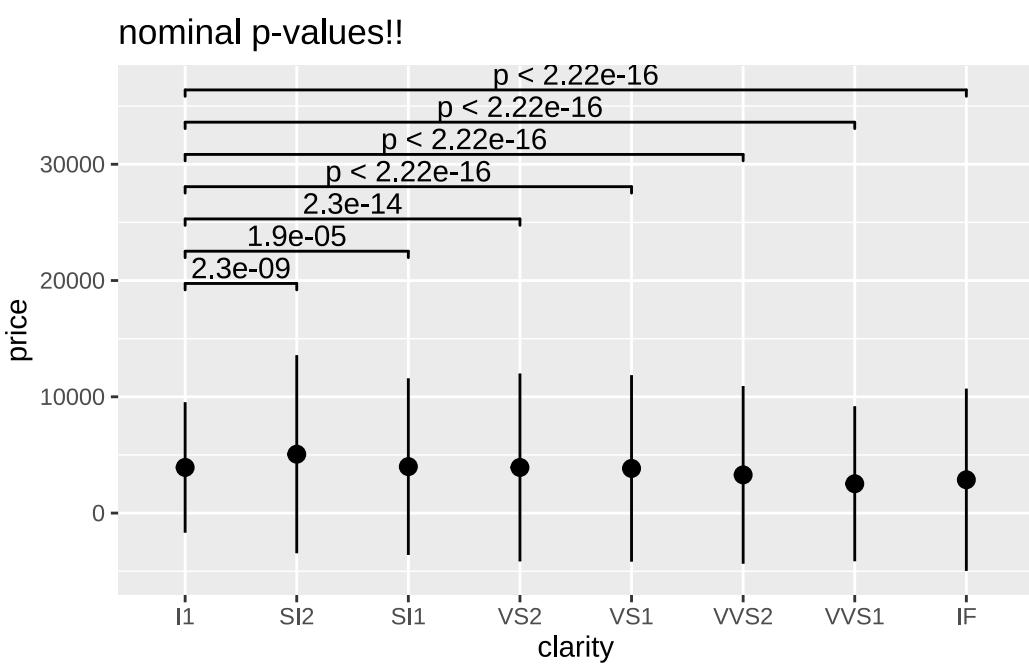


```
plottemp + geom_signif(  
  comparisons=list(1:2))+  
  scale_y_continuous(expand = expansion(mult=c(0.05,.2)))
```

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))+
  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!!")
```



```

ggplot(diamonds,aes(clarity, price))+  

  stat_summary(fun.data=mean_sdl)+  

  geom_signif(comparisons=list(c(1,2),c(2,3),c(3,4),c(4,5),  

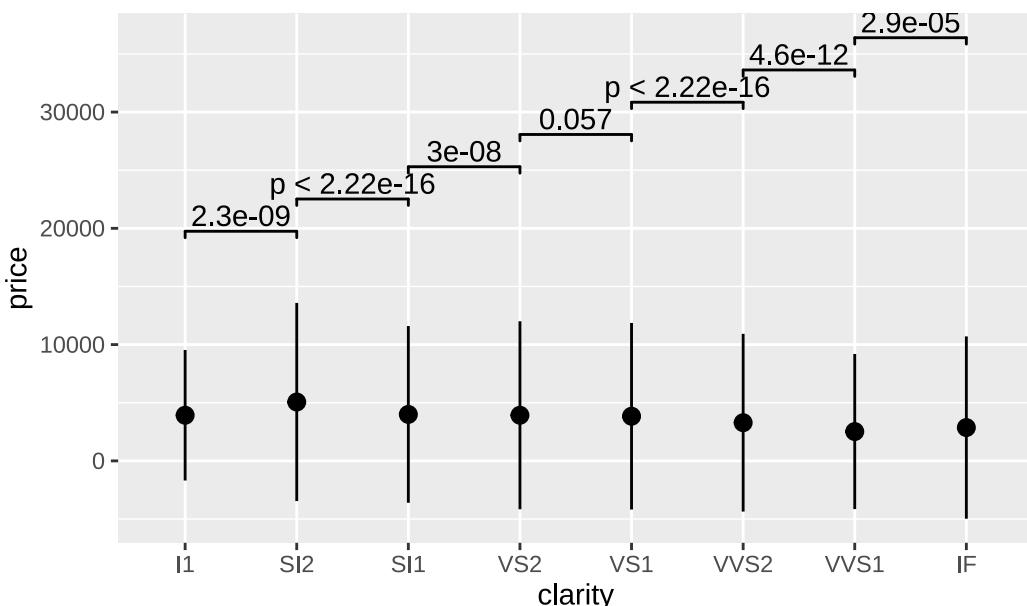
                               c(5,6),c(6,7),c(7,8)),  

              step_increase = 0.15)+  

  ggtitle("nominal p-values!!")

```

nominal p-values!!



```

ggplot(diamonds,aes(clarity, price))+  

  stat_summary(fun.data=mean_sdl)+  

  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)+  

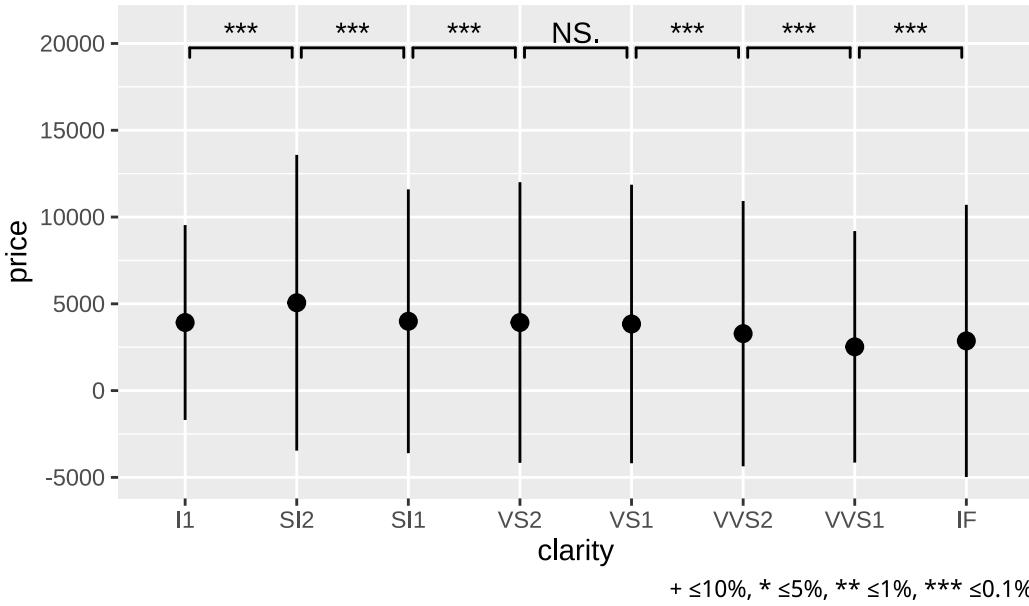
  ggtitle("nominal p-values!!") +  

  scale_y_continuous(expand = expansion(mult=c(0.05,.1)))+  

  labs(caption = "+ \u226410%, * \u22645%, ** \u22641%, *** \u22640.1%")

```

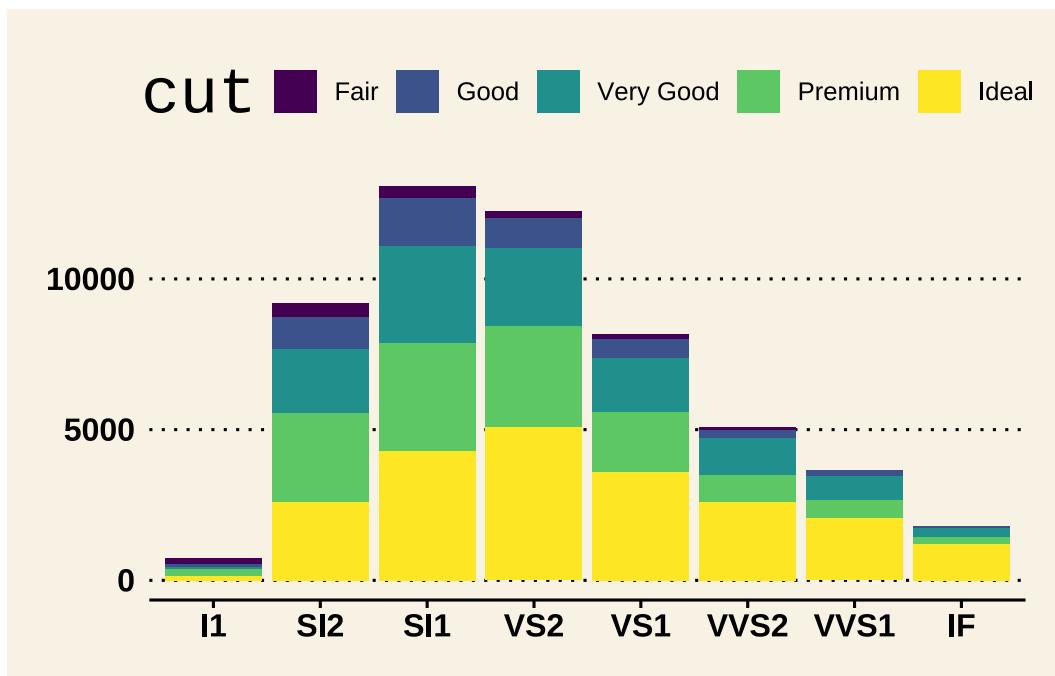
nominal p-values!!



7.15 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()`)

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

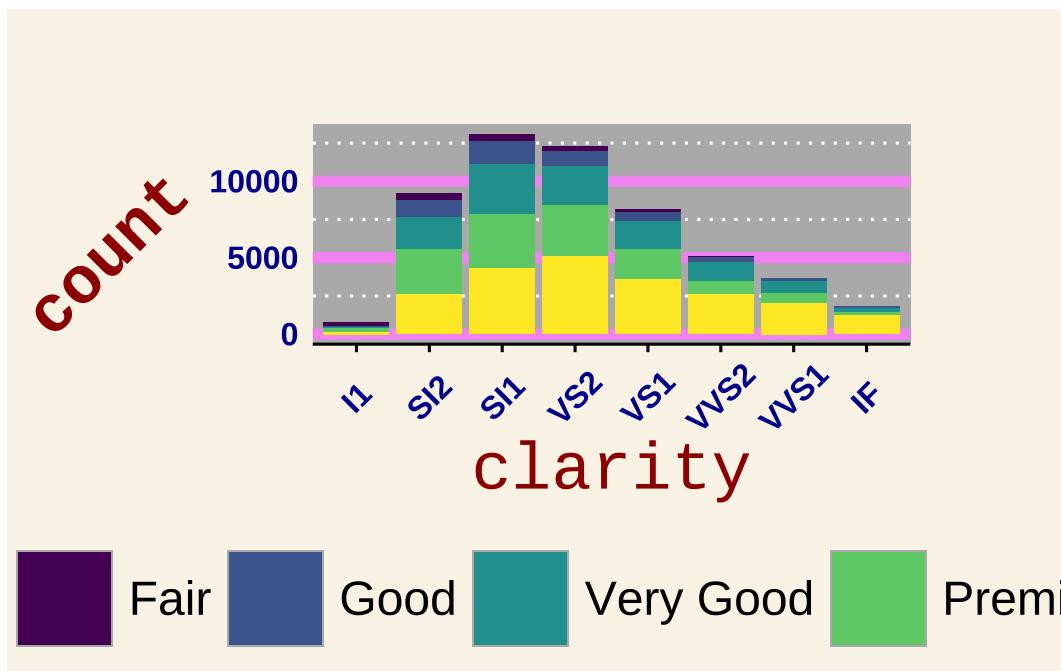


```

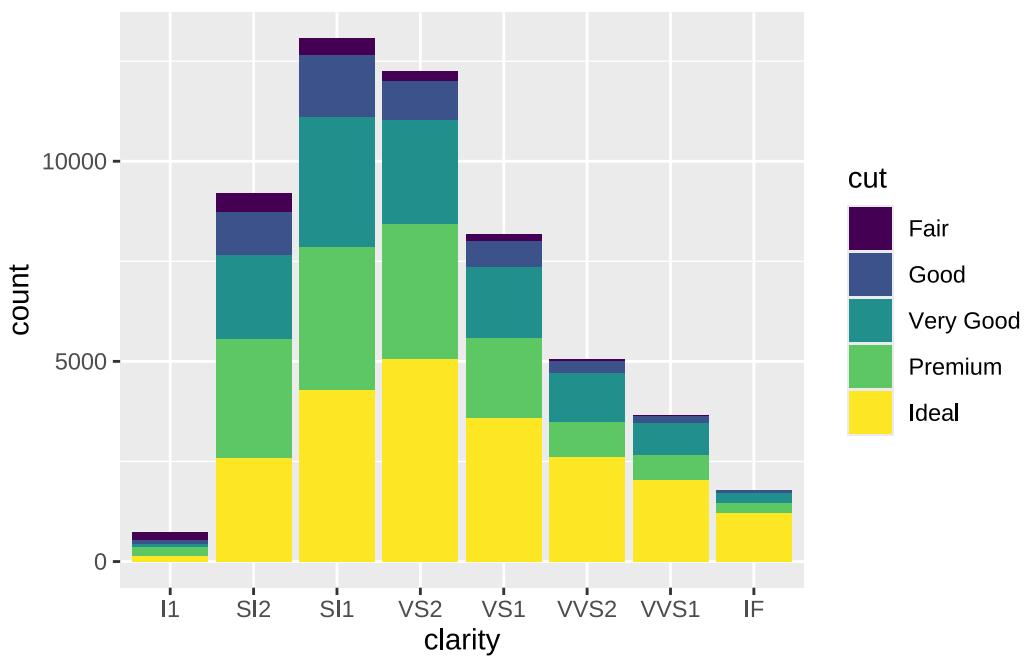
theme_update(legend.position="bottom",
             axis.text=element_text(colour = "darkblue",
                                    size=12),
             axis.text.x=element_text(vjust=0.5,angle=45,
                                      family="sans",
                                      face = "bold"),
             axis.title=element_text(size=25,
                                    color="darkred"),
             plot.margin=unit(c(3,4,.5,.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()

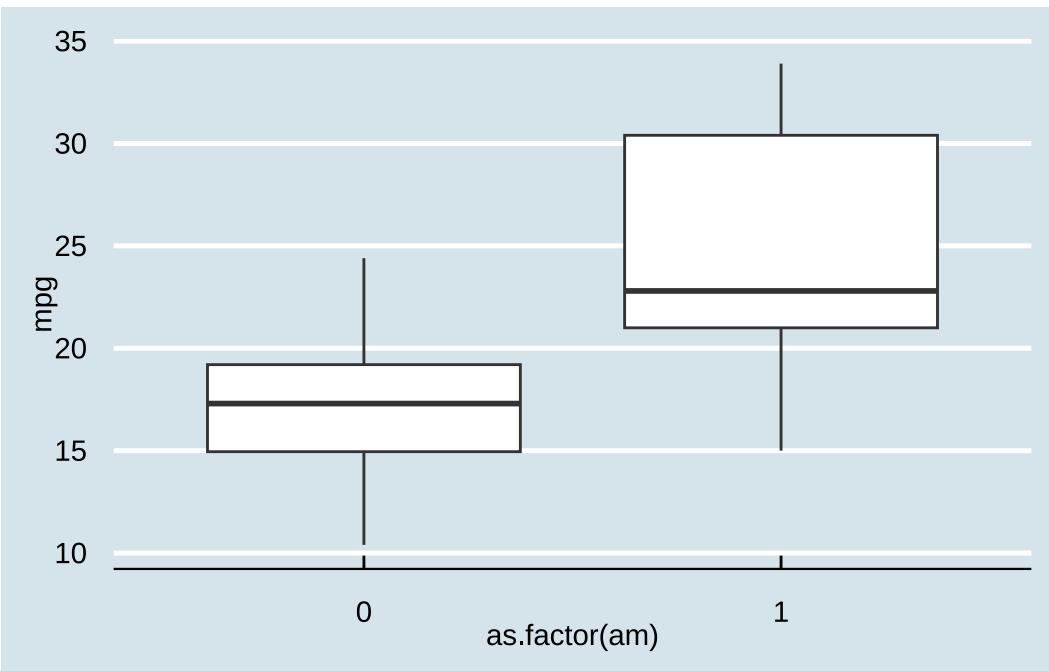
```



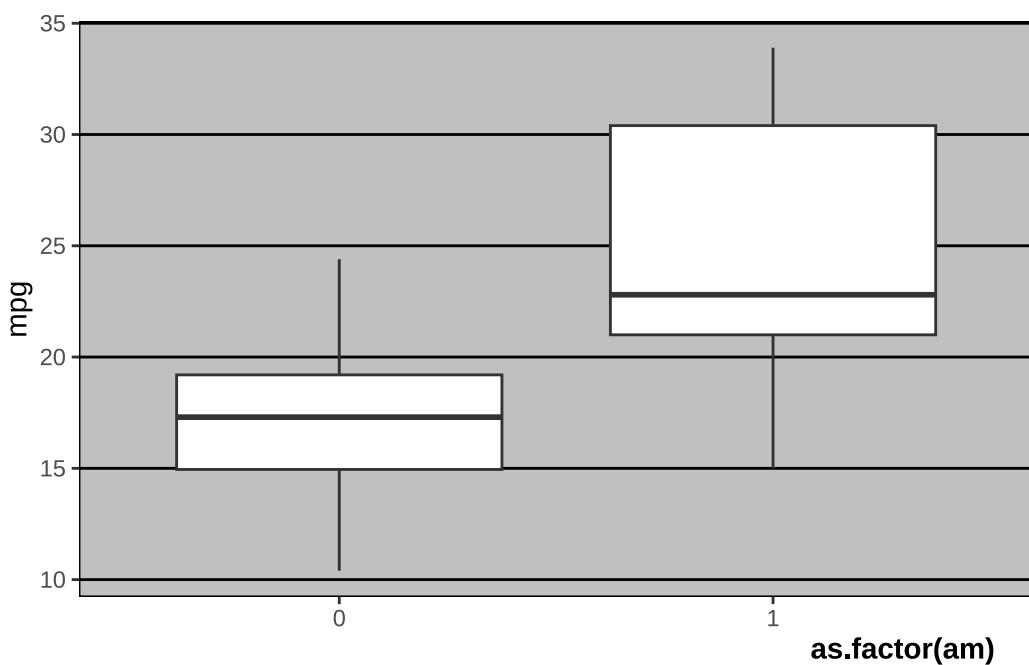
```
theme_set(theme_grey())
#theme_set(old)
ggplot(data=diamonds,aes(x=clarity,fill=cut))+  
  geom_bar()
```



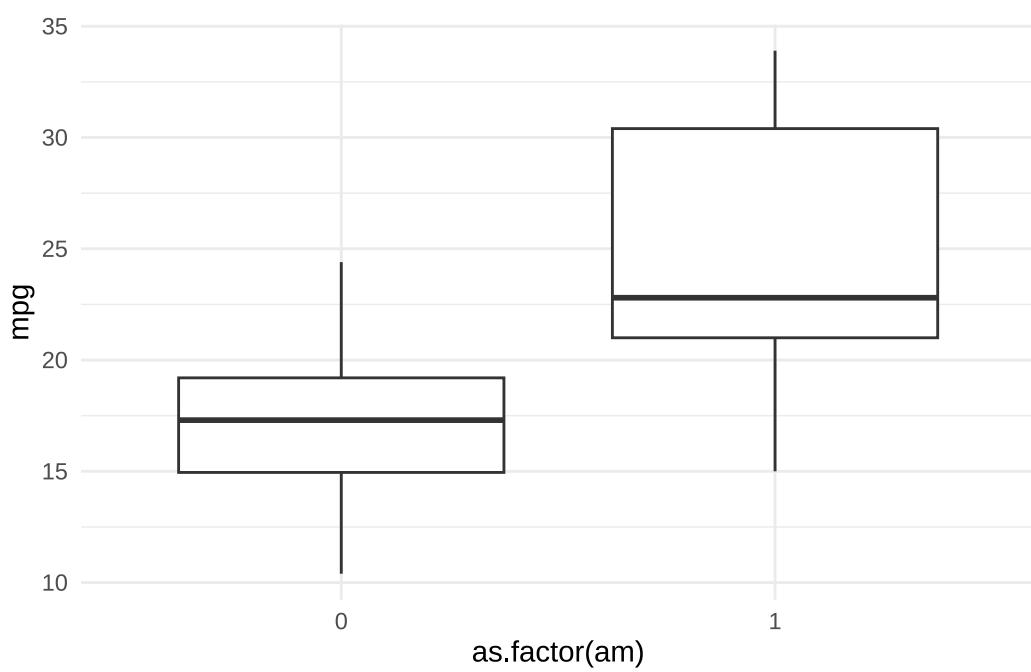
```
# ggthemes #####
plottemp+theme_economist()
```



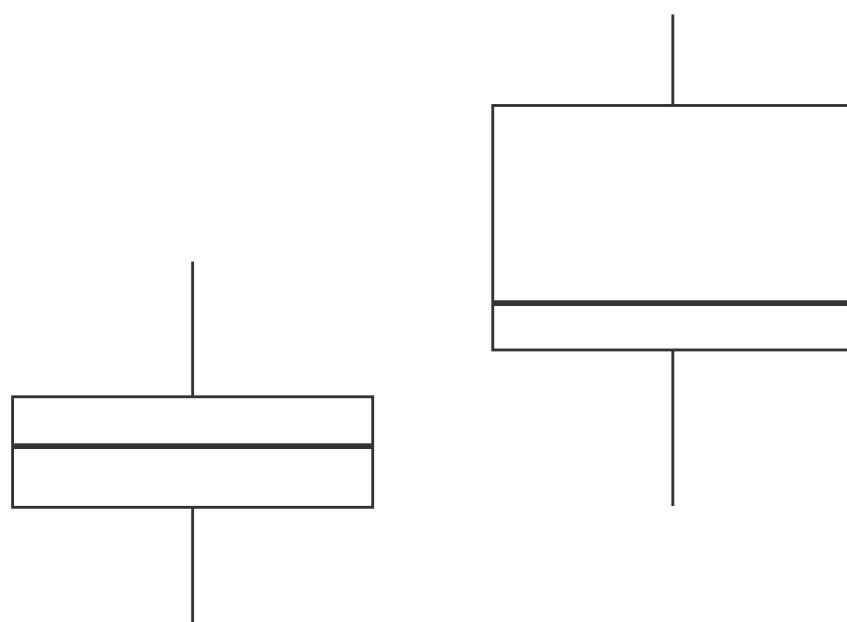
```
plottemp+theme_excel()+
  theme(axis.title.x = element_text(face="bold", hjust=0.95))
```



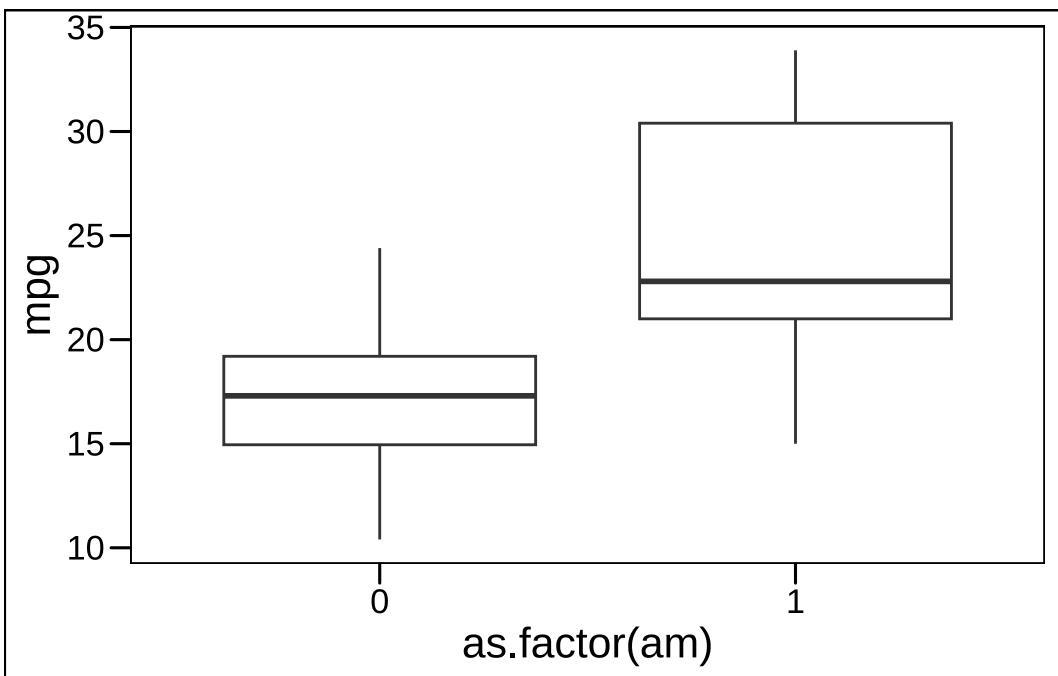
```
plottemp+theme_minimal()
```



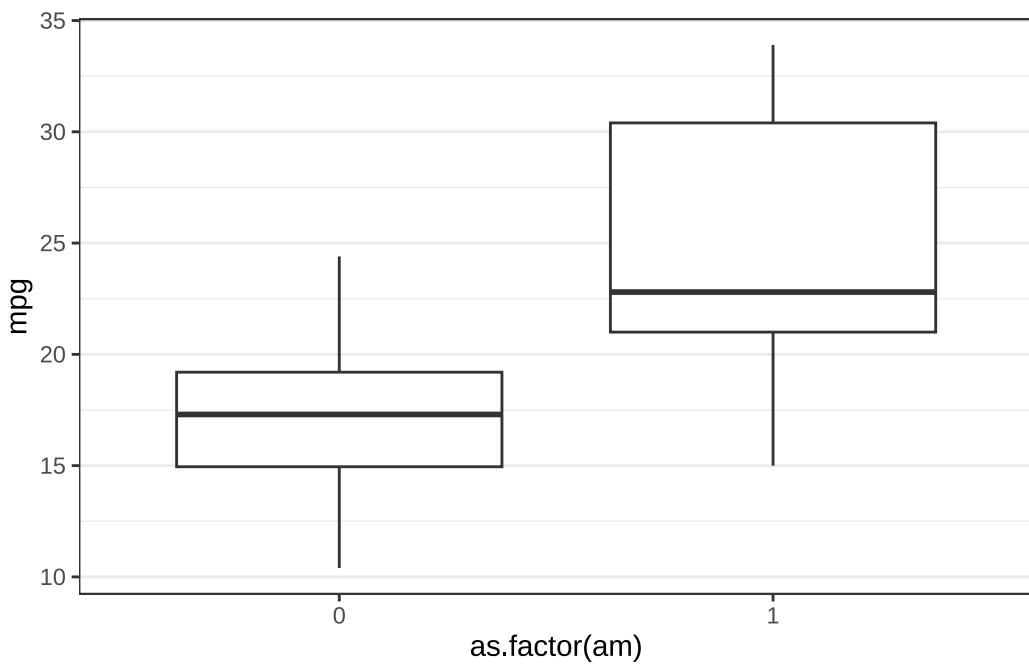
```
plottemp+theme_void()
```



```
plottemp+theme_base()
```



```
plottemp+theme_bw()+
  theme(panel.grid.major.x = element_blank())
```

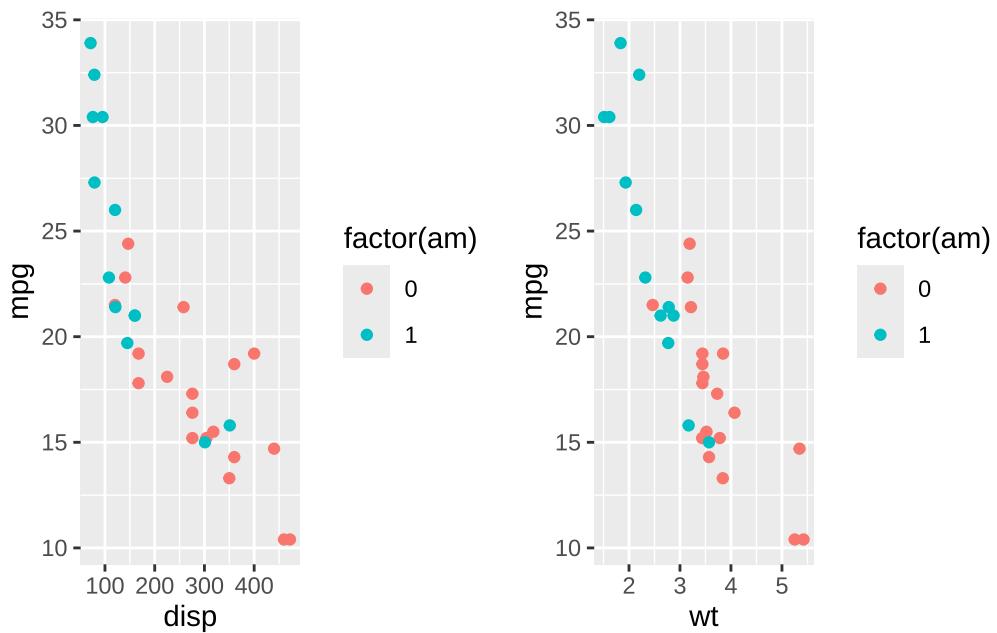


```
# https://www.data-imaginist.com/2019/a-flurry-of-facets/  
# https://github.com/thomasp85/gganimate
```

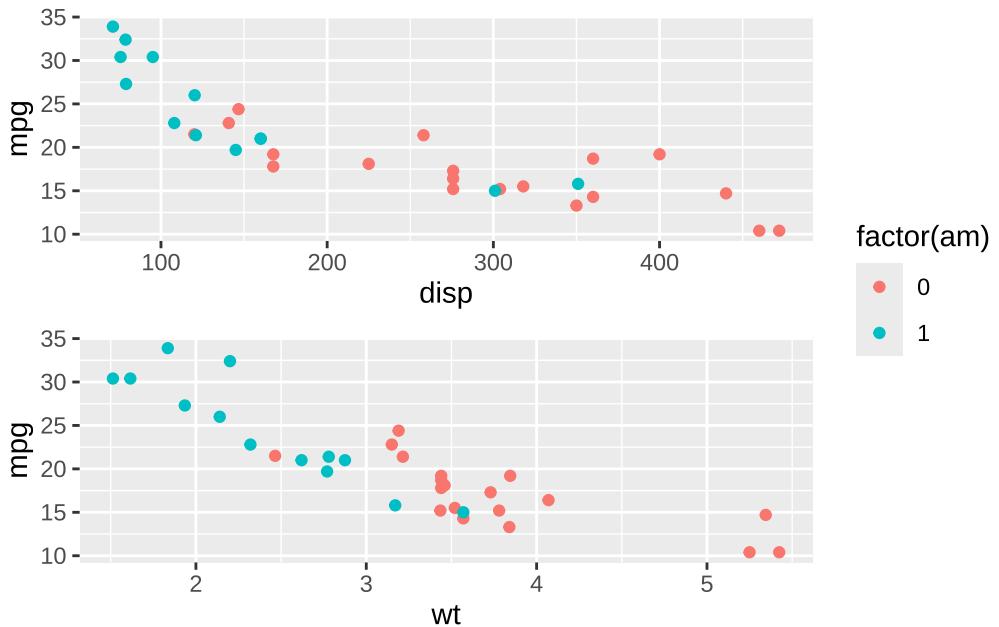
7.16 Combining figures with patchwork

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
```



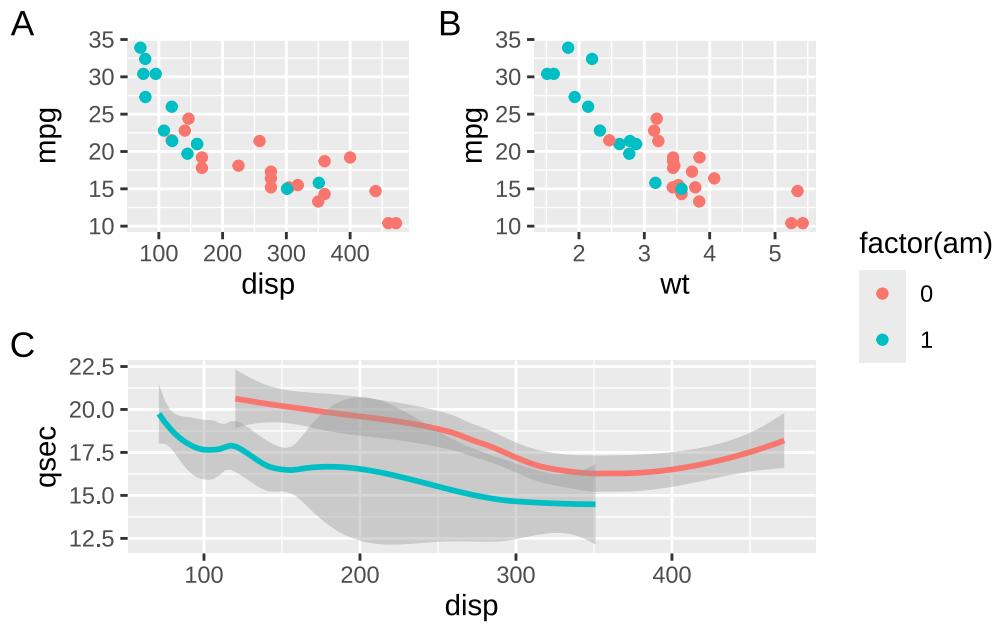
```
(p1 / p2) +  
  plot_layout(guides = "collect")
```



```
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 = "A") +
  plot_layout(guides = "collect")
```

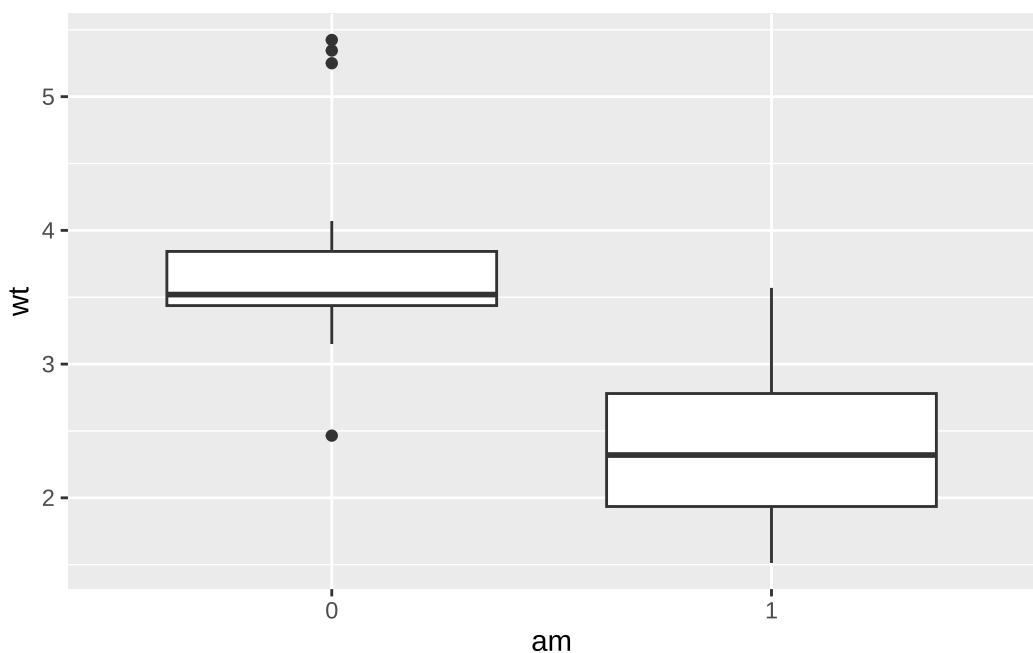
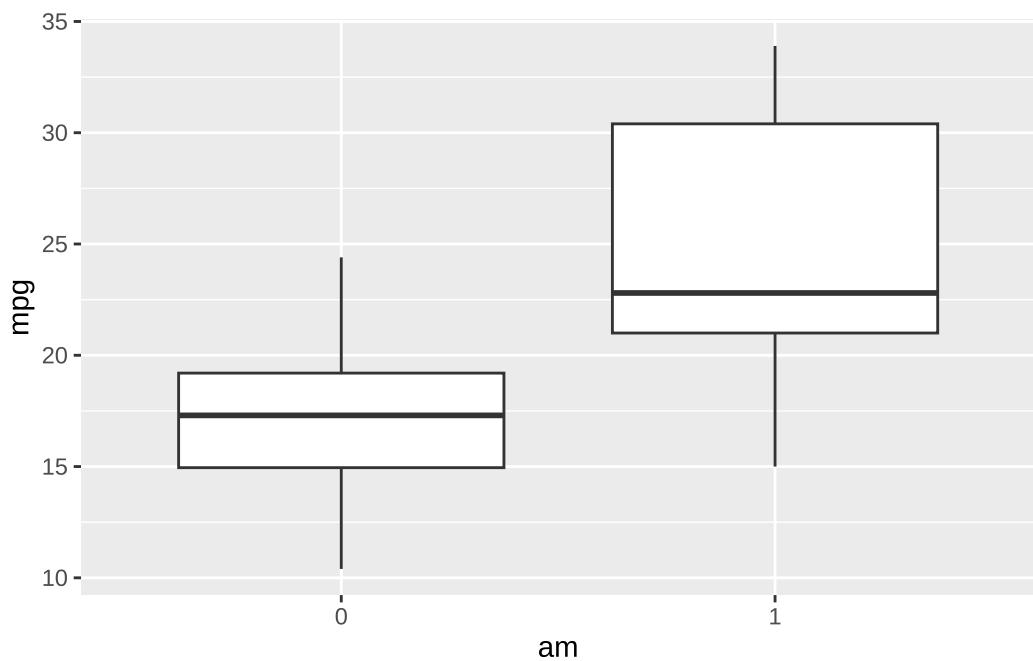
```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

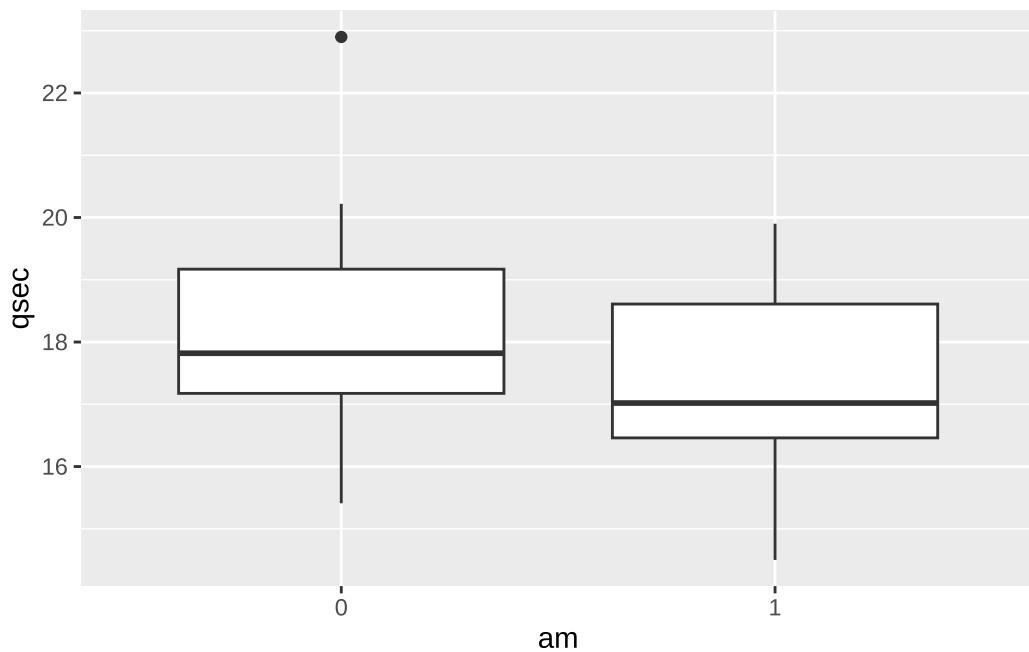


7.17 ggplots in loops

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(var_i in c("mpg", "wt", "qsec")){
  plot_temp <-
    mutate(mtcars, am = factor(am)) |>
    ggplot(aes(x = am, y = .data[[var_i]])) +
    geom_boxplot()
  print(plot_temp)
}
```





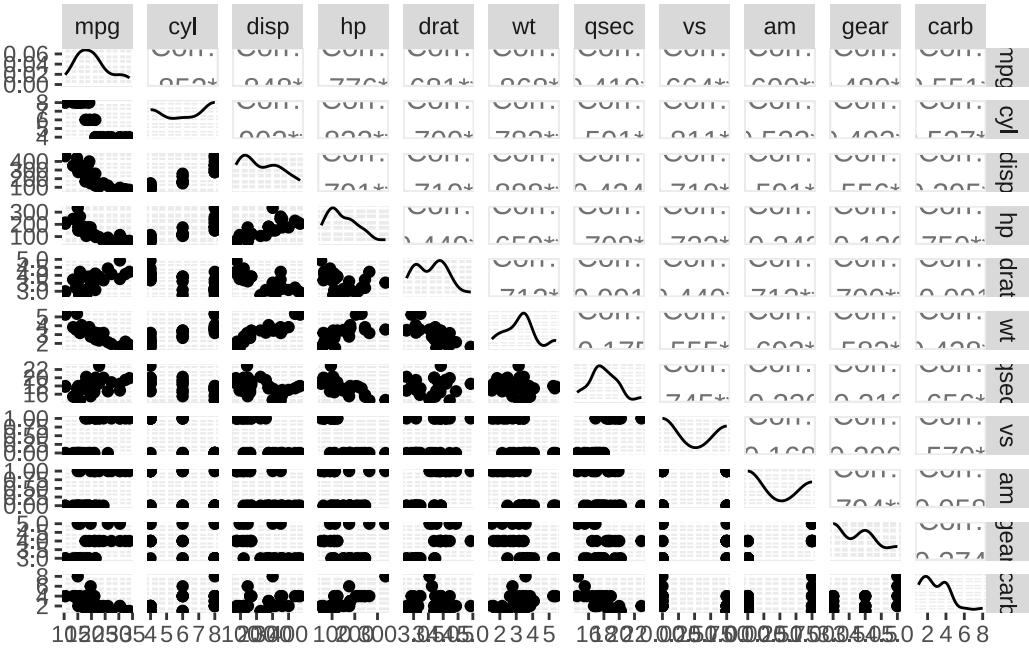
The function `aes_str()`, which expected a character representation instead of a variable name, is deprecated, but may still be suggested by chatbots

[more facets...](#)

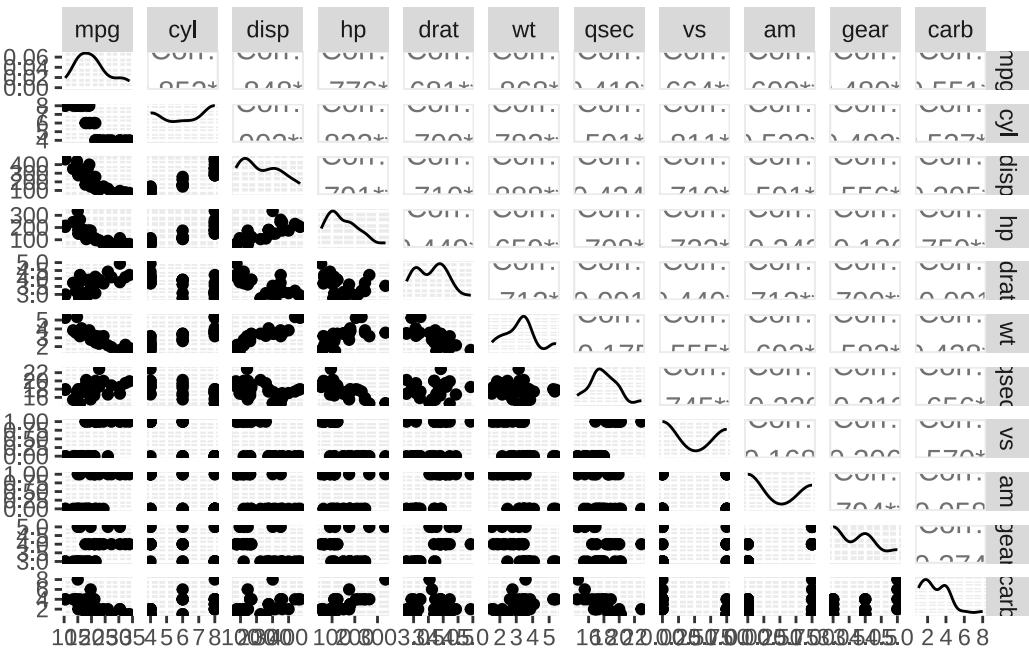
[animations...](#)

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

```
Registered S3 method overwritten by 'GGally':  
method from  
+.gg   ggplot2
```



```
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)
```

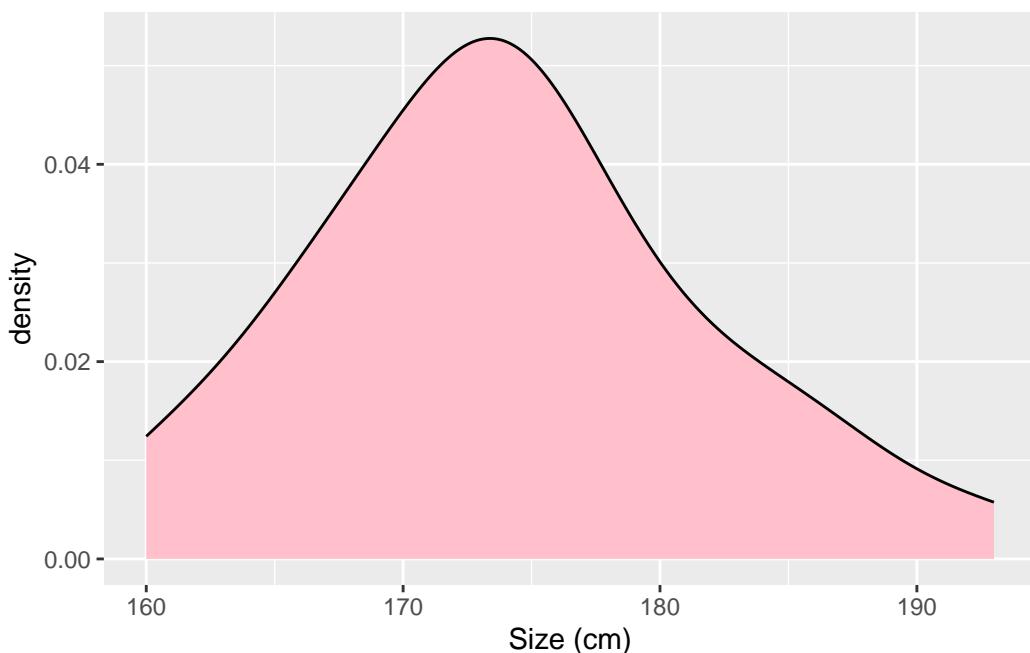
```
p_Normal_KS  
0.7063825
```

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

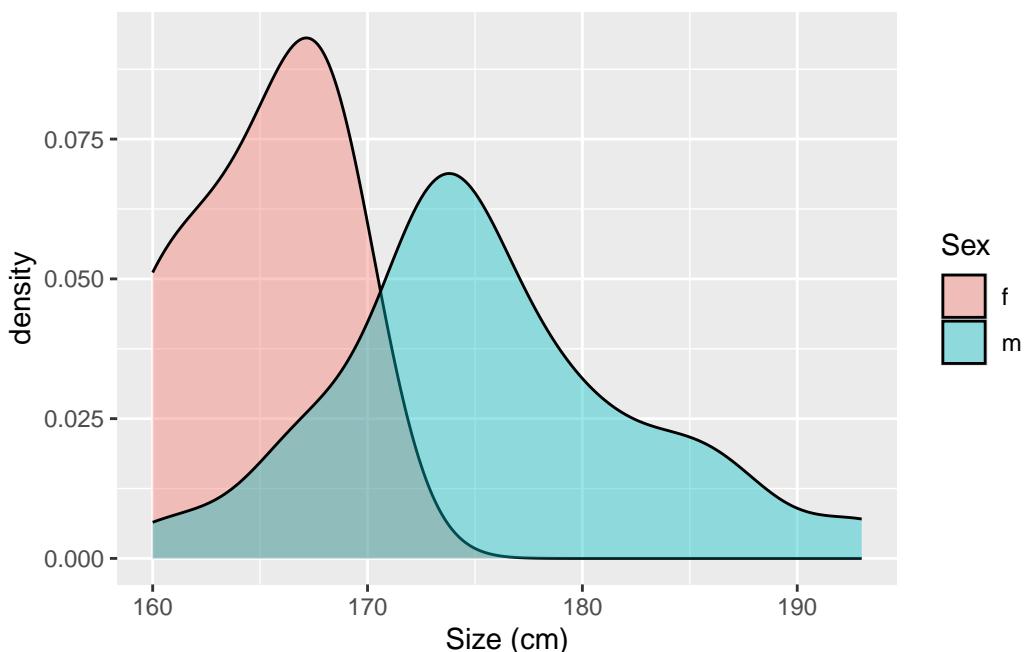
Shapiro-Wilk normality test

```
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)`)$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>        <dbl>    <dbl>
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

```

```

<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)
```

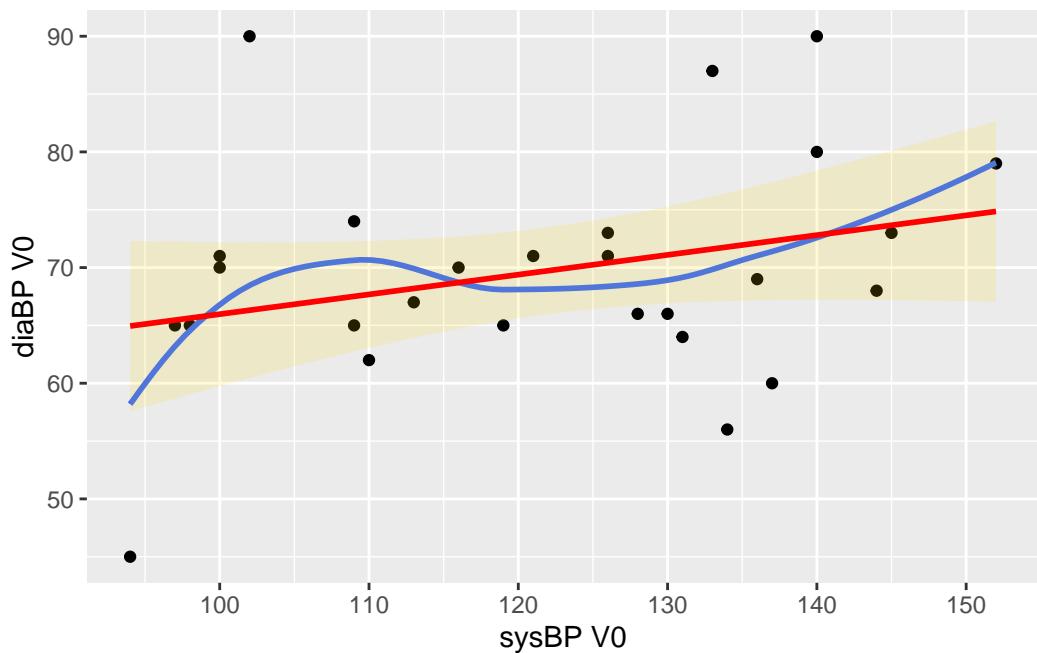
```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
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

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

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

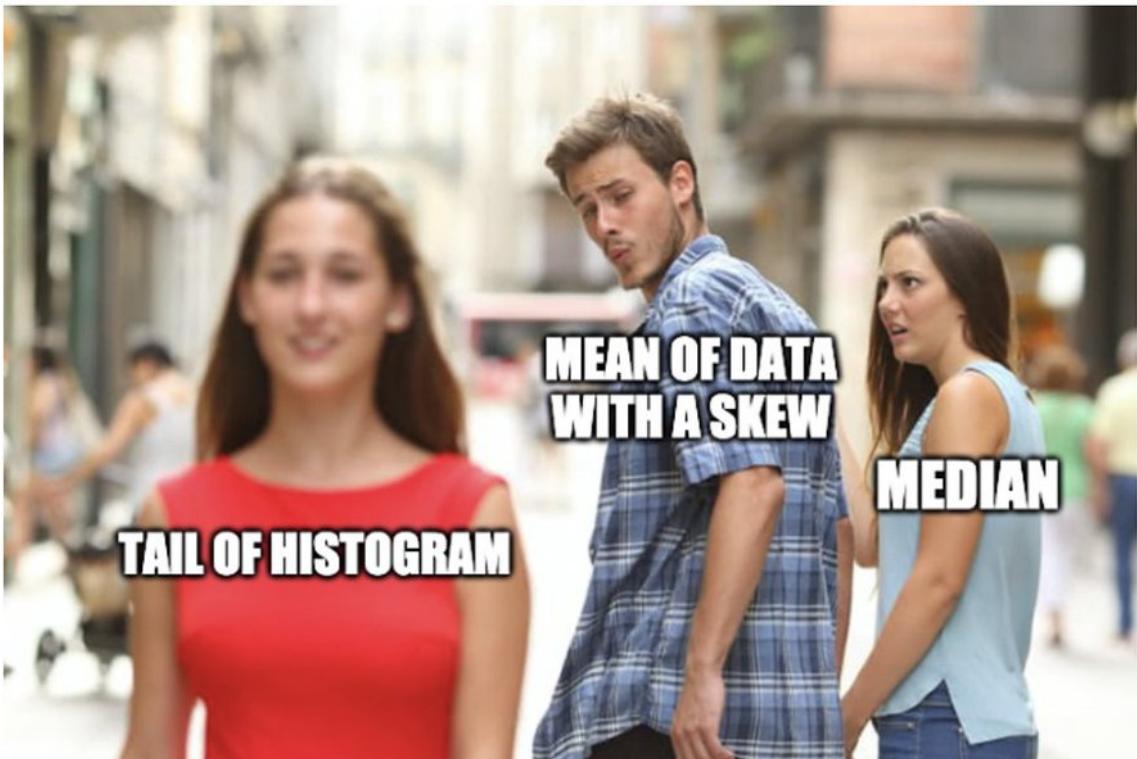
```
median_quart(rawdata$`Size (cm)`)
```

```
[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(
    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()

```

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]

Measure	f	m
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]
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

Variable	Mean (f)	Median (f)	SD (f)	Mean (m)	Median (m)	SD (m)
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

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.

- 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 ± 4	24 ± 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 ± 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	MedianQuartiles mpg
0	17 ± 4	17 (15/19)
1	24 ± 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 ± 4	24 ± 6
mpg	MedianQuartiles $17 (15/19)$	$23 (21/30)$

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 ± 6	231 ± 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 ± 6 [10 -> 34] [n = 32]	231 ± 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 ± 6 [n = 32]	19 (15/23)
disp	231 ± 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 ± 4 [n = 19]	17 (15/19)	290 ± 110 [n = 19]	276 (177/360)
1	24 ± 6 [n = 13]	23 (21/30)	144 ± 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		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!**

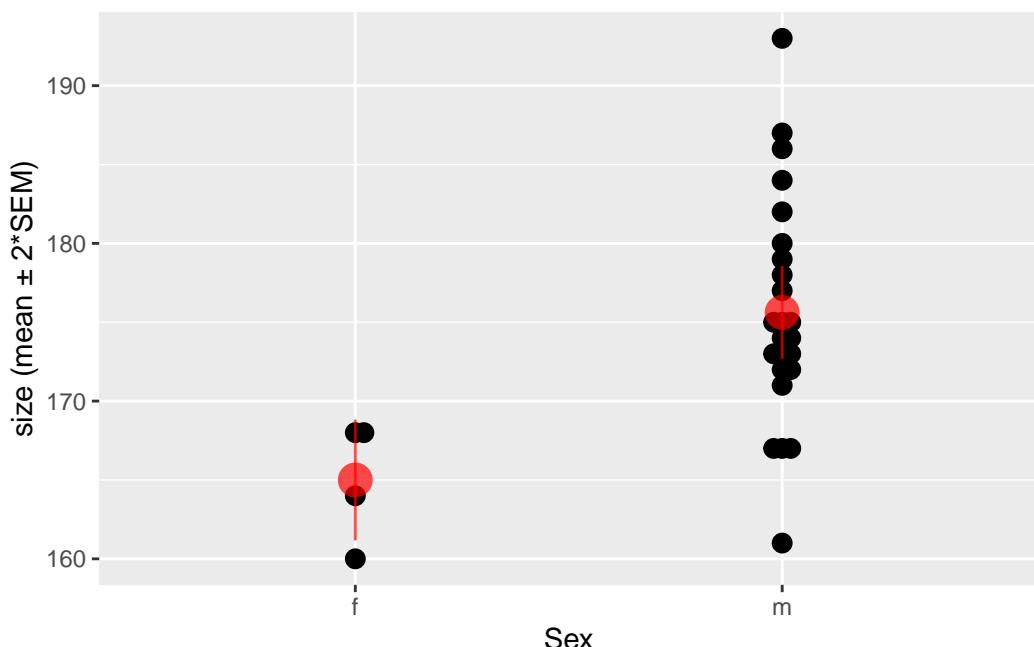
```
pacman::p_load(conflicted, plotrix, tidyverse, wrappedtools,
                 coin, ggsignif, patchwork, ggbeeswarm,
                 flextable, here)
#conflicted)
# conflict_prefer("filter", "dplyr")
load(here("data/bookdata1.RData"))
```

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
```

```
# manual entry
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

Variable	desc_all	Testmedication 0	Testmedication 1	p
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
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

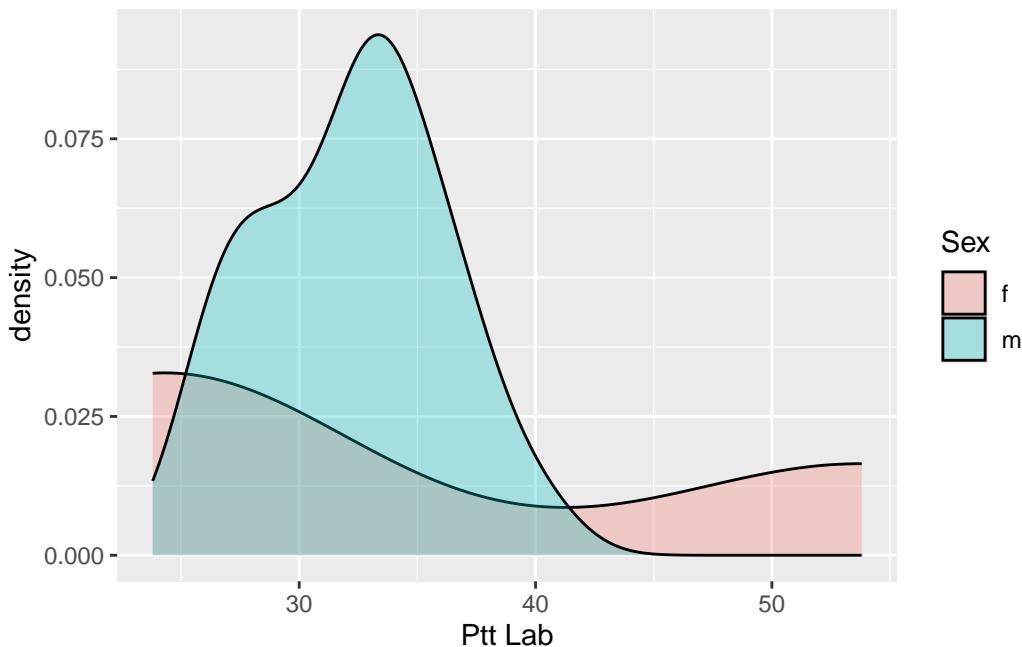
- nonparametric, no distribution is assumed
- based on rank-transformed data
- insensitive to extreme values

```
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)
```

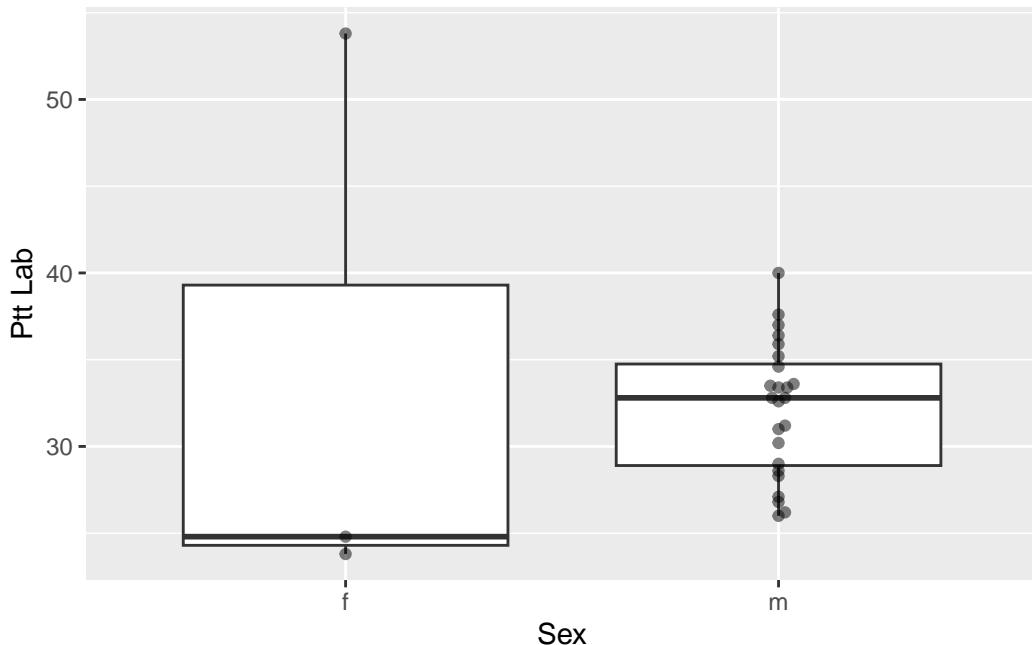
```
rawdata$Sex: f  
[1] "25 (24/49)"
```

```
-----  
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
m	12
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 = 0.1746
```

```
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 = 0.5961
```

```
tabletestOut$p.value
```

```
[1] 0.596114
```

```
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
Sex				1.000
f	4 (14.29%)	2 (14.29%)	2 (14.29%)	
m	24 (85.71%)	12 (85.71%)	12 (85.71%)	
NYHA V1				0.728
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%)	
NYHA V2				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%)	
NYHA V3				0.772
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 Covariance / Correlation

```
pacman::p_load(conflicted, plotrix, tidyverse, wrappedtools,
                 coin, ggsignif, patchwork, ggbeeswarm,
                 flextable, here, corrr)
load(here("data/bookdata1.RData"))
```

13.1 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
```

13.2 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

```

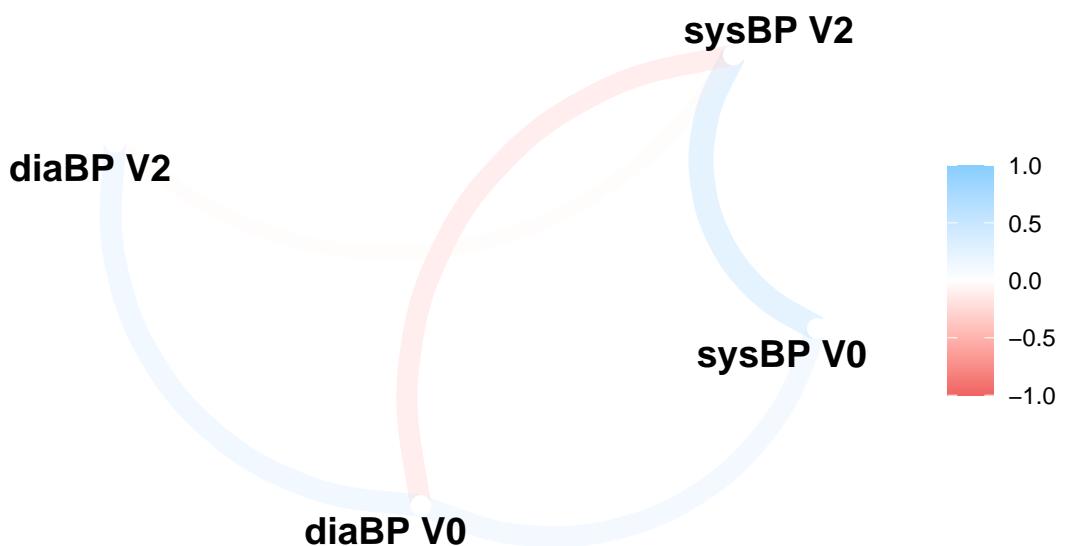
13.3 Vizualizations

```

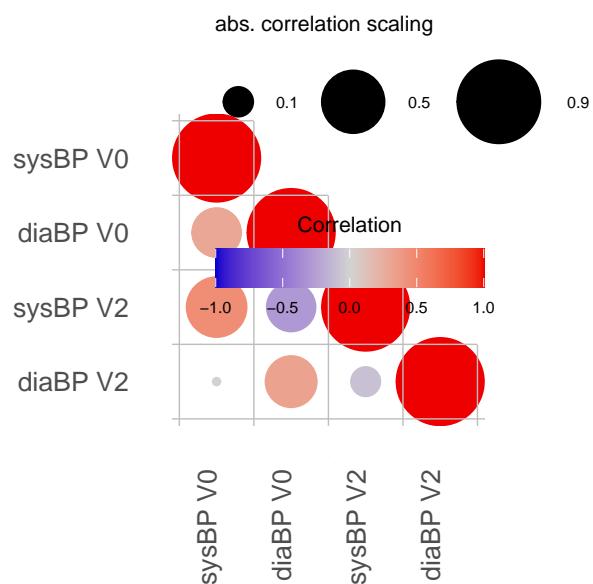
corrr::correlate(rawdata |> select(contains("BP"))) |>
  corrr::network_plot(min_cor = .05)

```

Correlation computed with
 * Method: 'pearson'
 * Missing treated using: 'pairwise.complete.obs'

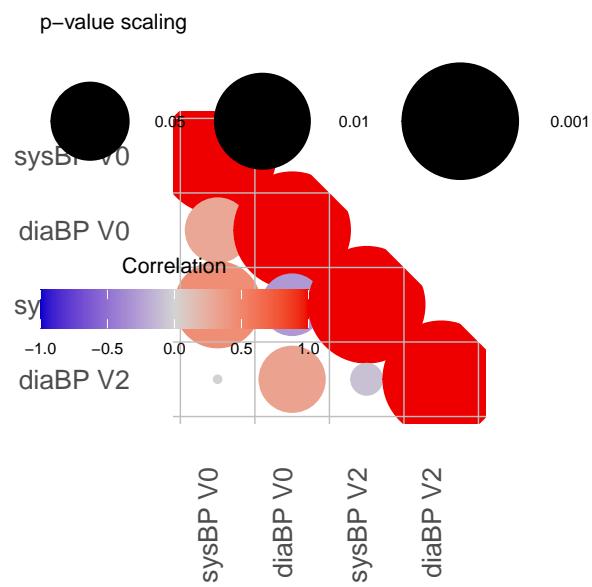


```
wrappedtools::ggcormat(cor_mat = cor_out$corout,
maxpoint = 15)
```



```
wrappedtools::ggcormat(cor_mat = cor_out$corout,
p_mat = cor_out$pout,
```

`maxpoint = 20)`



14 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.

14.1 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,
                 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()
```

14.2 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

14.3 Graphical exploration

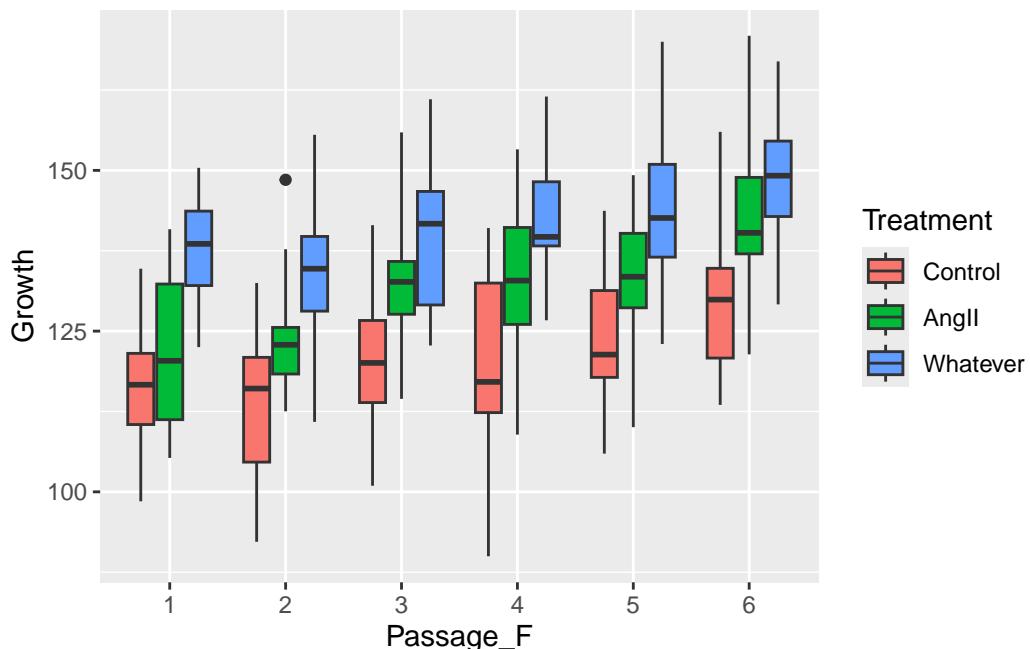
First impression of the data will be attempted by grouped boxplot, followed by interaction plots, both as basic and ggplot with variations.

```

ggplot(rawdata,aes(Passage_F,Growth, fill=Treatment))+  

  geom_boxplot(coef=3)

```



```

with(rawdata, interaction.plot(  

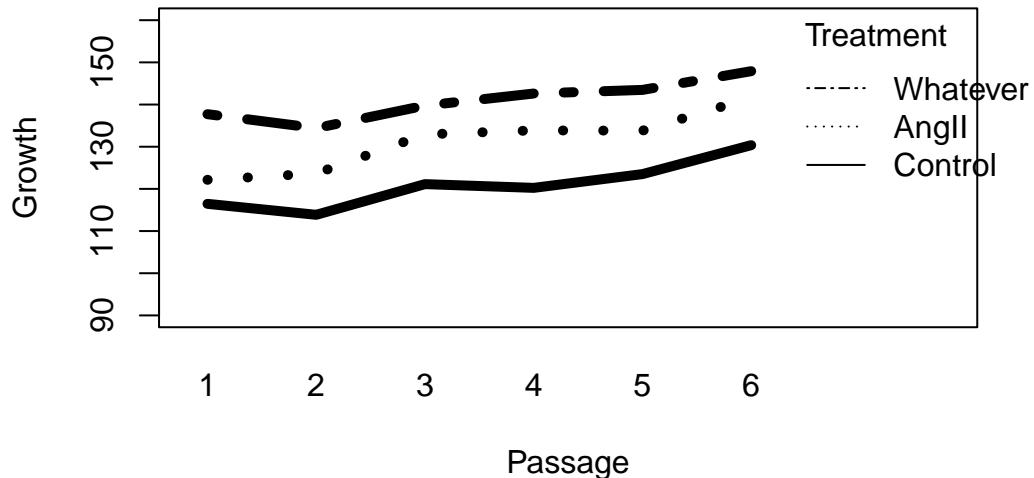
  x.factor=Passage, trace.factor=Treatment, response=Growth,  

  ylim = c(90, 160), lty = c(1,3,12), lwd = 5,  

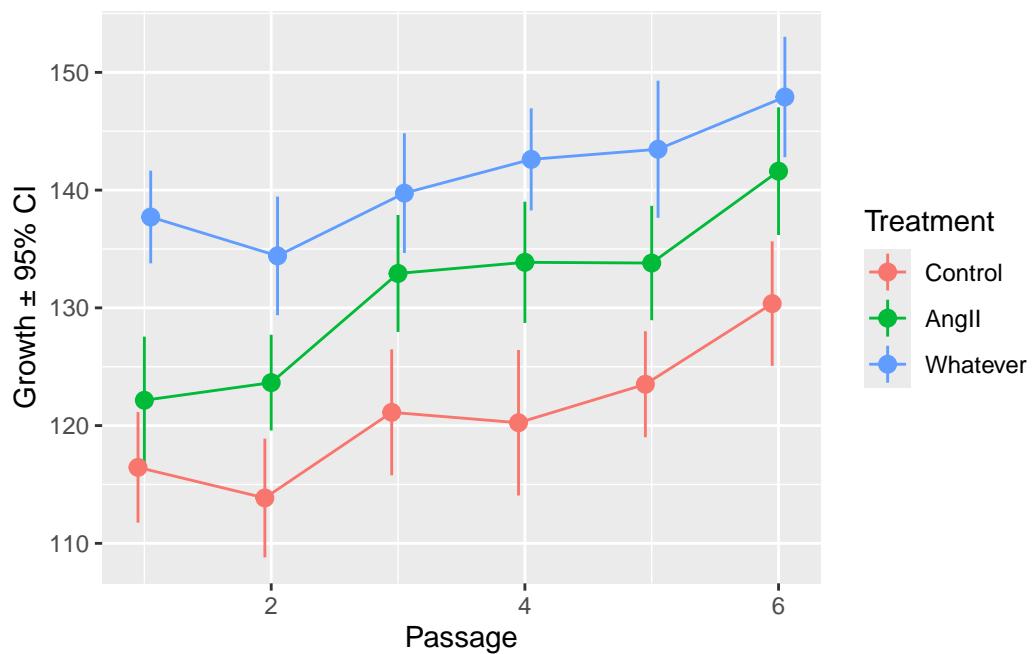
  ylab = "Growth", xlab = "Passage",

```

```
trace.label = "Treatment"))
```

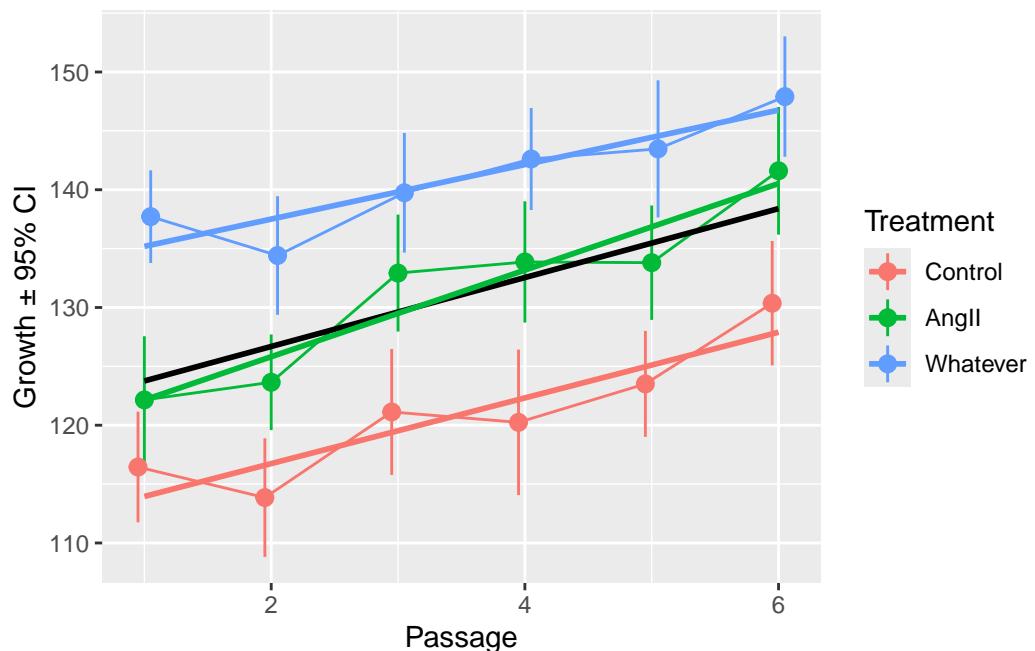


```
# p1<-ggplot(rawdata,aes(x=Passage,y=Growth))+  
#   stat_summary(geom='line',fun='mean',aes(color=Treatment))+  
#   stat_summary(geom='line',fun='mean')  
p1<-ggplot(rawdata,aes(x=Passage,y=Growth))+  
  stat_summary(geom='line',fun='mean',  
               aes(color=Treatment),  
               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
```



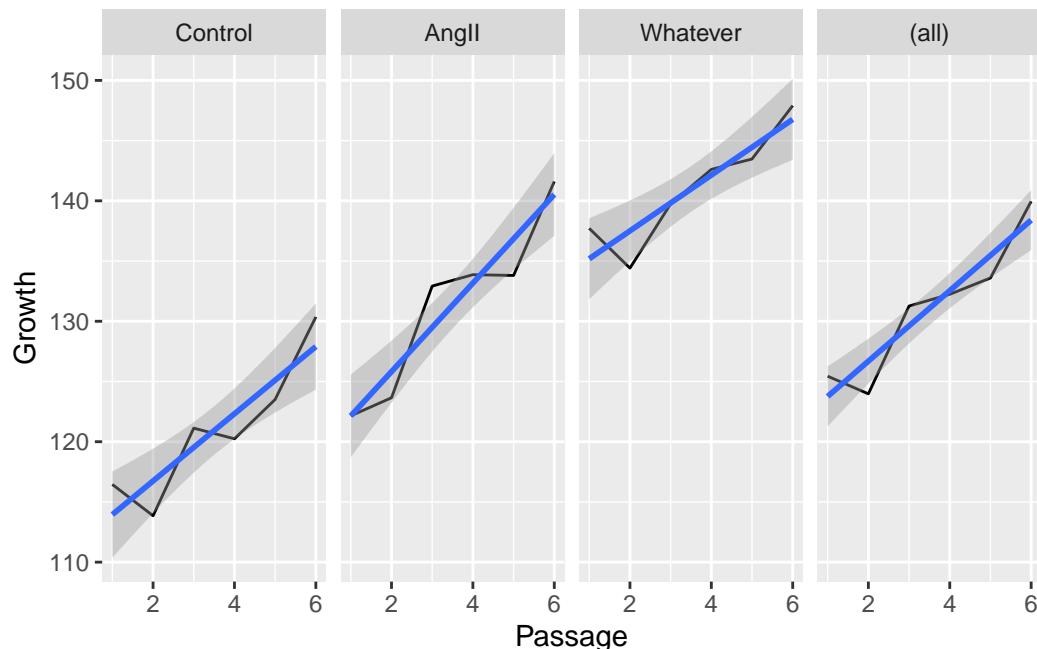
```
p1+geom_smooth(method='lm',color='black',se=F)+  
  geom_smooth(method='lm',aes(color=Treatment),se=F)
```

```
`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'



14.4 Linear Models

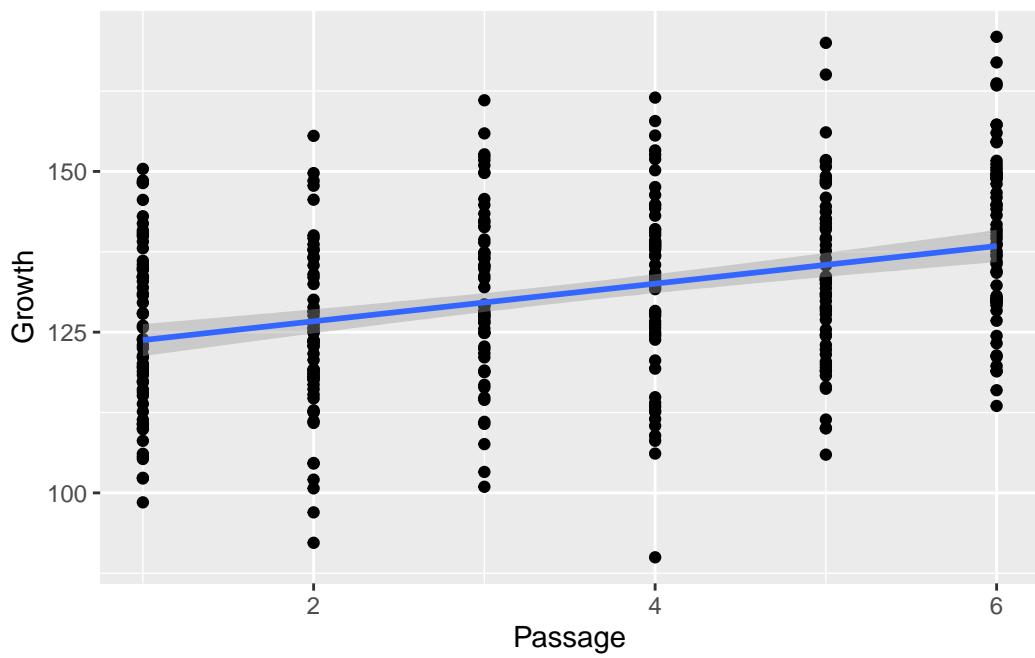
14.4.1 Linear regression

We will analyse the relation between independent variable (IV) Passage and dependent variable (DV) Growth.

14.4.1.1 Graphical exploration

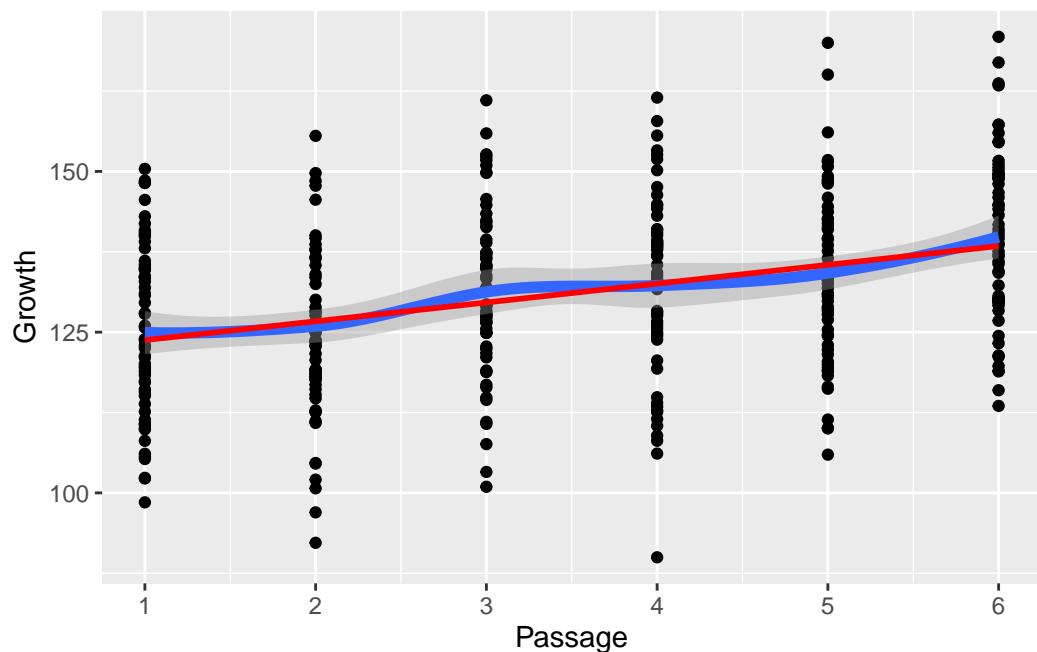
```
ggplot(rawdata,aes(Passage,Growth))+  
  geom_point() +  
  geom_smooth(method=lm)
```

`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'
```



14.4.1.2 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)
```

14.4.1.3 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

```

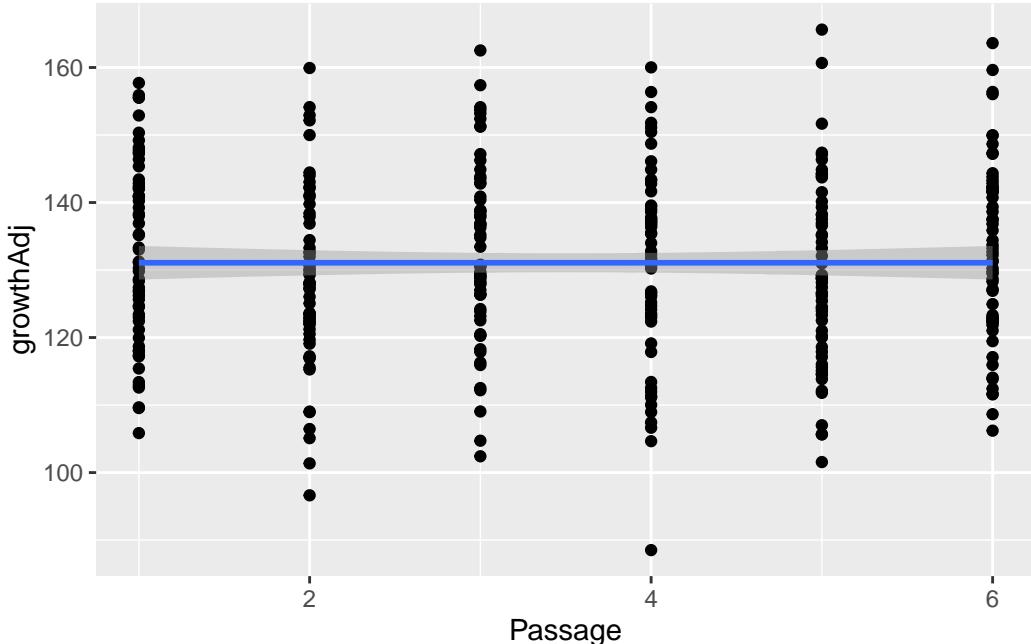
```

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
```

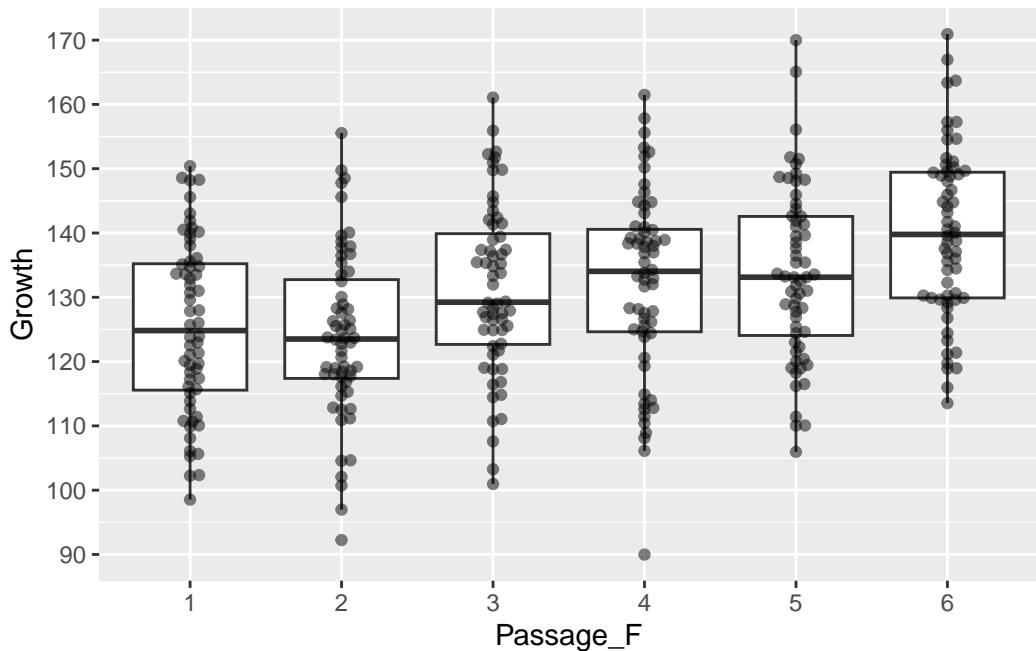
term	estimate	std.error	statistic	p.value
<chr>	<dbl>	<dbl>	<dbl>	<dbl>
1 (Intercept)	1.31e+ 2	1.63	8.05e+ 1	2.04 e-231
2 Passage	6.80e-15	0.418	1.63e-14	1.000e+ 0

14.4.2 ANOVA

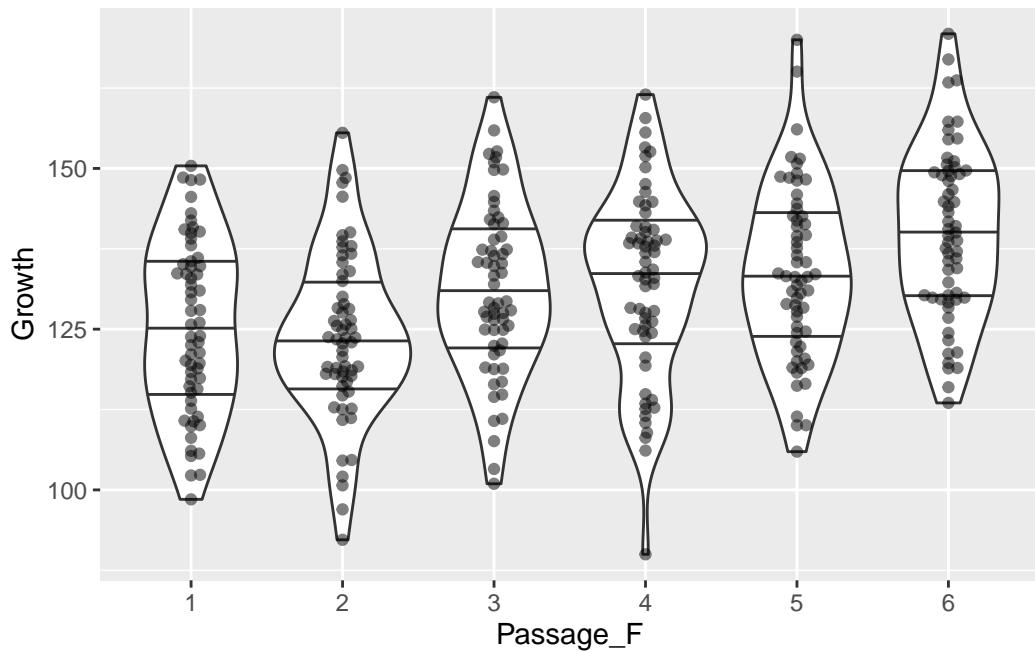
In the linear regression, we had Passage as a continuous IV, estimating a global ‘universal’ effect supposed to be constant. Now we look at Passage_F and model a discrete IV, allowing for specific effects, and thereby comparing means between groups.

14.4.2.1 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)
```



14.4.2.2 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
```

14.4.2.3 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
5.521460e-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
9.031123e-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'))
```

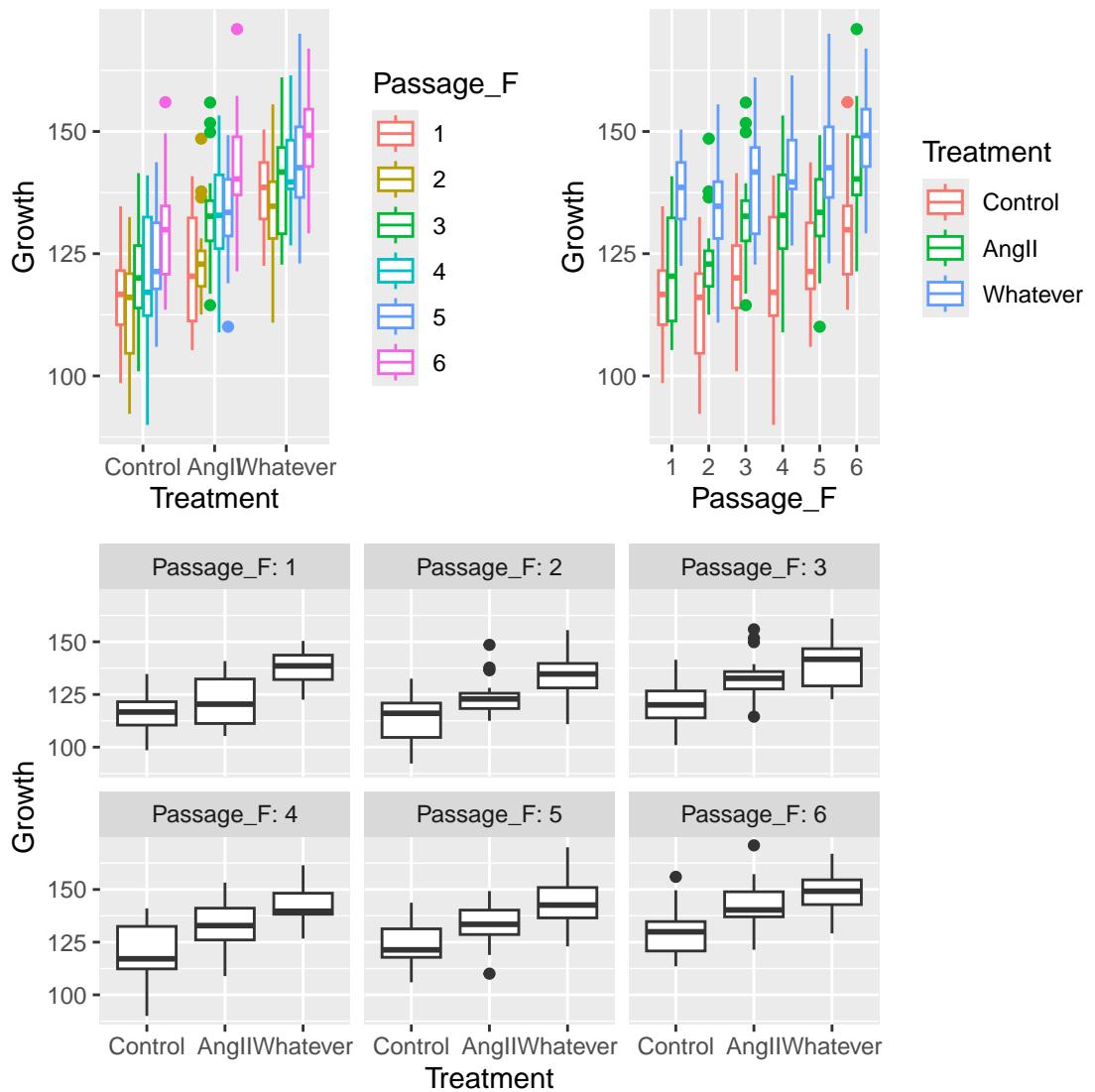
```
[1] "0.552" "0.023" "0.010" "0.003" "0.001"
```

14.4.3 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 $IV1*IV2$.

14.4.3.1 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
```



14.4.3.2 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
```

14.4.3.3 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 0.0263 *
5-1 8.156143 1.9335375 14.378748 0.0046 **
6-1 14.520106 8.2975011 20.742712 2.5e-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)
```

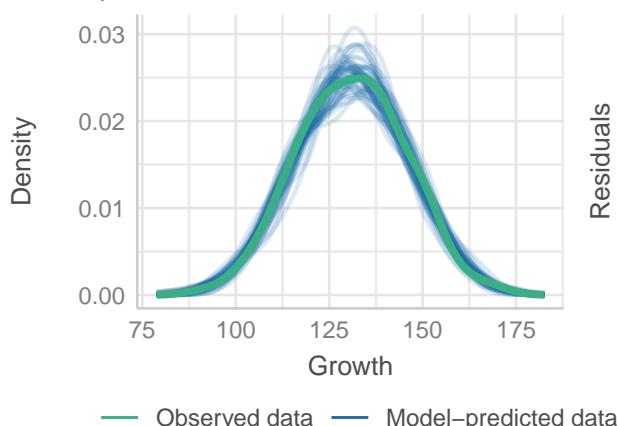
```
# A tibble: 3 x 5  
term      sumsq    df statistic  p.value  
<chr>     <dbl> <dbl>     <dbl>      <dbl>  
1 (Intercept) 1754743.     1    12391.  2.91e-279  
2 Treatment    24137.      2      85.2  5.55e- 31  
3 Residuals    50558.    357       NA      NA
```

14.4.4 Model exploration with package performance

```
# x11() #interactive only!
# quartz() for mac
# from package performance
check_model(lm0ut_additive)
```

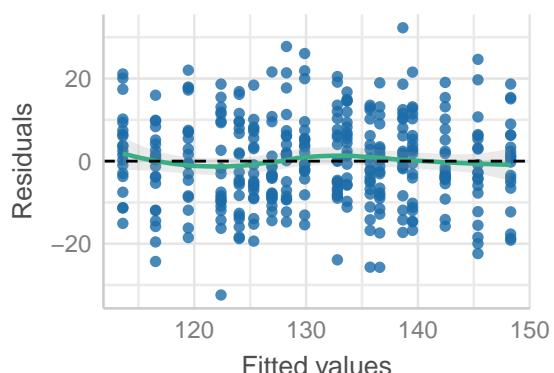
Posterior Predictive Check

Model-predicted lines should resemble observed data



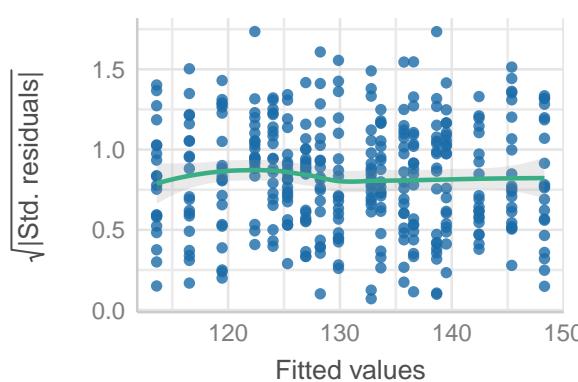
Linearity

Reference line should be flat and horizontal



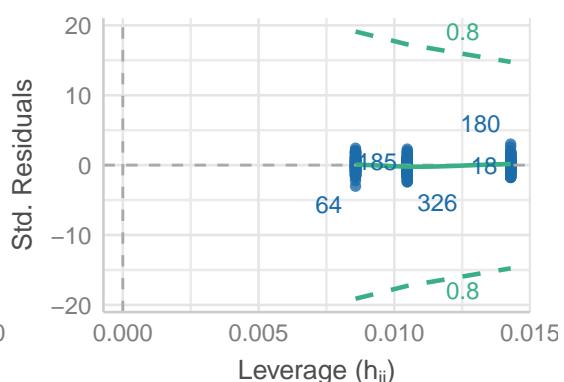
Homogeneity of Variance

Reference line should be flat and horizontal



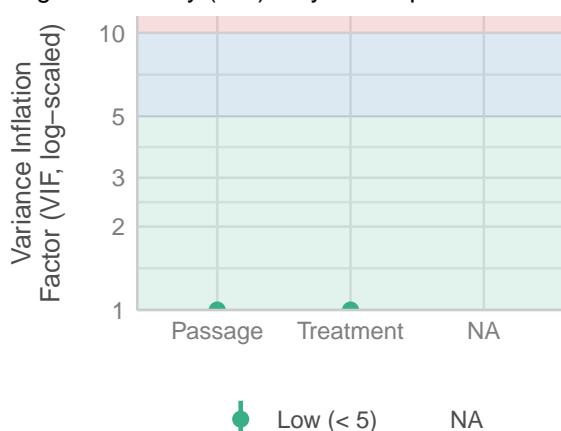
Influential Observations

Points should be inside the contour lines



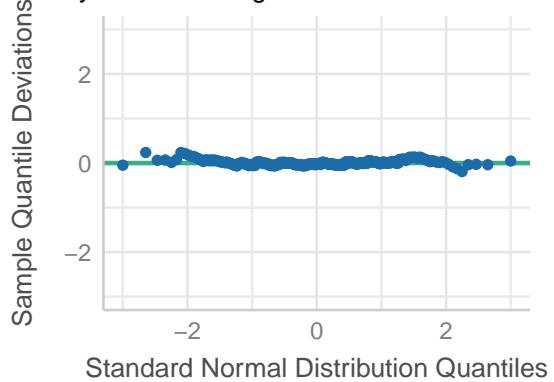
Collinearity

High collinearity (VIF) may inflate parameter uncertainty



Normality of Residuals

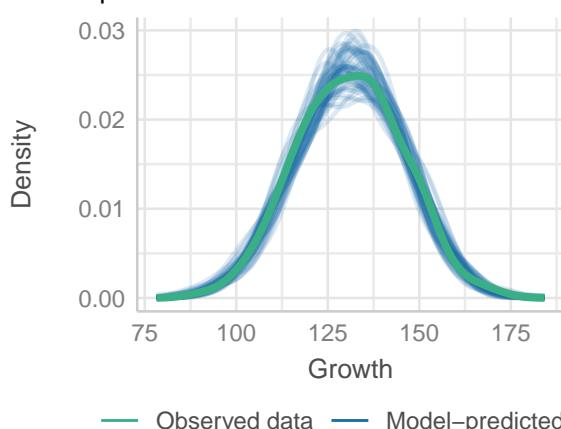
Density should fall along the line



```
check_model(lmOut_interaction)
```

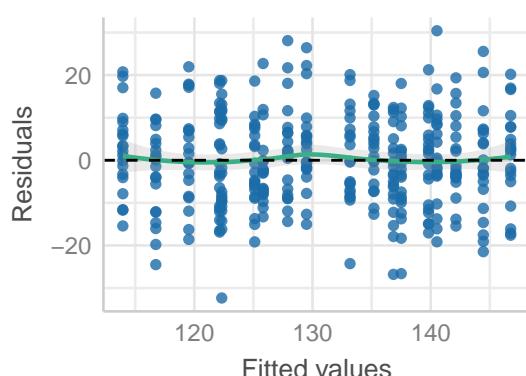
Posterior Predictive Check

Model-predicted lines should resemble observed



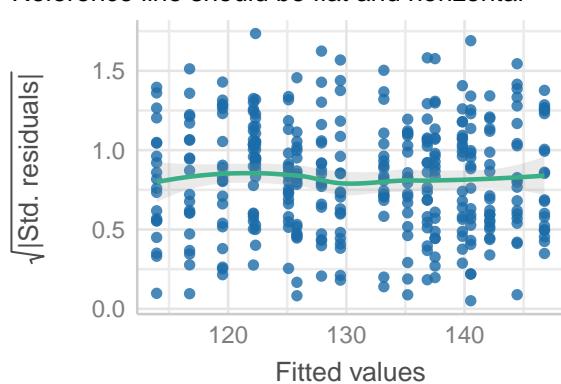
Linearity

Reference line should be flat and horizontal



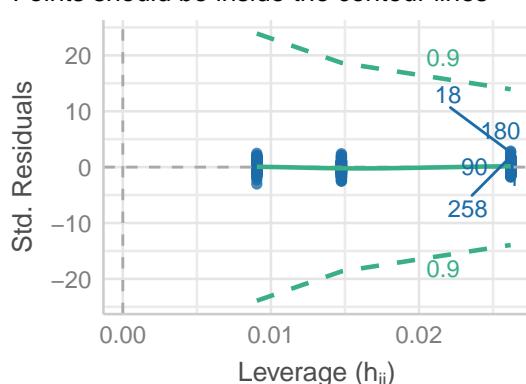
Homogeneity of Variance

Reference line should be flat and horizontal



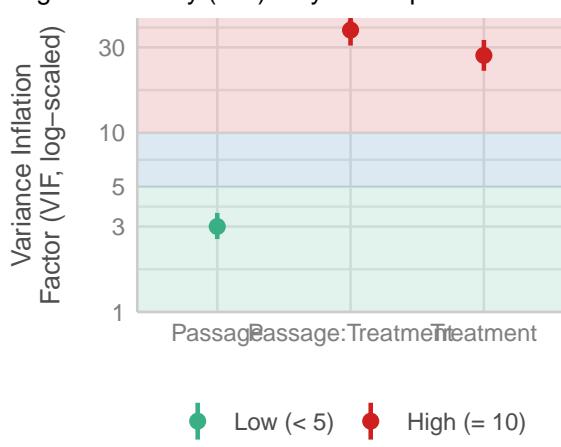
Influential Observations

Points should be inside the contour lines



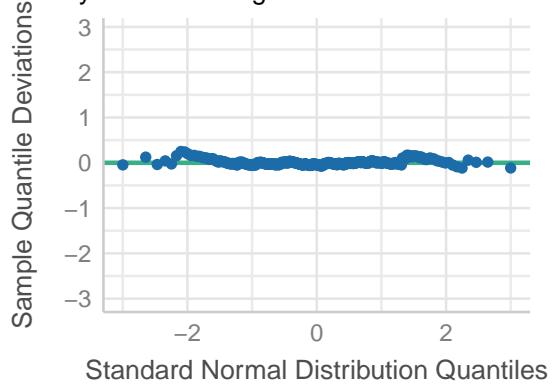
Collinearity

High collinearity (VIF) may inflate parameter uncertainty



Normality of Residuals

Density should fall along the line



```
# dev.off()
```

15 Interaction in linear models

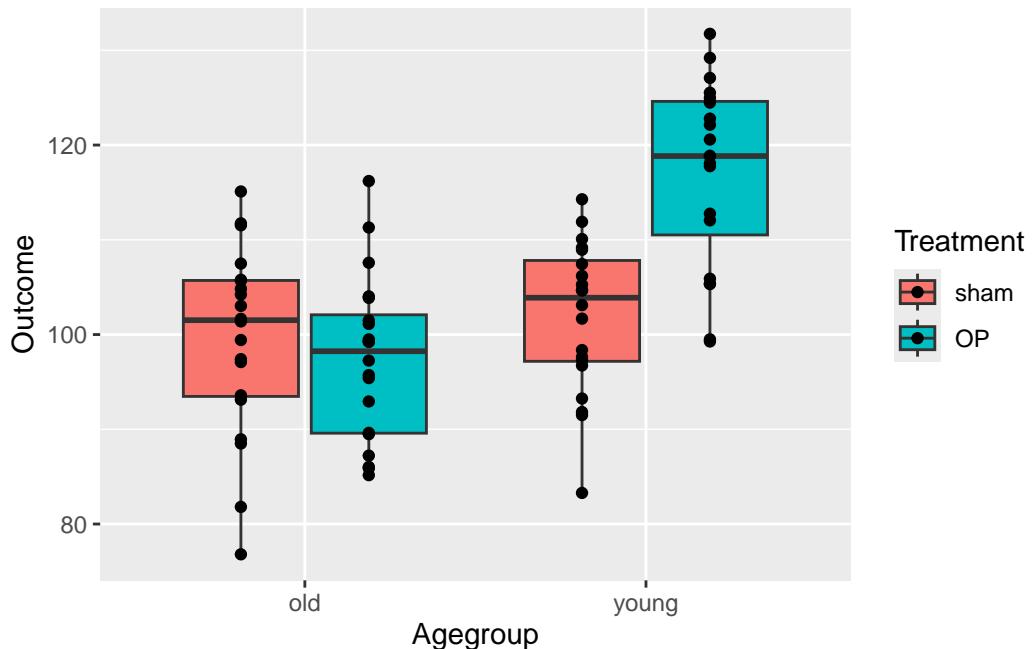
```
pacman::p_load(conflicted,tidyverse,car,multcomp,wrappedtools, broom)
conflicts_prefer(dplyr::select, dplyr::filter)
```

```
[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.

15.1 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)

15.2 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))+  

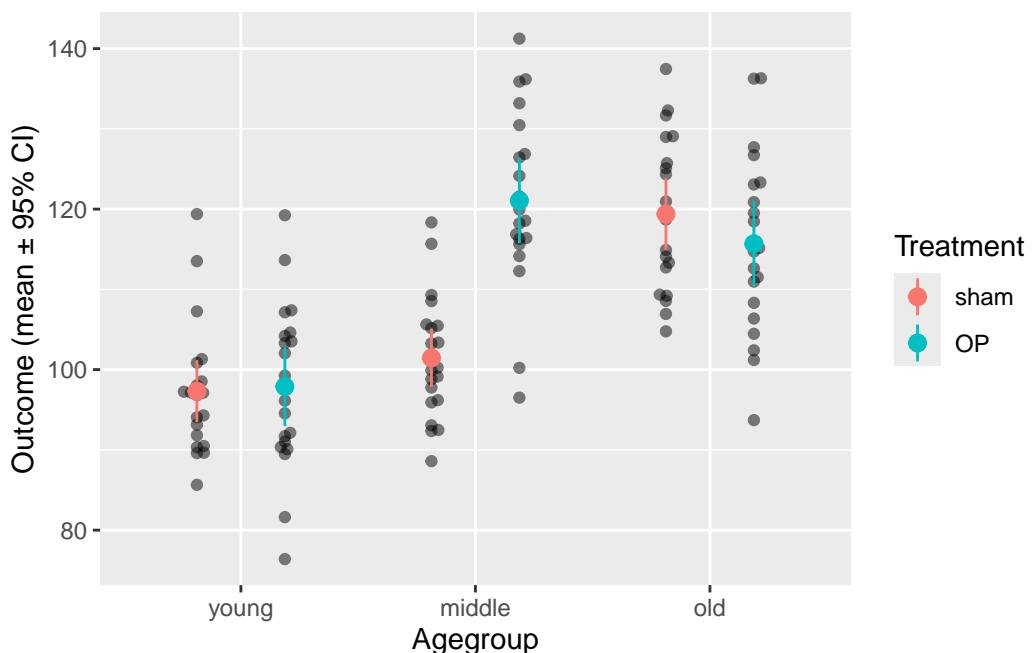
  # geom_boxplot()+
  ggbeeswarm::geom_beeswarm(dodge.width = .75, alpha=.5)+  

  stat_summary(aes(color=Treatment), fun.data= mean_cl_normal,  

  position=position_dodge(width = .75))+  

  ylab("Outcome (mean \u00b1 95% CI)")

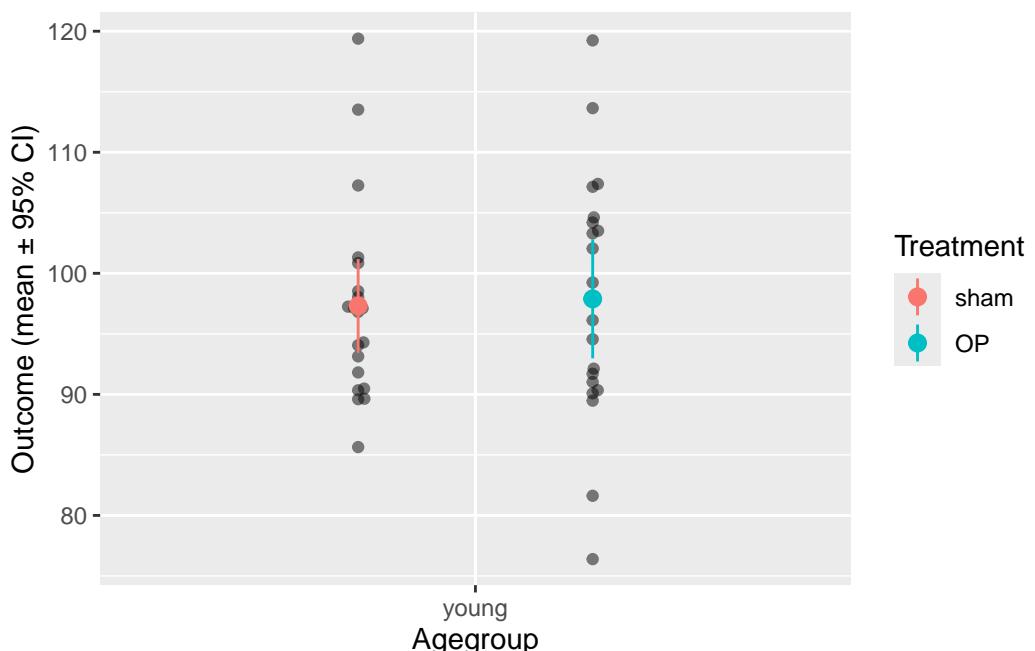
```



```

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)")

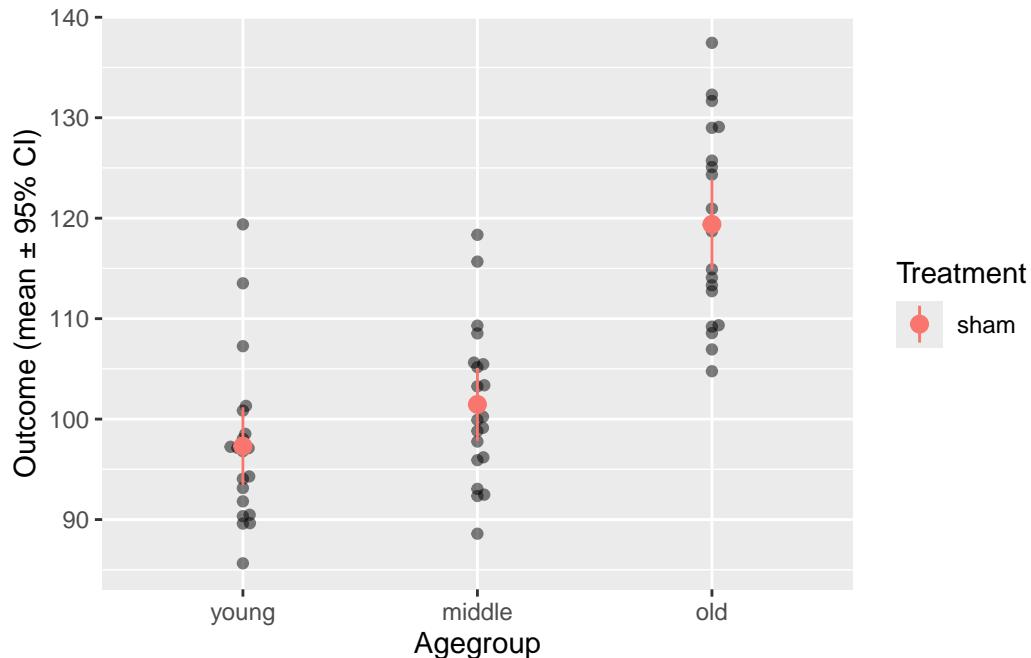
```



```

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:

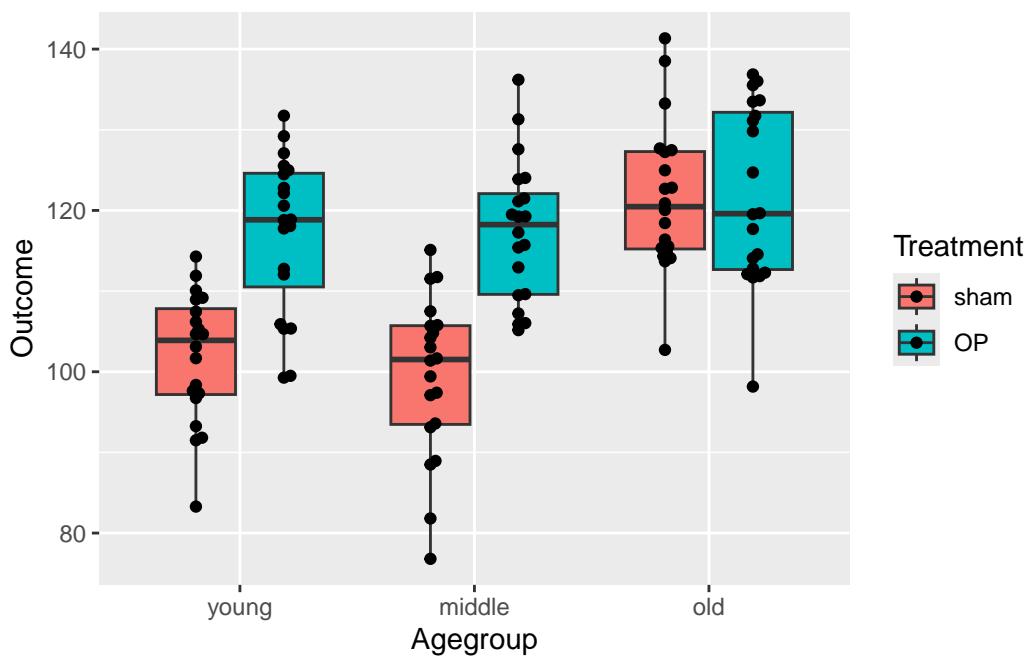
	Estimate	Std. Error	t value	Pr(> t)
middle - young == 0	4.149	3.131	1.325	0.384
old - young == 0	22.073	3.131	7.049	<1e-04 ***
old - middle == 0	17.924	3.131	5.724	<1e-04 ***

Signif. codes:	0 '***'	0.001 '**'	0.01 '*'	0.05 '.'
	0.1 '	'	1	
(Adjusted p values reported -- single-step method)				

15.3 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
```

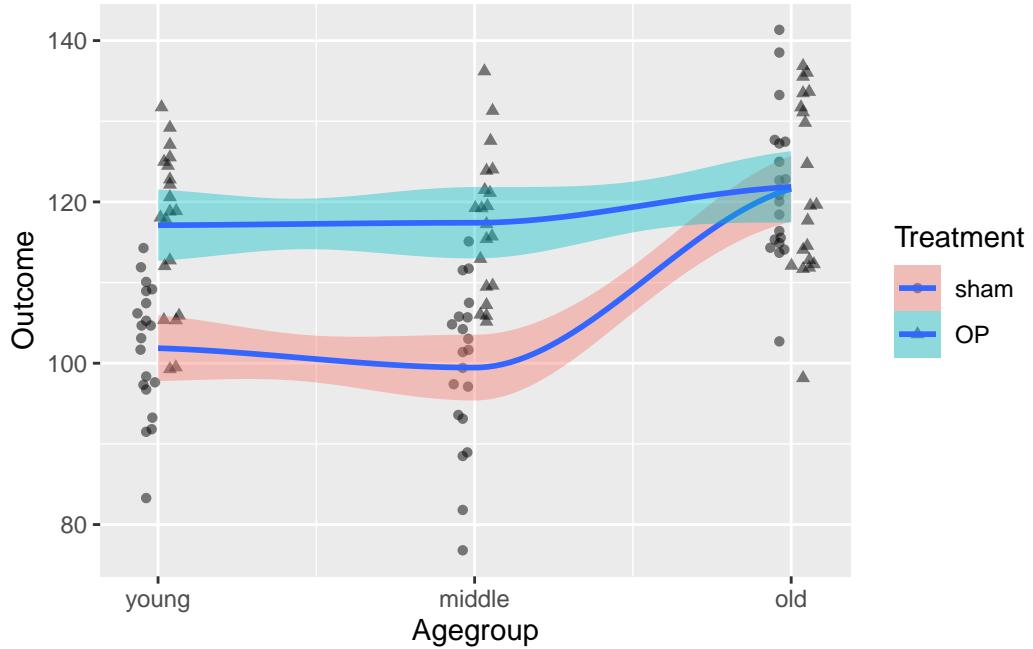
```
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: There are other near singularities as well. 4.0401
```

```
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 ***
```

15.4 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	13.91506
cyl	wt:gear	wt:factor(am)1	wt:cyl
3.13308	4.90387	-11.51010	0.06975
gear:factor(am)1	gear:cyl	factor(am)1:cyl	
2.23288	-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		

16 Logistic regression

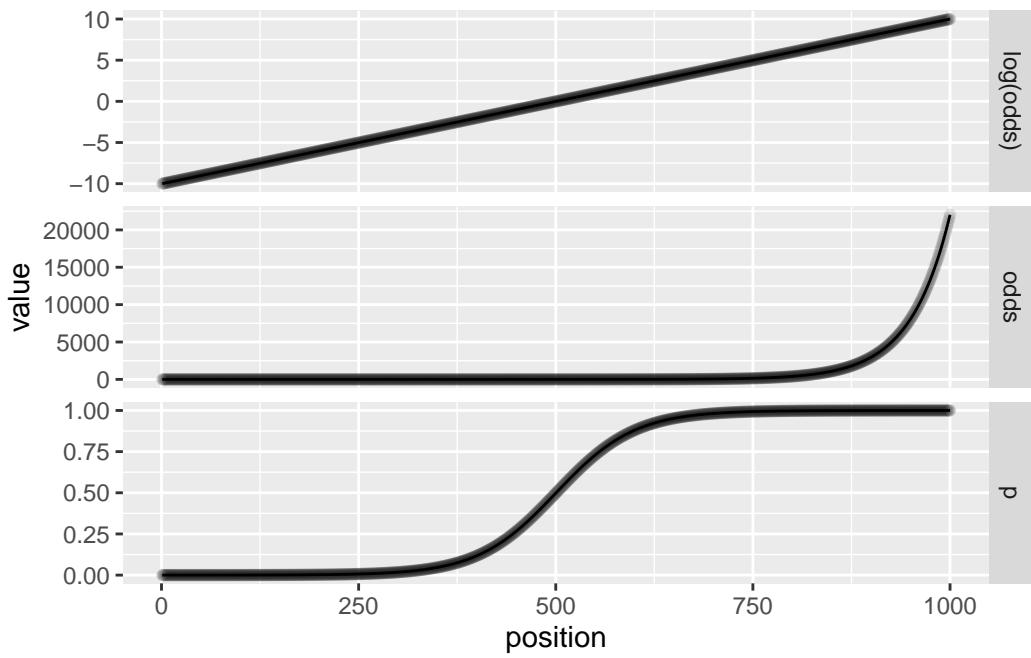
Generalized linear models will be introduced using logistic regression as an example. The data set `infert` contains information were the outcome follows a binomial distribution.

```
pacman::p_load(car, # Anova()
                wrappedtools, tidyverse, ggbeeswarm,
                rpart, rpart.plot,
                pROC # roc() ggroc()
                )
```

16.1 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.



16.2 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 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())
```

16.3 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	0.1820	0.4394
education0-5yrs	parity_grp2	parity_grp1
0.6082	1.4562	2.7048
induced_fone	induced_ftwo or more	spontaneous_f1
1.3591	2.8292	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)
```

16.4 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))

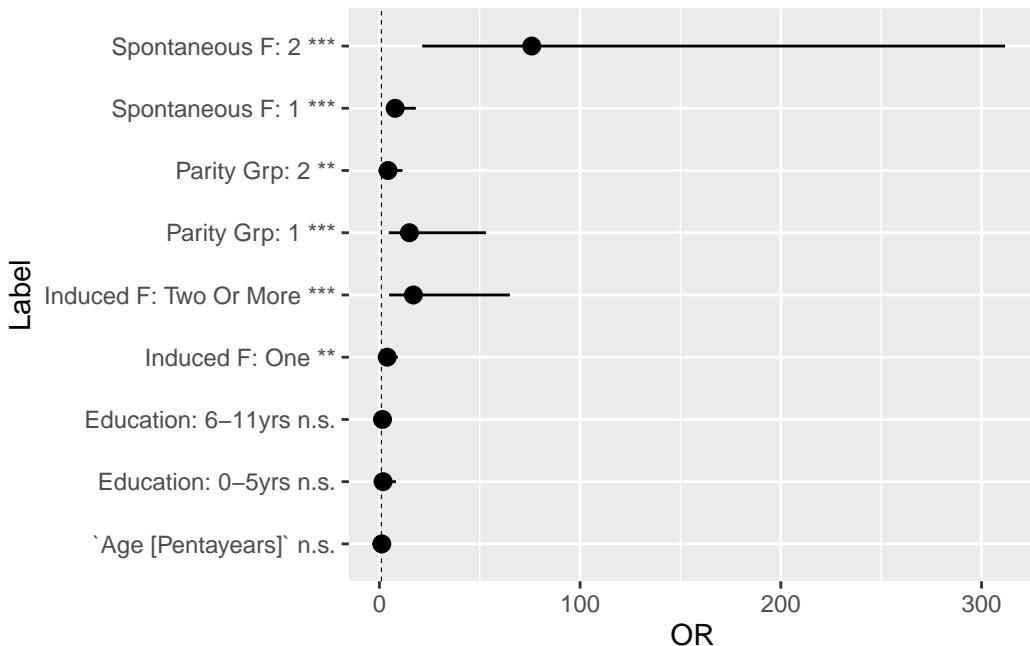
```

16.5 create forest plot

```

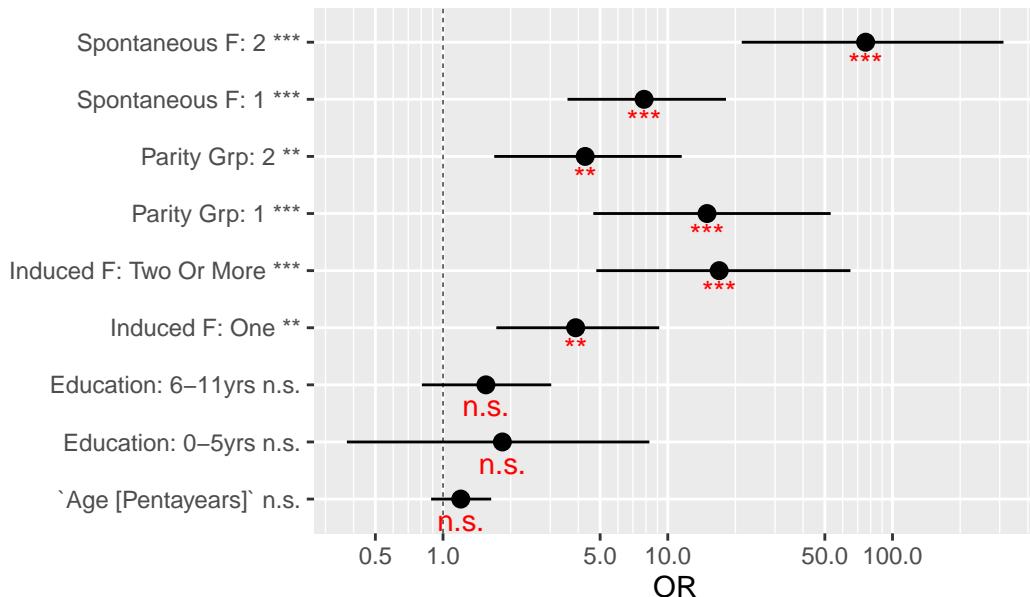
baseplot <-
  ggplot(OR_plotdata, aes(x = Label,y=OR))+ 
  geom_pointrange(aes(ymin=CI_low, ymax=CI_high))+ 
  geom_hline(yintercept = 1,linewidth=.2,linetype=2)+
```

```
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)
```

OddsRatios shown on log-scale



16.6 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

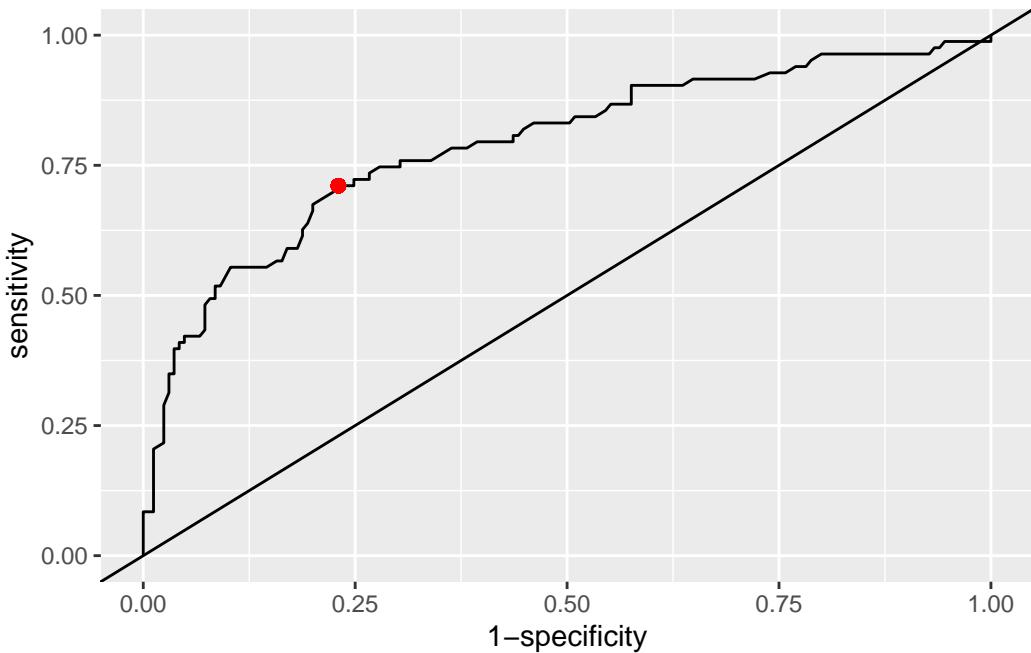
```
youden <- pROC::coords(roc_out,x='best',  
                        best.method='youden')  
youden
```

```
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)+ 

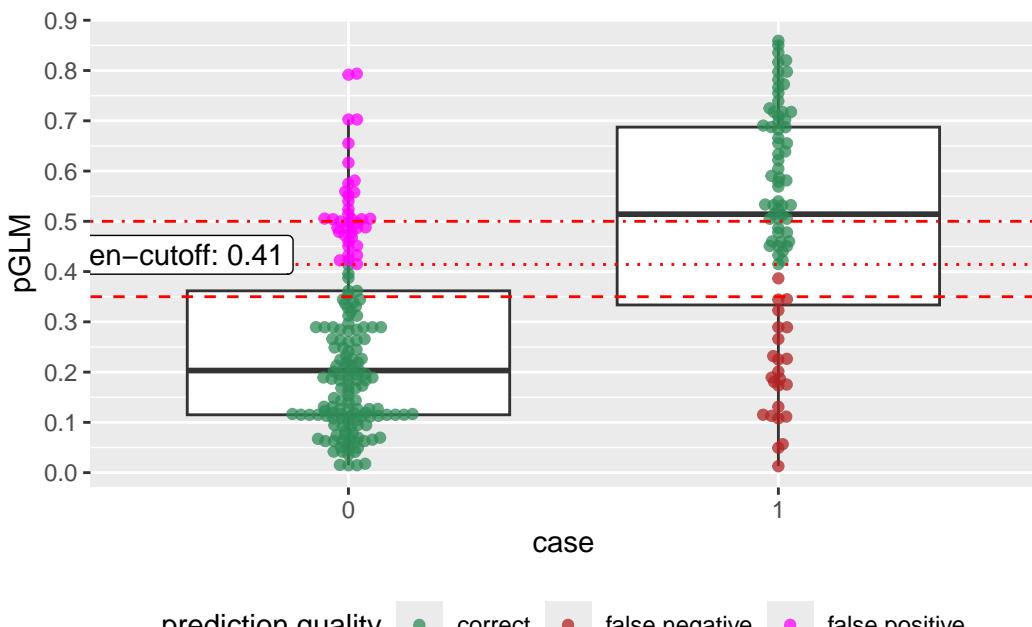
```

```

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(' <br>')
```

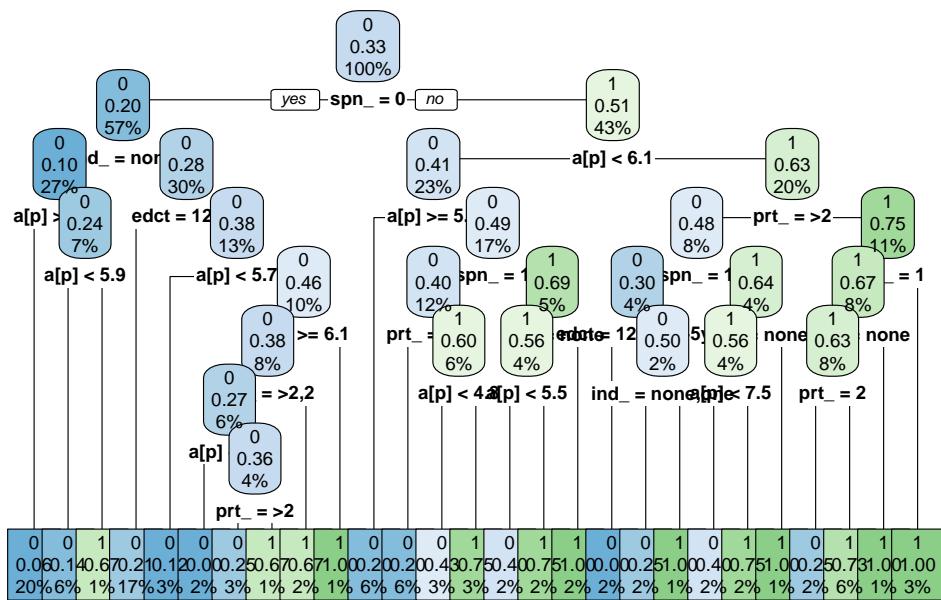
&nbs;

16.7 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"
```

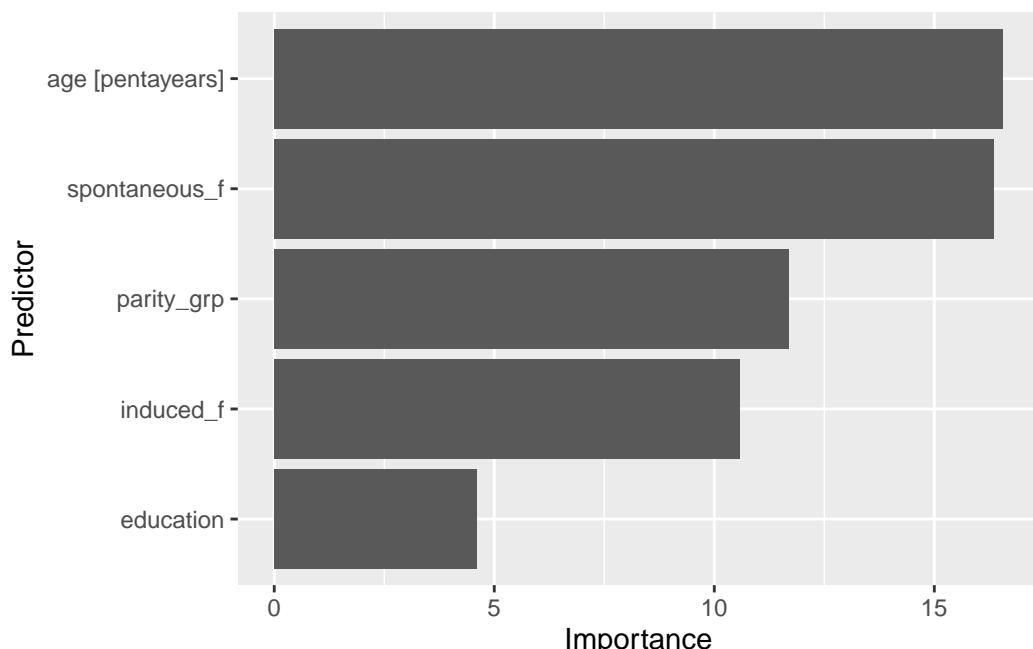
```
predvars <- ColSeeker(namepattern =  
                      c("penta", "edu", "_grp", "_f"))  
rtformula <- paste("case~",  
                    paste(predvars$btricked, collapse = "+"))  
regtree_out<-rpart(rtformula,  
                     minsplit=5, cp=.001,  
                     data=rawdata)  
rpart.plot(regtree_out, type = 2, tweak=2.0, varlen=4, faclen=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 )

```

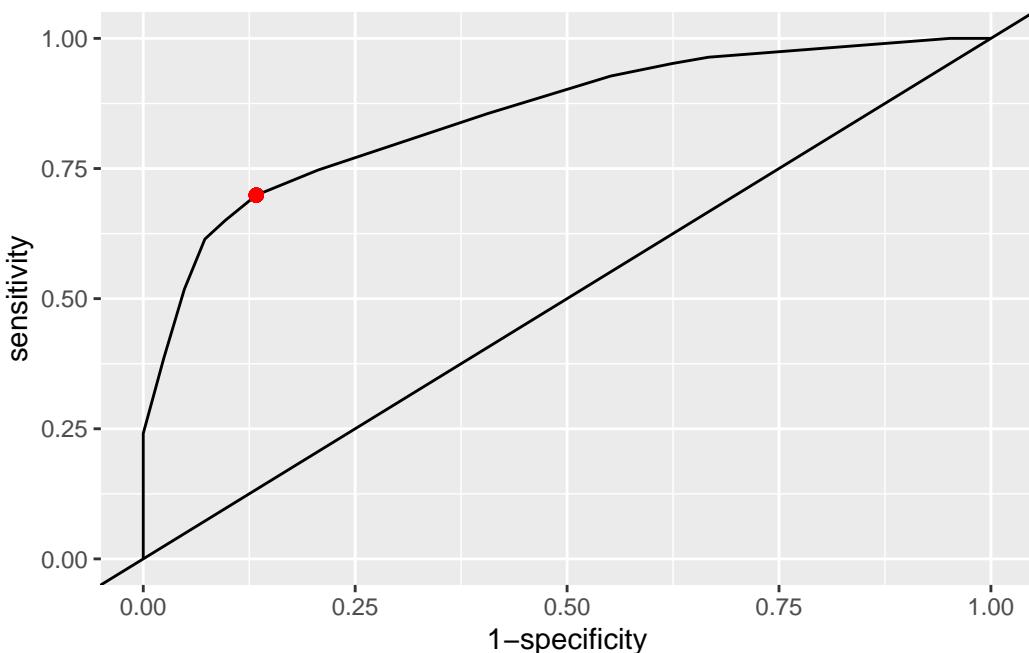
Setting levels: control = 0, case = 1

Setting direction: controls < cases

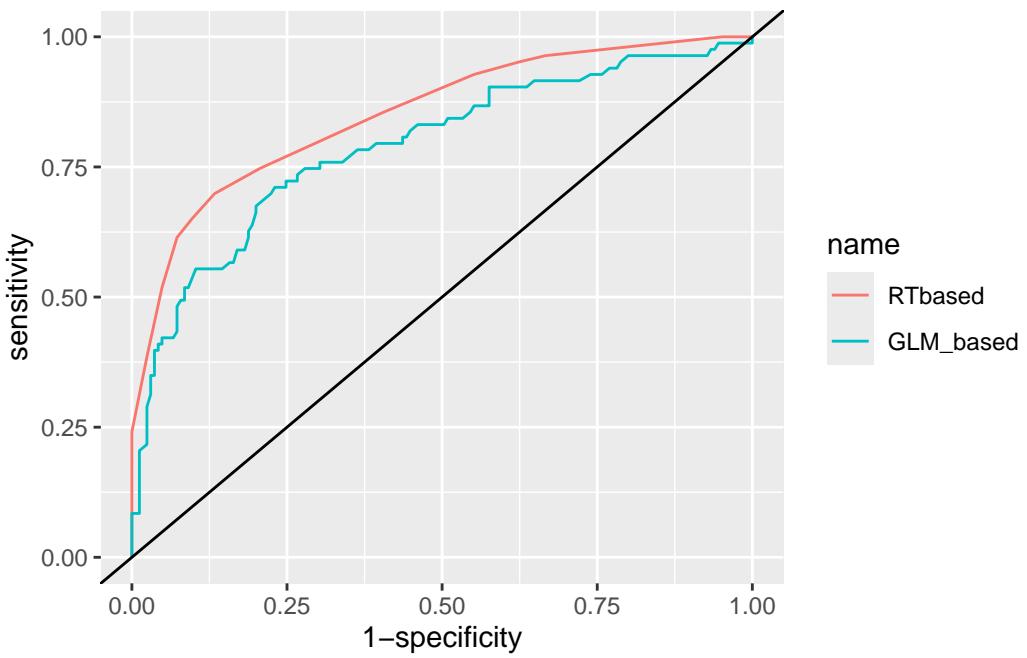
```
youden <- pROC::coords(roc_out_rt, x='best',
                        best.method='youden')
youden
```

	threshold	specificity	sensitivity
1	0.325	0.8666667	0.6987952

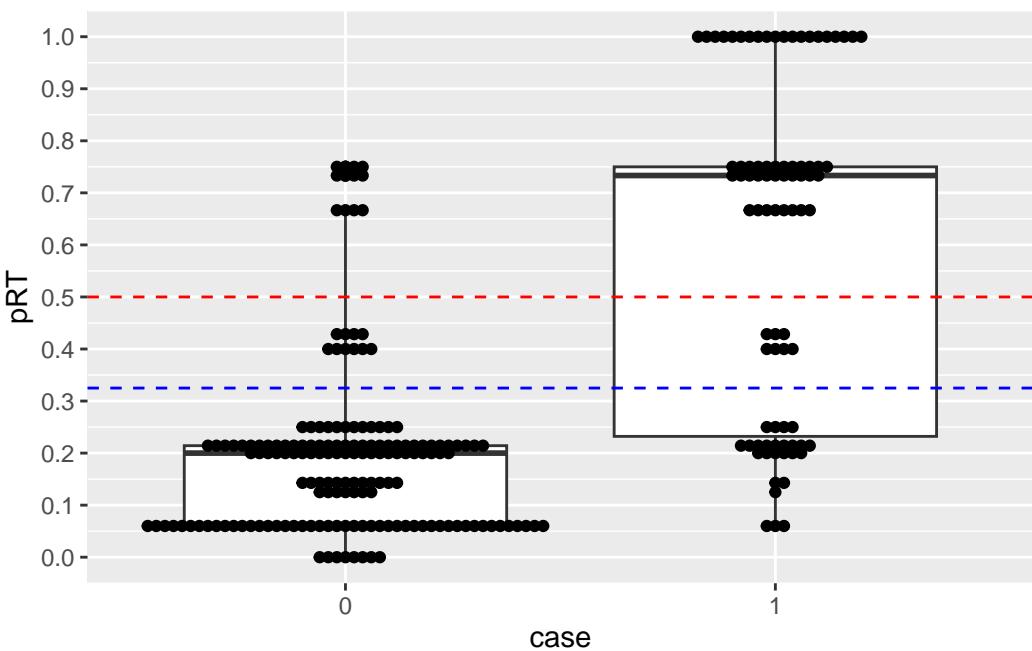
```
ggroc(roc_out_rt, legacy.axes = T) +
  geom_abline(slope = 1, intercept = 0) +
  geom_point(x=1-youden$specificity,
             y=youden$sensitivity, color='red', size=2 )
```



```
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(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)+  

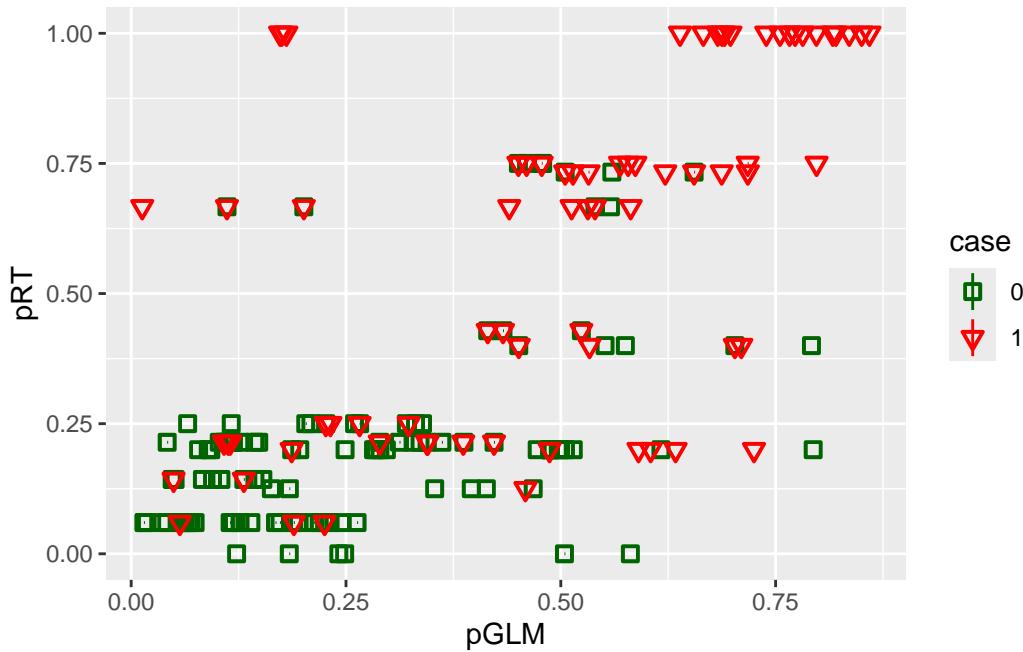
  scale_color_manual(values = c('darkgreen','red'))+  

  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(from = 0,to = 1,by = .1))+  

  geom_hline(yintercept = .5,color='red')

```



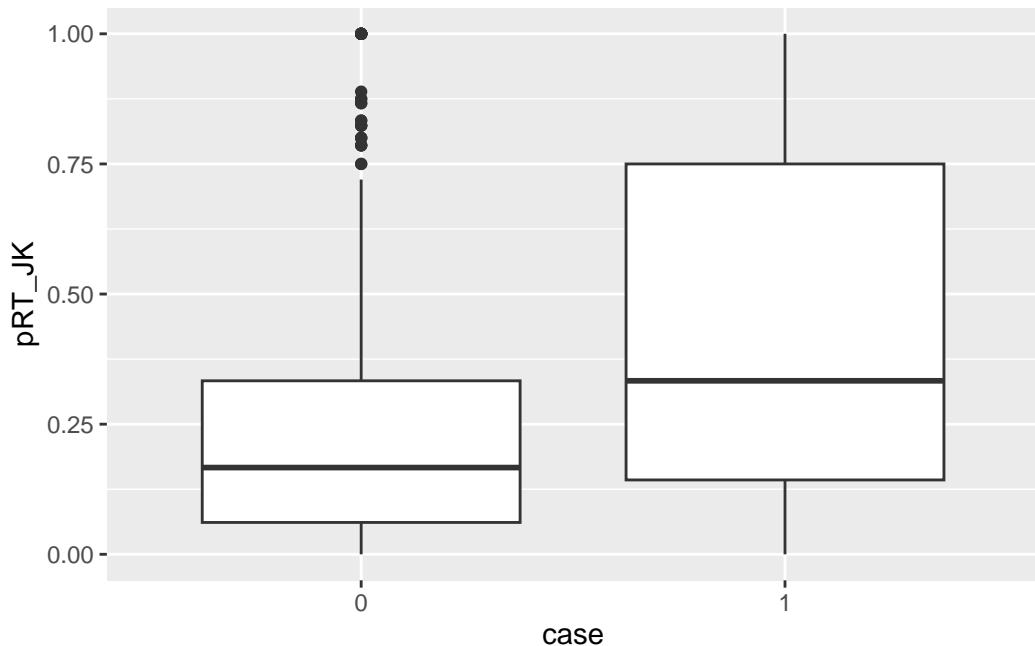
16.8 Jackknife

```

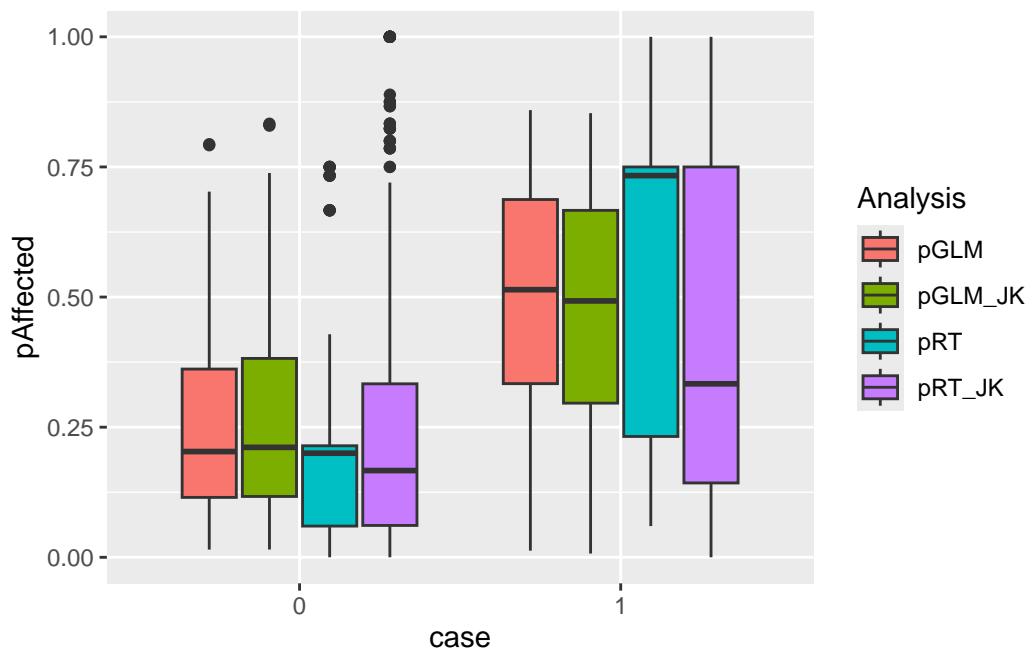
rawdata$pRT_JK <- NA_real_
rawdata$pGLM_JK <- NA_real_
for(pat_i in 1: nrow(rawdata)){
  tempdata <- rawdata[-pat_i,]
  regtree_out_tmp<-rpart(rtformula,
                           minsplit=5,
                           cp=.001, data=tempdata)
  rawdata$pRT_JK[pat_i] <-
    predict(regtree_out_tmp,
           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()
```



17 Machine Learning with R: Basic concepts

17.1 structured vs. unstructured data

17.2 supervised vs. unsupervised methods

17.3 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

17.3.1 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	-	

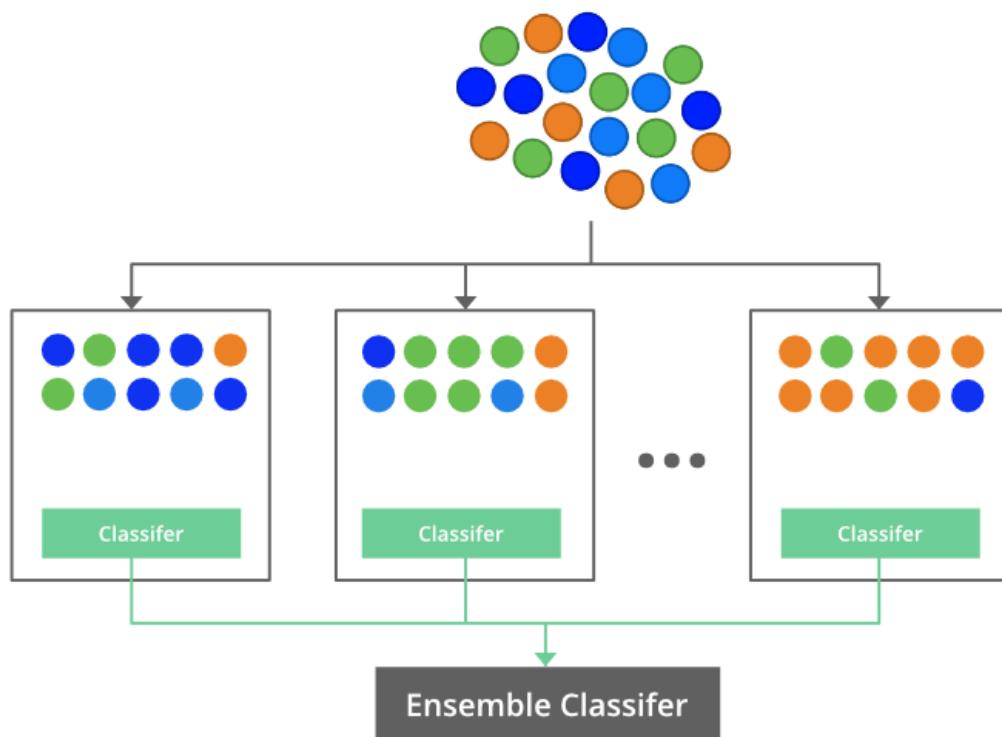
- Permutation 1

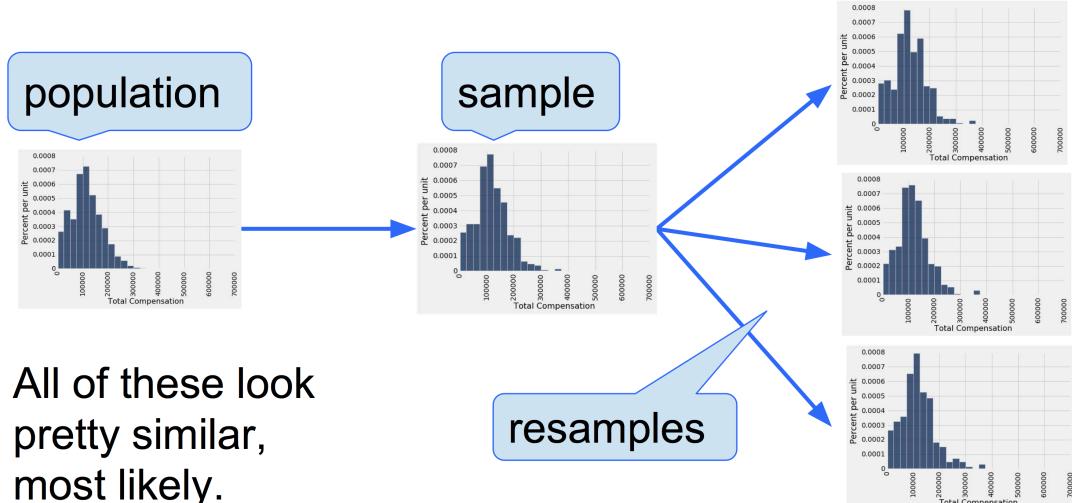
	Class 1	Class 2
Affected	-	
Affected		Exposed
Affected	-	
Control	-	
Control		Exposed

- repeated x times

17.3.2 Bootstrapping

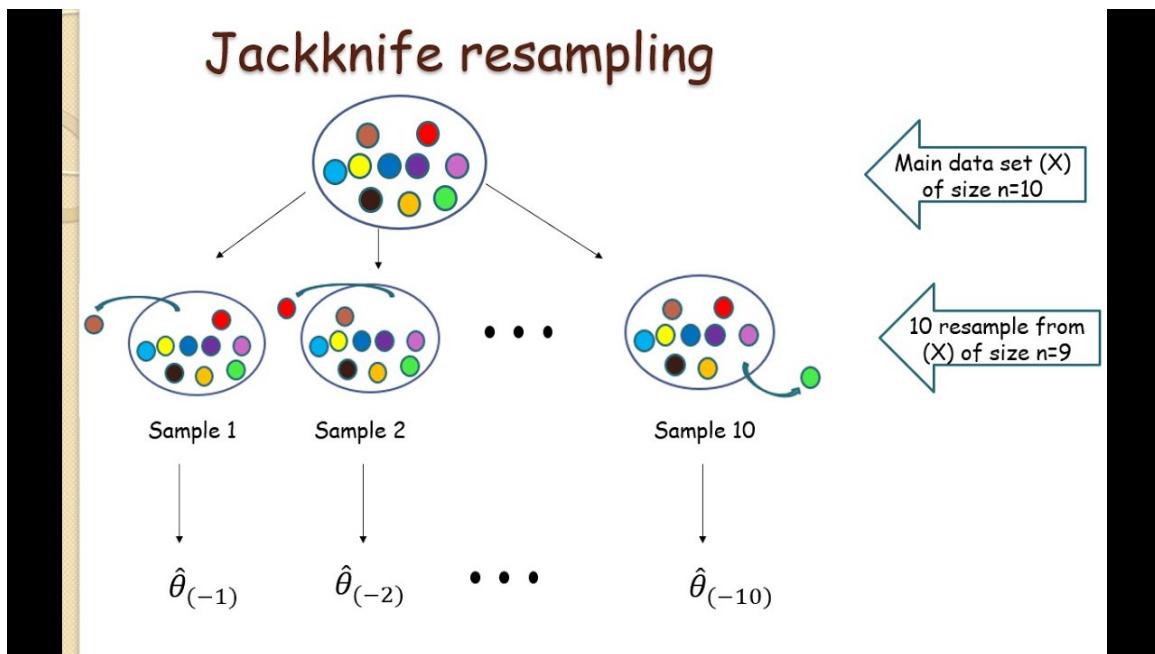
- Bootstrapping selects from the population of observed cases, sampling with replacement.
- Bootstrap is a powerful statistical tool used to quantify the uncertainty of a given model.





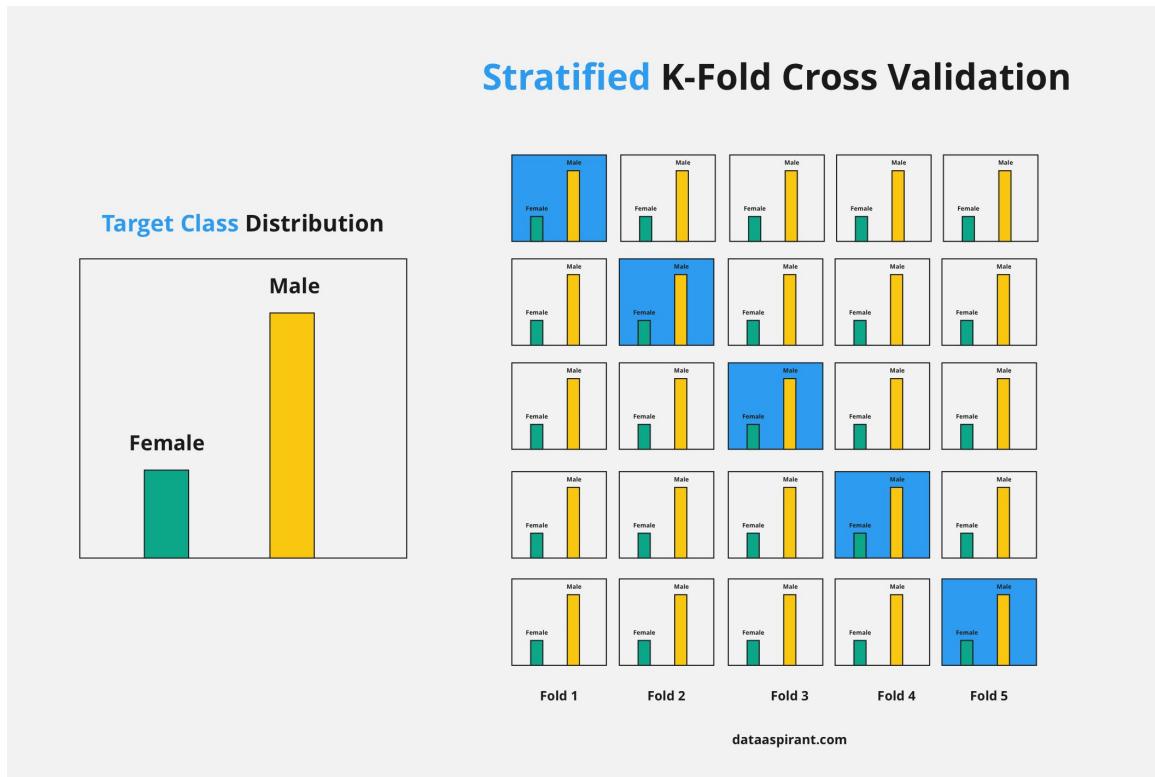
17.3.3 Jackknife

- Jackknife is resampling without replacement
- Creates n samples of size n-1



17.3.4 k-fold CV

- Splitting sample into k groups
- One group for testing,
- k-1 groups for training
- Repeated k times, no resampling



18 Markov Chain Monte Carlo: Metropolis algorithm simulation

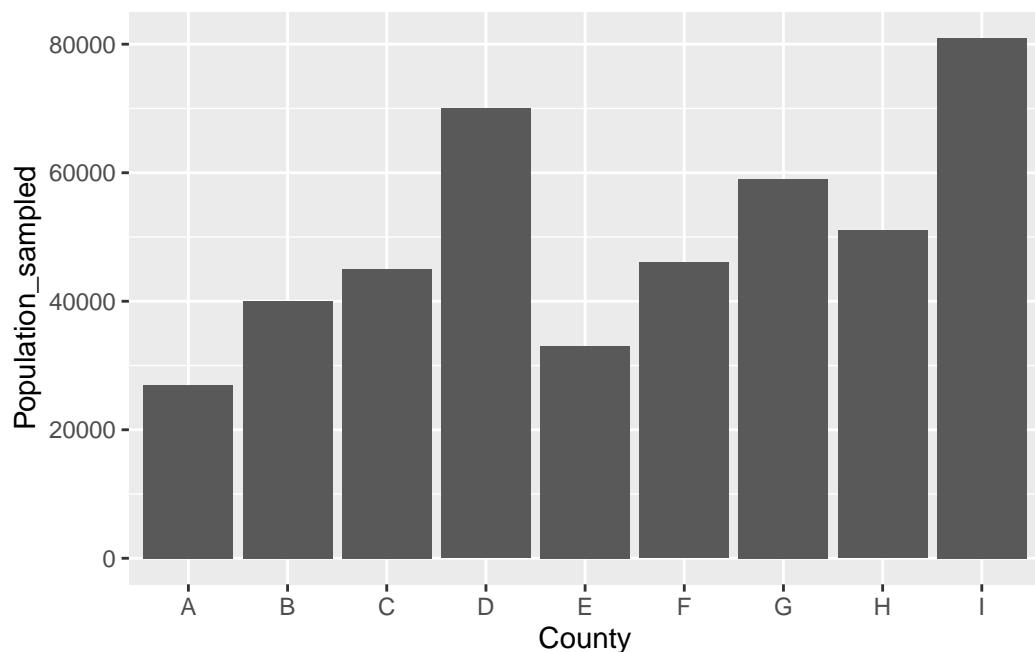
based on [Markov Chain Monte Carlo | Columbia Public Health](#) with slightly changed assumptions (10 counties, pre-defined populations either systematically chosen or randomized) and these rules:

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()
```



18.1 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
```

18.2 Data structures for simulation

```
moves <- tibble(move=seq_len(n_moves),
                 position=NA_integer_)
moves$position[1] <- start_county
```

18.3 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....: 23.11 sec elapsed

18.4 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))+  

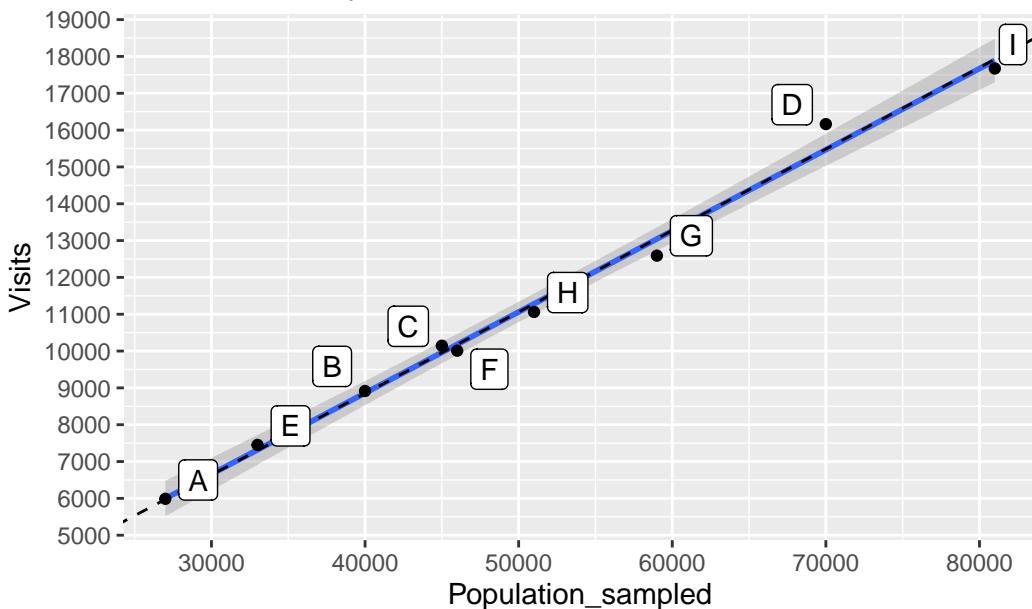
  scale_y_continuous(breaks=seq(0,10^5,10^3))+  

  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))+  

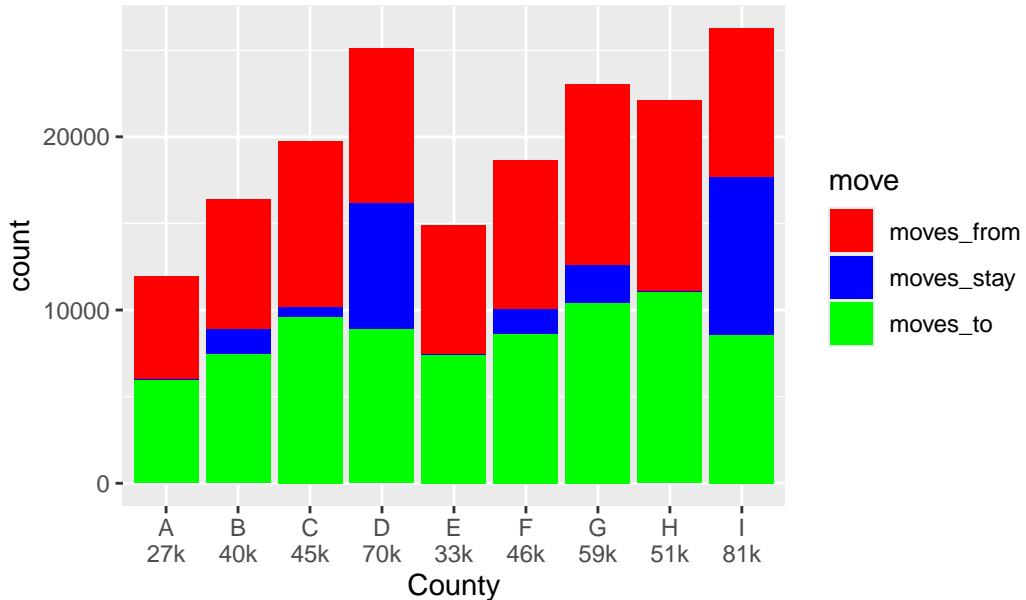
  geom_col()  

  scale_fill_manual(values=c('red','blue','green'))+  

  ggtitle('Moves from, to, and stay')

```

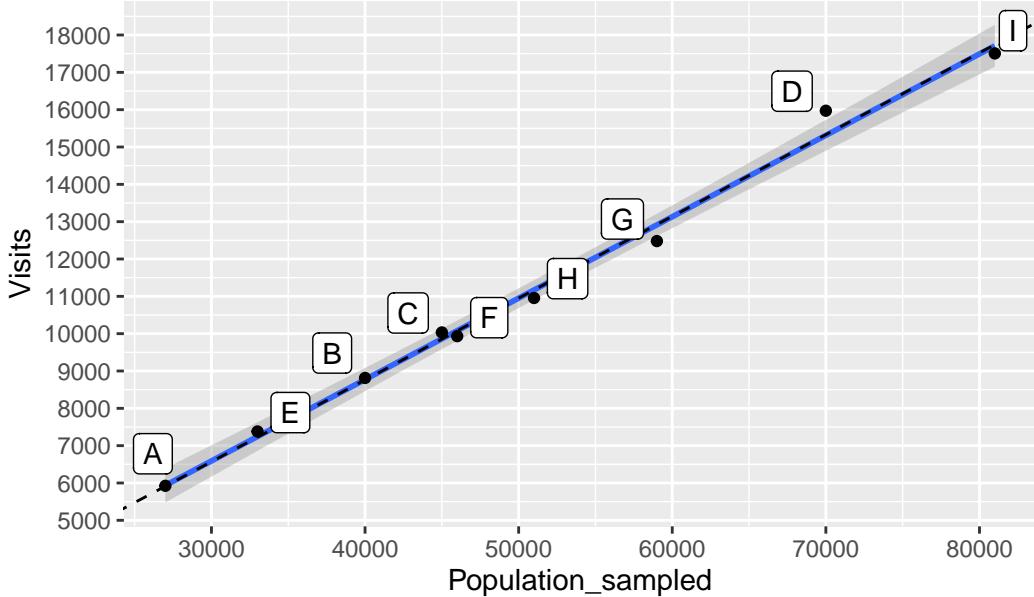
Moves from, to, and stay



```
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

```

```

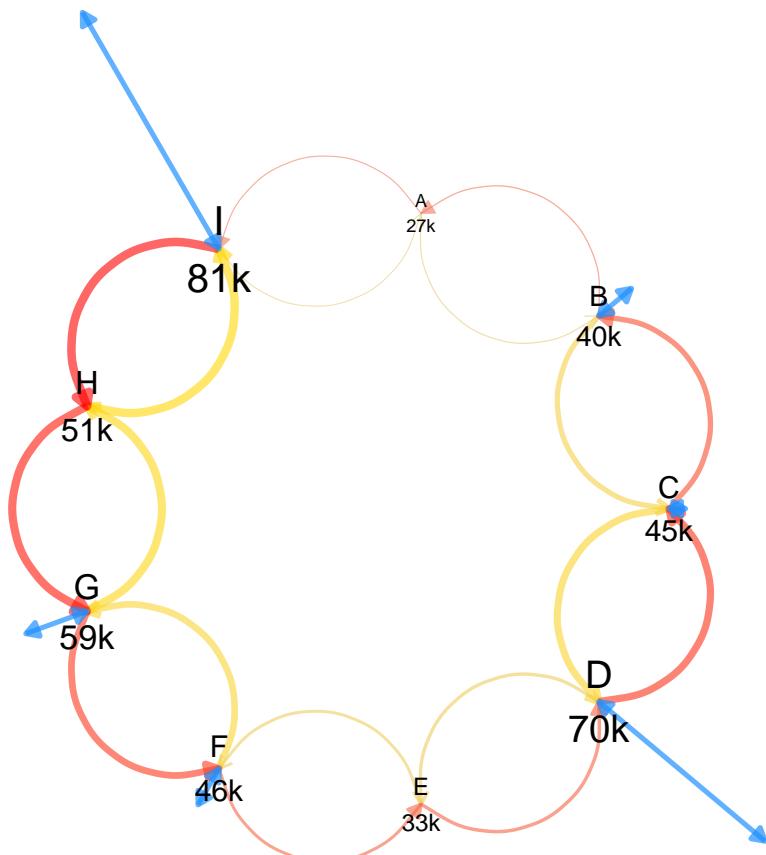
# geom_circle(aes(r=1,x0=0,y0=0),
#             color="darkorange2")+
geom_curve(data=arrow_data |>
    filter((from<to & !(to==9 & from==1)) | (to==1 & from==9)),
    aes(x=from_x,y=from_y,
        xend=to_x,yend=to_y,
        color=count, linewidth=abs(count)),
    arrow=arrow(length=unit(0.1,"inches")),
    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))+  

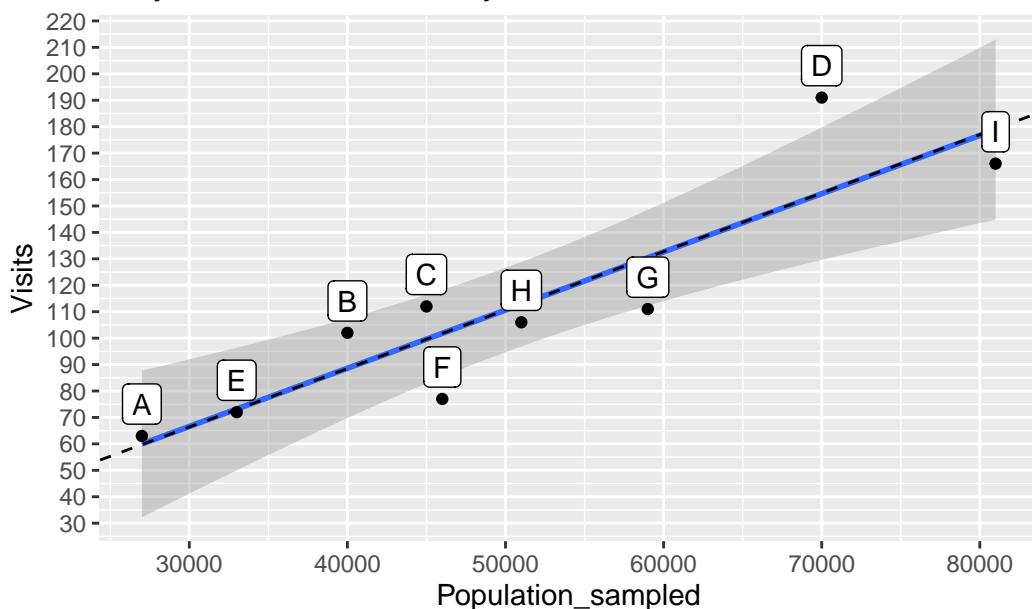
scale_y_continuous(breaks=seq(0,10^5,10^1))+  

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))+  

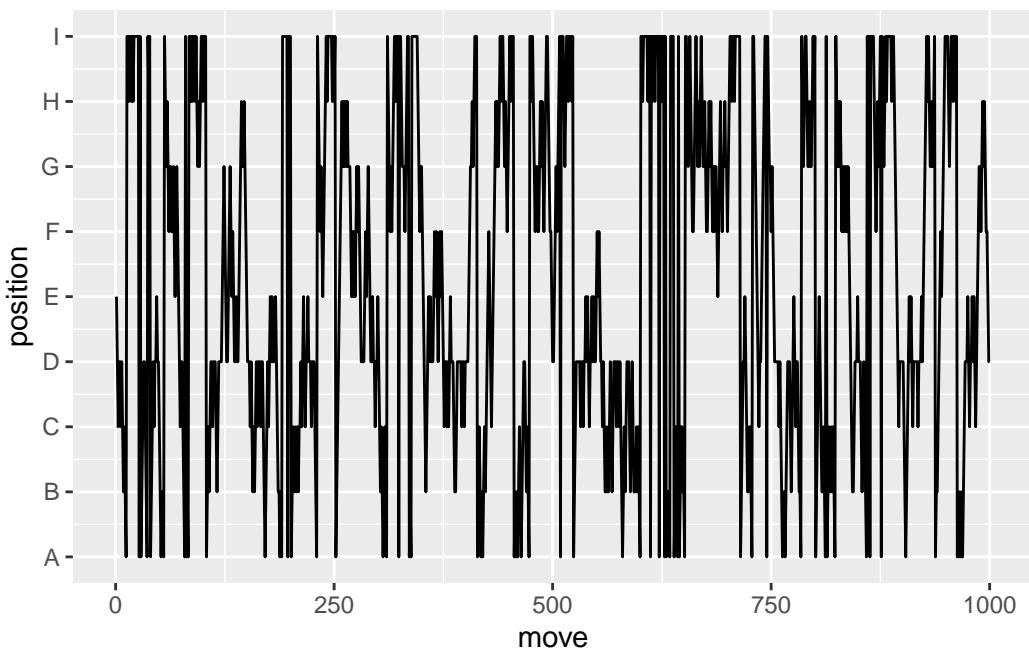
  # geom_point() +  

  geom_line() +  

  scale_y_continuous(breaks=1:9,  

                     labels = LETTERS[1:9])

```



```

moves |>
  filter(move<=100) |>
  ggplot(aes(move,position))+  

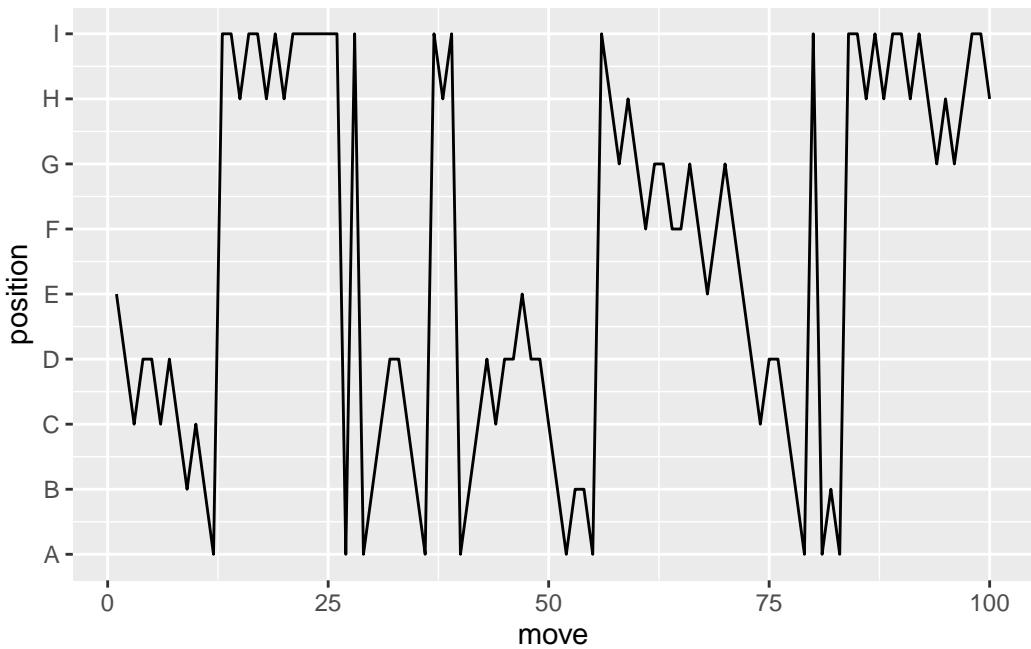
  # geom_point() +  

  geom_line() +  

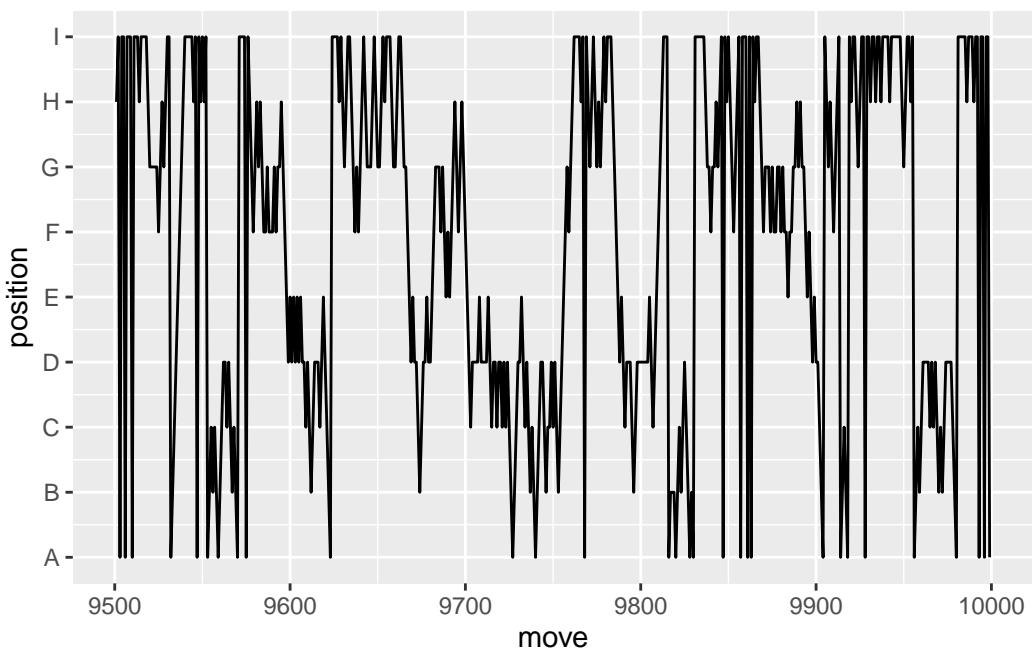
  scale_y_continuous(breaks=1:9,  

                     labels = LETTERS[1:9])

```

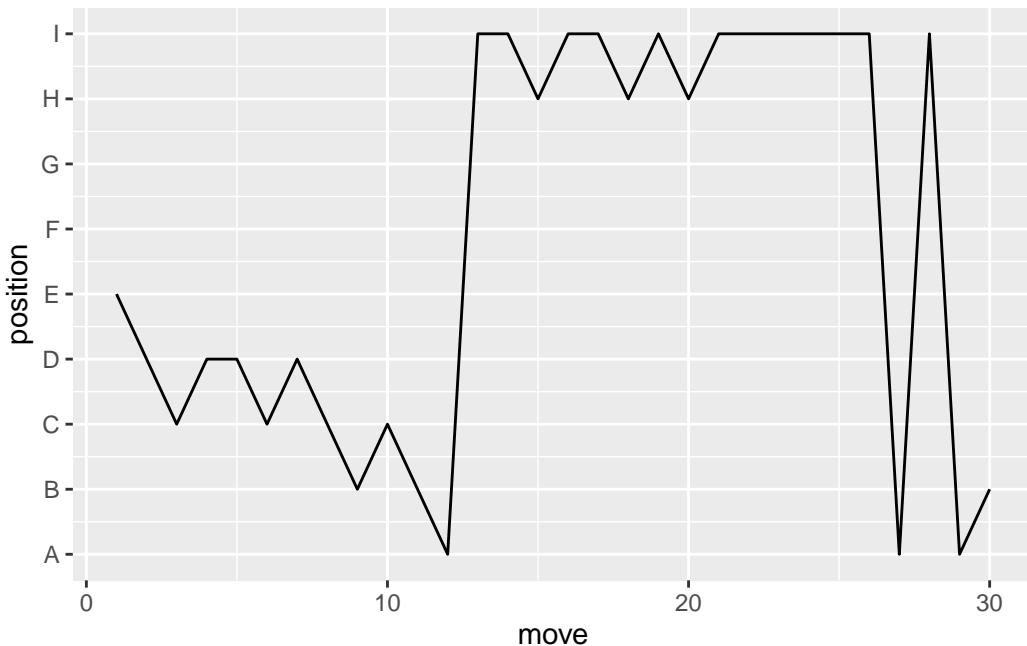


```
moves |>
  filter(move>9500, move<10000) |>
  ggplot(aes(move,position))+
  # geom_point()+
  geom_line()+
  scale_y_continuous(breaks=1:9,
                     labels = LETTERS[1:9])
```

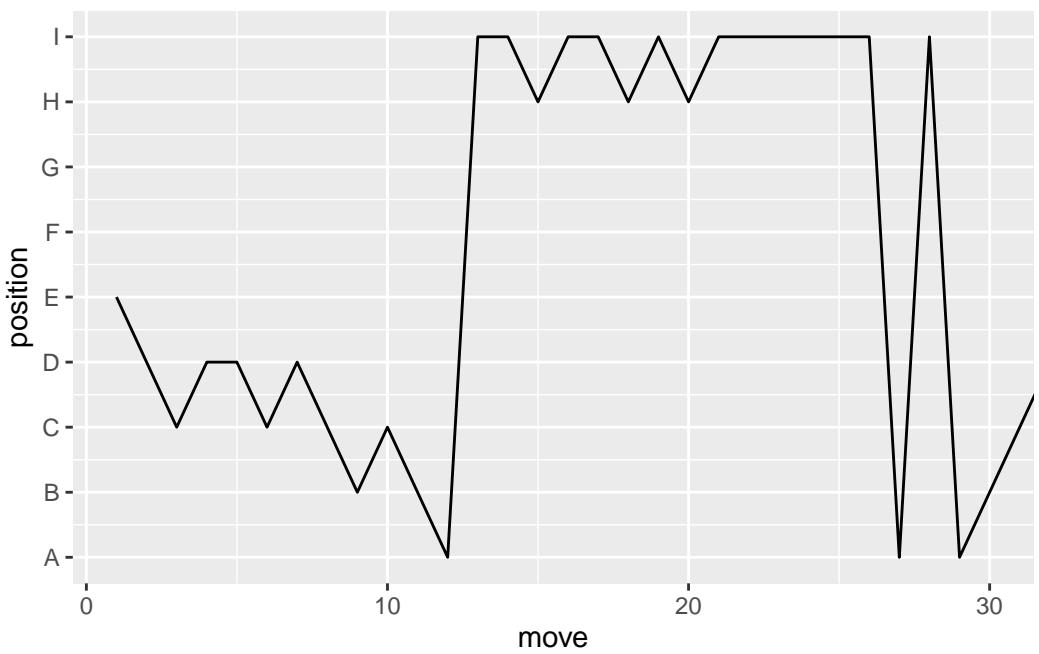


```
moves |>
  # filter(move<=n_burnin) |>
  ggplot(aes(move,position))+
  # geom_point()+
  geom_line()+
  scale_x_continuous(limits = c(1,30))+ 
  scale_y_continuous(breaks=1:9,
                     labels = LETTERS[1:9])
```

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))
```



19 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
  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.
```

19.1 Data preparation

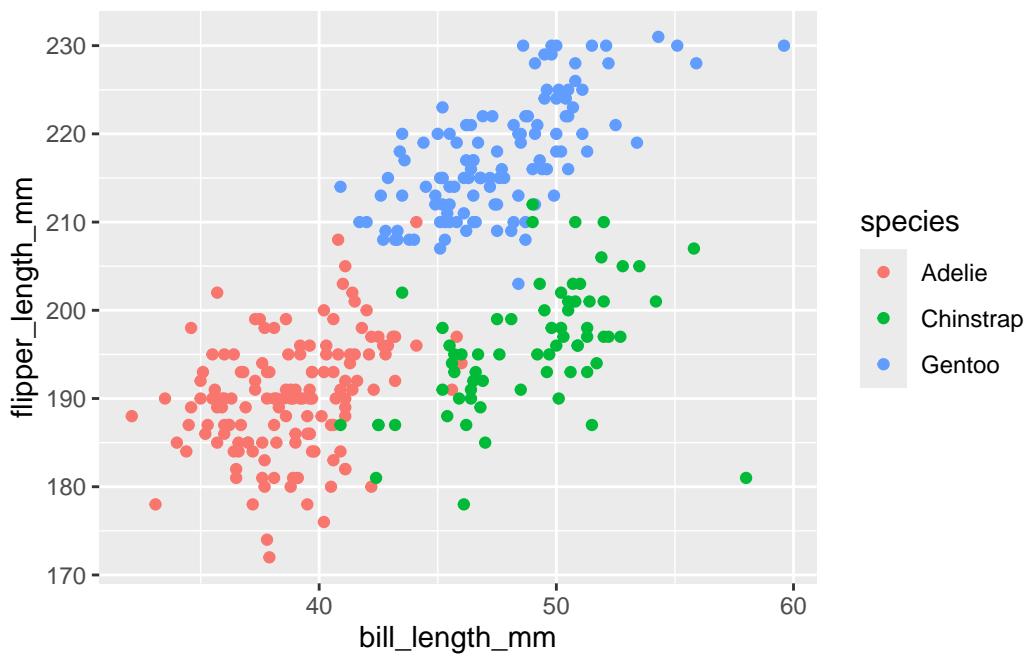
In the penguin data set, the four numerical variables related to body are `bill_length_mm`, `bill_depth_mm`, `flipper_length_mm`, and `body_mass_g`. We will use the first two variables to demonstrate the use knn.

```
rawdata <- penguins |>
  drop_na()
rawdata <- mutate(rawdata,
  ID=paste('P', 1:nrow(rawdata))) |>
  select(ID, everything())
predvars <- ColSeeker(rawdata, 'length')
```

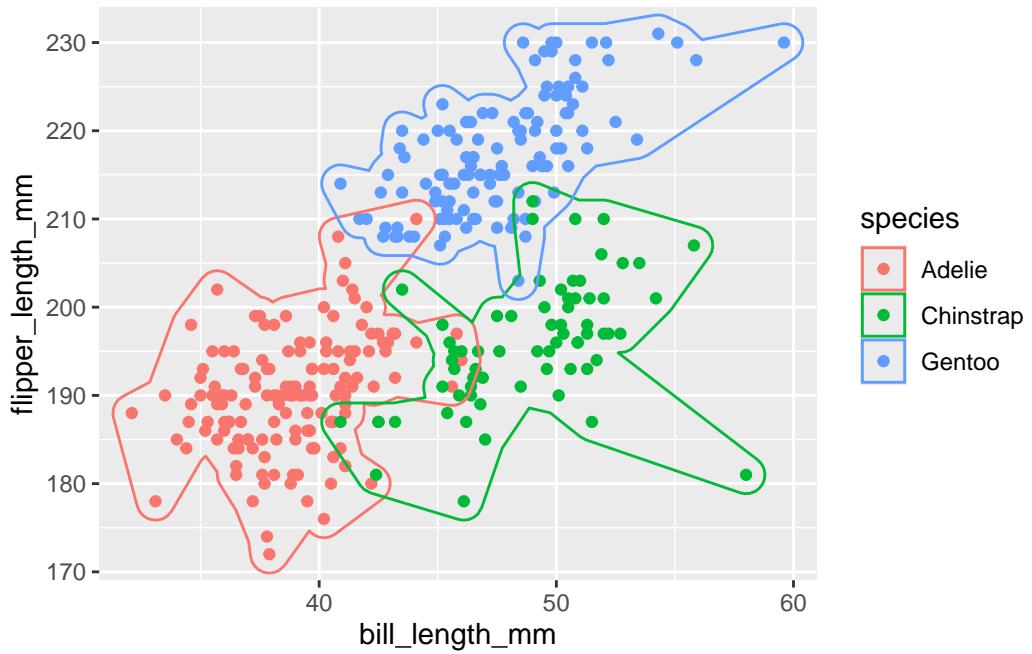
19.2 Exploratory plots

Package `ggforce` provides functions to plot convex hulls, ellipses, and zoomed facets. This is helpful in exploring group separation across 2 dimensions.

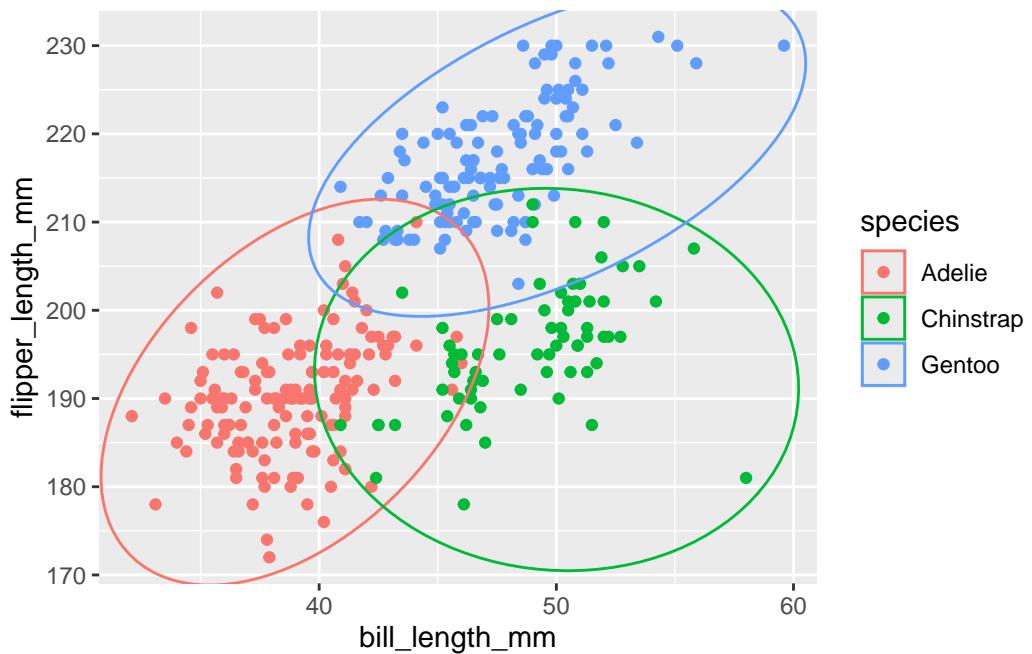
```
rawplot <-
  ggplot(rawdata,
    aes(.data[[predvars$names[1]]],
        .data[[predvars$names[2]]]),
    color=species))+
  geom_point()
rawplot
```



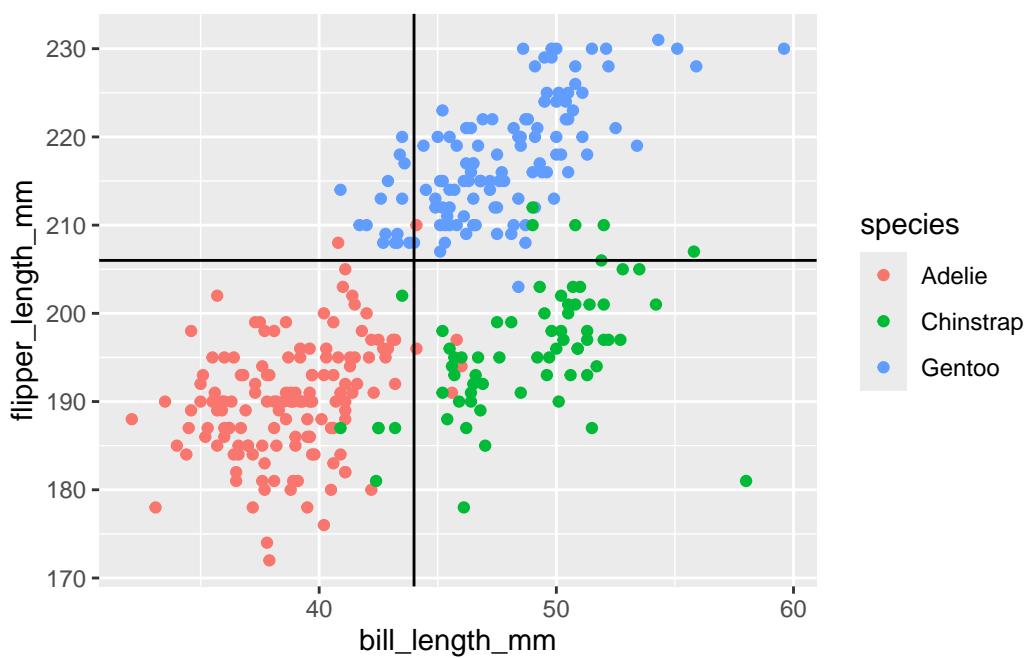
```
rawplot+
  geom_mark_hull(expand = unit(2.5, 'mm'))
```



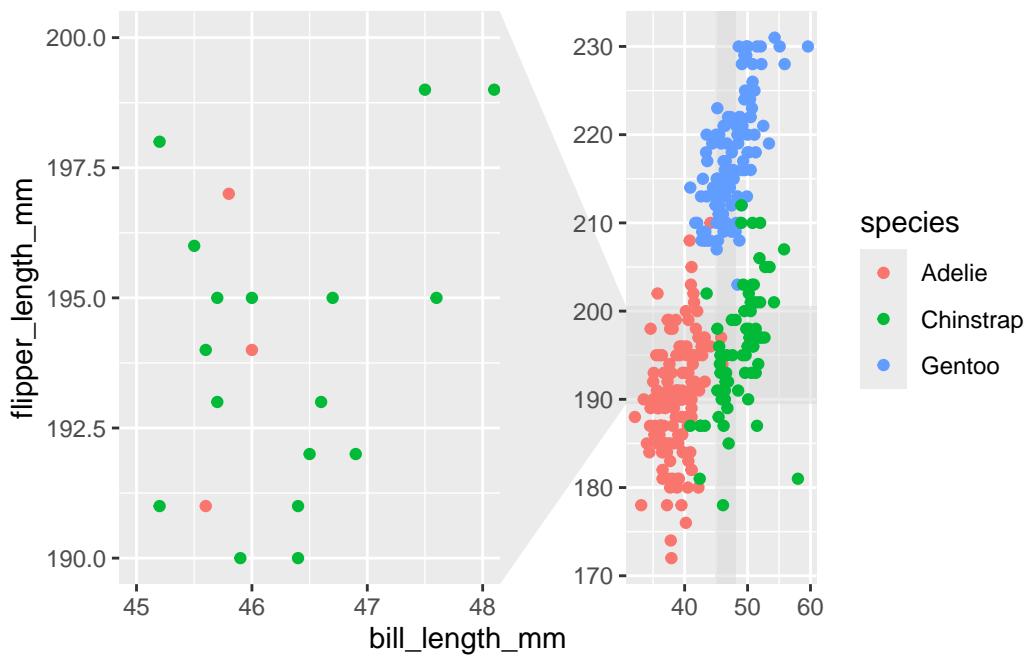
```
rawplot+
  geom_mark_ellipse(expand = unit(2.5, 'mm'))
```



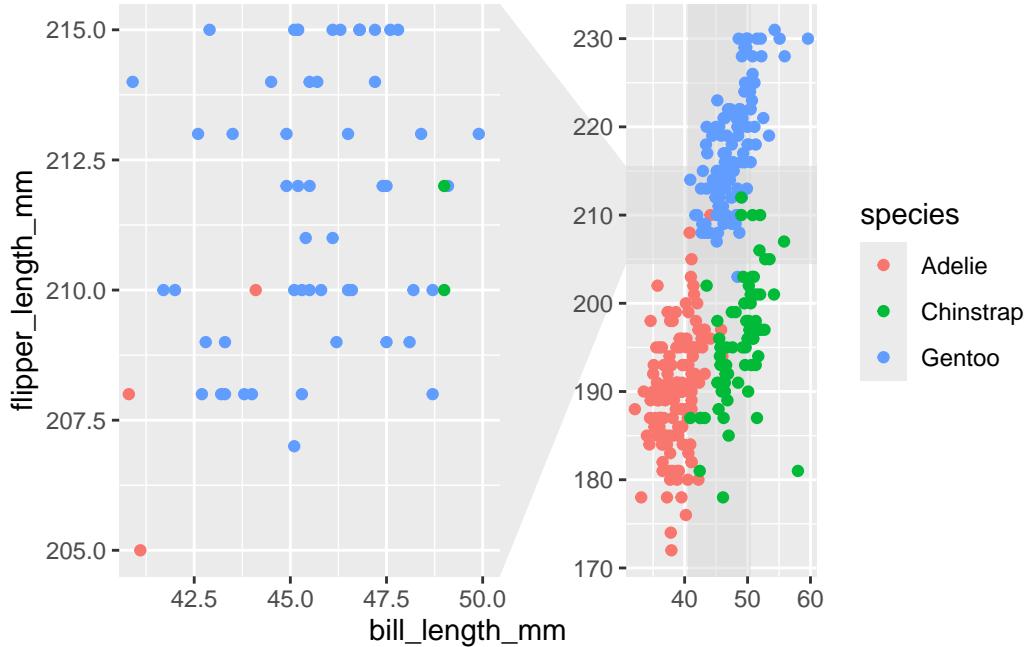
```
rawplot+
  geom_hline(yintercept = 206) +
  geom_vline(xintercept = 44)
```



```
rawplot+
  facet_zoom(xlim = c(45,48),
              ylim = c(190,200))
```



```
rawplot+
  facet_zoom(xlim = c(41,50),
              ylim = c(205,215))
```



19.3 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.

19.3.1 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')
```

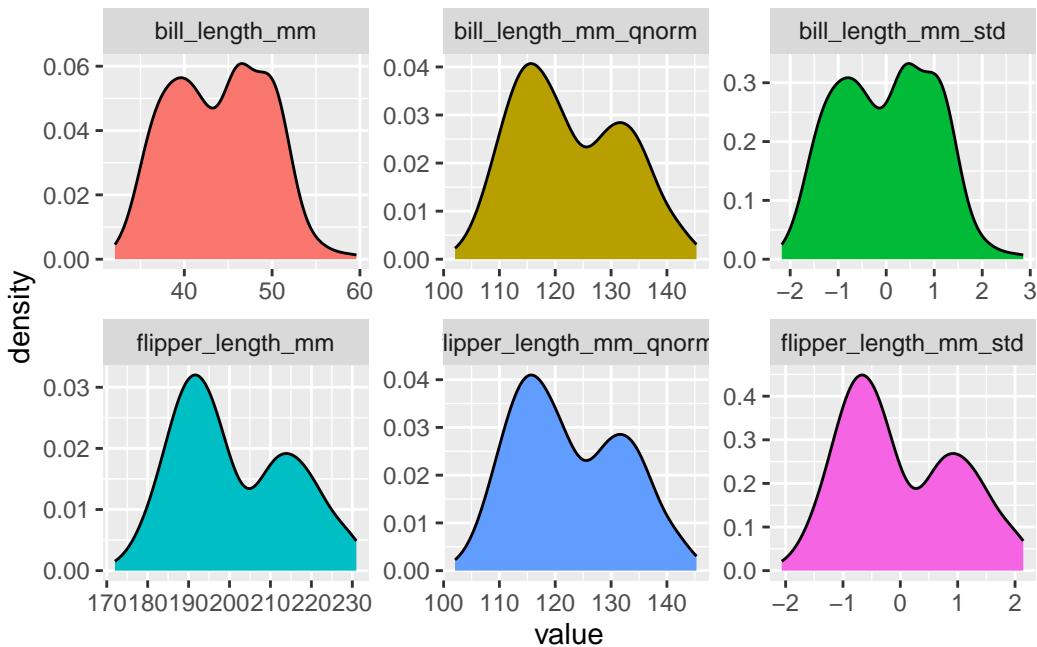
19.3.2 preprocessCore function normalize.quantiles

For ...omics data, bioconductor provides functions for ML including preprocessing. Here we use the `normalize.quantiles` function from the `preprocessCore` package.

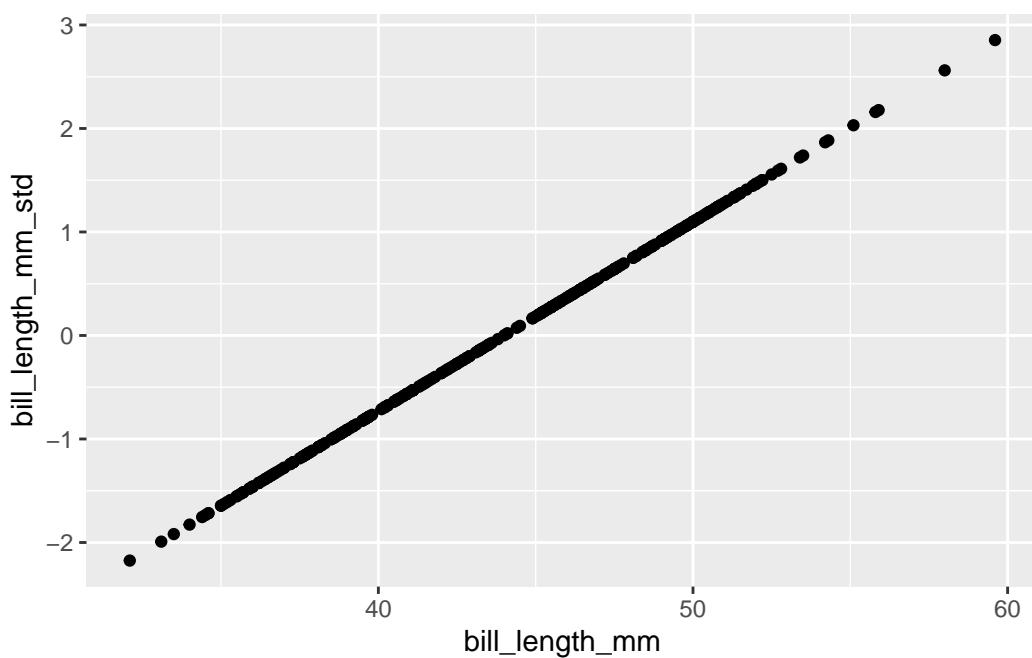
```
rawdata <- rawdata |>
  select(predvars$names) |>
  as.matrix() |>
  preprocessCore::normalize.quantiles(keep.names = TRUE) |>
  as_tibble() |>
  rename_with(~paste0(., '_qnorm')) |>
  bind_cols(rawdata)
```

19.3.3 Visual comparison of the scaled data

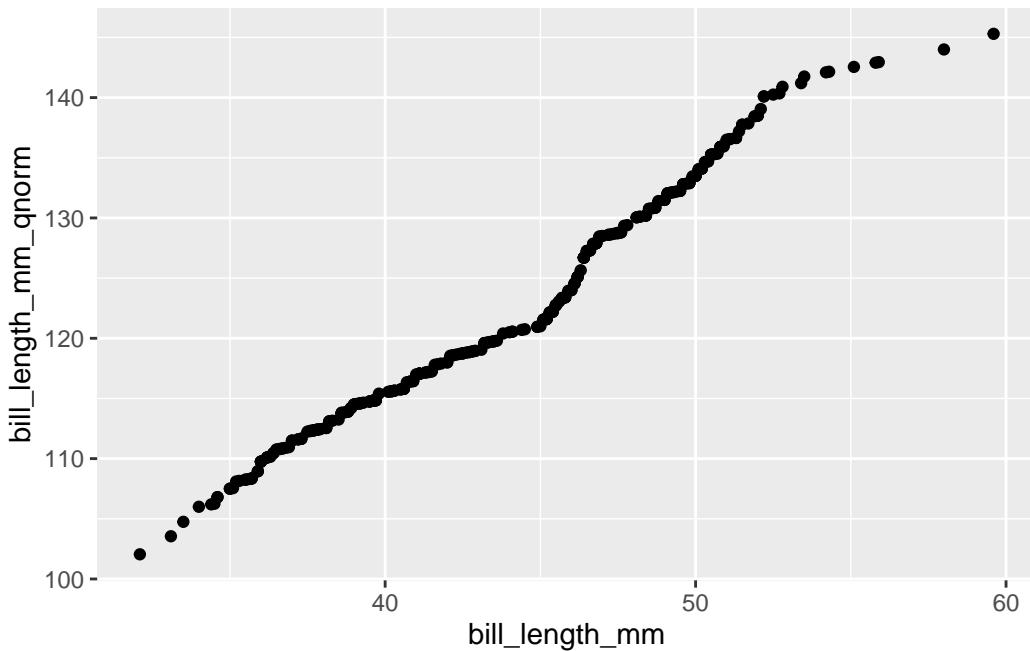
```
rawdata |>
  select(contains('length')) |>
  pivot_longer(everything()) |>
  ggplot(aes(value, fill=name)) +
  geom_density() +
  facet_wrap(facets = vars(name), scales='free') +
  guides(fill="none")
```



```
rawdata |>  
  ggplot(aes(bill_length_mm,bill_length_mm_std))+  
    geom_point()
```



```
rawdata |>  
  ggplot(aes(bill_length_mm,bill_length_mm_qnorm))+  
    geom_point()
```



19.4 Modelling

19.4.1 Definition of predictor variables (IV)

```
predvars_std <- ColSeeker(namepattern = '_std')
```

19.4.2 Data splitting

```
set.seed(2025)
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)
```

19.4.3 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)
```

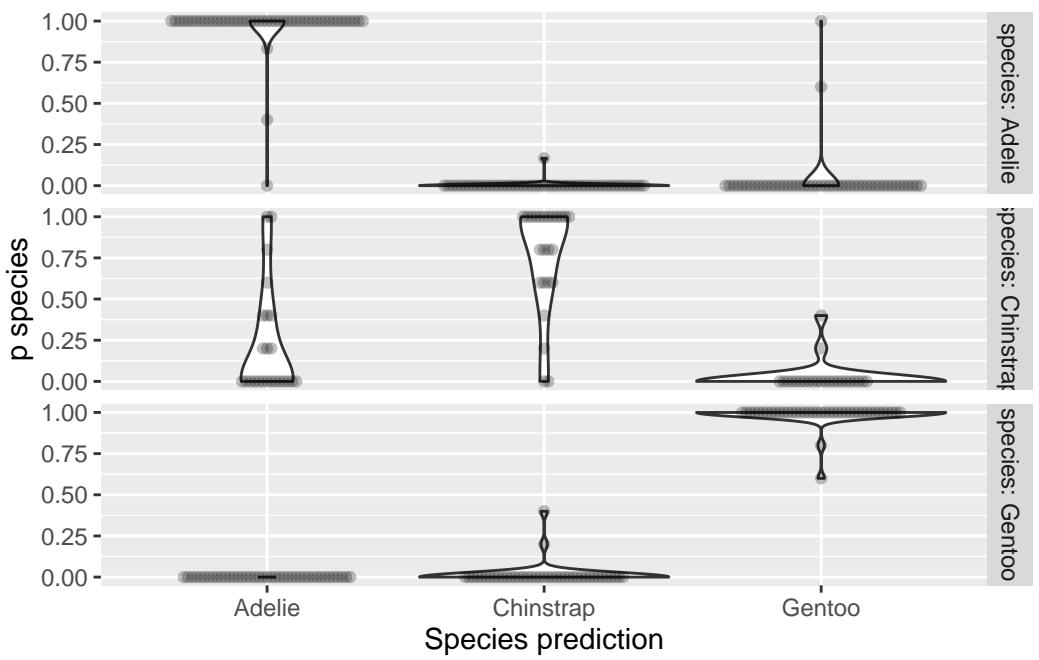
```
chr [1:115] "Adelie" "Adelie" "Adelie" "Adelie" "Adelie" "Adelie" "Adelie" ...
- attr(*, "prob")= num [1:115, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
..- attr(*, "dimnames")=List of 2
... .:$ : NULL
... .:$ : chr [1:3] "Adelie" "Chinstrap" "Gentoo"
```

```
head(train_out)
```

```
[1] "Adelie" "Adelie" "Adelie" "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')
```



19.4.4 Model evaluation

```
CrossTable(train_res$predicted,  
          train_res$species, prop.chisq = FALSE, prop.t = FALSE,  
          format = 'SAS')
```

Cell Contents					
	train_res\$species				
train_res\$predicted	Adelie	Chinstrap	Gentoo	Row Total	
Adelie	48	4	0	52	
	0.923	0.077	0.000	0.452	
	0.960	0.167	0.000		
Chinstrap	0	20	0	20	
	0.000	1.000	0.000	0.174	
	0.000	0.833	0.000		
Gentoo	2	0	41	43	
	0.047	0.000	0.953	0.374	
	0.040	0.000	1.000		
Column Total	50	24	41	115	
	0.435	0.209	0.357		

19.4.5 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()
```

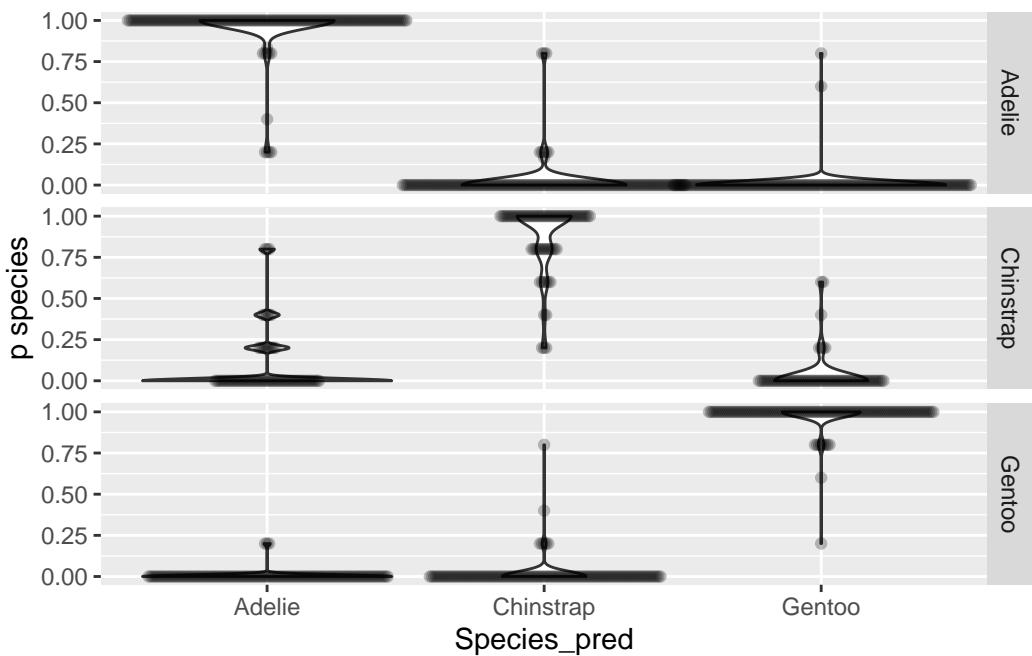
19.4.6 Evaluation of the alternative approach

```
yardstick::accuracy(data = pred_all, truth=species, estimate=predicted)

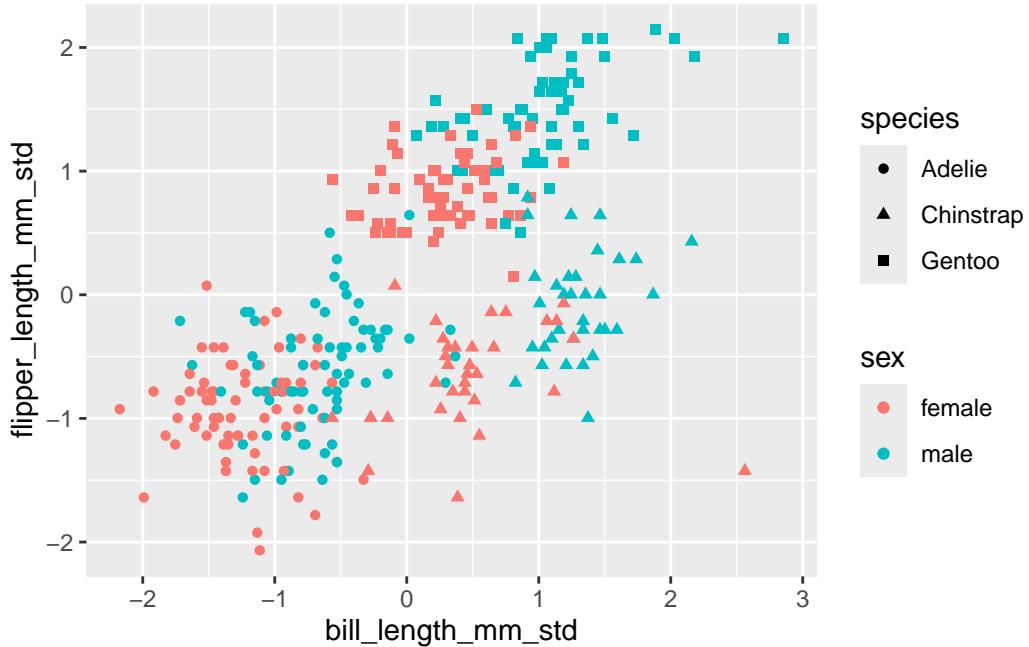
# A tibble: 1 x 3
  .metric   .estimator .estimate
  <chr>     <chr>        <dbl>
1 accuracy  multiclass  0.967

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()
```



19.4.7 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.

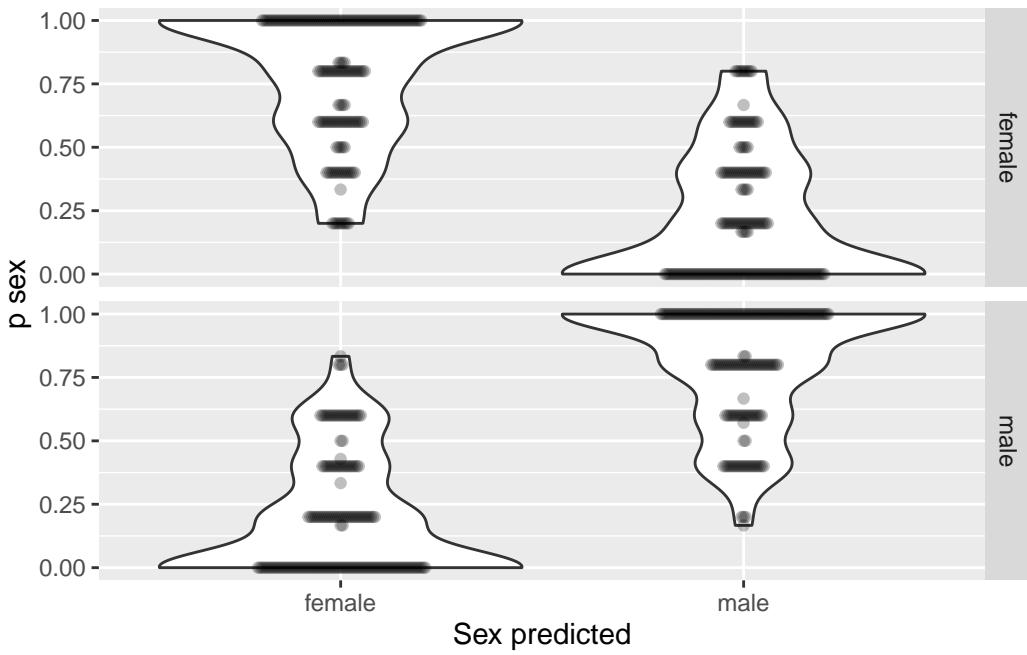
```

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%		

20 Regression and classification trees / Random forrests

20.1 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())
```

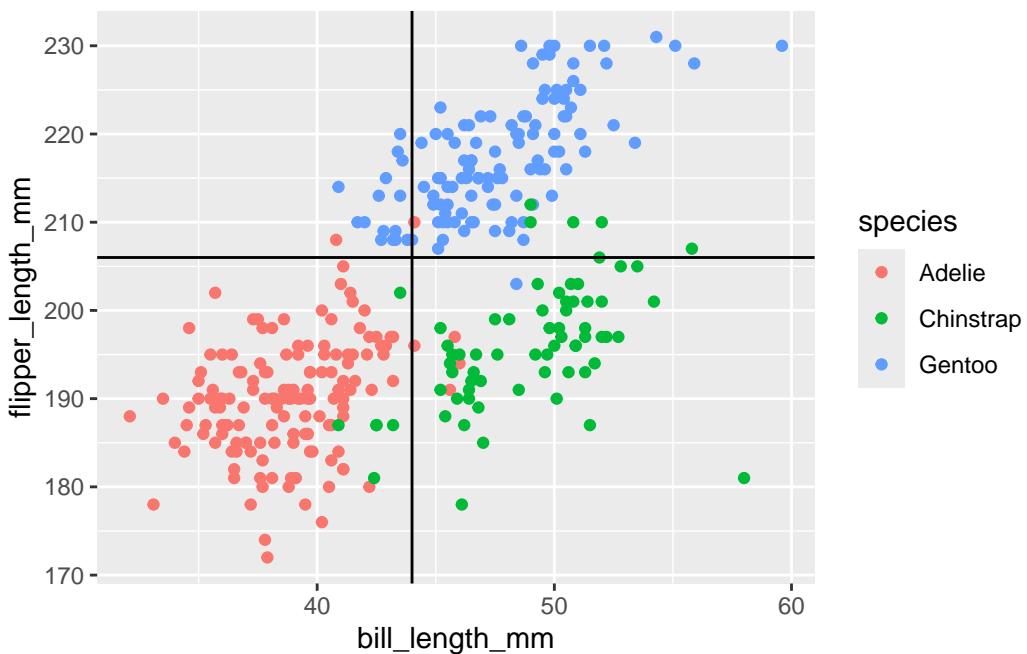
20.1.1 Graphical exploration

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)

```



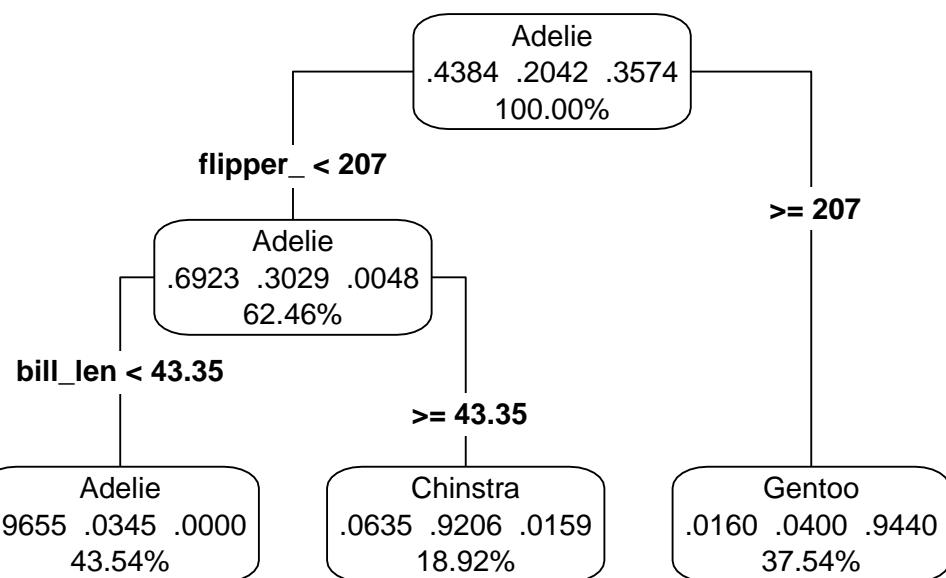
20.1.2 Modelling

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)

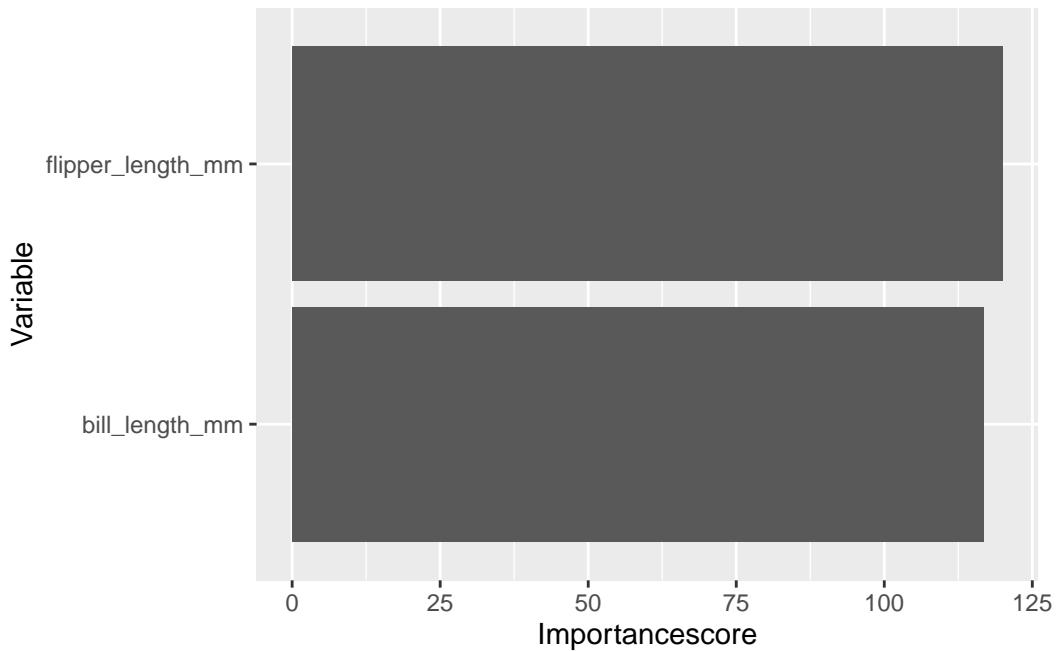
```



```

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)
```

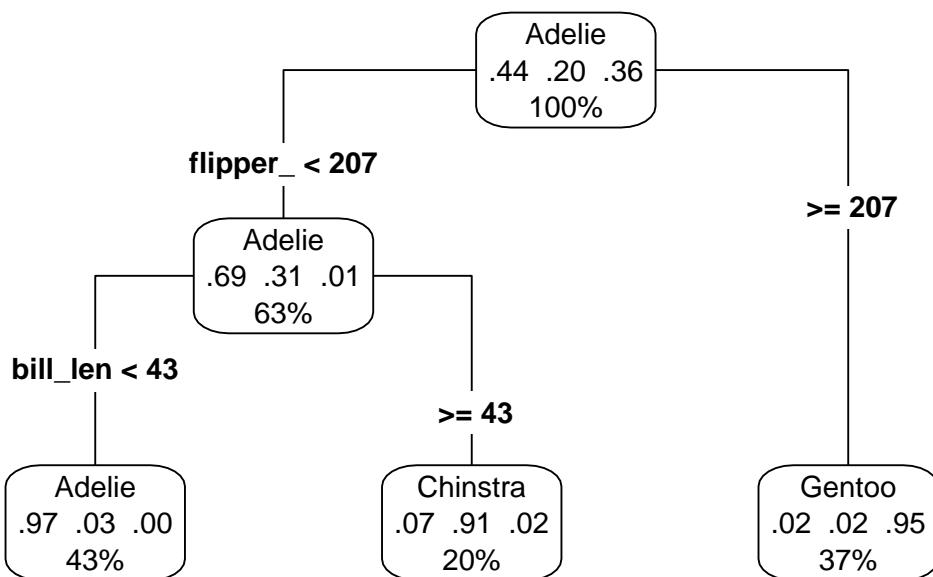
```
# tree for training sample
rpart_out_tr <- rpart(formula = rpart_formula_4,
                      data = traindata)
rpart_out$variable.importance
```

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)
```



20.1.3 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

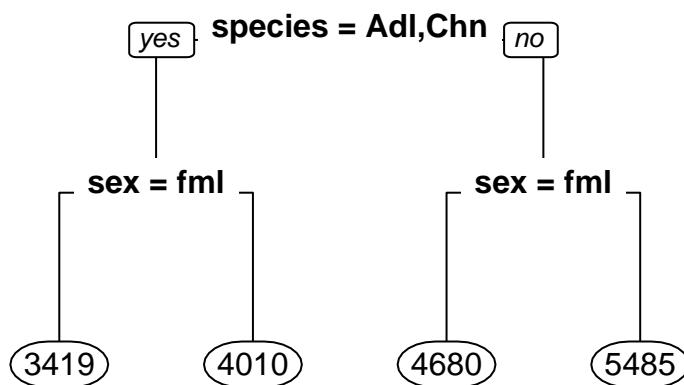
Statistics by Class:

	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

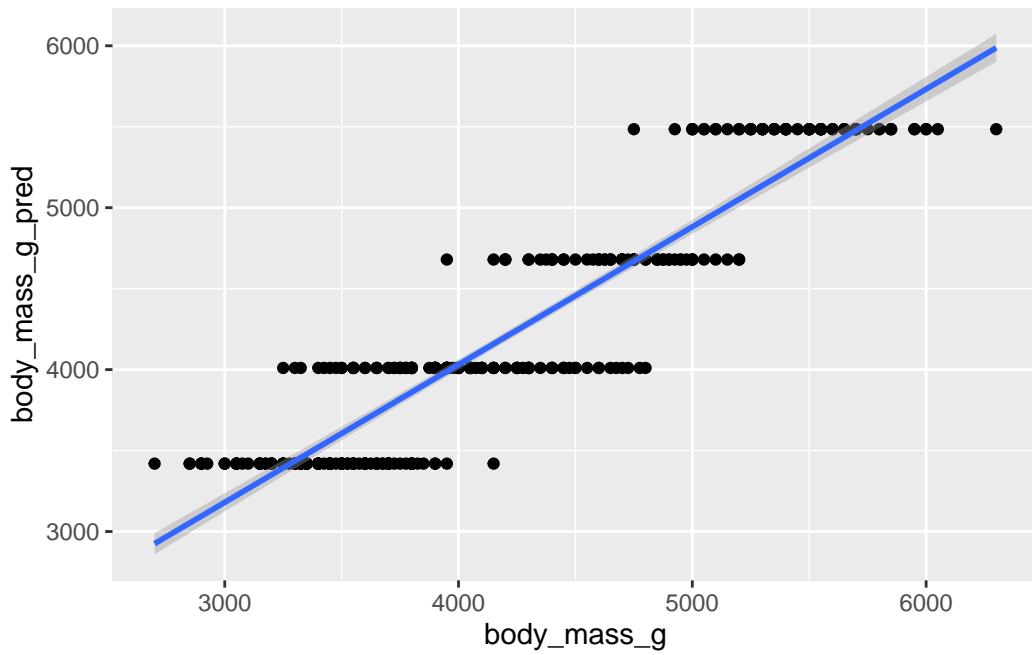
20.2 RT for continuous outcomes

```
rpart_out_cont <- rpart(body_mass_g~species+sex+
                           flipper_length_mm+bill_length_mm+
                           bill_depth_mm,
                           control = list(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'
```



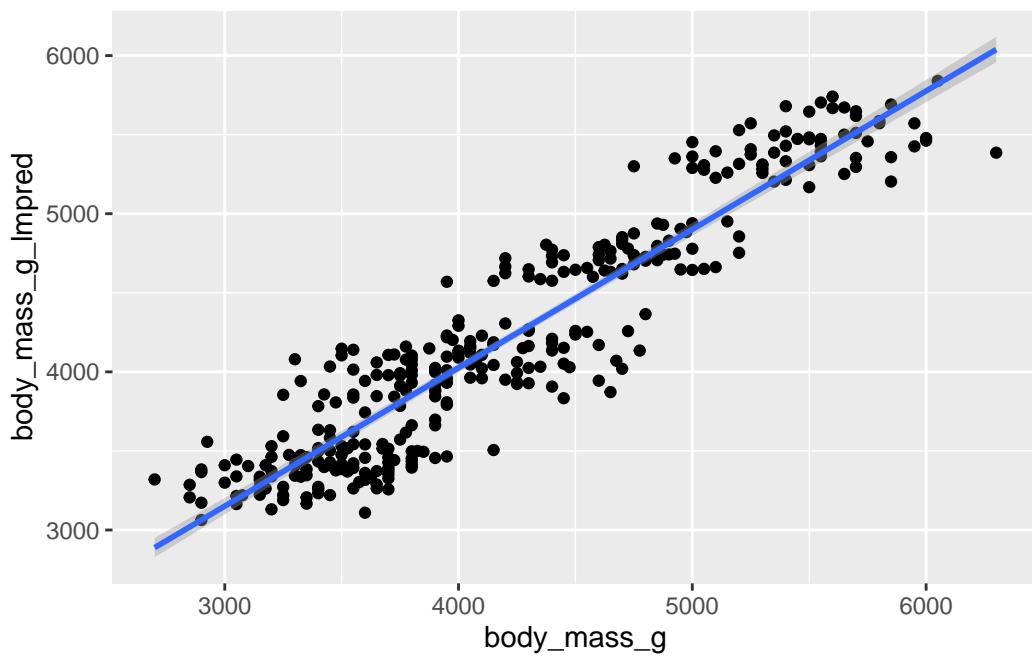
```

lm_out_cont <- lm(body_mass_g~species+sex+
                     flipper_length_mm+bill_length_mm+
                     bill_depth_mm,
                     data=rawdata)

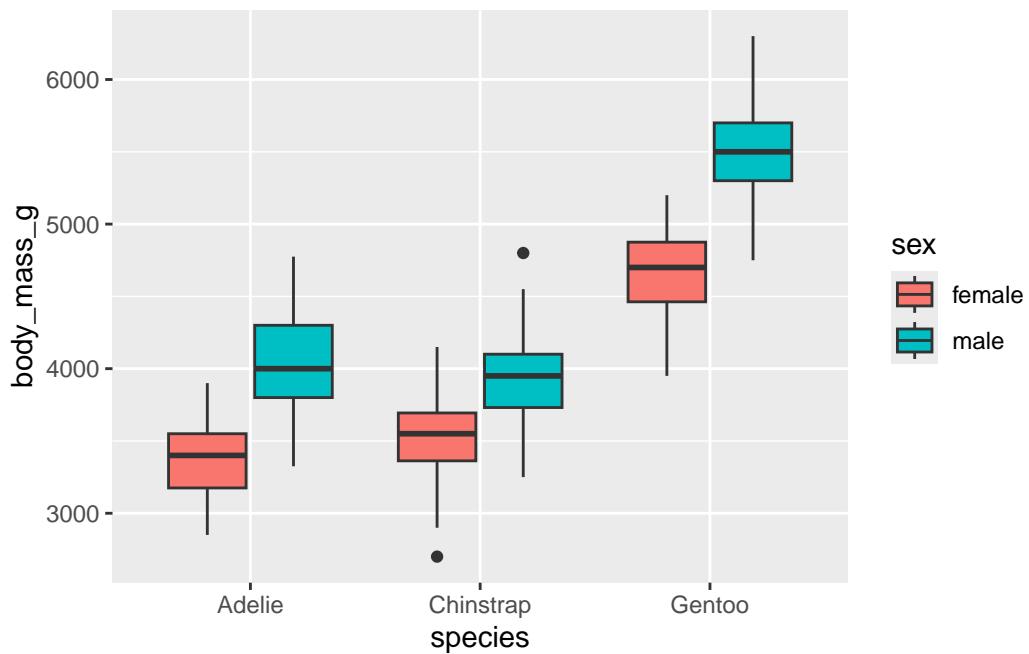
rawdata <-
  mutate(rawdata,
        body_mass_g_lmpred = predict(lm_out_cont))
ggplot(rawdata,aes(body_mass_g, body_mass_g_lmpred))+ 
  geom_point()+
  geom_smooth(method="lm")

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

```



```
ggplot(rawdata,aes(x=species,y=body_mass_g, fill=sex))+  
  geom_boxplot()
```



20.3 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)

20.3.1 Modelling

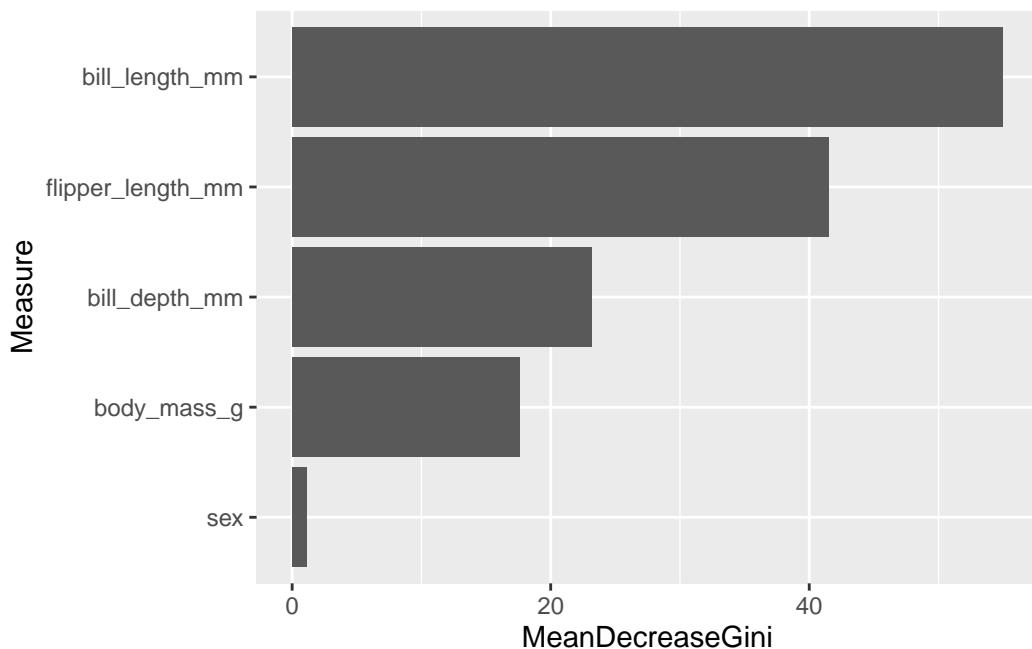
```
forrest_formula <-
  paste('species',
    paste(predvars$names, collapse='+'),
    sep='~') |>
  as.formula()

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()
```



20.3.2 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

Prediction	Reference		
	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")
```

21 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
```

```
#nthread=1)
prediction <- predict(xgb_out,boostdata) |>
  as_tibble() |> # just 1 col, 3 rows per penguin
  mutate(
    ID=rep(rawdata$ID, each=3), # 3 rows per penguin
    species_pred=rep(levels(rawdata$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 288  0.00178  0.997  0.00119  Chinstrap
2 P 51   0.999   0.000672 0.000318 Adelie
3 P 134  1.000   0.000302 0.000170 Adelie
4 P 186  0.00227  0.00220  0.996   Gentoo
5 P 114  0.999   0.000854 0.000249 Adelie
6 P 207  0.000557 0.00106  0.998   Gentoo
7 P 177  0.00176  0.000824 0.997   Gentoo
8 P 284  0.000857 0.999   0.000366 Chinstrap
9 P 158  0.000614 0.00117  0.998   Gentoo
10 P 131  0.998   0.00173  0.000293 Adelie
11 P 74   0.999   0.000315 0.000231 Adelie
12 P 18   0.999   0.000572 0.000211 Adelie
13 P 308  0.00109 0.999   0.000366 Chinstrap
14 P 226  0.000557 0.00106  0.998   Gentoo
15 P 254  0.000557 0.00106  0.998   Gentoo
16 P 57   0.998   0.00166  0.000787 Adelie
17 P 253  0.000557 0.00106  0.998   Gentoo
18 P 105  0.996   0.00215  0.00189  Adelie
19 P 21   0.999   0.000572 0.000211 Adelie
20 P 270  0.00146 0.998   0.000394 Chinstrap

```

```

train_res <- full_join(rawdata,prediction)

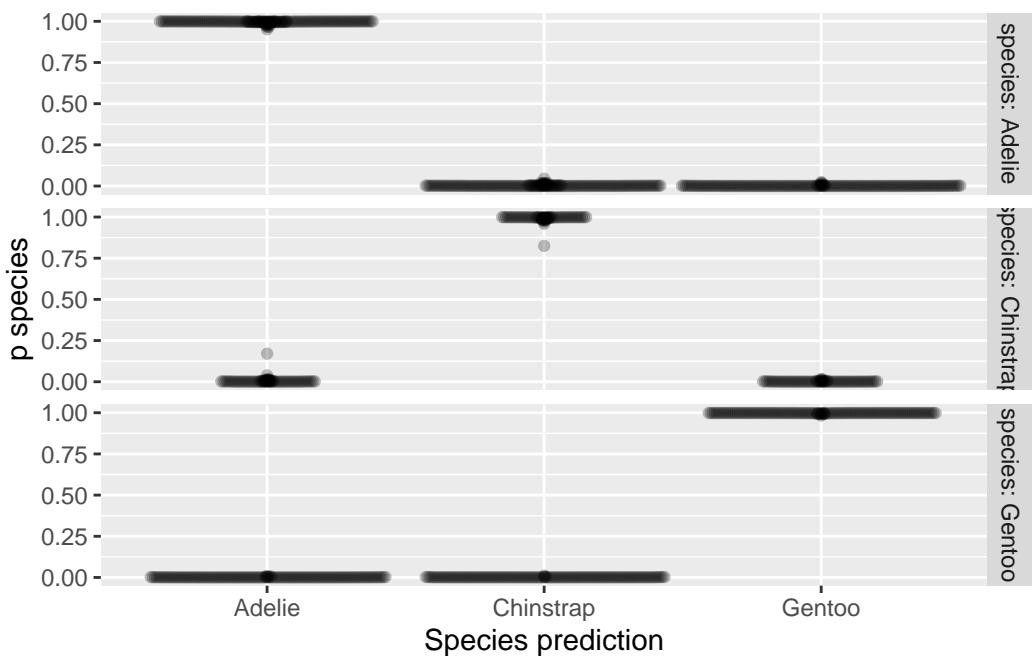
```

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

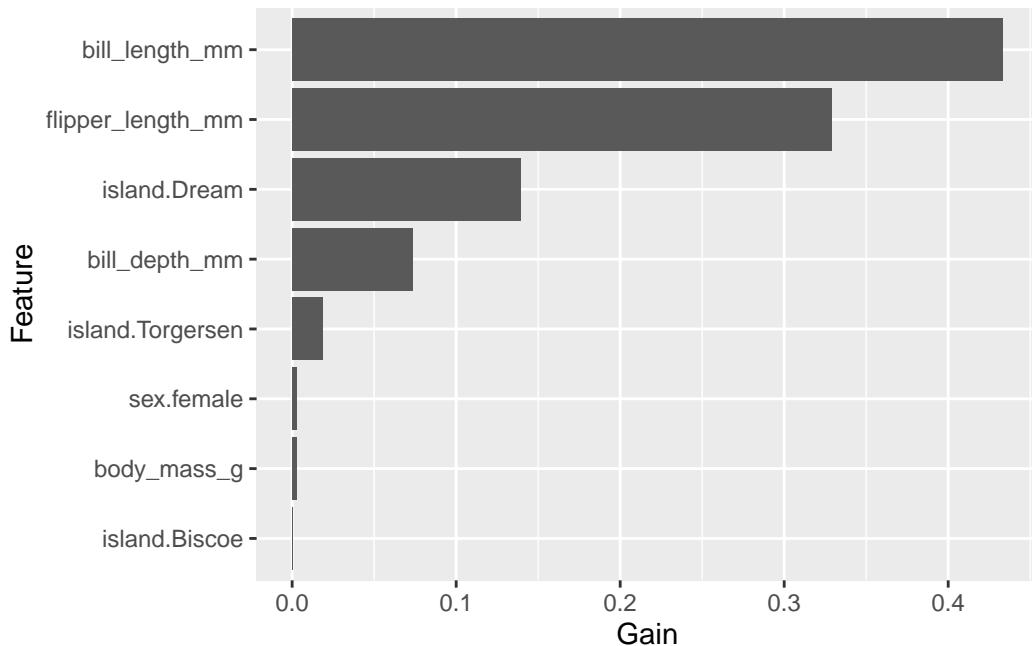
Kappa : 1

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,
  max.depth = 5,
  objective = "multi:softprob", # requests class probabilities
  num_class=3,
  watchlist=watchlist, #objects used in eval
  nrounds = 10^3) # 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
[101] train-mlogloss:0.006217 test-mlogloss:0.042995
[102] train-mlogloss:0.006217 test-mlogloss:0.042996
[103] train-mlogloss:0.006217 test-mlogloss:0.042997
[104] train-mlogloss:0.006217 test-mlogloss:0.042998
[105] train-mlogloss:0.006217 test-mlogloss:0.042999
[106] train-mlogloss:0.006217 test-mlogloss:0.042999
[107] train-mlogloss:0.006217 test-mlogloss:0.043000
[108] train-mlogloss:0.006217 test-mlogloss:0.043000
[109] train-mlogloss:0.006217 test-mlogloss:0.043001
[110] train-mlogloss:0.006217 test-mlogloss:0.043001
[111] train-mlogloss:0.006217 test-mlogloss:0.043002
[112] train-mlogloss:0.006217 test-mlogloss:0.043002
[113] train-mlogloss:0.006217 test-mlogloss:0.043002
[114] train-mlogloss:0.006217 test-mlogloss:0.043002
[115] train-mlogloss:0.006217 test-mlogloss:0.043003
[116] train-mlogloss:0.006217 test-mlogloss:0.043003
[117] train-mlogloss:0.006217 test-mlogloss:0.043003
[118] train-mlogloss:0.006217 test-mlogloss:0.043003
[119] train-mlogloss:0.006217 test-mlogloss:0.043003
[120] train-mlogloss:0.006217 test-mlogloss:0.043003
[121] train-mlogloss:0.006217 test-mlogloss:0.043003
```



```
[996] train-mlogloss:0.006217 test-mlogloss:0.043004
[997] train-mlogloss:0.006217 test-mlogloss:0.043004
[998] train-mlogloss:0.006217 test-mlogloss:0.043004
[999] train-mlogloss:0.006217 test-mlogloss:0.043004
[1000] train-mlogloss:0.006217 test-mlogloss:0.043004
```

```
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,
  max.depth = 5,
  objective = "multi:softprob",
  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)`

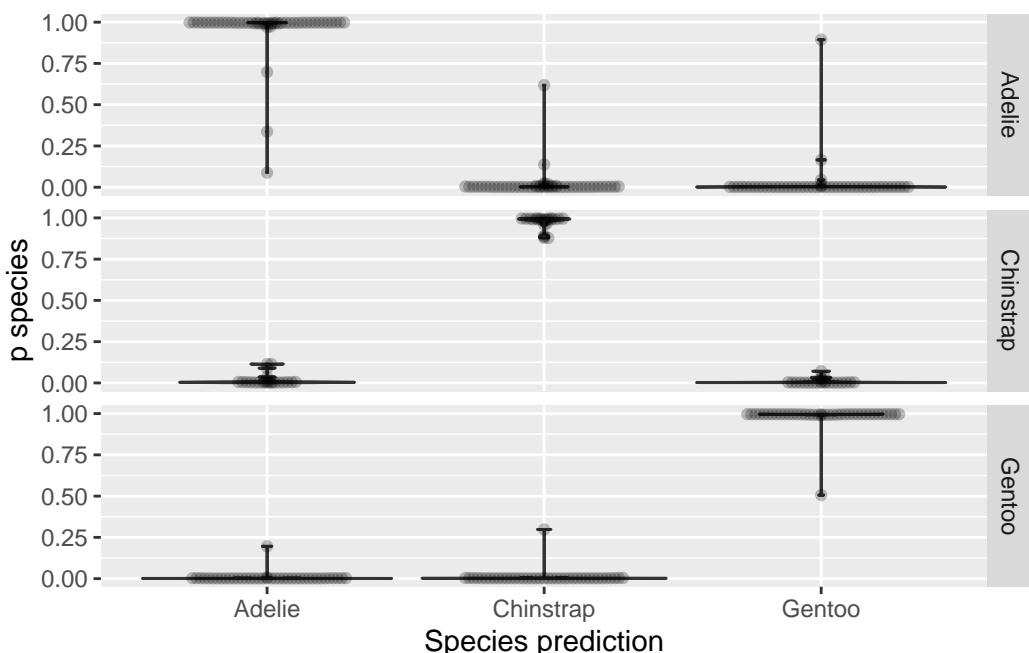
```

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%	
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

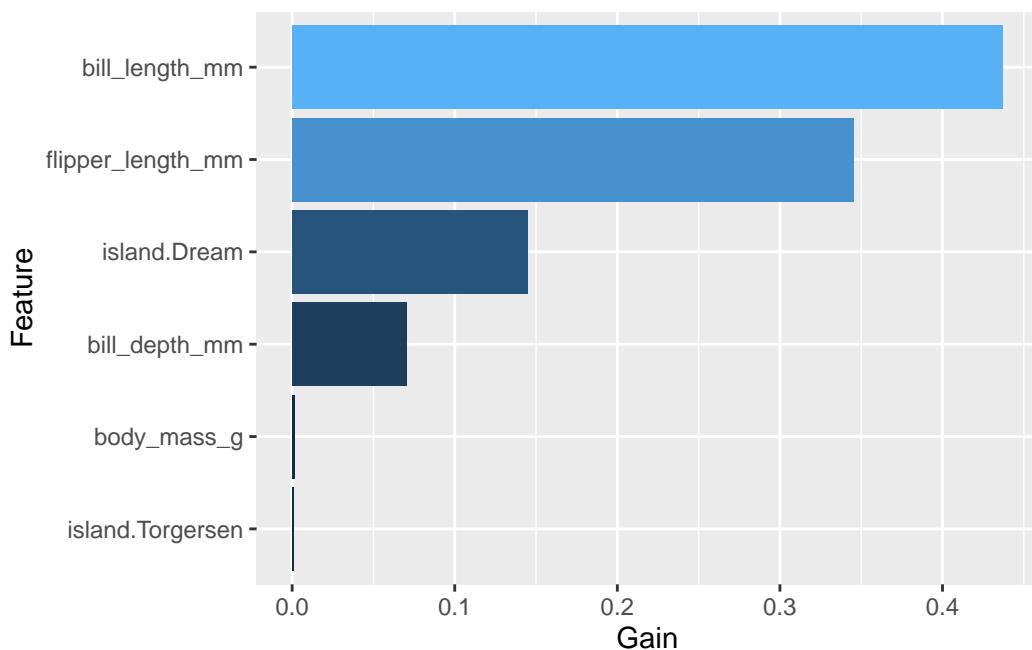
McNemar's Test P-Value : NA

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]

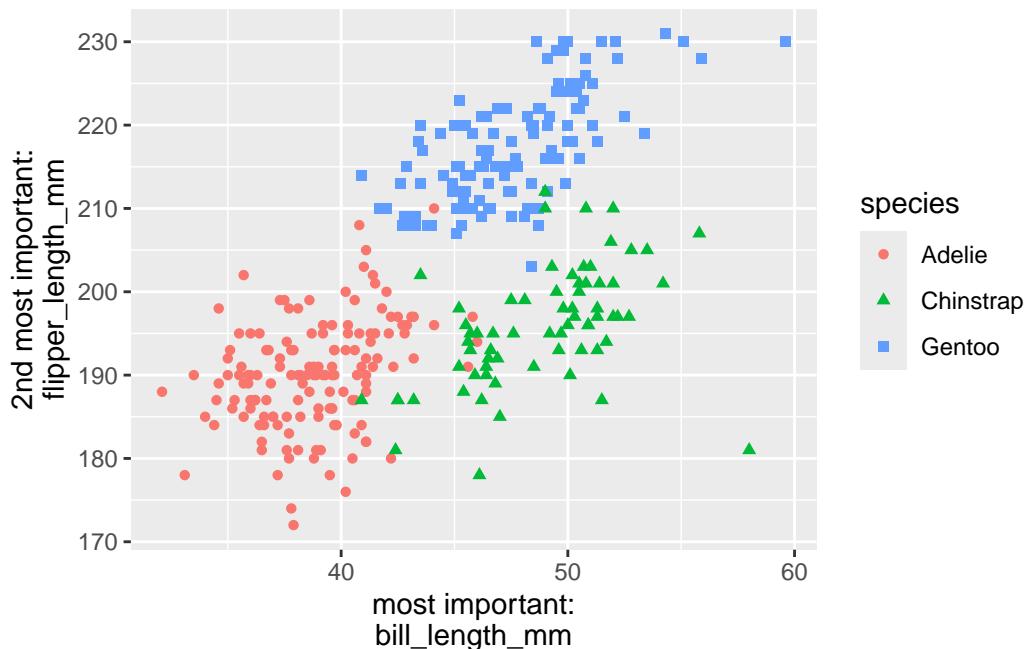
impo <- xgb.importance(model = xgb.Train)
plotfeatures_first <-
  impo$Feature[1]
plotfeatures_2nd <-
  impo$Feature[2]

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

22 Model tuning with caret

Package `caret` is a powerful tool for model tuning and comparison. It provides a unified interface to a wide range of models and allows for easy comparison of different models. In this example, we will use `caret` to tune and compare several classification models on the `penguins` dataset.

```
pacman::p_load(conflicted,
                 tidyverse,
                 wrappedtools,
                 palmerpenguins,
                 ggfortify, GGally, nnet,
                 caret,randomForest,kernlab,naivebayes,
                 mlbench)
conflict_scout()
```

```
10 conflicts
```

```
* `alpha()`': kernlab and ggplot2

* `combine()`': randomForest and dplyr

* `cross()`': kernlab and purrr

* `filter()`': dplyr and stats

* `lag()`': dplyr and stats

* `lift()`': caret and purrr

* `margin()`': randomForest and ggplot2

* `mean_cl_boot()`': wrappedtools and ggplot2

* `penguins()`': palmerpenguins and datasets

* `penguins_raw()`': palmerpenguins and datasets
```

```
#conflict_prefer(name = "filter", winner = "dplyr")
conflicts_prefer(
  dplyr::filter(),
  ggplot2::alpha,
  dplyr::combine,
  dplyr::slice,
  palmerpenguins::penguins)
```

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

[conflicted] Will prefer ggplot2::alpha over any other package.

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

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

[conflicted] Will prefer palmerpenguins::penguins over any other package.

```
rawdata <- penguins |>
  na.omit()
predvars <- ColSeeker(namepattern = c('_mm', '_g'))
rawdata <- select(rawdata,
                  species, predvars$names)
```

22.1 Create training/test data

```
set.seed(2001)
trainindex <- createDataPartition(
  y = rawdata$species,
  times = 1,
  p = 0.75,
  list = FALSE
)

traindata <- rawdata |> slice(trainindex[,1])
testdata <- rawdata |> slice(-trainindex[,1])

# cat_desc_stats(traindata$species,singleline = T)
# cat_desc_stats(testdata$species,singleline = T)
```

22.2 Define global modelling options

```
ctrl <- trainControl(method = "repeatedcv",
                      number=5, # 5-fold cross-validation
                      repeats = 25) # 25 repeats
```

22.3 Define method-specific options and tune models

```
# define method-specific options #####
tune <- expand.grid(k=seq(1,9,2))
# tune a model #####
knnfit <- train(form = species~.,
                 data = traindata,
                 preProcess = c('center','scale'),
                 method='knn',
                 metric='Accuracy',
                 trControl = ctrl,
                 tuneGrid=tune)

# define method-specific options #####
tune <- expand.grid(nrounds=c(50,100, 200),
                     max_depth=seq(5,15,5),
                     eta=1,
                     gamma=c(.01,.001),
                     colsample_bytree=1,
                     min_child_weight=1,
                     subsample=1)
xgbfit <- train(form = species~.,
                 data = traindata,
                 preProcess = c('center','scale'),
                 method='xgbTree',
                 objective = "multi:softprob",
                 metric='Accuracy',
                 trControl = ctrl,
                 tuneGrid=tune,
                 verbosity = 0)
```

```
Warning in check.booster.params(params, ...): The following parameters were provided multi
objective
Only the last value for each of them will be used.
Warning in check.booster.params(params, ...): The following parameters were provided multi
objective
```


Only the last value for each of them will be used.

Warning in check.booster.params(params, ...): The following parameters were provided multi-objective

Only the last value for each of them will be used.

Warning in check.booster.params(params, ...): The following parameters were provided multi-objective

Only the last value for each of them will be used.

Warning in check.booster.params(params, ...): The following parameters were provided multi-objective

Only the last value for each of them will be used.

Warning in check.booster.params(params, ...): The following parameters were provided multi-objective

Only the last value for each of them will be used.

Warning in check.booster.params(params, ...): The following parameters were provided multi-objective

Only the last value for each of them will be used.

Warning in check.booster.params(params, ...): The following parameters were provided multi-objective

Only the last value for each of them will be used.

Warning in check.booster.params(params, ...): The following parameters were provided multi-objective

Only the last value for each of them will be used.

Warning in check.booster.params(params, ...): The following parameters were provided multi-objective

Only the last value for each of them will be used.

Warning in check.booster.params(params, ...): The following parameters were provided multi-objective

Only the last value for each of them will be used.

Warning in check.booster.params(params, ...): The following parameters were provided multi-objective

Only the last value for each of them will be used

Warning in check.booster.params(params, ...): The following parameters were provided multi-objective

Only the last value for each of them will be used

Warning in check.booster.params(params, ...): The following parameters were provided multi-objective.

Only the last value for each of them will be used

Warning in check.booster.params(params, ...): The following parameters were provided multi-objective

Only the last value for each of them will be used

```
ldafit <- train(form = species~.,
                 data = traindata,
                 preProcess = c('center', 'scale'))
```

```

    method='lda',
    metric='Accuracy',
    trControl = ctrl)

# define method-specific options #####
tune <- expand.grid(mtry = seq(2,3,1))
rffit <- train(form = species~.,
                data = traindata,
                preProcess = c('center','scale'),
                method='rf',
                metric='Accuracy',
                ntree=300,
                trControl = ctrl,
                tuneGrid=tune)

svmfit <- train(form = species~.,
                  data = traindata,
                  preProcess = c('center','scale'),
                  method='svmLinear',
                  metric='Accuracy',
                  trControl = ctrl)

bayesfit <- train(form = species~.,
                     data = traindata,
                     preProcess = c('center','scale'),
                     method='naive_bayes',
                     metric='Accuracy',
                     trControl = ctrl)

nnfit <- train(form=species~.,
                 data=traindata,
                 method="nnet",
                 preProcess=c("center","scale"),
                 metric="Accuracy",
                 trControl=ctrl)

```

```

# weights:  11
initial  value 230.026921
iter   10 value 86.166424
iter   20 value 80.664125
final  value 80.654382
converged
# weights:  27
initial  value 208.418464
iter   10 value 2.691935

```

```
iter 20 value 0.060854
iter 30 value 0.013348
iter 40 value 0.000454
final value 0.000055
converged
# weights: 43
initial value 248.149396
iter 10 value 0.621044
iter 20 value 0.020251
iter 30 value 0.001716
iter 40 value 0.000209
iter 50 value 0.000108
iter 50 value 0.000088
iter 50 value 0.000088
final value 0.000088
converged
# weights: 11
initial value 218.130307
iter 10 value 92.568556
iter 20 value 67.720515
iter 30 value 61.689926
final value 61.688702
converged
# weights: 27
initial value 259.592632
iter 10 value 26.453432
iter 20 value 20.333879
iter 30 value 18.946681
iter 40 value 18.929200
iter 50 value 18.928508
final value 18.928504
converged
# weights: 43
initial value 215.852565
iter 10 value 23.764346
iter 20 value 15.680602
iter 30 value 15.484272
iter 40 value 15.036471
iter 50 value 14.801654
iter 60 value 14.788067
iter 70 value 14.782906
final value 14.782895
converged
# weights: 11
initial value 226.180088
iter 10 value 86.643801
```

```

iter 20 value 72.533925
iter 30 value 29.658354
iter 40 value 18.121435
iter 50 value 17.837940
iter 60 value 17.493095
iter 70 value 17.148007
iter 80 value 17.134627
iter 90 value 17.108196
iter 100 value 17.104394
final value 17.104394
stopped after 100 iterations
# weights: 27
initial value 259.549697
iter 10 value 3.899993
iter 20 value 0.217116
iter 30 value 0.186886
iter 40 value 0.164751
iter 50 value 0.117857
iter 60 value 0.109357
iter 70 value 0.104011
iter 80 value 0.098781
iter 90 value 0.092050
iter 100 value 0.086751
final value 0.086751
stopped after 100 iterations
# weights: 43
initial value 214.942466
iter 10 value 12.093201
iter 20 value 0.524532
iter 30 value 0.439493
iter 40 value 0.353839
iter 50 value 0.172208
iter 60 value 0.146512
iter 70 value 0.127009
iter 80 value 0.106705
iter 90 value 0.086661
iter 100 value 0.082117
final value 0.082117
stopped after 100 iterations
# weights: 11
initial value 228.734455
iter 10 value 131.758454
iter 20 value 62.089657
iter 30 value 37.685458
iter 40 value 18.915258
iter 50 value 18.078775

```

```
iter 60 value 17.668531
iter 70 value 17.528290
iter 80 value 17.497908
iter 90 value 17.240869
iter 100 value 17.103139
final value 17.103139
stopped after 100 iterations
# weights: 27
initial value 222.274118
iter 10 value 2.411766
iter 20 value 0.003476
iter 30 value 0.000228
final value 0.000062
converged
# weights: 43
initial value 256.339331
iter 10 value 2.284448
iter 20 value 0.017665
iter 30 value 0.000619
final value 0.000084
converged
# weights: 11
initial value 285.444960
iter 10 value 93.064487
iter 20 value 91.292447
iter 30 value 86.803075
iter 40 value 86.128897
final value 86.128636
converged
# weights: 27
initial value 283.327852
iter 10 value 30.803441
iter 20 value 20.321679
iter 30 value 19.625666
iter 40 value 19.555224
iter 50 value 19.356661
iter 60 value 19.344480
final value 19.344075
converged
# weights: 43
initial value 205.296026
iter 10 value 24.082988
iter 20 value 17.147471
iter 30 value 16.411462
iter 40 value 16.269255
iter 50 value 16.256364
```

```
iter 60 value 16.244190
iter 70 value 16.240655
final value 16.240605
converged
# weights: 11
initial value 273.245246
iter 10 value 112.234870
iter 20 value 79.014471
iter 30 value 28.051004
iter 40 value 17.955420
iter 50 value 17.683362
iter 60 value 17.587351
iter 70 value 17.561484
iter 80 value 17.550854
iter 90 value 17.546390
iter 100 value 17.546352
final value 17.546352
stopped after 100 iterations
# weights: 27
initial value 213.777188
iter 10 value 1.288321
iter 20 value 0.149527
iter 30 value 0.139767
iter 40 value 0.137404
iter 50 value 0.128285
iter 60 value 0.123138
iter 70 value 0.116265
iter 80 value 0.107901
iter 90 value 0.103916
iter 100 value 0.102032
final value 0.102032
stopped after 100 iterations
# weights: 43
initial value 257.154200
iter 10 value 3.573040
iter 20 value 0.164354
iter 30 value 0.138734
iter 40 value 0.126646
iter 50 value 0.121249
iter 60 value 0.117440
iter 70 value 0.114694
iter 80 value 0.111703
iter 90 value 0.109771
iter 100 value 0.107090
final value 0.107090
stopped after 100 iterations
```

```

# weights: 11
initial value 233.961065
iter 10 value 82.710098
iter 20 value 79.582532
iter 30 value 79.493888
iter 40 value 79.490172
final value 79.490034
converged
# weights: 27
initial value 220.338577
iter 10 value 0.845782
iter 20 value 0.057892
iter 30 value 0.009074
final value 0.000057
converged
# weights: 43
initial value 229.743661
iter 10 value 1.745179
iter 20 value 0.068796
iter 30 value 0.005262
iter 40 value 0.000726
iter 50 value 0.000250
iter 60 value 0.000124
final value 0.000094
converged
# weights: 11
initial value 228.097742
iter 10 value 119.562948
iter 20 value 72.695121
iter 30 value 64.151691
final value 64.098438
converged
# weights: 27
initial value 217.099632
iter 10 value 37.251298
iter 20 value 21.056258
iter 30 value 17.503542
iter 40 value 17.011798
iter 50 value 16.688623
iter 60 value 16.390771
iter 70 value 16.387659
final value 16.387657
converged
# weights: 43
initial value 222.061085
iter 10 value 21.565229

```

```

iter 20 value 15.313982
iter 30 value 14.972680
iter 40 value 14.914414
iter 50 value 14.660778
iter 60 value 14.618276
iter 70 value 14.618159
final value 14.618154
converged
# weights: 11
initial value 237.950532
iter 10 value 84.456993
iter 20 value 46.673612
iter 30 value 25.140753
iter 40 value 21.732294
iter 50 value 21.584391
iter 60 value 21.503190
iter 70 value 21.360887
iter 80 value 21.360662
iter 90 value 21.359992
final value 21.359982
converged
# weights: 27
initial value 220.210848
iter 10 value 4.434413
iter 20 value 0.161639
iter 30 value 0.148688
iter 40 value 0.143470
iter 50 value 0.138616
iter 60 value 0.136573
iter 70 value 0.132626
iter 80 value 0.130213
iter 90 value 0.124751
iter 100 value 0.121673
final value 0.121673
stopped after 100 iterations
# weights: 43
initial value 214.968643
iter 10 value 1.206305
iter 20 value 0.136904
iter 30 value 0.121233
iter 40 value 0.119779
iter 50 value 0.112393
iter 60 value 0.107544
iter 70 value 0.102179
iter 80 value 0.098258
iter 90 value 0.095452

```

```

iter 100 value 0.091771
final  value 0.091771
stopped after 100 iterations
# weights: 11
initial  value 204.687950
iter   10 value 64.982095
iter   20 value 23.648618
iter   30 value 20.622223
iter   40 value 19.622833
iter   50 value 19.371125
iter   60 value 19.273008
iter   70 value 19.172249
iter   80 value 19.137232
iter   90 value 19.124213
iter  100 value 19.083155
final  value 19.083155
stopped after 100 iterations
# weights: 27
initial  value 218.929310
iter   10 value 6.064461
iter   20 value 0.213400
iter   30 value 0.003475
iter   40 value 0.001556
final  value 0.000080
converged
# weights: 43
initial  value 235.727912
iter   10 value 2.404889
iter   20 value 0.018384
iter   30 value 0.003361
iter   40 value 0.000537
iter   50 value 0.000455
final  value 0.000084
converged
# weights: 11
initial  value 221.073866
iter   10 value 101.572615
iter   20 value 75.072105
iter   30 value 64.342439
final  value 64.223469
converged
# weights: 27
initial  value 225.541926
iter   10 value 32.120401
iter   20 value 20.375590
iter   30 value 19.496181

```

```
iter 40 value 19.473607
iter 50 value 19.467783
iter 60 value 19.466955
final value 19.466916
converged
# weights: 43
initial value 204.722992
iter 10 value 38.672721
iter 20 value 16.669904
iter 30 value 16.494620
iter 40 value 16.479213
iter 50 value 16.477903
iter 60 value 16.477479
final value 16.477471
converged
# weights: 11
initial value 222.866952
iter 10 value 67.364016
iter 20 value 28.456854
iter 30 value 21.333458
iter 40 value 19.945011
iter 50 value 19.841024
iter 60 value 19.829055
iter 70 value 19.784345
final value 19.784342
converged
# weights: 27
initial value 232.778488
iter 10 value 3.959633
iter 20 value 0.355459
iter 30 value 0.227655
iter 40 value 0.210324
iter 50 value 0.178133
iter 60 value 0.153433
iter 70 value 0.148631
iter 80 value 0.141859
iter 90 value 0.136873
iter 100 value 0.128656
final value 0.128656
stopped after 100 iterations
# weights: 43
initial value 216.829543
iter 10 value 5.075491
iter 20 value 0.180699
iter 30 value 0.122114
iter 40 value 0.116237
```

```

iter 50 value 0.112264
iter 60 value 0.108530
iter 70 value 0.104136
iter 80 value 0.101895
iter 90 value 0.099865
iter 100 value 0.097778
final value 0.097778
stopped after 100 iterations
# weights: 11
initial value 247.271069
iter 10 value 66.487641
iter 20 value 15.508991
iter 30 value 14.615543
iter 40 value 14.503221
iter 50 value 14.457888
iter 60 value 14.366419
iter 70 value 14.103720
iter 80 value 14.091457
iter 90 value 14.064522
iter 100 value 14.018209
final value 14.018209
stopped after 100 iterations
# weights: 27
initial value 236.257967
iter 10 value 3.081115
iter 20 value 0.006636
final value 0.000079
converged
# weights: 43
initial value 216.390544
iter 10 value 1.981284
iter 20 value 0.122969
iter 30 value 0.001182
iter 40 value 0.000230
final value 0.000071
converged
# weights: 11
initial value 216.398224
iter 10 value 93.346515
iter 20 value 83.500469
iter 30 value 62.461992
final value 61.934086
converged
# weights: 27
initial value 267.014199
iter 10 value 42.145954

```

```

iter 20 value 20.463674
iter 30 value 19.299561
iter 40 value 19.210847
iter 50 value 19.198799
iter 60 value 19.197690
final value 19.197680
converged
# weights: 43
initial value 243.194575
iter 10 value 28.074779
iter 20 value 16.965174
iter 30 value 16.351260
iter 40 value 16.273313
iter 50 value 16.191100
iter 60 value 15.751087
iter 70 value 15.744494
final value 15.744485
converged
# weights: 11
initial value 225.683836
iter 10 value 71.625092
iter 20 value 41.646437
iter 30 value 19.363786
iter 40 value 14.979450
iter 50 value 14.655568
iter 60 value 14.589393
iter 70 value 14.583542
iter 80 value 14.549855
iter 90 value 14.549221
iter 100 value 14.548604
final value 14.548604
stopped after 100 iterations
# weights: 27
initial value 228.813566
iter 10 value 23.844157
iter 20 value 5.683862
iter 30 value 0.572635
iter 40 value 0.260643
iter 50 value 0.235208
iter 60 value 0.222967
iter 70 value 0.202466
iter 80 value 0.141049
iter 90 value 0.134560
iter 100 value 0.128132
final value 0.128132
stopped after 100 iterations

```

```

# weights: 43
initial value 207.024866
iter 10 value 3.015245
iter 20 value 0.226930
iter 30 value 0.182281
iter 40 value 0.163707
iter 50 value 0.148461
iter 60 value 0.141944
iter 70 value 0.133982
iter 80 value 0.119970
iter 90 value 0.113074
iter 100 value 0.107053
final value 0.107053
stopped after 100 iterations
# weights: 11
initial value 226.559972
iter 10 value 62.474760
iter 20 value 24.604734
iter 30 value 22.522928
iter 40 value 22.192741
iter 50 value 21.818320
iter 60 value 21.760127
iter 70 value 21.701847
iter 80 value 21.633126
iter 90 value 21.626597
iter 100 value 21.578172
final value 21.578172
stopped after 100 iterations
# weights: 27
initial value 275.696738
iter 10 value 2.808215
iter 20 value 0.004556
iter 30 value 0.000395
final value 0.000092
converged
# weights: 43
initial value 228.804819
iter 10 value 4.248314
iter 20 value 0.009988
final value 0.000086
converged
# weights: 11
initial value 232.872876
iter 10 value 127.743343
iter 20 value 97.101337
iter 30 value 64.816911

```

```
iter 40 value 64.464702
iter 40 value 64.464702
iter 40 value 64.464702
final value 64.464702
converged
# weights: 27
initial value 235.884130
iter 10 value 25.733697
iter 20 value 19.240521
iter 30 value 18.996351
final value 18.994293
converged
# weights: 43
initial value 232.137873
iter 10 value 21.056018
iter 20 value 15.962785
iter 30 value 15.766887
iter 40 value 15.751222
iter 50 value 15.749109
iter 60 value 15.748440
final value 15.748440
converged
# weights: 11
initial value 248.116890
iter 10 value 65.294686
iter 20 value 32.121165
iter 30 value 23.999068
iter 40 value 22.367362
iter 50 value 22.200174
iter 60 value 22.188675
iter 70 value 22.174691
final value 22.174689
converged
# weights: 27
initial value 233.906021
iter 10 value 35.702776
iter 20 value 0.307344
iter 30 value 0.235865
iter 40 value 0.199022
iter 50 value 0.180548
iter 60 value 0.158847
iter 70 value 0.148262
iter 80 value 0.143216
iter 90 value 0.129794
iter 100 value 0.119103
final value 0.119103
```

```
stopped after 100 iterations
# weights: 43
initial value 297.523977
iter 10 value 6.399327
iter 20 value 0.312008
iter 30 value 0.285883
iter 40 value 0.227667
iter 50 value 0.192143
iter 60 value 0.153889
iter 70 value 0.118277
iter 80 value 0.107601
iter 90 value 0.104848
iter 100 value 0.099947
final value 0.099947
stopped after 100 iterations
# weights: 11
initial value 211.279534
iter 10 value 81.067722
iter 20 value 80.297806
iter 30 value 79.794156
iter 40 value 78.848882
iter 50 value 78.543267
iter 60 value 78.433752
iter 70 value 78.218951
iter 80 value 75.910500
iter 90 value 64.342142
iter 100 value 38.435491
final value 38.435491
stopped after 100 iterations
# weights: 27
initial value 237.949785
iter 10 value 7.061673
iter 20 value 0.016065
iter 30 value 0.001501
iter 40 value 0.001069
iter 50 value 0.000559
final value 0.000076
converged
# weights: 43
initial value 234.538554
iter 10 value 2.179415
iter 20 value 0.005638
final value 0.000097
converged
# weights: 11
initial value 222.866088
```

```

iter 10 value 92.478878
iter 20 value 63.178625
iter 30 value 62.752667
iter 30 value 62.752667
iter 30 value 62.752667
final value 62.752667
converged
# weights: 27
initial value 218.964131
iter 10 value 44.250211
iter 20 value 19.225871
iter 30 value 18.995631
iter 40 value 18.842112
iter 50 value 18.109869
iter 60 value 17.934897
final value 17.934403
converged
# weights: 43
initial value 250.074513
iter 10 value 31.292729
iter 20 value 17.264791
iter 30 value 16.850733
iter 40 value 16.556128
iter 50 value 16.543958
final value 16.543920
converged
# weights: 11
initial value 230.587898
iter 10 value 92.062403
iter 20 value 74.508711
iter 30 value 73.152213
iter 40 value 71.105601
iter 50 value 61.921147
iter 60 value 25.249419
iter 70 value 20.177186
iter 80 value 19.054621
iter 90 value 18.913346
iter 100 value 18.846367
final value 18.846367
stopped after 100 iterations
# weights: 27
initial value 243.145349
iter 10 value 1.810210
iter 20 value 0.199701
iter 30 value 0.192593
iter 40 value 0.180518

```

```

iter 50 value 0.173134
iter 60 value 0.159150
iter 70 value 0.150060
iter 80 value 0.139108
iter 90 value 0.132283
iter 100 value 0.124891
final value 0.124891
stopped after 100 iterations
# weights: 43
initial value 262.383584
iter 10 value 0.689890
iter 20 value 0.328236
iter 30 value 0.280125
iter 40 value 0.235459
iter 50 value 0.183170
iter 60 value 0.164449
iter 70 value 0.148303
iter 80 value 0.132818
iter 90 value 0.127831
iter 100 value 0.125031
final value 0.125031
stopped after 100 iterations
# weights: 11
initial value 230.042483
iter 10 value 82.260116
iter 20 value 73.964989
iter 30 value 73.402101
iter 40 value 73.218679
iter 50 value 73.211451
iter 60 value 73.194558
iter 70 value 73.142380
iter 80 value 73.120300
iter 90 value 73.110748
iter 100 value 73.110544
final value 73.110544
stopped after 100 iterations
# weights: 27
initial value 305.807457
iter 10 value 6.808776
iter 20 value 0.037396
iter 30 value 0.000474
final value 0.000089
converged
# weights: 43
initial value 238.529712
iter 10 value 1.179185

```

```
iter 20 value 0.039882
iter 30 value 0.003741
iter 40 value 0.000173
final value 0.000098
converged
# weights: 11
initial value 225.175326
iter 10 value 92.794565
iter 20 value 85.768137
iter 30 value 83.406238
final value 83.322296
converged
# weights: 27
initial value 236.341824
iter 10 value 38.166027
iter 20 value 19.655184
iter 30 value 18.112787
iter 40 value 17.588545
iter 50 value 17.460420
iter 60 value 17.423814
iter 70 value 17.419694
final value 17.419693
converged
# weights: 43
initial value 246.211641
iter 10 value 30.475135
iter 20 value 17.133251
iter 30 value 16.039793
iter 40 value 15.961152
iter 50 value 15.959101
iter 60 value 15.955940
iter 70 value 15.955856
final value 15.955840
converged
# weights: 11
initial value 254.820990
iter 10 value 82.080470
iter 20 value 76.983735
iter 30 value 76.437510
iter 40 value 76.382864
iter 50 value 75.672802
iter 60 value 75.046248
iter 70 value 75.027965
iter 80 value 75.017226
iter 90 value 74.957767
iter 100 value 74.939780
```

```
final value 74.939780
stopped after 100 iterations
# weights: 27
initial value 235.534974
iter 10 value 9.356003
iter 20 value 0.286266
iter 30 value 0.249751
iter 40 value 0.174818
iter 50 value 0.153120
iter 60 value 0.138475
iter 70 value 0.129052
iter 80 value 0.119244
iter 90 value 0.111701
iter 100 value 0.108041
final value 0.108041
stopped after 100 iterations
# weights: 43
initial value 232.193625
iter 10 value 2.963269
iter 20 value 0.249458
iter 30 value 0.172344
iter 40 value 0.165203
iter 50 value 0.157537
iter 60 value 0.143683
iter 70 value 0.123616
iter 80 value 0.108386
iter 90 value 0.105624
iter 100 value 0.102312
final value 0.102312
stopped after 100 iterations
# weights: 11
initial value 212.905897
iter 10 value 83.636325
iter 20 value 57.893828
iter 30 value 30.921437
iter 40 value 23.009332
iter 50 value 22.372160
iter 60 value 21.737725
iter 70 value 21.629920
iter 80 value 21.558320
iter 90 value 21.500676
iter 100 value 21.486333
final value 21.486333
stopped after 100 iterations
# weights: 27
initial value 204.685402
```

```

iter 10 value 2.100030
iter 20 value 0.005987
iter 30 value 0.001003
iter 40 value 0.000688
final value 0.000059
converged
# weights: 43
initial value 210.048705
iter 10 value 3.518782
iter 20 value 0.014235
iter 30 value 0.002290
iter 40 value 0.000420
iter 50 value 0.000190
final value 0.000088
converged
# weights: 11
initial value 236.198193
iter 10 value 146.703836
iter 20 value 130.479397
iter 30 value 127.394696
iter 40 value 98.549818
iter 50 value 76.235157
iter 60 value 65.643519
final value 65.639617
converged
# weights: 27
initial value 231.925872
iter 10 value 28.287656
iter 20 value 19.168328
iter 30 value 18.769951
iter 40 value 18.766940
iter 50 value 18.766698
iter 50 value 18.766698
iter 50 value 18.766698
final value 18.766698
converged
# weights: 43
initial value 204.603647
iter 10 value 16.789920
iter 20 value 15.598940
iter 30 value 15.463638
iter 40 value 15.449991
iter 50 value 15.449181
iter 60 value 15.448325
iter 70 value 15.446321
final value 15.446306

```

```

converged
# weights: 11
initial value 223.349925
iter 10 value 80.780985
iter 20 value 80.681682
iter 30 value 80.408072
iter 40 value 77.692752
iter 50 value 68.701887
iter 60 value 40.465413
iter 70 value 37.466565
iter 80 value 36.540091
iter 90 value 36.426291
iter 100 value 36.423048
final value 36.423048
stopped after 100 iterations
# weights: 27
initial value 216.816231
iter 10 value 6.483753
iter 20 value 0.488979
iter 30 value 0.348002
iter 40 value 0.316094
iter 50 value 0.282845
iter 60 value 0.203315
iter 70 value 0.178065
iter 80 value 0.166427
iter 90 value 0.145638
iter 100 value 0.133911
final value 0.133911
stopped after 100 iterations
# weights: 43
initial value 311.646836
iter 10 value 1.649845
iter 20 value 0.118105
iter 30 value 0.110753
iter 40 value 0.107761
iter 50 value 0.104759
iter 60 value 0.099352
iter 70 value 0.095674
iter 80 value 0.092031
iter 90 value 0.090787
iter 100 value 0.090065
final value 0.090065
stopped after 100 iterations
# weights: 11
initial value 220.913537
iter 10 value 78.499121

```

```
iter 20 value 43.400839
iter 30 value 14.589405
iter 40 value 12.178387
iter 50 value 12.128987
iter 60 value 12.089804
iter 70 value 11.986658
iter 80 value 11.963421
iter 90 value 11.918493
iter 100 value 11.889152
final value 11.889152
stopped after 100 iterations
# weights: 27
initial value 212.071450
iter 10 value 0.960520
iter 20 value 0.005035
iter 30 value 0.000445
final value 0.000065
converged
# weights: 43
initial value 303.200440
iter 10 value 4.754189
iter 20 value 0.004127
iter 30 value 0.000571
final value 0.000027
converged
# weights: 11
initial value 246.642868
iter 10 value 94.657981
iter 20 value 83.049297
iter 30 value 83.024760
final value 83.024443
converged
# weights: 27
initial value 219.762750
iter 10 value 37.715648
iter 20 value 21.590084
iter 30 value 18.124464
iter 40 value 17.631988
iter 50 value 17.583779
iter 60 value 17.575891
iter 70 value 17.572290
iter 80 value 17.538186
iter 90 value 17.077974
iter 100 value 16.234307
final value 16.234307
stopped after 100 iterations
```

```

# weights: 43
initial value 207.626718
iter 10 value 25.409093
iter 20 value 17.318097
iter 30 value 17.226018
iter 40 value 17.218349
iter 50 value 17.218197
iter 60 value 17.218191
iter 60 value 17.218191
iter 60 value 17.218191
final value 17.218191
converged
# weights: 11
initial value 230.967336
iter 10 value 40.553677
iter 20 value 14.209902
iter 30 value 13.787536
iter 40 value 13.333723
iter 50 value 12.978750
iter 60 value 12.959437
iter 70 value 12.908777
iter 80 value 12.907429
iter 90 value 12.902448
final value 12.902417
converged
# weights: 27
initial value 209.289029
iter 10 value 0.787666
iter 20 value 0.092554
iter 30 value 0.081600
iter 40 value 0.080783
iter 50 value 0.078818
iter 60 value 0.076951
iter 70 value 0.076125
iter 80 value 0.075025
iter 90 value 0.074147
iter 100 value 0.073487
final value 0.073487
stopped after 100 iterations
# weights: 43
initial value 245.312238
iter 10 value 0.759429
iter 20 value 0.587366
iter 30 value 0.488156
iter 40 value 0.267279
iter 50 value 0.174189

```

```

iter 60 value 0.142894
iter 70 value 0.123942
iter 80 value 0.102568
iter 90 value 0.097572
iter 100 value 0.092776
final value 0.092776
stopped after 100 iterations
# weights: 11
initial value 201.827197
iter 10 value 75.486825
iter 20 value 72.529679
iter 30 value 72.387216
iter 40 value 68.874699
iter 50 value 61.296112
iter 60 value 27.476564
iter 70 value 20.354137
iter 80 value 19.982256
iter 90 value 19.208951
iter 100 value 19.088076
final value 19.088076
stopped after 100 iterations
# weights: 27
initial value 264.010639
iter 10 value 1.834100
iter 20 value 0.078481
iter 30 value 0.006368
iter 40 value 0.000356
final value 0.000078
converged
# weights: 43
initial value 217.073232
iter 10 value 0.400422
iter 20 value 0.008035
iter 30 value 0.000106
final value 0.000094
converged
# weights: 11
initial value 292.232944
iter 10 value 79.884272
iter 20 value 63.780812
iter 30 value 63.388143
final value 63.384902
converged
# weights: 27
initial value 228.605293
iter 10 value 28.684922

```

```

iter 20 value 19.545331
iter 30 value 19.237663
iter 40 value 19.173389
iter 50 value 19.165331
final value 19.165137
converged
# weights: 43
initial value 282.804981
iter 10 value 50.716957
iter 20 value 17.171599
iter 30 value 16.090218
iter 40 value 15.892189
iter 50 value 15.865006
iter 60 value 15.862523
iter 70 value 15.861911
final value 15.861910
converged
# weights: 11
initial value 220.678965
iter 10 value 147.132973
iter 20 value 118.436638
iter 30 value 118.070253
iter 40 value 117.951853
iter 50 value 117.544139
iter 60 value 117.400169
iter 70 value 117.050001
iter 80 value 117.048725
iter 90 value 117.040191
final value 117.039538
converged
# weights: 27
initial value 249.674578
iter 10 value 6.816643
iter 20 value 0.198768
iter 30 value 0.177474
iter 40 value 0.165113
iter 50 value 0.140277
iter 60 value 0.125801
iter 70 value 0.121382
iter 80 value 0.101688
iter 90 value 0.096255
iter 100 value 0.091205
final value 0.091205
stopped after 100 iterations
# weights: 43
initial value 221.338726

```

```
iter 10 value 7.654887
iter 20 value 0.244672
iter 30 value 0.188519
iter 40 value 0.179353
iter 50 value 0.115279
iter 60 value 0.105865
iter 70 value 0.099890
iter 80 value 0.095813
iter 90 value 0.094906
iter 100 value 0.090947
final value 0.090947
stopped after 100 iterations
# weights: 11
initial value 244.796763
iter 10 value 70.821843
iter 20 value 38.601484
iter 30 value 35.410421
iter 40 value 32.236444
iter 50 value 30.803691
iter 60 value 30.735491
iter 70 value 30.570981
iter 80 value 30.568720
iter 90 value 30.534984
final value 30.534354
converged
# weights: 27
initial value 241.818504
iter 10 value 11.923685
iter 20 value 4.123546
iter 30 value 0.738495
iter 40 value 0.020010
iter 50 value 0.008484
iter 60 value 0.001573
final value 0.000078
converged
# weights: 43
initial value 235.451549
iter 10 value 4.551882
iter 20 value 0.033132
iter 30 value 0.000808
iter 40 value 0.000301
iter 50 value 0.000145
final value 0.000094
converged
# weights: 11
initial value 222.579282
```

```
iter 10 value 110.212919
iter 20 value 77.559728
iter 30 value 62.543437
final value 62.156145
converged
# weights: 27
initial value 217.824008
iter 10 value 42.866402
iter 20 value 21.122173
iter 30 value 19.733425
iter 40 value 19.667688
iter 50 value 19.660939
iter 60 value 19.598578
iter 70 value 19.460903
final value 19.460238
converged
# weights: 43
initial value 223.778866
iter 10 value 32.976583
iter 20 value 16.095442
iter 30 value 15.824753
iter 40 value 15.802884
iter 50 value 15.802528
final value 15.802527
converged
# weights: 11
initial value 245.625005
iter 10 value 83.236747
iter 20 value 80.934447
iter 30 value 80.911420
iter 40 value 80.854751
iter 50 value 77.616251
iter 60 value 72.221857
iter 70 value 59.829208
iter 80 value 19.891106
iter 90 value 13.975919
iter 100 value 13.014070
final value 13.014070
stopped after 100 iterations
# weights: 27
initial value 251.102322
iter 10 value 9.766608
iter 20 value 0.540525
iter 30 value 0.470059
iter 40 value 0.412987
iter 50 value 0.318088
```

```

iter 60 value 0.190395
iter 70 value 0.164123
iter 80 value 0.148074
iter 90 value 0.128041
iter 100 value 0.123772
final value 0.123772
stopped after 100 iterations
# weights: 43
initial value 243.737491
iter 10 value 2.551278
iter 20 value 0.278006
iter 30 value 0.250010
iter 40 value 0.209014
iter 50 value 0.188639
iter 60 value 0.135997
iter 70 value 0.126576
iter 80 value 0.120206
iter 90 value 0.111875
iter 100 value 0.108264
final value 0.108264
stopped after 100 iterations
# weights: 11
initial value 232.160965
iter 10 value 57.788925
iter 20 value 22.247437
iter 30 value 20.416401
iter 40 value 19.440931
iter 50 value 18.859407
iter 60 value 18.743537
iter 70 value 18.714412
iter 80 value 18.683147
iter 90 value 18.649225
iter 100 value 18.620069
final value 18.620069
stopped after 100 iterations
# weights: 27
initial value 233.032221
iter 10 value 3.672803
iter 20 value 0.095195
iter 30 value 0.010029
iter 40 value 0.002314
iter 50 value 0.000141
iter 50 value 0.000095
iter 50 value 0.000095
final value 0.000095
converged

```

```

# weights: 43
initial value 217.121704
iter 10 value 1.371619
iter 20 value 0.055609
iter 30 value 0.006181
iter 40 value 0.000812
iter 50 value 0.000186
iter 60 value 0.000171
final value 0.000089
converged
# weights: 11
initial value 216.438189
iter 10 value 110.280167
iter 20 value 84.597753
iter 30 value 84.174050
final value 84.173316
converged
# weights: 27
initial value 262.517893
iter 10 value 45.153987
iter 20 value 20.126346
iter 30 value 17.867252
iter 40 value 17.792839
iter 50 value 17.792089
final value 17.792070
converged
# weights: 43
initial value 217.808101
iter 10 value 30.444167
iter 20 value 16.125241
iter 30 value 15.016284
iter 40 value 14.593632
iter 50 value 14.527973
iter 60 value 14.521457
iter 70 value 14.521115
final value 14.521110
converged
# weights: 11
initial value 272.304731
iter 10 value 80.803836
iter 20 value 78.997635
iter 30 value 78.823107
iter 40 value 76.651232
iter 50 value 54.573734
iter 60 value 38.248226
iter 70 value 36.354715

```

```
iter 80 value 36.063888
iter 90 value 36.035790
final value 36.026272
converged
# weights: 27
initial value 261.178919
iter 10 value 19.548982
iter 20 value 4.147935
iter 30 value 0.260785
iter 40 value 0.234840
iter 50 value 0.202960
iter 60 value 0.176875
iter 70 value 0.149609
iter 80 value 0.133222
iter 90 value 0.122098
iter 100 value 0.099057
final value 0.099057
stopped after 100 iterations
# weights: 43
initial value 261.837807
iter 10 value 0.809731
iter 20 value 0.286248
iter 30 value 0.258647
iter 40 value 0.199713
iter 50 value 0.149774
iter 60 value 0.120304
iter 70 value 0.108477
iter 80 value 0.095810
iter 90 value 0.089234
iter 100 value 0.084433
final value 0.084433
stopped after 100 iterations
# weights: 11
initial value 271.757381
iter 10 value 62.197095
iter 20 value 23.687057
iter 30 value 22.611846
iter 40 value 21.660558
iter 50 value 21.410629
iter 60 value 21.400388
iter 70 value 21.297752
iter 80 value 21.284889
iter 90 value 21.276155
iter 100 value 21.259718
final value 21.259718
stopped after 100 iterations
```

```

# weights: 27
initial value 219.157378
iter 10 value 4.862372
iter 20 value 0.146582
iter 30 value 0.001303
final value 0.000063
converged
# weights: 43
initial value 232.587331
iter 10 value 0.338206
iter 20 value 0.016998
final value 0.000038
converged
# weights: 11
initial value 219.724130
iter 10 value 93.163233
iter 20 value 84.544525
iter 30 value 83.298151
final value 83.208263
converged
# weights: 27
initial value 228.271366
iter 10 value 27.345178
iter 20 value 17.485927
iter 30 value 17.274663
iter 40 value 17.255368
final value 17.254749
converged
# weights: 43
initial value 235.740350
iter 10 value 20.655651
iter 20 value 15.864200
iter 30 value 15.544059
iter 40 value 15.521009
iter 50 value 15.511836
iter 60 value 15.511231
final value 15.511229
converged
# weights: 11
initial value 221.377450
iter 10 value 80.377414
iter 20 value 33.922781
iter 30 value 25.399771
iter 40 value 22.689210
iter 50 value 21.971911
iter 60 value 21.955929

```

```
iter 70 value 21.900377
final value 21.899888
converged
# weights: 27
initial value 304.530026
iter 10 value 1.764354
iter 20 value 0.121848
iter 30 value 0.111538
iter 40 value 0.109636
iter 50 value 0.104775
iter 60 value 0.102899
iter 70 value 0.098638
iter 80 value 0.096571
iter 90 value 0.094510
iter 100 value 0.091479
final value 0.091479
stopped after 100 iterations
# weights: 43
initial value 274.061099
iter 10 value 3.020523
iter 20 value 0.139121
iter 30 value 0.109859
iter 40 value 0.105135
iter 50 value 0.101050
iter 60 value 0.097753
iter 70 value 0.094298
iter 80 value 0.092895
iter 90 value 0.091093
iter 100 value 0.089510
final value 0.089510
stopped after 100 iterations
# weights: 11
initial value 209.733615
iter 10 value 79.167468
iter 20 value 25.505124
iter 30 value 17.810155
iter 40 value 16.773476
iter 50 value 16.412414
iter 60 value 16.283251
iter 70 value 16.233051
iter 80 value 15.979052
iter 90 value 15.976772
iter 100 value 15.965872
final value 15.965872
stopped after 100 iterations
# weights: 27
```

```
initial value 227.681993
iter 10 value 8.859030
iter 20 value 0.095758
iter 30 value 0.000119
iter 30 value 0.000063
iter 30 value 0.000062
final value 0.000062
converged
# weights: 43
initial value 216.299861
iter 10 value 23.882815
iter 20 value 1.463282
iter 30 value 0.025669
iter 40 value 0.002673
iter 50 value 0.000136
iter 50 value 0.000087
iter 50 value 0.000087
final value 0.000087
converged
# weights: 11
initial value 234.332677
iter 10 value 104.314668
iter 20 value 88.402409
iter 30 value 83.777535
iter 40 value 83.359900
iter 40 value 83.359900
iter 40 value 83.359900
final value 83.359900
converged
# weights: 27
initial value 242.788275
iter 10 value 34.204618
iter 20 value 19.922087
iter 30 value 19.702093
iter 40 value 19.602444
final value 19.602358
converged
# weights: 43
initial value 312.880339
iter 10 value 25.066487
iter 20 value 16.096507
iter 30 value 15.732944
iter 40 value 15.715073
iter 50 value 15.713877
iter 60 value 15.713744
final value 15.713735
```

```

converged
# weights: 11
initial value 284.732108
iter 10 value 78.402109
iter 20 value 77.544113
iter 30 value 77.243716
iter 40 value 74.453565
iter 50 value 61.162185
iter 60 value 38.130470
iter 70 value 36.517649
iter 80 value 36.178995
iter 90 value 36.064215
iter 100 value 36.059569
final value 36.059569
stopped after 100 iterations
# weights: 27
initial value 273.163968
iter 10 value 5.103781
iter 20 value 0.255283
iter 30 value 0.192175
iter 40 value 0.174528
iter 50 value 0.163201
iter 60 value 0.151847
iter 70 value 0.140846
iter 80 value 0.132752
iter 90 value 0.124889
iter 100 value 0.114686
final value 0.114686
stopped after 100 iterations
# weights: 43
initial value 245.129424
iter 10 value 2.670508
iter 20 value 0.138824
iter 30 value 0.099738
iter 40 value 0.097118
iter 50 value 0.094835
iter 60 value 0.093635
iter 70 value 0.092235
iter 80 value 0.090538
iter 90 value 0.089038
iter 100 value 0.088329
final value 0.088329
stopped after 100 iterations
# weights: 11
initial value 216.859473
iter 10 value 87.258848

```

```

iter 20 value 37.941269
iter 30 value 31.081814
iter 40 value 30.902487
iter 50 value 30.785310
final value 30.784558
converged
# weights: 27
initial value 268.224501
iter 10 value 2.700440
iter 20 value 0.012607
final value 0.000092
converged
# weights: 43
initial value 296.823674
iter 10 value 14.341967
iter 20 value 0.069861
iter 30 value 0.004087
iter 40 value 0.001692
iter 50 value 0.000386
iter 60 value 0.000158
iter 70 value 0.000114
iter 70 value 0.000076
iter 70 value 0.000076
final value 0.000076
converged
# weights: 11
initial value 236.450576
iter 10 value 95.750464
iter 20 value 72.142569
iter 30 value 64.244124
iter 40 value 64.242604
final value 64.242567
converged
# weights: 27
initial value 238.249565
iter 10 value 25.395610
iter 20 value 18.985689
iter 30 value 18.870461
final value 18.870363
converged
# weights: 43
initial value 290.417442
iter 10 value 32.144864
iter 20 value 17.113841
iter 30 value 15.464261
iter 40 value 15.322394

```

```
iter 50 value 15.313232
iter 60 value 15.310386
iter 70 value 15.309685
iter 80 value 15.309563
final value 15.309539
converged
# weights: 11
initial value 257.628278
iter 10 value 85.806534
iter 20 value 79.555735
iter 30 value 78.501228
iter 40 value 74.056220
iter 50 value 50.744015
iter 60 value 32.083522
iter 70 value 31.302429
iter 80 value 31.208708
iter 90 value 31.197986
final value 31.197950
converged
# weights: 27
initial value 233.055108
iter 10 value 5.078667
iter 20 value 0.515419
iter 30 value 0.197218
iter 40 value 0.182991
iter 50 value 0.164465
iter 60 value 0.140163
iter 70 value 0.121600
iter 80 value 0.113025
iter 90 value 0.103378
iter 100 value 0.098062
final value 0.098062
stopped after 100 iterations
# weights: 43
initial value 238.247630
iter 10 value 1.334234
iter 20 value 0.184633
iter 30 value 0.173903
iter 40 value 0.161683
iter 50 value 0.152073
iter 60 value 0.120098
iter 70 value 0.103632
iter 80 value 0.100061
iter 90 value 0.096382
iter 100 value 0.093957
final value 0.093957
```

```

stopped after 100 iterations
# weights: 11
initial value 212.543721
iter 10 value 80.224987
iter 20 value 46.521555
iter 30 value 18.174025
iter 40 value 17.542595
iter 50 value 17.402547
iter 60 value 17.200355
iter 70 value 17.108119
iter 80 value 16.770159
iter 90 value 16.623780
iter 100 value 16.561465
final value 16.561465
stopped after 100 iterations
# weights: 27
initial value 212.408430
iter 10 value 8.177971
iter 20 value 0.044736
iter 30 value 0.002041
final value 0.000053
converged
# weights: 43
initial value 204.977866
iter 10 value 8.162433
iter 20 value 0.173920
iter 30 value 0.015607
iter 40 value 0.006500
iter 50 value 0.003597
iter 60 value 0.000958
iter 70 value 0.000687
iter 80 value 0.000634
iter 90 value 0.000130
iter 100 value 0.000103
final value 0.000103
stopped after 100 iterations
# weights: 11
initial value 264.770977
iter 10 value 122.945131
iter 20 value 94.315340
iter 30 value 64.711831
iter 40 value 63.586723
iter 40 value 63.586723
iter 40 value 63.586723
final value 63.586723
converged

```

```

# weights: 27
initial value 250.678493
iter 10 value 37.804217
iter 20 value 20.621162
iter 30 value 18.871345
iter 40 value 18.167793
iter 50 value 18.143252
iter 60 value 18.140759
final value 18.140754
converged
# weights: 43
initial value 208.141810
iter 10 value 26.978875
iter 20 value 17.102744
iter 30 value 16.517689
iter 40 value 16.482330
iter 50 value 16.474528
iter 60 value 16.474330
iter 70 value 16.474240
final value 16.474240
converged
# weights: 11
initial value 269.927007
iter 10 value 81.191056
iter 20 value 80.894365
iter 30 value 80.546710
iter 40 value 79.789756
iter 50 value 79.761078
iter 60 value 79.742763
iter 70 value 79.734268
iter 80 value 79.682802
iter 90 value 79.538377
iter 100 value 78.930385
final value 78.930385
stopped after 100 iterations
# weights: 27
initial value 212.604148
iter 10 value 1.755941
iter 20 value 0.242682
iter 30 value 0.226763
iter 40 value 0.214495
iter 50 value 0.178951
iter 60 value 0.158865
iter 70 value 0.152386
iter 80 value 0.146505
iter 90 value 0.122560

```

```

iter 100 value 0.108105
final  value 0.108105
stopped after 100 iterations
# weights: 43
initial  value 239.416763
iter  10 value 2.735327
iter  20 value 0.118425
iter  30 value 0.102658
iter  40 value 0.100397
iter  50 value 0.099385
iter  60 value 0.098102
iter  70 value 0.096127
iter  80 value 0.094946
iter  90 value 0.092170
iter 100 value 0.091065
final  value 0.091065
stopped after 100 iterations
# weights: 11
initial  value 240.673302
iter  10 value 83.673710
iter  20 value 71.119723
iter  30 value 69.097317
iter  40 value 68.947566
iter  50 value 68.853183
iter  60 value 66.260840
iter  70 value 42.168070
iter  80 value 22.517082
iter  90 value 18.627052
iter 100 value 18.206543
final  value 18.206543
stopped after 100 iterations
# weights: 27
initial  value 244.959529
iter  10 value 3.080465
iter  20 value 0.091415
iter  30 value 0.009554
final  value 0.000085
converged
# weights: 43
initial  value 320.355585
iter  10 value 1.546887
iter  20 value 0.013691
iter  30 value 0.001026
iter  40 value 0.000175
final  value 0.000093
converged

```

```

# weights: 11
initial value 214.381785
iter 10 value 116.132029
iter 20 value 85.399536
iter 30 value 84.356702
final value 84.314590
converged
# weights: 27
initial value 233.886505
iter 10 value 21.066368
iter 20 value 17.416826
iter 30 value 16.698431
iter 40 value 16.678874
iter 50 value 16.678795
final value 16.678794
converged
# weights: 43
initial value 222.028377
iter 10 value 18.779715
iter 20 value 15.904886
iter 30 value 15.581341
iter 40 value 15.512486
iter 50 value 15.497106
iter 60 value 15.496623
final value 15.496321
converged
# weights: 11
initial value 224.728129
iter 10 value 80.636806
iter 20 value 72.994886
iter 30 value 64.869069
iter 40 value 21.917012
iter 50 value 19.324810
iter 60 value 18.746775
iter 70 value 18.538986
iter 80 value 18.509074
iter 90 value 18.494903
iter 100 value 18.481794
final value 18.481794
stopped after 100 iterations
# weights: 27
initial value 215.508492
iter 10 value 1.723308
iter 20 value 0.125199
iter 30 value 0.114198
iter 40 value 0.111415

```

```

iter 50 value 0.108151
iter 60 value 0.102618
iter 70 value 0.098812
iter 80 value 0.094095
iter 90 value 0.086786
iter 100 value 0.080202
final value 0.080202
stopped after 100 iterations
# weights: 43
initial value 222.506055
iter 10 value 4.977817
iter 20 value 0.358574
iter 30 value 0.220559
iter 40 value 0.201394
iter 50 value 0.183500
iter 60 value 0.149318
iter 70 value 0.122781
iter 80 value 0.105405
iter 90 value 0.096883
iter 100 value 0.091362
final value 0.091362
stopped after 100 iterations
# weights: 11
initial value 264.147187
iter 10 value 77.408856
iter 20 value 40.269050
iter 30 value 32.809376
iter 40 value 32.452656
iter 50 value 32.425224
final value 32.424594
converged
# weights: 27
initial value 197.645042
iter 10 value 2.419357
iter 20 value 0.050514
iter 30 value 0.016005
iter 40 value 0.004753
iter 50 value 0.000273
final value 0.000089
converged
# weights: 43
initial value 233.566850
iter 10 value 3.298301
iter 20 value 0.074539
iter 30 value 0.000169
iter 30 value 0.000086

```

```
iter 30 value 0.000086
final value 0.000086
converged
# weights: 11
initial value 269.686129
iter 10 value 70.923593
iter 20 value 64.770699
final value 64.770489
converged
# weights: 27
initial value 253.659388
iter 10 value 40.052301
iter 20 value 22.001425
iter 30 value 18.754325
iter 40 value 18.568919
iter 50 value 18.565012
iter 60 value 18.564922
final value 18.564919
converged
# weights: 43
initial value 272.249197
iter 10 value 45.080531
iter 20 value 16.715396
iter 30 value 15.794153
iter 40 value 15.724561
iter 50 value 15.714999
iter 60 value 15.713218
iter 70 value 15.712967
final value 15.712911
converged
# weights: 11
initial value 233.544219
iter 10 value 134.856583
iter 20 value 84.518223
iter 30 value 83.111384
iter 40 value 82.797071
iter 50 value 82.528424
iter 60 value 82.507117
iter 70 value 82.501510
iter 80 value 82.499763
iter 90 value 82.498547
iter 100 value 82.497417
final value 82.497417
stopped after 100 iterations
# weights: 27
initial value 239.517218
```

```
iter 10 value 3.577215
iter 20 value 0.203841
iter 30 value 0.194293
iter 40 value 0.171529
iter 50 value 0.158716
iter 60 value 0.132969
iter 70 value 0.127573
iter 80 value 0.120408
iter 90 value 0.115741
iter 100 value 0.109081
final value 0.109081
stopped after 100 iterations
# weights: 43
initial value 238.671764
iter 10 value 4.255957
iter 20 value 0.283833
iter 30 value 0.217574
iter 40 value 0.190213
iter 50 value 0.154216
iter 60 value 0.131984
iter 70 value 0.112467
iter 80 value 0.109322
iter 90 value 0.104124
iter 100 value 0.101374
final value 0.101374
stopped after 100 iterations
# weights: 11
initial value 226.906057
iter 10 value 69.284924
iter 20 value 39.867857
iter 30 value 35.535194
iter 40 value 32.603265
iter 50 value 31.410093
iter 60 value 31.268993
iter 70 value 31.229705
iter 80 value 31.223678
iter 90 value 31.189040
iter 100 value 31.078991
final value 31.078991
stopped after 100 iterations
# weights: 27
initial value 216.186890
iter 10 value 5.397450
iter 20 value 0.024669
iter 30 value 0.001757
iter 40 value 0.000203
```

```
iter 50 value 0.000136
iter 60 value 0.000106
final value 0.000098
converged
# weights: 43
initial value 198.190356
iter 10 value 0.891080
iter 20 value 0.023800
final value 0.000092
converged
# weights: 11
initial value 252.249283
iter 10 value 85.208632
iter 20 value 61.551245
iter 30 value 61.450633
final value 61.449935
converged
# weights: 27
initial value 216.056193
iter 10 value 31.963697
iter 20 value 20.151990
iter 30 value 18.938457
iter 40 value 18.897378
iter 50 value 18.894683
iter 60 value 18.894012
final value 18.894008
converged
# weights: 43
initial value 286.400802
iter 10 value 24.102269
iter 20 value 16.609489
iter 30 value 15.955448
iter 40 value 15.832473
iter 50 value 15.821997
iter 60 value 15.821130
iter 70 value 15.821009
final value 15.821009
converged
# weights: 11
initial value 261.421920
iter 10 value 104.104252
iter 20 value 47.496496
iter 30 value 37.874570
iter 40 value 34.306181
iter 50 value 33.883809
iter 60 value 33.709710
```

```

iter 70 value 33.317781
iter 80 value 33.295614
iter 90 value 33.288074
final value 33.283495
converged
# weights: 27
initial value 234.224311
iter 10 value 6.608059
iter 20 value 0.205938
iter 30 value 0.136167
iter 40 value 0.128879
iter 50 value 0.124067
iter 60 value 0.106518
iter 70 value 0.099676
iter 80 value 0.096879
iter 90 value 0.092417
iter 100 value 0.090601
final value 0.090601
stopped after 100 iterations
# weights: 43
initial value 269.504341
iter 10 value 2.250685
iter 20 value 0.132417
iter 30 value 0.112699
iter 40 value 0.108466
iter 50 value 0.103855
iter 60 value 0.094324
iter 70 value 0.089698
iter 80 value 0.086002
iter 90 value 0.084285
iter 100 value 0.082580
final value 0.082580
stopped after 100 iterations
# weights: 11
initial value 221.101294
iter 10 value 80.995281
iter 20 value 79.097114
iter 30 value 74.976297
iter 40 value 46.021062
iter 50 value 36.497970
iter 60 value 35.908568
iter 70 value 35.409239
final value 35.407764
converged
# weights: 27
initial value 232.111976

```

```
iter 10 value 19.146389
iter 20 value 0.315542
iter 30 value 0.007594
iter 40 value 0.000845
iter 50 value 0.000312
final value 0.000075
converged
# weights: 43
initial value 213.003905
iter 10 value 0.466335
iter 20 value 0.008889
iter 30 value 0.000919
final value 0.000056
converged
# weights: 11
initial value 226.794778
iter 10 value 120.971343
iter 20 value 91.617048
iter 30 value 72.146860
iter 40 value 63.881442
final value 63.873764
converged
# weights: 27
initial value 190.014438
iter 10 value 19.924238
iter 20 value 17.403819
iter 30 value 17.025413
iter 40 value 16.935707
final value 16.935550
converged
# weights: 43
initial value 222.002520
iter 10 value 23.908731
iter 20 value 15.346227
iter 30 value 15.228463
iter 40 value 15.193907
iter 50 value 15.191426
iter 60 value 15.191253
final value 15.191250
converged
# weights: 11
initial value 220.292771
iter 10 value 88.298934
iter 20 value 76.425323
iter 30 value 65.532296
iter 40 value 26.035457
```

```

iter 50 value 22.702099
iter 60 value 21.915218
iter 70 value 21.704432
iter 80 value 21.703149
iter 90 value 21.702091
final value 21.702062
converged
# weights: 27
initial value 268.482109
iter 10 value 1.624224
iter 20 value 0.256092
iter 30 value 0.225188
iter 40 value 0.200100
iter 50 value 0.182127
iter 60 value 0.165669
iter 70 value 0.155975
iter 80 value 0.139912
iter 90 value 0.121703
iter 100 value 0.110950
final value 0.110950
stopped after 100 iterations
# weights: 43
initial value 309.988612
iter 10 value 2.459340
iter 20 value 0.212504
iter 30 value 0.189204
iter 40 value 0.158504
iter 50 value 0.119827
iter 60 value 0.103820
iter 70 value 0.094926
iter 80 value 0.085875
iter 90 value 0.083120
iter 100 value 0.082456
final value 0.082456
stopped after 100 iterations
# weights: 11
initial value 231.011158
iter 10 value 83.735535
iter 20 value 41.254589
iter 30 value 35.415946
iter 40 value 35.136640
iter 50 value 34.946606
iter 60 value 34.945476
iter 70 value 34.931794
iter 70 value 34.931794
final value 34.931794

```

```

converged
# weights: 27
initial value 222.603115
iter 10 value 1.219726
iter 20 value 0.043885
iter 30 value 0.010910
iter 40 value 0.000481
final value 0.000086
converged
# weights: 43
initial value 232.090286
iter 10 value 4.195100
iter 20 value 0.091110
iter 30 value 0.002487
iter 40 value 0.001208
iter 50 value 0.000658
final value 0.000086
converged
# weights: 11
initial value 217.232756
iter 10 value 86.383974
iter 20 value 64.322751
final value 63.930268
converged
# weights: 27
initial value 228.997687
iter 10 value 20.356245
iter 20 value 17.593603
iter 30 value 17.268027
iter 40 value 17.200900
final value 17.198673
converged
# weights: 43
initial value 222.778925
iter 10 value 26.402980
iter 20 value 17.746118
iter 30 value 15.999372
iter 40 value 15.729960
iter 50 value 15.602488
iter 60 value 15.594573
iter 70 value 15.593045
final value 15.593044
converged
# weights: 11
initial value 254.751696
iter 10 value 119.475388

```

```
iter 20 value 79.670430
iter 30 value 79.007875
iter 40 value 66.441480
iter 50 value 39.243495
iter 60 value 35.642277
iter 70 value 35.329480
iter 80 value 35.198286
iter 90 value 35.193627
iter 100 value 35.190692
final value 35.190692
stopped after 100 iterations
# weights: 27
initial value 208.110511
iter 10 value 2.167910
iter 20 value 0.124799
iter 30 value 0.118708
iter 40 value 0.113846
iter 50 value 0.110750
iter 60 value 0.106951
iter 70 value 0.101242
iter 80 value 0.098860
iter 90 value 0.097147
iter 100 value 0.095969
final value 0.095969
stopped after 100 iterations
# weights: 43
initial value 285.267211
iter 10 value 2.636082
iter 20 value 0.341274
iter 30 value 0.313510
iter 40 value 0.228736
iter 50 value 0.190986
iter 60 value 0.176353
iter 70 value 0.155667
iter 80 value 0.138469
iter 90 value 0.126162
iter 100 value 0.120911
final value 0.120911
stopped after 100 iterations
# weights: 11
initial value 244.621476
iter 10 value 104.697127
iter 20 value 80.043900
iter 30 value 78.467277
iter 40 value 73.133222
iter 50 value 72.832523
```

```
iter 60 value 72.587470
iter 70 value 72.234782
iter 80 value 65.321815
iter 90 value 23.559711
iter 100 value 15.622533
final value 15.622533
stopped after 100 iterations
# weights: 27
initial value 214.820967
iter 10 value 4.954871
iter 20 value 0.013160
final value 0.000096
converged
# weights: 43
initial value 222.822491
iter 10 value 0.979522
iter 20 value 0.053782
iter 30 value 0.008463
iter 40 value 0.004366
final value 0.000082
converged
# weights: 11
initial value 215.216355
iter 10 value 87.474487
iter 20 value 63.724816
iter 30 value 63.438474
final value 63.426851
converged
# weights: 27
initial value 231.144427
iter 10 value 46.211622
iter 20 value 21.951549
iter 30 value 19.005397
iter 40 value 18.693475
iter 50 value 18.637004
iter 60 value 18.635538
final value 18.635529
converged
# weights: 43
initial value 263.135698
iter 10 value 27.862277
iter 20 value 16.432686
iter 30 value 15.637659
iter 40 value 15.526516
iter 50 value 15.366907
iter 60 value 15.329843
```

```
iter 70 value 15.318236
iter 80 value 15.317941
final value 15.317911
converged
# weights: 11
initial value 250.018481
iter 10 value 155.584097
iter 20 value 81.727489
iter 30 value 80.023332
iter 40 value 78.366128
iter 50 value 76.123421
iter 60 value 74.045812
iter 70 value 73.151662
iter 80 value 72.853619
iter 90 value 71.338942
iter 100 value 40.187431
final value 40.187431
stopped after 100 iterations
# weights: 27
initial value 229.493241
iter 10 value 6.557326
iter 20 value 0.245049
iter 30 value 0.225498
iter 40 value 0.203848
iter 50 value 0.148686
iter 60 value 0.126331
iter 70 value 0.122104
iter 80 value 0.108489
iter 90 value 0.095390
iter 100 value 0.091142
final value 0.091142
stopped after 100 iterations
# weights: 43
initial value 279.293020
iter 10 value 2.479080
iter 20 value 0.118054
iter 30 value 0.102744
iter 40 value 0.096921
iter 50 value 0.089365
iter 60 value 0.084307
iter 70 value 0.080337
iter 80 value 0.078206
iter 90 value 0.077278
iter 100 value 0.076209
final value 0.076209
stopped after 100 iterations
```

```

# weights: 11
initial value 248.215417
iter 10 value 87.484339
iter 20 value 80.668621
final value 80.655187
converged
# weights: 27
initial value 216.805463
iter 10 value 6.670243
iter 20 value 0.031270
iter 30 value 0.002091
final value 0.000096
converged
# weights: 43
initial value 259.179895
iter 10 value 1.289782
iter 20 value 0.019090
iter 30 value 0.000683
final value 0.000092
converged
# weights: 11
initial value 238.703767
iter 10 value 72.520434
iter 20 value 63.943552
iter 30 value 63.890202
final value 63.890194
converged
# weights: 27
initial value 246.711254
iter 10 value 32.507609
iter 20 value 20.793693
iter 30 value 18.146105
iter 40 value 17.976660
iter 50 value 17.974106
iter 60 value 17.973948
final value 17.973942
converged
# weights: 43
initial value 233.732476
iter 10 value 27.963739
iter 20 value 16.952380
iter 30 value 16.467320
iter 40 value 16.329183
iter 50 value 16.257716
iter 60 value 16.246681
final value 16.244743

```

```

converged
# weights: 11
initial value 250.369292
iter 10 value 86.849074
iter 20 value 80.264667
iter 30 value 79.937136
iter 40 value 79.883896
iter 50 value 79.808165
iter 60 value 79.776139
iter 70 value 79.666327
iter 80 value 79.146926
iter 90 value 77.345854
iter 100 value 76.003963
final value 76.003963
stopped after 100 iterations
# weights: 27
initial value 255.140902
iter 10 value 9.509734
iter 20 value 0.251898
iter 30 value 0.212087
iter 40 value 0.204152
iter 50 value 0.183168
iter 60 value 0.122848
iter 70 value 0.117374
iter 80 value 0.113351
iter 90 value 0.106756
iter 100 value 0.104370
final value 0.104370
stopped after 100 iterations
# weights: 43
initial value 223.352702
iter 10 value 2.538384
iter 20 value 0.171394
iter 30 value 0.133310
iter 40 value 0.127097
iter 50 value 0.122329
iter 60 value 0.114651
iter 70 value 0.111367
iter 80 value 0.107316
iter 90 value 0.101418
iter 100 value 0.099087
final value 0.099087
stopped after 100 iterations
# weights: 11
initial value 227.159978
iter 10 value 80.566633

```

```

iter 20 value 77.647601
iter 30 value 77.177477
iter 40 value 72.603908
iter 50 value 58.840553
iter 60 value 41.148297
iter 70 value 38.240104
iter 80 value 37.632069
iter 90 value 37.404026
iter 100 value 37.365336
final value 37.365336
stopped after 100 iterations
# weights: 27
initial value 220.708993
iter 10 value 5.274867
iter 20 value 0.049188
iter 30 value 0.004013
iter 40 value 0.000102
iter 40 value 0.000051
iter 40 value 0.000051
final value 0.000051
converged
# weights: 43
initial value 212.226779
iter 10 value 1.560798
iter 20 value 0.013941
final value 0.000069
converged
# weights: 11
initial value 263.230312
iter 10 value 98.069162
iter 20 value 60.954136
iter 30 value 60.171270
iter 30 value 60.171270
iter 30 value 60.171270
final value 60.171270
converged
# weights: 27
initial value 208.076258
iter 10 value 29.705368
iter 20 value 17.044131
iter 30 value 16.431052
iter 40 value 16.429952
iter 50 value 16.429792
iter 50 value 16.429792
iter 50 value 16.429792
final value 16.429792

```

```

converged
# weights: 43
initial value 273.349865
iter 10 value 20.465010
iter 20 value 16.032821
iter 30 value 15.094436
iter 40 value 14.928722
iter 50 value 14.864335
iter 60 value 14.835500
final value 14.834802
converged
# weights: 11
initial value 223.308417
iter 10 value 82.088025
iter 20 value 74.591843
iter 30 value 73.484426
iter 40 value 70.903983
iter 50 value 69.889314
iter 60 value 69.411207
iter 70 value 69.356041
iter 80 value 69.239525
iter 90 value 48.458828
iter 100 value 15.253117
final value 15.253117
stopped after 100 iterations
# weights: 27
initial value 266.023541
iter 10 value 3.948886
iter 20 value 0.145676
iter 30 value 0.133726
iter 40 value 0.124580
iter 50 value 0.119491
iter 60 value 0.116383
iter 70 value 0.112596
iter 80 value 0.106482
iter 90 value 0.098777
iter 100 value 0.092855
final value 0.092855
stopped after 100 iterations
# weights: 43
initial value 215.163263
iter 10 value 3.578727
iter 20 value 0.488826
iter 30 value 0.444644
iter 40 value 0.308697
iter 50 value 0.183200

```

```

iter 60 value 0.152075
iter 70 value 0.131440
iter 80 value 0.110217
iter 90 value 0.094383
iter 100 value 0.086019
final value 0.086019
stopped after 100 iterations
# weights: 11
initial value 241.202679
iter 10 value 66.037521
iter 20 value 22.829693
iter 30 value 21.028685
iter 40 value 20.413842
iter 50 value 19.898498
iter 60 value 19.810222
iter 70 value 19.699828
iter 80 value 19.683881
iter 90 value 19.625595
iter 100 value 19.543072
final value 19.543072
stopped after 100 iterations
# weights: 27
initial value 219.417713
iter 10 value 22.390922
iter 20 value 0.421819
iter 30 value 0.018334
iter 40 value 0.001981
iter 50 value 0.000467
final value 0.000087
converged
# weights: 43
initial value 208.246923
iter 10 value 2.178653
iter 20 value 0.029995
iter 30 value 0.001327
iter 40 value 0.000596
final value 0.000085
converged
# weights: 11
initial value 203.717952
iter 10 value 99.665332
iter 20 value 65.691802
iter 30 value 63.465719
final value 63.454842
converged
# weights: 27

```

```

initial value 245.380079
iter 10 value 31.097686
iter 20 value 18.566609
iter 30 value 17.308670
iter 40 value 16.831098
iter 50 value 16.777767
final value 16.777763
converged
# weights: 43
initial value 264.045354
iter 10 value 37.328885
iter 20 value 16.677975
iter 30 value 15.561691
iter 40 value 15.307264
iter 50 value 15.252551
iter 60 value 15.237094
iter 70 value 15.236893
final value 15.236893
converged
# weights: 11
initial value 235.542845
iter 10 value 80.769327
iter 20 value 80.750514
iter 30 value 80.727170
iter 40 value 80.582030
iter 50 value 79.775354
iter 60 value 78.511241
iter 70 value 77.015258
iter 80 value 52.044696
iter 90 value 37.590146
iter 100 value 35.745981
final value 35.745981
stopped after 100 iterations
# weights: 27
initial value 262.441921
iter 10 value 2.726389
iter 20 value 0.124394
iter 30 value 0.115110
iter 40 value 0.109630
iter 50 value 0.101646
iter 60 value 0.094771
iter 70 value 0.088574
iter 80 value 0.086238
iter 90 value 0.082473
iter 100 value 0.080678
final value 0.080678

```

```
stopped after 100 iterations
# weights: 43
initial value 220.011093
iter 10 value 3.033027
iter 20 value 0.157260
iter 30 value 0.141122
iter 40 value 0.126294
iter 50 value 0.105436
iter 60 value 0.099453
iter 70 value 0.092900
iter 80 value 0.086521
iter 90 value 0.083968
iter 100 value 0.082278
final value 0.082278
stopped after 100 iterations
# weights: 11
initial value 212.840845
iter 10 value 74.812882
iter 20 value 60.249888
iter 30 value 26.622668
iter 40 value 21.125847
iter 50 value 20.844176
iter 60 value 20.388609
iter 70 value 20.367029
iter 80 value 20.265924
iter 90 value 20.247591
iter 100 value 20.231682
final value 20.231682
stopped after 100 iterations
# weights: 27
initial value 241.687746
iter 10 value 16.601499
iter 20 value 0.347755
iter 30 value 0.005627
iter 40 value 0.000699
iter 50 value 0.000355
final value 0.000088
converged
# weights: 43
initial value 226.357479
iter 10 value 7.180801
iter 20 value 0.022762
iter 30 value 0.001246
iter 40 value 0.000202
final value 0.000079
converged
```

```

# weights: 11
initial value 214.837618
iter 10 value 111.769914
iter 20 value 71.438452
iter 30 value 63.838706
final value 63.837637
converged
# weights: 27
initial value 291.538843
iter 10 value 34.632233
iter 20 value 21.532913
iter 30 value 20.125143
iter 40 value 20.042228
iter 50 value 20.037059
iter 50 value 20.037059
iter 50 value 20.037059
final value 20.037059
converged
# weights: 43
initial value 225.266304
iter 10 value 26.411971
iter 20 value 17.374437
iter 30 value 16.590030
iter 40 value 16.564471
iter 50 value 16.561169
iter 60 value 16.560656
iter 70 value 16.560648
iter 70 value 16.560648
iter 70 value 16.560648
final value 16.560648
converged
# weights: 11
initial value 241.039646
iter 10 value 83.229207
iter 20 value 80.926640
iter 30 value 80.897762
iter 40 value 80.862704
iter 50 value 80.836864
iter 60 value 79.860501
iter 70 value 76.547823
iter 80 value 74.889334
iter 90 value 71.147744
iter 100 value 35.033301
final value 35.033301
stopped after 100 iterations
# weights: 27

```

```

initial value 232.265127
iter 10 value 8.801272
iter 20 value 0.184918
iter 30 value 0.168774
iter 40 value 0.158498
iter 50 value 0.143931
iter 60 value 0.126583
iter 70 value 0.122939
iter 80 value 0.114602
iter 90 value 0.103698
iter 100 value 0.101841
final value 0.101841
stopped after 100 iterations
# weights: 43
initial value 263.186672
iter 10 value 1.313064
iter 20 value 0.168932
iter 30 value 0.148108
iter 40 value 0.143982
iter 50 value 0.125777
iter 60 value 0.122145
iter 70 value 0.112367
iter 80 value 0.104009
iter 90 value 0.101577
iter 100 value 0.099679
final value 0.099679
stopped after 100 iterations
# weights: 11
initial value 198.120168
iter 10 value 81.103597
iter 20 value 79.497925
iter 30 value 79.490252
iter 40 value 79.460198
iter 50 value 78.648778
iter 60 value 76.577415
iter 70 value 30.980473
iter 80 value 17.133870
iter 90 value 16.360670
iter 100 value 16.096320
final value 16.096320
stopped after 100 iterations
# weights: 27
initial value 233.358551
iter 10 value 16.656656
iter 20 value 3.115194
iter 30 value 0.038777

```

```
iter 40 value 0.004797
iter 50 value 0.001554
final value 0.000058
converged
# weights: 43
initial value 242.424027
iter 10 value 2.345915
iter 20 value 0.022681
iter 30 value 0.004133
iter 40 value 0.000911
final value 0.000085
converged
# weights: 11
initial value 224.126130
iter 10 value 116.643697
iter 20 value 89.112930
iter 30 value 84.255720
final value 83.263433
converged
# weights: 27
initial value 218.678342
iter 10 value 26.453730
iter 20 value 19.606625
iter 30 value 18.348117
iter 40 value 18.227942
iter 50 value 18.188134
iter 60 value 17.805115
iter 70 value 16.947198
iter 80 value 16.854483
final value 16.854482
converged
# weights: 43
initial value 228.788171
iter 10 value 42.078952
iter 20 value 16.388126
iter 30 value 15.265019
iter 40 value 15.075002
iter 50 value 15.060355
iter 60 value 15.052502
iter 70 value 15.051955
iter 80 value 15.051831
iter 80 value 15.051831
iter 80 value 15.051831
final value 15.051831
converged
# weights: 11
```

```

initial value 258.471581
iter 10 value 60.612213
iter 20 value 19.007845
iter 30 value 17.169933
iter 40 value 16.682660
iter 50 value 16.399909
iter 60 value 16.374683
iter 70 value 16.371702
final value 16.371122
converged
# weights: 27
initial value 232.421772
iter 10 value 4.046875
iter 20 value 0.310238
iter 30 value 0.258448
iter 40 value 0.230891
iter 50 value 0.203866
iter 60 value 0.188369
iter 70 value 0.172405
iter 80 value 0.141153
iter 90 value 0.125753
iter 100 value 0.121316
final value 0.121316
stopped after 100 iterations
# weights: 43
initial value 203.980559
iter 10 value 2.782239
iter 20 value 0.122655
iter 30 value 0.111930
iter 40 value 0.105697
iter 50 value 0.100735
iter 60 value 0.098304
iter 70 value 0.094263
iter 80 value 0.091764
iter 90 value 0.089537
iter 100 value 0.088565
final value 0.088565
stopped after 100 iterations
# weights: 11
initial value 231.503729
iter 10 value 94.336333
iter 20 value 61.282740
iter 30 value 39.484712
iter 40 value 38.116234
iter 50 value 38.068551
iter 60 value 38.046383

```

```
iter 70 value 37.995107
iter 80 value 37.994396
iter 90 value 37.925340
iter 100 value 37.885506
final value 37.885506
stopped after 100 iterations
# weights: 27
initial value 273.504680
iter 10 value 1.572601
iter 20 value 0.030069
iter 30 value 0.005143
iter 40 value 0.000423
final value 0.000088
converged
# weights: 43
initial value 257.118412
iter 10 value 0.885250
iter 20 value 0.011911
iter 30 value 0.001709
iter 40 value 0.000262
final value 0.000088
converged
# weights: 11
initial value 226.825836
iter 10 value 94.376982
iter 20 value 71.341330
iter 30 value 62.854878
final value 62.849235
converged
# weights: 27
initial value 253.434855
iter 10 value 25.545328
iter 20 value 18.449772
iter 30 value 17.496586
iter 40 value 17.472125
iter 50 value 17.471136
final value 17.471135
converged
# weights: 43
initial value 266.341546
iter 10 value 20.813728
iter 20 value 16.478358
iter 30 value 16.031794
iter 40 value 15.769625
iter 50 value 15.759659
iter 60 value 15.758411
```

```
final value 15.758382
converged
# weights: 11
initial value 223.983671
iter 10 value 84.830861
iter 20 value 79.711248
iter 30 value 79.084447
iter 40 value 78.540891
iter 50 value 77.490648
iter 60 value 77.351156
iter 70 value 76.416831
iter 80 value 76.208505
iter 90 value 76.167721
iter 100 value 76.086142
final value 76.086142
stopped after 100 iterations
# weights: 27
initial value 252.155929
iter 10 value 6.667920
iter 20 value 0.285485
iter 30 value 0.241178
iter 40 value 0.206851
iter 50 value 0.168234
iter 60 value 0.142094
iter 70 value 0.133925
iter 80 value 0.128049
iter 90 value 0.120798
iter 100 value 0.114021
final value 0.114021
stopped after 100 iterations
# weights: 43
initial value 233.746274
iter 10 value 2.379044
iter 20 value 0.126222
iter 30 value 0.114577
iter 40 value 0.110919
iter 50 value 0.105914
iter 60 value 0.101507
iter 70 value 0.097481
iter 80 value 0.090452
iter 90 value 0.089737
iter 100 value 0.087383
final value 0.087383
stopped after 100 iterations
# weights: 11
initial value 212.265225
```

```
iter 10 value 79.554088
iter 20 value 63.415502
iter 30 value 38.691975
iter 40 value 20.038899
iter 50 value 19.552168
iter 60 value 19.136564
iter 70 value 19.122107
iter 80 value 19.105167
iter 90 value 18.715794
iter 100 value 18.610736
final value 18.610736
stopped after 100 iterations
# weights: 27
initial value 244.328317
iter 10 value 7.652757
iter 20 value 1.029107
iter 30 value 0.018288
iter 40 value 0.001206
final value 0.000059
converged
# weights: 43
initial value 249.792683
iter 10 value 1.973329
iter 20 value 0.018797
iter 30 value 0.001692
final value 0.000090
converged
# weights: 11
initial value 232.778059
iter 10 value 69.261944
iter 20 value 64.157615
final value 64.155713
converged
# weights: 27
initial value 302.779267
iter 10 value 59.851320
iter 20 value 19.628711
iter 30 value 17.567550
iter 40 value 16.949792
iter 50 value 16.910972
iter 60 value 16.906959
final value 16.906957
converged
# weights: 43
initial value 254.412376
iter 10 value 20.566838
```

```

iter 20 value 15.573398
iter 30 value 15.317239
iter 40 value 15.305747
iter 50 value 15.303167
final value 15.303094
converged
# weights: 11
initial value 213.558004
iter 10 value 51.381149
iter 20 value 20.863551
iter 30 value 19.645084
iter 40 value 19.292314
iter 50 value 19.194271
iter 60 value 19.103703
iter 70 value 19.047203
final value 19.031778
converged
# weights: 27
initial value 250.546391
iter 10 value 0.984387
iter 20 value 0.522743
iter 30 value 0.462063
iter 40 value 0.330127
iter 50 value 0.240397
iter 60 value 0.149258
iter 70 value 0.134379
iter 80 value 0.126832
iter 90 value 0.117297
iter 100 value 0.098154
final value 0.098154
stopped after 100 iterations
# weights: 43
initial value 218.517994
iter 10 value 1.300215
iter 20 value 0.133358
iter 30 value 0.117087
iter 40 value 0.114229
iter 50 value 0.108908
iter 60 value 0.102974
iter 70 value 0.098027
iter 80 value 0.093852
iter 90 value 0.091624
iter 100 value 0.090227
final value 0.090227
stopped after 100 iterations
# weights: 11

```

```

initial value 237.510176
iter 10 value 78.167537
iter 20 value 74.625584
iter 30 value 73.297065
iter 40 value 69.750440
iter 50 value 62.577681
iter 60 value 28.121687
iter 70 value 18.135097
iter 80 value 17.788619
iter 90 value 17.389085
iter 100 value 17.007677
final value 17.007677
stopped after 100 iterations
# weights: 27
initial value 311.771345
iter 10 value 2.672137
iter 20 value 0.059409
iter 30 value 0.007449
iter 40 value 0.000540
final value 0.000083
converged
# weights: 43
initial value 249.086353
iter 10 value 3.481655
iter 20 value 0.013982
iter 30 value 0.000546
final value 0.000064
converged
# weights: 11
initial value 223.718356
iter 10 value 102.236745
iter 20 value 77.763888
iter 30 value 62.389077
final value 62.341589
converged
# weights: 27
initial value 235.128147
iter 10 value 24.224029
iter 20 value 17.396330
iter 30 value 16.693373
iter 40 value 16.643090
final value 16.642690
converged
# weights: 43
initial value 255.910370
iter 10 value 39.841408

```

```
iter 20 value 16.972497
iter 30 value 15.454650
iter 40 value 15.264022
iter 50 value 15.024515
iter 60 value 15.003373
iter 70 value 15.002316
final value 15.002305
converged
# weights: 11
initial value 227.179840
iter 10 value 86.615118
iter 20 value 55.240440
iter 30 value 21.784652
iter 40 value 18.494250
iter 50 value 18.268466
iter 60 value 17.828181
iter 70 value 17.729098
iter 80 value 17.722842
final value 17.722565
converged
# weights: 27
initial value 256.609884
iter 10 value 9.099527
iter 20 value 0.420317
iter 30 value 0.358625
iter 40 value 0.299371
iter 50 value 0.273634
iter 60 value 0.199334
iter 70 value 0.159932
iter 80 value 0.140728
iter 90 value 0.133047
iter 100 value 0.114065
final value 0.114065
stopped after 100 iterations
# weights: 43
initial value 261.835256
iter 10 value 1.056388
iter 20 value 0.118736
iter 30 value 0.108893
iter 40 value 0.098054
iter 50 value 0.088302
iter 60 value 0.083254
iter 70 value 0.076280
iter 80 value 0.072793
iter 90 value 0.070701
iter 100 value 0.070026
```

```

final value 0.070026
stopped after 100 iterations
# weights: 11
initial value 240.322509
iter 10 value 130.442142
iter 20 value 116.951829
iter 30 value 115.940134
iter 40 value 115.417230
iter 50 value 111.534090
iter 60 value 47.560226
iter 70 value 40.405054
iter 80 value 40.313811
iter 90 value 40.265690
iter 100 value 40.190811
final value 40.190811
stopped after 100 iterations
# weights: 27
initial value 223.486455
iter 10 value 14.411213
iter 20 value 1.171536
iter 30 value 0.033839
iter 40 value 0.003551
iter 50 value 0.000590
final value 0.000084
converged
# weights: 43
initial value 247.234861
iter 10 value 1.121937
iter 20 value 0.036793
iter 30 value 0.003426
iter 40 value 0.000213
final value 0.000071
converged
# weights: 11
initial value 213.411920
iter 10 value 124.050628
iter 20 value 89.115126
iter 30 value 61.913002
iter 40 value 60.916442
iter 40 value 60.916441
iter 40 value 60.916441
final value 60.916441
converged
# weights: 27
initial value 257.788046
iter 10 value 30.008070

```

```

iter 20 value 20.187039
iter 30 value 20.033102
iter 40 value 20.024957
final value 20.024592
converged
# weights: 43
initial value 184.212497
iter 10 value 21.413970
iter 20 value 16.823851
iter 30 value 16.543994
iter 40 value 16.526670
iter 50 value 16.522394
final value 16.522393
converged
# weights: 11
initial value 213.713506
iter 10 value 106.766741
iter 20 value 78.990665
iter 30 value 77.928111
iter 40 value 77.319747
iter 50 value 76.315539
iter 60 value 60.037537
iter 70 value 41.644402
iter 80 value 40.733706
iter 90 value 40.592098
iter 100 value 40.573520
final value 40.573520
stopped after 100 iterations
# weights: 27
initial value 313.537929
iter 10 value 4.802529
iter 20 value 0.204441
iter 30 value 0.192620
iter 40 value 0.185232
iter 50 value 0.179956
iter 60 value 0.168101
iter 70 value 0.160915
iter 80 value 0.141643
iter 90 value 0.121739
iter 100 value 0.115385
final value 0.115385
stopped after 100 iterations
# weights: 43
initial value 219.464260
iter 10 value 2.027237
iter 20 value 0.137145

```

```
iter 30 value 0.118563
iter 40 value 0.115783
iter 50 value 0.110744
iter 60 value 0.106339
iter 70 value 0.101238
iter 80 value 0.098847
iter 90 value 0.098095
iter 100 value 0.096963
final value 0.096963
stopped after 100 iterations
# weights: 11
initial value 211.830632
iter 10 value 78.895675
iter 20 value 73.581603
final value 73.566154
converged
# weights: 27
initial value 241.900445
iter 10 value 1.056530
iter 20 value 0.002734
final value 0.000085
converged
# weights: 43
initial value 204.153178
iter 10 value 0.815451
iter 20 value 0.013165
iter 30 value 0.000292
iter 40 value 0.000137
final value 0.000076
converged
# weights: 11
initial value 287.510502
iter 10 value 96.967952
iter 20 value 78.688443
iter 30 value 63.423739
iter 40 value 62.585137
iter 40 value 62.585137
iter 40 value 62.585137
final value 62.585137
converged
# weights: 27
initial value 241.960765
iter 10 value 31.042167
iter 20 value 16.610898
iter 30 value 16.412026
iter 40 value 16.348897
```

```
iter 50 value 16.227873
iter 60 value 16.212696
final value 16.212631
converged
# weights: 43
initial value 266.583636
iter 10 value 29.618702
iter 20 value 15.290613
iter 30 value 14.602847
iter 40 value 14.388181
iter 50 value 14.363256
iter 60 value 14.358141
iter 70 value 14.356291
final value 14.356278
converged
# weights: 11
initial value 212.809676
iter 10 value 73.591837
iter 20 value 25.681550
iter 30 value 19.397713
iter 40 value 17.664159
iter 50 value 17.521870
iter 60 value 17.511484
iter 70 value 17.501936
final value 17.501935
converged
# weights: 27
initial value 234.561570
iter 10 value 0.821212
iter 20 value 0.096564
iter 30 value 0.092798
iter 40 value 0.088705
iter 50 value 0.087250
iter 60 value 0.083336
iter 70 value 0.080134
iter 80 value 0.075655
iter 90 value 0.072467
iter 100 value 0.071038
final value 0.071038
stopped after 100 iterations
# weights: 43
initial value 253.521354
iter 10 value 2.109364
iter 20 value 0.127403
iter 30 value 0.117454
iter 40 value 0.101900
```

```
iter 50 value 0.090023
iter 60 value 0.085020
iter 70 value 0.078834
iter 80 value 0.074273
iter 90 value 0.069851
iter 100 value 0.068397
final value 0.068397
stopped after 100 iterations
# weights: 11
initial value 215.811606
iter 10 value 82.317100
iter 20 value 80.617301
iter 30 value 79.292727
iter 40 value 77.965213
iter 50 value 77.222478
iter 60 value 77.000152
iter 70 value 74.964449
iter 80 value 74.092665
iter 90 value 73.742785
iter 100 value 73.577755
final value 73.577755
stopped after 100 iterations
# weights: 27
initial value 235.151194
iter 10 value 6.782380
iter 20 value 0.388613
iter 30 value 0.001104
final value 0.000068
converged
# weights: 43
initial value 193.380732
iter 10 value 1.142331
iter 20 value 0.003051
iter 30 value 0.000610
iter 40 value 0.000330
final value 0.000072
converged
# weights: 11
initial value 221.707885
iter 10 value 119.533606
iter 20 value 66.744356
iter 30 value 64.840970
final value 64.725619
converged
# weights: 27
initial value 277.285548
```

```
iter 10 value 39.642843
iter 20 value 20.412628
iter 30 value 17.823158
iter 40 value 17.527905
iter 50 value 17.487902
iter 60 value 17.479549
iter 70 value 17.478851
final value 17.478851
converged
# weights: 43
initial value 211.037794
iter 10 value 30.002217
iter 20 value 16.170059
iter 30 value 15.814787
iter 40 value 15.783362
iter 50 value 15.780677
iter 60 value 15.780161
iter 60 value 15.780161
iter 60 value 15.780161
final value 15.780161
converged
# weights: 11
initial value 223.711660
iter 10 value 87.739776
iter 20 value 80.866925
iter 30 value 80.852840
iter 40 value 80.829584
iter 50 value 80.736252
iter 60 value 78.869226
iter 70 value 76.409864
iter 80 value 69.585068
iter 90 value 31.430155
iter 100 value 24.347968
final value 24.347968
stopped after 100 iterations
# weights: 27
initial value 277.932940
iter 10 value 7.847664
iter 20 value 0.236711
iter 30 value 0.177293
iter 40 value 0.154009
iter 50 value 0.150156
iter 60 value 0.145430
iter 70 value 0.140190
iter 80 value 0.133275
iter 90 value 0.119495
```

```

iter 100 value 0.114180
final  value 0.114180
stopped after 100 iterations
# weights: 43
initial  value 264.651620
iter  10 value 25.455954
iter  20 value 16.769249
iter  30 value 3.289033
iter  40 value 0.879140
iter  50 value 0.816147
iter  60 value 0.764759
iter  70 value 0.525246
iter  80 value 0.339221
iter  90 value 0.251171
iter 100 value 0.221978
final  value 0.221978
stopped after 100 iterations
# weights: 11
initial  value 220.525790
iter  10 value 91.297317
iter  20 value 84.598122
iter  30 value 82.287199
iter  40 value 61.137197
iter  50 value 25.470832
iter  60 value 23.083992
iter  70 value 22.056466
iter  80 value 21.663738
iter  90 value 21.648744
iter 100 value 21.579425
final  value 21.579425
stopped after 100 iterations
# weights: 27
initial  value 315.403542
iter  10 value 10.652956
iter  20 value 0.414060
iter  30 value 0.018294
iter  40 value 0.001463
final  value 0.000059
converged
# weights: 43
initial  value 189.810146
iter  10 value 2.747410
iter  20 value 0.001784
iter  30 value 0.000251
final  value 0.000092
converged

```

```

# weights: 11
initial value 231.904594
iter 10 value 104.138433
iter 20 value 77.889589
iter 30 value 66.379845
iter 40 value 65.208803
final value 65.208792
converged
# weights: 27
initial value 236.630286
iter 10 value 40.171739
iter 20 value 20.722699
iter 30 value 18.712978
iter 40 value 17.831613
iter 50 value 17.541240
iter 60 value 17.511349
iter 70 value 17.509513
final value 17.509513
converged
# weights: 43
initial value 256.192249
iter 10 value 26.824933
iter 20 value 16.644026
iter 30 value 16.323930
iter 40 value 15.982725
iter 50 value 15.920681
iter 60 value 15.916716
iter 70 value 15.915507
final value 15.915490
converged
# weights: 11
initial value 236.648457
iter 10 value 83.236262
iter 20 value 80.718272
iter 30 value 80.716752
iter 40 value 80.709890
iter 50 value 80.557951
iter 60 value 78.483399
iter 70 value 52.649156
iter 80 value 39.579233
iter 90 value 38.453779
iter 100 value 38.358118
final value 38.358118
stopped after 100 iterations
# weights: 27
initial value 229.850767

```

```

iter 10 value 2.324730
iter 20 value 0.147034
iter 30 value 0.124299
iter 40 value 0.120861
iter 50 value 0.111215
iter 60 value 0.109024
iter 70 value 0.106308
iter 80 value 0.104147
iter 90 value 0.102935
iter 100 value 0.102298
final value 0.102298
stopped after 100 iterations
# weights: 43
initial value 234.250622
iter 10 value 2.091051
iter 20 value 0.116406
iter 30 value 0.099246
iter 40 value 0.096261
iter 50 value 0.094979
iter 60 value 0.093018
iter 70 value 0.089990
iter 80 value 0.088141
iter 90 value 0.086815
iter 100 value 0.085153
final value 0.085153
stopped after 100 iterations
# weights: 11
initial value 289.015913
iter 10 value 110.760729
iter 20 value 82.475153
iter 30 value 80.341408
iter 40 value 79.739498
iter 50 value 78.496532
iter 60 value 77.903448
iter 70 value 77.360684
iter 80 value 45.897447
iter 90 value 38.312604
iter 100 value 38.070984
final value 38.070984
stopped after 100 iterations
# weights: 27
initial value 279.085657
iter 10 value 5.010680
iter 20 value 0.097485
iter 30 value 0.004674
iter 40 value 0.001102

```

```

iter 50 value 0.000433
iter 60 value 0.000156
iter 60 value 0.000091
iter 60 value 0.000091
final value 0.000091
converged
# weights: 43
initial value 258.736009
iter 10 value 14.230885
iter 20 value 0.370011
iter 30 value 0.016300
iter 40 value 0.000905
iter 50 value 0.000241
iter 60 value 0.000220
final value 0.000090
converged
# weights: 11
initial value 265.890482
iter 10 value 116.322964
iter 20 value 93.705476
iter 30 value 86.872175
iter 40 value 74.593209
iter 50 value 62.160227
iter 60 value 62.159632
iter 60 value 62.159632
iter 60 value 62.159632
final value 62.159632
converged
# weights: 27
initial value 241.054260
iter 10 value 36.903854
iter 20 value 19.335151
iter 30 value 19.284947
iter 40 value 19.248632
iter 50 value 18.874858
iter 60 value 18.777782
iter 70 value 18.770739
final value 18.770714
converged
# weights: 43
initial value 285.323038
iter 10 value 29.221378
iter 20 value 16.641131
iter 30 value 15.954498
iter 40 value 15.753808
iter 50 value 15.741469

```

```

iter 60 value 15.740972
final value 15.740968
converged
# weights: 11
initial value 229.200555
iter 10 value 80.734294
iter 20 value 53.996923
iter 30 value 38.594938
iter 40 value 38.247088
iter 50 value 38.220904
iter 60 value 38.220542
iter 70 value 38.220113
iter 80 value 38.219407
final value 38.219406
converged
# weights: 27
initial value 276.604301
iter 10 value 1.370199
iter 20 value 0.152474
iter 30 value 0.124793
iter 40 value 0.116052
iter 50 value 0.112328
iter 60 value 0.107279
iter 70 value 0.105590
iter 80 value 0.100443
iter 90 value 0.097922
iter 100 value 0.095359
final value 0.095359
stopped after 100 iterations
# weights: 43
initial value 263.504732
iter 10 value 2.367703
iter 20 value 0.127903
iter 30 value 0.107752
iter 40 value 0.102072
iter 50 value 0.090509
iter 60 value 0.086866
iter 70 value 0.083854
iter 80 value 0.081333
iter 90 value 0.079711
iter 100 value 0.078617
final value 0.078617
stopped after 100 iterations
# weights: 11
initial value 223.340227
iter 10 value 71.793469

```

```
iter 20 value 22.942574
iter 30 value 10.959379
iter 40 value 10.226512
iter 50 value 9.938780
iter 60 value 9.837098
iter 70 value 9.777919
iter 80 value 9.727658
iter 90 value 9.486721
iter 100 value 9.318393
final value 9.318393
stopped after 100 iterations
# weights: 27
initial value 212.831427
iter 10 value 3.955400
iter 20 value 0.035635
iter 30 value 0.000409
final value 0.000073
converged
# weights: 43
initial value 232.325327
iter 10 value 1.409431
iter 20 value 0.021082
iter 30 value 0.000942
final value 0.000081
converged
# weights: 11
initial value 208.390820
iter 10 value 92.836579
iter 20 value 62.648642
final value 62.464906
converged
# weights: 27
initial value 222.929430
iter 10 value 56.508831
iter 20 value 22.258882
iter 30 value 20.220294
iter 40 value 18.809957
iter 50 value 17.396136
iter 60 value 17.366096
iter 70 value 17.364358
iter 70 value 17.364357
final value 17.364357
converged
# weights: 43
initial value 252.726356
```

```
iter 10 value 20.836416
iter 20 value 15.934053
iter 30 value 15.779187
iter 40 value 15.672689
iter 50 value 15.640396
iter 60 value 15.633346
final value 15.633342
converged
# weights: 11
initial value 224.310146
iter 10 value 43.606876
iter 20 value 12.062649
iter 30 value 10.809034
iter 40 value 10.710770
iter 50 value 10.622422
iter 60 value 10.619746
iter 70 value 10.607187
iter 80 value 10.605038
iter 90 value 10.604569
iter 100 value 10.603772
final value 10.603772
stopped after 100 iterations
# weights: 27
initial value 203.982998
iter 10 value 6.866722
iter 20 value 0.290525
iter 30 value 0.207739
iter 40 value 0.196408
iter 50 value 0.173571
iter 60 value 0.149332
iter 70 value 0.142813
iter 80 value 0.135704
iter 90 value 0.118947
iter 100 value 0.098272
final value 0.098272
stopped after 100 iterations
# weights: 43
initial value 252.304859
iter 10 value 3.816153
iter 20 value 0.161017
iter 30 value 0.117860
iter 40 value 0.115218
iter 50 value 0.113181
iter 60 value 0.110249
iter 70 value 0.107352
iter 80 value 0.105617
```

```

iter 90 value 0.104337
iter 100 value 0.101913
final value 0.101913
stopped after 100 iterations
# weights: 11
initial value 218.867635
iter 10 value 81.516803
iter 20 value 79.348720
iter 30 value 78.715870
iter 40 value 78.685017
iter 50 value 78.571222
iter 60 value 78.435028
iter 70 value 78.421240
iter 80 value 78.374325
iter 90 value 78.281400
iter 100 value 75.637772
final value 75.637772
stopped after 100 iterations
# weights: 27
initial value 271.290773
iter 10 value 1.332127
iter 20 value 0.024560
iter 30 value 0.007212
iter 40 value 0.000232
final value 0.000060
converged
# weights: 43
initial value 204.117377
iter 10 value 2.116588
iter 20 value 0.026793
iter 30 value 0.000332
final value 0.000073
converged
# weights: 11
initial value 243.254701
iter 10 value 95.114007
iter 20 value 87.703178
iter 30 value 63.566664
final value 63.190530
converged
# weights: 27
initial value 222.870334
iter 10 value 32.036538
iter 20 value 19.562219
iter 30 value 18.777867
iter 40 value 18.736567

```

```
iter 50 value 18.728777
final value 18.728775
converged
# weights: 43
initial value 210.121166
iter 10 value 19.437555
iter 20 value 15.363058
iter 30 value 15.154240
iter 40 value 15.117535
iter 50 value 15.116751
final value 15.116743
converged
# weights: 11
initial value 263.599847
iter 10 value 81.532876
iter 20 value 36.706027
iter 30 value 22.558706
iter 40 value 21.053445
iter 50 value 20.828136
iter 60 value 20.818518
iter 70 value 20.809478
iter 70 value 20.809478
final value 20.809478
converged
# weights: 27
initial value 250.637277
iter 10 value 8.472102
iter 20 value 0.269247
iter 30 value 0.209534
iter 40 value 0.199351
iter 50 value 0.171022
iter 60 value 0.154277
iter 70 value 0.147467
iter 80 value 0.139579
iter 90 value 0.116644
iter 100 value 0.110218
final value 0.110218
stopped after 100 iterations
# weights: 43
initial value 249.159196
iter 10 value 4.611687
iter 20 value 0.140145
iter 30 value 0.114682
iter 40 value 0.110276
iter 50 value 0.106390
iter 60 value 0.102918
```

```
iter 70 value 0.099127
iter 80 value 0.095764
iter 90 value 0.092831
iter 100 value 0.090708
final value 0.090708
stopped after 100 iterations
# weights: 11
initial value 238.050211
iter 10 value 81.022238
iter 20 value 63.019230
iter 30 value 33.821147
iter 40 value 33.425056
iter 50 value 33.380718
iter 60 value 33.311670
iter 70 value 33.308500
final value 33.308493
converged
# weights: 27
initial value 221.365962
iter 10 value 1.605728
iter 20 value 0.017157
iter 30 value 0.000231
final value 0.000095
converged
# weights: 43
initial value 274.191240
iter 10 value 5.954635
iter 20 value 0.049812
final value 0.000091
converged
# weights: 11
initial value 236.841216
iter 10 value 90.411888
iter 20 value 66.972991
iter 30 value 65.399673
final value 65.301933
converged
# weights: 27
initial value 217.684245
iter 10 value 30.951210
iter 20 value 19.691158
iter 30 value 19.000651
iter 40 value 18.885327
iter 50 value 18.876490
final value 18.876477
converged
```

```

# weights: 43
initial value 224.945177
iter 10 value 41.557266
iter 20 value 19.058158
iter 30 value 16.963593
iter 40 value 15.914149
iter 50 value 15.795404
iter 60 value 15.673072
iter 70 value 15.490376
iter 80 value 15.382435
iter 90 value 15.379888
iter 100 value 15.379816
final value 15.379816
stopped after 100 iterations
# weights: 11
initial value 260.323342
iter 10 value 56.820488
iter 20 value 33.078301
iter 30 value 24.350853
iter 40 value 20.587918
iter 50 value 20.377281
iter 60 value 20.367312
iter 70 value 20.360289
iter 70 value 20.360289
final value 20.360289
converged
# weights: 27
initial value 196.605061
iter 10 value 0.730443
iter 20 value 0.119829
iter 30 value 0.106324
iter 40 value 0.105020
iter 50 value 0.104164
iter 60 value 0.101400
iter 70 value 0.099357
iter 80 value 0.096487
iter 90 value 0.094303
iter 100 value 0.092325
final value 0.092325
stopped after 100 iterations
# weights: 43
initial value 248.841811
iter 10 value 0.790980
iter 20 value 0.358904
iter 30 value 0.285149
iter 40 value 0.220373

```

```

iter 50 value 0.203989
iter 60 value 0.177128
iter 70 value 0.160189
iter 80 value 0.145662
iter 90 value 0.135482
iter 100 value 0.126278
final value 0.126278
stopped after 100 iterations
# weights: 11
initial value 225.790664
iter 10 value 86.330861
iter 20 value 79.509493
final value 79.499746
converged
# weights: 27
initial value 237.020880
iter 10 value 6.057106
iter 20 value 0.609869
iter 30 value 0.009805
iter 40 value 0.001168
iter 50 value 0.000118
iter 50 value 0.000072
iter 50 value 0.000072
final value 0.000072
converged
# weights: 43
initial value 321.845907
iter 10 value 5.566426
iter 20 value 1.341585
iter 30 value 0.060239
iter 40 value 0.002317
iter 50 value 0.000590
iter 60 value 0.000280
final value 0.000087
converged
# weights: 11
initial value 253.423853
iter 10 value 66.047555
iter 20 value 63.231753
iter 30 value 63.062217
final value 63.062057
converged
# weights: 27
initial value 244.293087
iter 10 value 24.624882
iter 20 value 18.499141

```

```

iter 30 value 17.734610
iter 40 value 17.407988
iter 50 value 17.367794
final value 17.367781
converged
# weights: 43
initial value 244.914355
iter 10 value 20.881382
iter 20 value 15.859302
iter 30 value 15.666224
iter 40 value 15.656694
iter 50 value 15.655718
iter 60 value 15.655530
final value 15.655490
converged
# weights: 11
initial value 210.764355
iter 10 value 37.487488
iter 20 value 21.997946
iter 30 value 21.569972
iter 40 value 21.232005
iter 50 value 21.072124
iter 60 value 21.062195
iter 70 value 21.050011
iter 80 value 21.049941
iter 90 value 21.049786
iter 90 value 21.049786
iter 90 value 21.049786
final value 21.049786
converged
# weights: 27
initial value 236.704159
iter 10 value 1.635889
iter 20 value 0.133036
iter 30 value 0.120516
iter 40 value 0.117574
iter 50 value 0.110651
iter 60 value 0.107234
iter 70 value 0.101690
iter 80 value 0.096832
iter 90 value 0.095937
iter 100 value 0.095539
final value 0.095539
stopped after 100 iterations
# weights: 43
initial value 344.981384

```

```

iter 10 value 13.901252
iter 20 value 0.542138
iter 30 value 0.417620
iter 40 value 0.374328
iter 50 value 0.248417
iter 60 value 0.213988
iter 70 value 0.180656
iter 80 value 0.148485
iter 90 value 0.133583
iter 100 value 0.123713
final value 0.123713
stopped after 100 iterations
# weights: 11
initial value 238.307817
iter 10 value 80.660917
iter 20 value 80.653828
final value 80.653818
converged
# weights: 27
initial value 219.180471
iter 10 value 3.256841
iter 20 value 0.050317
iter 30 value 0.000423
iter 40 value 0.000249
final value 0.000092
converged
# weights: 43
initial value 244.260065
iter 10 value 10.983788
iter 20 value 0.035555
iter 30 value 0.004498
iter 40 value 0.000722
iter 50 value 0.000317
final value 0.000090
converged
# weights: 11
initial value 235.740751
iter 10 value 81.396706
iter 20 value 63.402378
iter 30 value 63.238148
iter 30 value 63.238147
iter 30 value 63.238147
final value 63.238147
converged
# weights: 27
initial value 216.599821

```

```
iter 10 value 39.971622
iter 20 value 20.480365
iter 30 value 18.498283
iter 40 value 18.249745
iter 50 value 17.704249
iter 60 value 17.456246
iter 70 value 17.428433
final value 17.428425
converged
# weights: 43
initial value 257.378344
iter 10 value 25.789882
iter 20 value 16.236904
iter 30 value 15.814538
iter 40 value 15.685818
iter 50 value 15.665593
iter 60 value 15.663334
final value 15.662966
converged
# weights: 11
initial value 216.277523
iter 10 value 78.853771
iter 20 value 76.717949
iter 30 value 51.621895
iter 40 value 39.030153
iter 50 value 38.621213
iter 60 value 38.579030
iter 70 value 38.389057
iter 80 value 38.364320
iter 90 value 38.291763
iter 100 value 38.267362
final value 38.267362
stopped after 100 iterations
# weights: 27
initial value 286.444154
iter 10 value 3.752165
iter 20 value 0.167491
iter 30 value 0.159936
iter 40 value 0.138501
iter 50 value 0.131988
iter 60 value 0.117872
iter 70 value 0.113347
iter 80 value 0.108395
iter 90 value 0.099575
iter 100 value 0.097333
final value 0.097333
```

```
stopped after 100 iterations
# weights: 43
initial value 219.648708
iter 10 value 2.971292
iter 20 value 0.123324
iter 30 value 0.113667
iter 40 value 0.111432
iter 50 value 0.107751
iter 60 value 0.099030
iter 70 value 0.096719
iter 80 value 0.094029
iter 90 value 0.092044
iter 100 value 0.090891
final value 0.090891
stopped after 100 iterations
# weights: 11
initial value 212.598555
iter 10 value 50.457759
iter 20 value 28.442647
iter 30 value 22.789342
iter 40 value 22.383740
iter 50 value 22.246951
iter 60 value 22.082566
iter 70 value 21.972958
iter 80 value 21.940800
iter 90 value 21.935584
iter 100 value 21.894137
final value 21.894137
stopped after 100 iterations
# weights: 27
initial value 243.949294
iter 10 value 3.373144
iter 20 value 0.030224
iter 30 value 0.000770
final value 0.000050
converged
# weights: 43
initial value 289.449294
iter 10 value 1.230986
iter 20 value 0.047012
iter 30 value 0.002762
iter 40 value 0.000572
iter 50 value 0.000451
final value 0.000094
converged
# weights: 11
```

```
initial value 254.103380
iter 10 value 102.008974
iter 20 value 80.453714
iter 30 value 65.154635
final value 64.934035
converged
# weights: 27
initial value 239.755502
iter 10 value 42.072464
iter 20 value 20.786351
iter 30 value 19.550612
iter 40 value 19.379582
iter 50 value 19.361425
iter 60 value 19.358136
final value 19.358121
converged
# weights: 43
initial value 273.544910
iter 10 value 23.916232
iter 20 value 16.062707
iter 30 value 15.743153
iter 40 value 15.718312
iter 50 value 15.718006
iter 60 value 15.717918
iter 60 value 15.717918
iter 60 value 15.717918
final value 15.717918
converged
# weights: 11
initial value 211.607665
iter 10 value 79.860752
iter 20 value 48.150382
iter 30 value 25.135422
iter 40 value 22.917050
iter 50 value 22.718454
iter 60 value 22.504922
iter 70 value 22.496875
iter 80 value 22.491025
final value 22.489388
converged
# weights: 27
initial value 231.954214
iter 10 value 2.857993
iter 20 value 0.125964
iter 30 value 0.121407
iter 40 value 0.116393
```

```
iter 50 value 0.114726
iter 60 value 0.112915
iter 70 value 0.111597
iter 80 value 0.110639
iter 90 value 0.109175
iter 100 value 0.106921
final value 0.106921
stopped after 100 iterations
# weights: 43
initial value 221.289682
iter 10 value 13.553036
iter 20 value 0.976519
iter 30 value 0.881430
iter 40 value 0.694551
iter 50 value 0.371483
iter 60 value 0.278716
iter 70 value 0.185211
iter 80 value 0.158387
iter 90 value 0.143997
iter 100 value 0.136078
final value 0.136078
stopped after 100 iterations
# weights: 11
initial value 233.642742
iter 10 value 101.554839
iter 20 value 75.381398
iter 30 value 73.880055
iter 40 value 71.409217
iter 50 value 62.929749
iter 60 value 19.072975
iter 70 value 13.242826
iter 80 value 12.846889
iter 90 value 12.500601
iter 100 value 12.228295
final value 12.228295
stopped after 100 iterations
# weights: 27
initial value 230.670832
iter 10 value 4.419343
iter 20 value 0.075918
iter 30 value 0.000168
iter 30 value 0.000079
iter 30 value 0.000079
final value 0.000079
converged
# weights: 43
```

```
initial value 313.749728
iter 10 value 2.495541
iter 20 value 0.034300
iter 30 value 0.000154
final value 0.000099
converged
# weights: 11
initial value 240.058012
iter 10 value 98.086659
iter 20 value 62.531636
iter 30 value 60.297572
iter 40 value 60.240623
iter 40 value 60.240623
iter 40 value 60.240623
final value 60.240623
converged
# weights: 27
initial value 255.279913
iter 10 value 35.686965
iter 20 value 19.337593
iter 30 value 18.075402
iter 40 value 17.307466
iter 50 value 17.175701
iter 60 value 17.169133
final value 17.169096
converged
# weights: 43
initial value 195.227751
iter 10 value 19.681415
iter 20 value 16.802022
iter 30 value 15.949016
iter 40 value 15.682137
iter 50 value 15.585820
iter 60 value 15.558464
final value 15.558381
converged
# weights: 11
initial value 232.111991
iter 10 value 81.753594
iter 20 value 75.031855
iter 30 value 35.784568
iter 40 value 13.541955
iter 50 value 13.036747
iter 60 value 12.946630
iter 70 value 12.926118
iter 80 value 12.924268
```

```
final value 12.920689
converged
# weights: 27
initial value 321.334893
iter 10 value 5.299237
iter 20 value 0.138157
iter 30 value 0.127628
iter 40 value 0.122408
iter 50 value 0.118207
iter 60 value 0.112416
iter 70 value 0.109578
iter 80 value 0.106422
iter 90 value 0.103977
iter 100 value 0.101567
final value 0.101567
stopped after 100 iterations
# weights: 43
initial value 285.076515
iter 10 value 3.051245
iter 20 value 0.138871
iter 30 value 0.108914
iter 40 value 0.101540
iter 50 value 0.099404
iter 60 value 0.097318
iter 70 value 0.094894
iter 80 value 0.093638
iter 90 value 0.092468
iter 100 value 0.090277
final value 0.090277
stopped after 100 iterations
# weights: 11
initial value 217.022708
iter 10 value 75.280905
iter 20 value 36.311653
iter 30 value 30.044173
iter 40 value 28.532267
iter 50 value 28.426113
iter 60 value 28.402994
iter 70 value 28.389873
final value 28.389861
converged
# weights: 27
initial value 267.030413
iter 10 value 0.204880
iter 20 value 0.020150
iter 30 value 0.002419
```

```
iter 40 value 0.000108
final value 0.000100
converged
# weights: 43
initial value 228.112544
iter 10 value 2.497467
iter 20 value 0.028631
iter 30 value 0.003571
iter 40 value 0.001592
iter 50 value 0.000422
iter 60 value 0.000188
iter 70 value 0.000113
iter 70 value 0.000097
iter 70 value 0.000097
final value 0.000097
converged
# weights: 11
initial value 197.199439
iter 10 value 89.391388
iter 20 value 64.106270
iter 30 value 63.745774
final value 63.745362
converged
# weights: 27
initial value 235.366887
iter 10 value 33.560769
iter 20 value 22.855962
iter 30 value 19.804175
iter 40 value 19.601957
iter 50 value 19.587425
iter 60 value 19.583173
final value 19.583091
converged
# weights: 43
initial value 189.708165
iter 10 value 23.007214
iter 20 value 17.477685
iter 30 value 16.555287
iter 40 value 16.355771
iter 50 value 16.295569
iter 60 value 16.291543
final value 16.291534
converged
# weights: 11
initial value 250.557257
iter 10 value 104.959020
```

```
iter 20 value 79.458391
iter 30 value 55.405774
iter 40 value 22.301806
iter 50 value 18.597570
iter 60 value 17.897018
iter 70 value 17.863924
iter 80 value 17.847797
iter 90 value 17.816161
iter 100 value 17.793659
final value 17.793659
stopped after 100 iterations
# weights: 27
initial value 264.214116
iter 10 value 2.884944
iter 20 value 0.131865
iter 30 value 0.126046
iter 40 value 0.122286
iter 50 value 0.119998
iter 60 value 0.118360
iter 70 value 0.115372
iter 80 value 0.111701
iter 90 value 0.106703
iter 100 value 0.103111
final value 0.103111
stopped after 100 iterations
# weights: 43
initial value 214.693413
iter 10 value 7.369694
iter 20 value 0.233592
iter 30 value 0.159848
iter 40 value 0.153988
iter 50 value 0.150047
iter 60 value 0.135864
iter 70 value 0.126187
iter 80 value 0.121931
iter 90 value 0.112941
iter 100 value 0.107976
final value 0.107976
stopped after 100 iterations
# weights: 11
initial value 230.354315
iter 10 value 89.582035
iter 20 value 80.527494
iter 30 value 80.277523
iter 40 value 80.274991
final value 80.270129
```

```

converged
# weights: 27
initial value 275.533891
iter 10 value 0.350055
iter 20 value 0.023599
iter 30 value 0.001462
iter 40 value 0.000352
final value 0.000086
converged
# weights: 43
initial value 366.859449
iter 10 value 1.920315
iter 20 value 0.013574
iter 30 value 0.000575
iter 40 value 0.000306
final value 0.000099
converged
# weights: 11
initial value 233.472055
iter 10 value 93.415545
iter 20 value 82.601945
iter 30 value 82.266690
final value 82.265952
converged
# weights: 27
initial value 246.400709
iter 10 value 34.233669
iter 20 value 18.822147
iter 30 value 18.690854
iter 40 value 18.688507
final value 18.688440
converged
# weights: 43
initial value 305.892984
iter 10 value 41.520487
iter 20 value 16.184068
iter 30 value 15.111347
iter 40 value 14.822630
iter 50 value 14.810621
iter 60 value 14.807521
final value 14.807488
converged
# weights: 11
initial value 274.697333
iter 10 value 85.960348
iter 20 value 72.449074

```

```

iter 30 value 67.974098
iter 40 value 33.026073
iter 50 value 18.759662
iter 60 value 18.146788
iter 70 value 17.763271
iter 80 value 17.711171
iter 90 value 17.703914
final value 17.698888
converged
# weights: 27
initial value 195.754985
iter 10 value 3.165564
iter 20 value 0.259508
iter 30 value 0.247879
iter 40 value 0.225529
iter 50 value 0.148281
iter 60 value 0.133800
iter 70 value 0.120536
iter 80 value 0.109297
iter 90 value 0.097403
iter 100 value 0.090460
final value 0.090460
stopped after 100 iterations
# weights: 43
initial value 218.522216
iter 10 value 0.544181
iter 20 value 0.189607
iter 30 value 0.175584
iter 40 value 0.142962
iter 50 value 0.125074
iter 60 value 0.111626
iter 70 value 0.100137
iter 80 value 0.084749
iter 90 value 0.077392
iter 100 value 0.074053
final value 0.074053
stopped after 100 iterations
# weights: 11
initial value 217.076193
iter 10 value 93.202447
iter 20 value 84.285214
iter 30 value 82.580932
iter 40 value 80.665180
iter 50 value 80.492632
iter 60 value 80.469181
iter 70 value 80.439376

```

```
iter 80 value 80.437838
iter 90 value 80.436919
iter 100 value 80.435418
final value 80.435418
stopped after 100 iterations
# weights: 27
initial value 236.635174
iter 10 value 2.807360
iter 20 value 0.094365
iter 30 value 0.007140
iter 40 value 0.001663
iter 50 value 0.000763
iter 60 value 0.000611
iter 70 value 0.000433
iter 80 value 0.000397
final value 0.000088
converged
# weights: 43
initial value 220.329549
iter 10 value 0.382652
iter 20 value 0.017831
iter 30 value 0.000957
iter 40 value 0.000444
iter 50 value 0.000372
final value 0.000100
converged
# weights: 11
initial value 212.580585
iter 10 value 92.356784
iter 20 value 90.238454
iter 30 value 83.926317
iter 40 value 83.093436
final value 83.093427
converged
# weights: 27
initial value 240.059494
iter 10 value 24.315113
iter 20 value 18.678404
iter 30 value 18.529443
iter 40 value 18.438687
iter 50 value 17.442329
iter 60 value 17.425322
iter 60 value 17.425321
iter 60 value 17.425321
final value 17.425321
converged
```

```

# weights: 43
initial value 249.636062
iter 10 value 35.817527
iter 20 value 17.718706
iter 30 value 16.994844
iter 40 value 16.741675
iter 50 value 16.115844
iter 60 value 16.012378
iter 70 value 15.990409
iter 80 value 15.728951
iter 90 value 15.627833
iter 100 value 15.623606
final value 15.623606
stopped after 100 iterations
# weights: 11
initial value 268.621451
iter 10 value 85.427778
iter 20 value 78.702666
iter 30 value 78.224350
iter 40 value 71.930509
iter 50 value 42.703517
iter 60 value 37.219340
iter 70 value 37.176574
iter 80 value 37.144563
final value 37.136608
converged
# weights: 27
initial value 260.218870
iter 10 value 8.534232
iter 20 value 3.211974
iter 30 value 0.171498
iter 40 value 0.145085
iter 50 value 0.134032
iter 60 value 0.121747
iter 70 value 0.114894
iter 80 value 0.114032
iter 90 value 0.112410
iter 100 value 0.111052
final value 0.111052
stopped after 100 iterations
# weights: 43
initial value 209.809911
iter 10 value 2.440205
iter 20 value 0.111746
iter 30 value 0.104162
iter 40 value 0.099805

```

```

iter 50 value 0.098060
iter 60 value 0.095288
iter 70 value 0.093298
iter 80 value 0.091315
iter 90 value 0.090127
iter 100 value 0.088948
final value 0.088948
stopped after 100 iterations
# weights: 11
initial value 251.450234
iter 10 value 81.949506
iter 20 value 74.662387
iter 30 value 74.299720
iter 40 value 73.712863
iter 50 value 73.292338
iter 60 value 70.425855
iter 70 value 68.127643
iter 80 value 47.941518
iter 90 value 25.795995
iter 100 value 18.751198
final value 18.751198
stopped after 100 iterations
# weights: 27
initial value 254.544192
iter 10 value 29.071695
iter 20 value 2.370565
iter 30 value 0.048259
iter 40 value 0.006845
iter 50 value 0.002848
iter 60 value 0.001779
iter 70 value 0.000104
iter 70 value 0.000066
iter 70 value 0.000066
final value 0.000066
converged
# weights: 43
initial value 283.793556
iter 10 value 4.731283
iter 20 value 0.042240
iter 30 value 0.001359
final value 0.000088
converged
# weights: 11
initial value 267.786163
iter 10 value 101.679698
iter 20 value 65.430814

```

```
iter 30 value 62.871162
final value 62.826799
converged
# weights: 27
initial value 232.542659
iter 10 value 39.374333
iter 20 value 20.259861
iter 30 value 17.955908
iter 40 value 16.793027
iter 50 value 16.593389
iter 60 value 16.544338
iter 70 value 16.541097
iter 70 value 16.541097
iter 70 value 16.541097
final value 16.541097
converged
# weights: 43
initial value 229.876059
iter 10 value 22.523931
iter 20 value 16.052557
iter 30 value 15.413589
iter 40 value 14.902557
iter 50 value 14.836538
iter 60 value 14.828408
iter 70 value 14.828246
iter 80 value 14.828215
iter 80 value 14.828215
iter 80 value 14.828215
final value 14.828215
converged
# weights: 11
initial value 234.111915
iter 10 value 68.101184
iter 20 value 19.391866
iter 30 value 18.455430
iter 40 value 18.222796
iter 50 value 18.043446
iter 60 value 18.034149
iter 70 value 18.028159
final value 18.028159
converged
# weights: 27
initial value 287.281702
iter 10 value 9.336956
iter 20 value 0.262431
iter 30 value 0.213876
```

```

iter 40 value 0.197844
iter 50 value 0.184523
iter 60 value 0.167908
iter 70 value 0.155871
iter 80 value 0.148361
iter 90 value 0.130800
iter 100 value 0.108823
final value 0.108823
stopped after 100 iterations
# weights: 43
initial value 218.892128
iter 10 value 1.266555
iter 20 value 0.111641
iter 30 value 0.102834
iter 40 value 0.097455
iter 50 value 0.090997
iter 60 value 0.082321
iter 70 value 0.080516
iter 80 value 0.077755
iter 90 value 0.075829
iter 100 value 0.074886
final value 0.074886
stopped after 100 iterations
# weights: 11
initial value 234.907013
iter 10 value 117.087899
iter 20 value 34.491600
iter 30 value 32.484037
iter 40 value 32.277317
iter 50 value 31.705291
iter 60 value 31.608696
iter 70 value 31.561351
iter 80 value 31.533526
iter 90 value 31.434956
iter 100 value 30.910765
final value 30.910765
stopped after 100 iterations
# weights: 27
initial value 238.523723
iter 10 value 5.338512
iter 20 value 0.010711
iter 30 value 0.001411
iter 40 value 0.000176
final value 0.000098
converged
# weights: 43

```

```

initial value 225.808559
iter 10 value 5.866428
iter 20 value 0.028766
iter 30 value 0.000493
final value 0.000087
converged
# weights: 11
initial value 269.328177
iter 10 value 104.793770
iter 20 value 86.707956
iter 30 value 85.159674
final value 85.061587
converged
# weights: 27
initial value 240.657802
iter 10 value 44.445690
iter 20 value 19.781593
iter 30 value 19.272660
iter 40 value 19.260736
iter 50 value 19.203745
iter 60 value 19.113626
iter 70 value 19.112578
final value 19.112577
converged
# weights: 43
initial value 299.203614
iter 10 value 22.723775
iter 20 value 16.899403
iter 30 value 16.094061
iter 40 value 16.025334
iter 50 value 16.018796
iter 60 value 16.016290
iter 70 value 16.012435
final value 16.012369
converged
# weights: 11
initial value 215.725160
iter 10 value 48.504189
iter 20 value 14.009398
iter 30 value 9.826478
iter 40 value 9.749692
iter 50 value 9.604873
iter 60 value 9.602784
iter 70 value 9.602565
final value 9.602564
converged

```

```

# weights: 27
initial value 234.356677
iter 10 value 9.740916
iter 20 value 0.212125
iter 30 value 0.185176
iter 40 value 0.178268
iter 50 value 0.157716
iter 60 value 0.136700
iter 70 value 0.132446
iter 80 value 0.124043
iter 90 value 0.116835
iter 100 value 0.113952
final value 0.113952
stopped after 100 iterations
# weights: 43
initial value 226.227653
iter 10 value 0.473747
iter 20 value 0.165052
iter 30 value 0.151131
iter 40 value 0.141639
iter 50 value 0.120316
iter 60 value 0.108452
iter 70 value 0.104383
iter 80 value 0.102687
iter 90 value 0.099891
iter 100 value 0.097516
final value 0.097516
stopped after 100 iterations
# weights: 11
initial value 226.834661
iter 10 value 81.581185
iter 20 value 80.230449
iter 30 value 79.764221
iter 40 value 79.119405
iter 50 value 77.922559
iter 60 value 60.115672
iter 70 value 40.608853
iter 80 value 38.857972
iter 90 value 38.131089
iter 100 value 38.030804
final value 38.030804
stopped after 100 iterations
# weights: 27
initial value 295.638036
iter 10 value 4.705799
iter 20 value 0.063563

```

```

iter 30 value 0.000909
final value 0.000064
converged
# weights: 43
initial value 243.374966
iter 10 value 33.577329
iter 20 value 12.853369
iter 30 value 9.612988
iter 40 value 8.506941
iter 50 value 3.485669
iter 60 value 3.306939
iter 70 value 3.044639
iter 80 value 2.969264
iter 90 value 2.739380
iter 100 value 2.733151
final value 2.733151
stopped after 100 iterations
# weights: 11
initial value 214.899430
iter 10 value 93.743315
iter 20 value 68.195779
iter 30 value 62.917244
final value 62.883847
converged
# weights: 27
initial value 233.652262
iter 10 value 25.542415
iter 20 value 18.785723
iter 30 value 18.298997
iter 40 value 18.258217
iter 50 value 18.252856
final value 18.252848
converged
# weights: 43
initial value 205.505108
iter 10 value 21.389405
iter 20 value 15.664804
iter 30 value 15.066058
iter 40 value 15.037785
iter 50 value 15.035632
iter 60 value 15.032309
final value 15.031669
converged
# weights: 11
initial value 212.154627
iter 10 value 82.544745

```

```

iter 20 value 79.880059
iter 30 value 79.860463
iter 40 value 79.774742
iter 50 value 74.593747
iter 60 value 72.884640
iter 70 value 68.015935
iter 80 value 26.446465
iter 90 value 20.585055
iter 100 value 18.850897
final value 18.850897
stopped after 100 iterations
# weights: 27
initial value 224.814593
iter 10 value 1.148472
iter 20 value 0.160896
iter 30 value 0.129222
iter 40 value 0.125892
iter 50 value 0.117577
iter 60 value 0.110051
iter 70 value 0.104860
iter 80 value 0.099310
iter 90 value 0.094348
iter 100 value 0.085335
final value 0.085335
stopped after 100 iterations
# weights: 43
initial value 258.413049
iter 10 value 1.561042
iter 20 value 0.102337
iter 30 value 0.089828
iter 40 value 0.086869
iter 50 value 0.083982
iter 60 value 0.080461
iter 70 value 0.078604
iter 80 value 0.077811
iter 90 value 0.076769
iter 100 value 0.075637
final value 0.075637
stopped after 100 iterations
# weights: 11
initial value 228.539416
iter 10 value 79.687395
iter 20 value 78.395019
iter 30 value 75.031061
iter 40 value 36.931651
iter 50 value 21.578159

```

```
iter 60 value 20.739305
iter 70 value 20.372786
iter 80 value 20.140479
iter 90 value 20.098653
iter 100 value 19.975156
final value 19.975156
stopped after 100 iterations
# weights: 27
initial value 252.695686
iter 10 value 31.238596
iter 20 value 0.257038
iter 30 value 0.030178
iter 40 value 0.007974
iter 50 value 0.000459
final value 0.000098
converged
# weights: 43
initial value 237.566789
iter 10 value 2.869913
iter 20 value 0.017979
iter 30 value 0.005574
iter 40 value 0.002792
iter 50 value 0.000582
iter 60 value 0.000377
iter 70 value 0.000241
iter 80 value 0.000123
final value 0.000074
converged
# weights: 11
initial value 227.400106
iter 10 value 100.435622
iter 20 value 65.299371
iter 30 value 62.753923
final value 62.753918
converged
# weights: 27
initial value 226.787516
iter 10 value 24.645546
iter 20 value 18.389826
iter 30 value 18.114631
iter 40 value 18.005322
iter 50 value 18.003326
iter 50 value 18.003326
iter 50 value 18.003326
final value 18.003326
converged
```

```

# weights: 43
initial value 266.595353
iter 10 value 22.764184
iter 20 value 16.801024
iter 30 value 16.560687
iter 40 value 16.554294
iter 50 value 16.554008
iter 60 value 16.553883
final value 16.553881
converged
# weights: 11
initial value 208.014583
iter 10 value 92.923308
iter 20 value 79.112281
iter 30 value 78.064278
iter 40 value 71.840089
iter 50 value 42.067090
iter 60 value 35.306909
iter 70 value 35.041985
iter 80 value 34.987562
iter 90 value 34.983120
iter 90 value 34.983120
iter 90 value 34.983120
final value 34.983120
converged
# weights: 27
initial value 245.918741
iter 10 value 9.880944
iter 20 value 0.363453
iter 30 value 0.311523
iter 40 value 0.297202
iter 50 value 0.185559
iter 60 value 0.157016
iter 70 value 0.146610
iter 80 value 0.134859
iter 90 value 0.121367
iter 100 value 0.115674
final value 0.115674
stopped after 100 iterations
# weights: 43
initial value 203.602145
iter 10 value 2.088364
iter 20 value 0.137222
iter 30 value 0.130111
iter 40 value 0.127108
iter 50 value 0.120118

```

```

iter 60 value 0.110895
iter 70 value 0.105942
iter 80 value 0.100962
iter 90 value 0.098644
iter 100 value 0.097937
final value 0.097937
stopped after 100 iterations
# weights: 11
initial value 290.747947
iter 10 value 131.628577
iter 20 value 111.975947
iter 30 value 111.238191
iter 40 value 110.874869
iter 50 value 110.383841
iter 60 value 110.254059
iter 70 value 110.087406
iter 80 value 110.075198
iter 90 value 109.551068
iter 100 value 109.236371
final value 109.236371
stopped after 100 iterations
# weights: 27
initial value 204.343794
iter 10 value 0.221560
iter 20 value 0.000979
final value 0.000062
converged
# weights: 43
initial value 207.248588
iter 10 value 12.759044
iter 20 value 0.000139
iter 20 value 0.000074
iter 20 value 0.000072
final value 0.000072
converged
# weights: 11
initial value 256.502620
iter 10 value 121.291139
iter 20 value 65.405653
iter 30 value 62.049938
iter 40 value 61.920678
final value 61.920671
converged
# weights: 27
initial value 247.169804
iter 10 value 20.924045

```

```

iter 20 value 17.973884
iter 30 value 17.662871
iter 40 value 17.644810
iter 50 value 17.643356
final value 17.643345
converged
# weights: 43
initial value 241.879166
iter 10 value 21.529388
iter 20 value 14.142812
iter 30 value 14.094124
iter 40 value 14.087685
iter 50 value 14.086994
final value 14.086992
converged
# weights: 11
initial value 242.834291
iter 10 value 76.871664
iter 20 value 74.354109
iter 30 value 73.889283
iter 40 value 73.559476
iter 50 value 73.520523
iter 60 value 73.478590
iter 70 value 71.638168
iter 80 value 69.781665
iter 90 value 32.997261
iter 100 value 16.925833
final value 16.925833
stopped after 100 iterations
# weights: 27
initial value 234.482108
iter 10 value 5.518383
iter 20 value 0.489284
iter 30 value 0.185689
iter 40 value 0.149399
iter 50 value 0.133343
iter 60 value 0.122257
iter 70 value 0.112173
iter 80 value 0.095518
iter 90 value 0.080001
iter 100 value 0.071764
final value 0.071764
stopped after 100 iterations
# weights: 43
initial value 269.135696
iter 10 value 0.456704

```

```

iter 20 value 0.101067
iter 30 value 0.092280
iter 40 value 0.080359
iter 50 value 0.074850
iter 60 value 0.071660
iter 70 value 0.068405
iter 80 value 0.064871
iter 90 value 0.063157
iter 100 value 0.062177
final value 0.062177
stopped after 100 iterations
# weights: 11
initial value 256.887655
iter 10 value 80.522491
iter 20 value 78.457750
iter 30 value 77.522144
iter 40 value 76.425236
iter 50 value 76.120501
iter 60 value 75.229761
iter 70 value 74.176549
iter 80 value 50.435207
iter 90 value 33.476938
iter 100 value 32.225196
final value 32.225196
stopped after 100 iterations
# weights: 27
initial value 277.299333
iter 10 value 33.139249
iter 20 value 24.923268
iter 30 value 24.778915
iter 40 value 24.675153
iter 50 value 24.365609
iter 60 value 21.162697
iter 70 value 4.756198
iter 80 value 3.562622
iter 90 value 3.418804
iter 100 value 3.048823
final value 3.048823
stopped after 100 iterations
# weights: 43
initial value 246.814157
iter 10 value 1.743126
iter 20 value 0.016286
iter 30 value 0.002554
iter 40 value 0.001227
final value 0.000073

```

```

converged
# weights: 11
initial value 298.438846
iter 10 value 98.577734
iter 20 value 71.134881
iter 30 value 65.096314
final value 65.095949
converged
# weights: 27
initial value 243.446879
iter 10 value 31.299700
iter 20 value 19.029798
iter 30 value 17.995871
iter 40 value 17.983869
iter 50 value 17.982413
final value 17.982402
converged
# weights: 43
initial value 364.399896
iter 10 value 44.142426
iter 20 value 17.040983
iter 30 value 16.464237
iter 40 value 16.386565
iter 50 value 16.383850
iter 60 value 16.383587
final value 16.383570
converged
# weights: 11
initial value 242.181888
iter 10 value 108.106462
iter 20 value 69.576439
iter 30 value 46.656278
iter 40 value 33.304496
iter 50 value 32.278659
iter 60 value 32.171393
iter 70 value 31.975159
iter 80 value 31.974259
iter 90 value 31.969929
iter 100 value 31.956530
final value 31.956530
stopped after 100 iterations
# weights: 27
initial value 227.489474
iter 10 value 6.134720
iter 20 value 0.280102
iter 30 value 0.167864

```

```
iter 40 value 0.157842
iter 50 value 0.151018
iter 60 value 0.146966
iter 70 value 0.144046
iter 80 value 0.140334
iter 90 value 0.132218
iter 100 value 0.125901
final value 0.125901
stopped after 100 iterations
# weights: 43
initial value 287.082053
iter 10 value 3.779520
iter 20 value 0.159085
iter 30 value 0.129552
iter 40 value 0.123244
iter 50 value 0.116354
iter 60 value 0.109060
iter 70 value 0.106856
iter 80 value 0.105860
iter 90 value 0.103949
iter 100 value 0.100433
final value 0.100433
stopped after 100 iterations
# weights: 11
initial value 254.988671
iter 10 value 82.519051
iter 20 value 73.692975
iter 30 value 54.671142
iter 40 value 39.447451
iter 50 value 38.638443
iter 60 value 38.609194
iter 70 value 38.568623
iter 80 value 38.567428
iter 90 value 38.563684
iter 100 value 38.562199
final value 38.562199
stopped after 100 iterations
# weights: 27
initial value 217.482157
iter 10 value 4.814127
iter 20 value 0.333911
iter 30 value 0.024147
iter 40 value 0.003086
iter 50 value 0.000883
iter 60 value 0.000413
final value 0.000097
```

```
converged
# weights: 43
initial value 253.458347
iter 10 value 5.891446
iter 20 value 0.026091
final value 0.000083
converged
# weights: 11
initial value 246.566326
iter 10 value 79.969674
iter 20 value 61.277444
final value 60.897528
converged
# weights: 27
initial value 243.908823
iter 10 value 34.540179
iter 20 value 21.207354
iter 30 value 20.661670
iter 40 value 20.650643
iter 50 value 20.648363
iter 60 value 20.648319
final value 20.648317
converged
# weights: 43
initial value 288.894400
iter 10 value 59.845125
iter 20 value 18.717698
iter 30 value 16.789898
iter 40 value 15.998274
iter 50 value 15.958555
iter 60 value 15.952541
iter 70 value 15.952393
final value 15.952366
converged
# weights: 11
initial value 218.080957
iter 10 value 80.885838
iter 20 value 78.731963
iter 30 value 59.129876
iter 40 value 39.490779
iter 50 value 39.133710
iter 60 value 38.908931
iter 70 value 38.861363
iter 80 value 38.860061
iter 90 value 38.854963
iter 100 value 38.854415
```

```
final  value 38.854415
stopped after 100 iterations
# weights: 27
initial  value 217.351854
iter  10 value 3.335727
iter  20 value 0.203951
iter  30 value 0.184274
iter  40 value 0.179944
iter  50 value 0.170338
iter  60 value 0.157175
iter  70 value 0.151140
iter  80 value 0.145058
iter  90 value 0.138442
iter 100 value 0.124895
final  value 0.124895
stopped after 100 iterations
# weights: 43
initial  value 262.973482
iter  10 value 4.596587
iter  20 value 0.227719
iter  30 value 0.160156
iter  40 value 0.157195
iter  50 value 0.150627
iter  60 value 0.143448
iter  70 value 0.138525
iter  80 value 0.133230
iter  90 value 0.126028
iter 100 value 0.120906
final  value 0.120906
stopped after 100 iterations
# weights: 11
initial  value 228.554742
iter  10 value 79.648918
iter  20 value 50.078215
iter  30 value 20.714345
iter  40 value 14.678269
iter  50 value 14.554686
iter  60 value 14.305023
iter  70 value 14.301646
iter  80 value 14.194781
iter  90 value 14.188263
iter 100 value 14.179620
final  value 14.179620
stopped after 100 iterations
# weights: 27
initial  value 256.091263
```

```
iter 10 value 3.265804
iter 20 value 0.064966
iter 30 value 0.001484
iter 40 value 0.000732
iter 50 value 0.000479
iter 60 value 0.000144
final value 0.000094
converged
# weights: 43
initial value 201.353963
iter 10 value 3.475269
iter 20 value 0.052833
iter 30 value 0.001638
iter 40 value 0.000295
final value 0.000082
converged
# weights: 11
initial value 251.001705
iter 10 value 101.189752
iter 20 value 68.323303
iter 30 value 61.957080
final value 61.952615
converged
# weights: 27
initial value 240.763099
iter 10 value 27.458603
iter 20 value 19.546841
iter 30 value 19.537235
final value 19.536667
converged
# weights: 43
initial value 248.314792
iter 10 value 18.533470
iter 20 value 16.315189
iter 30 value 15.973260
iter 40 value 15.924332
iter 50 value 15.922856
final value 15.922616
converged
# weights: 11
initial value 300.527533
iter 10 value 91.064496
iter 20 value 79.433825
iter 30 value 78.771545
iter 40 value 78.459515
iter 50 value 78.218071
```

```
iter 60 value 77.509181
iter 70 value 60.536231
iter 80 value 34.603064
iter 90 value 33.144658
iter 100 value 32.731621
final value 32.731621
stopped after 100 iterations
# weights: 27
initial value 315.366150
iter 10 value 5.618135
iter 20 value 0.106011
iter 30 value 0.102754
iter 40 value 0.099264
iter 50 value 0.095788
iter 60 value 0.093690
iter 70 value 0.092464
iter 80 value 0.090158
iter 90 value 0.088703
iter 100 value 0.088275
final value 0.088275
stopped after 100 iterations
# weights: 43
initial value 232.653595
iter 10 value 1.132916
iter 20 value 0.285842
iter 30 value 0.246101
iter 40 value 0.182576
iter 50 value 0.132107
iter 60 value 0.114336
iter 70 value 0.102265
iter 80 value 0.088248
iter 90 value 0.086542
iter 100 value 0.084761
final value 0.084761
stopped after 100 iterations
# weights: 11
initial value 218.760182
iter 10 value 75.026347
iter 20 value 25.734987
iter 30 value 23.736486
iter 40 value 22.745957
iter 50 value 22.623180
iter 60 value 22.560325
iter 70 value 22.469241
iter 80 value 22.456498
iter 90 value 22.436826
```

```
iter 100 value 22.405442
final  value 22.405442
stopped after 100 iterations
# weights: 27
initial  value 233.213607
iter   10 value 17.351281
iter   20 value 0.059467
iter   30 value 0.007511
iter   40 value 0.002776
iter   50 value 0.000212
final  value 0.000087
converged
# weights: 43
initial  value 229.007749
iter   10 value 0.901635
iter   20 value 0.011756
iter   30 value 0.000654
iter   40 value 0.000106
final  value 0.000097
converged
# weights: 11
initial  value 197.630206
iter   10 value 93.432893
iter   20 value 89.418649
iter   30 value 84.721085
final  value 84.336781
converged
# weights: 27
initial  value 231.516978
iter   10 value 35.873633
iter   20 value 20.983937
iter   30 value 20.340199
iter   40 value 20.307125
iter   50 value 20.305496
iter   60 value 20.304181
final  value 20.304175
converged
# weights: 43
initial  value 267.963505
iter   10 value 18.339657
iter   20 value 15.801190
iter   30 value 15.578291
iter   40 value 15.570909
iter   50 value 15.570359
final  value 15.570344
converged
```

```

# weights: 11
initial value 262.554143
iter 10 value 80.799655
iter 20 value 80.763973
iter 30 value 79.507459
iter 40 value 77.697399
iter 50 value 72.720580
iter 60 value 49.971841
iter 70 value 37.470765
iter 80 value 36.534509
iter 90 value 36.391200
iter 100 value 36.361974
final value 36.361974
stopped after 100 iterations
# weights: 27
initial value 257.861884
iter 10 value 39.134532
iter 20 value 1.370419
iter 30 value 0.295340
iter 40 value 0.235646
iter 50 value 0.202124
iter 60 value 0.187813
iter 70 value 0.172726
iter 80 value 0.159891
iter 90 value 0.151476
iter 100 value 0.141446
final value 0.141446
stopped after 100 iterations
# weights: 43
initial value 225.545392
iter 10 value 3.525180
iter 20 value 0.154673
iter 30 value 0.128901
iter 40 value 0.124755
iter 50 value 0.121276
iter 60 value 0.112800
iter 70 value 0.107525
iter 80 value 0.104887
iter 90 value 0.102427
iter 100 value 0.100567
final value 0.100567
stopped after 100 iterations
# weights: 11
initial value 278.305566
iter 10 value 100.878705
iter 20 value 76.262051

```

```

iter 30 value 74.014927
iter 40 value 73.833774
iter 50 value 73.678423
iter 60 value 73.633422
iter 70 value 73.618186
iter 80 value 73.610167
iter 90 value 73.596156
iter 100 value 73.595257
final value 73.595257
stopped after 100 iterations
# weights: 27
initial value 213.033564
iter 10 value 9.662170
iter 20 value 0.854985
iter 30 value 0.019790
final value 0.000052
converged
# weights: 43
initial value 218.929848
iter 10 value 1.423156
iter 20 value 0.025099
iter 30 value 0.000187
final value 0.000084
converged
# weights: 11
initial value 252.933346
iter 10 value 109.395606
iter 20 value 88.847944
iter 30 value 65.651240
iter 40 value 62.073857
final value 62.073783
converged
# weights: 27
initial value 207.624164
iter 10 value 25.573062
iter 20 value 17.166816
iter 30 value 16.977482
iter 40 value 16.948846
iter 50 value 16.947603
final value 16.947601
converged
# weights: 43
initial value 220.522051
iter 10 value 59.134066
iter 20 value 15.786335
iter 30 value 15.389281

```

```
iter 40 value 15.322392
iter 50 value 15.256017
iter 60 value 15.240178
iter 70 value 15.239178
final value 15.239064
converged
# weights: 11
initial value 222.924700
iter 10 value 64.142421
iter 20 value 26.260683
iter 30 value 18.897324
iter 40 value 17.934674
iter 50 value 17.544691
iter 60 value 17.535422
iter 70 value 17.499356
iter 80 value 17.499171
iter 90 value 17.498437
final value 17.498434
converged
# weights: 27
initial value 227.946548
iter 10 value 2.483753
iter 20 value 0.112996
iter 30 value 0.107638
iter 40 value 0.105956
iter 50 value 0.101560
iter 60 value 0.098962
iter 70 value 0.096301
iter 80 value 0.094918
iter 90 value 0.091508
iter 100 value 0.089117
final value 0.089117
stopped after 100 iterations
# weights: 43
initial value 224.149368
iter 10 value 1.000181
iter 20 value 0.129998
iter 30 value 0.119765
iter 40 value 0.108024
iter 50 value 0.101191
iter 60 value 0.094497
iter 70 value 0.090324
iter 80 value 0.085127
iter 90 value 0.081975
iter 100 value 0.080913
final value 0.080913
```

```

stopped after 100 iterations
# weights: 11
initial value 202.690337
iter 10 value 41.950247
iter 20 value 20.031084
iter 30 value 19.068036
iter 40 value 18.608913
iter 50 value 18.450866
iter 60 value 18.363794
iter 70 value 18.312038
iter 80 value 18.293679
iter 90 value 18.255178
iter 100 value 18.230324
final value 18.230324
stopped after 100 iterations
# weights: 27
initial value 242.432137
iter 10 value 7.416972
iter 20 value 0.079851
iter 30 value 0.007866
final value 0.000051
converged
# weights: 43
initial value 283.337335
iter 10 value 2.099310
iter 20 value 0.042400
iter 30 value 0.000638
iter 40 value 0.000124
final value 0.000075
converged
# weights: 11
initial value 285.280475
iter 10 value 96.251331
iter 20 value 85.609712
iter 30 value 85.008458
final value 85.004508
converged
# weights: 27
initial value 236.283243
iter 10 value 47.221659
iter 20 value 18.234593
iter 30 value 17.126635
iter 40 value 16.924580
iter 50 value 16.918563
iter 60 value 16.918535
final value 16.918533

```

```

converged
# weights: 43
initial value 209.976233
iter 10 value 38.223558
iter 20 value 15.839650
iter 30 value 15.350301
iter 40 value 15.311546
iter 50 value 15.309805
iter 60 value 15.309787
final value 15.309785
converged
# weights: 11
initial value 272.599249
iter 10 value 80.691479
iter 20 value 58.369385
iter 30 value 38.752652
iter 40 value 34.839471
iter 50 value 33.465011
iter 60 value 33.450410
iter 70 value 33.393384
iter 80 value 33.392632
iter 90 value 33.386820
final value 33.386275
converged
# weights: 27
initial value 202.497197
iter 10 value 1.430160
iter 20 value 0.229077
iter 30 value 0.154011
iter 40 value 0.142978
iter 50 value 0.129951
iter 60 value 0.119867
iter 70 value 0.114814
iter 80 value 0.104755
iter 90 value 0.099006
iter 100 value 0.093685
final value 0.093685
stopped after 100 iterations
# weights: 43
initial value 270.744339
iter 10 value 2.175436
iter 20 value 0.131320
iter 30 value 0.119423
iter 40 value 0.117414
iter 50 value 0.114759
iter 60 value 0.110991

```

```

iter 70 value 0.106432
iter 80 value 0.101529
iter 90 value 0.098680
iter 100 value 0.095452
final value 0.095452
stopped after 100 iterations
# weights: 11
initial value 213.478301
iter 10 value 79.156289
iter 20 value 70.690607
iter 30 value 58.559159
iter 40 value 24.923634
iter 50 value 19.001323
iter 60 value 18.658227
iter 70 value 17.863139
iter 80 value 17.684084
iter 90 value 17.633197
iter 100 value 17.533082
final value 17.533082
stopped after 100 iterations
# weights: 27
initial value 223.621194
iter 10 value 1.049270
iter 20 value 0.013039
iter 30 value 0.001452
iter 40 value 0.000633
iter 50 value 0.000174
final value 0.000089
converged
# weights: 43
initial value 241.851414
iter 10 value 2.982298
iter 20 value 0.017490
iter 30 value 0.000358
final value 0.000070
converged
# weights: 11
initial value 252.958177
iter 10 value 104.123036
iter 20 value 81.879608
iter 30 value 65.199882
final value 64.990981
converged
# weights: 27
initial value 247.968713
iter 10 value 25.697491

```

```
iter 20 value 17.603953
iter 30 value 17.228060
iter 40 value 16.699686
iter 50 value 16.682225
final value 16.681592
converged
# weights: 43
initial value 294.746062
iter 10 value 24.777446
iter 20 value 16.537273
iter 30 value 15.133799
iter 40 value 15.043412
iter 50 value 15.034247
iter 60 value 15.019561
iter 70 value 15.003194
iter 80 value 14.999163
final value 14.999079
converged
# weights: 11
initial value 268.039614
iter 10 value 96.654015
iter 20 value 59.841248
iter 30 value 38.768501
iter 40 value 36.942870
iter 50 value 36.505747
iter 60 value 36.478858
iter 60 value 36.478858
iter 70 value 36.477793
iter 80 value 36.469184
final value 36.469173
converged
# weights: 27
initial value 237.594325
iter 10 value 1.540643
iter 20 value 0.107991
iter 30 value 0.103601
iter 40 value 0.097572
iter 50 value 0.092465
iter 60 value 0.090711
iter 70 value 0.087068
iter 80 value 0.085586
iter 90 value 0.084606
iter 100 value 0.081880
final value 0.081880
stopped after 100 iterations
# weights: 43
```

```

initial value 294.436977
iter 10 value 27.896618
iter 20 value 4.811514
iter 30 value 1.071103
iter 40 value 0.975117
iter 50 value 0.845306
iter 60 value 0.667944
iter 70 value 0.345661
iter 80 value 0.232072
iter 90 value 0.183813
iter 100 value 0.142571
final value 0.142571
stopped after 100 iterations
# weights: 11
initial value 245.226744
iter 10 value 118.114922
iter 20 value 111.633358
iter 30 value 110.609773
iter 40 value 43.202707
iter 50 value 29.255375
iter 60 value 28.325088
iter 70 value 28.275247
iter 80 value 28.268243
iter 90 value 28.262987
iter 100 value 28.250209
final value 28.250209
stopped after 100 iterations
# weights: 27
initial value 227.819311
iter 10 value 4.681489
iter 20 value 0.010107
final value 0.000074
converged
# weights: 43
initial value 221.315366
iter 10 value 0.810212
iter 20 value 0.025480
iter 30 value 0.002417
iter 40 value 0.001030
final value 0.000091
converged
# weights: 11
initial value 232.652524
iter 10 value 91.735152
iter 20 value 90.103013
iter 30 value 84.147450

```

```
iter 40 value 81.516254
final value 81.516137
converged
# weights: 27
initial value 202.857378
iter 10 value 32.425973
iter 20 value 19.522602
iter 30 value 18.939899
iter 40 value 18.114148
iter 50 value 17.788937
iter 60 value 17.757291
final value 17.757144
converged
# weights: 43
initial value 223.391196
iter 10 value 33.178434
iter 20 value 18.089900
iter 30 value 16.418451
iter 40 value 16.013275
iter 50 value 15.995862
iter 60 value 15.990455
iter 70 value 15.989364
iter 80 value 15.988852
final value 15.988843
converged
# weights: 11
initial value 221.329627
iter 10 value 83.113626
iter 20 value 79.564464
iter 30 value 79.492213
iter 40 value 77.400120
iter 50 value 71.433260
iter 60 value 27.524819
iter 70 value 23.358276
iter 80 value 22.401656
iter 90 value 22.035753
iter 100 value 22.033241
final value 22.033241
stopped after 100 iterations
# weights: 27
initial value 211.903202
iter 10 value 4.202529
iter 20 value 0.671987
iter 30 value 0.579744
iter 40 value 0.516268
iter 50 value 0.462214
```

```

iter 60 value 0.377564
iter 70 value 0.346738
iter 80 value 0.293420
iter 90 value 0.195545
iter 100 value 0.184940
final value 0.184940
stopped after 100 iterations
# weights: 43
initial value 245.431234
iter 10 value 7.149902
iter 20 value 0.194305
iter 30 value 0.172094
iter 40 value 0.153250
iter 50 value 0.135939
iter 60 value 0.124327
iter 70 value 0.113476
iter 80 value 0.107102
iter 90 value 0.102791
iter 100 value 0.100976
final value 0.100976
stopped after 100 iterations
# weights: 11
initial value 218.839996
iter 10 value 114.673512
iter 20 value 57.876191
iter 30 value 44.856453
iter 40 value 39.471891
iter 50 value 39.392397
iter 60 value 39.384421
iter 70 value 39.341118
iter 80 value 39.337361
iter 90 value 39.331638
iter 100 value 39.323408
final value 39.323408
stopped after 100 iterations
# weights: 27
initial value 280.412505
iter 10 value 3.018469
iter 20 value 0.102038
iter 30 value 0.006322
iter 40 value 0.004214
iter 50 value 0.000796
iter 60 value 0.000262
iter 70 value 0.000230
iter 80 value 0.000153
final value 0.000089

```

```

converged
# weights: 43
initial value 262.275899
iter 10 value 18.508786
iter 20 value 4.750405
iter 30 value 0.057191
iter 40 value 0.001781
final value 0.000074
converged
# weights: 11
initial value 210.918064
iter 10 value 87.096246
iter 20 value 84.228284
final value 84.227280
converged
# weights: 27
initial value 190.332428
iter 10 value 24.494274
iter 20 value 17.465405
iter 30 value 17.318471
final value 17.318097
converged
# weights: 43
initial value 209.800217
iter 10 value 23.452442
iter 20 value 15.838855
iter 30 value 15.664300
iter 40 value 15.642271
iter 50 value 15.633789
iter 60 value 15.625635
iter 70 value 15.625398
final value 15.625397
converged
# weights: 11
initial value 234.343379
iter 10 value 81.657950
iter 20 value 80.822872
iter 30 value 80.806536
iter 40 value 80.782220
iter 50 value 80.736146
iter 60 value 80.716385
iter 70 value 80.564420
iter 80 value 79.293017
iter 90 value 65.395090
iter 100 value 44.379560
final value 44.379560

```

```
stopped after 100 iterations
# weights: 27
initial value 237.497627
iter 10 value 2.460950
iter 20 value 0.120887
iter 30 value 0.112801
iter 40 value 0.109627
iter 50 value 0.105187
iter 60 value 0.103675
iter 70 value 0.102129
iter 80 value 0.101082
iter 90 value 0.100428
iter 100 value 0.099400
final value 0.099400
stopped after 100 iterations
# weights: 43
initial value 300.953924
iter 10 value 1.632366
iter 20 value 0.126127
iter 30 value 0.110764
iter 40 value 0.107627
iter 50 value 0.103970
iter 60 value 0.100007
iter 70 value 0.097696
iter 80 value 0.095510
iter 90 value 0.094642
iter 100 value 0.091719
final value 0.091719
stopped after 100 iterations
# weights: 11
initial value 223.999743
iter 10 value 79.751795
iter 20 value 77.965640
iter 30 value 73.873505
iter 40 value 50.223283
iter 50 value 32.340692
iter 60 value 31.523401
iter 70 value 30.874338
iter 80 value 30.681301
iter 90 value 30.680497
iter 100 value 30.653490
final value 30.653490
stopped after 100 iterations
# weights: 27
initial value 223.410261
iter 10 value 0.110857
```

```

iter 20 value 0.002968
iter 30 value 0.000520
iter 40 value 0.000345
final value 0.000065
converged
# weights: 43
initial value 197.537888
iter 10 value 3.731170
iter 20 value 0.069136
iter 30 value 0.002957
iter 40 value 0.000228
final value 0.000091
converged
# weights: 11
initial value 235.793433
iter 10 value 140.325177
iter 20 value 75.089974
iter 30 value 63.628841
final value 63.286823
converged
# weights: 27
initial value 311.451479
iter 10 value 47.414312
iter 20 value 20.615842
iter 30 value 18.978579
iter 40 value 18.625211
iter 50 value 18.606364
iter 60 value 18.605148
iter 70 value 18.604794
iter 70 value 18.604794
iter 70 value 18.604794
final value 18.604794
converged
# weights: 43
initial value 238.282591
iter 10 value 35.825287
iter 20 value 15.600755
iter 30 value 15.169958
iter 40 value 15.103277
iter 50 value 15.101785
iter 60 value 15.101681
final value 15.101673
converged
# weights: 11
initial value 224.497355
iter 10 value 84.152952

```

```

iter 20 value 79.824729
iter 30 value 79.348909
iter 40 value 76.071219
iter 50 value 48.264590
iter 60 value 32.844347
iter 70 value 31.954733
iter 80 value 31.265964
iter 90 value 31.248542
iter 100 value 31.237963
final value 31.237963
stopped after 100 iterations
# weights: 27
initial value 231.057897
iter 10 value 42.210451
iter 20 value 4.749274
iter 30 value 3.263886
iter 40 value 3.139121
iter 50 value 2.927545
iter 60 value 0.697141
iter 70 value 0.429420
iter 80 value 0.384823
iter 90 value 0.223377
iter 100 value 0.175649
final value 0.175649
stopped after 100 iterations
# weights: 43
initial value 247.815035
iter 10 value 1.416280
iter 20 value 0.172488
iter 30 value 0.157892
iter 40 value 0.145938
iter 50 value 0.138698
iter 60 value 0.124692
iter 70 value 0.109337
iter 80 value 0.098205
iter 90 value 0.090255
iter 100 value 0.086390
final value 0.086390
stopped after 100 iterations
# weights: 11
initial value 256.978990
iter 10 value 86.157070
iter 20 value 80.658238
iter 30 value 79.145504
iter 40 value 77.870060
iter 50 value 74.294887

```

```

iter 60 value 72.612725
iter 70 value 69.710562
iter 80 value 32.513121
iter 90 value 23.718438
iter 100 value 22.812001
final value 22.812001
stopped after 100 iterations
# weights: 27
initial value 250.169097
iter 10 value 11.596878
iter 20 value 4.831665
iter 30 value 0.038789
iter 40 value 0.013923
iter 50 value 0.003702
iter 60 value 0.002160
final value 0.000064
converged
# weights: 43
initial value 347.847568
iter 10 value 1.596996
iter 20 value 0.034851
iter 30 value 0.006339
iter 40 value 0.002059
iter 50 value 0.000488
iter 60 value 0.000183
final value 0.000086
converged
# weights: 11
initial value 211.189469
iter 10 value 97.902359
iter 20 value 70.486930
iter 30 value 65.535702
final value 65.430728
converged
# weights: 27
initial value 269.430030
iter 10 value 28.534081
iter 20 value 22.540197
iter 30 value 20.402726
iter 40 value 18.932526
iter 50 value 18.098045
iter 60 value 18.039585
final value 18.039584
converged
# weights: 43
initial value 242.495478

```

```
iter 10 value 32.261191
iter 20 value 18.460700
iter 30 value 16.940954
iter 40 value 16.494693
iter 50 value 16.339162
iter 60 value 16.310321
iter 70 value 16.301603
iter 80 value 16.299349
iter 90 value 16.298775
final value 16.298735
converged
# weights: 11
initial value 262.165126
iter 10 value 69.591123
iter 20 value 30.903890
iter 30 value 24.231538
iter 40 value 22.625960
iter 50 value 22.390490
iter 60 value 22.354836
iter 70 value 22.289931
iter 80 value 22.286993
iter 90 value 22.284465
final value 22.284226
converged
# weights: 27
initial value 224.114923
iter 10 value 3.729851
iter 20 value 0.203638
iter 30 value 0.178392
iter 40 value 0.165669
iter 50 value 0.145040
iter 60 value 0.132757
iter 70 value 0.124365
iter 80 value 0.121084
iter 90 value 0.114227
iter 100 value 0.108749
final value 0.108749
stopped after 100 iterations
# weights: 43
initial value 213.414410
iter 10 value 3.086442
iter 20 value 0.159845
iter 30 value 0.128454
iter 40 value 0.119731
iter 50 value 0.115676
iter 60 value 0.111107
```

```

iter 70 value 0.108158
iter 80 value 0.104348
iter 90 value 0.102262
iter 100 value 0.101341
final value 0.101341
stopped after 100 iterations
# weights: 11
initial value 236.287566
iter 10 value 88.923865
iter 20 value 79.952024
iter 30 value 79.300008
iter 40 value 79.156282
iter 50 value 78.061898
iter 60 value 75.988801
iter 70 value 48.620600
iter 80 value 33.825966
iter 90 value 33.005546
iter 100 value 32.739857
final value 32.739857
stopped after 100 iterations
# weights: 27
initial value 219.966950
iter 10 value 0.504847
iter 20 value 0.020839
iter 30 value 0.007240
iter 40 value 0.000315
iter 50 value 0.000281
final value 0.000058
converged
# weights: 43
initial value 238.034135
iter 10 value 0.989992
iter 20 value 0.007596
iter 30 value 0.000318
final value 0.000097
converged
# weights: 11
initial value 223.844135
iter 10 value 75.104240
iter 20 value 63.627874
iter 30 value 63.627565
final value 63.627545
converged
# weights: 27
initial value 252.656578
iter 10 value 36.527949

```

```
iter 20 value 20.597554
iter 30 value 19.676526
iter 40 value 19.512077
final value 19.507961
converged
# weights: 43
initial value 247.837803
iter 10 value 31.386496
iter 20 value 16.810140
iter 30 value 16.480016
iter 40 value 16.412261
iter 50 value 16.353335
iter 60 value 16.348578
final value 16.348521
converged
# weights: 11
initial value 238.583454
iter 10 value 67.848139
iter 20 value 35.744591
iter 30 value 19.784733
iter 40 value 18.289854
iter 50 value 18.233364
iter 60 value 18.182426
final value 18.182422
converged
# weights: 27
initial value 239.215949
iter 10 value 2.111425
iter 20 value 0.402823
iter 30 value 0.366068
iter 40 value 0.339906
iter 50 value 0.254482
iter 60 value 0.179186
iter 70 value 0.165662
iter 80 value 0.158255
iter 90 value 0.134304
iter 100 value 0.122866
final value 0.122866
stopped after 100 iterations
# weights: 43
initial value 225.726387
iter 10 value 1.242309
iter 20 value 0.163697
iter 30 value 0.158046
iter 40 value 0.150940
iter 50 value 0.139076
```

```
iter 60 value 0.127978
iter 70 value 0.119302
iter 80 value 0.113153
iter 90 value 0.107559
iter 100 value 0.103977
final value 0.103977
stopped after 100 iterations
# weights: 11
initial value 211.314437
iter 10 value 80.230783
iter 20 value 79.505102
iter 30 value 79.499340
final value 79.499057
converged
# weights: 27
initial value 263.976785
iter 10 value 4.492069
iter 20 value 0.047065
iter 30 value 0.000245
final value 0.000086
converged
# weights: 43
initial value 235.038761
iter 10 value 0.917913
iter 20 value 0.016160
iter 30 value 0.001277
iter 40 value 0.000314
iter 50 value 0.000219
final value 0.000087
converged
# weights: 11
initial value 256.194098
iter 10 value 145.774288
iter 20 value 129.183371
iter 30 value 108.399330
iter 40 value 82.394106
iter 50 value 62.665400
iter 60 value 61.132352
final value 61.132349
converged
# weights: 27
initial value 209.778321
iter 10 value 30.548372
iter 20 value 19.467186
iter 30 value 18.499493
iter 40 value 18.344750
```

```
iter 50 value 18.275082
iter 60 value 18.255748
final value 18.255739
converged
# weights: 43
initial value 266.563900
iter 10 value 36.095932
iter 20 value 17.195234
iter 30 value 17.100421
iter 40 value 16.885827
iter 50 value 15.279573
iter 60 value 14.799001
iter 70 value 14.536694
iter 80 value 14.523632
iter 90 value 14.518487
iter 100 value 14.518036
final value 14.518036
stopped after 100 iterations
# weights: 11
initial value 217.727868
iter 10 value 78.428284
iter 20 value 62.163635
iter 30 value 39.770485
iter 40 value 32.138783
iter 50 value 32.029088
iter 60 value 31.886428
iter 70 value 31.806627
iter 80 value 31.766149
iter 90 value 31.724469
iter 100 value 31.718724
final value 31.718724
stopped after 100 iterations
# weights: 27
initial value 229.250172
iter 10 value 18.090255
iter 20 value 0.267629
iter 30 value 0.228079
iter 40 value 0.210100
iter 50 value 0.197122
iter 60 value 0.176293
iter 70 value 0.165632
iter 80 value 0.137777
iter 90 value 0.128798
iter 100 value 0.120700
final value 0.120700
stopped after 100 iterations
```

```

# weights: 43
initial value 307.057335
iter 10 value 1.248844
iter 20 value 0.119420
iter 30 value 0.108364
iter 40 value 0.103699
iter 50 value 0.099683
iter 60 value 0.095460
iter 70 value 0.092677
iter 80 value 0.089567
iter 90 value 0.084371
iter 100 value 0.082394
final value 0.082394
stopped after 100 iterations
# weights: 11
initial value 272.319520
iter 10 value 45.451367
iter 20 value 20.625122
iter 30 value 18.380865
iter 40 value 17.737616
iter 50 value 17.180782
iter 60 value 17.051080
iter 70 value 16.899171
iter 80 value 16.816004
iter 90 value 16.752098
iter 100 value 16.637003
final value 16.637003
stopped after 100 iterations
# weights: 27
initial value 207.574831
iter 10 value 18.447065
iter 20 value 0.268253
iter 30 value 0.006321
iter 40 value 0.000907
iter 50 value 0.000119
final value 0.000099
converged
# weights: 43
initial value 217.248229
iter 10 value 3.184334
iter 20 value 0.025160
iter 30 value 0.000742
iter 40 value 0.000526
final value 0.000094
converged
# weights: 11

```

```

initial value 230.023628
iter 10 value 96.586297
iter 20 value 67.429127
iter 30 value 62.569510
final value 62.545549
converged
# weights: 27
initial value 216.708109
iter 10 value 57.816895
iter 20 value 18.909957
iter 30 value 18.397343
iter 40 value 18.372954
iter 50 value 18.368649
final value 18.368621
converged
# weights: 43
initial value 312.139012
iter 10 value 46.244022
iter 20 value 16.754112
iter 30 value 15.478263
iter 40 value 15.379956
iter 50 value 15.371862
iter 60 value 15.371542
iter 70 value 15.371532
final value 15.371531
converged
# weights: 11
initial value 238.954256
iter 10 value 78.130847
iter 20 value 69.168895
iter 30 value 68.414858
iter 40 value 65.816340
iter 50 value 59.078838
iter 60 value 26.808608
iter 70 value 18.877669
iter 80 value 18.535506
iter 90 value 17.886991
iter 100 value 17.826540
final value 17.826540
stopped after 100 iterations
# weights: 27
initial value 247.121493
iter 10 value 1.146445
iter 20 value 0.173951
iter 30 value 0.143791
iter 40 value 0.136481

```

```

iter 50 value 0.126877
iter 60 value 0.116434
iter 70 value 0.109447
iter 80 value 0.102157
iter 90 value 0.096906
iter 100 value 0.091976
final value 0.091976
stopped after 100 iterations
# weights: 43
initial value 202.011648
iter 10 value 0.268358
iter 20 value 0.155419
iter 30 value 0.143495
iter 40 value 0.123826
iter 50 value 0.109162
iter 60 value 0.104700
iter 70 value 0.101301
iter 80 value 0.093480
iter 90 value 0.086176
iter 100 value 0.084245
final value 0.084245
stopped after 100 iterations
# weights: 11
initial value 221.435954
iter 10 value 60.329945
iter 20 value 20.488089
iter 30 value 18.202630
iter 40 value 17.604961
iter 50 value 17.103057
iter 60 value 17.061745
iter 70 value 16.862867
iter 80 value 16.851696
iter 90 value 16.837814
iter 100 value 16.819138
final value 16.819138
stopped after 100 iterations
# weights: 27
initial value 225.465737
iter 10 value 5.695988
iter 20 value 0.848927
iter 30 value 0.144999
iter 40 value 0.004241
iter 50 value 0.000723
iter 60 value 0.000449
iter 70 value 0.000253
final value 0.000094

```

```

converged
# weights: 43
initial value 248.484685
iter 10 value 13.272741
iter 20 value 4.910142
iter 30 value 2.821638
iter 40 value 1.884040
iter 50 value 0.085563
iter 60 value 0.002832
iter 70 value 0.000457
final value 0.000070
converged
# weights: 11
initial value 280.892547
iter 10 value 104.379076
iter 20 value 92.129931
iter 30 value 65.467845
iter 40 value 62.026566
iter 40 value 62.026566
iter 40 value 62.026566
final value 62.026566
converged
# weights: 27
initial value 272.204144
iter 10 value 34.450526
iter 20 value 20.796806
iter 30 value 18.262064
iter 40 value 18.201011
iter 50 value 18.196588
final value 18.196588
converged
# weights: 43
initial value 201.868013
iter 10 value 19.285517
iter 20 value 15.467986
iter 30 value 15.314961
iter 40 value 15.276827
iter 50 value 15.262223
final value 15.261688
converged
# weights: 11
initial value 253.723768
iter 10 value 98.599817
iter 20 value 82.595471
iter 30 value 80.969040
iter 40 value 79.956721

```

```

iter 50 value 79.840147
iter 60 value 79.839301
iter 70 value 79.839038
iter 70 value 79.839038
final value 79.839038
converged
# weights: 27
initial value 246.614911
iter 10 value 0.901612
iter 20 value 0.176308
iter 30 value 0.157887
iter 40 value 0.144868
iter 50 value 0.139263
iter 60 value 0.135151
iter 70 value 0.127305
iter 80 value 0.116159
iter 90 value 0.107639
iter 100 value 0.096222
final value 0.096222
stopped after 100 iterations
# weights: 43
initial value 255.809507
iter 10 value 2.747778
iter 20 value 0.149407
iter 30 value 0.104854
iter 40 value 0.098581
iter 50 value 0.096059
iter 60 value 0.091691
iter 70 value 0.088112
iter 80 value 0.084970
iter 90 value 0.083112
iter 100 value 0.081336
final value 0.081336
stopped after 100 iterations
# weights: 11
initial value 216.791114
iter 10 value 87.833130
iter 20 value 38.951690
iter 30 value 21.633360
iter 40 value 12.266223
iter 50 value 11.317986
iter 60 value 10.782454
iter 70 value 10.354592
iter 80 value 10.261939
iter 90 value 10.184640
iter 100 value 10.159837

```

```
final  value 10.159837
stopped after 100 iterations
# weights: 27
initial  value 213.829179
iter  10 value 1.139899
iter  20 value 0.045111
iter  30 value 0.001313
iter  40 value 0.000151
iter  40 value 0.000072
iter  40 value 0.000071
final  value 0.000071
converged
# weights: 43
initial  value 261.767522
iter  10 value 13.173789
iter  20 value 10.076197
iter  30 value 0.060645
iter  40 value 0.006256
iter  50 value 0.000279
final  value 0.000088
converged
# weights: 11
initial  value 260.415093
iter  10 value 86.254841
iter  20 value 62.255905
iter  30 value 61.790841
final  value 61.775145
converged
# weights: 27
initial  value 222.193481
iter  10 value 31.120858
iter  20 value 17.497807
iter  30 value 17.233643
iter  40 value 17.166362
iter  50 value 17.119848
final  value 17.119790
converged
# weights: 43
initial  value 266.653294
iter  10 value 41.833676
iter  20 value 16.999200
iter  30 value 15.773192
iter  40 value 15.608109
iter  50 value 15.574982
iter  60 value 15.564879
iter  70 value 15.562600
```

```

final  value 15.562584
converged
# weights: 11
initial  value 273.508295
iter   10 value 134.197725
iter   20 value 113.062612
iter   30 value 66.662319
iter   40 value 38.731850
iter   50 value 38.011224
iter   60 value 37.963810
iter   70 value 37.944041
iter   70 value 37.944041
final  value 37.944041
converged
# weights: 27
initial  value 273.744393
iter   10 value 17.708898
iter   20 value 0.431873
iter   30 value 0.301281
iter   40 value 0.272737
iter   50 value 0.243644
iter   60 value 0.202279
iter   70 value 0.169134
iter   80 value 0.162855
iter   90 value 0.145039
iter  100 value 0.132034
final  value 0.132034
stopped after 100 iterations
# weights: 43
initial  value 206.931563
iter   10 value 5.228960
iter   20 value 0.199108
iter   30 value 0.185150
iter   40 value 0.132792
iter   50 value 0.119083
iter   60 value 0.109431
iter   70 value 0.103103
iter   80 value 0.098916
iter   90 value 0.097518
iter  100 value 0.096752
final  value 0.096752
stopped after 100 iterations
# weights: 11
initial  value 256.443959
iter   10 value 80.910057
iter   20 value 80.188007

```

```
iter 30 value 78.637570
iter 40 value 76.245887
iter 50 value 51.714673
iter 60 value 38.118528
iter 70 value 37.473062
iter 80 value 37.372973
iter 90 value 37.261777
iter 100 value 37.260569
final value 37.260569
stopped after 100 iterations
# weights: 27
initial value 217.876334
iter 10 value 5.395228
iter 20 value 0.017533
final value 0.000096
converged
# weights: 43
initial value 210.180192
iter 10 value 6.645866
iter 20 value 0.040027
iter 30 value 0.002699
final value 0.000069
converged
# weights: 11
initial value 260.287721
iter 10 value 74.872000
iter 20 value 64.393179
iter 30 value 64.386360
final value 64.383939
converged
# weights: 27
initial value 313.714698
iter 10 value 40.071937
iter 20 value 19.852850
iter 30 value 19.540523
iter 40 value 19.465867
iter 50 value 19.463544
iter 60 value 19.463348
final value 19.463335
converged
# weights: 43
initial value 250.939669
iter 10 value 21.797873
iter 20 value 16.594536
iter 30 value 16.377708
iter 40 value 16.360934
```

```
iter 50 value 16.360743
final value 16.360639
converged
# weights: 11
initial value 230.981757
iter 10 value 95.896826
iter 20 value 76.652928
iter 30 value 74.482887
iter 40 value 74.142749
iter 50 value 73.222291
iter 60 value 73.094436
iter 70 value 73.081351
iter 80 value 72.741891
iter 90 value 71.986484
iter 100 value 54.415416
final value 54.415416
stopped after 100 iterations
# weights: 27
initial value 212.902311
iter 10 value 2.146002
iter 20 value 0.303056
iter 30 value 0.234635
iter 40 value 0.184213
iter 50 value 0.178069
iter 60 value 0.140879
iter 70 value 0.133038
iter 80 value 0.128831
iter 90 value 0.118918
iter 100 value 0.113556
final value 0.113556
stopped after 100 iterations
# weights: 43
initial value 198.816399
iter 10 value 5.825207
iter 20 value 0.436314
iter 30 value 0.344034
iter 40 value 0.274977
iter 50 value 0.210571
iter 60 value 0.171355
iter 70 value 0.148956
iter 80 value 0.130880
iter 90 value 0.122302
iter 100 value 0.117262
final value 0.117262
stopped after 100 iterations
# weights: 11
```

```

initial value 219.407967
iter 10 value 81.213895
iter 20 value 73.276923
iter 30 value 45.410528
iter 40 value 32.320197
iter 50 value 31.656231
iter 60 value 31.650889
iter 70 value 31.639626
iter 80 value 31.627471
iter 90 value 31.618799
iter 100 value 31.618129
final value 31.618129
stopped after 100 iterations
# weights: 27
initial value 206.105944
iter 10 value 1.702266
iter 20 value 0.132757
iter 30 value 0.001177
iter 40 value 0.000117
iter 40 value 0.000087
iter 40 value 0.000083
final value 0.000083
converged
# weights: 43
initial value 197.998616
iter 10 value 0.610056
iter 20 value 0.003874
iter 30 value 0.000143
iter 30 value 0.000087
iter 30 value 0.000087
final value 0.000087
converged
# weights: 11
initial value 240.152241
iter 10 value 95.176716
iter 20 value 84.022532
iter 30 value 65.733063
iter 40 value 64.762097
iter 40 value 64.762097
iter 40 value 64.762097
final value 64.762097
converged
# weights: 27
initial value 228.157113
iter 10 value 27.353327
iter 20 value 17.731584

```

```
iter 30 value 16.493577
iter 40 value 16.367992
iter 50 value 16.356636
iter 50 value 16.356636
iter 50 value 16.356636
final value 16.356636
converged
# weights: 43
initial value 235.818570
iter 10 value 25.399385
iter 20 value 15.599327
iter 30 value 15.138884
iter 40 value 14.891874
iter 50 value 14.877901
iter 60 value 14.876585
final value 14.876584
converged
# weights: 11
initial value 216.152846
iter 10 value 81.359503
iter 20 value 51.584133
iter 30 value 32.553296
iter 40 value 32.045343
iter 50 value 31.949822
final value 31.942670
converged
# weights: 27
initial value 238.459960
iter 10 value 2.463972
iter 20 value 0.220683
iter 30 value 0.199029
iter 40 value 0.166172
iter 50 value 0.142904
iter 60 value 0.133076
iter 70 value 0.126675
iter 80 value 0.110806
iter 90 value 0.103824
iter 100 value 0.095894
final value 0.095894
stopped after 100 iterations
# weights: 43
initial value 213.101936
iter 10 value 0.550803
iter 20 value 0.201843
iter 30 value 0.179468
iter 40 value 0.131784
```

```
iter 50 value 0.105426
iter 60 value 0.092458
iter 70 value 0.083032
iter 80 value 0.079736
iter 90 value 0.075559
iter 100 value 0.074859
final value 0.074859
stopped after 100 iterations
# weights: 11
initial value 242.385282
iter 10 value 63.905996
iter 20 value 23.205198
iter 30 value 20.117014
iter 40 value 19.672909
iter 50 value 19.250513
iter 60 value 19.244228
iter 70 value 19.159943
iter 80 value 19.143660
iter 90 value 19.137940
iter 100 value 19.120943
final value 19.120943
stopped after 100 iterations
# weights: 27
initial value 220.048225
iter 10 value 15.551701
iter 20 value 0.051093
iter 30 value 0.005949
iter 40 value 0.000863
final value 0.000092
converged
# weights: 43
initial value 186.898881
iter 10 value 9.189225
iter 20 value 0.085033
iter 30 value 0.005108
iter 40 value 0.000570
final value 0.000066
converged
# weights: 11
initial value 237.864679
iter 10 value 114.275897
iter 20 value 92.854308
iter 30 value 85.080077
final value 84.897299
converged
# weights: 27
```

```
initial value 273.103163
iter 10 value 31.681302
iter 20 value 19.801334
iter 30 value 19.420460
iter 40 value 17.440989
iter 50 value 17.350686
iter 60 value 17.345294
final value 17.345199
converged
# weights: 43
initial value 293.181287
iter 10 value 18.824053
iter 20 value 16.072080
iter 30 value 15.732135
iter 40 value 15.601209
iter 50 value 15.588016
iter 60 value 15.586275
final value 15.586222
converged
# weights: 11
initial value 222.687659
iter 10 value 80.823988
iter 20 value 80.247632
iter 30 value 70.693663
iter 40 value 46.794731
iter 50 value 36.791492
iter 60 value 36.134255
iter 70 value 36.024411
iter 80 value 36.013233
iter 90 value 35.983398
iter 100 value 35.967102
final value 35.967102
stopped after 100 iterations
# weights: 27
initial value 260.157095
iter 10 value 3.011955
iter 20 value 0.118838
iter 30 value 0.104414
iter 40 value 0.099641
iter 50 value 0.097340
iter 60 value 0.094933
iter 70 value 0.092363
iter 80 value 0.091154
iter 90 value 0.090249
iter 100 value 0.089779
final value 0.089779
```

```

stopped after 100 iterations
# weights: 43
initial value 213.518960
iter 10 value 2.197955
iter 20 value 0.266114
iter 30 value 0.242320
iter 40 value 0.201775
iter 50 value 0.149992
iter 60 value 0.121400
iter 70 value 0.109309
iter 80 value 0.102237
iter 90 value 0.097233
iter 100 value 0.091265
final value 0.091265
stopped after 100 iterations
# weights: 11
initial value 276.987663
iter 10 value 83.042253
iter 20 value 77.753999
iter 30 value 76.543536
iter 40 value 56.780163
iter 50 value 37.454580
iter 60 value 34.674292
iter 70 value 34.243104
iter 80 value 34.228659
final value 34.227907
converged
# weights: 27
initial value 217.497394
iter 10 value 2.594675
iter 20 value 0.080460
iter 30 value 0.007355
iter 40 value 0.000342
final value 0.000094
converged
# weights: 43
initial value 294.026694
iter 10 value 2.947559
iter 20 value 0.033285
iter 30 value 0.012165
iter 40 value 0.001393
iter 50 value 0.000754
iter 60 value 0.000414
iter 70 value 0.000301
iter 80 value 0.000115
final value 0.000097

```

```

converged
# weights: 11
initial value 229.972604
iter 10 value 92.128651
iter 20 value 85.397703
iter 30 value 63.555517
final value 63.473300
converged
# weights: 27
initial value 230.150421
iter 10 value 27.544130
iter 20 value 17.779227
iter 30 value 16.804644
iter 40 value 16.452698
iter 50 value 16.445413
final value 16.445413
converged
# weights: 43
initial value 272.989120
iter 10 value 26.154555
iter 20 value 16.931219
iter 30 value 16.836201
iter 40 value 16.831950
iter 50 value 16.831689
iter 60 value 16.831572
iter 60 value 16.831572
iter 60 value 16.831572
final value 16.831572
converged
# weights: 11
initial value 218.815325
iter 10 value 80.358927
iter 20 value 78.378501
iter 30 value 73.575470
iter 40 value 43.711212
iter 50 value 34.994781
iter 60 value 34.737833
iter 70 value 34.532135
final value 34.525613
converged
# weights: 27
initial value 251.993808
iter 10 value 4.389506
iter 20 value 0.375590
iter 30 value 0.241058
iter 40 value 0.195043

```

```

iter 50 value 0.160643
iter 60 value 0.126521
iter 70 value 0.121253
iter 80 value 0.104732
iter 90 value 0.096588
iter 100 value 0.092788
final value 0.092788
stopped after 100 iterations
# weights: 43
initial value 260.356699
iter 10 value 1.629176
iter 20 value 0.145502
iter 30 value 0.121213
iter 40 value 0.118672
iter 50 value 0.116445
iter 60 value 0.106861
iter 70 value 0.103054
iter 80 value 0.095852
iter 90 value 0.093574
iter 100 value 0.091747
final value 0.091747
stopped after 100 iterations
# weights: 11
initial value 290.866912
iter 10 value 102.559958
iter 20 value 46.998686
iter 30 value 36.008320
iter 40 value 34.821932
iter 50 value 34.804801
iter 50 value 34.804800
iter 60 value 34.800039
iter 70 value 34.769913
iter 80 value 34.767798
iter 90 value 34.752498
iter 100 value 34.742985
final value 34.742985
stopped after 100 iterations
# weights: 27
initial value 208.561736
iter 10 value 16.434326
iter 20 value 0.311247
iter 30 value 0.006252
final value 0.000044
converged
# weights: 43
initial value 237.882781

```

```

iter 10 value 1.642101
iter 20 value 0.030165
iter 30 value 0.001221
iter 40 value 0.000268
final value 0.000060
converged
# weights: 11
initial value 230.770084
iter 10 value 94.242929
iter 20 value 89.460996
iter 30 value 84.794403
iter 40 value 84.452229
iter 40 value 84.452229
iter 40 value 84.452229
final value 84.452229
converged
# weights: 27
initial value 236.239770
iter 10 value 23.747888
iter 20 value 18.027352
iter 30 value 17.542470
iter 40 value 17.537029
iter 50 value 17.536916
final value 17.536912
converged
# weights: 43
initial value 226.852603
iter 10 value 32.091261
iter 20 value 16.055815
iter 30 value 15.837501
iter 40 value 15.815705
iter 50 value 15.786647
iter 60 value 15.764788
iter 70 value 15.759810
final value 15.759755
converged
# weights: 11
initial value 288.746304
iter 10 value 75.579306
iter 20 value 28.061658
iter 30 value 22.919490
iter 40 value 22.075827
iter 50 value 21.780887
iter 60 value 21.745210
iter 70 value 21.742507
final value 21.737892

```

```

converged
# weights: 27
initial value 273.489028
iter 10 value 1.618679
iter 20 value 0.159075
iter 30 value 0.147122
iter 40 value 0.138046
iter 50 value 0.120829
iter 60 value 0.115734
iter 70 value 0.109083
iter 80 value 0.104645
iter 90 value 0.097542
iter 100 value 0.093987
final value 0.093987
stopped after 100 iterations
# weights: 43
initial value 229.231364
iter 10 value 3.352605
iter 20 value 0.128085
iter 30 value 0.103850
iter 40 value 0.095686
iter 50 value 0.093697
iter 60 value 0.091426
iter 70 value 0.089579
iter 80 value 0.088618
iter 90 value 0.087660
iter 100 value 0.087084
final value 0.087084
stopped after 100 iterations
# weights: 11
initial value 233.177236
iter 10 value 49.545478
iter 20 value 21.100256
iter 30 value 19.685890
iter 40 value 18.380426
iter 50 value 17.989890
iter 60 value 17.795120
iter 70 value 17.697489
iter 80 value 17.674472
iter 90 value 17.653959
iter 100 value 17.596345
final value 17.596345
stopped after 100 iterations
# weights: 27
initial value 200.233555
iter 10 value 1.438719

```

```
iter 20 value 0.006521
iter 30 value 0.000390
final value 0.000076
converged
# weights: 43
initial value 212.726081
iter 10 value 2.618154
iter 20 value 0.051541
iter 30 value 0.007471
iter 40 value 0.000907
iter 50 value 0.000150
final value 0.000091
converged
# weights: 11
initial value 220.598324
iter 10 value 67.960135
iter 20 value 62.566716
iter 30 value 62.554575
final value 62.554565
converged
# weights: 27
initial value 227.640354
iter 10 value 51.430200
iter 20 value 22.051987
iter 30 value 17.815898
iter 40 value 17.332095
iter 50 value 17.087196
iter 60 value 16.824192
iter 70 value 16.789372
final value 16.789353
converged
# weights: 43
initial value 270.900044
iter 10 value 26.375738
iter 20 value 15.864787
iter 30 value 15.518551
iter 40 value 15.353259
iter 50 value 15.290850
iter 60 value 15.249840
final value 15.249615
converged
# weights: 11
initial value 218.409287
iter 10 value 79.403202
iter 20 value 50.627263
iter 30 value 19.772700
```

```

iter 40 value 18.477913
iter 50 value 18.433768
iter 60 value 18.302665
iter 70 value 18.300578
iter 80 value 18.299680
final value 18.299649
converged
# weights: 27
initial value 218.046442
iter 10 value 3.253623
iter 20 value 0.224736
iter 30 value 0.156160
iter 40 value 0.148513
iter 50 value 0.142239
iter 60 value 0.129553
iter 70 value 0.124999
iter 80 value 0.116040
iter 90 value 0.109624
iter 100 value 0.105424
final value 0.105424
stopped after 100 iterations
# weights: 43
initial value 264.774620
iter 10 value 2.548373
iter 20 value 0.148965
iter 30 value 0.140608
iter 40 value 0.132357
iter 50 value 0.128219
iter 60 value 0.120070
iter 70 value 0.110606
iter 80 value 0.105165
iter 90 value 0.093921
iter 100 value 0.091353
final value 0.091353
stopped after 100 iterations
# weights: 11
initial value 240.306246
iter 10 value 82.096829
iter 20 value 74.638214
iter 30 value 73.760374
iter 40 value 73.749756
iter 50 value 73.720829
iter 60 value 73.709978
iter 70 value 73.706752
iter 80 value 73.704757
iter 90 value 73.701638

```

```
iter 100 value 73.699473
final  value 73.699473
stopped after 100 iterations
# weights: 27
initial  value 292.551455
iter  10 value 2.391462
iter  20 value 0.083731
iter  30 value 0.001929
iter  40 value 0.000214
final   value 0.000091
converged
# weights: 43
initial  value 200.400708
iter  10 value 1.317733
iter  20 value 0.016753
iter  30 value 0.000278
final   value 0.000095
converged
# weights: 11
initial  value 220.247511
iter  10 value 133.599844
iter  20 value 92.140504
iter  30 value 86.459334
iter  40 value 83.725596
final   value 83.725390
converged
# weights: 27
initial  value 249.921577
iter  10 value 39.411507
iter  20 value 19.710575
iter  30 value 18.828645
iter  40 value 18.135673
iter  50 value 17.968647
final   value 17.968537
converged
# weights: 43
initial  value 247.170840
iter  10 value 48.045498
iter  20 value 18.339742
iter  30 value 16.611902
iter  40 value 16.357657
iter  50 value 16.335834
iter  60 value 16.327921
iter  70 value 16.326110
final   value 16.325895
converged
```

```

# weights: 11
initial value 234.164143
iter 10 value 87.194511
iter 20 value 78.687715
iter 30 value 70.487403
iter 40 value 33.809131
iter 50 value 22.087480
iter 60 value 21.131970
iter 70 value 20.368851
iter 80 value 20.336024
iter 90 value 20.330475
final value 20.325311
converged
# weights: 27
initial value 246.766235
iter 10 value 3.271004
iter 20 value 0.199156
iter 30 value 0.160903
iter 40 value 0.153347
iter 50 value 0.146525
iter 60 value 0.131646
iter 70 value 0.125250
iter 80 value 0.120530
iter 90 value 0.111870
iter 100 value 0.108583
final value 0.108583
stopped after 100 iterations
# weights: 43
initial value 273.386882
iter 10 value 4.352789
iter 20 value 0.268655
iter 30 value 0.237298
iter 40 value 0.204854
iter 50 value 0.172746
iter 60 value 0.143322
iter 70 value 0.135959
iter 80 value 0.119185
iter 90 value 0.114152
iter 100 value 0.111210
final value 0.111210
stopped after 100 iterations
# weights: 11
initial value 215.153928
iter 10 value 80.724632
iter 20 value 80.648482
iter 30 value 78.499533

```

```
iter 40 value 73.107568
iter 50 value 54.530583
iter 60 value 18.149042
iter 70 value 13.081112
iter 80 value 12.026010
iter 90 value 10.896493
iter 100 value 10.507511
final value 10.507511
stopped after 100 iterations
# weights: 27
initial value 334.204934
iter 10 value 22.221894
iter 20 value 13.360823
iter 30 value 13.117289
iter 40 value 5.580773
iter 50 value 5.090692
iter 60 value 0.001657
final value 0.000064
converged
# weights: 43
initial value 312.566482
iter 10 value 2.755355
iter 20 value 0.028955
iter 30 value 0.002552
final value 0.000063
converged
# weights: 11
initial value 215.284877
iter 10 value 105.729143
iter 20 value 85.147415
iter 30 value 84.801551
final value 84.773288
converged
# weights: 27
initial value 247.426682
iter 10 value 57.804835
iter 20 value 27.073166
iter 30 value 21.377860
iter 40 value 19.390229
iter 50 value 17.618161
iter 60 value 17.554931
iter 70 value 17.553981
iter 70 value 17.553981
iter 70 value 17.553981
final value 17.553981
converged
```

```

# weights: 43
initial value 236.503887
iter 10 value 30.060548
iter 20 value 16.790278
iter 30 value 16.030121
iter 40 value 15.910117
iter 50 value 15.843010
iter 60 value 15.800571
iter 70 value 15.793492
final value 15.793462
converged
# weights: 11
initial value 240.973328
iter 10 value 92.604399
iter 20 value 61.259342
iter 30 value 14.125928
iter 40 value 12.123694
iter 50 value 11.861075
iter 60 value 11.706180
iter 70 value 11.614072
iter 80 value 11.506994
iter 90 value 11.491748
iter 100 value 11.350326
final value 11.350326
stopped after 100 iterations
# weights: 27
initial value 208.641888
iter 10 value 3.412077
iter 20 value 0.245717
iter 30 value 0.209621
iter 40 value 0.194810
iter 50 value 0.169561
iter 60 value 0.152712
iter 70 value 0.137186
iter 80 value 0.127745
iter 90 value 0.116847
iter 100 value 0.107012
final value 0.107012
stopped after 100 iterations
# weights: 43
initial value 259.743537
iter 10 value 0.976534
iter 20 value 0.138933
iter 30 value 0.123325
iter 40 value 0.119603
iter 50 value 0.115527

```

```

iter 60 value 0.108455
iter 70 value 0.104790
iter 80 value 0.100986
iter 90 value 0.098755
iter 100 value 0.097672
final value 0.097672
stopped after 100 iterations
# weights: 11
initial value 225.877780
iter 10 value 52.279179
iter 20 value 22.370421
iter 30 value 21.032051
iter 40 value 20.041860
iter 50 value 19.765764
iter 60 value 19.744587
iter 70 value 19.640131
iter 80 value 19.627278
iter 90 value 19.598294
iter 100 value 19.578072
final value 19.578072
stopped after 100 iterations
# weights: 27
initial value 197.246909
iter 10 value 4.363805
iter 20 value 0.043908
iter 30 value 0.001874
iter 40 value 0.000171
iter 50 value 0.000117
iter 50 value 0.000099
iter 50 value 0.000099
final value 0.000099
converged
# weights: 43
initial value 274.162824
iter 10 value 12.156244
iter 20 value 0.141265
iter 30 value 0.015792
iter 40 value 0.000126
iter 40 value 0.000072
iter 40 value 0.000072
final value 0.000072
converged
# weights: 11
initial value 221.587621
iter 10 value 94.346018
iter 20 value 69.624959

```

```
iter 30 value 65.636830
final value 65.230459
converged
# weights: 27
initial value 237.417818
iter 10 value 28.937709
iter 20 value 18.224124
iter 30 value 18.172452
iter 40 value 18.171650
final value 18.171649
converged
# weights: 43
initial value 290.078127
iter 10 value 30.682229
iter 20 value 18.260497
iter 30 value 16.588428
iter 40 value 16.422642
iter 50 value 16.412093
iter 60 value 16.412015
iter 70 value 16.411990
final value 16.411990
converged
# weights: 11
initial value 239.474848
iter 10 value 102.079801
iter 20 value 81.893460
iter 30 value 79.885350
iter 40 value 76.710874
iter 50 value 74.578533
iter 60 value 74.379728
iter 70 value 74.068863
iter 80 value 73.365304
iter 90 value 73.296481
iter 100 value 73.261612
final value 73.261612
stopped after 100 iterations
# weights: 27
initial value 243.507733
iter 10 value 2.391394
iter 20 value 0.194043
iter 30 value 0.187229
iter 40 value 0.170937
iter 50 value 0.150138
iter 60 value 0.140103
iter 70 value 0.134866
iter 80 value 0.131818
```

```

iter 90 value 0.127023
iter 100 value 0.120716
final value 0.120716
stopped after 100 iterations
# weights: 43
initial value 225.448516
iter 10 value 1.923757
iter 20 value 0.142954
iter 30 value 0.133238
iter 40 value 0.130854
iter 50 value 0.126075
iter 60 value 0.113965
iter 70 value 0.106527
iter 80 value 0.103609
iter 90 value 0.102093
iter 100 value 0.100666
final value 0.100666
stopped after 100 iterations
# weights: 11
initial value 214.014205
iter 10 value 84.006232
iter 20 value 57.024931
iter 30 value 29.121530
iter 40 value 22.289609
iter 50 value 21.476495
iter 60 value 21.336455
iter 70 value 21.183238
iter 80 value 21.123300
iter 90 value 21.063488
iter 100 value 21.060430
final value 21.060430
stopped after 100 iterations
# weights: 27
initial value 226.995470
iter 10 value 5.215955
iter 20 value 0.014389
iter 30 value 0.000131
final value 0.000099
converged
# weights: 43
initial value 222.043401
iter 10 value 1.506137
iter 20 value 0.076181
iter 30 value 0.004301
final value 0.000073
converged

```

```

# weights: 11
initial value 252.164354
iter 10 value 107.761297
iter 20 value 86.217854
iter 30 value 63.878193
final value 63.803127
converged
# weights: 27
initial value 278.464964
iter 10 value 29.162604
iter 20 value 18.442633
iter 30 value 18.100207
iter 40 value 18.097615
iter 50 value 18.097460
final value 18.097447
converged
# weights: 43
initial value 210.064167
iter 10 value 19.404022
iter 20 value 16.413262
iter 30 value 16.374276
iter 40 value 16.368565
iter 50 value 16.366027
final value 16.365897
converged
# weights: 11
initial value 242.883716
iter 10 value 81.837301
iter 20 value 45.318114
iter 30 value 25.855267
iter 40 value 22.522800
iter 50 value 21.918707
iter 60 value 21.832511
iter 70 value 21.635589
iter 80 value 21.629800
iter 90 value 21.628712
final value 21.628696
converged
# weights: 27
initial value 260.866745
iter 10 value 17.775827
iter 20 value 0.539927
iter 30 value 0.381552
iter 40 value 0.356218
iter 50 value 0.298994
iter 60 value 0.263467

```

```

iter 70 value 0.240101
iter 80 value 0.224158
iter 90 value 0.183194
iter 100 value 0.172154
final value 0.172154
stopped after 100 iterations
# weights: 43
initial value 270.897447
iter 10 value 1.845752
iter 20 value 0.145728
iter 30 value 0.130917
iter 40 value 0.126174
iter 50 value 0.119496
iter 60 value 0.117099
iter 70 value 0.111983
iter 80 value 0.108603
iter 90 value 0.105573
iter 100 value 0.103751
final value 0.103751
stopped after 100 iterations
# weights: 11
initial value 200.390828
iter 10 value 79.810321
iter 20 value 27.346186
iter 30 value 14.459597
iter 40 value 14.093192
iter 50 value 13.807238
iter 60 value 13.429580
iter 70 value 13.368763
iter 80 value 13.285087
iter 90 value 13.273537
iter 100 value 13.261954
final value 13.261954
stopped after 100 iterations
# weights: 27
initial value 219.734455
iter 10 value 2.759314
iter 20 value 0.007769
final value 0.000077
converged
# weights: 43
initial value 300.041531
iter 10 value 0.317993
iter 20 value 0.002457
iter 30 value 0.000231
final value 0.000069

```

```

converged
# weights: 11
initial value 217.185842
iter 10 value 79.147293
iter 20 value 60.747185
iter 30 value 60.362053
final value 60.359822
converged
# weights: 27
initial value 222.072779
iter 10 value 28.362865
iter 20 value 17.753542
iter 30 value 16.118547
iter 40 value 16.108535
iter 50 value 16.107809
final value 16.107802
converged
# weights: 43
initial value 185.292215
iter 10 value 35.899011
iter 20 value 16.348460
iter 30 value 15.366383
iter 40 value 14.890094
iter 50 value 14.808769
iter 60 value 14.803686
iter 70 value 14.780791
iter 80 value 14.643520
iter 90 value 14.540193
iter 100 value 14.476730
final value 14.476730
stopped after 100 iterations
# weights: 11
initial value 215.642582
iter 10 value 84.975538
iter 20 value 65.546189
iter 30 value 29.213195
iter 40 value 15.134653
iter 50 value 14.406349
iter 60 value 14.190331
iter 70 value 14.102083
iter 80 value 14.092406
final value 14.087416
converged
# weights: 27
initial value 200.995236
iter 10 value 14.374197

```

```

iter 20 value 0.286356
iter 30 value 0.216352
iter 40 value 0.193789
iter 50 value 0.187647
iter 60 value 0.153320
iter 70 value 0.135021
iter 80 value 0.128062
iter 90 value 0.120587
iter 100 value 0.106930
final value 0.106930
stopped after 100 iterations
# weights: 43
initial value 202.387098
iter 10 value 0.787654
iter 20 value 0.104906
iter 30 value 0.094063
iter 40 value 0.089063
iter 50 value 0.085807
iter 60 value 0.080614
iter 70 value 0.075978
iter 80 value 0.071492
iter 90 value 0.069312
iter 100 value 0.067659
final value 0.067659
stopped after 100 iterations
# weights: 11
initial value 216.283774
iter 10 value 82.455520
iter 20 value 79.184840
iter 30 value 78.481312
iter 40 value 78.345233
iter 50 value 77.873207
iter 60 value 71.066577
iter 70 value 65.953586
iter 80 value 28.647510
iter 90 value 25.015401
iter 100 value 20.513655
final value 20.513655
stopped after 100 iterations
# weights: 27
initial value 212.511738
iter 10 value 7.782851
iter 20 value 0.096402
iter 30 value 0.001646
iter 40 value 0.000185
final value 0.000074

```

```
converged
# weights: 43
initial value 221.072679
iter 10 value 8.837359
iter 20 value 0.154506
iter 30 value 0.002805
final value 0.000066
converged
# weights: 11
initial value 222.308867
iter 10 value 88.016491
iter 20 value 68.156917
iter 30 value 62.779071
final value 62.762397
converged
# weights: 27
initial value 227.641411
iter 10 value 26.967949
iter 20 value 19.390137
iter 30 value 18.930448
iter 40 value 18.865161
iter 50 value 18.853376
final value 18.853291
converged
# weights: 43
initial value 198.656035
iter 10 value 21.801880
iter 20 value 15.528278
iter 30 value 15.034360
iter 40 value 14.823422
iter 50 value 14.811520
iter 60 value 14.801856
iter 70 value 14.791794
final value 14.791608
converged
# weights: 11
initial value 215.519043
iter 10 value 77.051683
iter 20 value 73.062254
iter 30 value 72.249204
iter 40 value 65.633086
iter 50 value 28.381409
iter 60 value 21.501883
iter 70 value 20.218076
iter 80 value 19.899043
iter 90 value 19.841715
```

```
iter 100 value 19.711357
final  value 19.711357
stopped after 100 iterations
# weights: 27
initial  value 258.053201
iter   10 value 11.863869
iter   20 value 0.258021
iter   30 value 0.187193
iter   40 value 0.167238
iter   50 value 0.155365
iter   60 value 0.140045
iter   70 value 0.134580
iter   80 value 0.121864
iter   90 value 0.112391
iter 100 value 0.107648
final  value 0.107648
stopped after 100 iterations
# weights: 43
initial  value 264.624679
iter   10 value 2.191219
iter   20 value 0.120058
iter   30 value 0.103588
iter   40 value 0.100248
iter   50 value 0.097219
iter   60 value 0.095187
iter   70 value 0.094224
iter   80 value 0.092540
iter   90 value 0.091137
iter 100 value 0.089957
final  value 0.089957
stopped after 100 iterations
# weights: 11
initial  value 221.236412
iter   10 value 86.850088
iter   20 value 80.767534
iter   30 value 79.386716
iter   40 value 72.870157
iter   50 value 41.989352
iter   60 value 31.029965
iter   70 value 17.147094
iter   80 value 16.893486
iter   90 value 16.803423
iter 100 value 16.514420
final  value 16.514420
stopped after 100 iterations
# weights: 27
```

```

initial value 230.719664
iter 10 value 16.989216
iter 20 value 0.411076
iter 30 value 0.005955
iter 40 value 0.000419
iter 50 value 0.000236
final value 0.000071
converged
# weights: 43
initial value 269.431611
iter 10 value 0.300930
iter 20 value 0.029275
iter 30 value 0.004909
final value 0.000087
converged
# weights: 11
initial value 229.592034
iter 10 value 92.205035
iter 20 value 83.572135
iter 30 value 63.323095
iter 40 value 62.990588
iter 40 value 62.990588
iter 40 value 62.990588
final value 62.990588
converged
# weights: 27
initial value 228.209933
iter 10 value 29.960309
iter 20 value 17.703011
iter 30 value 17.054915
iter 40 value 17.034971
final value 17.034600
converged
# weights: 43
initial value 217.262482
iter 10 value 26.878814
iter 20 value 16.371043
iter 30 value 15.608195
iter 40 value 15.550663
iter 50 value 15.542924
iter 60 value 15.535926
final value 15.535795
converged
# weights: 11
initial value 234.404797
iter 10 value 75.461281

```

```

iter 20 value 43.184695
iter 30 value 23.099339
iter 40 value 17.597477
iter 50 value 17.386053
iter 60 value 17.354503
iter 70 value 17.343880
iter 80 value 17.334674
iter 90 value 17.326726
iter 100 value 17.323920
final value 17.323920
stopped after 100 iterations
# weights: 27
initial value 229.722857
iter 10 value 0.973167
iter 20 value 0.131593
iter 30 value 0.121680
iter 40 value 0.118455
iter 50 value 0.112413
iter 60 value 0.104747
iter 70 value 0.099665
iter 80 value 0.093903
iter 90 value 0.087798
iter 100 value 0.083634
final value 0.083634
stopped after 100 iterations
# weights: 43
initial value 200.539391
iter 10 value 1.126555
iter 20 value 0.184108
iter 30 value 0.167209
iter 40 value 0.129240
iter 50 value 0.116047
iter 60 value 0.110334
iter 70 value 0.104769
iter 80 value 0.101440
iter 90 value 0.097602
iter 100 value 0.093731
final value 0.093731
stopped after 100 iterations
# weights: 11
initial value 232.288321
iter 10 value 135.319912
iter 20 value 113.935057
iter 30 value 86.407978
iter 40 value 83.241189
iter 50 value 69.077205

```

```

iter 60 value 29.674332
iter 70 value 17.660768
iter 80 value 16.312125
iter 90 value 16.143562
iter 100 value 15.623168
final value 15.623168
stopped after 100 iterations
# weights: 27
initial value 227.946265
iter 10 value 47.273988
iter 20 value 47.137318
iter 30 value 47.131356
iter 40 value 47.100011
iter 50 value 46.466756
iter 60 value 44.529532
iter 70 value 39.974087
iter 80 value 9.551367
iter 90 value 6.202128
iter 100 value 4.958450
final value 4.958450
stopped after 100 iterations
# weights: 43
initial value 328.144711
iter 10 value 5.249234
iter 20 value 0.070143
iter 30 value 0.002258
iter 40 value 0.000472
iter 50 value 0.000303
final value 0.000066
converged
# weights: 11
initial value 221.053259
iter 10 value 103.232316
iter 20 value 75.503770
iter 30 value 62.570834
final value 62.568209
converged
# weights: 27
initial value 233.694583
iter 10 value 31.418034
iter 20 value 20.546907
iter 30 value 20.476361
iter 40 value 20.468989
final value 20.468013
converged
# weights: 43

```

```
initial value 274.092534
iter 10 value 30.713811
iter 20 value 18.221439
iter 30 value 16.707030
iter 40 value 16.326632
iter 50 value 16.227845
iter 60 value 16.224456
iter 70 value 16.223888
final value 16.223881
converged
# weights: 11
initial value 235.436720
iter 10 value 56.026339
iter 20 value 19.456533
iter 30 value 16.989502
iter 40 value 16.535767
iter 50 value 16.289339
iter 60 value 16.261415
iter 70 value 16.236512
iter 80 value 16.236389
iter 90 value 16.236089
iter 90 value 16.236089
iter 90 value 16.236089
final value 16.236089
converged
# weights: 27
initial value 301.911031
iter 10 value 3.960020
iter 20 value 0.150460
iter 30 value 0.142457
iter 40 value 0.129571
iter 50 value 0.120840
iter 60 value 0.116422
iter 70 value 0.112491
iter 80 value 0.107878
iter 90 value 0.103126
iter 100 value 0.100941
final value 0.100941
stopped after 100 iterations
# weights: 43
initial value 227.264704
iter 10 value 1.894506
iter 20 value 0.278400
iter 30 value 0.229694
iter 40 value 0.218113
iter 50 value 0.202837
```

```

iter 60 value 0.169842
iter 70 value 0.145056
iter 80 value 0.130041
iter 90 value 0.117340
iter 100 value 0.113348
final value 0.113348
stopped after 100 iterations
# weights: 11
initial value 217.789924
iter 10 value 107.568130
iter 20 value 83.372721
iter 30 value 82.423869
iter 40 value 75.944952
iter 50 value 74.276994
iter 60 value 67.079689
iter 70 value 24.138285
iter 80 value 15.581603
iter 90 value 15.288036
iter 100 value 15.101049
final value 15.101049
stopped after 100 iterations
# weights: 27
initial value 231.872936
iter 10 value 1.063930
iter 20 value 0.009534
final value 0.000071
converged
# weights: 43
initial value 282.304322
iter 10 value 0.596266
iter 20 value 0.024376
iter 30 value 0.003935
iter 40 value 0.000729
iter 50 value 0.000360
final value 0.000097
converged
# weights: 11
initial value 262.036965
iter 10 value 135.128755
iter 20 value 96.011611
iter 30 value 83.710580
iter 40 value 61.805169
iter 50 value 61.165097
final value 61.165086
converged
# weights: 27

```

```
initial value 280.471704
iter 10 value 38.895782
iter 20 value 20.210588
iter 30 value 18.545939
iter 40 value 18.080891
iter 50 value 18.011420
iter 60 value 17.996892
iter 70 value 17.996714
iter 70 value 17.996714
iter 70 value 17.996714
final value 17.996714
converged
# weights: 43
initial value 227.627887
iter 10 value 29.984822
iter 20 value 15.063915
iter 30 value 14.325800
iter 40 value 14.301232
iter 50 value 14.298492
iter 60 value 14.297536
final value 14.297511
converged
# weights: 11
initial value 282.168409
iter 10 value 119.151417
iter 20 value 81.791983
iter 30 value 80.819560
iter 40 value 80.766435
iter 50 value 80.711982
iter 60 value 80.705531
iter 70 value 80.636676
iter 80 value 55.423335
iter 90 value 17.321315
iter 100 value 16.212271
final value 16.212271
stopped after 100 iterations
# weights: 27
initial value 233.549978
iter 10 value 5.503650
iter 20 value 0.296360
iter 30 value 0.205957
iter 40 value 0.196625
iter 50 value 0.169912
iter 60 value 0.122876
iter 70 value 0.116379
iter 80 value 0.108875
```

```

iter 90 value 0.101981
iter 100 value 0.095831
final value 0.095831
stopped after 100 iterations
# weights: 43
initial value 219.321891
iter 10 value 0.291591
iter 20 value 0.094470
iter 30 value 0.085850
iter 40 value 0.083564
iter 50 value 0.079461
iter 60 value 0.076153
iter 70 value 0.074398
iter 80 value 0.071525
iter 90 value 0.070799
iter 100 value 0.070195
final value 0.070195
stopped after 100 iterations
# weights: 11
initial value 221.678998
iter 10 value 56.090194
iter 20 value 26.642139
iter 30 value 23.247005
iter 40 value 23.023423
iter 50 value 22.782210
iter 60 value 22.710404
iter 70 value 22.675469
iter 80 value 22.612027
iter 90 value 22.602814
iter 100 value 22.542137
final value 22.542137
stopped after 100 iterations
# weights: 27
initial value 221.851036
iter 10 value 7.840916
iter 20 value 0.081592
iter 30 value 0.000370
final value 0.000061
converged
# weights: 43
initial value 211.003031
iter 10 value 1.988353
iter 20 value 0.022730
iter 30 value 0.000687
iter 40 value 0.000189
final value 0.000074

```

```

converged
# weights: 11
initial value 211.900426
iter 10 value 105.243528
iter 20 value 77.168000
iter 30 value 65.571592
final value 65.569280
converged
# weights: 27
initial value 224.666347
iter 10 value 28.325351
iter 20 value 20.998165
iter 30 value 18.678950
iter 40 value 18.430527
iter 50 value 18.065640
iter 60 value 18.048999
iter 70 value 18.048399
final value 18.048399
converged
# weights: 43
initial value 249.745041
iter 10 value 26.767723
iter 20 value 16.830453
iter 30 value 16.501015
iter 40 value 16.310038
iter 50 value 16.282694
iter 60 value 16.273734
final value 16.273443
converged
# weights: 11
initial value 209.004136
iter 10 value 77.736484
iter 20 value 51.176977
iter 30 value 38.866241
iter 40 value 37.202079
iter 50 value 36.081703
iter 60 value 35.837795
iter 70 value 35.785611
iter 80 value 35.757872
iter 90 value 35.756804
iter 100 value 35.753831
final value 35.753831
stopped after 100 iterations
# weights: 27
initial value 310.235667
iter 10 value 1.271073

```

```
iter 20 value 0.222841
iter 30 value 0.206771
iter 40 value 0.190740
iter 50 value 0.180238
iter 60 value 0.153049
iter 70 value 0.148109
iter 80 value 0.141571
iter 90 value 0.130380
iter 100 value 0.125632
final value 0.125632
stopped after 100 iterations
# weights: 43
initial value 230.159104
iter 10 value 3.060361
iter 20 value 0.450560
iter 30 value 0.390439
iter 40 value 0.262139
iter 50 value 0.213602
iter 60 value 0.168146
iter 70 value 0.141767
iter 80 value 0.127672
iter 90 value 0.123038
iter 100 value 0.115930
final value 0.115930
stopped after 100 iterations
# weights: 11
initial value 210.352081
iter 10 value 49.571117
iter 20 value 26.417678
iter 30 value 22.952611
iter 40 value 21.886292
iter 50 value 21.723345
iter 60 value 21.684975
iter 70 value 21.634470
iter 80 value 21.629658
iter 90 value 21.622990
iter 100 value 21.608643
final value 21.608643
stopped after 100 iterations
# weights: 27
initial value 223.699469
iter 10 value 1.397215
iter 20 value 0.002929
final value 0.000095
converged
# weights: 43
```

```
initial value 214.339015
iter 10 value 4.417938
iter 20 value 0.113785
iter 30 value 0.009311
iter 40 value 0.000342
iter 50 value 0.000167
final value 0.000073
converged
# weights: 11
initial value 237.213695
iter 10 value 74.409845
iter 20 value 64.949757
iter 30 value 64.809205
final value 64.794290
converged
# weights: 27
initial value 204.274782
iter 10 value 32.758749
iter 20 value 18.090831
iter 30 value 17.540961
iter 40 value 17.156184
iter 50 value 17.078999
final value 17.078713
converged
# weights: 43
initial value 249.904214
iter 10 value 19.617020
iter 20 value 15.577635
iter 30 value 15.536572
iter 40 value 15.532789
iter 50 value 15.532608
final value 15.532593
converged
# weights: 11
initial value 229.423526
iter 10 value 51.261515
iter 20 value 26.840030
iter 30 value 23.351669
iter 40 value 22.447504
iter 50 value 22.261779
iter 60 value 22.246295
iter 70 value 22.243154
final value 22.243145
converged
# weights: 27
initial value 223.290840
```

```
iter 10 value 2.519786
iter 20 value 0.132670
iter 30 value 0.116524
iter 40 value 0.114982
iter 50 value 0.112090
iter 60 value 0.108336
iter 70 value 0.105927
iter 80 value 0.104027
iter 90 value 0.103125
iter 100 value 0.101893
final value 0.101893
stopped after 100 iterations
# weights: 43
initial value 239.627755
iter 10 value 2.402430
iter 20 value 0.105778
iter 30 value 0.092961
iter 40 value 0.087980
iter 50 value 0.086398
iter 60 value 0.084077
iter 70 value 0.080492
iter 80 value 0.078585
iter 90 value 0.076944
iter 100 value 0.076349
final value 0.076349
stopped after 100 iterations
# weights: 11
initial value 217.276075
iter 10 value 85.908412
iter 20 value 77.341655
iter 30 value 59.302344
iter 40 value 22.471964
iter 50 value 18.026217
iter 60 value 17.414263
iter 70 value 17.123379
iter 80 value 16.879452
iter 90 value 16.493264
iter 100 value 16.374846
final value 16.374846
stopped after 100 iterations
# weights: 27
initial value 304.810828
iter 10 value 3.435410
iter 20 value 0.036172
final value 0.000093
converged
```

```
# weights: 43
initial value 233.226840
iter 10 value 0.517075
iter 20 value 0.010773
iter 30 value 0.002171
iter 40 value 0.001089
iter 50 value 0.000250
final value 0.000093
converged
# weights: 11
initial value 258.649373
iter 10 value 94.570611
iter 20 value 70.752079
iter 30 value 62.560701
final value 62.503448
converged
# weights: 27
initial value 205.249101
iter 10 value 28.921143
iter 20 value 17.556548
iter 30 value 17.054982
iter 40 value 16.898086
final value 16.897261
converged
# weights: 43
initial value 218.944757
iter 10 value 18.345750
iter 20 value 15.556666
iter 30 value 15.394932
iter 40 value 15.389670
iter 50 value 15.389558
final value 15.389547
converged
# weights: 11
initial value 242.373601
iter 10 value 89.802834
iter 20 value 80.403333
iter 30 value 73.689318
iter 40 value 56.967895
iter 50 value 24.120234
iter 60 value 17.647198
iter 70 value 17.490741
iter 80 value 17.254545
iter 90 value 17.234039
iter 100 value 17.178210
final value 17.178210
```

```
stopped after 100 iterations
# weights: 27
initial value 221.890722
iter 10 value 2.148259
iter 20 value 0.145115
iter 30 value 0.118547
iter 40 value 0.111730
iter 50 value 0.104494
iter 60 value 0.098543
iter 70 value 0.094892
iter 80 value 0.090898
iter 90 value 0.084648
iter 100 value 0.082590
final value 0.082590
stopped after 100 iterations
# weights: 43
initial value 242.997048
iter 10 value 3.746020
iter 20 value 0.119886
iter 30 value 0.108726
iter 40 value 0.105296
iter 50 value 0.102118
iter 60 value 0.100959
iter 70 value 0.099396
iter 80 value 0.096071
iter 90 value 0.091773
iter 100 value 0.088721
final value 0.088721
stopped after 100 iterations
# weights: 11
initial value 245.744125
iter 10 value 75.548284
iter 20 value 22.322838
iter 30 value 18.509874
iter 40 value 18.206074
iter 50 value 18.103339
iter 60 value 18.064064
iter 70 value 17.889990
iter 80 value 17.881092
iter 90 value 17.850085
iter 100 value 17.794473
final value 17.794473
stopped after 100 iterations
# weights: 27
initial value 236.900220
iter 10 value 6.958407
```

```
iter 20 value 0.196550
iter 30 value 0.002052
iter 40 value 0.000335
final value 0.000095
converged
# weights: 43
initial value 277.732546
iter 10 value 3.030945
iter 20 value 0.023442
iter 30 value 0.001179
iter 40 value 0.000254
final value 0.000075
converged
# weights: 11
initial value 250.424870
iter 10 value 129.422387
iter 20 value 85.135311
iter 30 value 64.525471
iter 40 value 63.076904
final value 63.076900
converged
# weights: 27
initial value 259.061248
iter 10 value 36.251595
iter 20 value 17.593923
iter 30 value 17.192269
iter 40 value 17.044420
iter 50 value 17.043593
final value 17.043577
converged
# weights: 43
initial value 249.532847
iter 10 value 18.245931
iter 20 value 15.820914
iter 30 value 15.334717
iter 40 value 15.296595
iter 50 value 15.289711
iter 60 value 15.288658
final value 15.288658
converged
# weights: 11
initial value 225.082711
iter 10 value 82.752038
iter 20 value 79.832120
iter 30 value 77.512476
iter 40 value 74.323792
```

```

iter 50 value 71.181123
iter 60 value 69.845353
iter 70 value 30.417858
iter 80 value 20.580843
iter 90 value 19.587078
iter 100 value 19.481255
final value 19.481255
stopped after 100 iterations
# weights: 27
initial value 210.971444
iter 10 value 24.621701
iter 20 value 0.402230
iter 30 value 0.311407
iter 40 value 0.252267
iter 50 value 0.212828
iter 60 value 0.173815
iter 70 value 0.162926
iter 80 value 0.151362
iter 90 value 0.139867
iter 100 value 0.126160
final value 0.126160
stopped after 100 iterations
# weights: 43
initial value 244.405516
iter 10 value 1.269312
iter 20 value 0.155203
iter 30 value 0.144605
iter 40 value 0.135526
iter 50 value 0.122868
iter 60 value 0.111566
iter 70 value 0.103111
iter 80 value 0.100674
iter 90 value 0.099761
iter 100 value 0.097428
final value 0.097428
stopped after 100 iterations
# weights: 11
initial value 258.355161
iter 10 value 89.697180
iter 20 value 77.104486
iter 30 value 74.947698
iter 40 value 74.761515
iter 50 value 74.707141
iter 60 value 74.703681
iter 70 value 74.703178
iter 70 value 74.703178

```

```
final value 74.703178
converged
# weights: 27
initial value 219.689103
iter 10 value 4.440304
iter 20 value 0.036887
iter 30 value 0.001295
iter 40 value 0.000361
final value 0.000083
converged
# weights: 43
initial value 218.428917
iter 10 value 1.415901
iter 20 value 0.033571
iter 30 value 0.001008
final value 0.000074
converged
# weights: 11
initial value 214.875293
iter 10 value 97.467945
iter 20 value 87.013937
iter 30 value 84.279118
iter 40 value 84.133130
final value 84.133111
converged
# weights: 27
initial value 193.531561
iter 10 value 35.782911
iter 20 value 21.043604
iter 30 value 19.826094
iter 40 value 19.582609
iter 50 value 19.580476
final value 19.580468
converged
# weights: 43
initial value 210.362029
iter 10 value 21.246895
iter 20 value 16.903922
iter 30 value 16.415378
iter 40 value 16.298598
iter 50 value 16.288985
iter 60 value 16.288417
iter 70 value 16.288400
final value 16.288400
converged
# weights: 11
```

```
initial value 247.055044
iter 10 value 85.807399
iter 20 value 74.847190
iter 30 value 73.457112
iter 40 value 73.179884
iter 50 value 73.172477
iter 60 value 73.167231
iter 70 value 73.145693
iter 80 value 73.137904
iter 90 value 73.137491
iter 100 value 73.136740
final value 73.136740
stopped after 100 iterations
# weights: 27
initial value 208.550512
iter 10 value 5.177019
iter 20 value 0.205752
iter 30 value 0.172538
iter 40 value 0.159985
iter 50 value 0.147746
iter 60 value 0.135370
iter 70 value 0.128532
iter 80 value 0.119162
iter 90 value 0.110322
iter 100 value 0.106657
final value 0.106657
stopped after 100 iterations
# weights: 43
initial value 220.331085
iter 10 value 4.582290
iter 20 value 0.278382
iter 30 value 0.231740
iter 40 value 0.220487
iter 50 value 0.198025
iter 60 value 0.175148
iter 70 value 0.146942
iter 80 value 0.127147
iter 90 value 0.121226
iter 100 value 0.117215
final value 0.117215
stopped after 100 iterations
# weights: 11
initial value 230.514113
iter 10 value 81.665439
iter 20 value 74.635212
iter 30 value 73.622077
```

```

iter 40 value 73.619400
iter 50 value 73.616123
iter 60 value 73.615115
iter 70 value 73.608651
iter 80 value 73.606454
iter 90 value 73.603256
iter 100 value 73.603127
final value 73.603127
stopped after 100 iterations
# weights: 27
initial value 248.829339
iter 10 value 2.031858
iter 20 value 0.007969
iter 30 value 0.000240
final value 0.000097
converged
# weights: 43
initial value 253.535170
iter 10 value 3.832361
iter 20 value 0.025272
iter 30 value 0.003171
iter 40 value 0.000992
iter 50 value 0.000133
iter 50 value 0.000098
iter 50 value 0.000098
final value 0.000098
converged
# weights: 11
initial value 216.881559
iter 10 value 96.575025
iter 20 value 69.803815
iter 30 value 65.548721
final value 65.502490
converged
# weights: 27
initial value 217.987334
iter 10 value 29.888190
iter 20 value 19.679331
iter 30 value 17.746779
iter 40 value 17.254178
iter 50 value 17.253893
final value 17.253893
converged
# weights: 43
initial value 229.490638
iter 10 value 23.533499

```

```
iter 20 value 16.388819
iter 30 value 15.750620
iter 40 value 15.476653
iter 50 value 15.374286
iter 60 value 15.370509
iter 70 value 15.370429
iter 80 value 15.370375
iter 80 value 15.370375
iter 80 value 15.370375
final value 15.370375
converged
# weights: 11
initial value 194.441743
iter 10 value 56.062561
iter 20 value 27.083371
iter 30 value 23.796885
iter 40 value 22.679268
iter 50 value 22.291526
iter 60 value 22.281938
iter 70 value 22.268035
final value 22.268034
converged
# weights: 27
initial value 278.946390
iter 10 value 7.366095
iter 20 value 0.189454
iter 30 value 0.150160
iter 40 value 0.141775
iter 50 value 0.131569
iter 60 value 0.126347
iter 70 value 0.120698
iter 80 value 0.117627
iter 90 value 0.109629
iter 100 value 0.106125
final value 0.106125
stopped after 100 iterations
# weights: 43
initial value 251.001362
iter 10 value 1.117408
iter 20 value 0.147426
iter 30 value 0.108336
iter 40 value 0.102625
iter 50 value 0.101021
iter 60 value 0.099310
iter 70 value 0.097876
iter 80 value 0.096942
```

```
iter 90 value 0.096205
iter 100 value 0.095661
final value 0.095661
stopped after 100 iterations
# weights: 11
initial value 225.195654
iter 10 value 90.288772
iter 20 value 83.788101
iter 30 value 83.627853
iter 40 value 83.372817
iter 50 value 83.227754
iter 60 value 83.168310
iter 70 value 83.027387
iter 80 value 83.017533
iter 90 value 82.940159
iter 100 value 82.926612
final value 82.926612
stopped after 100 iterations
# weights: 27
initial value 307.668185
iter 10 value 0.893932
iter 20 value 0.041661
iter 30 value 0.000689
iter 40 value 0.000145
final value 0.000084
converged
# weights: 43
initial value 247.513266
iter 10 value 0.510117
iter 20 value 0.013442
iter 30 value 0.000982
iter 40 value 0.000639
final value 0.000079
converged
# weights: 11
initial value 234.847188
iter 10 value 80.808829
iter 20 value 62.187207
final value 62.184492
converged
# weights: 27
initial value 229.664722
iter 10 value 24.998722
iter 20 value 17.664215
iter 30 value 17.095076
iter 40 value 16.523913
```

```
iter 50 value 16.386594
final value 16.386578
converged
# weights: 43
initial value 228.651413
iter 10 value 19.881946
iter 20 value 15.450663
iter 30 value 14.728572
iter 40 value 14.697831
iter 50 value 14.692928
final value 14.692922
converged
# weights: 11
initial value 216.933719
iter 10 value 104.617626
iter 20 value 87.223706
iter 30 value 87.068568
iter 40 value 87.026051
iter 50 value 77.808435
iter 60 value 68.825017
iter 70 value 32.837163
iter 80 value 18.252663
iter 90 value 17.567039
iter 100 value 17.483336
final value 17.483336
stopped after 100 iterations
# weights: 27
initial value 223.572254
iter 10 value 1.905625
iter 20 value 0.187480
iter 30 value 0.172496
iter 40 value 0.146099
iter 50 value 0.126068
iter 60 value 0.118989
iter 70 value 0.113464
iter 80 value 0.103479
iter 90 value 0.091657
iter 100 value 0.085960
final value 0.085960
stopped after 100 iterations
# weights: 43
initial value 256.255989
iter 10 value 0.881593
iter 20 value 0.439344
iter 30 value 0.406279
iter 40 value 0.320938
```

```

iter 50 value 0.245650
iter 60 value 0.215475
iter 70 value 0.195554
iter 80 value 0.090438
iter 90 value 0.081291
iter 100 value 0.076491
final value 0.076491
stopped after 100 iterations
# weights: 11
initial value 210.031761
iter 10 value 81.080259
iter 20 value 79.542465
iter 30 value 79.501934
iter 40 value 79.499571
iter 50 value 79.499063
final value 79.499057
converged
# weights: 27
initial value 221.401497
iter 10 value 6.694013
iter 20 value 0.054815
iter 30 value 0.001166
final value 0.000065
converged
# weights: 43
initial value 242.654613
iter 10 value 5.400023
iter 20 value 0.013261
final value 0.000099
converged
# weights: 11
initial value 236.079084
iter 10 value 90.909124
iter 20 value 64.821462
iter 30 value 62.704046
final value 62.682170
converged
# weights: 27
initial value 223.511271
iter 10 value 82.578270
iter 20 value 27.403902
iter 30 value 18.984888
iter 40 value 18.104617
iter 50 value 18.069607
iter 60 value 18.005023
iter 70 value 17.964759

```

```

final  value 17.964751
converged
# weights: 43
initial  value 250.517403
iter   10 value 42.129141
iter   20 value 18.646580
iter   30 value 16.997815
iter   40 value 16.569006
iter   50 value 16.516536
iter   60 value 16.512338
iter   70 value 16.507498
iter   80 value 16.507146
final  value 16.507130
converged
# weights: 11
initial  value 221.223756
iter   10 value 81.721973
iter   20 value 80.874678
iter   30 value 80.827387
iter   40 value 80.770698
iter   50 value 80.716172
iter   60 value 80.710507
iter   70 value 80.707652
iter   80 value 80.704079
iter   90 value 80.698259
iter  100 value 80.688398
final  value 80.688398
stopped after 100 iterations
# weights: 27
initial  value 214.668428
iter   10 value 3.364332
iter   20 value 0.246780
iter   30 value 0.226277
iter   40 value 0.211206
iter   50 value 0.200967
iter   60 value 0.169374
iter   70 value 0.160833
iter   80 value 0.149599
iter   90 value 0.140525
iter  100 value 0.130978
final  value 0.130978
stopped after 100 iterations
# weights: 43
initial  value 240.046147
iter   10 value 5.357144
iter   20 value 0.295068

```

```

iter 30 value 0.245344
iter 40 value 0.229571
iter 50 value 0.182941
iter 60 value 0.147340
iter 70 value 0.118639
iter 80 value 0.105157
iter 90 value 0.100273
iter 100 value 0.094674
final value 0.094674
stopped after 100 iterations
# weights: 11
initial value 241.988589
iter 10 value 133.992504
iter 20 value 115.443995
iter 30 value 86.920497
iter 40 value 44.729887
iter 50 value 36.614522
iter 60 value 35.994188
iter 70 value 35.851790
iter 80 value 35.809122
iter 90 value 35.791205
iter 100 value 35.758886
final value 35.758886
stopped after 100 iterations
# weights: 27
initial value 234.412874
iter 10 value 11.506486
iter 20 value 0.098389
iter 30 value 0.002607
final value 0.000076
converged
# weights: 43
initial value 336.906525
iter 10 value 1.423307
iter 20 value 0.035188
iter 30 value 0.004302
iter 40 value 0.002668
iter 50 value 0.000801
final value 0.000074
converged
# weights: 11
initial value 233.856705
iter 10 value 80.768402
iter 20 value 60.719689
iter 30 value 60.192224
final value 60.175927

```

```

converged
# weights: 27
initial value 216.448865
iter 10 value 25.240192
iter 20 value 17.450245
iter 30 value 16.399926
iter 40 value 16.365982
iter 50 value 16.360951
final value 16.360939
converged
# weights: 43
initial value 216.687515
iter 10 value 26.613546
iter 20 value 15.486131
iter 30 value 14.682097
iter 40 value 14.454402
iter 50 value 14.447770
iter 60 value 14.445716
final value 14.445701
converged
# weights: 11
initial value 238.948550
iter 10 value 81.867184
iter 20 value 79.833876
iter 30 value 78.583120
iter 40 value 76.116917
iter 50 value 56.616363
iter 60 value 36.827824
iter 70 value 36.413541
iter 80 value 36.159924
iter 90 value 36.125009
iter 100 value 36.121421
final value 36.121421
stopped after 100 iterations
# weights: 27
initial value 246.087985
iter 10 value 1.025070
iter 20 value 0.128164
iter 30 value 0.113024
iter 40 value 0.110504
iter 50 value 0.106389
iter 60 value 0.096600
iter 70 value 0.094253
iter 80 value 0.087041
iter 90 value 0.082852
iter 100 value 0.078974

```

```

final value 0.078974
stopped after 100 iterations
# weights: 43
initial value 243.915343
iter 10 value 2.308889
iter 20 value 0.105254
iter 30 value 0.090986
iter 40 value 0.086986
iter 50 value 0.085076
iter 60 value 0.083281
iter 70 value 0.081868
iter 80 value 0.080348
iter 90 value 0.078237
iter 100 value 0.076222
final value 0.076222
stopped after 100 iterations
# weights: 11
initial value 261.570226
iter 10 value 84.276957
iter 20 value 80.657746
final value 80.653972
converged
# weights: 27
initial value 246.114813
iter 10 value 37.335079
iter 20 value 5.372291
iter 30 value 1.871398
iter 40 value 0.984589
iter 50 value 0.013859
final value 0.000078
converged
# weights: 43
initial value 217.423848
iter 10 value 1.797230
iter 20 value 0.006384
final value 0.000096
converged
# weights: 11
initial value 251.029096
iter 10 value 85.981965
iter 20 value 63.266992
iter 30 value 63.106025
final value 63.105780
converged
# weights: 27
initial value 233.705626

```

```

iter 10 value 25.953084
iter 20 value 19.970191
iter 30 value 19.690991
iter 40 value 19.674356
final value 19.673496
converged
# weights: 43
initial value 275.647992
iter 10 value 24.432065
iter 20 value 16.368954
iter 30 value 15.873171
iter 40 value 15.654284
iter 50 value 15.641368
iter 60 value 15.637737
iter 60 value 15.637737
iter 60 value 15.637737
final value 15.637737
converged
# weights: 11
initial value 203.650194
iter 10 value 71.270854
iter 20 value 30.136470
iter 30 value 22.812841
iter 40 value 21.169490
iter 50 value 20.979261
iter 60 value 20.894523
iter 70 value 20.878622
iter 80 value 20.878348
iter 90 value 20.877851
final value 20.877822
converged
# weights: 27
initial value 221.427892
iter 10 value 7.358508
iter 20 value 0.272707
iter 30 value 0.216000
iter 40 value 0.207513
iter 50 value 0.199634
iter 60 value 0.188913
iter 70 value 0.178006
iter 80 value 0.158043
iter 90 value 0.142984
iter 100 value 0.131296
final value 0.131296
stopped after 100 iterations
# weights: 43

```

```

initial value 284.968641
iter 10 value 3.257012
iter 20 value 0.263174
iter 30 value 0.211285
iter 40 value 0.190454
iter 50 value 0.167613
iter 60 value 0.140386
iter 70 value 0.123102
iter 80 value 0.112733
iter 90 value 0.099405
iter 100 value 0.094790
final value 0.094790
stopped after 100 iterations
# weights: 11
initial value 229.555393
iter 10 value 91.333216
iter 20 value 79.545089
iter 30 value 78.751034
iter 40 value 77.005712
iter 50 value 75.012277
iter 60 value 60.000004
iter 70 value 40.021816
iter 80 value 38.861966
iter 90 value 38.720190
iter 100 value 38.657836
final value 38.657836
stopped after 100 iterations
# weights: 27
initial value 293.311660
iter 10 value 3.929897
iter 20 value 0.012122
iter 30 value 0.000487
final value 0.000095
converged
# weights: 43
initial value 223.987133
iter 10 value 1.413535
iter 20 value 0.008366
iter 30 value 0.001449
final value 0.000086
converged
# weights: 11
initial value 247.452991
iter 10 value 84.167591
iter 20 value 64.440139
iter 30 value 64.154260

```

```
final  value 64.153537
converged
# weights: 27
initial  value 252.750242
iter   10 value 43.889175
iter   20 value 19.740615
iter   30 value 17.884595
iter   40 value 17.777644
iter   50 value 17.775613
iter   60 value 17.775266
final  value 17.775239
converged
# weights: 43
initial  value 258.242583
iter   10 value 28.177719
iter   20 value 16.291883
iter   30 value 16.008718
iter   40 value 15.999155
iter   50 value 15.998217
final  value 15.998206
converged
# weights: 11
initial  value 223.237946
iter   10 value 42.122375
iter   20 value 19.056114
iter   30 value 16.323988
iter   40 value 15.623997
iter   50 value 15.609316
iter   60 value 15.596894
iter   70 value 15.523638
iter   80 value 15.523500
iter   90 value 15.523250
final  value 15.523243
converged
# weights: 27
initial  value 245.408335
iter   10 value 1.726631
iter   20 value 0.146646
iter   30 value 0.104571
iter   40 value 0.098270
iter   50 value 0.095648
iter   60 value 0.095122
iter   70 value 0.094440
iter   80 value 0.093816
iter   90 value 0.092868
iter  100 value 0.092268
```

```

final value 0.092268
stopped after 100 iterations
# weights: 43
initial value 257.063790
iter 10 value 1.036936
iter 20 value 0.144450
iter 30 value 0.114876
iter 40 value 0.110638
iter 50 value 0.108639
iter 60 value 0.107746
iter 70 value 0.105969
iter 80 value 0.102993
iter 90 value 0.098163
iter 100 value 0.096189
final value 0.096189
stopped after 100 iterations
# weights: 11
initial value 222.759043
iter 10 value 81.126203
iter 20 value 37.539134
iter 30 value 19.409619
iter 40 value 18.059617
iter 50 value 17.199620
iter 60 value 16.297190
iter 70 value 16.181804
iter 80 value 16.024015
iter 90 value 15.966491
iter 100 value 15.838594
final value 15.838594
stopped after 100 iterations
# weights: 27
initial value 247.308580
iter 10 value 3.272364
iter 20 value 0.079388
iter 30 value 0.007491
iter 40 value 0.001036
iter 50 value 0.000222
final value 0.000063
converged
# weights: 43
initial value 214.625127
iter 10 value 1.203864
iter 20 value 0.045668
iter 30 value 0.010902
iter 40 value 0.003027
iter 50 value 0.001450

```

```
iter 60 value 0.000229
final value 0.000089
converged
# weights: 11
initial value 212.530721
iter 10 value 94.121034
iter 20 value 84.744592
iter 30 value 66.239500
iter 40 value 63.211759
final value 63.211757
converged
# weights: 27
initial value 220.804296
iter 10 value 40.018410
iter 20 value 17.040450
iter 30 value 16.614790
iter 40 value 16.438298
iter 50 value 16.430589
final value 16.430589
converged
# weights: 43
initial value 213.719745
iter 10 value 31.718389
iter 20 value 15.486316
iter 30 value 15.135156
iter 40 value 15.057395
iter 50 value 14.975686
iter 60 value 14.966760
final value 14.966745
converged
# weights: 11
initial value 209.053755
iter 10 value 80.495928
iter 20 value 79.587330
iter 30 value 79.216706
iter 40 value 70.864824
iter 50 value 57.706706
iter 60 value 25.151675
iter 70 value 17.854095
iter 80 value 17.576339
iter 90 value 16.844249
iter 100 value 16.801623
final value 16.801623
stopped after 100 iterations
# weights: 27
initial value 229.084672
```

```

iter 10 value 0.830262
iter 20 value 0.135855
iter 30 value 0.122118
iter 40 value 0.120475
iter 50 value 0.117478
iter 60 value 0.114704
iter 70 value 0.113298
iter 80 value 0.110798
iter 90 value 0.109033
iter 100 value 0.106221
final value 0.106221
stopped after 100 iterations
# weights: 43
initial value 223.893141
iter 10 value 5.065483
iter 20 value 0.338310
iter 30 value 0.301414
iter 40 value 0.283589
iter 50 value 0.229402
iter 60 value 0.204131
iter 70 value 0.139085
iter 80 value 0.112159
iter 90 value 0.093284
iter 100 value 0.087892
final value 0.087892
stopped after 100 iterations
# weights: 11
initial value 253.381877
iter 10 value 89.269514
iter 20 value 81.171090
iter 30 value 72.607610
iter 40 value 41.649483
iter 50 value 24.282269
iter 60 value 22.732404
iter 70 value 22.344084
iter 80 value 21.999707
iter 90 value 21.975479
iter 100 value 21.967109
final value 21.967109
stopped after 100 iterations
# weights: 27
initial value 218.335769
iter 10 value 2.596993
iter 20 value 0.028791
final value 0.000092
converged

```

```

# weights: 43
initial value 253.559474
iter 10 value 2.305882
iter 20 value 0.140979
iter 30 value 0.006124
iter 40 value 0.001513
iter 50 value 0.000645
iter 60 value 0.000415
final value 0.000096
converged
# weights: 11
initial value 244.808759
iter 10 value 130.057982
iter 20 value 87.812483
iter 30 value 68.373247
iter 40 value 65.281281
final value 65.281278
converged
# weights: 27
initial value 220.126703
iter 10 value 26.238145
iter 20 value 19.446177
iter 30 value 19.269663
iter 40 value 19.268221
iter 40 value 19.268221
iter 40 value 19.268221
final value 19.268221
converged
# weights: 43
initial value 221.664639
iter 10 value 24.955243
iter 20 value 15.964582
iter 30 value 15.914914
iter 40 value 15.897674
iter 50 value 15.849672
iter 60 value 15.839764
iter 70 value 15.837163
final value 15.837152
converged
# weights: 11
initial value 241.733453
iter 10 value 74.691498
iter 20 value 30.011315
iter 30 value 23.753901
iter 40 value 22.769455
iter 50 value 22.743139

```

```

iter 60 value 22.715886
iter 70 value 22.642900
iter 80 value 22.642259
iter 90 value 22.639636
final value 22.639424
converged
# weights: 27
initial value 233.091370
iter 10 value 3.831841
iter 20 value 0.208231
iter 30 value 0.169229
iter 40 value 0.161463
iter 50 value 0.141751
iter 60 value 0.134860
iter 70 value 0.129991
iter 80 value 0.124396
iter 90 value 0.117443
iter 100 value 0.110108
final value 0.110108
stopped after 100 iterations
# weights: 43
initial value 223.939659
iter 10 value 2.043421
iter 20 value 0.141216
iter 30 value 0.129944
iter 40 value 0.126262
iter 50 value 0.118069
iter 60 value 0.113824
iter 70 value 0.108667
iter 80 value 0.106276
iter 90 value 0.104930
iter 100 value 0.103048
final value 0.103048
stopped after 100 iterations
# weights: 11
initial value 274.907456
iter 10 value 107.469947
iter 20 value 26.626462
iter 30 value 18.963116
iter 40 value 18.415235
iter 50 value 18.239549
iter 60 value 18.208809
iter 70 value 18.035689
iter 80 value 18.011575
iter 90 value 18.005652
iter 100 value 17.992251

```

```
final value 17.992251
stopped after 100 iterations
# weights: 27
initial value 273.574219
iter 10 value 3.945598
iter 20 value 0.056424
final value 0.000067
converged
# weights: 43
initial value 224.551423
iter 10 value 1.754234
iter 20 value 0.035346
iter 30 value 0.000843
final value 0.000092
converged
# weights: 11
initial value 278.348726
iter 10 value 90.767033
iter 20 value 64.359153
iter 30 value 64.319682
final value 64.317368
converged
# weights: 27
initial value 245.953170
iter 10 value 25.307289
iter 20 value 18.880764
iter 30 value 18.811306
iter 40 value 18.809590
iter 50 value 18.809161
iter 60 value 18.669113
final value 18.669003
converged
# weights: 43
initial value 201.540471
iter 10 value 33.892810
iter 20 value 15.777146
iter 30 value 15.430553
iter 40 value 15.397359
iter 50 value 15.372305
iter 60 value 15.343790
final value 15.343590
converged
# weights: 11
initial value 216.922431
iter 10 value 73.363394
iter 20 value 30.265109
```

```

iter 30 value 18.990485
iter 40 value 18.779331
iter 50 value 18.622276
iter 60 value 18.618927
iter 70 value 18.607429
iter 80 value 18.591123
iter 90 value 18.590959
iter 100 value 18.590197
final value 18.590197
stopped after 100 iterations
# weights: 27
initial value 201.399942
iter 10 value 3.189387
iter 20 value 0.141838
iter 30 value 0.126597
iter 40 value 0.120741
iter 50 value 0.119572
iter 60 value 0.116097
iter 70 value 0.111127
iter 80 value 0.105404
iter 90 value 0.101677
iter 100 value 0.096324
final value 0.096324
stopped after 100 iterations
# weights: 43
initial value 238.099129
iter 10 value 2.667195
iter 20 value 0.195653
iter 30 value 0.133575
iter 40 value 0.129713
iter 50 value 0.126826
iter 60 value 0.117326
iter 70 value 0.110273
iter 80 value 0.106860
iter 90 value 0.104504
iter 100 value 0.101985
final value 0.101985
stopped after 100 iterations
# weights: 11
initial value 280.203858
iter 10 value 80.409542
iter 20 value 79.499971
final value 79.499108
converged
# weights: 27
initial value 265.518273

```

```

iter 10 value 10.270302
iter 20 value 0.186642
iter 30 value 0.007363
iter 40 value 0.000595
iter 50 value 0.000150
final value 0.000081
converged
# weights: 43
initial value 254.452114
iter 10 value 1.942683
iter 20 value 0.023160
iter 30 value 0.001134
iter 40 value 0.000202
final value 0.000061
converged
# weights: 11
initial value 217.591191
iter 10 value 93.579287
iter 20 value 69.533903
iter 30 value 64.312878
final value 64.097506
converged
# weights: 27
initial value 240.881618
iter 10 value 39.407196
iter 20 value 18.978187
iter 30 value 17.695892
iter 40 value 17.669696
iter 50 value 17.669217
iter 50 value 17.669217
iter 50 value 17.669217
final value 17.669217
converged
# weights: 43
initial value 284.684444
iter 10 value 24.188663
iter 20 value 17.252075
iter 30 value 16.284421
iter 40 value 16.072756
iter 50 value 16.006303
iter 60 value 16.003356
final value 16.003093
converged
# weights: 11
initial value 215.444098
iter 10 value 85.524276

```

```
iter 20 value 78.804157
iter 30 value 42.559303
iter 40 value 21.365022
iter 50 value 20.632331
iter 60 value 20.238511
iter 70 value 20.134970
iter 80 value 20.121548
iter 90 value 20.098215
iter 100 value 20.098202
final value 20.098202
stopped after 100 iterations
# weights: 27
initial value 241.527656
iter 10 value 3.308476
iter 20 value 0.145195
iter 30 value 0.136416
iter 40 value 0.131268
iter 50 value 0.128244
iter 60 value 0.122941
iter 70 value 0.119027
iter 80 value 0.115152
iter 90 value 0.110758
iter 100 value 0.109334
final value 0.109334
stopped after 100 iterations
# weights: 43
initial value 284.849848
iter 10 value 1.968594
iter 20 value 0.256724
iter 30 value 0.218537
iter 40 value 0.188306
iter 50 value 0.169105
iter 60 value 0.137594
iter 70 value 0.114581
iter 80 value 0.107317
iter 90 value 0.103245
iter 100 value 0.101490
final value 0.101490
stopped after 100 iterations
# weights: 11
initial value 233.214733
iter 10 value 71.013322
iter 20 value 24.125877
iter 30 value 20.939866
iter 40 value 20.229111
iter 50 value 20.067124
```

```
iter 60 value 19.967327
iter 70 value 19.730349
iter 80 value 19.722875
iter 90 value 19.652933
iter 100 value 19.576833
final value 19.576833
stopped after 100 iterations
# weights: 27
initial value 217.906832
iter 10 value 2.883643
iter 20 value 0.049437
iter 30 value 0.004698
iter 40 value 0.000104
final value 0.000096
converged
# weights: 43
initial value 220.718137
iter 10 value 1.437157
iter 20 value 0.016499
iter 30 value 0.004416
iter 40 value 0.002200
iter 50 value 0.000231
final value 0.000096
converged
# weights: 11
initial value 258.660086
iter 10 value 118.325717
iter 20 value 88.479098
iter 30 value 68.143316
iter 40 value 63.333147
final value 63.333132
converged
# weights: 27
initial value 213.531680
iter 10 value 28.738641
iter 20 value 18.481998
iter 30 value 17.485193
iter 40 value 17.226771
iter 50 value 17.193047
iter 60 value 17.191610
final value 17.191588
converged
# weights: 43
initial value 236.255716
iter 10 value 25.007483
iter 20 value 16.681534
```

```
iter 30 value 16.064339
iter 40 value 15.796383
iter 50 value 15.700762
iter 60 value 15.691308
final value 15.691289
converged
# weights: 11
initial value 221.613419
iter 10 value 51.862225
iter 20 value 26.141819
iter 30 value 21.121062
iter 40 value 20.606404
iter 50 value 20.486224
iter 60 value 20.356223
iter 70 value 20.355958
iter 80 value 20.355216
iter 80 value 20.355216
iter 80 value 20.355216
final value 20.355216
converged
# weights: 27
initial value 253.204365
iter 10 value 6.482858
iter 20 value 0.693011
iter 30 value 0.566939
iter 40 value 0.497394
iter 50 value 0.354532
iter 60 value 0.259044
iter 70 value 0.240190
iter 80 value 0.224891
iter 90 value 0.196398
iter 100 value 0.130435
final value 0.130435
stopped after 100 iterations
# weights: 43
initial value 261.408583
iter 10 value 1.960634
iter 20 value 0.165774
iter 30 value 0.117747
iter 40 value 0.113366
iter 50 value 0.107150
iter 60 value 0.100075
iter 70 value 0.097270
iter 80 value 0.091818
iter 90 value 0.087102
iter 100 value 0.085562
```

```

final value 0.085562
stopped after 100 iterations
# weights: 11
initial value 216.993416
iter 10 value 74.058799
iter 20 value 24.645561
iter 30 value 20.261241
iter 40 value 18.790763
iter 50 value 18.601997
iter 60 value 18.412740
iter 70 value 18.368980
iter 80 value 18.343392
iter 90 value 18.291727
iter 100 value 18.255879
final value 18.255879
stopped after 100 iterations
# weights: 27
initial value 274.600590
iter 10 value 4.820199
iter 20 value 0.176990
iter 30 value 0.000995
iter 40 value 0.000341
final value 0.000084
converged
# weights: 43
initial value 238.512316
iter 10 value 5.075000
iter 20 value 0.038651
iter 30 value 0.007426
iter 40 value 0.000500
iter 50 value 0.000304
final value 0.000083
converged
# weights: 11
initial value 287.515000
iter 10 value 105.215521
iter 20 value 92.499925
iter 30 value 67.761218
iter 40 value 62.055870
iter 40 value 62.055870
iter 40 value 62.055870
final value 62.055870
converged
# weights: 27
initial value 288.260215
iter 10 value 29.601191

```

```

iter 20 value 19.773711
iter 30 value 19.299200
iter 40 value 19.255156
iter 50 value 19.253313
final value 19.253292
converged
# weights: 43
initial value 229.032455
iter 10 value 22.768878
iter 20 value 15.639673
iter 30 value 15.351771
iter 40 value 15.348717
iter 50 value 15.347373
final value 15.347274
converged
# weights: 11
initial value 250.849745
iter 10 value 80.778099
iter 20 value 80.618499
iter 30 value 75.078332
iter 40 value 59.361788
iter 50 value 24.036818
iter 60 value 20.318232
iter 70 value 19.115281
iter 80 value 18.895820
iter 90 value 18.872682
final value 18.867334
converged
# weights: 27
initial value 234.875862
iter 10 value 7.717981
iter 20 value 0.418179
iter 30 value 0.254177
iter 40 value 0.228120
iter 50 value 0.175117
iter 60 value 0.164327
iter 70 value 0.149230
iter 80 value 0.138264
iter 90 value 0.112062
iter 100 value 0.106170
final value 0.106170
stopped after 100 iterations
# weights: 43
initial value 215.938037
iter 10 value 4.454928
iter 20 value 0.891339

```

```

iter 30 value 0.366964
iter 40 value 0.313869
iter 50 value 0.191627
iter 60 value 0.158783
iter 70 value 0.145330
iter 80 value 0.118517
iter 90 value 0.110654
iter 100 value 0.106633
final value 0.106633
stopped after 100 iterations
# weights: 11
initial value 236.065210
iter 10 value 55.248193
iter 20 value 20.610232
iter 30 value 10.161425
iter 40 value 8.414410
iter 50 value 8.295063
iter 60 value 8.171692
iter 70 value 7.508997
iter 80 value 7.498254
iter 90 value 7.478688
iter 100 value 7.467027
final value 7.467027
stopped after 100 iterations
# weights: 27
initial value 298.325546
iter 10 value 10.607498
iter 20 value 0.024029
iter 30 value 0.000544
final value 0.000086
converged
# weights: 43
initial value 271.261193
iter 10 value 0.879566
iter 20 value 0.005458
iter 30 value 0.000224
final value 0.000099
converged
# weights: 11
initial value 270.313428
iter 10 value 111.109778
iter 20 value 68.702732
iter 30 value 62.408029
final value 62.008539
converged
# weights: 27

```

```
initial value 215.220190
iter 10 value 35.229383
iter 20 value 17.962387
iter 30 value 17.256856
iter 40 value 16.959117
iter 50 value 16.863907
iter 60 value 16.841038
final value 16.841015
converged
# weights: 43
initial value 269.067336
iter 10 value 35.440920
iter 20 value 15.846323
iter 30 value 15.367559
iter 40 value 15.179223
iter 50 value 15.133214
iter 60 value 15.129570
iter 70 value 15.129392
iter 70 value 15.129392
iter 70 value 15.129392
final value 15.129392
converged
# weights: 11
initial value 225.083061
iter 10 value 88.051131
iter 20 value 22.328188
iter 30 value 11.963422
iter 40 value 9.817619
iter 50 value 9.499781
iter 60 value 9.319954
iter 70 value 9.228925
iter 80 value 9.215325
iter 90 value 9.213282
iter 100 value 9.209750
final value 9.209750
stopped after 100 iterations
# weights: 27
initial value 228.330862
iter 10 value 0.494885
iter 20 value 0.101393
iter 30 value 0.098863
iter 40 value 0.098098
iter 50 value 0.094883
iter 60 value 0.089808
iter 70 value 0.087944
iter 80 value 0.080752
```

```

iter 90 value 0.079397
iter 100 value 0.078871
final value 0.078871
stopped after 100 iterations
# weights: 43
initial value 246.581022
iter 10 value 8.045050
iter 20 value 0.158426
iter 30 value 0.108424
iter 40 value 0.105165
iter 50 value 0.102948
iter 60 value 0.097604
iter 70 value 0.089164
iter 80 value 0.083715
iter 90 value 0.083187
iter 100 value 0.081591
final value 0.081591
stopped after 100 iterations
# weights: 11
initial value 227.766248
iter 10 value 80.689498
iter 20 value 65.678949
iter 30 value 37.000824
iter 40 value 20.048454
iter 50 value 19.164231
iter 60 value 18.833448
iter 70 value 18.489616
iter 80 value 18.436293
iter 90 value 18.295387
iter 100 value 18.279762
final value 18.279762
stopped after 100 iterations
# weights: 27
initial value 235.470287
iter 10 value 12.679912
iter 20 value 4.481832
iter 30 value 0.353513
iter 40 value 0.001627
final value 0.000074
converged
# weights: 43
initial value 263.849517
iter 10 value 2.496992
iter 20 value 0.044056
iter 30 value 0.000415
final value 0.000078

```

```

converged
# weights: 11
initial value 290.227079
iter 10 value 102.865082
iter 20 value 89.630304
iter 30 value 65.793923
final value 65.178550
converged
# weights: 27
initial value 254.037222
iter 10 value 23.180151
iter 20 value 18.392378
iter 30 value 17.456513
iter 40 value 17.338006
final value 17.334196
converged
# weights: 43
initial value 238.822011
iter 10 value 36.630402
iter 20 value 17.810940
iter 30 value 16.290208
iter 40 value 16.103912
iter 50 value 15.945421
iter 60 value 15.794454
iter 70 value 15.781112
iter 80 value 15.780389
iter 80 value 15.780389
iter 80 value 15.780389
final value 15.780389
converged
# weights: 11
initial value 231.030091
iter 10 value 84.830136
iter 20 value 79.724402
iter 30 value 79.352726
iter 40 value 73.176993
iter 50 value 53.615603
iter 60 value 37.260305
iter 70 value 36.684206
iter 80 value 36.639941
iter 90 value 36.498125
iter 100 value 36.494944
final value 36.494944
stopped after 100 iterations
# weights: 27
initial value 232.177417

```

```

iter 10 value 1.371429
iter 20 value 0.564214
iter 30 value 0.478438
iter 40 value 0.319130
iter 50 value 0.155620
iter 60 value 0.149706
iter 70 value 0.137225
iter 80 value 0.128030
iter 90 value 0.117594
iter 100 value 0.105437
final value 0.105437
stopped after 100 iterations
# weights: 43
initial value 224.463464
iter 10 value 1.018825
iter 20 value 0.149583
iter 30 value 0.134991
iter 40 value 0.129984
iter 50 value 0.125555
iter 60 value 0.115649
iter 70 value 0.111054
iter 80 value 0.100355
iter 90 value 0.094506
iter 100 value 0.093308
final value 0.093308
stopped after 100 iterations
# weights: 11
initial value 218.732502
iter 10 value 85.520170
iter 20 value 39.421018
iter 30 value 25.038162
iter 40 value 21.831793
iter 50 value 21.397405
iter 60 value 21.328396
iter 70 value 21.311113
iter 80 value 21.304158
iter 90 value 21.290446
iter 100 value 21.267061
final value 21.267061
stopped after 100 iterations
# weights: 27
initial value 224.714137
iter 10 value 1.621945
iter 20 value 0.034524
iter 30 value 0.010357
iter 40 value 0.005259

```

```
iter 50 value 0.000831
iter 60 value 0.000324
final value 0.000093
converged
# weights: 43
initial value 228.399141
iter 10 value 5.570186
iter 20 value 0.035412
iter 30 value 0.001150
final value 0.000079
converged
# weights: 11
initial value 276.533660
iter 10 value 68.881648
iter 20 value 63.573994
final value 63.570432
converged
# weights: 27
initial value 205.928139
iter 10 value 34.309639
iter 20 value 19.265781
iter 30 value 19.138284
iter 40 value 19.134644
iter 50 value 19.134457
final value 19.134455
converged
# weights: 43
initial value 293.458955
iter 10 value 45.197397
iter 20 value 16.550020
iter 30 value 15.923228
iter 40 value 15.854265
iter 50 value 15.796604
iter 60 value 15.775963
final value 15.775823
converged
# weights: 11
initial value 237.695821
iter 10 value 93.911495
iter 20 value 85.688566
iter 30 value 76.292700
iter 40 value 74.561174
iter 50 value 74.272751
iter 60 value 73.720841
iter 70 value 73.700746
iter 80 value 73.665807
```

```
iter 90 value 73.639534
final value 73.639118
converged
# weights: 27
initial value 254.873695
iter 10 value 4.136252
iter 20 value 0.336076
iter 30 value 0.263759
iter 40 value 0.246886
iter 50 value 0.233788
iter 60 value 0.223164
iter 70 value 0.214285
iter 80 value 0.208709
iter 90 value 0.199803
iter 100 value 0.191141
final value 0.191141
stopped after 100 iterations
# weights: 43
initial value 244.510555
iter 10 value 1.210789
iter 20 value 0.915799
iter 30 value 0.542026
iter 40 value 0.254023
iter 50 value 0.188242
iter 60 value 0.146055
iter 70 value 0.129038
iter 80 value 0.118870
iter 90 value 0.113815
iter 100 value 0.109751
final value 0.109751
stopped after 100 iterations
# weights: 11
initial value 208.821709
iter 10 value 45.019890
iter 20 value 21.068701
iter 30 value 18.796930
iter 40 value 18.611712
iter 50 value 18.258549
iter 60 value 18.036399
iter 70 value 18.020003
iter 80 value 18.000299
iter 90 value 17.935003
iter 100 value 17.910076
final value 17.910076
stopped after 100 iterations
# weights: 27
```

```

initial value 231.254762
iter 10 value 0.912817
iter 20 value 0.026474
iter 30 value 0.004682
iter 40 value 0.001702
iter 50 value 0.000519
iter 60 value 0.000187
iter 70 value 0.000115
iter 80 value 0.000102
iter 80 value 0.000095
iter 80 value 0.000095
final value 0.000095
converged
# weights: 43
initial value 212.944490
iter 10 value 0.616688
iter 20 value 0.041716
iter 30 value 0.004359
iter 40 value 0.001260
iter 50 value 0.000513
final value 0.000072
converged
# weights: 11
initial value 225.974939
iter 10 value 92.646207
iter 20 value 88.339508
iter 30 value 65.399943
iter 40 value 63.231400
final value 63.231316
converged
# weights: 27
initial value 241.610575
iter 10 value 33.001759
iter 20 value 18.263916
iter 30 value 16.993277
iter 40 value 16.837261
iter 50 value 16.822718
final value 16.822288
converged
# weights: 43
initial value 224.934674
iter 10 value 20.873142
iter 20 value 15.696200
iter 30 value 15.105727
iter 40 value 15.028563
iter 50 value 15.019623

```

```
iter 60 value 15.017155
final value 15.017098
converged
# weights: 11
initial value 217.208093
iter 10 value 80.417904
iter 20 value 74.859890
iter 30 value 73.619831
iter 40 value 70.059996
iter 50 value 29.187868
iter 60 value 19.724370
iter 70 value 18.660824
iter 80 value 18.636883
iter 90 value 18.544457
iter 100 value 18.543880
final value 18.543880
stopped after 100 iterations
# weights: 27
initial value 253.790219
iter 10 value 6.777712
iter 20 value 0.167928
iter 30 value 0.146634
iter 40 value 0.140687
iter 50 value 0.130098
iter 60 value 0.123628
iter 70 value 0.120183
iter 80 value 0.115952
iter 90 value 0.112079
iter 100 value 0.110685
final value 0.110685
stopped after 100 iterations
# weights: 43
initial value 251.185625
iter 10 value 1.779712
iter 20 value 0.395772
iter 30 value 0.347821
iter 40 value 0.246530
iter 50 value 0.157107
iter 60 value 0.134544
iter 70 value 0.121057
iter 80 value 0.104967
iter 90 value 0.099091
iter 100 value 0.095671
final value 0.095671
stopped after 100 iterations
# weights: 11
```

```

initial value 245.789064
iter 10 value 82.305208
iter 20 value 76.183198
iter 30 value 72.534911
iter 40 value 69.556696
iter 50 value 55.187032
iter 60 value 14.995008
iter 70 value 12.103419
iter 80 value 11.996476
iter 90 value 11.824391
iter 100 value 10.980780
final value 10.980780
stopped after 100 iterations
# weights: 27
initial value 221.040318
iter 10 value 10.284721
iter 20 value 1.876077
iter 30 value 0.020907
iter 40 value 0.005423
iter 50 value 0.002311
iter 60 value 0.000251
final value 0.000094
converged
# weights: 43
initial value 226.439139
iter 10 value 2.691956
iter 20 value 0.054240
iter 30 value 0.004855
iter 40 value 0.002367
iter 50 value 0.000108
iter 50 value 0.000086
iter 50 value 0.000086
final value 0.000086
converged
# weights: 11
initial value 205.898816
iter 10 value 94.716035
iter 20 value 68.282790
iter 30 value 62.246598
final value 62.190470
converged
# weights: 27
initial value 232.087942
iter 10 value 24.173254
iter 20 value 17.812889
iter 30 value 16.802787

```

```

iter 40 value 16.756161
final value 16.756154
converged
# weights: 43
initial value 220.640025
iter 10 value 26.383262
iter 20 value 17.153821
iter 30 value 15.310770
iter 40 value 14.984682
iter 50 value 14.956768
iter 60 value 14.950991
iter 70 value 14.950657
final value 14.950654
converged
# weights: 11
initial value 227.827068
iter 10 value 105.349455
iter 20 value 74.370242
iter 30 value 73.974986
iter 40 value 73.956349
iter 50 value 73.943819
iter 60 value 73.940153
iter 70 value 73.927851
iter 80 value 73.926924
iter 90 value 73.926644
iter 100 value 73.926579
final value 73.926579
stopped after 100 iterations
# weights: 27
initial value 241.412638
iter 10 value 1.341514
iter 20 value 0.113767
iter 30 value 0.103093
iter 40 value 0.099116
iter 50 value 0.096581
iter 60 value 0.094941
iter 70 value 0.090719
iter 80 value 0.088311
iter 90 value 0.086477
iter 100 value 0.084665
final value 0.084665
stopped after 100 iterations
# weights: 43
initial value 273.641852
iter 10 value 3.328309
iter 20 value 0.130233

```

```

iter 30 value 0.106159
iter 40 value 0.102600
iter 50 value 0.101129
iter 60 value 0.095872
iter 70 value 0.092049
iter 80 value 0.085996
iter 90 value 0.084774
iter 100 value 0.083060
final value 0.083060
stopped after 100 iterations
# weights: 11
initial value 206.307197
iter 10 value 78.657031
iter 20 value 47.535654
iter 30 value 25.275261
iter 40 value 20.074996
iter 50 value 17.627622
iter 60 value 17.361667
iter 70 value 17.095322
iter 80 value 16.889392
iter 90 value 16.876908
iter 100 value 16.816249
final value 16.816249
stopped after 100 iterations
# weights: 27
initial value 227.935645
iter 10 value 6.179114
iter 20 value 3.820185
iter 30 value 3.750210
iter 40 value 0.056815
iter 50 value 0.009976
iter 60 value 0.005735
iter 70 value 0.004395
iter 80 value 0.001036
iter 90 value 0.000611
iter 100 value 0.000501
final value 0.000501
stopped after 100 iterations
# weights: 43
initial value 272.433697
iter 10 value 0.371712
iter 20 value 0.028493
iter 30 value 0.007305
iter 40 value 0.000980
iter 50 value 0.000507
iter 60 value 0.000483

```

```
iter 70 value 0.000139
final value 0.000076
converged
# weights: 11
initial value 274.635680
iter 10 value 106.808241
iter 20 value 78.451474
iter 30 value 63.791050
iter 40 value 62.332170
final value 62.332169
converged
# weights: 27
initial value 210.113231
iter 10 value 28.046705
iter 20 value 18.626888
iter 30 value 17.947236
iter 40 value 17.908063
iter 50 value 17.906294
final value 17.906293
converged
# weights: 43
initial value 217.929406
iter 10 value 30.962124
iter 20 value 17.790981
iter 30 value 16.269628
iter 40 value 16.077536
iter 50 value 16.047302
iter 60 value 16.041552
iter 70 value 16.041481
final value 16.041474
converged
# weights: 11
initial value 214.294333
iter 10 value 79.830667
iter 20 value 73.158278
iter 30 value 63.110885
iter 40 value 30.183863
iter 50 value 20.343415
iter 60 value 18.343994
iter 70 value 17.660823
iter 80 value 17.610172
iter 90 value 17.535427
iter 100 value 17.535135
final value 17.535135
stopped after 100 iterations
# weights: 27
```

```

initial value 199.371241
iter 10 value 5.619914
iter 20 value 0.334413
iter 30 value 0.275914
iter 40 value 0.234345
iter 50 value 0.184749
iter 60 value 0.165784
iter 70 value 0.152035
iter 80 value 0.141342
iter 90 value 0.122179
iter 100 value 0.115802
final value 0.115802
stopped after 100 iterations
# weights: 43
initial value 219.362401
iter 10 value 2.601869
iter 20 value 0.178260
iter 30 value 0.150890
iter 40 value 0.144801
iter 50 value 0.136344
iter 60 value 0.130549
iter 70 value 0.121676
iter 80 value 0.113995
iter 90 value 0.107592
iter 100 value 0.105189
final value 0.105189
stopped after 100 iterations
# weights: 11
initial value 230.600804
iter 10 value 80.784220
iter 20 value 80.669503
iter 30 value 80.656287
iter 40 value 80.654702
final value 80.653789
converged
# weights: 27
initial value 247.288287
iter 10 value 0.462240
iter 20 value 0.041927
iter 30 value 0.000926
final value 0.000092
converged
# weights: 43
initial value 211.609266
iter 10 value 1.894839
iter 20 value 0.055593

```

```
iter 30 value 0.005260
final value 0.000078
converged
# weights: 11
initial value 225.643806
iter 10 value 96.230337
iter 20 value 91.404225
iter 30 value 86.534782
iter 40 value 85.622664
final value 85.622653
converged
# weights: 27
initial value 223.019087
iter 10 value 32.431030
iter 20 value 19.880659
iter 30 value 19.655905
iter 40 value 19.416989
iter 50 value 19.403726
final value 19.403691
converged
# weights: 43
initial value 233.122395
iter 10 value 22.666287
iter 20 value 16.395582
iter 30 value 15.953543
iter 40 value 15.937227
iter 50 value 15.935738
iter 60 value 15.935698
final value 15.935694
converged
# weights: 11
initial value 253.496026
iter 10 value 117.389138
iter 20 value 80.821401
iter 30 value 79.105207
iter 40 value 76.735506
iter 50 value 62.383406
iter 60 value 36.817119
iter 70 value 34.490752
iter 80 value 34.212637
iter 90 value 34.070882
iter 100 value 33.973099
final value 33.973099
stopped after 100 iterations
# weights: 27
initial value 267.905364
```

```

iter 10 value 7.818129
iter 20 value 1.525215
iter 30 value 0.200527
iter 40 value 0.190737
iter 50 value 0.177140
iter 60 value 0.156905
iter 70 value 0.143296
iter 80 value 0.137079
iter 90 value 0.133776
iter 100 value 0.129244
final value 0.129244
stopped after 100 iterations
# weights: 43
initial value 258.686334
iter 10 value 2.220009
iter 20 value 0.167843
iter 30 value 0.134200
iter 40 value 0.123233
iter 50 value 0.118745
iter 60 value 0.114467
iter 70 value 0.109844
iter 80 value 0.105677
iter 90 value 0.103103
iter 100 value 0.100460
final value 0.100460
stopped after 100 iterations
# weights: 11
initial value 244.549405
iter 10 value 84.154834
iter 20 value 80.649706
iter 30 value 80.562118
iter 40 value 80.100996
iter 50 value 79.117356
iter 60 value 78.206823
iter 70 value 75.523139
iter 80 value 53.369591
iter 90 value 34.429589
iter 100 value 33.200509
final value 33.200509
stopped after 100 iterations
# weights: 27
initial value 231.103589
iter 10 value 0.139111
iter 20 value 0.014226
iter 30 value 0.001088
iter 40 value 0.000245

```

```
final value 0.000082
converged
# weights: 43
initial value 205.707563
iter 10 value 0.850946
iter 20 value 0.007988
final value 0.000087
converged
# weights: 11
initial value 215.541881
iter 10 value 120.662456
iter 20 value 81.908806
iter 30 value 64.831934
iter 40 value 64.374182
iter 40 value 64.374182
iter 40 value 64.374182
final value 64.374182
converged
# weights: 27
initial value 232.242119
iter 10 value 36.030349
iter 20 value 19.736326
iter 30 value 17.845010
iter 40 value 17.443599
iter 50 value 17.424737
iter 60 value 17.424200
final value 17.424198
converged
# weights: 43
initial value 223.215019
iter 10 value 22.133842
iter 20 value 16.163361
iter 30 value 15.753739
iter 40 value 15.744516
iter 50 value 15.744246
final value 15.744229
converged
# weights: 11
initial value 248.824009
iter 10 value 79.687291
iter 20 value 44.208211
iter 30 value 34.383105
iter 40 value 33.422495
iter 50 value 33.365599
iter 60 value 33.333554
iter 70 value 33.299274
```

```

iter 80 value 33.294979
iter 90 value 33.292861
iter 100 value 33.292379
final value 33.292379
stopped after 100 iterations
# weights: 27
initial value 338.304073
iter 10 value 1.602882
iter 20 value 0.143187
iter 30 value 0.132354
iter 40 value 0.124098
iter 50 value 0.116144
iter 60 value 0.103491
iter 70 value 0.101008
iter 80 value 0.094906
iter 90 value 0.089962
iter 100 value 0.087818
final value 0.087818
stopped after 100 iterations
# weights: 43
initial value 244.522840
iter 10 value 1.167983
iter 20 value 0.189230
iter 30 value 0.175466
iter 40 value 0.149328
iter 50 value 0.118680
iter 60 value 0.100218
iter 70 value 0.092711
iter 80 value 0.089214
iter 90 value 0.086481
iter 100 value 0.085444
final value 0.085444
stopped after 100 iterations
# weights: 11
initial value 230.699584
iter 10 value 67.741435
iter 20 value 22.418950
iter 30 value 20.839187
iter 40 value 20.098303
iter 50 value 19.643547
iter 60 value 19.606845
iter 70 value 19.433894
iter 80 value 19.413115
iter 90 value 19.390120
iter 100 value 19.375593
final value 19.375593

```

```

stopped after 100 iterations
# weights: 27
initial value 261.508765
iter 10 value 15.319876
iter 20 value 0.818285
iter 30 value 0.023042
iter 40 value 0.002572
iter 50 value 0.000593
final value 0.000096
converged
# weights: 43
initial value 279.443335
iter 10 value 0.638195
iter 20 value 0.013108
iter 30 value 0.001778
iter 40 value 0.001016
iter 50 value 0.000270
iter 60 value 0.000163
iter 70 value 0.000114
final value 0.000100
converged
# weights: 11
initial value 203.352332
iter 10 value 90.324511
iter 20 value 70.255848
iter 30 value 63.590950
final value 63.527233
converged
# weights: 27
initial value 246.385593
iter 10 value 32.381144
iter 20 value 18.553465
iter 30 value 17.938151
iter 40 value 17.485550
iter 50 value 17.435839
iter 60 value 17.434074
final value 17.434055
converged
# weights: 43
initial value 242.739103
iter 10 value 20.325257
iter 20 value 15.888352
iter 30 value 15.531354
iter 40 value 15.439580
iter 50 value 15.437974
iter 60 value 15.437628

```

```
final  value 15.437626
converged
# weights: 11
initial  value 228.259490
iter   10 value 92.029650
iter   20 value 78.143994
iter   30 value 73.881703
iter   40 value 73.388069
iter   50 value 56.132830
iter   60 value 24.916085
iter   70 value 20.733557
iter   80 value 20.290825
iter   90 value 20.111609
iter  100 value 20.051700
final  value 20.051700
stopped after 100 iterations
# weights: 27
initial  value 261.819953
iter   10 value 37.022974
iter   20 value 1.161671
iter   30 value 0.292255
iter   40 value 0.268085
iter   50 value 0.254686
iter   60 value 0.242239
iter   70 value 0.222699
iter   80 value 0.216541
iter   90 value 0.201107
iter  100 value 0.189186
final  value 0.189186
stopped after 100 iterations
# weights: 43
initial  value 229.874738
iter   10 value 3.207091
iter   20 value 0.114332
iter   30 value 0.103895
iter   40 value 0.099679
iter   50 value 0.097248
iter   60 value 0.094517
iter   70 value 0.093018
iter   80 value 0.090720
iter   90 value 0.088489
iter  100 value 0.086935
final  value 0.086935
stopped after 100 iterations
# weights: 11
initial  value 207.238292
```

```
iter 10 value 80.862601
iter 20 value 80.496260
iter 30 value 80.058698
iter 40 value 79.959466
iter 50 value 79.518384
iter 60 value 79.182805
iter 70 value 79.072430
iter 80 value 79.045750
iter 90 value 78.789421
iter 100 value 78.413599
final value 78.413599
stopped after 100 iterations
# weights: 27
initial value 229.286812
iter 10 value 16.419396
iter 20 value 0.173286
iter 30 value 0.006039
iter 40 value 0.002496
iter 50 value 0.001687
iter 60 value 0.000227
iter 70 value 0.000218
iter 80 value 0.000106
final value 0.000095
converged
# weights: 43
initial value 261.243714
iter 10 value 0.661150
iter 20 value 0.008640
iter 30 value 0.000619
final value 0.000090
converged
# weights: 11
initial value 279.086856
iter 10 value 100.446452
iter 20 value 91.562037
iter 30 value 87.233352
iter 40 value 82.984602
final value 82.979997
converged
# weights: 27
initial value 221.577066
iter 10 value 32.542457
iter 20 value 17.924430
iter 30 value 17.709021
iter 40 value 17.696176
iter 50 value 17.693471
```

```
iter 50 value 17.693470
iter 50 value 17.693470
final value 17.693470
converged
# weights: 43
initial value 255.528570
iter 10 value 19.272336
iter 20 value 15.096889
iter 30 value 14.619304
iter 40 value 14.549373
iter 50 value 14.540847
iter 60 value 14.540257
final value 14.540205
converged
# weights: 11
initial value 220.700784
iter 10 value 81.932983
iter 20 value 74.183771
iter 30 value 73.957967
iter 40 value 73.784605
iter 50 value 73.726169
iter 60 value 73.684189
iter 70 value 73.596851
iter 80 value 73.533121
iter 90 value 73.516153
iter 100 value 73.511304
final value 73.511304
stopped after 100 iterations
# weights: 27
initial value 289.452903
iter 10 value 3.845338
iter 20 value 0.152039
iter 30 value 0.130855
iter 40 value 0.112731
iter 50 value 0.106839
iter 60 value 0.102378
iter 70 value 0.097972
iter 80 value 0.094847
iter 90 value 0.091452
iter 100 value 0.086705
final value 0.086705
stopped after 100 iterations
# weights: 43
initial value 242.546039
iter 10 value 2.161107
iter 20 value 0.356589
```

```

iter 30 value 0.331482
iter 40 value 0.265387
iter 50 value 0.157238
iter 60 value 0.113905
iter 70 value 0.100589
iter 80 value 0.083852
iter 90 value 0.079203
iter 100 value 0.075720
final value 0.075720
stopped after 100 iterations
# weights: 11
initial value 236.671387
iter 10 value 80.025974
iter 20 value 68.320131
iter 30 value 25.967052
iter 40 value 15.583480
iter 50 value 13.229836
iter 60 value 12.866705
iter 70 value 12.522363
iter 80 value 12.235639
iter 90 value 12.175979
iter 100 value 11.943247
final value 11.943247
stopped after 100 iterations
# weights: 27
initial value 285.647633
iter 10 value 2.379830
iter 20 value 0.021259
iter 30 value 0.000218
final value 0.000093
converged
# weights: 43
initial value 222.986366
iter 10 value 0.469430
iter 20 value 0.013398
iter 30 value 0.003488
iter 40 value 0.000864
iter 50 value 0.000101
iter 50 value 0.000075
iter 50 value 0.000074
final value 0.000074
converged
# weights: 11
initial value 231.664903
iter 10 value 105.520390
iter 20 value 90.636508

```

```
iter 30 value 72.539410
iter 40 value 62.577639
iter 50 value 62.568782
final value 62.568778
converged
# weights: 27
initial value 213.905537
iter 10 value 35.316520
iter 20 value 18.118398
iter 30 value 17.591444
iter 40 value 17.491964
iter 50 value 17.490675
final value 17.490674
converged
# weights: 43
initial value 192.756131
iter 10 value 27.406116
iter 20 value 16.422112
iter 30 value 15.923646
iter 40 value 15.852067
iter 50 value 15.809700
iter 60 value 15.808866
final value 15.808853
converged
# weights: 11
initial value 248.799114
iter 10 value 81.675665
iter 20 value 80.019211
iter 30 value 78.891553
iter 40 value 73.466115
iter 50 value 71.631721
iter 60 value 69.011138
iter 70 value 38.586511
iter 80 value 15.554937
iter 90 value 14.178454
iter 100 value 13.956013
final value 13.956013
stopped after 100 iterations
# weights: 27
initial value 248.401401
iter 10 value 19.831357
iter 20 value 10.130848
iter 30 value 0.510297
iter 40 value 0.397208
iter 50 value 0.273770
iter 60 value 0.186649
```

```

iter 70 value 0.177182
iter 80 value 0.169915
iter 90 value 0.142436
iter 100 value 0.126913
final value 0.126913
stopped after 100 iterations
# weights: 43
initial value 219.716496
iter 10 value 2.832805
iter 20 value 0.199179
iter 30 value 0.158590
iter 40 value 0.150640
iter 50 value 0.143229
iter 60 value 0.134378
iter 70 value 0.126576
iter 80 value 0.118777
iter 90 value 0.110083
iter 100 value 0.106826
final value 0.106826
stopped after 100 iterations
# weights: 11
initial value 237.349755
iter 10 value 82.208956
iter 20 value 77.515642
iter 30 value 75.024421
iter 40 value 74.157433
iter 50 value 73.124900
iter 60 value 71.723289
iter 70 value 69.581282
iter 80 value 62.761779
iter 90 value 32.643954
iter 100 value 24.373618
final value 24.373618
stopped after 100 iterations
# weights: 27
initial value 278.217367
iter 10 value 2.557502
iter 20 value 0.012679
iter 30 value 0.000919
iter 40 value 0.000494
iter 50 value 0.000274
final value 0.000066
converged
# weights: 43
initial value 249.108426
iter 10 value 2.482445

```

```
iter 20 value 0.106864
iter 30 value 0.009759
iter 40 value 0.000127
iter 40 value 0.000093
iter 40 value 0.000091
final value 0.000091
converged
# weights: 11
initial value 222.556197
iter 10 value 110.359697
iter 20 value 82.975108
iter 30 value 66.118193
iter 40 value 65.741285
final value 65.741026
converged
# weights: 27
initial value 266.620574
iter 10 value 30.818850
iter 20 value 18.841859
iter 30 value 18.348686
iter 40 value 18.147858
iter 50 value 18.080259
final value 18.079926
converged
# weights: 43
initial value 204.181680
iter 10 value 24.295021
iter 20 value 16.920358
iter 30 value 16.716173
iter 40 value 16.702921
iter 50 value 16.700666
final value 16.700618
converged
# weights: 11
initial value 232.706367
iter 10 value 72.918173
iter 20 value 33.551229
iter 30 value 25.153578
iter 40 value 23.230050
iter 50 value 22.880162
iter 60 value 22.872881
iter 70 value 22.856849
final value 22.856846
converged
# weights: 27
initial value 206.923743
```

```

iter 10 value 6.510519
iter 20 value 0.400808
iter 30 value 0.371303
iter 40 value 0.300153
iter 50 value 0.188858
iter 60 value 0.155438
iter 70 value 0.144874
iter 80 value 0.134416
iter 90 value 0.117327
iter 100 value 0.107468
final value 0.107468
stopped after 100 iterations
# weights: 43
initial value 240.657276
iter 10 value 5.301837
iter 20 value 0.214142
iter 30 value 0.193025
iter 40 value 0.172524
iter 50 value 0.161568
iter 60 value 0.131512
iter 70 value 0.125403
iter 80 value 0.121234
iter 90 value 0.117323
iter 100 value 0.113494
final value 0.113494
stopped after 100 iterations
# weights: 11
initial value 216.690232
iter 10 value 80.900481
iter 20 value 78.824230
iter 30 value 78.739745
iter 40 value 78.705462
iter 50 value 78.701915
iter 60 value 78.694863
iter 70 value 78.694750
iter 70 value 78.694750
final value 78.694750
converged
# weights: 27
initial value 258.510684
iter 10 value 27.911328
iter 20 value 3.430114
iter 30 value 2.781095
iter 40 value 2.008951
final value 0.000016
converged

```

```

# weights: 43
initial value 296.948609
iter 10 value 2.173771
iter 20 value 0.022494
iter 30 value 0.005346
iter 40 value 0.000517
iter 50 value 0.000165
final value 0.000098
converged
# weights: 11
initial value 219.581825
iter 10 value 94.804121
iter 20 value 70.993211
iter 30 value 63.044853
final value 62.620009
converged
# weights: 27
initial value 253.014926
iter 10 value 33.122166
iter 20 value 21.896228
iter 30 value 20.044265
iter 40 value 19.888724
iter 50 value 19.868584
iter 60 value 19.868452
final value 19.868449
converged
# weights: 43
initial value 233.846897
iter 10 value 21.224766
iter 20 value 17.244901
iter 30 value 16.762286
iter 40 value 16.689292
iter 50 value 16.682731
iter 60 value 16.682413
iter 60 value 16.682413
iter 60 value 16.682413
final value 16.682413
converged
# weights: 11
initial value 228.241866
iter 10 value 94.939736
iter 20 value 82.855010
iter 30 value 80.825570
iter 40 value 77.585700
iter 50 value 74.054398
iter 60 value 73.318175

```

```

iter 70 value 73.267610
iter 80 value 72.764752
iter 90 value 72.184825
iter 100 value 70.316524
final value 70.316524
stopped after 100 iterations
# weights: 27
initial value 220.049014
iter 10 value 5.155735
iter 20 value 0.142924
iter 30 value 0.133818
iter 40 value 0.129830
iter 50 value 0.126707
iter 60 value 0.117562
iter 70 value 0.113262
iter 80 value 0.111496
iter 90 value 0.107242
iter 100 value 0.105163
final value 0.105163
stopped after 100 iterations
# weights: 43
initial value 235.044747
iter 10 value 7.017035
iter 20 value 0.650546
iter 30 value 0.376032
iter 40 value 0.325543
iter 50 value 0.227805
iter 60 value 0.201321
iter 70 value 0.147401
iter 80 value 0.138615
iter 90 value 0.130878
iter 100 value 0.125245
final value 0.125245
stopped after 100 iterations
# weights: 11
initial value 228.266624
iter 10 value 112.324198
iter 20 value 76.614606
iter 30 value 75.890754
iter 40 value 72.878372
iter 50 value 72.206297
iter 60 value 72.067227
iter 70 value 71.831136
iter 80 value 71.779718
iter 90 value 71.758388
iter 100 value 71.710200

```

```

final value 71.710200
stopped after 100 iterations
# weights: 27
initial value 229.913181
iter 10 value 2.540542
iter 20 value 0.018643
iter 30 value 0.000106
iter 30 value 0.000092
iter 30 value 0.000091
final value 0.000091
converged
# weights: 43
initial value 217.449884
iter 10 value 0.025032
iter 20 value 0.000147
final value 0.000001
converged
# weights: 11
initial value 218.780761
iter 10 value 94.246609
iter 20 value 76.957926
iter 30 value 63.267645
final value 63.039103
converged
# weights: 27
initial value 227.682168
iter 10 value 25.354744
iter 20 value 19.099825
iter 30 value 19.043700
iter 40 value 19.037475
final value 19.037405
converged
# weights: 43
initial value 198.210331
iter 10 value 23.659946
iter 20 value 16.585982
iter 30 value 15.795547
iter 40 value 15.648931
iter 50 value 15.633462
iter 60 value 15.632994
iter 70 value 15.632847
final value 15.632838
converged
# weights: 11
initial value 220.837182
iter 10 value 81.737881

```

```

iter 20 value 80.746118
iter 30 value 80.730383
iter 40 value 80.425633
iter 50 value 78.916808
iter 60 value 74.232009
iter 70 value 43.579247
iter 80 value 38.358501
iter 90 value 37.347610
iter 100 value 37.319276
final value 37.319276
stopped after 100 iterations
# weights: 27
initial value 223.330708
iter 10 value 17.756638
iter 20 value 0.275236
iter 30 value 0.247376
iter 40 value 0.229317
iter 50 value 0.207000
iter 60 value 0.158499
iter 70 value 0.151215
iter 80 value 0.129289
iter 90 value 0.123635
iter 100 value 0.104918
final value 0.104918
stopped after 100 iterations
# weights: 43
initial value 209.789857
iter 10 value 1.357716
iter 20 value 0.131255
iter 30 value 0.123850
iter 40 value 0.117563
iter 50 value 0.112024
iter 60 value 0.103425
iter 70 value 0.099732
iter 80 value 0.094765
iter 90 value 0.091387
iter 100 value 0.090275
final value 0.090275
stopped after 100 iterations
# weights: 11
initial value 255.060080
iter 10 value 79.377999
iter 20 value 60.588351
iter 30 value 30.268030
iter 40 value 29.528994
iter 50 value 29.176136

```

```

final  value 29.172381
converged
# weights: 27
initial  value 293.496279
iter  10 value 7.221149
iter  20 value 0.285932
iter  30 value 0.003091
final  value 0.000066
converged
# weights: 43
initial  value 204.685799
iter  10 value 2.408112
iter  20 value 0.016795
final  value 0.000080
converged
# weights: 11
initial  value 216.592061
iter  10 value 83.105567
iter  20 value 62.566184
iter  30 value 62.474508
final  value 62.474492
converged
# weights: 27
initial  value 235.363927
iter  10 value 38.320896
iter  20 value 18.583165
iter  30 value 16.909190
iter  40 value 16.840257
iter  50 value 16.837150
final  value 16.837130
converged
# weights: 43
initial  value 231.802545
iter  10 value 59.711866
iter  20 value 18.744511
iter  30 value 16.428259
iter  40 value 15.977378
iter  50 value 15.711672
iter  60 value 15.637690
iter  70 value 15.366625
iter  80 value 15.348557
iter  90 value 15.347872
iter 100 value 15.347824
final  value 15.347824
stopped after 100 iterations
# weights: 11

```

```

initial value 222.994515
iter 10 value 49.357133
iter 20 value 19.385508
iter 30 value 18.122738
iter 40 value 17.528268
iter 50 value 17.408383
iter 60 value 17.377074
iter 70 value 17.352687
final value 17.352678
converged
# weights: 27
initial value 219.457689
iter 10 value 9.039892
iter 20 value 8.382910
iter 30 value 3.765833
iter 40 value 2.026325
iter 50 value 1.068611
iter 60 value 0.989428
iter 70 value 0.890266
iter 80 value 0.693997
iter 90 value 0.575903
iter 100 value 0.528967
final value 0.528967
stopped after 100 iterations
# weights: 43
initial value 191.549413
iter 10 value 9.709051
iter 20 value 0.374163
iter 30 value 0.306383
iter 40 value 0.282269
iter 50 value 0.204711
iter 60 value 0.154829
iter 70 value 0.129128
iter 80 value 0.109482
iter 90 value 0.099804
iter 100 value 0.093041
final value 0.093041
stopped after 100 iterations
# weights: 11
initial value 209.290487
iter 10 value 79.549465
iter 20 value 75.653137
iter 30 value 40.912073
iter 40 value 16.993945
iter 50 value 16.144147
iter 60 value 15.837701

```

```

iter 70 value 15.708403
iter 80 value 15.688361
iter 90 value 15.645733
iter 100 value 15.622898
final value 15.622898
stopped after 100 iterations
# weights: 27
initial value 215.305889
iter 10 value 4.192771
iter 20 value 0.041824
iter 30 value 0.011167
iter 40 value 0.000454
iter 50 value 0.000160
iter 60 value 0.000141
final value 0.000089
converged
# weights: 43
initial value 221.824924
iter 10 value 0.891445
iter 20 value 0.004822
iter 30 value 0.000557
final value 0.000080
converged
# weights: 11
initial value 221.887433
iter 10 value 74.494487
iter 20 value 61.955993
final value 61.893500
converged
# weights: 27
initial value 240.673468
iter 10 value 32.590144
iter 20 value 17.708436
iter 30 value 17.459339
iter 40 value 17.433308
iter 50 value 17.264137
final value 17.262612
converged
# weights: 43
initial value 210.534249
iter 10 value 21.427345
iter 20 value 15.229803
iter 30 value 14.315926
iter 40 value 14.301850
iter 50 value 14.298808
iter 60 value 14.298658

```

```
iter 60 value 14.298658
iter 60 value 14.298658
final value 14.298658
converged
# weights: 11
initial value 246.507153
iter 10 value 73.772796
iter 20 value 20.051897
iter 30 value 17.092398
iter 40 value 16.513076
iter 50 value 16.407019
iter 60 value 16.350499
iter 70 value 16.294636
iter 80 value 16.293743
iter 90 value 16.291371
final value 16.289971
converged
# weights: 27
initial value 273.620038
iter 10 value 0.605519
iter 20 value 0.465661
iter 30 value 0.347984
iter 40 value 0.253976
iter 50 value 0.202231
iter 60 value 0.173308
iter 70 value 0.162934
iter 80 value 0.121676
iter 90 value 0.102503
iter 100 value 0.085874
final value 0.085874
stopped after 100 iterations
# weights: 43
initial value 238.056605
iter 10 value 1.131979
iter 20 value 0.105066
iter 30 value 0.076811
iter 40 value 0.072387
iter 50 value 0.068545
iter 60 value 0.065456
iter 70 value 0.064417
iter 80 value 0.063222
iter 90 value 0.061891
iter 100 value 0.060498
final value 0.060498
stopped after 100 iterations
# weights: 11
```

```

initial value 249.179500
iter 10 value 68.489044
iter 20 value 26.542397
iter 30 value 24.015696
iter 40 value 23.056116
iter 50 value 22.809764
iter 60 value 22.782302
iter 70 value 22.660241
iter 80 value 22.651670
iter 90 value 22.619486
iter 100 value 22.604888
final value 22.604888
stopped after 100 iterations
# weights: 27
initial value 287.436356
iter 10 value 7.019985
iter 20 value 0.357126
iter 30 value 0.009355
iter 40 value 0.000222
final value 0.000059
converged
# weights: 43
initial value 263.629656
iter 10 value 2.050059
iter 20 value 0.003725
iter 30 value 0.000642
final value 0.000072
converged
# weights: 11
initial value 242.855638
iter 10 value 90.385359
iter 20 value 83.055302
iter 30 value 82.942492
final value 82.940115
converged
# weights: 27
initial value 209.129718
iter 10 value 28.477206
iter 20 value 20.307729
iter 30 value 19.802548
iter 40 value 19.727303
final value 19.726812
converged
# weights: 43
initial value 202.405710
iter 10 value 19.810895

```

```
iter 20 value 17.120713
iter 30 value 16.606052
iter 40 value 16.566102
iter 50 value 16.560110
iter 60 value 16.559817
final value 16.559808
converged
# weights: 11
initial value 255.697289
iter 10 value 42.086514
iter 20 value 25.823714
iter 30 value 23.876696
iter 40 value 23.373745
iter 50 value 23.235562
iter 60 value 23.231190
iter 70 value 23.217206
final value 23.217205
converged
# weights: 27
initial value 240.982876
iter 10 value 32.885946
iter 20 value 13.801523
iter 30 value 0.272591
iter 40 value 0.232843
iter 50 value 0.188576
iter 60 value 0.157615
iter 70 value 0.135540
iter 80 value 0.118846
iter 90 value 0.115484
iter 100 value 0.111763
final value 0.111763
stopped after 100 iterations
# weights: 43
initial value 221.342652
iter 10 value 5.136975
iter 20 value 0.185512
iter 30 value 0.162308
iter 40 value 0.155163
iter 50 value 0.148129
iter 60 value 0.142650
iter 70 value 0.135659
iter 80 value 0.128751
iter 90 value 0.123065
iter 100 value 0.119791
final value 0.119791
stopped after 100 iterations
```

```

# weights: 11
initial value 220.028893
iter 10 value 110.029542
iter 20 value 80.086760
iter 30 value 79.764218
iter 40 value 78.303605
iter 50 value 77.943974
iter 60 value 77.913982
iter 70 value 77.292724
iter 80 value 70.103962
iter 90 value 68.880471
iter 100 value 68.679634
final value 68.679634
stopped after 100 iterations
# weights: 27
initial value 209.046667
iter 10 value 8.406483
iter 20 value 0.069973
iter 30 value 0.006762
iter 40 value 0.000220
iter 50 value 0.000105
final value 0.000095
converged
# weights: 43
initial value 229.402558
iter 10 value 2.338123
iter 20 value 0.021552
iter 30 value 0.000639
final value 0.000064
converged
# weights: 11
initial value 244.143055
iter 10 value 90.669282
iter 20 value 66.758076
iter 30 value 64.463851
final value 64.436193
converged
# weights: 27
initial value 249.551488
iter 10 value 28.548254
iter 20 value 18.070075
iter 30 value 17.118554
iter 40 value 16.968285
iter 50 value 16.951117
iter 60 value 16.949222
final value 16.949205

```

```

converged
# weights: 43
initial value 223.229092
iter 10 value 29.214713
iter 20 value 15.903091
iter 30 value 15.271475
iter 40 value 15.252726
iter 50 value 15.251632
final value 15.251619
converged
# weights: 11
initial value 236.787237
iter 10 value 80.972727
iter 20 value 80.878635
iter 30 value 80.702730
iter 40 value 78.742284
iter 50 value 66.465358
iter 60 value 39.166031
iter 70 value 37.351417
iter 80 value 37.185327
iter 90 value 37.144181
iter 100 value 37.141538
final value 37.141538
stopped after 100 iterations
# weights: 27
initial value 246.532376
iter 10 value 55.847143
iter 20 value 22.670750
iter 30 value 20.503966
iter 40 value 0.545789
iter 50 value 0.459733
iter 60 value 0.406792
iter 70 value 0.352572
iter 80 value 0.299482
iter 90 value 0.278842
iter 100 value 0.244967
final value 0.244967
stopped after 100 iterations
# weights: 43
initial value 263.725410
iter 10 value 3.631955
iter 20 value 0.272027
iter 30 value 0.218450
iter 40 value 0.197254
iter 50 value 0.174039
iter 60 value 0.158014

```

```

iter 70 value 0.115301
iter 80 value 0.107133
iter 90 value 0.098639
iter 100 value 0.094931
final value 0.094931
stopped after 100 iterations
# weights: 11
initial value 275.209237
iter 10 value 82.520602
iter 20 value 72.548303
iter 30 value 70.995519
iter 40 value 70.686144
iter 50 value 70.555810
iter 60 value 70.441483
iter 70 value 67.441967
iter 80 value 66.815467
iter 90 value 46.692664
iter 100 value 13.061118
final value 13.061118
stopped after 100 iterations
# weights: 27
initial value 231.620572
iter 10 value 2.828157
iter 20 value 0.020152
iter 30 value 0.000845
iter 40 value 0.000115
final value 0.000094
converged
# weights: 43
initial value 211.920743
iter 10 value 1.678122
iter 20 value 0.017635
iter 30 value 0.001868
final value 0.000067
converged
# weights: 11
initial value 244.633310
iter 10 value 77.119969
iter 20 value 63.218239
iter 30 value 62.841422
final value 62.829791
converged
# weights: 27
initial value 224.228650
iter 10 value 54.934001
iter 20 value 21.932030

```

```

iter 30 value 19.280266
iter 40 value 19.172840
iter 50 value 19.161515
iter 60 value 19.149454
iter 70 value 19.148858
final value 19.148857
converged
# weights: 43
initial value 263.237873
iter 10 value 33.596108
iter 20 value 20.089821
iter 30 value 16.834983
iter 40 value 15.777043
iter 50 value 15.498910
iter 60 value 15.443415
iter 70 value 15.432996
iter 80 value 15.431340
iter 90 value 15.430714
final value 15.430704
converged
# weights: 11
initial value 248.988755
iter 10 value 83.621397
iter 20 value 78.438934
iter 30 value 74.780292
iter 40 value 74.282447
iter 50 value 74.246853
iter 60 value 74.155013
iter 70 value 74.054360
iter 80 value 73.993013
iter 90 value 73.905848
iter 100 value 73.898539
final value 73.898539
stopped after 100 iterations
# weights: 27
initial value 248.029346
iter 10 value 1.283131
iter 20 value 0.168707
iter 30 value 0.157445
iter 40 value 0.150001
iter 50 value 0.137299
iter 60 value 0.128135
iter 70 value 0.123578
iter 80 value 0.116851
iter 90 value 0.093891
iter 100 value 0.090194

```

```

final value 0.090194
stopped after 100 iterations
# weights: 43
initial value 200.364388
iter 10 value 2.176346
iter 20 value 0.143469
iter 30 value 0.107708
iter 40 value 0.103987
iter 50 value 0.101857
iter 60 value 0.096297
iter 70 value 0.092345
iter 80 value 0.089368
iter 90 value 0.088039
iter 100 value 0.086445
final value 0.086445
stopped after 100 iterations
# weights: 11
initial value 208.718431
iter 10 value 51.577200
iter 20 value 18.509986
iter 30 value 17.321810
iter 40 value 16.540480
iter 50 value 15.996109
iter 60 value 15.956033
iter 70 value 15.766436
iter 80 value 15.738503
iter 90 value 15.665147
iter 100 value 15.625427
final value 15.625427
stopped after 100 iterations
# weights: 27
initial value 193.210334
iter 10 value 2.036880
iter 20 value 0.013876
final value 0.000099
converged
# weights: 43
initial value 198.720235
iter 10 value 1.881544
iter 20 value 0.009598
iter 30 value 0.000739
final value 0.000069
converged
# weights: 11
initial value 234.355967
iter 10 value 85.356490

```

```
iter 20 value 61.288182
iter 30 value 60.551241
final value 60.545874
converged
# weights: 27
initial value 282.587648
iter 10 value 28.231917
iter 20 value 18.723390
iter 30 value 17.187286
iter 40 value 16.107101
iter 50 value 15.838099
iter 60 value 15.808748
iter 70 value 15.807241
iter 70 value 15.807241
iter 70 value 15.807241
final value 15.807241
converged
# weights: 43
initial value 198.394604
iter 10 value 36.524752
iter 20 value 15.896063
iter 30 value 14.624912
iter 40 value 14.348742
iter 50 value 14.325210
iter 60 value 14.317169
final value 14.317063
converged
# weights: 11
initial value 210.555888
iter 10 value 80.617241
iter 20 value 28.146359
iter 30 value 18.538454
iter 40 value 16.846656
iter 50 value 16.768830
iter 60 value 16.730524
iter 70 value 16.632912
iter 80 value 16.630996
iter 90 value 16.614247
final value 16.613844
converged
# weights: 27
initial value 252.254941
iter 10 value 2.722918
iter 20 value 0.188448
iter 30 value 0.176490
iter 40 value 0.146729
```

```
iter 50 value 0.125317
iter 60 value 0.117575
iter 70 value 0.111594
iter 80 value 0.105771
iter 90 value 0.096825
iter 100 value 0.093055
final value 0.093055
stopped after 100 iterations
# weights: 43
initial value 233.146310
iter 10 value 1.356588
iter 20 value 0.157346
iter 30 value 0.124130
iter 40 value 0.112825
iter 50 value 0.100599
iter 60 value 0.090940
iter 70 value 0.088193
iter 80 value 0.084507
iter 90 value 0.082479
iter 100 value 0.079238
final value 0.079238
stopped after 100 iterations
# weights: 11
initial value 217.793147
iter 10 value 76.457626
iter 20 value 38.298253
iter 30 value 32.948750
iter 40 value 29.981417
iter 50 value 29.610946
iter 60 value 29.293918
iter 70 value 29.242331
final value 29.242329
converged
# weights: 27
initial value 212.319770
iter 10 value 3.652899
iter 20 value 0.075881
iter 30 value 0.001137
final value 0.000089
converged
# weights: 43
initial value 388.628588
iter 10 value 3.971739
iter 20 value 0.055712
iter 30 value 0.003341
iter 40 value 0.000328
```

```
final  value 0.000097
converged
# weights: 11
initial  value 210.906602
iter   10 value 89.781896
iter   20 value 64.092678
iter   30 value 61.751096
final  value 61.549357
converged
# weights: 27
initial  value 266.903986
iter   10 value 37.733271
iter   20 value 19.804173
iter   30 value 19.221448
iter   40 value 19.128189
iter   50 value 19.091907
iter   60 value 19.090653
final  value 19.090508
converged
# weights: 43
initial  value 205.680964
iter   10 value 18.467331
iter   20 value 15.830550
iter   30 value 15.780779
iter   40 value 15.770627
final  value 15.770368
converged
# weights: 11
initial  value 299.285077
iter   10 value 102.603543
iter   20 value 41.450381
iter   30 value 32.729819
iter   40 value 31.364764
iter   50 value 30.747097
iter   60 value 30.734594
iter   70 value 30.700139
final  value 30.700107
converged
# weights: 27
initial  value 225.719312
iter   10 value 5.007770
iter   20 value 0.673112
iter   30 value 0.494163
iter   40 value 0.391562
iter   50 value 0.319785
iter   60 value 0.300175
```

```

iter 70 value 0.270050
iter 80 value 0.261715
iter 90 value 0.248036
iter 100 value 0.222488
final value 0.222488
stopped after 100 iterations
# weights: 43
initial value 216.420965
iter 10 value 3.511868
iter 20 value 0.168806
iter 30 value 0.135916
iter 40 value 0.132550
iter 50 value 0.127881
iter 60 value 0.122716
iter 70 value 0.119641
iter 80 value 0.115219
iter 90 value 0.112969
iter 100 value 0.107766
final value 0.107766
stopped after 100 iterations
# weights: 11
initial value 224.085175
iter 10 value 75.121207
iter 20 value 61.173502
iter 30 value 34.394212
iter 40 value 16.171889
iter 50 value 14.862376
iter 60 value 11.850410
iter 70 value 11.609717
iter 80 value 11.315392
iter 90 value 11.095774
iter 100 value 11.061523
final value 11.061523
stopped after 100 iterations
# weights: 27
initial value 238.201290
iter 10 value 7.608068
iter 20 value 0.019021
iter 30 value 0.002148
final value 0.000074
converged
# weights: 43
initial value 238.797283
iter 10 value 0.946560
iter 20 value 0.033751
iter 30 value 0.009492

```

```
iter 40 value 0.001762
iter 50 value 0.000350
iter 60 value 0.000267
final value 0.000072
converged
# weights: 11
initial value 230.891544
iter 10 value 89.066517
iter 20 value 66.166055
iter 30 value 63.307530
final value 63.298658
converged
# weights: 27
initial value 217.402444
iter 10 value 22.076180
iter 20 value 17.840497
iter 30 value 17.221978
iter 40 value 17.218803
final value 17.218650
converged
# weights: 43
initial value 314.977173
iter 10 value 19.173062
iter 20 value 15.610744
iter 30 value 15.522626
iter 40 value 15.510120
iter 50 value 15.508388
final value 15.508383
converged
# weights: 11
initial value 218.158490
iter 10 value 89.304764
iter 20 value 28.173751
iter 30 value 15.212876
iter 40 value 13.915035
iter 50 value 13.024247
iter 60 value 12.811103
iter 70 value 12.794285
iter 80 value 12.777507
iter 90 value 12.774029
final value 12.772931
converged
# weights: 27
initial value 240.849780
iter 10 value 0.364400
iter 20 value 0.239569
```

```

iter 30 value 0.214642
iter 40 value 0.148797
iter 50 value 0.131957
iter 60 value 0.120674
iter 70 value 0.112940
iter 80 value 0.104939
iter 90 value 0.099581
iter 100 value 0.096429
final value 0.096429
stopped after 100 iterations
# weights: 43
initial value 244.140492
iter 10 value 4.496082
iter 20 value 0.196001
iter 30 value 0.132459
iter 40 value 0.128591
iter 50 value 0.123478
iter 60 value 0.117043
iter 70 value 0.112577
iter 80 value 0.104963
iter 90 value 0.101719
iter 100 value 0.099441
final value 0.099441
stopped after 100 iterations
# weights: 11
initial value 265.754475
iter 10 value 82.522645
iter 20 value 70.764288
iter 30 value 33.026981
iter 40 value 23.493345
iter 50 value 22.833404
iter 60 value 22.609285
iter 70 value 22.363704
iter 80 value 22.313784
iter 90 value 22.235449
iter 100 value 22.213157
final value 22.213157
stopped after 100 iterations
# weights: 27
initial value 218.599278
iter 10 value 3.209216
iter 20 value 0.028512
iter 30 value 0.001577
iter 40 value 0.000318
iter 50 value 0.000196
final value 0.000093

```

```

converged
# weights: 43
initial value 198.841920
iter 10 value 1.634149
iter 20 value 0.044219
iter 30 value 0.002240
iter 40 value 0.000486
final value 0.000085
converged
# weights: 11
initial value 227.588368
iter 10 value 93.721209
iter 20 value 83.569157
iter 30 value 82.557894
final value 82.537766
converged
# weights: 27
initial value 197.609327
iter 10 value 24.639456
iter 20 value 18.632949
iter 30 value 17.495864
iter 40 value 17.485349
final value 17.485308
converged
# weights: 43
initial value 252.881023
iter 10 value 38.223293
iter 20 value 17.985809
iter 30 value 16.288042
iter 40 value 16.046024
iter 50 value 15.980169
iter 60 value 15.972208
iter 70 value 15.961927
final value 15.961784
converged
# weights: 11
initial value 219.793119
iter 10 value 78.098860
iter 20 value 69.243193
iter 30 value 32.800264
iter 40 value 30.704948
iter 50 value 30.123802
iter 60 value 29.545166
iter 70 value 29.400111
iter 80 value 29.376219
iter 90 value 29.358480

```

```
iter 100 value 29.358194
final  value 29.358194
stopped after 100 iterations
# weights: 27
initial  value 285.716660
iter   10 value 0.803156
iter   20 value 0.157886
iter   30 value 0.146835
iter   40 value 0.133233
iter   50 value 0.118574
iter   60 value 0.104783
iter   70 value 0.103284
iter   80 value 0.099299
iter   90 value 0.097609
iter 100 value 0.096471
final  value 0.096471
stopped after 100 iterations
# weights: 43
initial  value 287.963576
iter   10 value 3.005656
iter   20 value 0.116966
iter   30 value 0.111666
iter   40 value 0.107607
iter   50 value 0.102147
iter   60 value 0.097642
iter   70 value 0.095660
iter   80 value 0.093923
iter   90 value 0.091674
iter 100 value 0.090571
final  value 0.090571
stopped after 100 iterations
# weights: 11
initial  value 229.603541
iter   10 value 66.041497
iter   20 value 41.938320
iter   30 value 25.534958
iter   40 value 20.264426
iter   50 value 19.551883
iter   60 value 19.490138
iter   70 value 19.469639
iter   80 value 19.465920
iter   90 value 19.451246
iter 100 value 19.399635
final  value 19.399635
stopped after 100 iterations
# weights: 27
```

```
initial value 228.423388
iter 10 value 23.416680
iter 20 value 0.038986
iter 30 value 0.002298
iter 40 value 0.000690
iter 50 value 0.000237
final value 0.000062
converged
# weights: 43
initial value 221.247642
iter 10 value 6.045519
iter 20 value 0.113071
iter 30 value 0.022912
iter 40 value 0.002045
iter 50 value 0.000329
final value 0.000089
converged
# weights: 11
initial value 214.098194
iter 10 value 90.986641
iter 20 value 88.999521
iter 30 value 84.397677
iter 40 value 83.797120
final value 83.797113
converged
# weights: 27
initial value 238.322997
iter 10 value 31.295250
iter 20 value 20.259164
iter 30 value 19.385222
iter 40 value 19.293974
iter 50 value 19.283520
iter 60 value 19.281308
final value 19.281304
converged
# weights: 43
initial value 277.840702
iter 10 value 34.076917
iter 20 value 16.823209
iter 30 value 16.068585
iter 40 value 16.021727
iter 50 value 16.016571
iter 60 value 16.016438
iter 70 value 16.016425
final value 16.016424
converged
```

```

# weights: 11
initial value 208.983117
iter 10 value 51.614899
iter 20 value 26.842176
iter 30 value 20.872634
iter 40 value 20.465516
iter 50 value 20.039433
iter 60 value 20.018109
iter 70 value 20.014528
final value 20.013412
converged
# weights: 27
initial value 233.303224
iter 10 value 26.099027
iter 20 value 15.361868
iter 30 value 11.349909
iter 40 value 9.918448
iter 50 value 7.443010
iter 60 value 6.816107
iter 70 value 6.581361
iter 80 value 6.504850
iter 90 value 6.072344
iter 100 value 0.896988
final value 0.896988
stopped after 100 iterations
# weights: 43
initial value 246.240479
iter 10 value 1.535129
iter 20 value 0.272582
iter 30 value 0.249494
iter 40 value 0.225338
iter 50 value 0.197728
iter 60 value 0.157638
iter 70 value 0.147418
iter 80 value 0.134649
iter 90 value 0.125892
iter 100 value 0.119679
final value 0.119679
stopped after 100 iterations
# weights: 11
initial value 215.417469
iter 10 value 81.015264
iter 20 value 80.657321
final value 80.653845
converged
# weights: 27

```

```
initial value 298.218995
iter 10 value 4.623526
iter 20 value 0.071258
iter 30 value 0.007710
iter 40 value 0.000951
final value 0.000069
converged
# weights: 43
initial value 242.666500
iter 10 value 3.047357
iter 20 value 0.007536
final value 0.000085
converged
# weights: 11
initial value 208.503465
iter 10 value 64.744913
iter 20 value 61.565695
iter 30 value 61.471146
iter 30 value 61.471146
iter 30 value 61.471146
final value 61.471146
converged
# weights: 27
initial value 228.135874
iter 10 value 26.350068
iter 20 value 19.081747
iter 30 value 18.609313
iter 40 value 18.541491
iter 50 value 18.534913
final value 18.534908
converged
# weights: 43
initial value 217.066161
iter 10 value 23.656995
iter 20 value 15.227425
iter 30 value 15.178990
iter 40 value 15.171352
iter 50 value 15.160962
iter 60 value 15.158018
final value 15.158017
converged
# weights: 11
initial value 281.892857
iter 10 value 66.609404
iter 20 value 24.794409
iter 30 value 18.949052
```

```
iter 40 value 17.148485
iter 50 value 16.863405
iter 60 value 16.784013
iter 70 value 16.750449
iter 80 value 16.749237
iter 90 value 16.745371
iter 100 value 16.745102
final value 16.745102
stopped after 100 iterations
# weights: 27
initial value 219.083718
iter 10 value 3.079266
iter 20 value 0.231035
iter 30 value 0.193518
iter 40 value 0.188135
iter 50 value 0.166271
iter 60 value 0.136891
iter 70 value 0.125637
iter 80 value 0.106822
iter 90 value 0.095226
iter 100 value 0.084963
final value 0.084963
stopped after 100 iterations
# weights: 43
initial value 232.866135
iter 10 value 0.787752
iter 20 value 0.460552
iter 30 value 0.312775
iter 40 value 0.245118
iter 50 value 0.183445
iter 60 value 0.149109
iter 70 value 0.118696
iter 80 value 0.101360
iter 90 value 0.092758
iter 100 value 0.089411
final value 0.089411
stopped after 100 iterations
# weights: 11
initial value 230.271745
iter 10 value 70.565868
iter 20 value 25.896422
iter 30 value 17.266033
iter 40 value 17.191668
iter 50 value 17.108590
iter 60 value 17.031116
iter 70 value 16.983276
```

```

iter 80 value 16.956707
iter 90 value 16.781250
iter 100 value 16.736379
final value 16.736379
stopped after 100 iterations
# weights: 27
initial value 217.990581
iter 10 value 2.082898
iter 20 value 0.004669
iter 30 value 0.001265
iter 40 value 0.000440
iter 50 value 0.000252
iter 60 value 0.000231
iter 70 value 0.000204
iter 80 value 0.000159
iter 90 value 0.000100
final value 0.000100
converged
# weights: 43
initial value 246.507869
iter 10 value 1.882866
iter 20 value 0.023938
iter 30 value 0.000696
iter 40 value 0.000526
final value 0.000087
converged
# weights: 11
initial value 212.318341
iter 10 value 87.456242
iter 20 value 63.886459
iter 30 value 62.199280
final value 62.195612
converged
# weights: 27
initial value 248.900145
iter 10 value 31.381710
iter 20 value 18.072124
iter 30 value 17.081547
iter 40 value 16.969317
iter 50 value 16.966337
final value 16.966326
converged
# weights: 43
initial value 213.441495
iter 10 value 30.158017
iter 20 value 15.807874

```

```

iter 30 value 15.183783
iter 40 value 15.165896
iter 50 value 15.160784
iter 60 value 15.160060
final value 15.160016
converged
# weights: 11
initial value 253.961700
iter 10 value 83.635654
iter 20 value 80.937794
iter 30 value 80.900253
iter 40 value 80.799175
iter 50 value 80.734976
iter 60 value 80.685395
iter 70 value 80.001145
iter 80 value 78.927661
iter 90 value 77.844400
iter 100 value 67.384078
final value 67.384078
stopped after 100 iterations
# weights: 27
initial value 217.858173
iter 10 value 19.953710
iter 20 value 0.758563
iter 30 value 0.334822
iter 40 value 0.250776
iter 50 value 0.212145
iter 60 value 0.170831
iter 70 value 0.161672
iter 80 value 0.136285
iter 90 value 0.118880
iter 100 value 0.106850
final value 0.106850
stopped after 100 iterations
# weights: 43
initial value 227.693737
iter 10 value 2.231248
iter 20 value 0.118270
iter 30 value 0.100616
iter 40 value 0.096887
iter 50 value 0.093394
iter 60 value 0.090023
iter 70 value 0.087660
iter 80 value 0.086087
iter 90 value 0.084415
iter 100 value 0.083162

```

```

final  value 0.083162
stopped after 100 iterations
# weights:  43
initial  value 325.302391
iter  10 value 25.992237
iter  20 value 17.491949
iter  30 value 17.076073
iter  40 value 16.855215
iter  50 value 16.852081
iter  60 value 16.851901
iter  70 value 16.851887
iter  70 value 16.851887
iter  70 value 16.851887
final  value 16.851887
converged

```

22.4 Combine models for comparison

```

resamps <- resamples(list(knn = knnfit,
                           lda = ldafit,
                           rf=rffit,
                           xgb=xgbfit,
                           svm=svmfit,
                           bayes=bayesfit,
                           nn=nnfit))
summary(resamps)

```

Call:

```
summary.resamples(object = resamps)
```

Models: knn, lda, rf, xgb, svm, bayes, nn

Number of resamples: 125

Accuracy

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
knn	0.94	0.9800000	1.00	0.9894965	1.00	1	0
lda	0.94	0.9800000	0.98	0.9821365	1.00	1	0
rf	0.90	0.9607843	0.98	0.9770541	1.00	1	0
xgb	0.90	0.9607843	0.98	0.9804141	1.00	1	0
svm	0.94	0.9800000	0.98	0.9824878	1.00	1	0
bayes	0.90	0.9400000	0.96	0.9632125	0.98	1	0
nn	0.94	0.9800000	1.00	0.9872627	1.00	1	0

Kappa

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
knn	0.9050633	0.9683544	1.0000000	0.9834377	1.0000000	1	0
lda	0.9050633	0.9683544	0.9688279	0.9718328	1.0000000	1	0
rf	0.8464373	0.9389952	0.9688279	0.9640670	1.0000000	1	0
xgb	0.8393316	0.9387387	0.9688279	0.9692177	1.0000000	1	0
svm	0.9035990	0.9683544	0.9688279	0.9723858	1.0000000	1	0
bayes	0.8417722	0.9064838	0.9381188	0.9423283	0.9688279	1	0
nn	0.9050633	0.9683544	1.0000000	0.9799779	1.0000000	1	0

```
resamps$values |> head(10)
```

	Resample	knn~Accuracy	knn~Kappa	lda~Accuracy	lda~Kappa	rf~Accuracy
1	Fold1.Rep01	1.00	1.0000000	1.0000000	1.0000000	0.9800000
2	Fold1.Rep02	1.00	1.0000000	1.0000000	1.0000000	0.9800000
3	Fold1.Rep03	1.00	1.0000000	0.9600000	0.9371859	0.9800000
4	Fold1.Rep04	1.00	1.0000000	0.9600000	0.9362245	0.9600000
5	Fold1.Rep05	1.00	1.0000000	1.0000000	1.0000000	0.9607843
6	Fold1.Rep06	1.00	1.0000000	1.0000000	1.0000000	0.9800000
7	Fold1.Rep07	0.98	0.9683544	1.0000000	1.0000000	1.0000000
8	Fold1.Rep08	0.96	0.9362245	0.9803922	0.9696970	0.9600000
9	Fold1.Rep09	1.00	1.0000000	0.9800000	0.9683544	0.9600000
10	Fold1.Rep10	1.00	1.0000000	0.9800000	0.9688279	0.9800000
	rf~Kappa	xgb~Accuracy	xgb~Kappa	svm~Accuracy	svm~Kappa	bayes~Accuracy
1	0.9686717	0.9800000	0.9683544	1.0000000	1.0000000	0.9411765
2	0.9688279	0.9600000	0.9378109	0.9800000	0.9688279	0.9215686
3	0.9688279	0.9803922	0.9692956	0.9800000	0.9683544	0.9411765
4	0.9362245	0.9800000	0.9688279	1.0000000	1.0000000	0.9400000
5	0.9397875	0.9600000	0.9362245	0.9803922	0.9692956	0.9800000
6	0.9686717	1.0000000	1.0000000	0.9800000	0.9683544	0.9800000
7	1.0000000	0.9800000	0.9683544	0.9800000	0.9683544	0.9600000
8	0.9381188	0.9800000	0.9686717	1.0000000	1.0000000	0.9607843
9	0.9371859	0.9607843	0.9387387	0.9800000	0.9688279	1.0000000
10	0.9683544	0.9400000	0.9048223	1.0000000	1.0000000	1.0000000
	bayes~Kappa	nn~Accuracy	nn~Kappa			
1	0.9090909	1.00	1.0000000			
2	0.8763636	1.00	1.0000000			
3	0.9066504	1.00	1.0000000			
4	0.9064838	1.00	1.0000000			
5	0.9688279	0.98	0.9683544			
6	0.9688279	0.98	0.9683544			
7	0.9381188	0.98	0.9683544			
8	0.9389952	0.98	0.9683544			

```

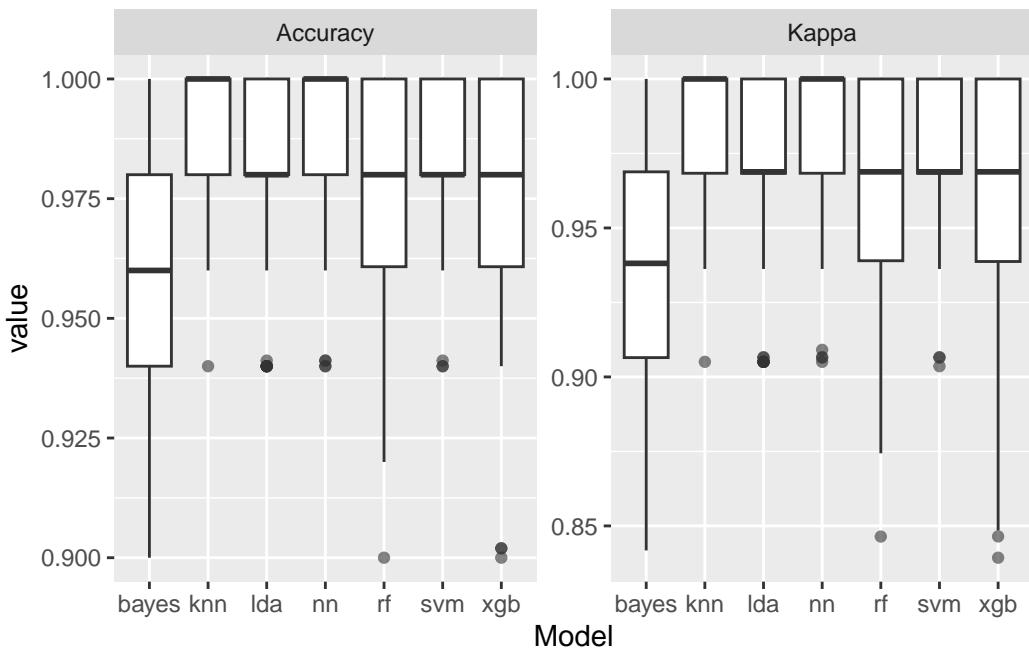
9     1.0000000      0.98 0.9688279
10    1.0000000      1.00 1.0000000

```

```

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')

```



```

diffs <- diff(resamps)
summary(diffs)

```

```

Call:
summary.diff.resamples(object = diffs)

p-value adjustment: bonferroni
Upper diagonal: estimates of the difference
Lower diagonal: p-value for H0: difference = 0

```

Accuracy					
knn	lda	rf	xgb	svm	bayes

knn	0.0073600	0.0124424	0.0090824	0.0070086	0.0262839	
lda	0.0054490	0.0050824	0.0017224	-0.0003514	0.0189239	
rf	4.088e-06	0.8366703		-0.0033600	-0.0054337	0.0138416
xgb	0.0004240	1.0000000	1.0000000		-0.0020737	0.0172016
svm	0.0050433	1.0000000	0.4638386	1.0000000		0.0192753
bayes	4.164e-15	5.341e-09	0.0002465	1.836e-05	4.374e-09	
nn	1.0000000	0.4401549	0.0006283	0.0650773	0.3866484	1.147e-12
	nn					
knn	0.0022337					
lda	-0.0051263					
rf	-0.0102086					
xgb	-0.0068486					
svm	-0.0047749					
bayes	-0.0240502					
nn						

Kappa

	knn	lda	rf	xgb	svm	bayes	nn
knn		0.011605	0.019371	0.014220	0.011052	0.041109	0.003460
lda	0.0055885		0.007766	0.002615	-0.000553	0.029505	-0.008145
rf	4.592e-06	0.9468259		-0.005151	-0.008319	0.021739	-0.015911
xgb	0.0004684	1.0000000	1.0000000		-0.003168	0.026889	-0.010760
svm	0.0051392	1.0000000	0.5300898	1.0000000		0.030058	-0.007592
bayes	5.622e-15	7.229e-09	0.0002339	2.084e-05	6.129e-09		-0.037650
nn	1.0000000	0.4206790	0.0006775	0.0655485	0.3646241	1.285e-12	

```
resamps2 <- resamples(list(knn = knnfit,
                           nn=nnfit))
diffs2 <- diff(resamps2)
summary(diffs2)
```

Call:

summary.diff.resamples(object = diffs2)

p-value adjustment: bonferroni

Upper diagonal: estimates of the difference

Lower diagonal: p-value for H0: difference = 0

Accuracy

	knn	nn
knn		0.002234
nn	0.2044	

```
Kappa
  knn      nn
knn      0.00346
nn   0.2123
```

```
resamps3 <- resamples(list(knn = knnfit,
                           rf=rffit))
diffs3 <- diff(resamps3)
summary(diffs3)
```

```
Call:
summary.diff.resamples(object = diffs3)

p-value adjustment: bonferroni
Upper diagonal: estimates of the difference
Lower diagonal: p-value for H0: difference = 0

Accuracy
  knn      rf
knn      0.01244
rf   1.947e-07

Kappa
  knn      rf
knn      0.01937
rf   2.187e-07
```

22.5 Access to individual models

```
xgbfit
```

```
eXtreme Gradient Boosting

251 samples
 4 predictor
 3 classes: 'Adelie', 'Chinstrap', 'Gentoo'

Pre-processing: centered (4), scaled (4)
Resampling: Cross-Validated (5 fold, repeated 25 times)
Summary of sample sizes: 201, 200, 201, 201, 201, ...
Resampling results across tuning parameters:
```

max_depth	gamma	nrounds	Accuracy	Kappa
5	0.001	50	0.9799373	0.9684694
5	0.001	100	0.9797773	0.9682163
5	0.001	200	0.9797773	0.9682163
5	0.010	50	0.9804141	0.9692177
5	0.010	100	0.9804141	0.9692177
5	0.010	200	0.9804141	0.9692177
10	0.001	50	0.9797804	0.9682270
10	0.001	100	0.9796204	0.9679739
10	0.001	200	0.9796204	0.9679739
10	0.010	50	0.9802573	0.9689752
10	0.010	100	0.9802573	0.9689752
10	0.010	200	0.9802573	0.9689752
15	0.001	50	0.9797804	0.9682270
15	0.001	100	0.9796204	0.9679739
15	0.001	200	0.9796204	0.9679739
15	0.010	50	0.9802573	0.9689752
15	0.010	100	0.9802573	0.9689752
15	0.010	200	0.9802573	0.9689752

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 = 50, max_depth = 5, 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
4	50	5	1	0.01		1	1

```
knnfit[["bestTune"]]
```

```
k
1 1
```

```
# use xgbfit to predict test data #####
xgbpred <- predict(xgbfit, newdata = testdata)
confusionMatrix(xgbpred, testdata$species)
```

Confusion Matrix and Statistics

		Reference		
Prediction		Adelie	Chinstrap	Gentoo
Adelie		35	1	0
Chinstrap		1	16	0
Gentoo		0	0	29

Overall Statistics

Accuracy : 0.9756
95% CI : (0.9147, 0.997)

No Information Rate : 0.439
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9618

McNemar's Test P-Value : NA

Statistics by Class:

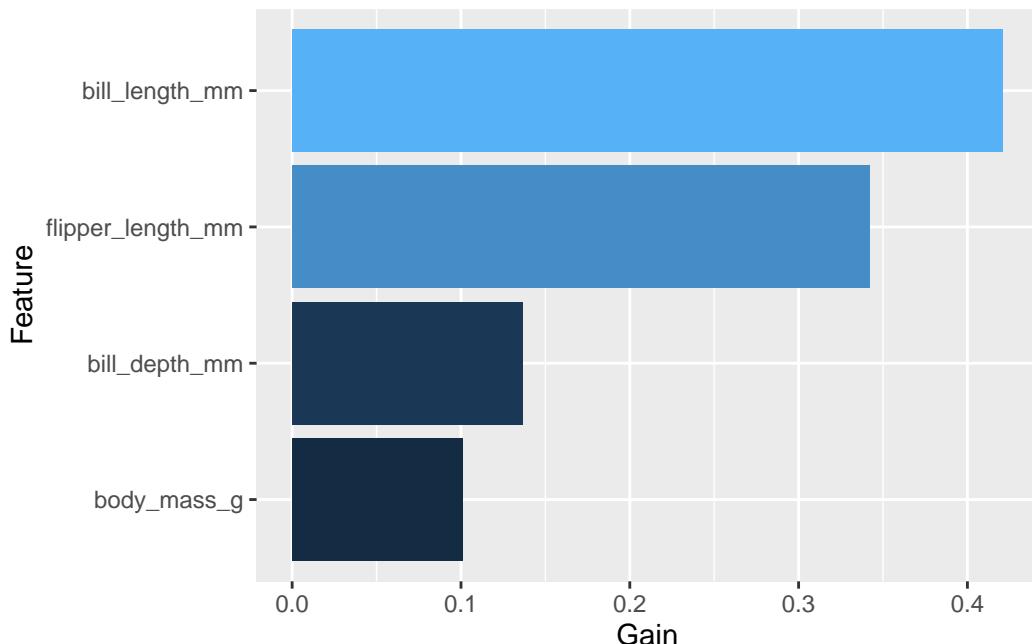
	Class: Adelie	Class: Chinstrap	Class: Gentoo
Sensitivity	0.9722	0.9412	1.0000
Specificity	0.9783	0.9846	1.0000
Pos Pred Value	0.9722	0.9412	1.0000
Neg Pred Value	0.9783	0.9846	1.0000
Prevalence	0.4390	0.2073	0.3537
Detection Rate	0.4268	0.1951	0.3537
Detection Prevalence	0.4390	0.2073	0.3537
Balanced Accuracy	0.9752	0.9629	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 0.1007493 0.1118378 0.21904762
2:   bill_depth_mm 0.1363051 0.2887459 0.35238095
3: flipper_length_mm 0.3419961 0.1381401 0.066666667
4:   bill_length_mm 0.4209495 0.4612762 0.36190476
```

```
importance_xgb |>
  ggplot(aes(Feature, Gain)) +
  geom_col(aes(fill=Gain)) +
  coord_flip() +
  guides(fill="none")
```



23 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)
```

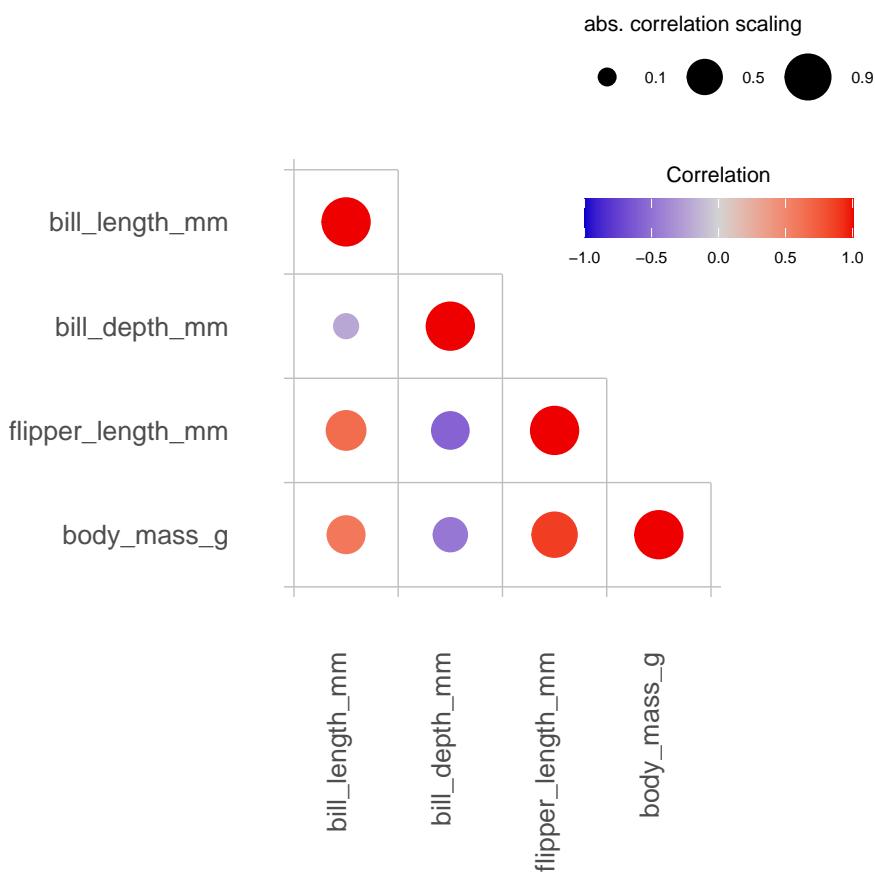
```
[conflicted] Will prefer dplyr::slice over any other package.
[conflicted] Will prefer dplyr::filter over any other package.
[conflicted] Will prefer stats::screeplot over any other package.
[conflicted] Will prefer stats::biplot over any other package.
[conflicted] Will prefer palmerpenguins::penguins 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'))
```

23.1 Exploration of correlations between predictor variables

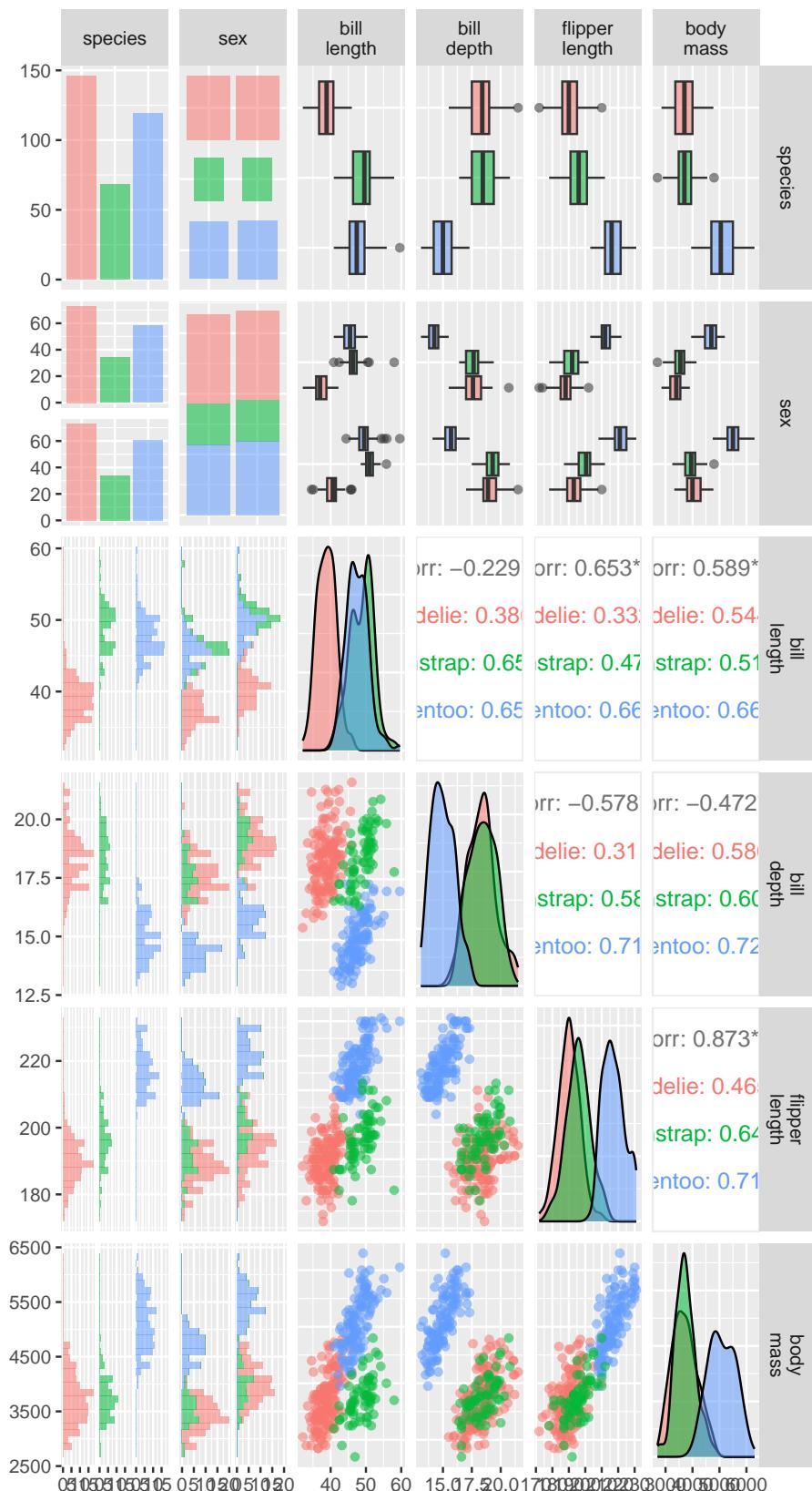
```
cortestR(rawdata |>
  # filter(species == "Adelie") |>
  select(predvars$names),
  split = T) |>
pluck('corout') |>
ggcormat(maxpoint = 8)
```



```
ggpairs(rawdata |> select(species, sex,
                           predvars$names) |>
  rename_with(~str_replace(.x, '(.+)_(.+)_+', '\\\1 \\\2')) |>
```

```
    str_wrap(8)),  
  aes(color=species, alpha=.5))
```

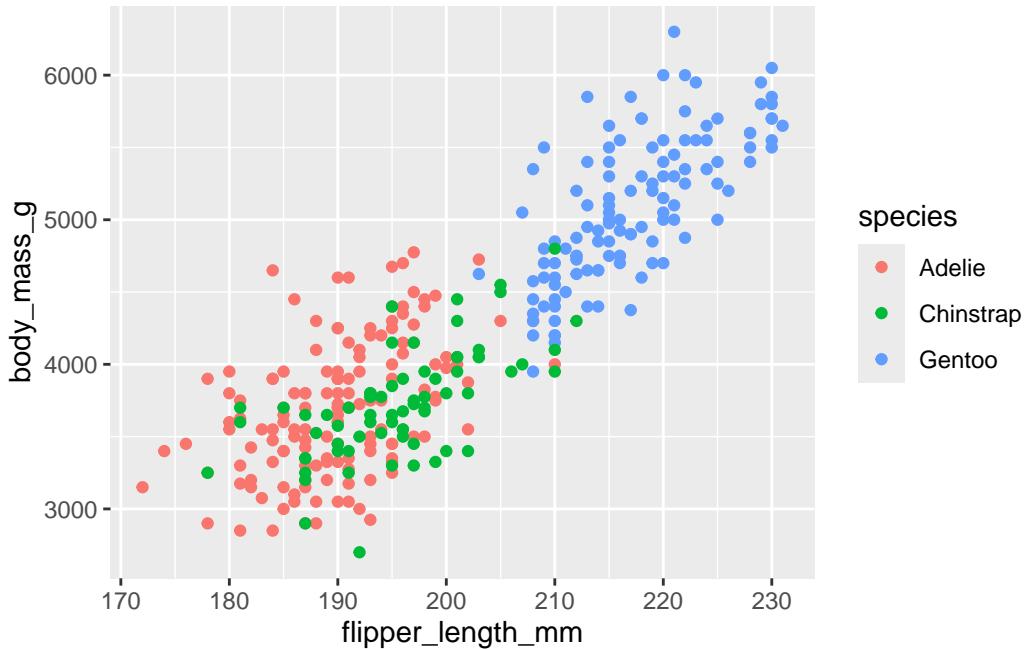
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.  
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.  
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.  
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.  
`stat_bin()` using `bins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



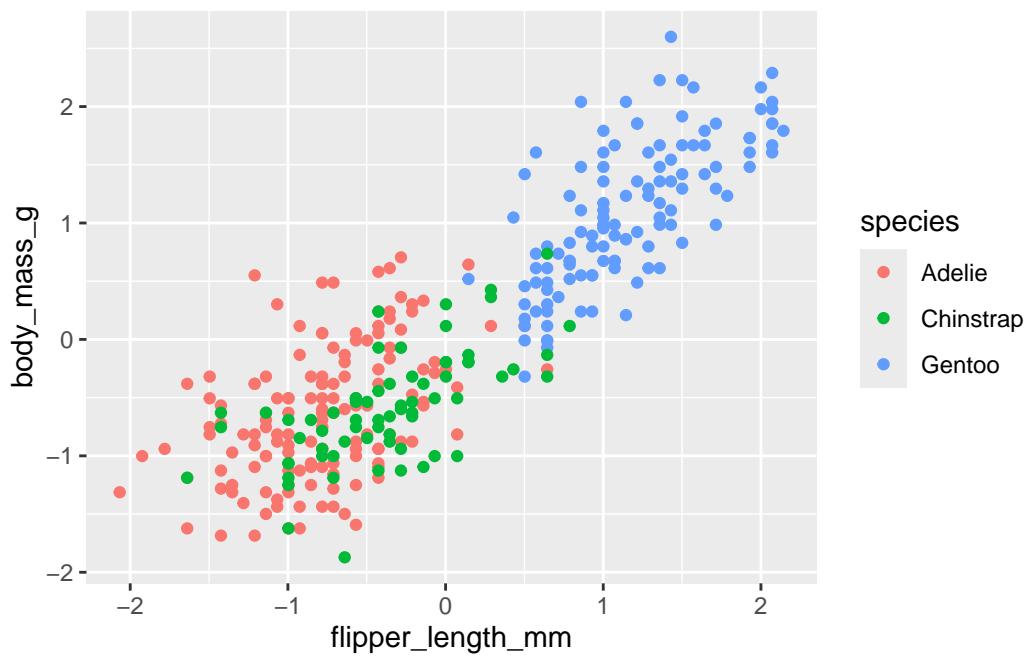
[Video on PCA](#)

23.2 Two variable example

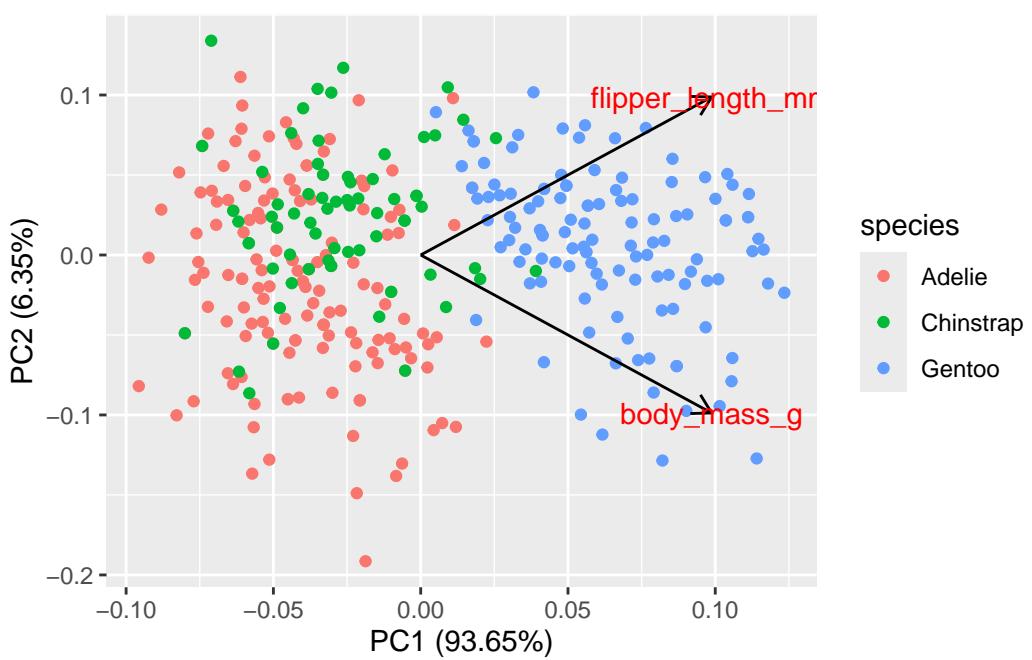
```
predvars2 <- ColSeeker(namepattern = c('body','flipper'))  
ggplot(rawdata, aes(.data[[predvars2$names[1]]], .data[[predvars2$names[2]]]))+  
  geom_point(aes(color=species))
```



```
v2data <- predict(preProcess(rawdata |>  
  select(predvars2$names),  
  method = c("center", "scale")),  
 rawdata) |>  
  select(species,predvars2$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)
```



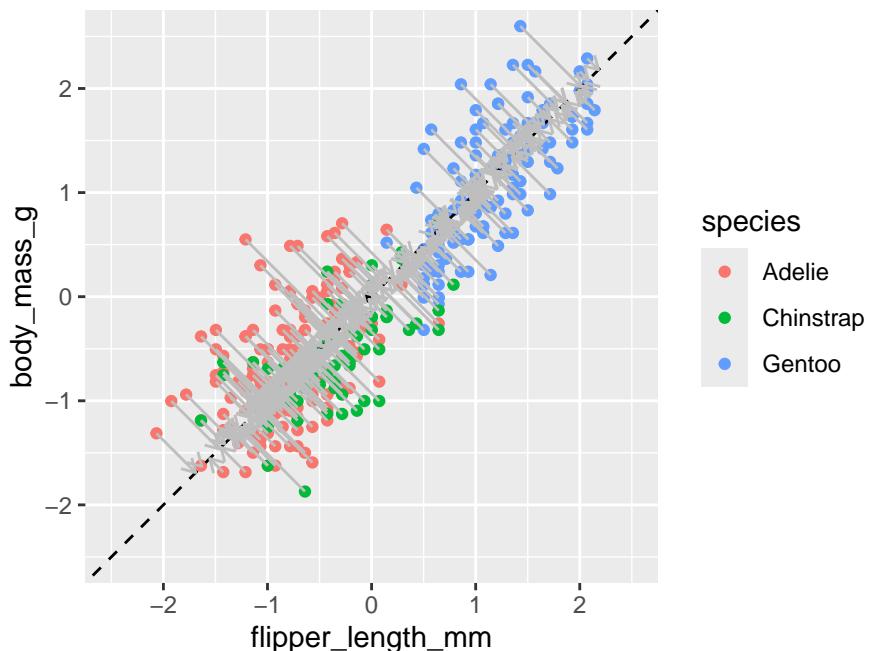
```

# 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))

```

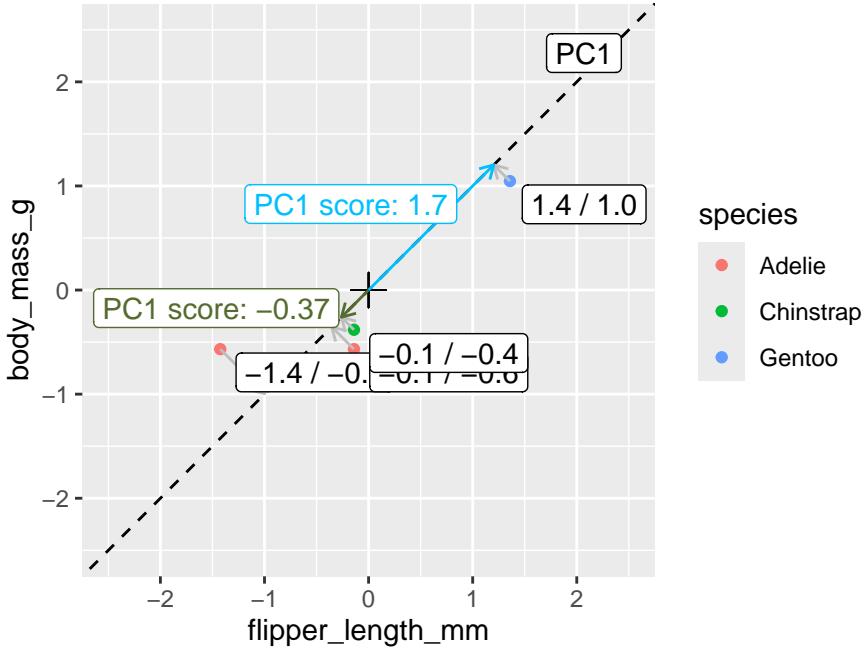
Warning in geom_point(aes(0, 0)): All aesthetics have length 1, but the data has 333 rows.
 i Please consider using `annotate()` or provide this layer with data containing
 a single row.



```

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

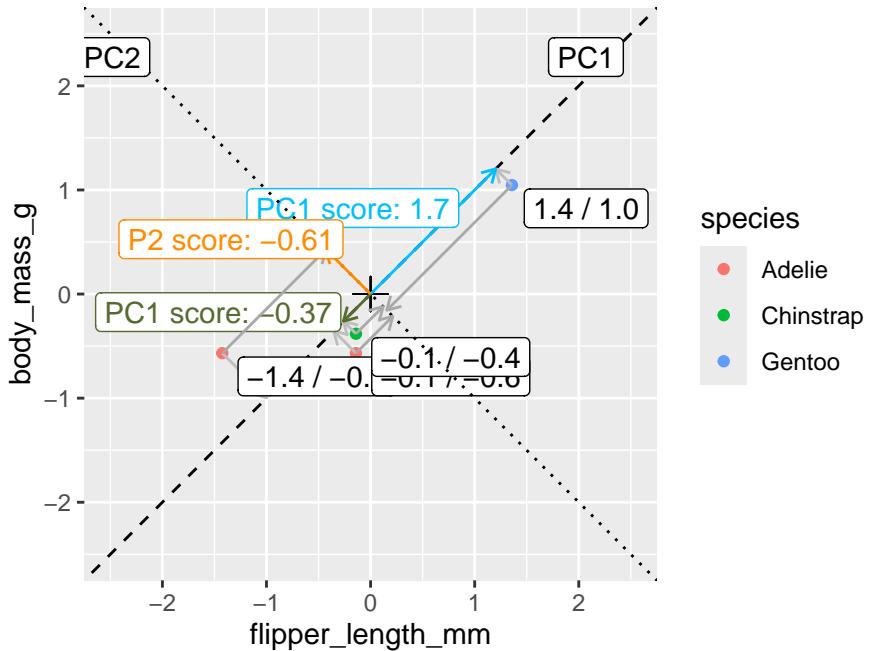
```



```

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,]	-1.4087181	-0.6059814
[2,]	-0.5006034	0.3021333
[3,]	1.7004692	0.2199949
[4,]	-0.3688797	0.1704096

```
v2data[c(1,101, 201,301),]
```

```
# A tibble: 4 x 3
  species   flipper_length_mm body_mass_g
  <fct>           <dbl>      <dbl>
1 Adelie        -1.42       -0.568
2 Adelie        -0.140      -0.568
3 Gentoo         1.36        1.05 
4 Chinstrap     -0.140      -0.381
```

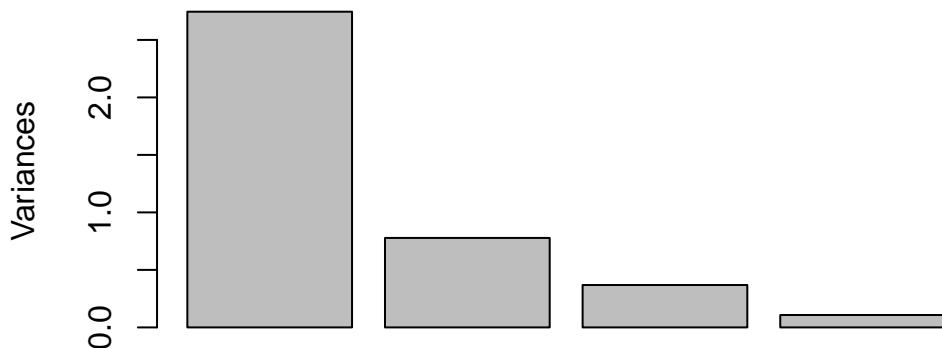
```
pca_out <- prcomp(rawdata |>
  select(predvars$names),
  center = T, scale. = T)
summary(pca_out)
```

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, n pcs = 4)
```

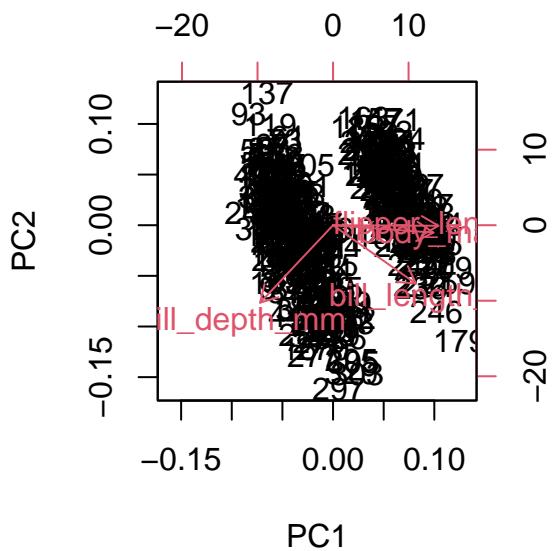
pca_out



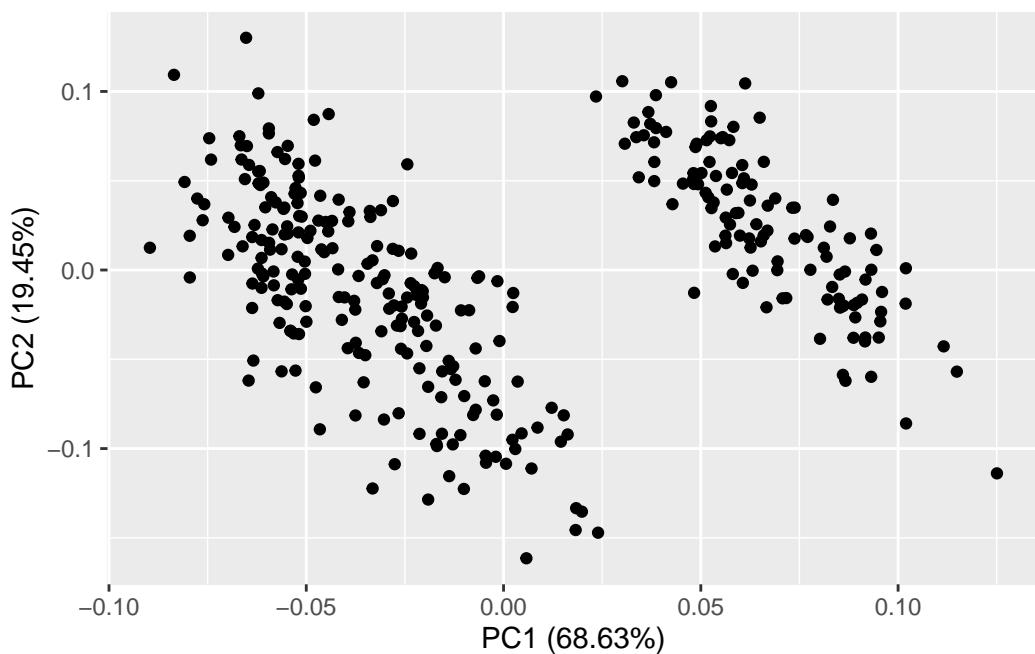
```
pca_out$rotation
```

	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

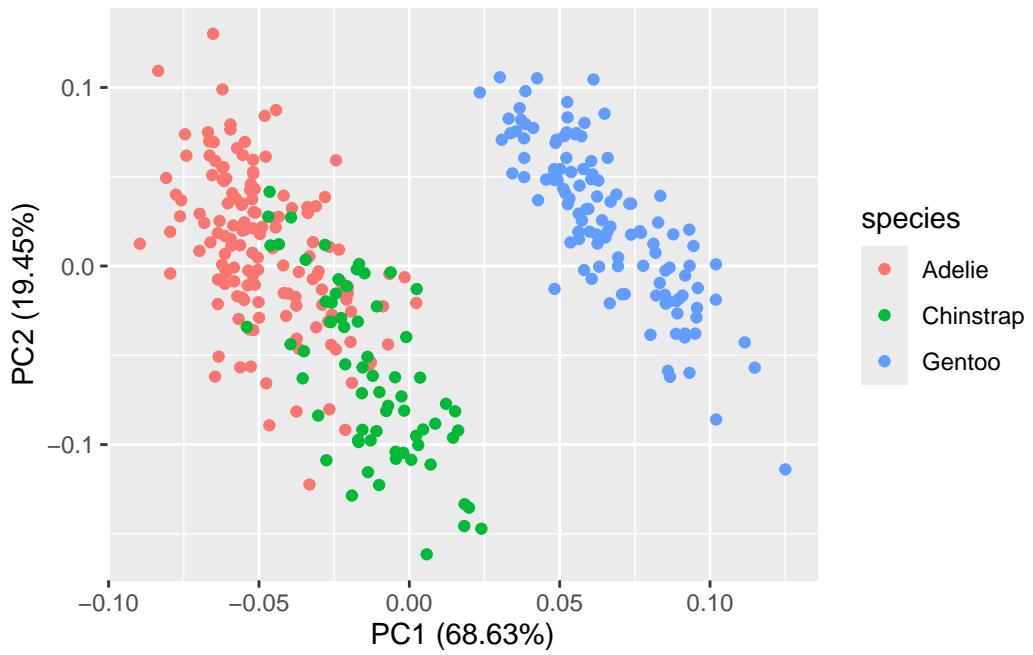
```
biplot(pca_out)
```



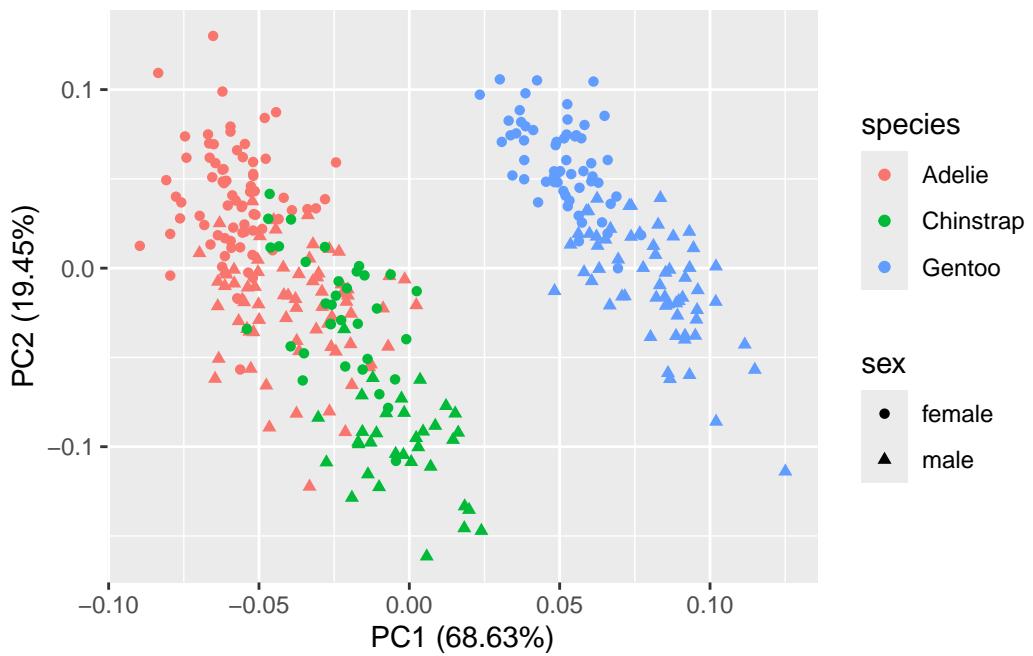
```
autoplots(pca_out)
```



```
autoplot(pca_out, data=rawdata, colour='species')
```

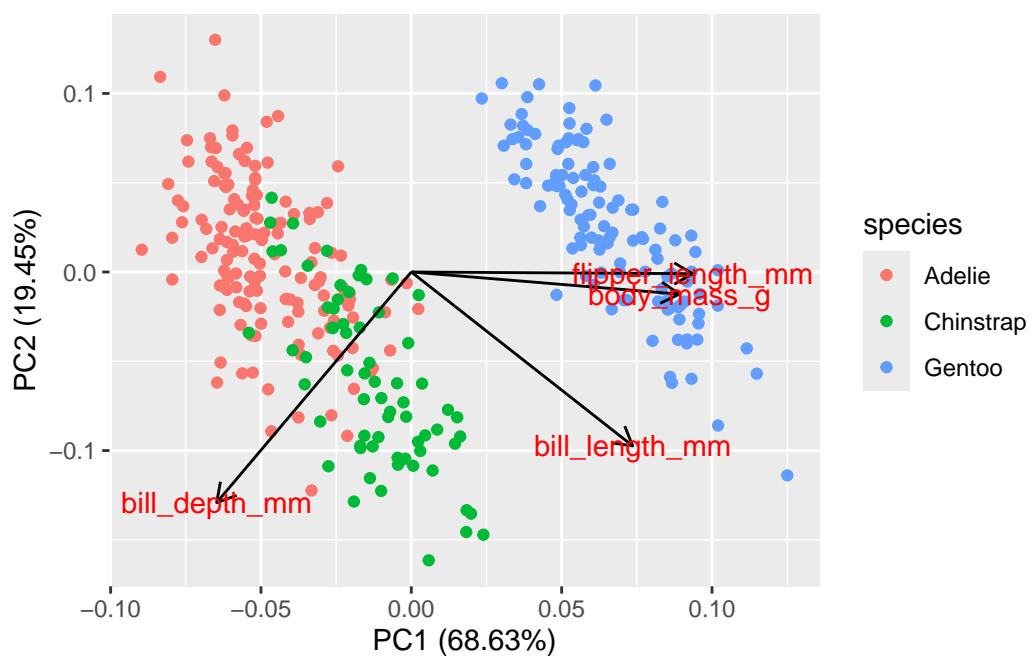


```
autoplot(pca_out, data=rawdata,
         colour='species', shape="sex")
```

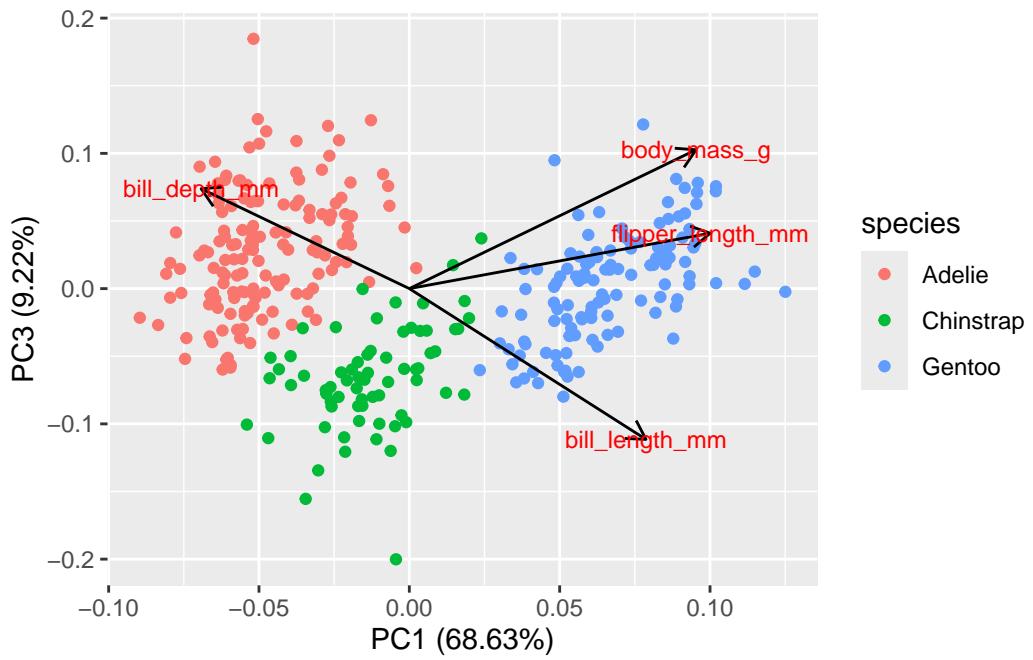


```
autoplot(pca_out, data=rawdata, colour='species',
         loadings = TRUE, loadings.colour = 'black',
```

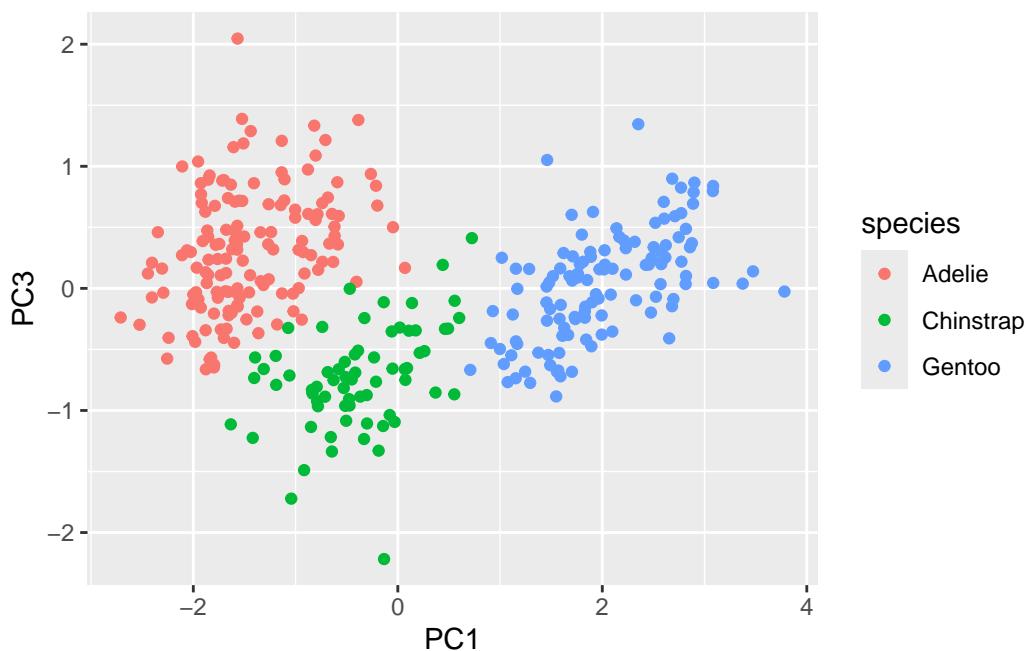
```
loadings.label = TRUE, loadings.label.size = 4)
```



```
#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)
```



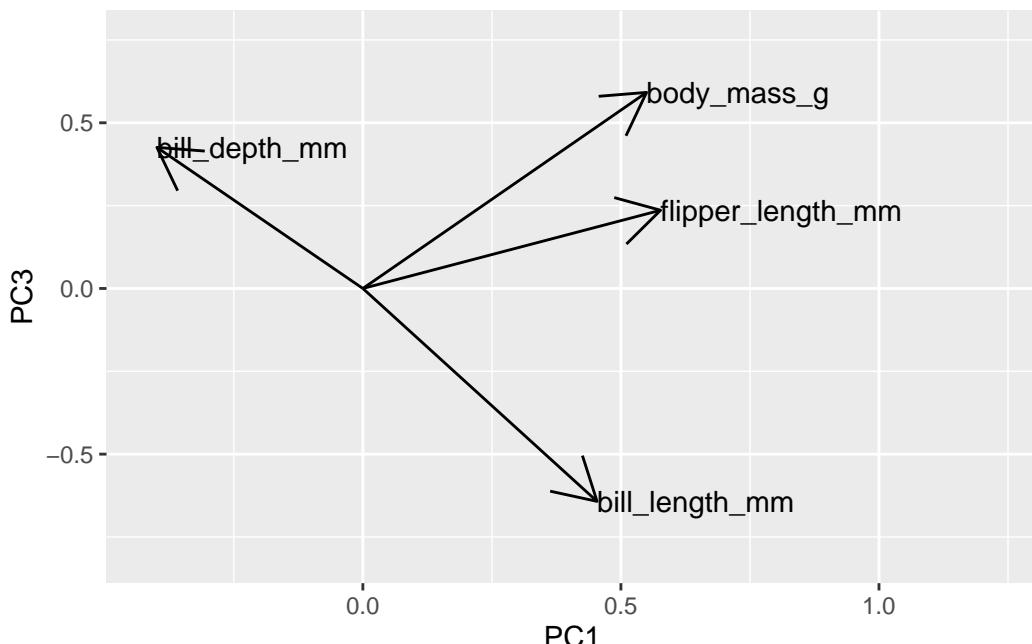
```
pca_out$x |>
  as_tibble() |>
  bind_cols(rawdata) |>
  ggplot(aes(PC1,PC3,color=species))+
  geom_point()
```



```

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)+ 

```

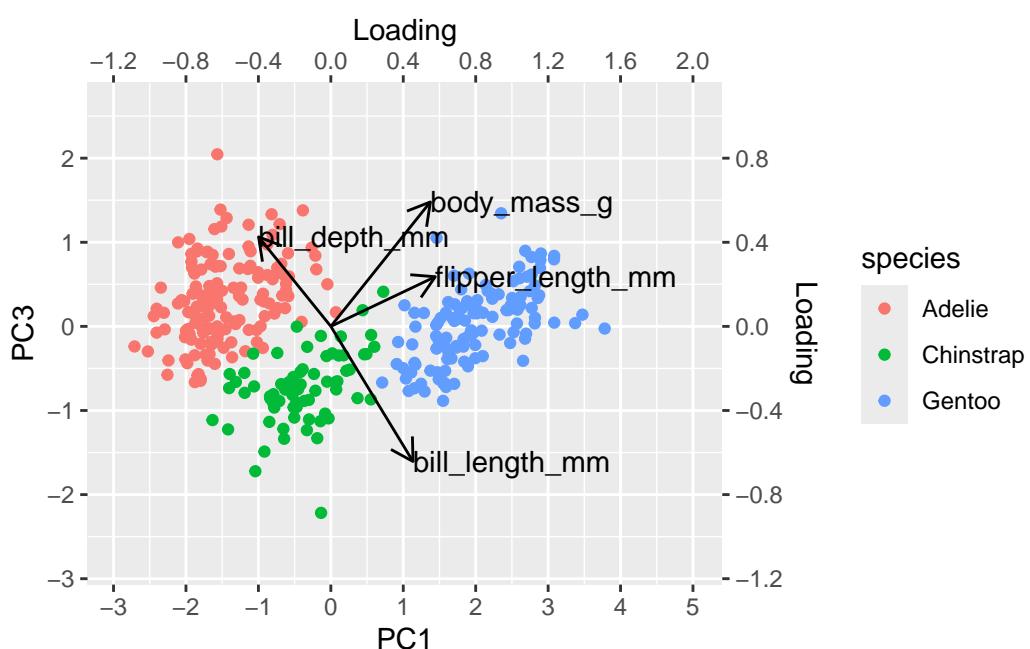
```

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))

```

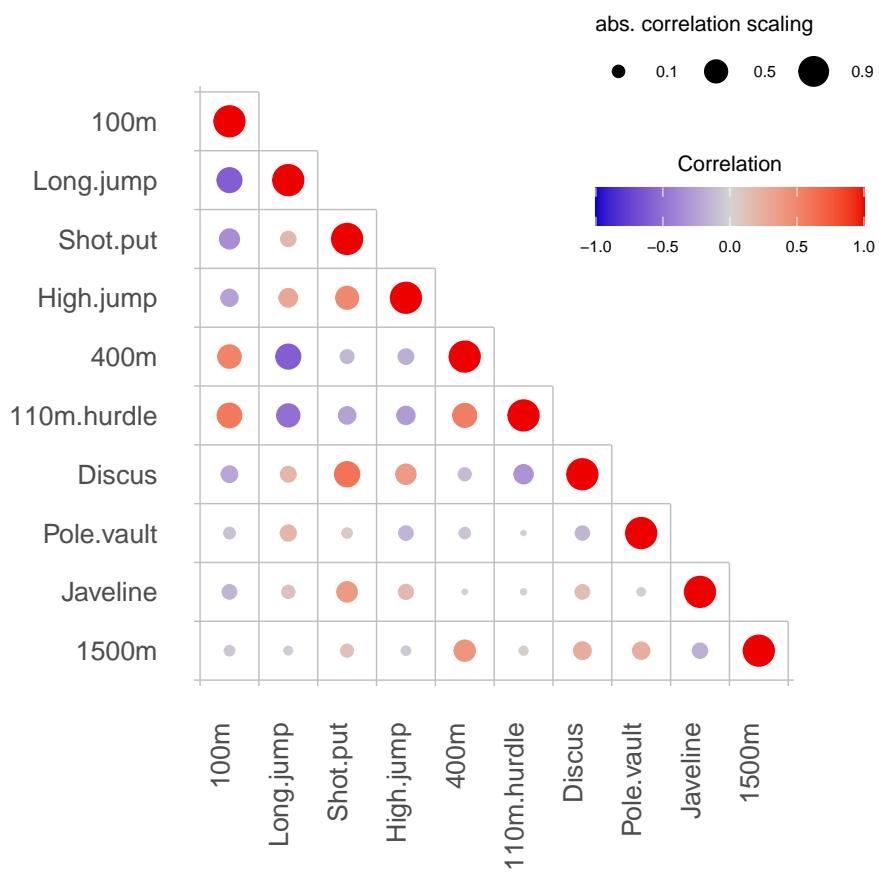


```

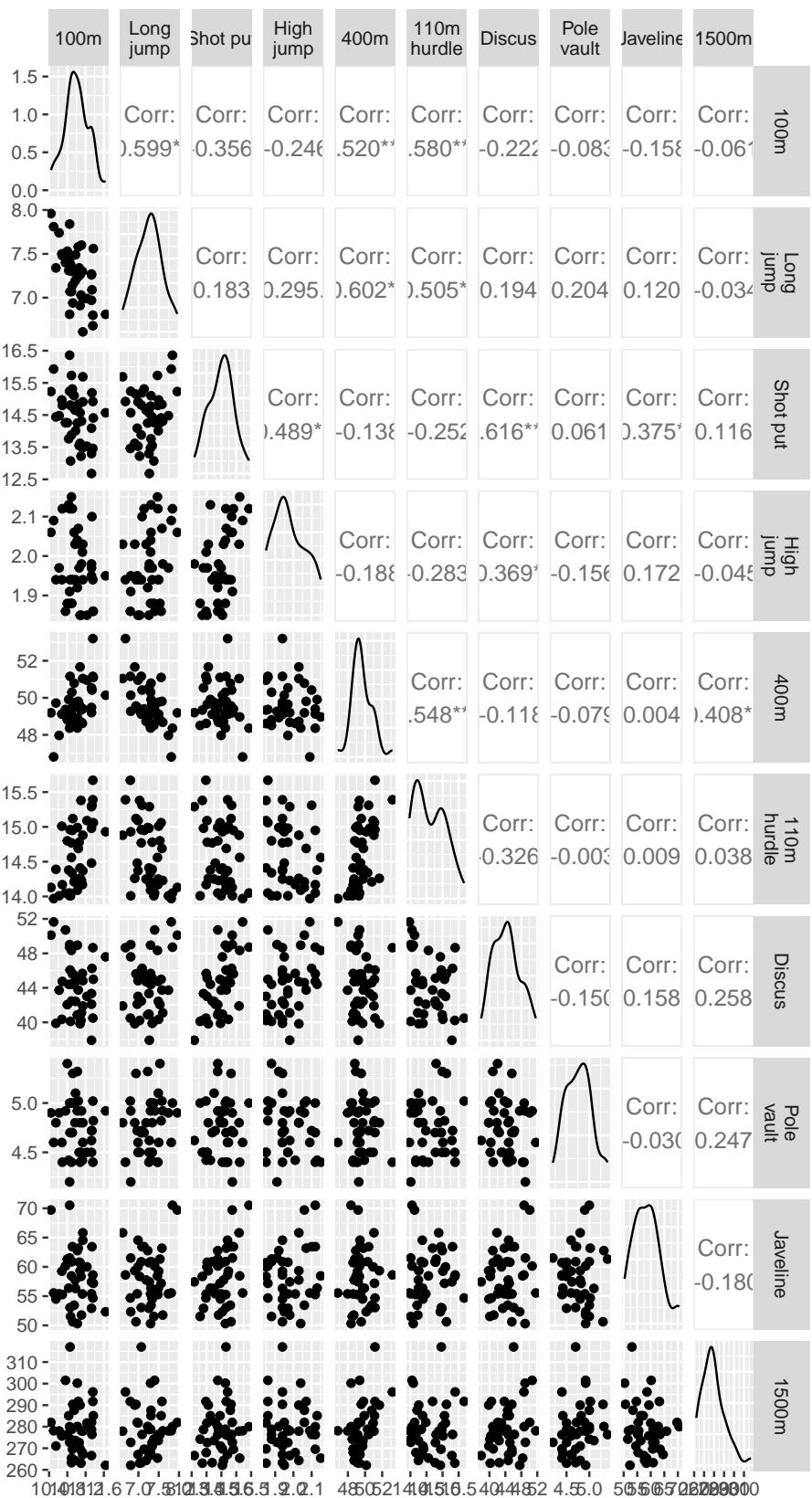
# decathlon
data("decathlon")
cortestR(decathlon |> select(1:10),
         split = T) |>
  pluck('corout') |>

```

```
ggcormat(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)
```

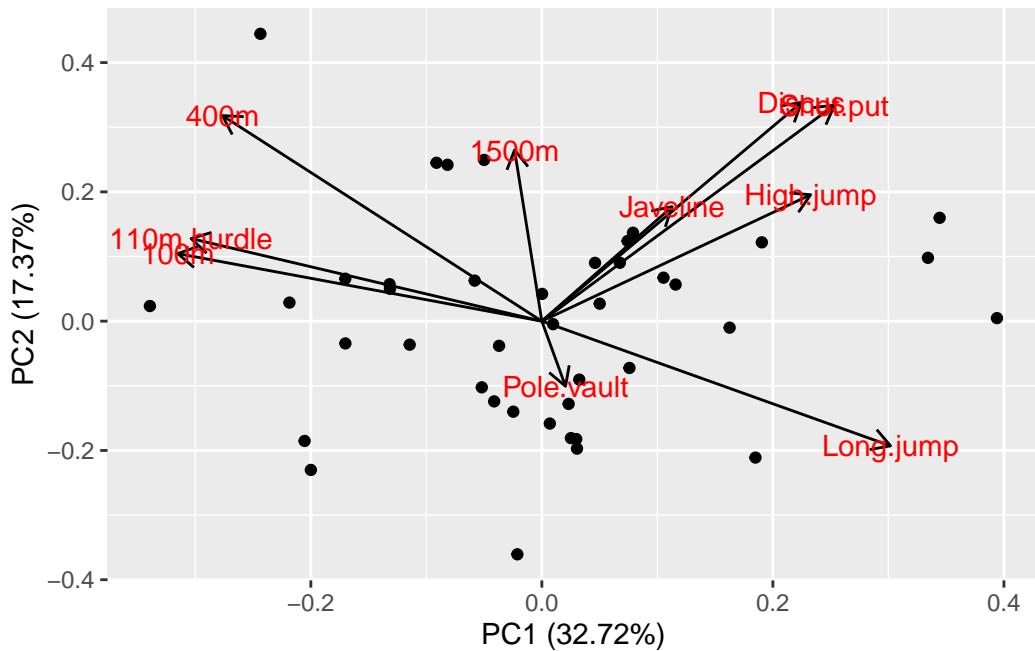
Importance of components:

	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)
```



23.3 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

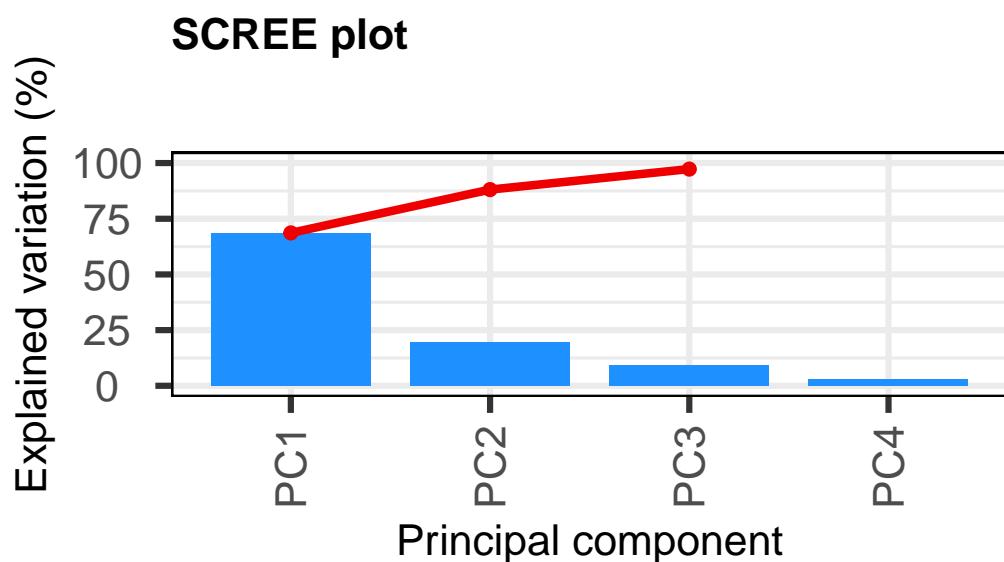
```
getLoadings(pca_out3)
```

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

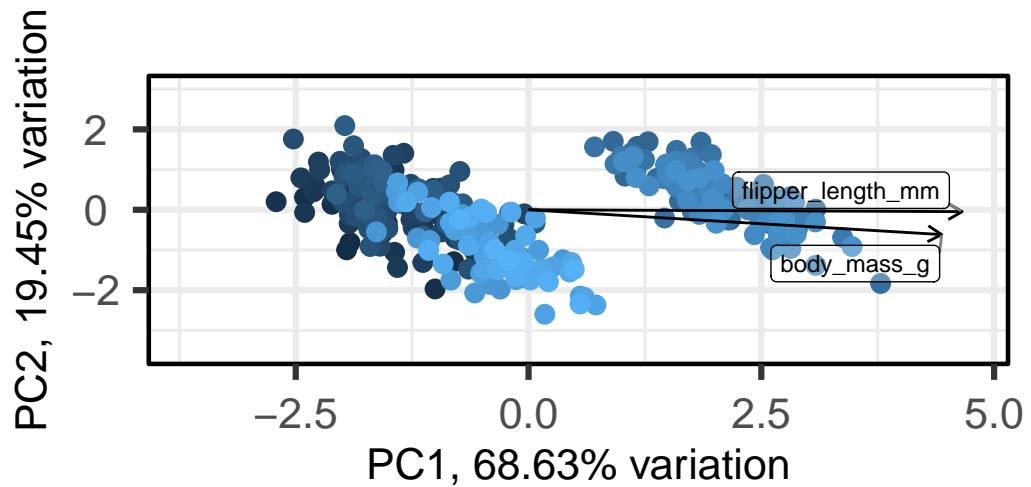
Warning: Removed 1 row containing missing values or values outside the scale range
(`geom_point()`).



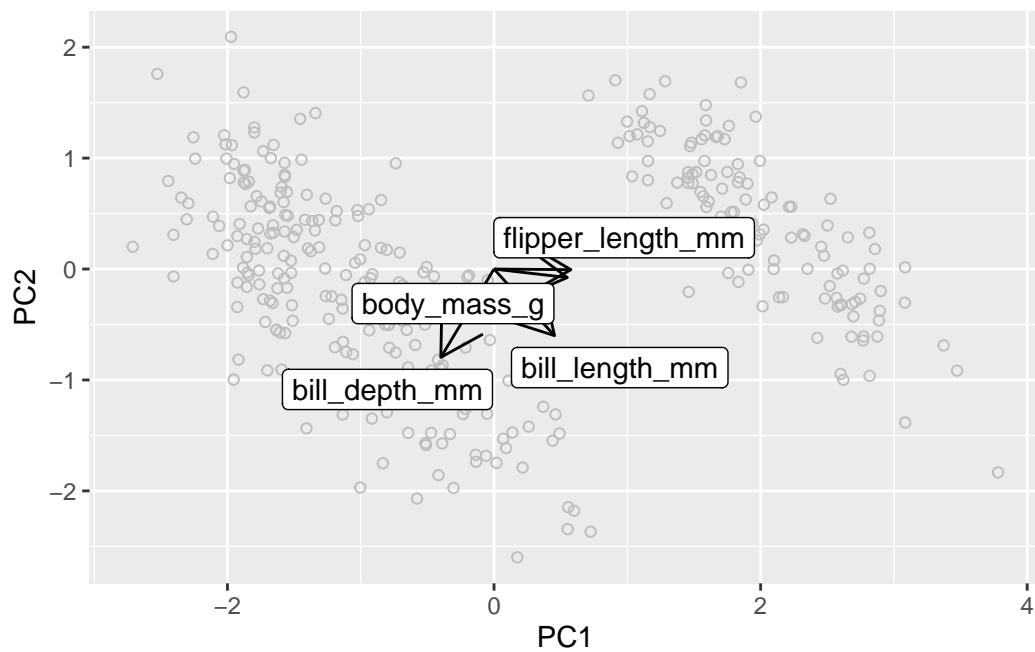
```
PCAtools::biplot(pca_out3,  
                  showLoadings = TRUE, ntopLoadings = 2,  
                  labSize = 5, pointSize = 3, sizeLoadingsNames = 3)
```

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))
```



24 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

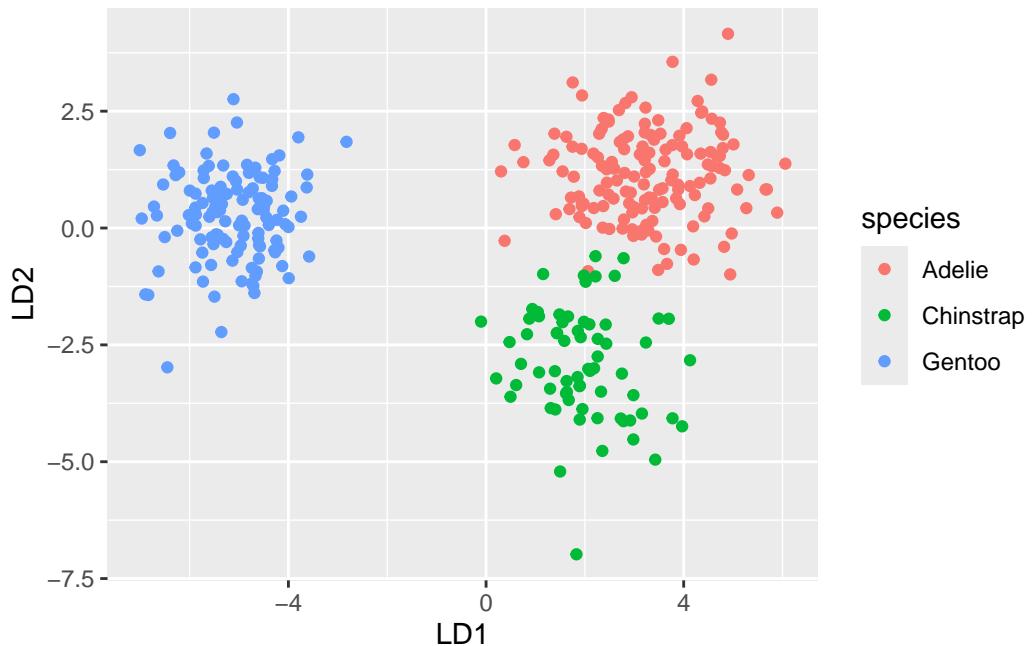
```
lda_out$scaling
```

	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

```

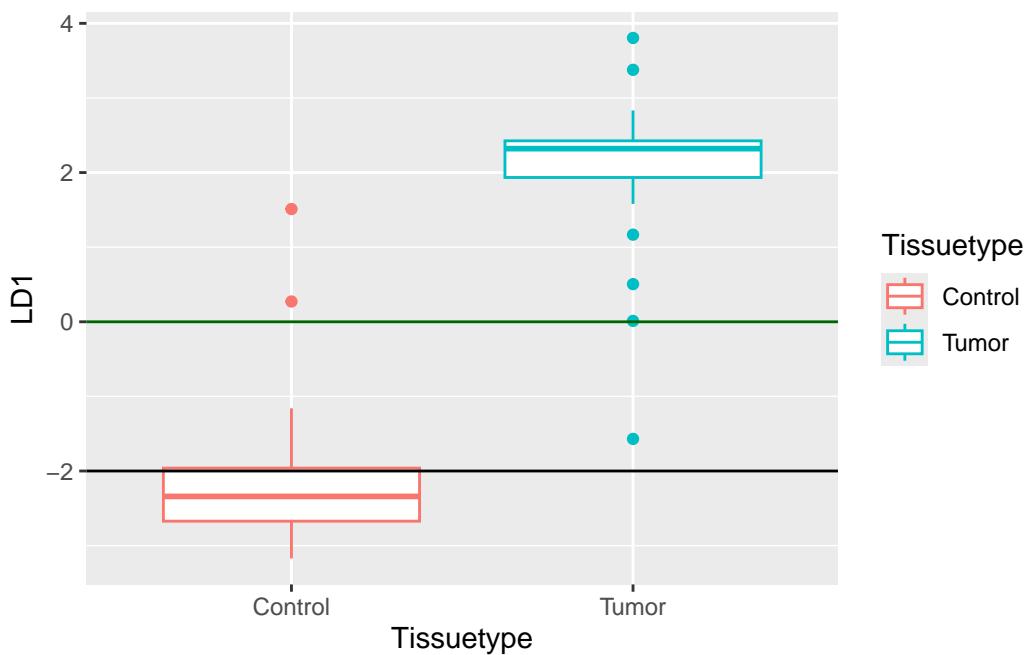
let-7d* -0.028231729
miR-1   -0.153359325
miR-100 -0.015324858

```

```

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

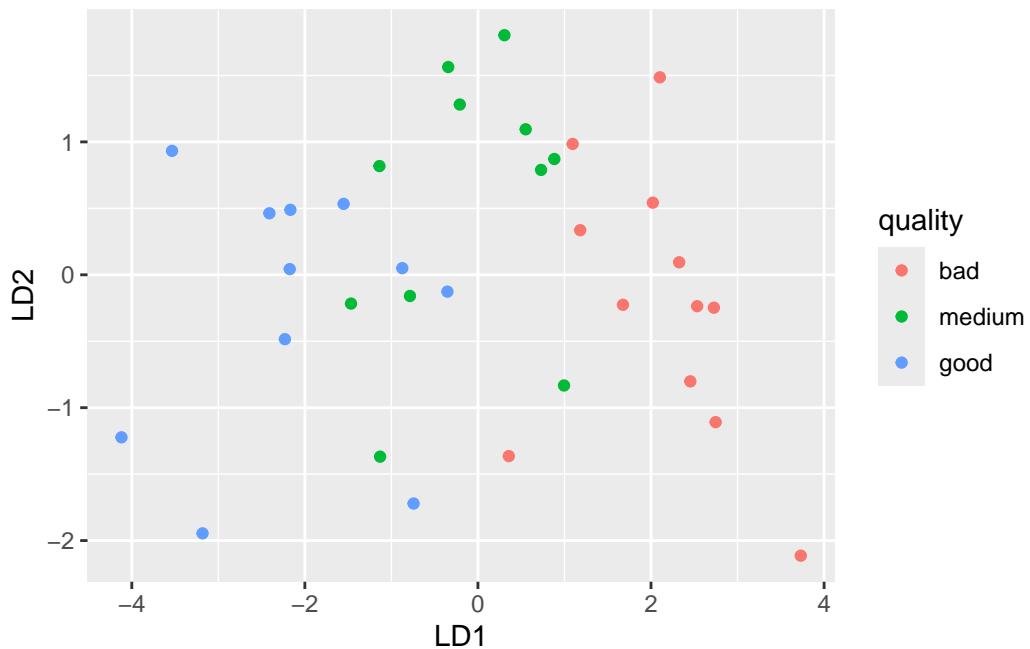
```

```
[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

25 Cluster analysis

25.1 kmeans

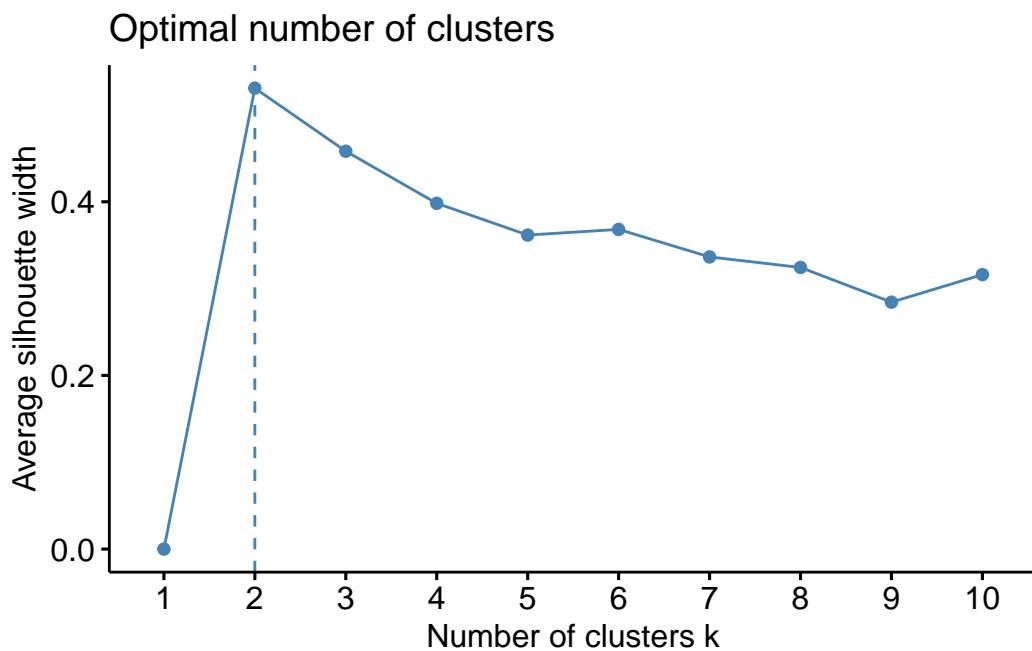
```
pacman::p_load(conflicted, tidyverse,
                 wrappedtools,
                 palmerpenguins,
                 ggfortify, GGally,
                 factoextra,
                 caret, clue,
                 dendextend, circlize,
                 easystats, NbClust, mclust
)

# 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.
```

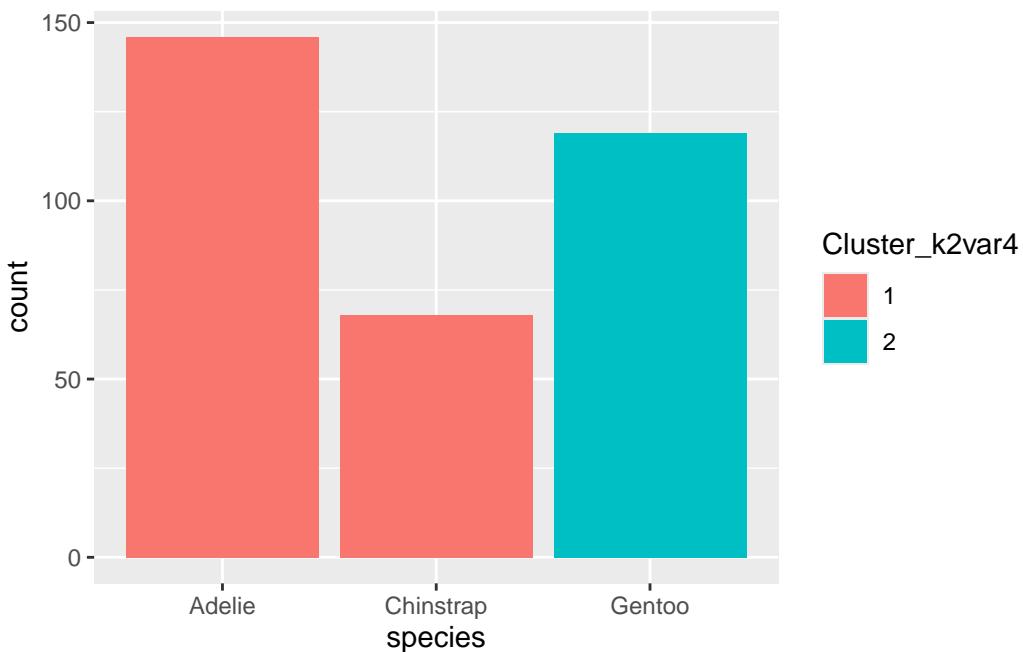
```
rawdata <- penguins |>
  na.omit()
rawdata <- mutate(rawdata,
                  ID=paste('P', 1:nrow(rawdata))) |>
  select(ID, everything())
predvars <- ColSeeker(namepattern = c('_mm', '_g'))

rawdata <- predict(preProcess(rawdata |>
                                select(predvars$names),
                                method = c("center", "scale")),
                    rawdata)
fviz_nbclust(rawdata |> select(predvars$names),
             FUNcluster = kmeans)
```

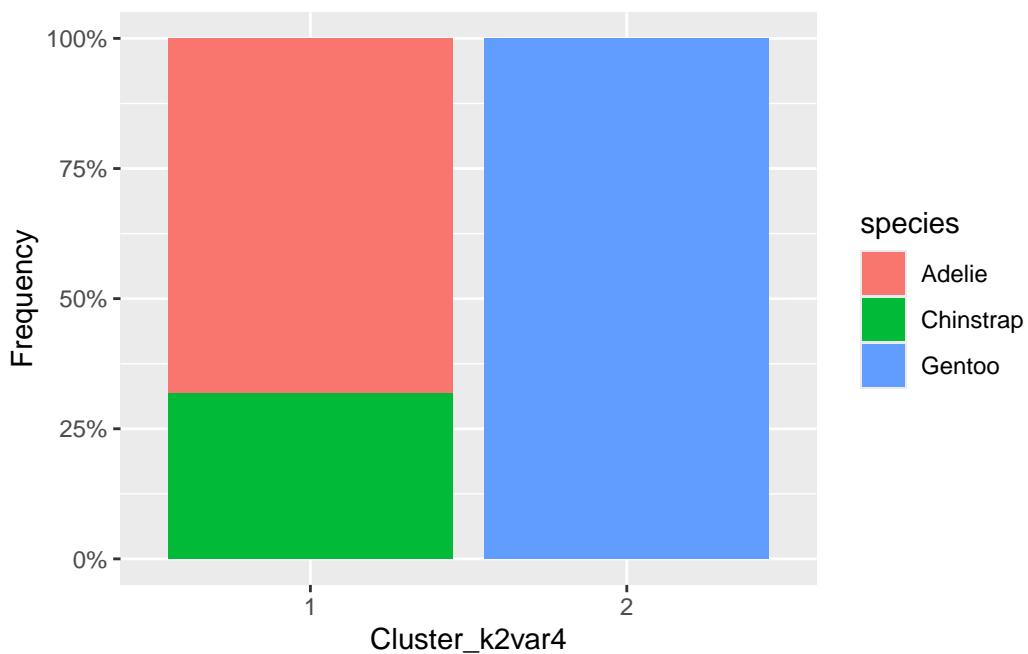


```
kmeans_out <- kmeans(rawdata |> select(predvars$names),
                      centers = 2)

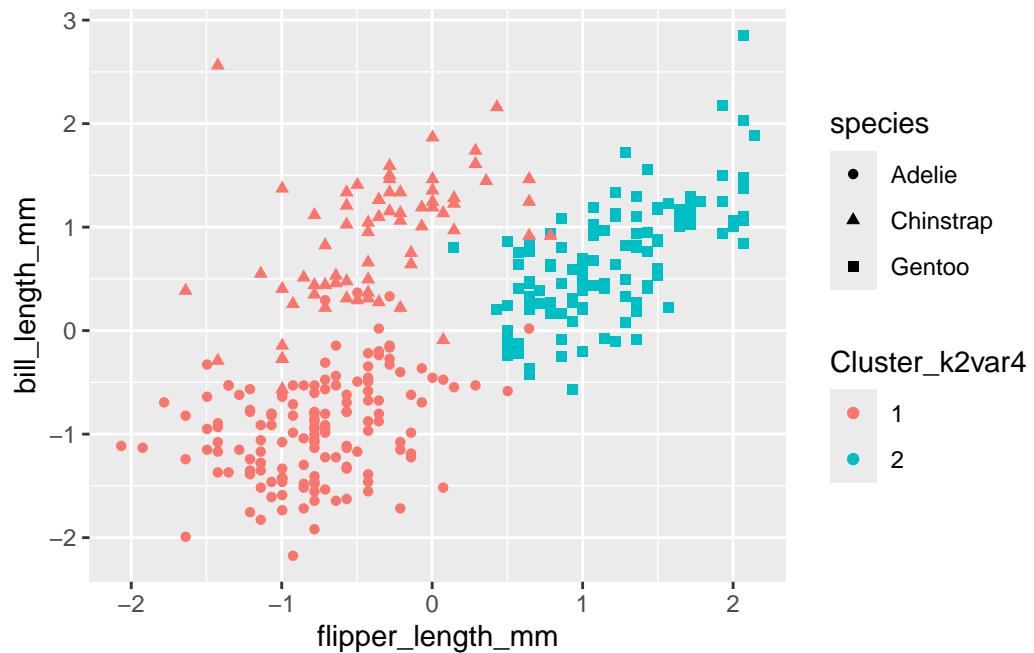
rawdata <-
  mutate(rawdata, Cluster_k2var4=kmeans_out$cluster |> as.factor())
rawdata |>
  ggplot(aes(species, fill=Cluster_k2var4)) +
  geom_bar()
```



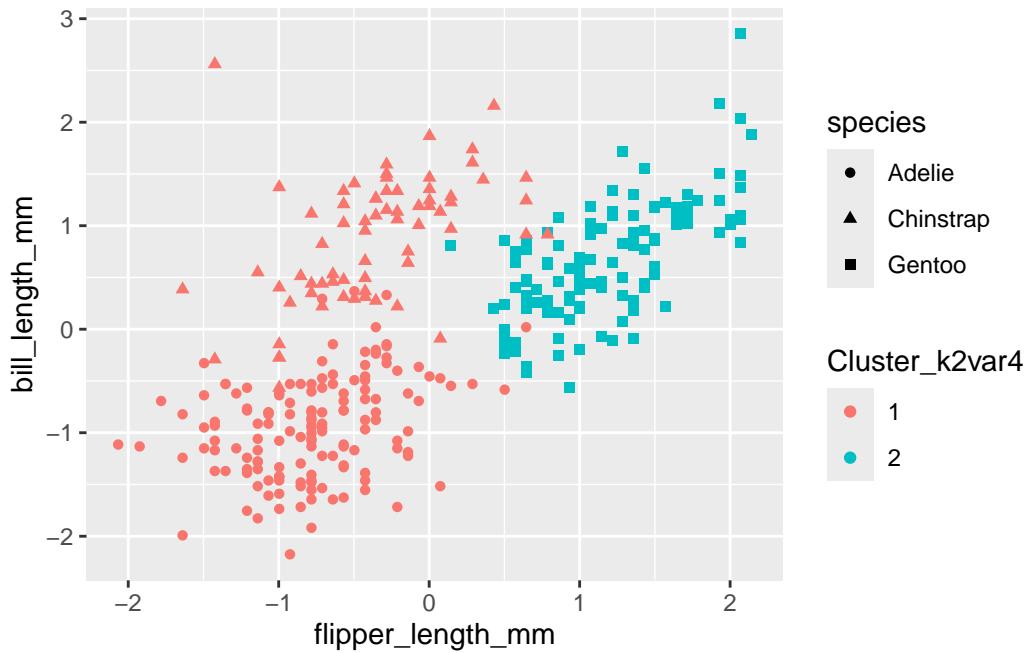
```
rawdata |>
  ggplot(aes(fill=species,x=Cluster_k2var4))+
  geom_bar(position = 'fill')+
  scale_y_continuous(name = 'Frequency', labels=scales::percent)
```



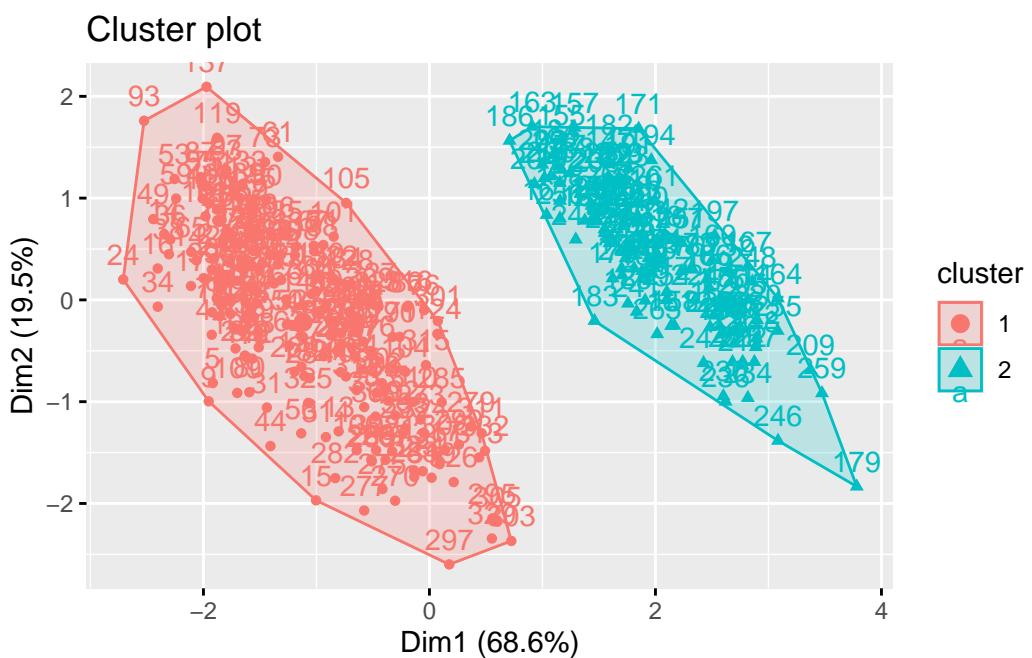
```
rawdata |>
  ggplot(aes(flipper_length_mm,bill_length_mm,
             shape=species,color=Cluster_k2var4))+  
  geom_point()
```



```
rawdata |>
  ggplot(aes(flipper_length_mm,bill_length_mm,
             shape=species,color=Cluster_k2var4))+  
  geom_point()
```



```
fviz_cluster(kmeans_out, rawdata |> select(predvars$names))
```

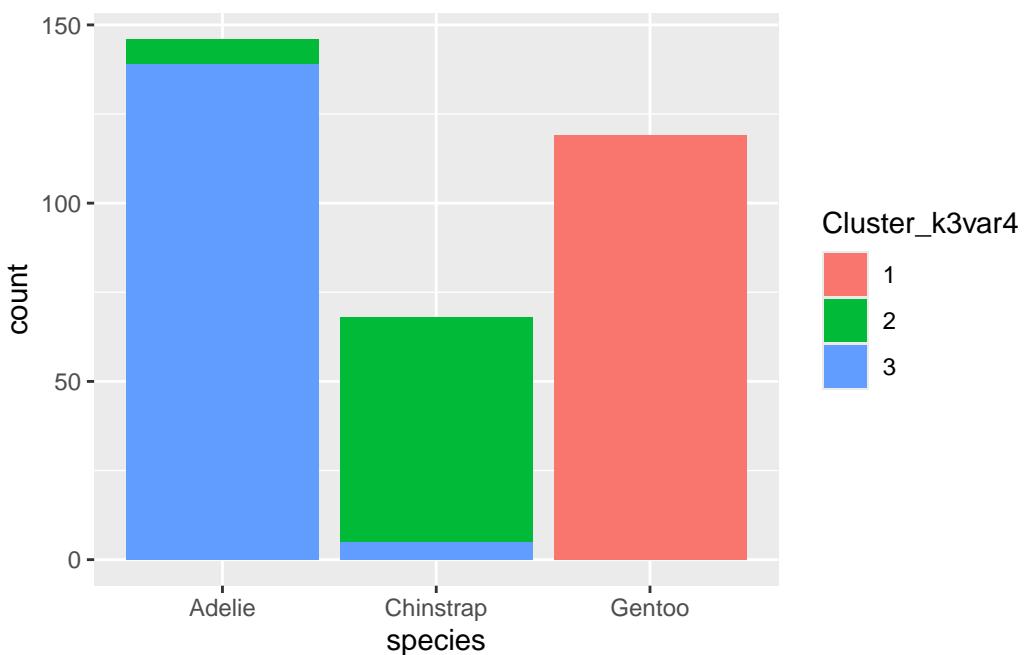


```
# predict(kmeans_out)
sample(clue::cl_predict(kmeans_out), 10)
```

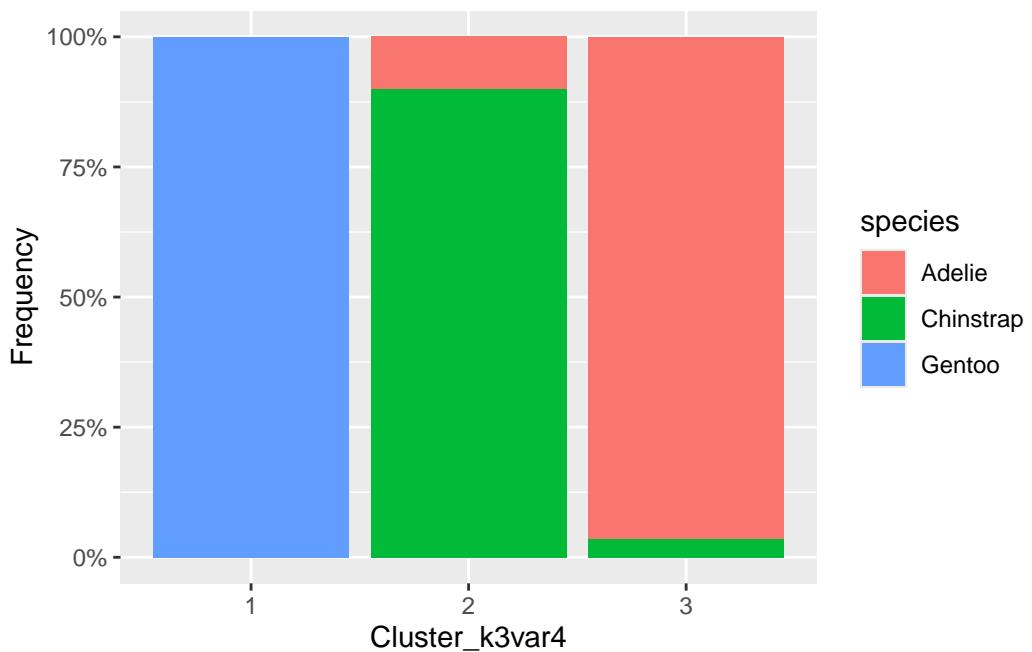
```
[1] 1 1 2 2 2 1 1 1 1 1
```

```
# 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()
```

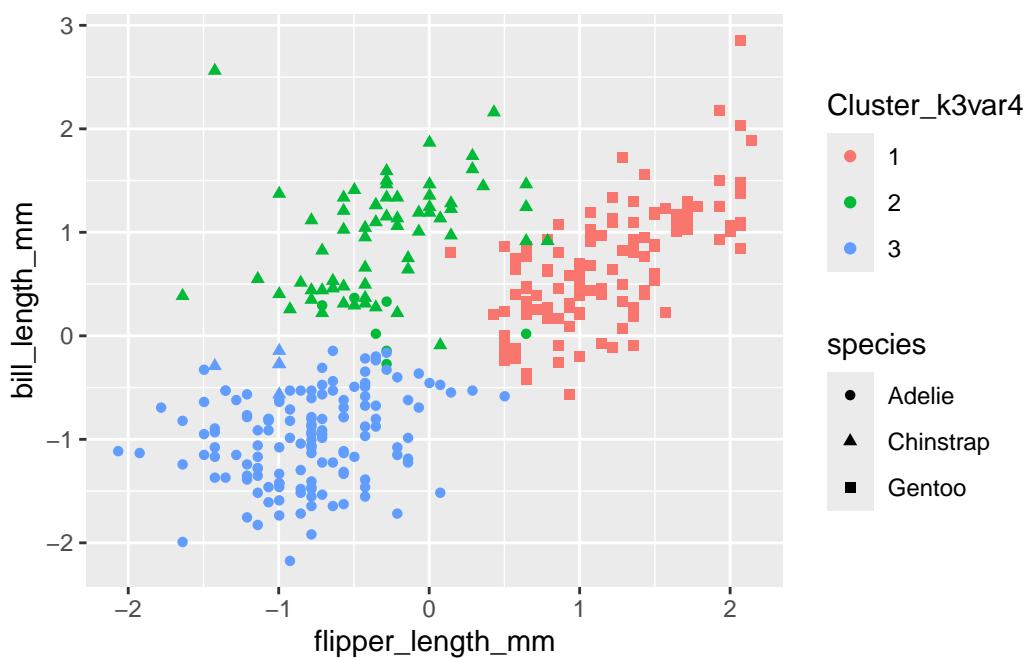


```
rawdata |>
  ggplot(aes(fill=species, x=Cluster_k3var4)) +
  geom_bar(position = 'fill') +
  scale_y_continuous(name = 'Frequency', labels=scales::percent)
```

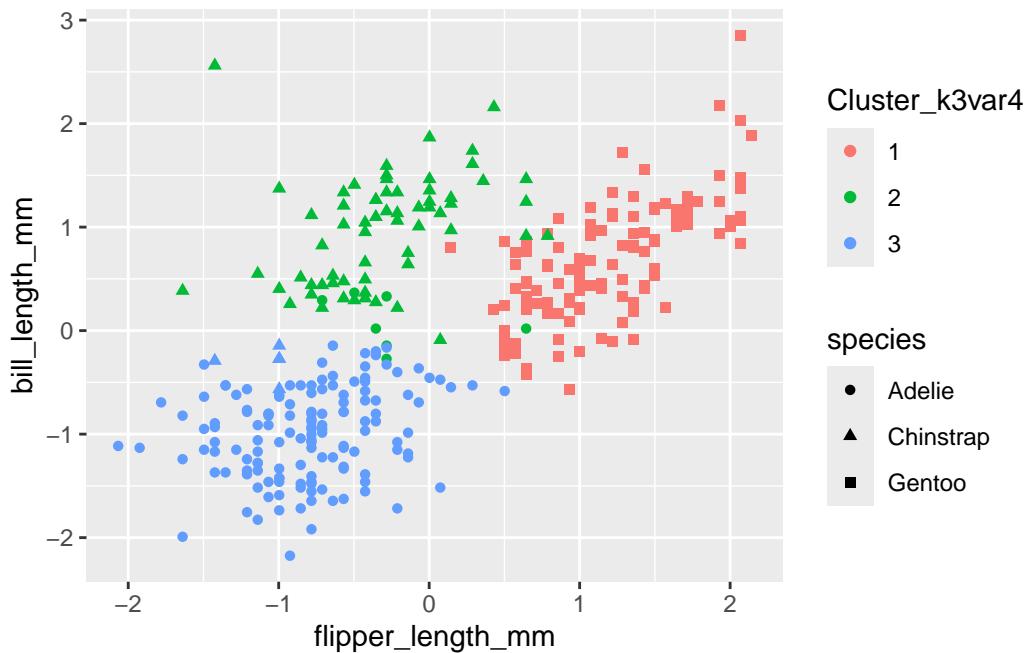


```
rawdata |>
  ggplot(aes(flipper_length_mm,bill_length_mm,
             shape=species,color=Cluster_k3var4))+  

  geom_point()
```

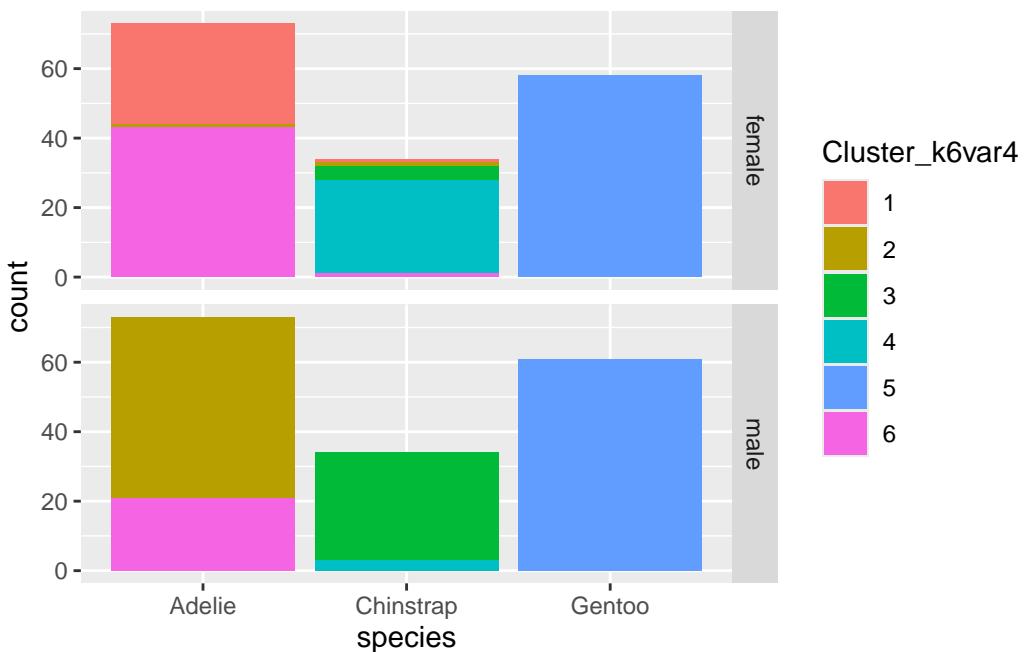


```
rawdata |>
  ggplot(aes(flipper_length_mm,bill_length_mm,
             shape=species,color=Cluster_k3var4))+  
  geom_point()
```

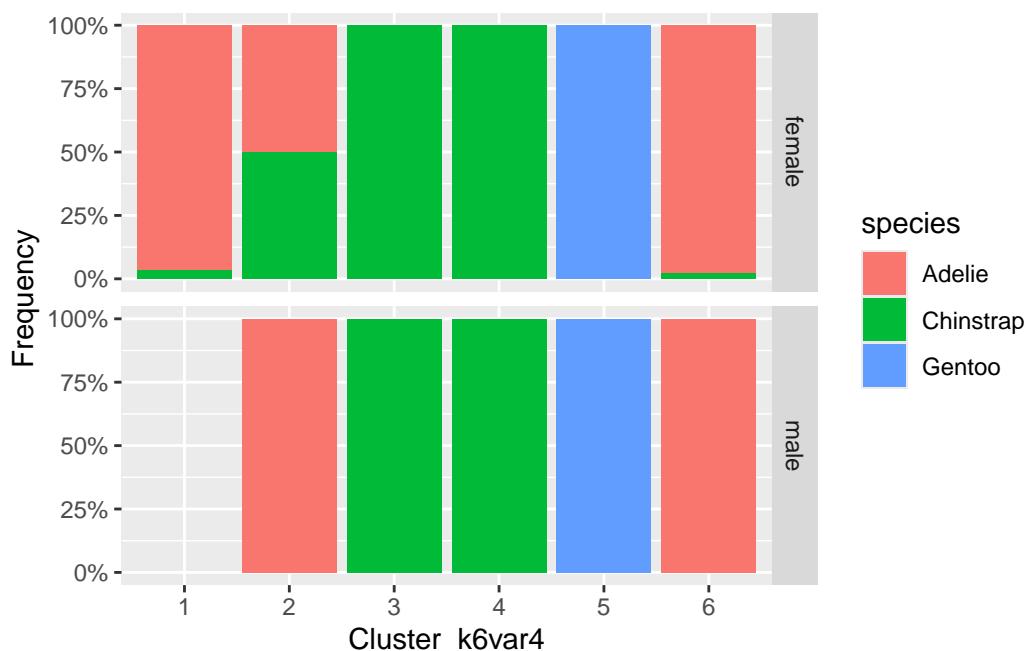


```
kmeans_out6 <- kmeans(rawdata |> select(predvars$names),
                        centers = 6)

rawdata <-
  mutate(rawdata,Cluster_k6var4=kmeans_out6$cluster |> as.factor())
rawdata |>
  ggplot(aes(species,fill=Cluster_k6var4))+  
  geom_bar()+
  facet_grid(rows=vars(sex))
```



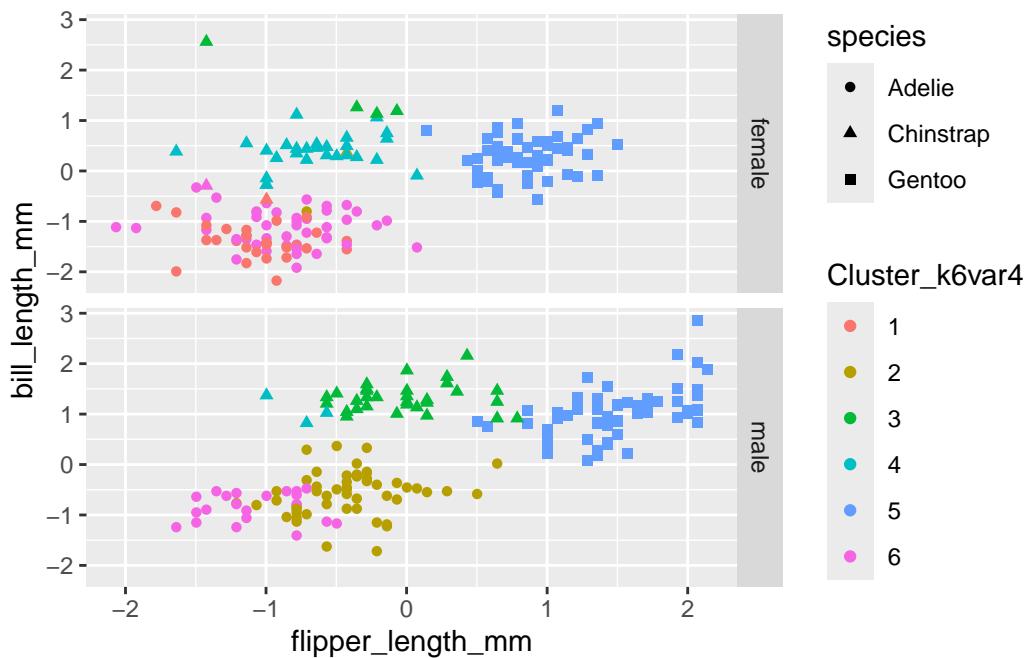
```
rawdata |>
  ggplot(aes(fill=species,x=Cluster_k6var4))+
  geom_bar(position = 'fill')+
  scale_y_continuous(name = 'Frequency', labels=scales::percent)+
  facet_grid(rows=vars(sex))
```



```
rawdata |>
  ggplot(aes(flipper_length_mm,bill_length_mm,
             shape=species,color=Cluster_k6var4))+  

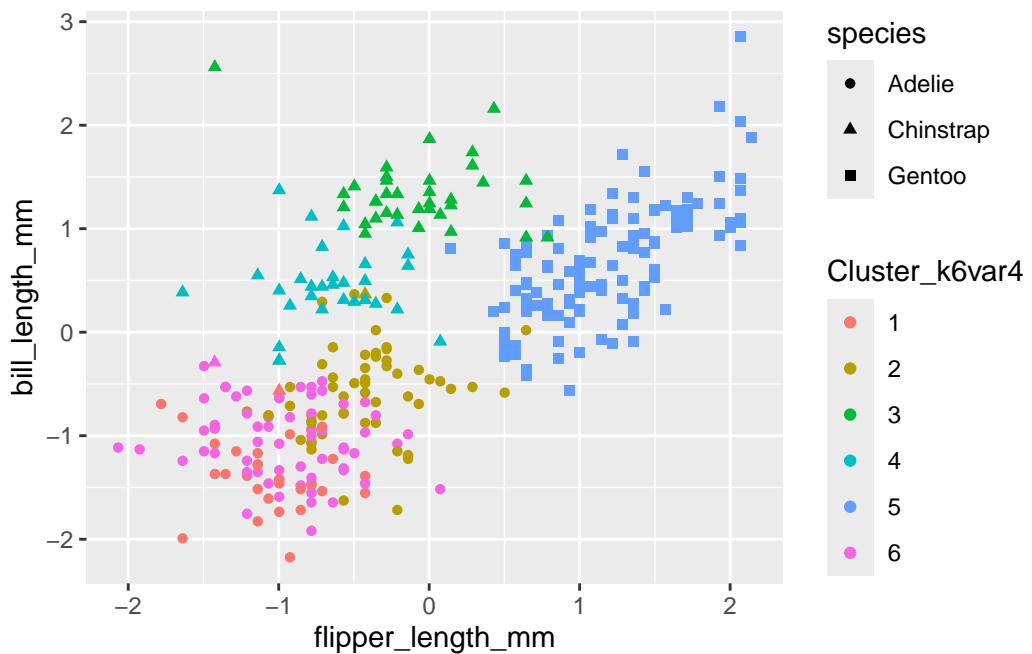
  geom_point() +  

  facet_grid(rows=vars(sex))
```



```
rawdata |>
  ggplot(aes(flipper_length_mm,bill_length_mm,
             shape=species,color=Cluster_k6var4))+  

  geom_point()
```



25.2 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.

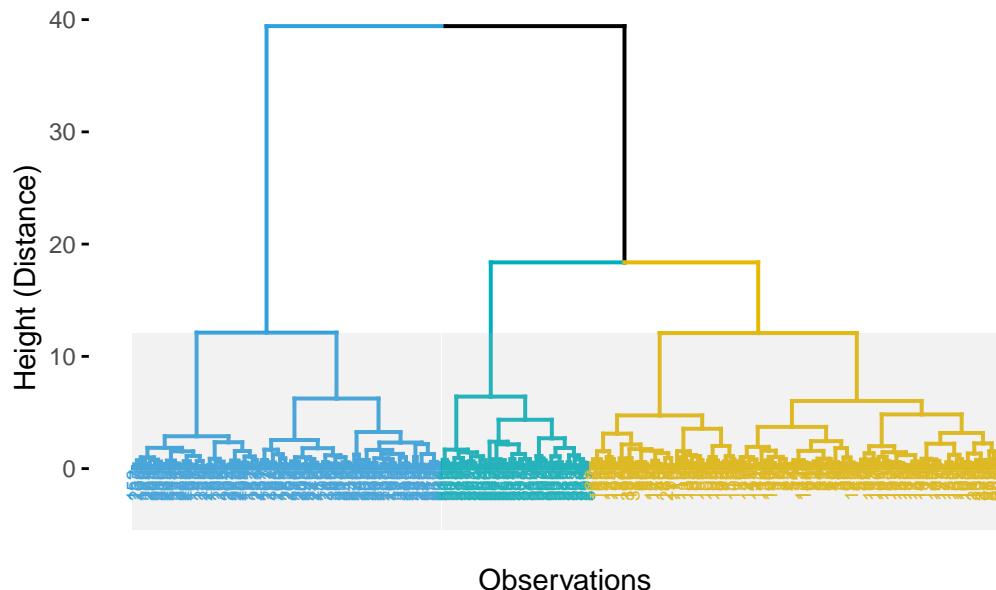
fviz_dend(hc_result,
          k = 3, # Cut the dendrogram into 3 groups (you can try other numbers)
          cex = 0.5, # Adjust label size
          k_colors = c("#2E9FDF", "#00AFBB", "#E7B800"), # Custom colors for clusters
          # color_labels_by_k = TRUE, # Color labels by group
          rect = 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: The `<scale>` argument of `guides()` cannot be `FALSE`. Use "none" instead as of ggplot2 3.3.4.

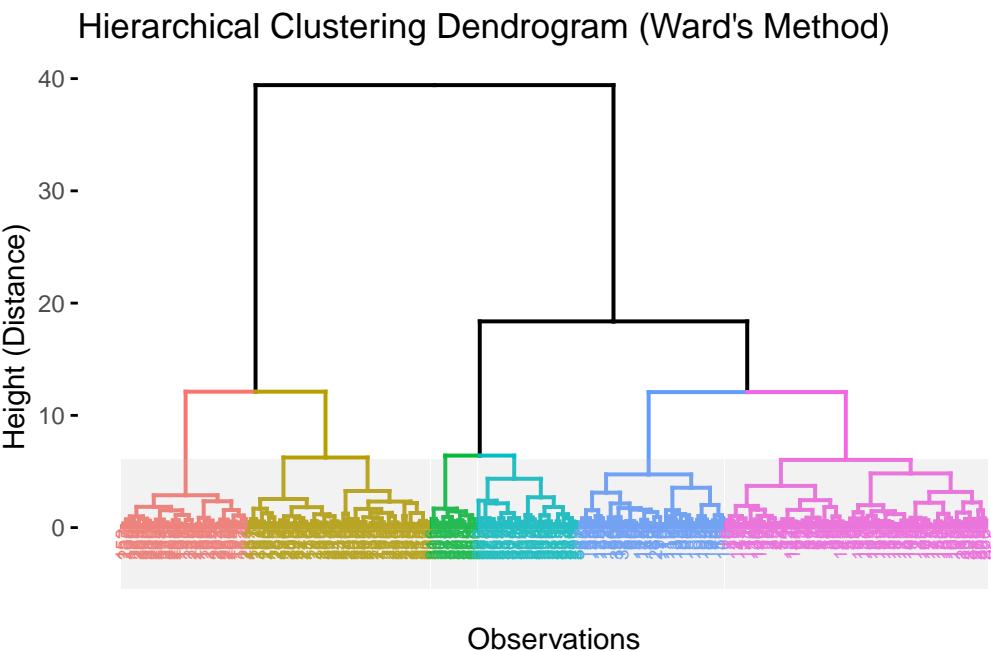
i The deprecated feature was likely used in the factoextra package.

Please report the issue at <https://github.com/kassambara/factoextra/issues>.

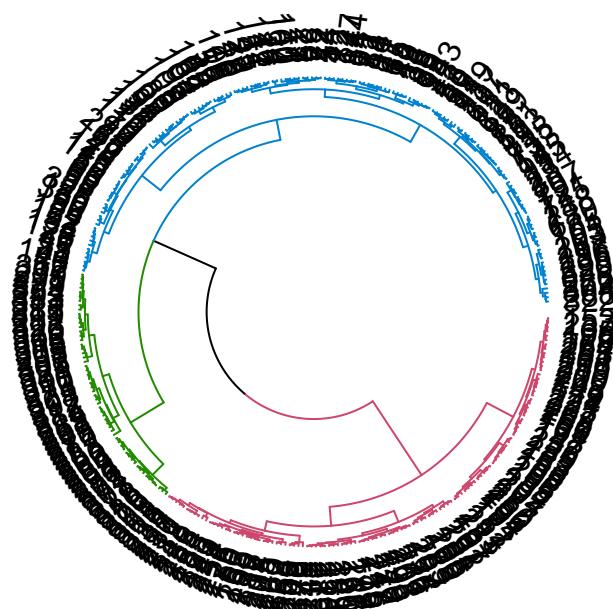
Hierarchical Clustering Dendrogram (Ward's Method)



```
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

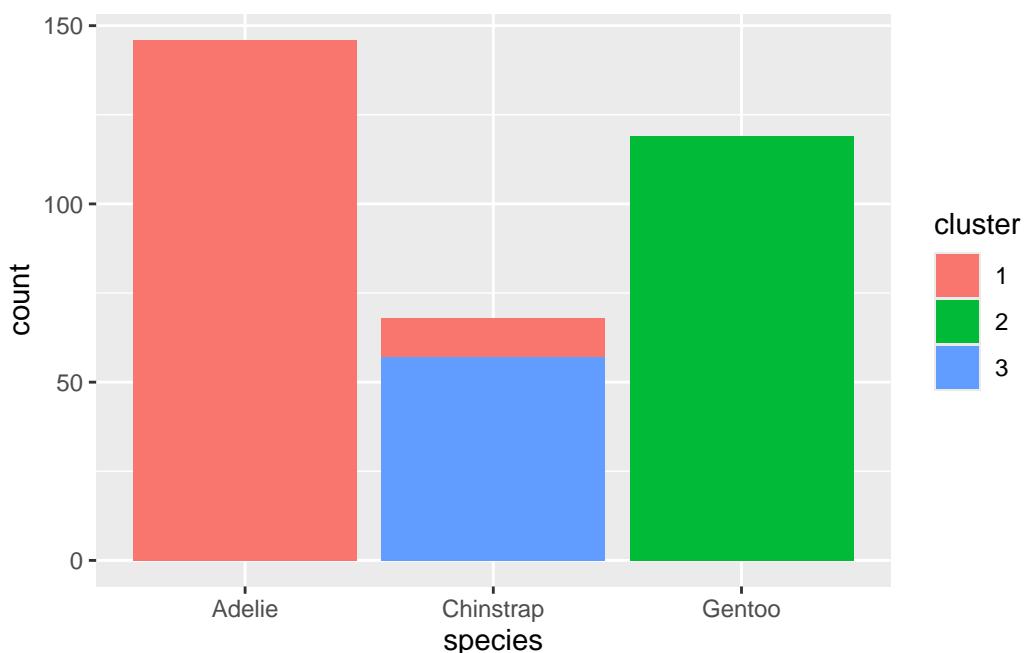
```

# 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()

```



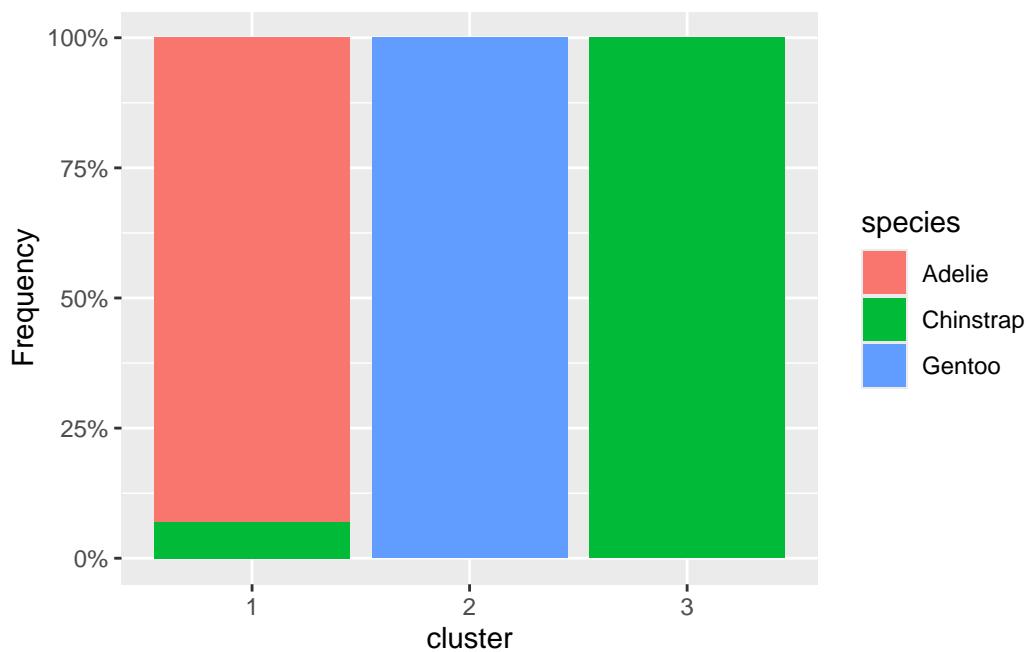
```

rawdata |>
  ggplot(aes(fill=species,x=cluster))+  

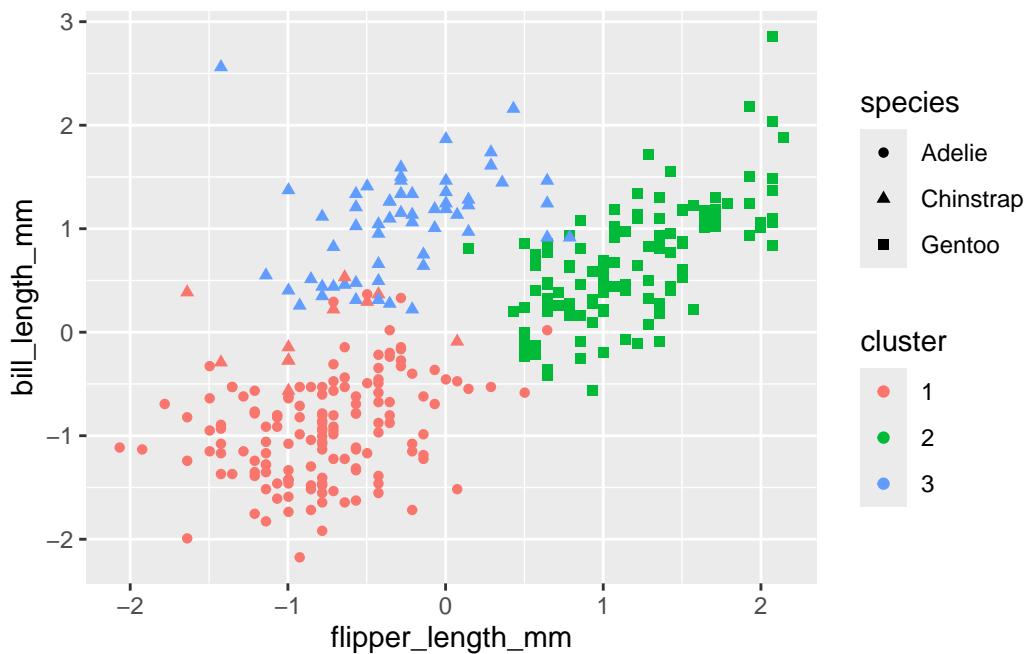
  geom_bar(position = 'fill')+

```

```
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

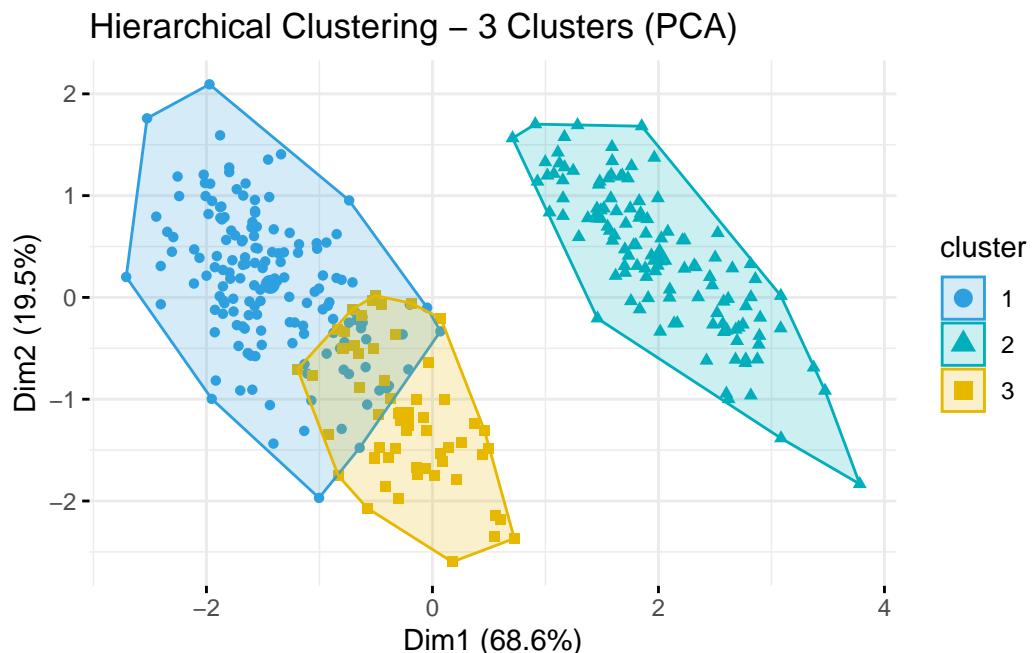
McNemar's Test P-Value : NA

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())
```



25.3 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

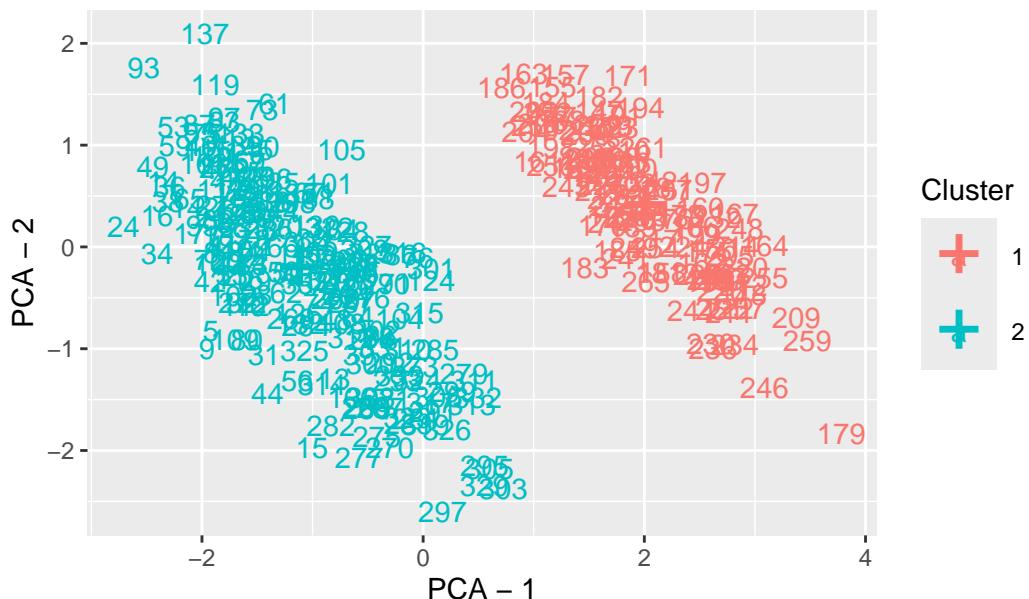
Indices of model performance

Sum_Squares_Total	Sum_Squares_Between	Sum_Squares_Within	R2
1328.000	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
-----
1328.000          |      1113.100 |      214.900 |  0.838

# You can access the predicted clusters via `predict()`.


```

```
plot(easycluster2)
```

