

RStatsbook

Andreas Busjahn

2023-10-07

Table of contents

Preface	4
1 Introduction	5
2 Syntax rules / basic things to know about R	6
2.1 Script preparation / basic setup	6
2.2 Numeric operations	7
2.3 Variables	8
2.3.1 Variable names	8
2.3.2 Basic classes of data	8
2.3.3 Indexing variables	14
2.3.4 Usage of variables	15
2.4 Functions	16
2.4.1 Function usage	16
2.4.2 Functions combined	18
2.4.3 Writing functions	20
2.5 More complex data types, created by functions	22
2.5.1 Matrix	22
2.5.2 Data frame	24
2.5.3 Tibble	27
2.5.4 List	38
2.6 Control structures	41
2.6.1 Loops	41
2.6.2 Conditions	44
3 Importing data	47
3.1 Import from text files (.txt, .csv)	47
3.2 Import from Excel	47
3.2.1 Tidy Excel files	47
3.2.2 Excel file with units row	47
3.3 Import from SPSS	49
3.4 Import from SAS	49
4 Changing structure wide <-> long	50
5 Grouping of variables by type / distribution / use	54
5.1 Test for Normal distribution	54
5.1.1 Testing a single variable	54
5.1.2 Testing several variables	57

5.2	Picking column names and positions	59
6	Visualize data with ggplot	61
6.1	Example data	61
6.2	Basic structure of a ggplot call	62
6.3	fill vs. color	66
6.4	Color systems	68
6.4.1	External color definitions from ggsci	71
6.5	Exporting ggplots	73
6.6	Other geoms	73
6.7	Combining and finetuning aesthetics	82
6.8	Positioning elements	90
6.9	Order of layers	95
6.10	Local aesthetics for layers	100
6.11	Faceting (splitting) plots	102
6.11.1	facet_grid	102
6.11.2	facet_wrap	105
6.11.3	Controlling scales in facets (default: scales="fixed")	108
6.12	Showing summaries	110
6.13	Indicating significances	115
6.14	Theme definitions / changes	118
6.15	Combining figures with patchwork	124
7	Descriptive statistics	127
8	Better tables with flextable	133
9	Simple test statistics	134
9.1	Quantitative measures with Gaussian distribution	134
9.2	Ordinal data	140
9.3	Categorial data	143
10	Intro to lm	147
10.1	Setup	147
10.2	Import / Preparation	147
10.3	Graphical exploration	148
10.4	Linear Models	152
10.4.1	Linear regression	152
10.4.2	ANOVA	157
10.4.3	LM with continuous AND categorical IV	162
10.4.4	Model exploration with package performance	168
11	Summary	170
References		171

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

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](#)

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.

2 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.

2.1 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.

```
Sys.setenv(LANG="en_EN.UTF-8") # to get errors/warnings in English
if(!requireNamespace("pacman", quietly = TRUE)){
  install.packages("pacman")
}
pacman::p_load(
  conflicted, # tests/solutions for name conflicts
  tidyverse, # metapackage
  wrappedtools, # my own tools package
  randomNames # used to create pseudo names
)
conflicts_prefer(dplyr::filter)
```

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

2.2 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
```

2.3 Variables

2.3.1 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
```

2.3.2 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.

2.3.2.1 Guessed classes

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

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

```
typeof(float_num)
```

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

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

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

```
typeof(int_num)
```

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

```
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
```

Factor: categorical variables with limited set of distinct values, internally stored as integers.
Everything intended to group subjects or representing categories should be stored as factor.
Packageforcats 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"
```

```
# 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
```

Dates / Time:

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

```
[1] "2023-07-28"
```

```
class(date_var)
```

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

```
typeof(date_var)
```

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

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

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

Mixed classes:

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

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

```
test2
```

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

2.3.2.2 Forcing / casting classes

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 NA NA NA NA NA 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"

# test_NA_factor <-
#   factor(rep(NA,10),
#         levels=c("WT","Mutation1", "Mutation2"))
# class(test_NA_factor)
```

2.3.3 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
```

2.3.4 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
```

2.4 Functions

2.4.1 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(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()
```

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

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

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

2.4.2 Functions combined

Functions often just solve one problem or task, so usually we need to combine them. This can be done by nesting or piping. Piping makes reading/understanding scripts easier, as it shows order of functions:

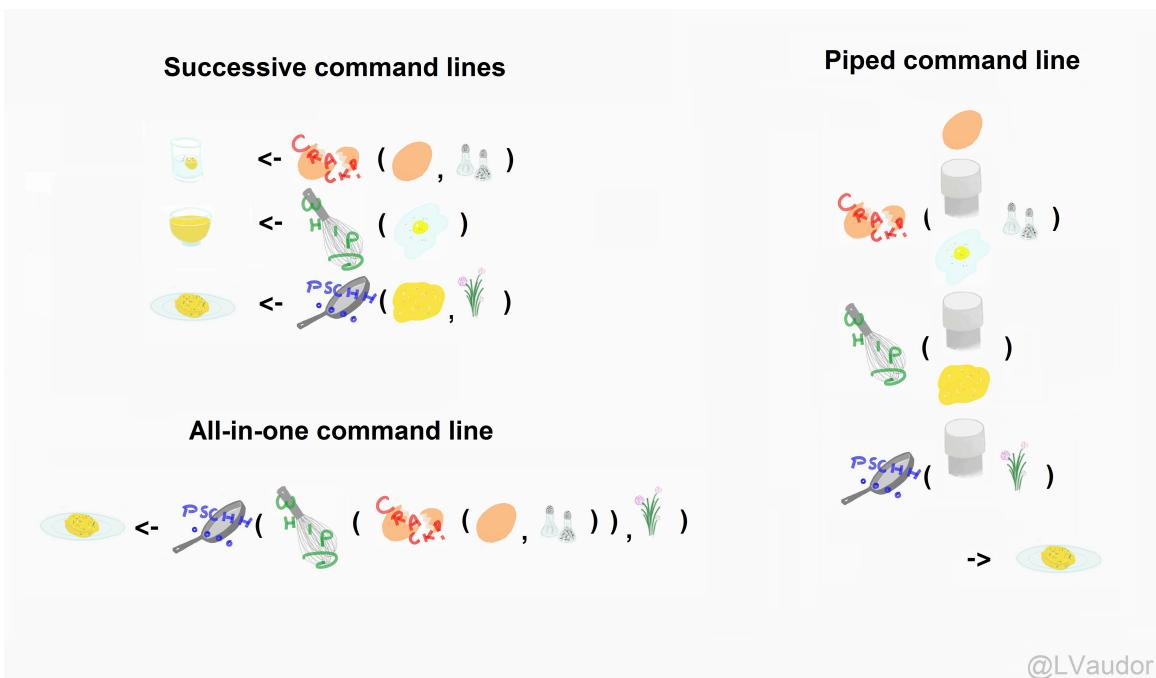


Figure 2.1: Piping functions

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

[1] 37

```
# or (usually better) piped:
mtcars |>
  mutate(am=factor(am)) |> # %>%
  filter(vs==1) |>
  group_by(am) |>
  summarize(across(.cols = c(wt, mpg, qsec, disp),
    .fns = mean)) |>
  pivot_longer(cols = -am, names_to = "Measure") |>
  pivot_wider(id_cols = Measure, names_from = am,
    values_from = value)
```

A tibble: 4 x 3

```

Measure      `0`   `1`
<chr>     <dbl> <dbl>
1 wt        3.19  2.03
2 mpg       20.7   28.4
3 qsec      20.0   18.7
4 disp      175.   89.8

```

If a sequence of functions is used often, combining them into a new function is advisable, e.g. this 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
<fct>   <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

```

2.4.3 Writing functions

```

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

```

```
[1] 64
```

```

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

```

```

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")
```

2.5 More complex data types, created by functions

2.5.1 Matrix

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

2.5.1.1 Creation

```
my1.Matrix<-
  matrix(data=1:12,
        # nrow=4, # this is not needed, can be derived from data
        ncol=3,
        byrow=TRUE, # date 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 <- 10
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 col6 col7 col8 col9 col10
row1    1    11   21   31   41   51   61   71   81   91
row2    2    12   22   32   42   52   62   72   82   92
row3    3    13   23   33   43   53   63   73   83   93
row4    4    14   24   34   44   54   64   74   84   94
row5    5    15   25   35   45   55   65   75   85   95
row6    6    16   26   36   46   56   66   76   86   96

```

```

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

```

2.5.1.2 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
```

2.5.2 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).

2.5.2.1 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

2.5.2.2 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$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)
```

2.5.3 Tibble

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

2.5.3.1 Creation

```

patientN <- 25
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
#> # ... with 15 more rows
#> # ... with 2 more variables: `diaRR (mmHg)` <dbl>, HR <dbl>
#>
#> # i 15 more rows
#> # i 2 more variables: `sysRR (mmHg)` <dbl>, Treatment <chr>
```

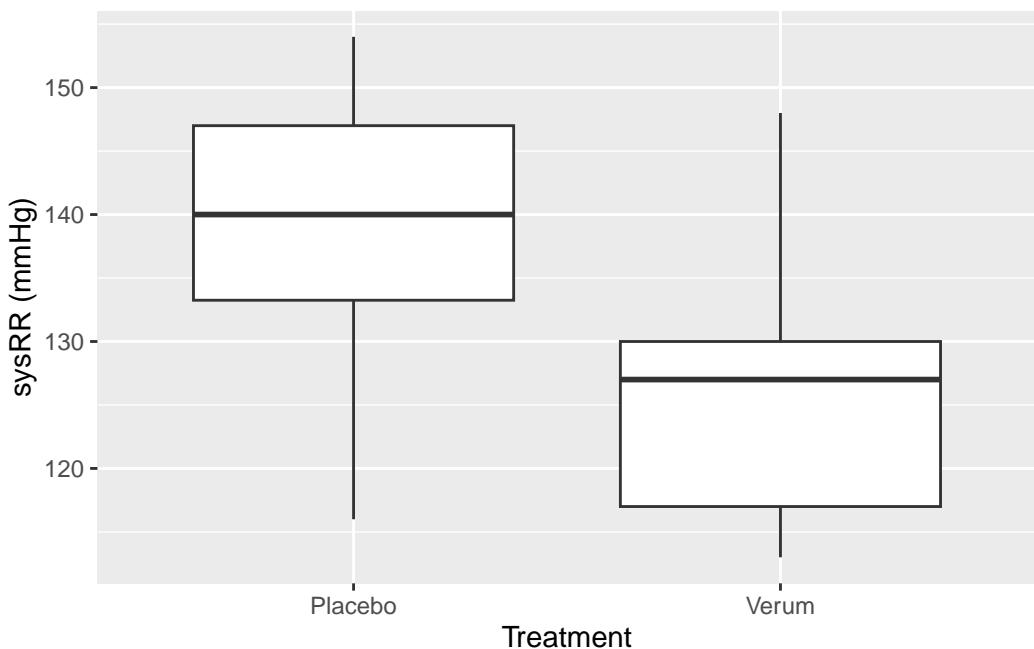
```
colnames(rawdata)
```

```
[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()
```



2.5.3.2 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   female
4 P 4   female
5 P 5   male
```

```
rawdata[,6]
```

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

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

```
rawdata[6]
```

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

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

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

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

```
[1] "Roy"          "Martin"        "Fisher"        "el-Karimi"
[5] "al-Bey"       "Melott"        "Burgess"       "Harper"
[9] "Tovrea"      "Barreras"     "Klamerus"     "al-Satter"
[13] "al-Sultana"  "Yi"            "Helm"          "Molina-Peinado"
[17] "Spilsted"    "Egan"          "Thompson"     "Schauss"
[21] "Owens"        "Vick"          "Reeves"        "Chesney"
[25] "Minnillo"
```

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 female
  4 female
  5 male
  6 male
  7 male
  8 male
  9 female
 10 male
# i 15 more rows
```

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

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

```
4 female
5 male
6 male
7 male
8 male
9 female
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     Black (not Hispanic)
3 female   White (not Hispanic)
4 female   Middle-Eastern, Arabic
5 male     Middle-Eastern, Arabic
6 male     White (not Hispanic)
7 male     White (not Hispanic)
8 male     White (not Hispanic)
9 female   White (not Hispanic)
10 male    Hispanic
# i 15 more rows
```

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

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

```
14 male
15 male
16 female
17 female
18 male
19 female
20 female
21 female
22 male
23 male
24 female
25 female
```

```
rawdata_df[[2]] #returns vector
```

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

```
rawdata_df[,2] #returns vector
```

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

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

	Sex	Ethnicity
1	male	White (not Hispanic)
2	male	Black (not Hispanic)
3	female	White (not Hispanic)
4	female	Middle-Eastern, Arabic
5	male	Middle-Eastern, Arabic
6	male	White (not Hispanic)
7	male	White (not Hispanic)
8	male	White (not Hispanic)
9	female	White (not Hispanic)
10	male	Hispanic

```

11 female      White (not Hispanic)
12 male        Middle-Eastern, Arabic
13 male        Middle-Eastern, Arabic
14 male        Asian or Pacific Islander
15 male        White (not Hispanic)
16 female      Hispanic
17 female      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        Black (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)    128            87    88
2 P 2   male  Black (not Hispanic)   116            72    82
3 P 3   female White (not Hispanic)  120            60    93
4 P 4   female Middle-Eastern, Arabic 130            76    82
5 P 5   male  Middle-Eastern, Arabic  135            60   100
6 P 6   male  White (not Hispanic)   148            85    85
7 P 7   male  White (not Hispanic)   137            54    98
8 P 8   male  White (not Hispanic)   139            95    94
9 P 9   female White (not Hispanic)  154            71    83
10 P 10  male  Hispanic           113            74    82
# 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)    128            87    88
2 P 2   male  Black (not Hispanic)   116            72    82
3 P 3   female White (not Hispanic)  120            60    93

```

```
4 P 4    female Middle-Eastern, Arabic          130      76      82
5 P 5    male   Middle-Eastern, Arabic          135      60      100
```

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

```
# A tibble: 25 x 2
`sysRR (mmHg)` `diaRR (mmHg)`
<dbl>           <dbl>
1     128           87
2     116           72
3     120           60
4     130           76
5     135           60
6     148           85
7     137           54
8     139           95
9     154           71
10    113           74
# i 15 more rows
```

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

```
# A tibble: 25 x 1
HR
<dbl>
1     88
2     82
3     93
4     82
5    100
6     85
7     98
8     94
9     83
10    82
# i 15 more rows
```

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

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

```
# A tibble: 25 x 1
`sysRR (mmHg)` <dbl>
  1 128
  2 116
  3 120
  4 130
  5 135
  6 148
  7 137
  8 139
  9 154
 10 113
# i 15 more rows
```

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

```
# A tibble: 25 x 2
  Treatment     HR
  <chr>      <dbl>
1 Placebo      88
2 Placebo      82
3 Verum        93
4 Verum        82
5 Placebo      100
```

```

6 Verum      85
7 Verum      98
8 Placebo    94
9 Placebo    83
10 Verum     82
# i 15 more rows

```

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

```
[1] 128 116 120 130 135 148 137 139 154 113 118 117 127 143 117 135 117 147 127
[20] 130 136 141 150 147 130
```

Exercise: 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
- print salt
- print 100 saltgrains



2.5.4 List

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

2.5.4.1 Creation

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

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

$snacks
[1] "chips"        "pretzels"

$nonfood
[1] "DVDs" "Akku"

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

$volumen
[1] 101.98092 99.72118 101.25978 99.50115 99.88413 103.65050 100.35412
[8] 103.72203 97.45847 98.51527 98.86176 100.25617 99.18075 101.11364
[15] 99.49699 99.67630 99.17729 100.27251 100.64437 98.00664 99.25581
[22] 100.43009 100.20727 100.25817 98.48912 96.71237 96.46227 101.04724
[29] 96.47107 100.85070 97.64126 100.82651 102.66924 98.58853 99.72314
[36] 100.51849 98.90033 101.05662 100.26360 97.92360 100.89800 98.84879
[43] 101.56033 98.08456 100.80067 101.36969 94.41230 100.80553 99.60254
[50] 100.62478
```

```
shopping$snacks
```

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

2.5.4.2 Indexing

```
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 -5.74  
   ..- attr(*, "names")= chr "t"  
 $ parameter : Named num 36.1  
   ..- attr(*, "names")= chr "df"  
 $ p.value    : num 1.52e-06  
 $ conf.int   : num [1:2] -2.35 -1.12  
   ..- attr(*, "conf.level")= num 0.95  
 $ estimate   : Named num [1:2] 10.2 11.9  
   ..- attr(*, "names")= chr [1:2] "mean of x" "mean of y"  
 $ null.value : Named num 0  
   ..- attr(*, "names")= chr "difference in means"  
 $ stderr     : num 0.302  
 $ 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] 1.523729e-06
```

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

```
[1] 1.523729e-06
```

2.6 Control structures

2.6.1 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.

2.6.1.1 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] 101.98092 99.72118 101.25978 99.50115 99.88413 103.65050 100.35412
[8] 103.72203 97.45847 98.51527 98.86176 100.25617 99.18075 101.11364
[15] 99.49699 99.67630 99.17729 100.27251 100.64437 98.00664 99.25581
[22] 100.43009 100.20727 100.25817 98.48912 96.71237 96.46227 101.04724
[29] 96.47107 100.85070 97.64126 100.82651 102.66924 98.58853 99.72314

```

```
[36] 100.51849 98.90033 101.05662 100.26360 97.92360 100.89800 98.84879  
[43] 101.56033 98.08456 100.80067 101.36969 94.41230 100.80553 99.60254  
[50] 100.62478
```

```
# 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])  
}
```

2.6.1.2 while-loops

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

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

```
[1] 0  
[1] 1  
[1] 2  
[1] 3  
[1] 4  
[1] 5
```

```
[1] 6  
[1] 7  
[1] 8  
[1] 9
```

2.6.2 Conditions

2.6.2.1 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

```
[1] FALSE
```

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

```
[1] FALSE
```

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

```
[1] FALSE
```

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

```
[1] FALSE
```

```
TRUE | FALSE
```

```
[1] TRUE
```

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

```
[1] TRUE
```

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

```
[1] TRUE
```

2.6.2.2 `ifelse`

We can get text conditionally:

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

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

```

p <- .12
paste0("That is ",
       ifelse(test = p<=.05, yes = "", no = "not "),
       "significant")

```

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

2.6.2.3 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        128          87 hypertensive yes      
2        116          72 borderline   possibly  
3        120          60 borderline   possibly  
4        130          76 borderline   possibly  
5        135          60 borderline   possibly  
6        148          85 hypertensive yes      
7        137          54 borderline   possibly  
8        139          95 hypertensive yes      
9        154          71 borderline   possibly  
10       113          74 borderline   possibly  
# i 15 more rows

```

3 Importing data

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

3.1 Import from text files (.txt, .csv)

3.2 Import from Excel

3.2.1 Tidy Excel files

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

3.2.2 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 = "data/Medtest_e.xlsx",
                      range = "A1:Y2",col_names = FALSE,
                      sheet = 2)
```

```

New names:
* `` -> `...1`
* `` -> `...2`
* `` -> `...3`
* `` -> `...4`
* `` -> `...5`
* `` -> `...6`
* `` -> `...7`
* `` -> `...8`
* `` -> `...9`
* `` -> `...10`
* `` -> `...11`
* `` -> `...12`
* `` -> `...13`
* `` -> `...14`
* `` -> `...15`
* `` -> `...16`
* `` -> `...17`
* `` -> `...18`
* `` -> `...19`
* `` -> `...20`
* `` -> `...21`
* `` -> `...22`
* `` -> `...23`
* `` -> `...24`
* `` -> `...25`

```

```

for(col_i in colnames(cn_temp)){
  if(!is.na(cn_temp[1,] > 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 = "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,]
```

3.3 Import from SPSS

3.4 Import from SAS

4 Changing structure wide <-> long

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 `tidyverse` 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.

There are many examples and explanations in the vignette:

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

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

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        114.      122.
2 P 2    B        113.      129.
3 P 3    C        96.8      113.

```

```

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 114.
2 P 1    A    Var2Time2 122.
3 P 2    B    Var2Time1 113.
4 P 2    B    Var2Time2 129.
5 P 3    C    Var2Time1 96.8
6 P 3    C    Var2Time2 113.

```

Example 2: several repeated measures

```

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      101.      109.      12.2      11.5  A
2 P 2       79.1      86.6      9.75      9.82  B
3 P 3       92.1      94.6      11.9      10.1  C
```

```
#intermediate:
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  101.
2 P 1   A     Var1    Time2  109.
3 P 1   A     Var2    Time1  12.2
4 P 1   A     Var2    Time2  11.5
5 P 2   B     Var1    Time1  79.1
6 P 2   B     Var1    Time2  86.6
7 P 2   B     Var2    Time1  9.75
8 P 2   B     Var2    Time2  9.82
9 P 3   C     Var1    Time1  92.1
10 P 3  C     Var1    Time2  94.6
11 P 3  C     Var2    Time1  11.9
12 P 3  C     Var2    Time2  10.1
```

```
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> <chr>  <dbl> <dbl>
1 P 1   A     Time1  101.  12.2
```

2	P	1	A	Time2	109.	11.5
3	P	2	B	Time1	79.1	9.75
4	P	2	B	Time2	86.6	9.82
5	P	3	C	Time1	92.1	11.9
6	P	3	C	Time2	94.6	10.1

5 Grouping of variables by type / distribution / use

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

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

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

5.1 Test for Normal distribution

5.1.1 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.

```
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)` , : für den Komogorov-Smirnov-Test sollten keine
Bindungen vorhanden sein
```

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)`)
```

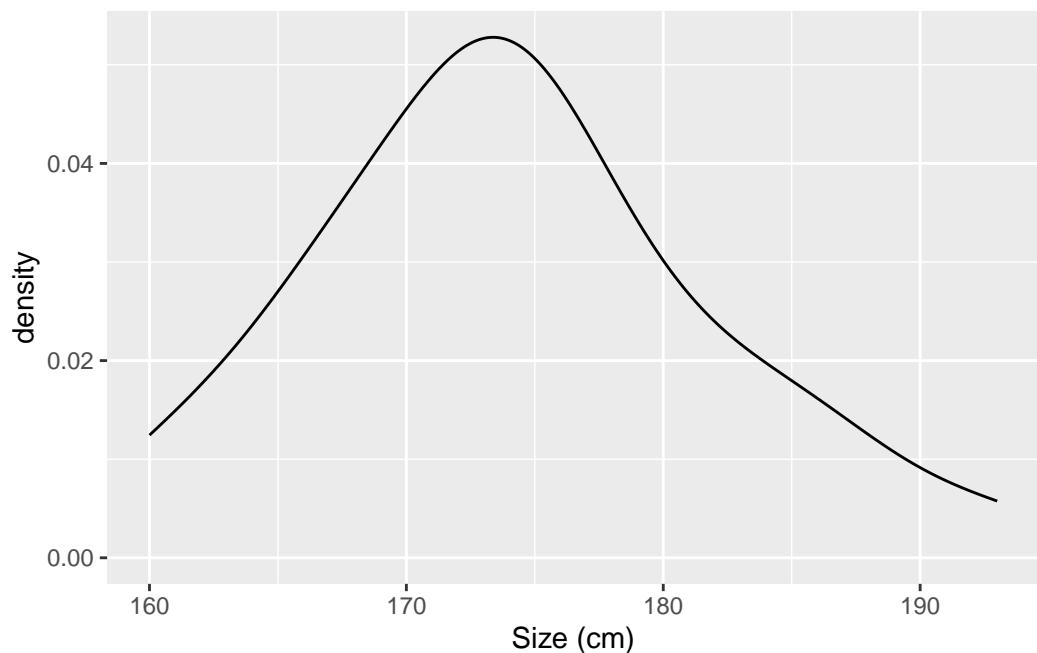
```
[1] 0.7063825
```

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

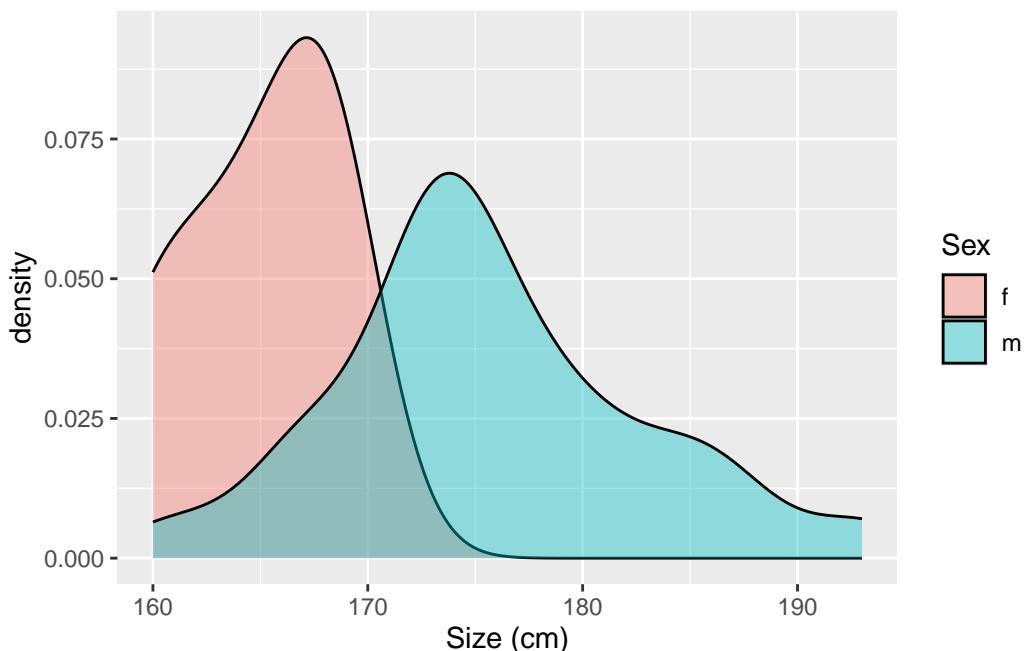
Shapiro-Wilk normality test

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

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



```
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()
```

```
[1] 0.5748486
```

```
rawdata |>  
  group_by(Sex) |>  
  summarize(p_KS = ksnormal(`Size (cm)`),  
            pGauss (Shapiro) = shapiro.test(`Size (cm)`)$p.value)
```

```
# A tibble: 2 x 3  
  Sex     p_KS `pGauss (Shapiro)`  
  <chr>   <dbl>          <dbl>  
1 f       0.905          0.272  
2 m       0.575          0.638
```

5.1.2 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(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      0.999    3.10e- 1
2 Included        0.000000170 2.25e-11
3 Finalized       0.000000800 1.86e- 9
4 Testmedication  0.00347   4.31e- 7
5 Size (cm)       0.706    7.63e- 1
6 Weight (kg)     0.760    8.96e- 2
```

Loop Version 2:

```
## result table v2, just structure
resulttable2 <- tibble(Variables=NA_character_,
                      pKS_Placebo=NA_character_,
                      pKS_Verum=NA_character_,
                      .rows = 0)
for(var_i in numvars$name){
  ks_tmp <- by(rawdata[[var_i]],
                INDICES=rawdata$Testmedication,
                FUN=ksnormal)
  resulttable2 <- add_row(resulttable2,
                           Variables=var_i,
                           pKS_Placebo=ks_tmp[[1]] |>
                             formatP(), # added rounding/formatting
                           pKS_Verum=ks_tmp[[2]] |>
                             formatP())
}
head(resulttable2)
```

```
# A tibble: 6 x 3
Variables      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:

```
resulttable3 <-
  rawdata |>
```

```

group_by(Testmedication) |>
summarize(across(all_of(numvars$names[-(2:4)]),
.fns = list(
  # n=~na.omit(.x) |> length() |> as.character(),
  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"),
  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 0.489 n.s. 0.425 n.s.
2          1 Age        60.429 60.000 0.935 n.s. 0.282 n.s.
3          0 BMI        30.244 29.246 0.956 n.s. 0.430 n.s.
4          1 BMI        28.016 27.484 0.992 n.s. 0.862 n.s.
5          0 Ferritin Lab 258.21 220.00 0.595 n.s. 0.176 n.s.
6          1 Ferritin Lab 305.23 222.00 0.139 n.s. 0.003 **

```

5.2 Picking column names and positions

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

```

gaussvars <- ColSeeker(
  namepattern = c("si","we","BMI","BP","mri"),
  casesensitive = FALSE)

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

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

```

```
rawdata <- mutate(rawdata,
                  across(all_of(factvars$names),
                         ~factor(.x)))
save(rawdata,list = ls(pattern = "vars"),
     file = "data/bookdata1.RData")
```

6 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,
  grid,gridExtra,car,
  ggsci,ggsignif, ggthemes, ggridges,
  # gganimate,
  ggforce,
  ggbeeswarm,
  wrappedtools,
  emojiifont,
  patchwork)
conflicts_prefer(dplyr::filter) # solves name conflict
```

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

6.1 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

6.2 Basic structure of a ggplot call

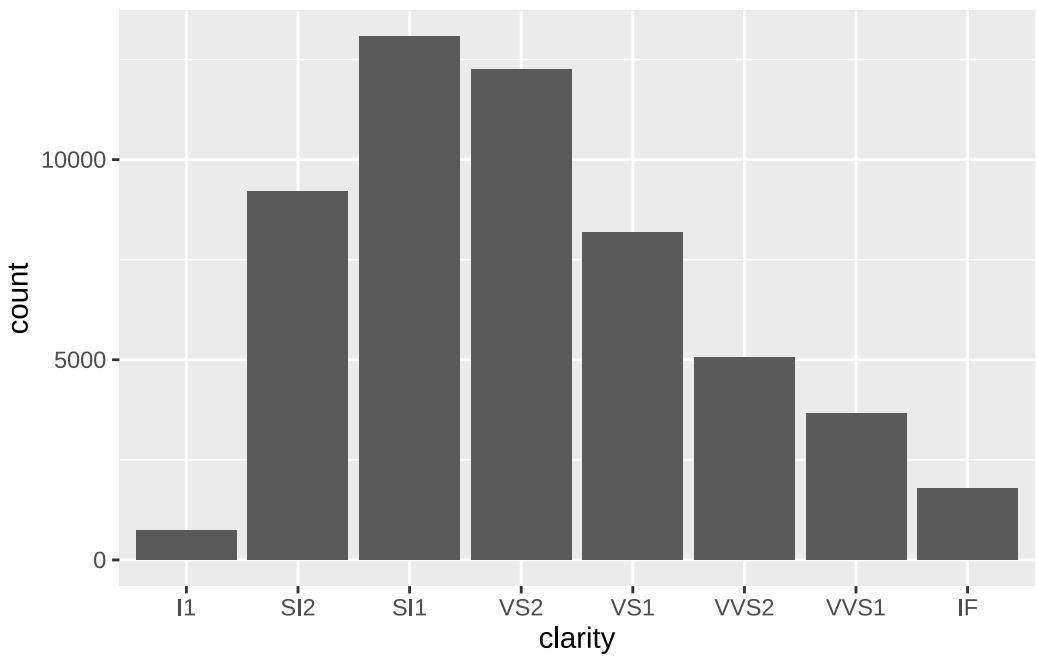
To create a figure, we at least use 2 function calls:

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:

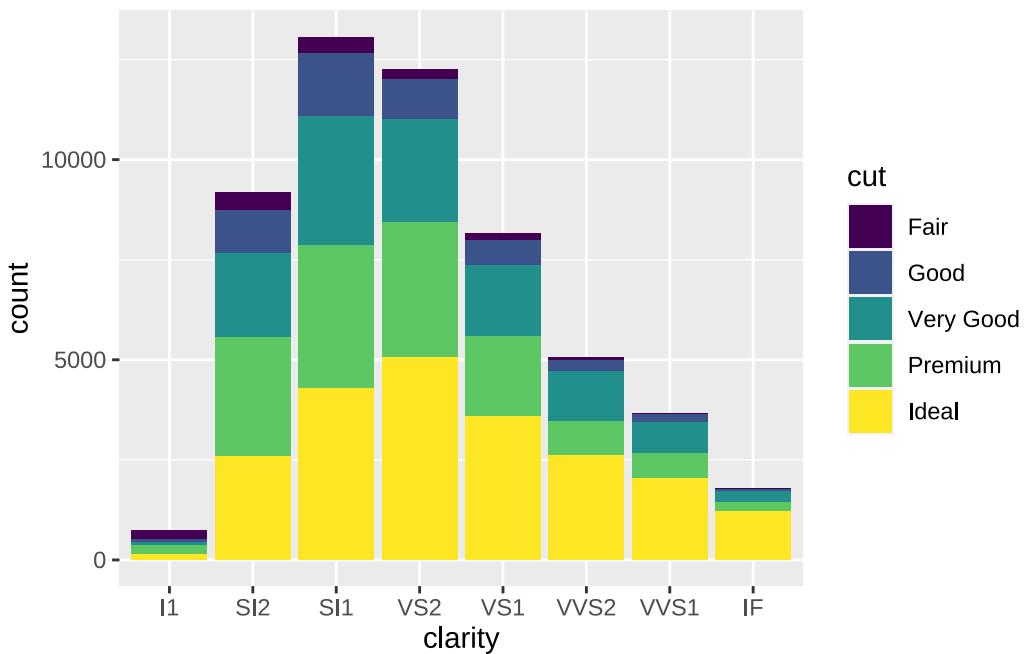
```
ggplot(data=diamonds, mapping = aes(x=clarity))+  
  geom_bar()
```



`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()
```



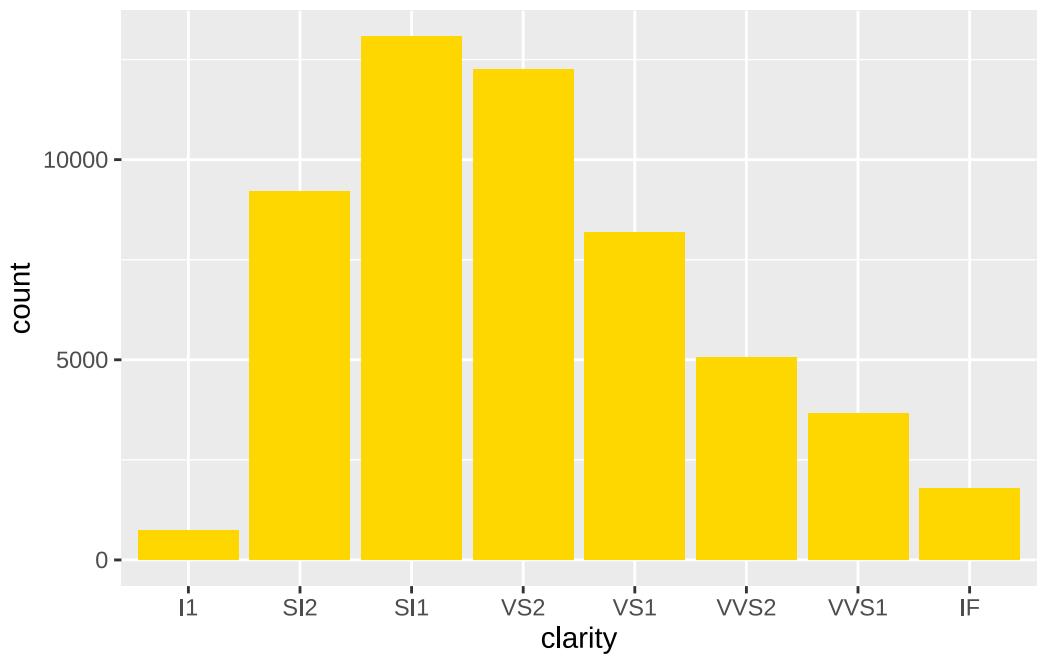
This graph represents this count table:

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

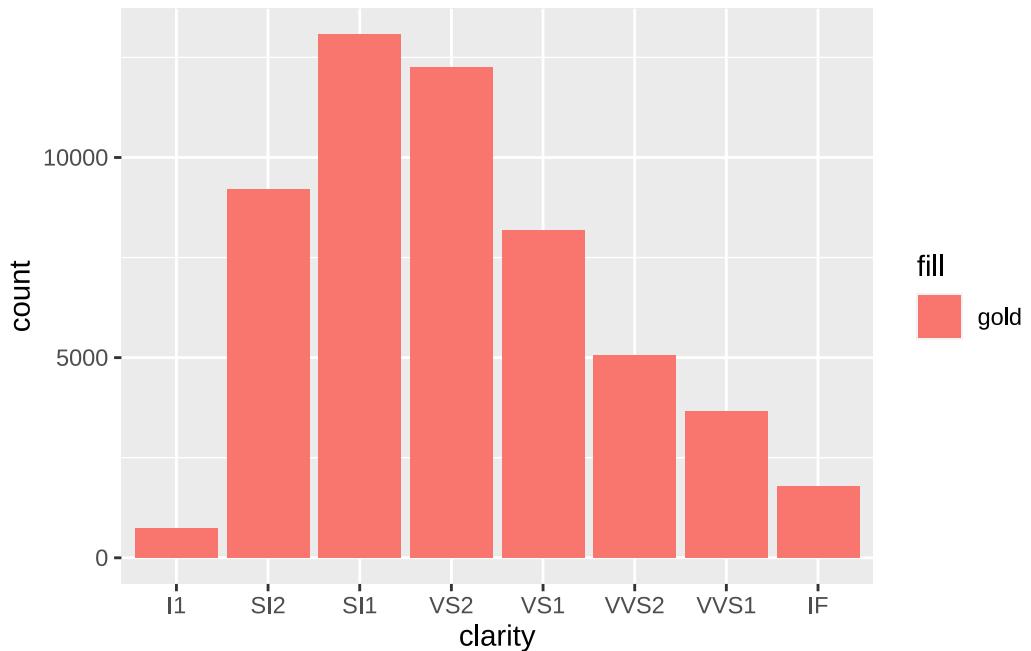
	I1	SI2	SI1	VS2	VS1	VVS2	VVS1	IF
1 Fair	210	466	408	261	170	69	17	9
2 Good	96	1081	1560	978	648	286	186	71
3 Very Good	84	2100	3240	2591	1775	1235	789	268
4 Premium	205	2949	3575	3357	1989	870	616	230
5 Ideal	146	2598	4282	5071	3589	2606	2047	1212

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:

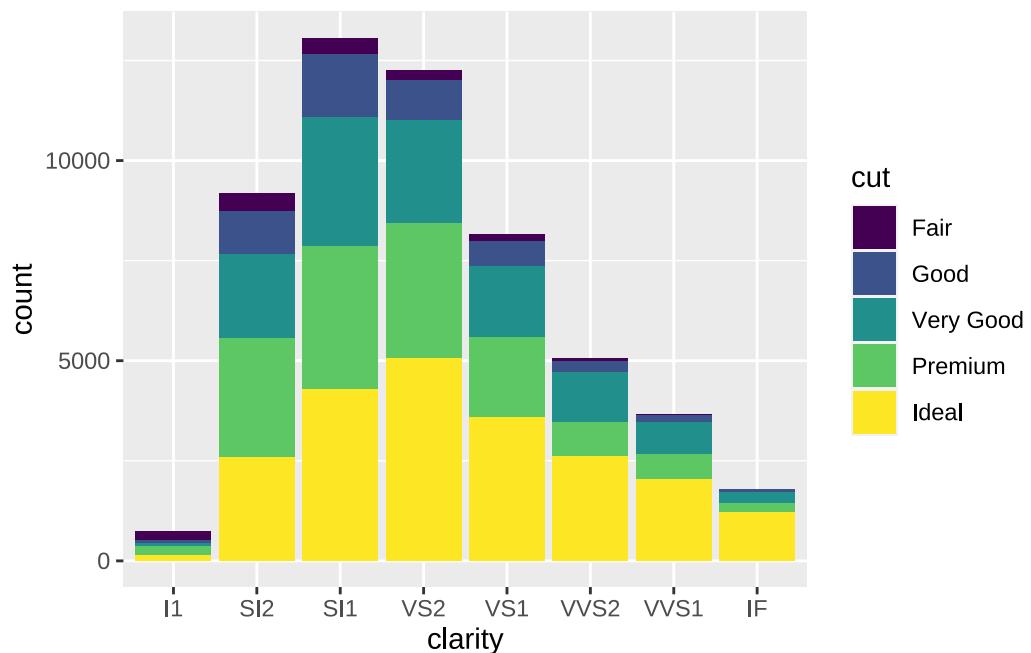
```
#aesthetics outside aes
ggplot(data=diamonds,aes(x=clarity))+
  geom_bar(fill="gold")
```



```
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
```



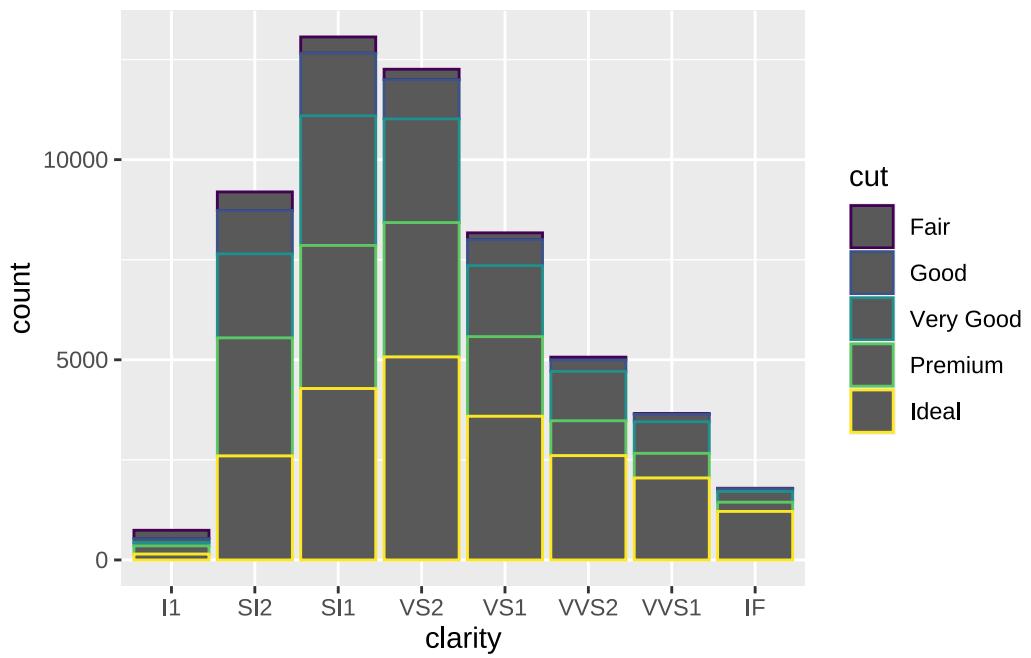
6.3 fill vs. color

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

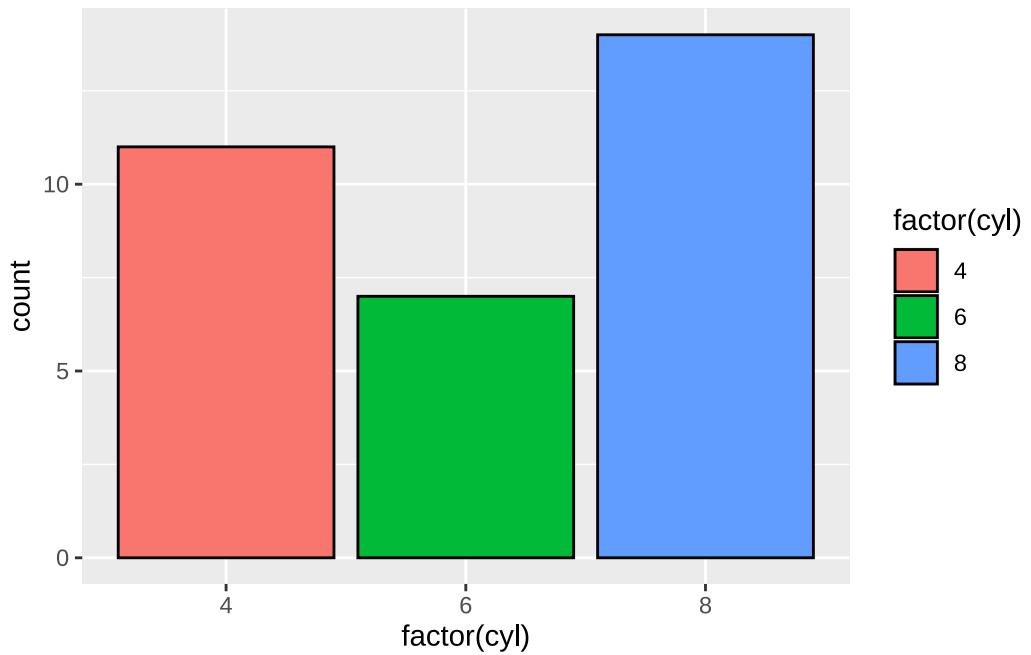
- inner color, defined by `fill`
- outer frame color, defined by `color`

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.

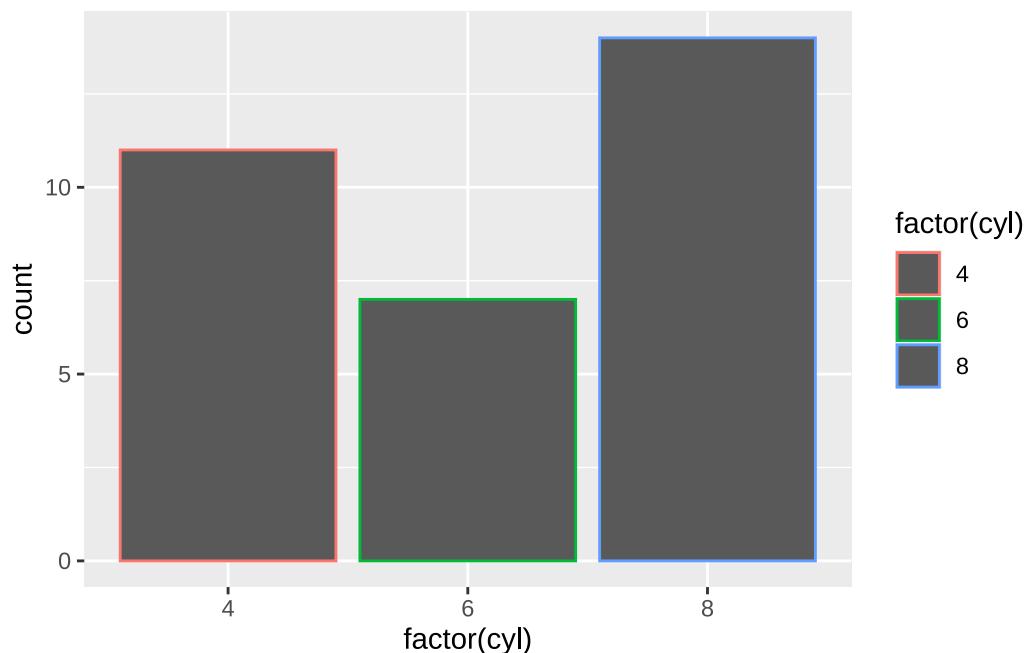
```
ggplot(data=diamonds,aes(x=clarity,color=cut))+  
  geom_bar()
```



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



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

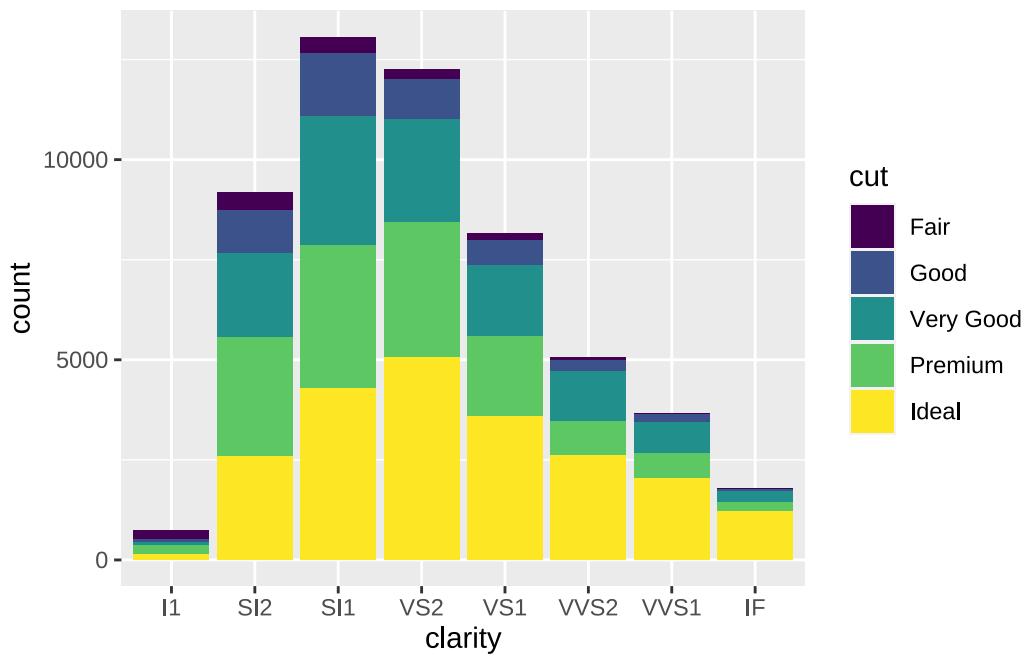


6.4 Color systems

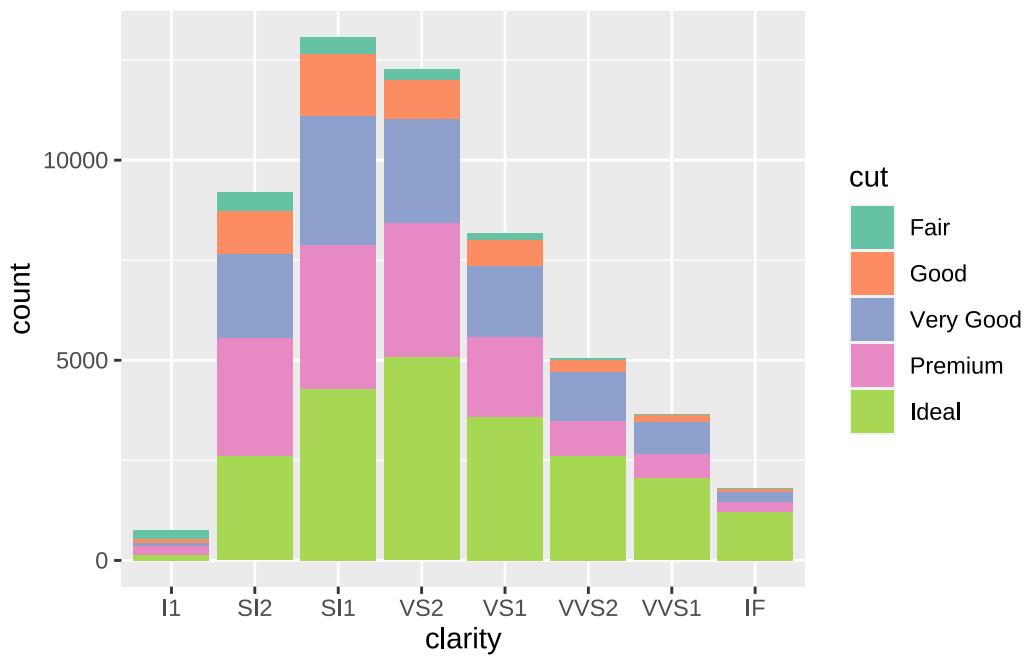
ggplot2 comes with various color definitions, many external packages extend that. Manual definition of colors is possible as well. Redefining the mapping between data and aesthetics can be done with scale_... functions

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

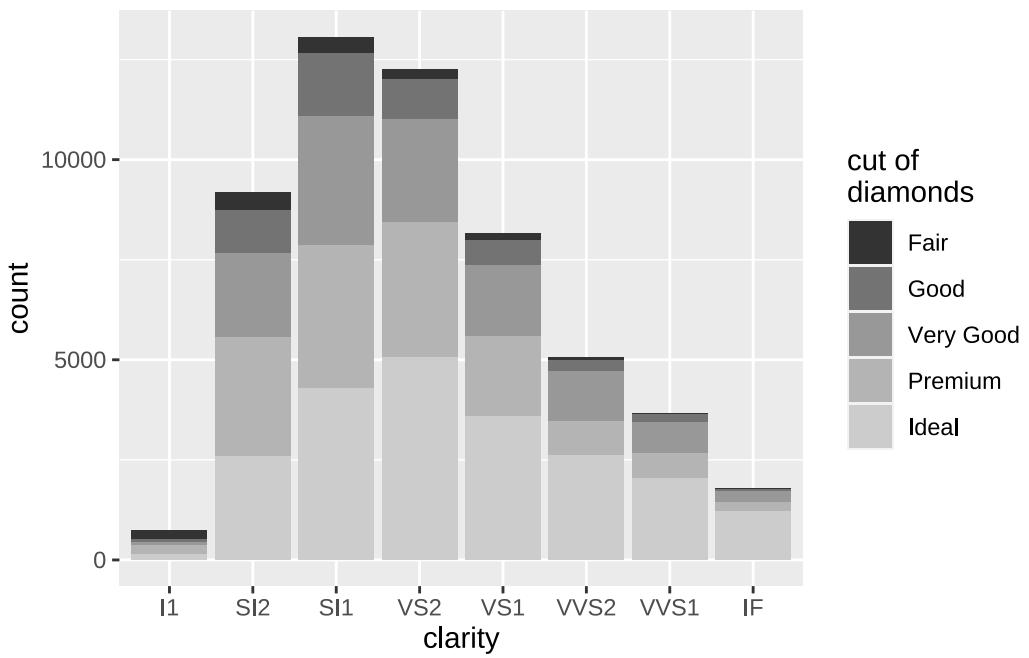
```
(plottemp <- ggplot(data=diamonds,aes(x=clarity,fill=cut))+  
  geom_bar())
```



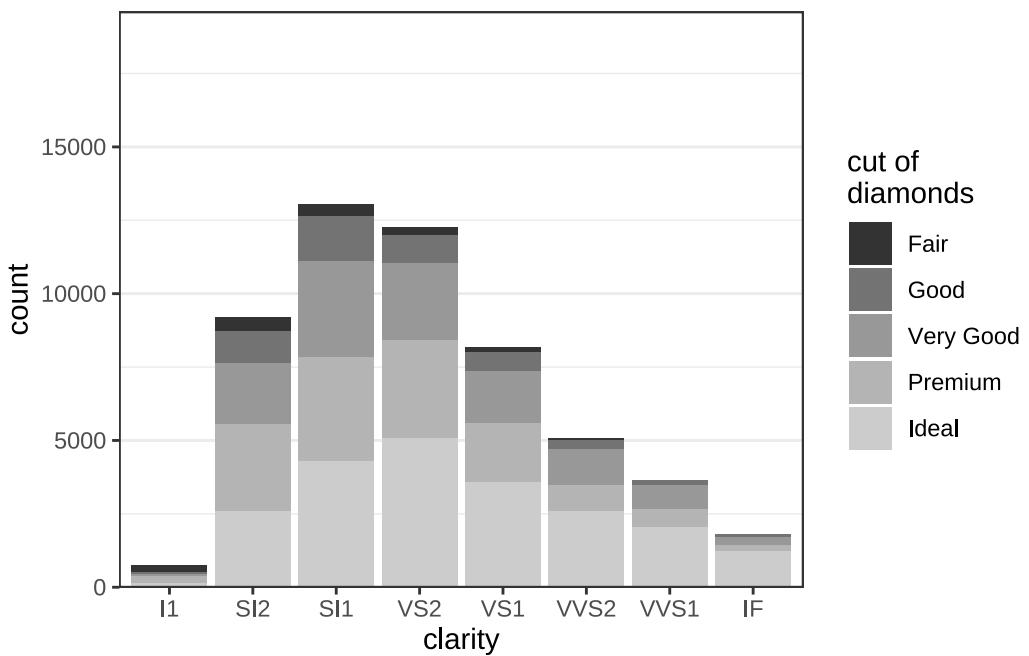
```
plottemp + scale_fill_brewer(palette="Set2") #in-built "scale" for fill
```



```
plottemp + scale_fill_grey(name = "cut of\ndiamonds")
```

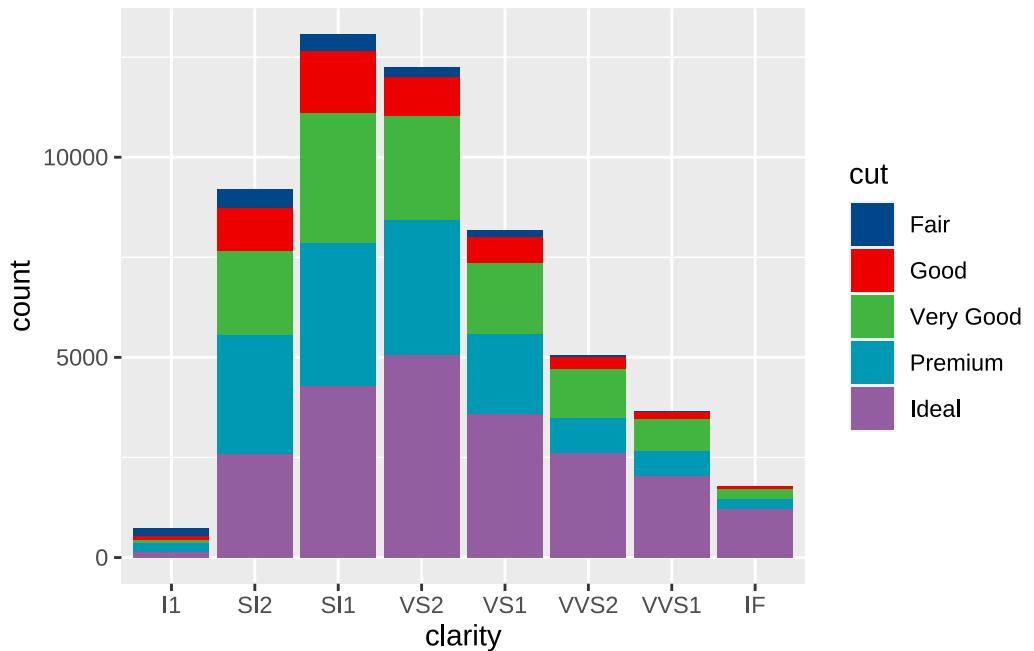


```
plottemp + scale_fill_grey(name = "cut of\ndiamonds") +
  scale_y_continuous(expand = expansion(mult = c(0,.5)))+ # rescaling y
  theme_bw()+
  theme(panel.grid.major.x = element_blank())
```

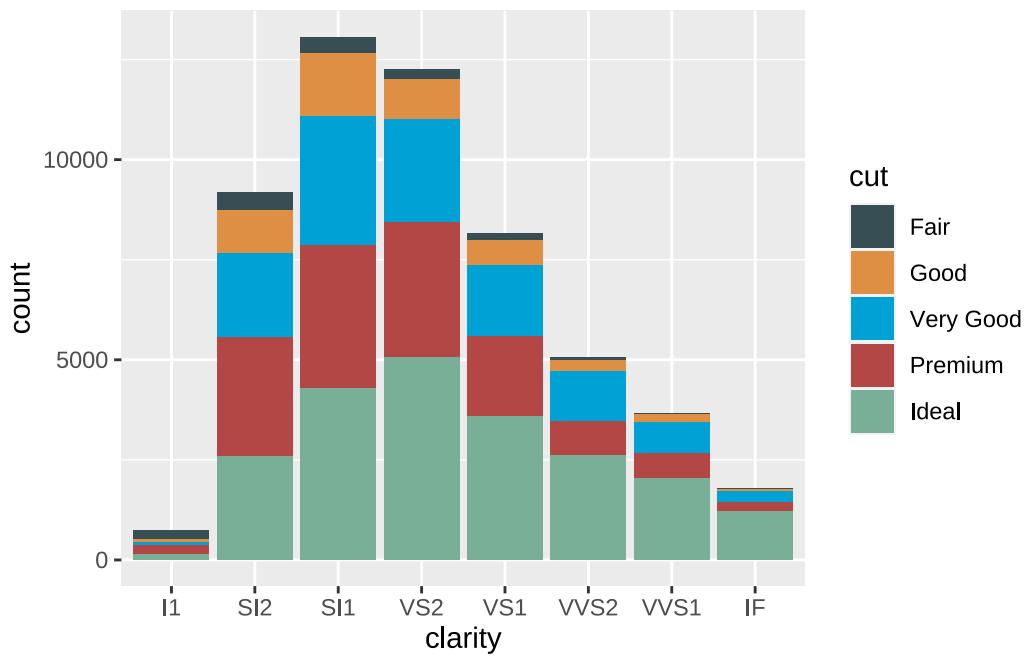


6.4.1 External color definitions from ggsci

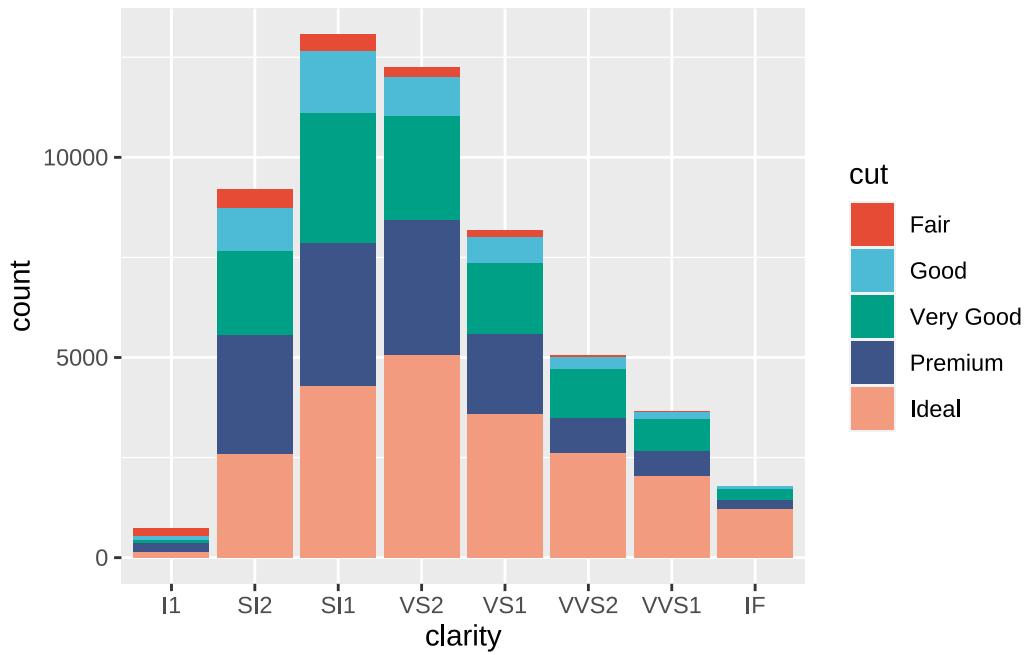
```
plottemp+scale_fill_lancet()
```



```
plottemp+scale_fill_jama()
```



```
plottemp+scale_fill_npg()
```



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

6.5 Exporting ggplots

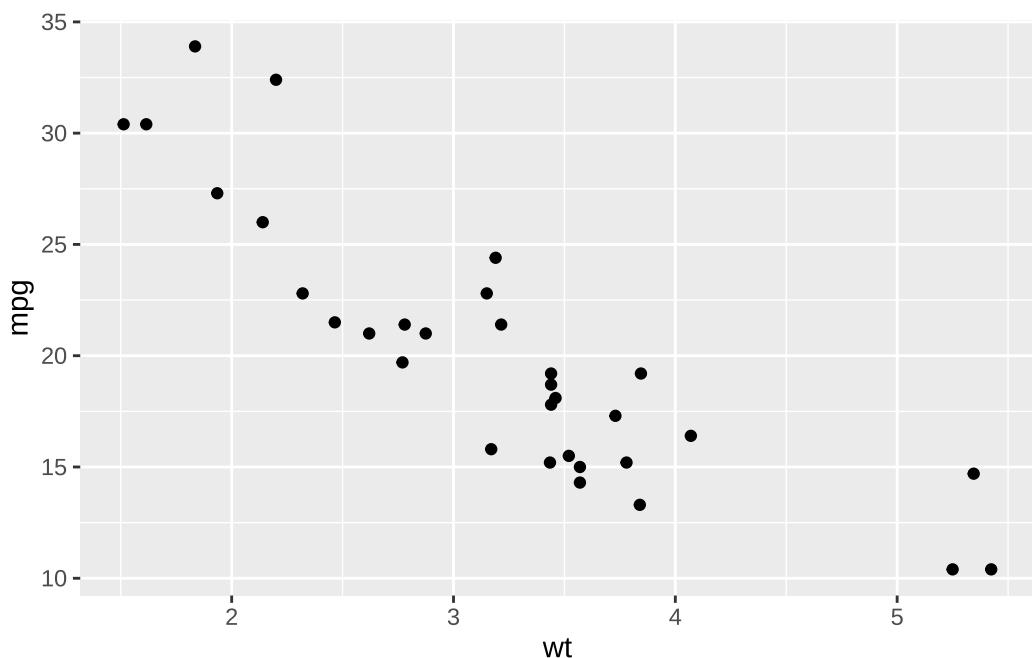
There are two distinct ways, using `ggsave()` or more generally creating an external graphic device with e.g. `png()` / `tiff()` / `pdf()`:

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

6.6 Other geoms

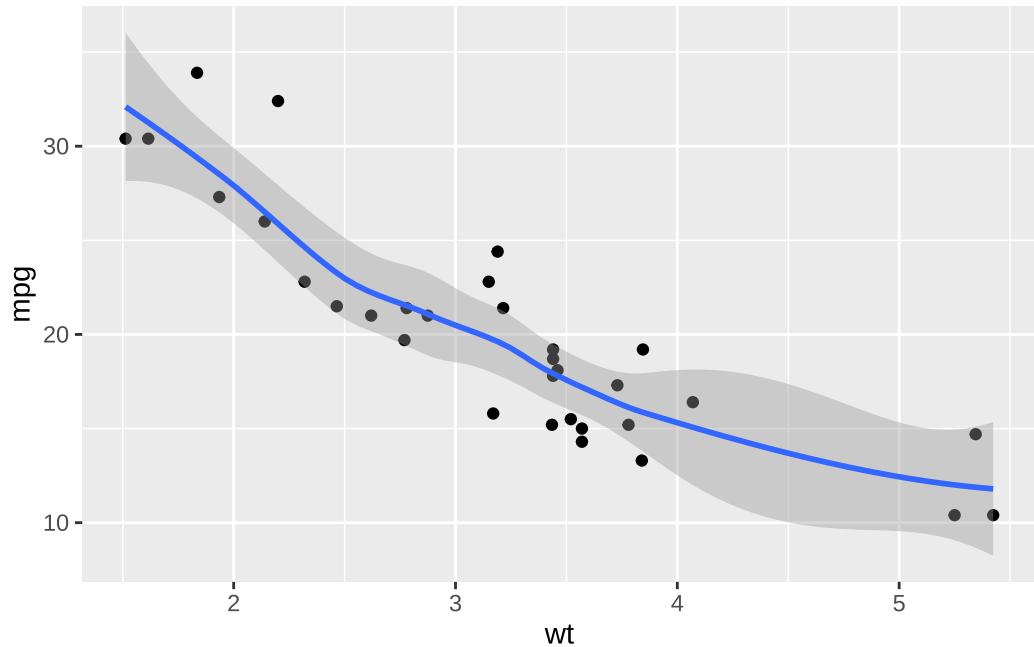
Common forms of plots are barplots, boxplots, scatterplots (possibly with regression line), and density-plots. For plotting dots for groups, there are various options to avoid overplotting of repeated data, with the beeswarm as my preference.

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



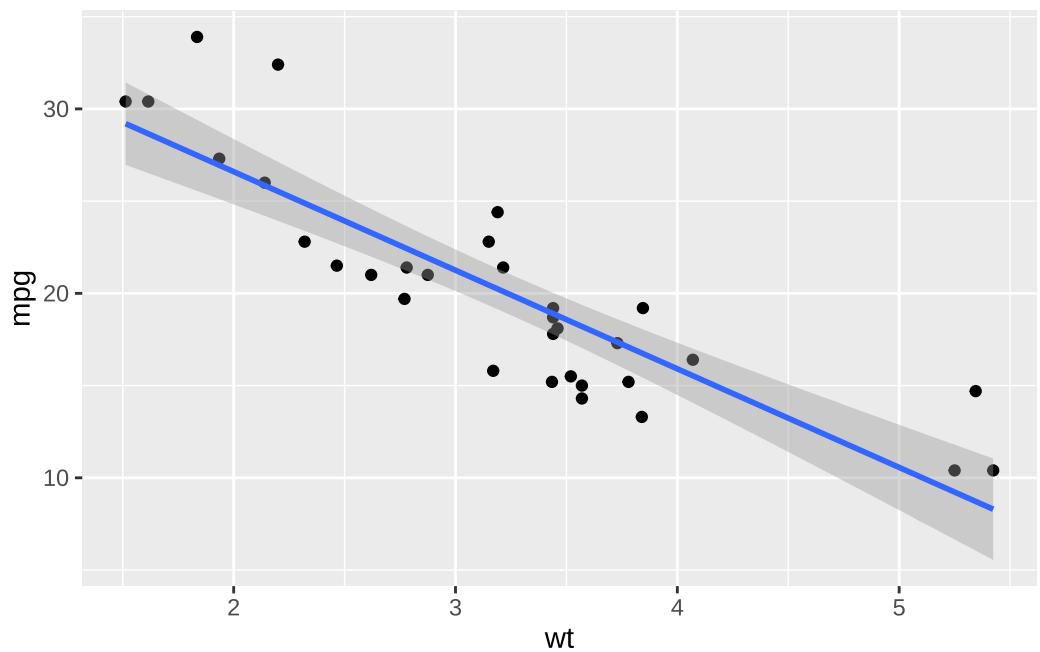
```
ggplot(data=mtcars,aes(x = wt,y = mpg))+  
  geom_point() +  
  geom_smooth()
```

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

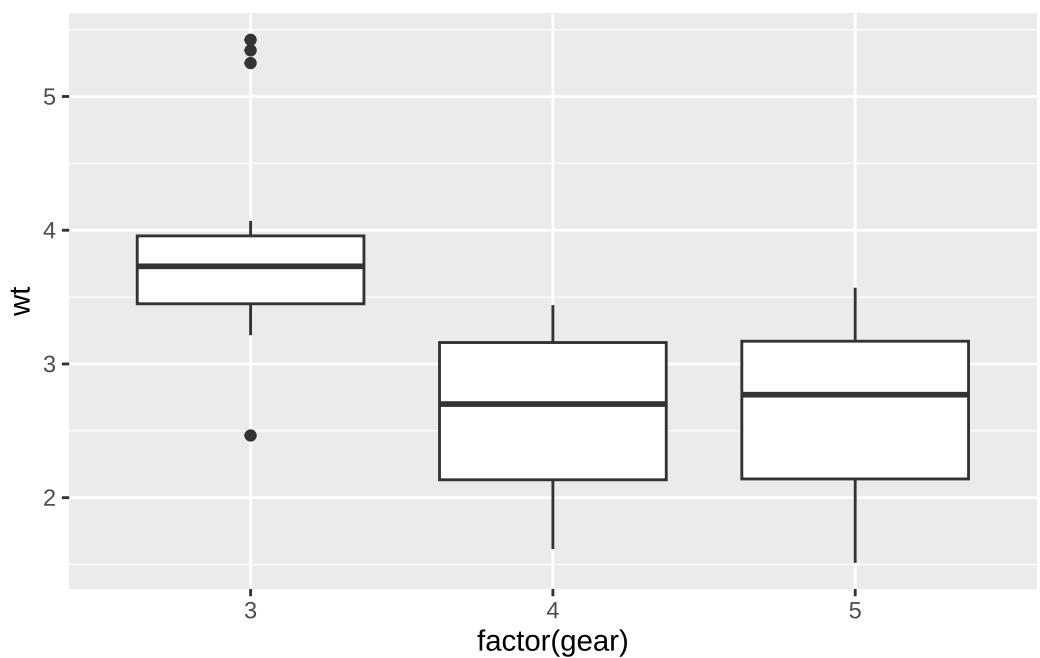


```
ggplot(data=mtcars,aes(x = wt,y = mpg))+  
  geom_point() +  
  geom_smooth(method="lm")
```

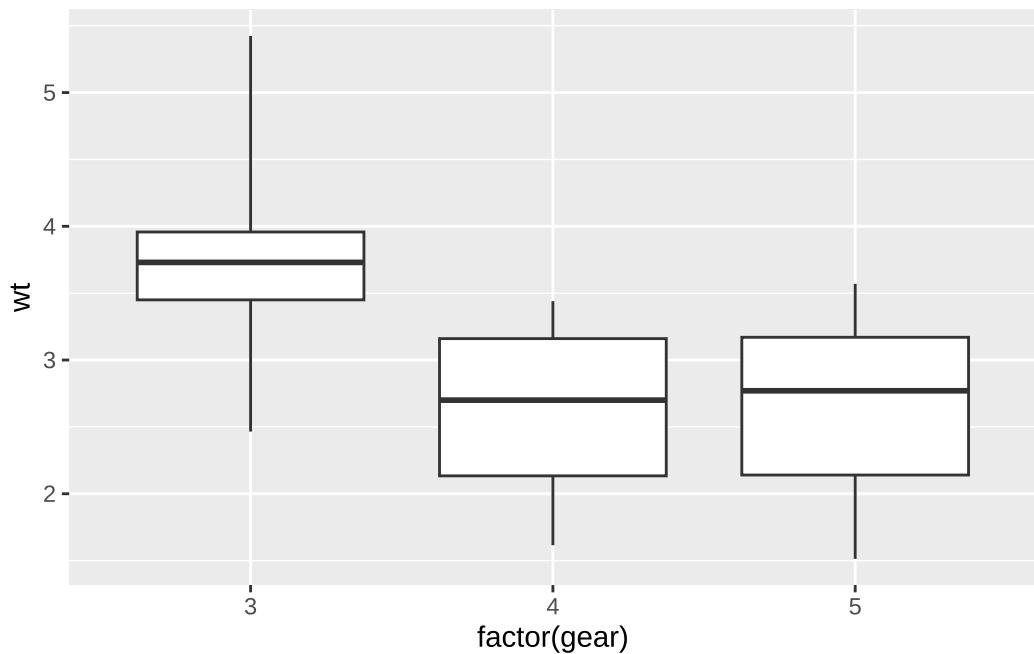
`geom_smooth()` using formula = 'y ~ x'



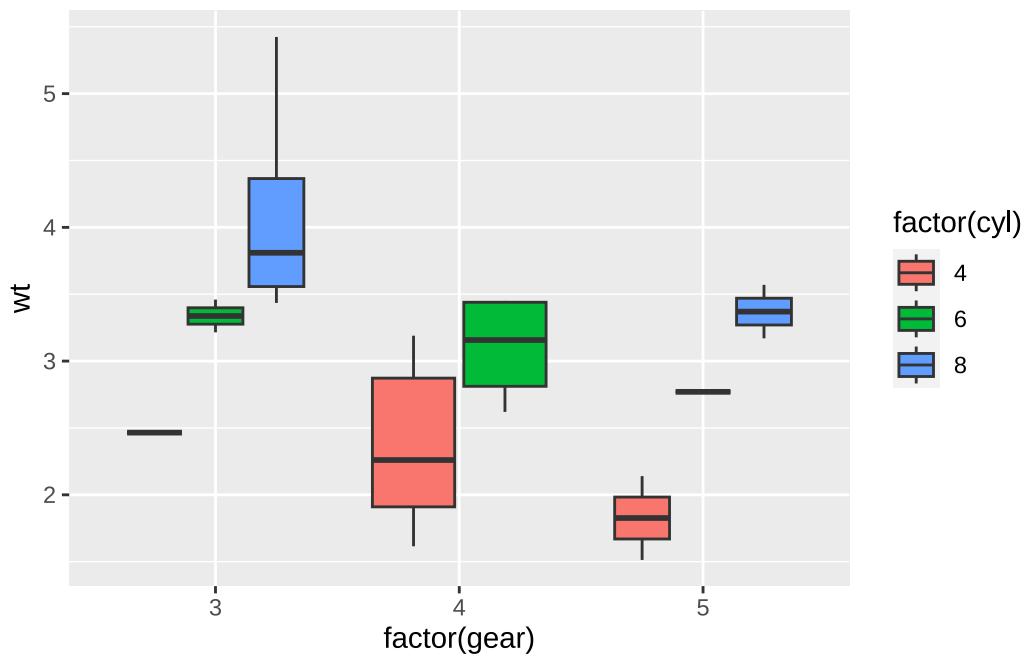
```
ggplot(mtcars,aes(x = factor(gear),y = wt))+  
  geom_boxplot() #default 1.5 IQR
```



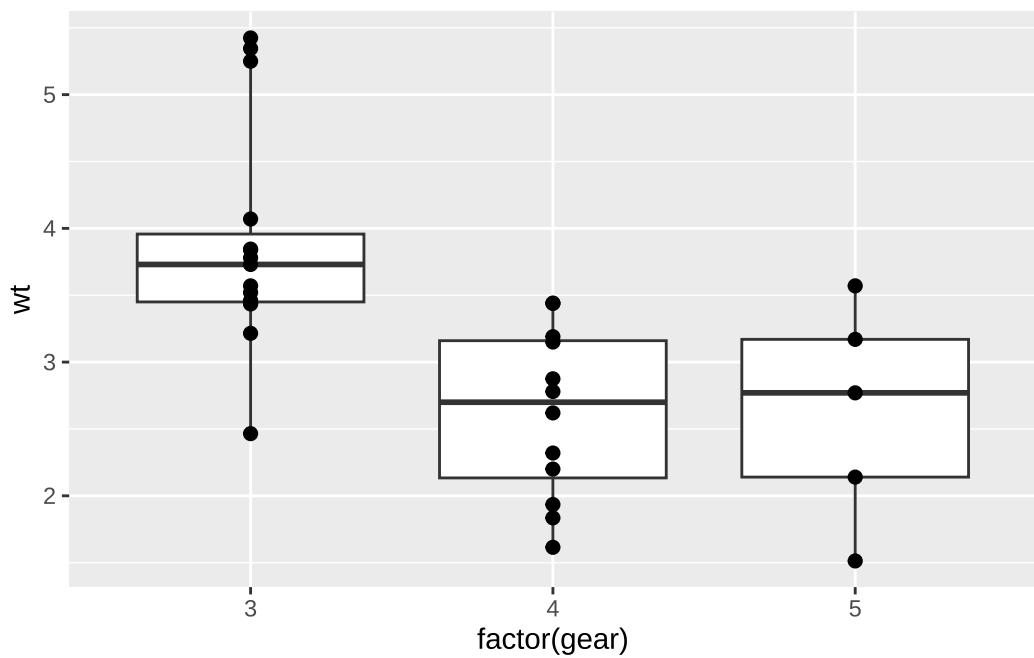
```
ggplot(mtcars,aes(x = factor(gear),y = wt))+  
  geom_boxplot(coef=3) # this extends range of expected values
```



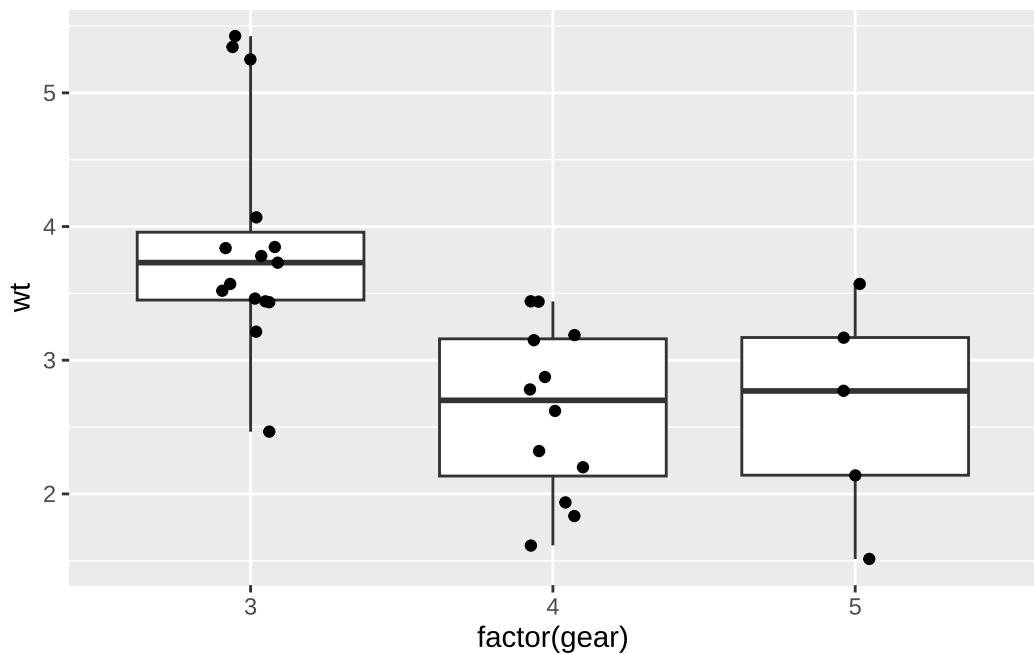
```
ggplot(mtcars,aes(x = factor(gear),y = wt,  
                   fill=factor(cyl)))+  
  geom_boxplot(coef=3) # group by cyl, as it is mapped to fill
```



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

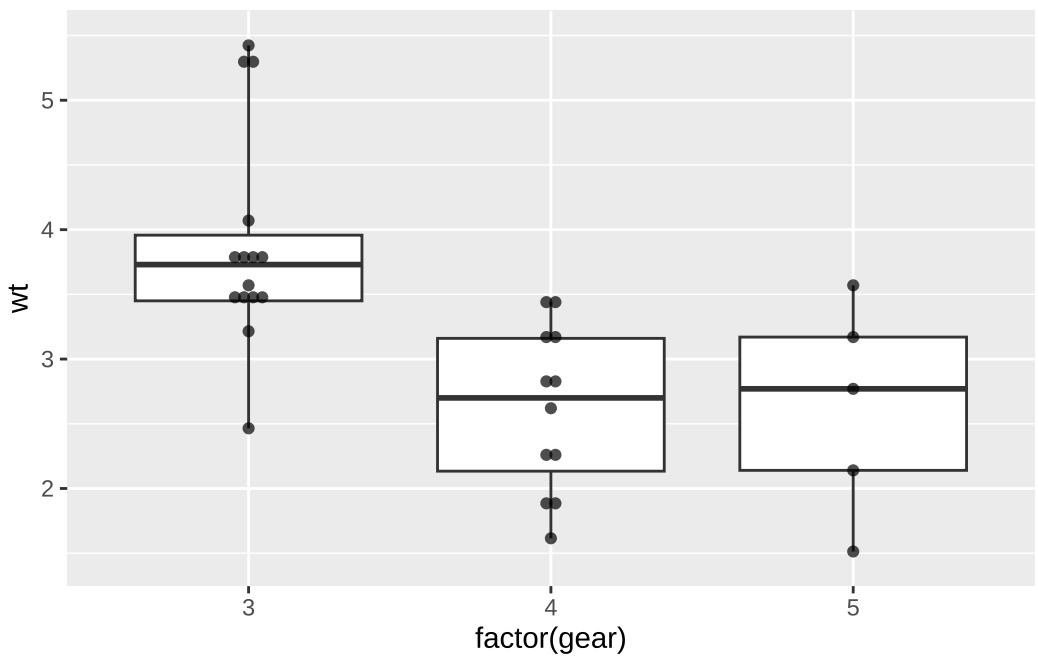


```
ggplot(mtcars,aes(x = factor(gear),y = wt))+  
  geom_boxplot(coef=3)+  
  geom_point(position = position_jitter(width = .1))
```

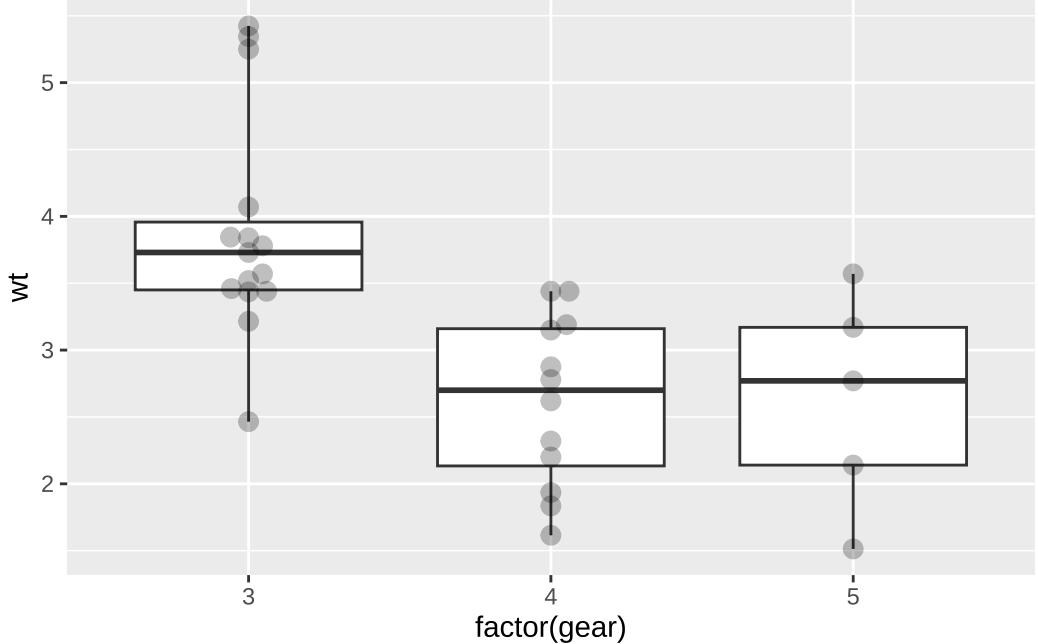


```
ggplot(mtcars,aes(x = factor(gear),y = wt))+  
  geom_boxplot(coef=3)+  
  geom_dotplot(alpha=.7, # group similar(ish) data on a line  
               binaxis = "y",stackdir = "center",  
               stackratio = .9,dotsize = .6)
```

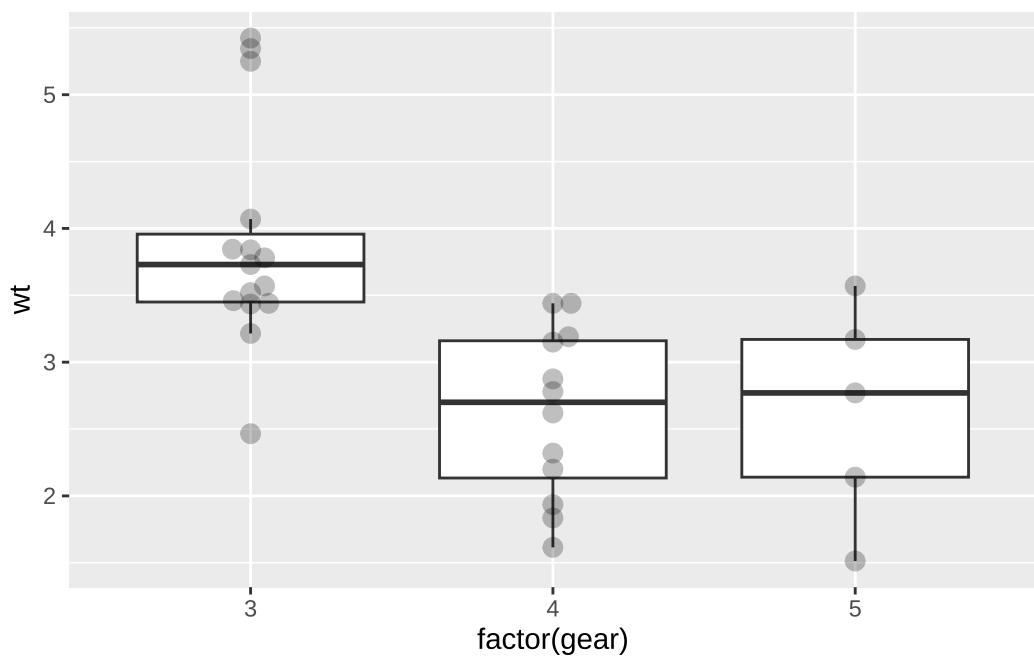
Bin width defaults to 1/30 of the range of the data. Pick better value with `binwidth`.



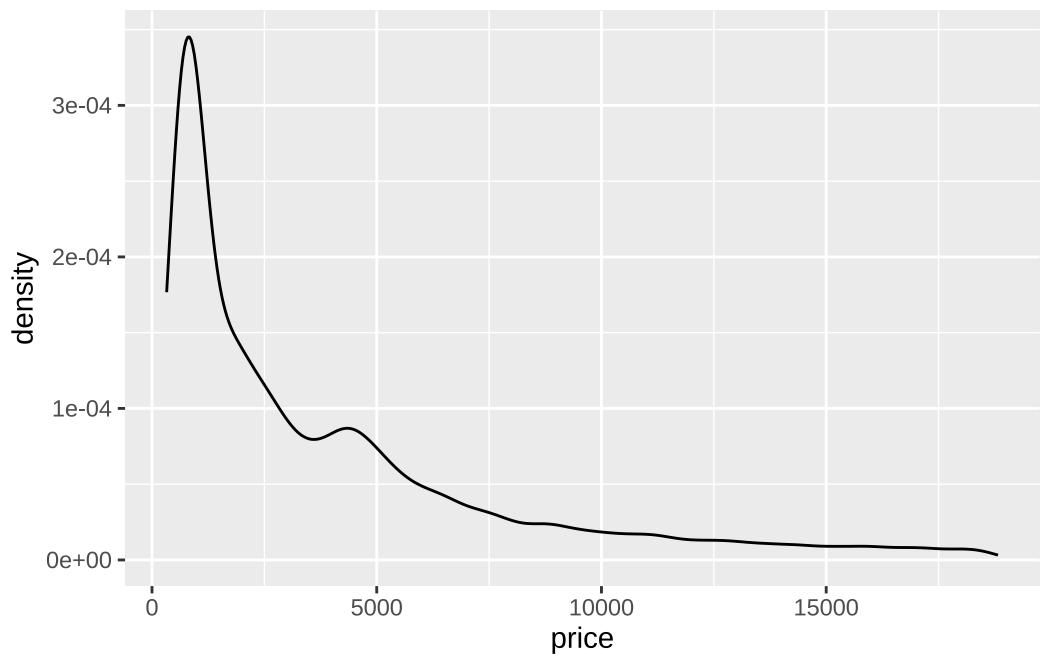
```
ggplot(mtcars,aes(x = factor(gear),y = wt))+  
  geom_boxplot(coef=3)+  
  ggbeeswarm::geom_beeswarm(cex = 2,size=3,alpha=.25)
```



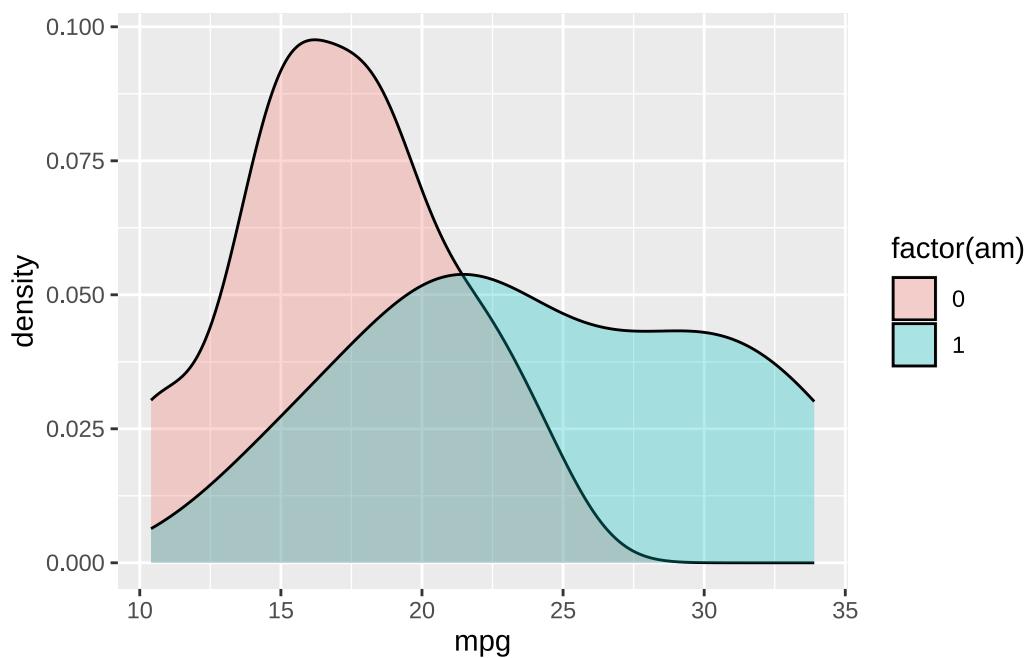
```
ggplot(mtcars,aes(x = factor(gear),y = wt))+  
  geom_boxplot(outlier.alpha = 0)+ # to avoid plotting outliers twice  
  geom_beeswarm(cex = 2,size=3,alpha=.25)
```



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

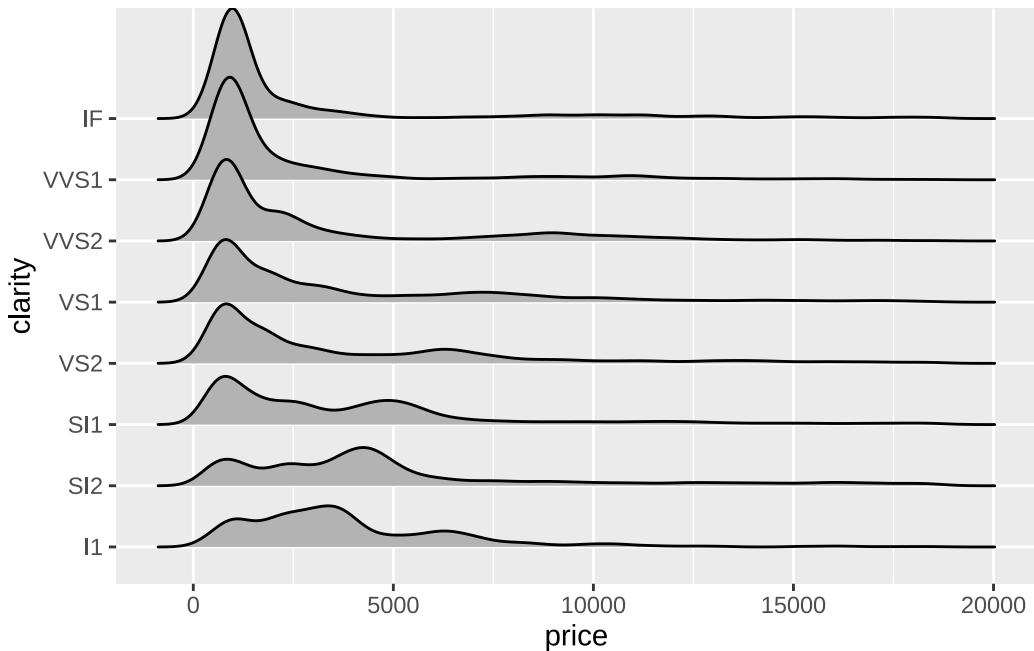


```
ggplot(mtcars,aes(mpg, fill=factor(am)))+  
  geom_density(alpha=.3)
```



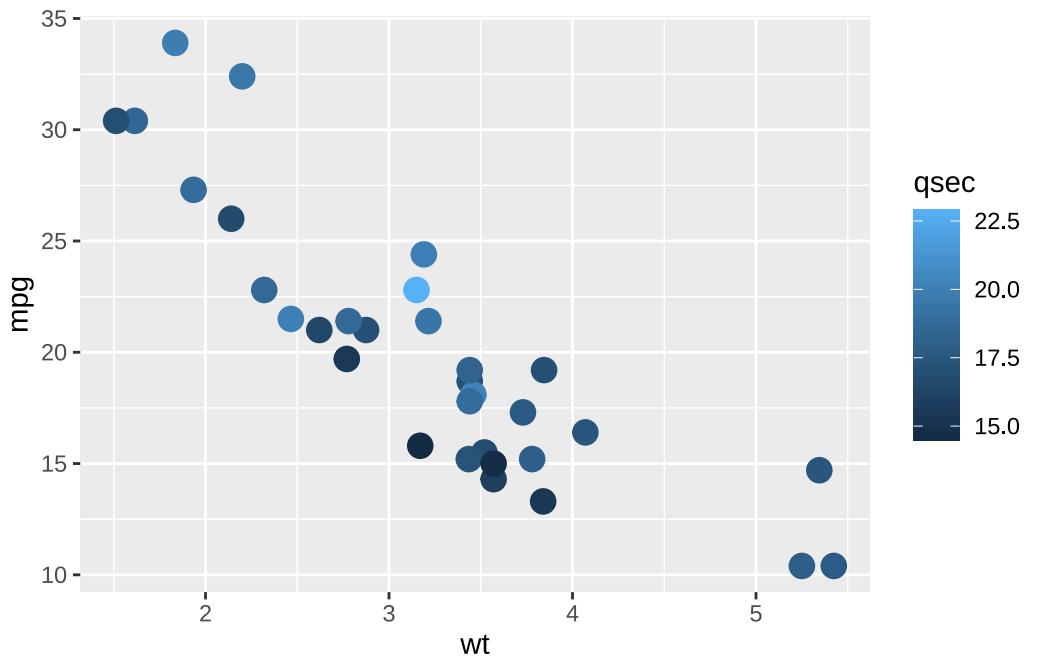
```
ggplot(diamonds,aes(price,y=clarity))+  
  geom_density_ridges()
```

Picking joint bandwidth of 403

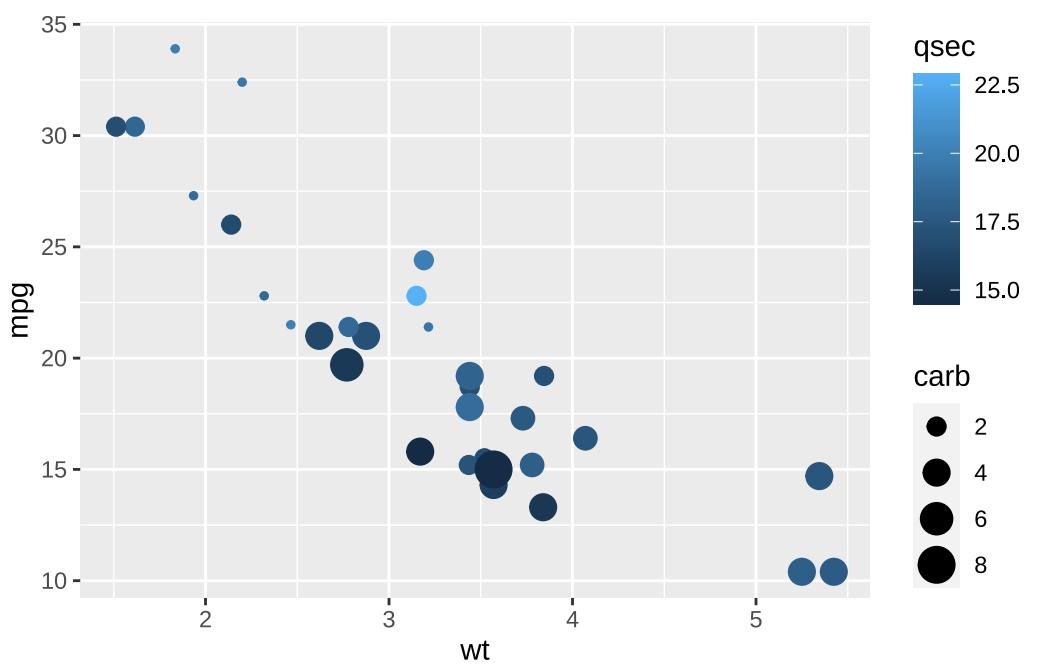


6.7 Combining and finetuning aesthetics

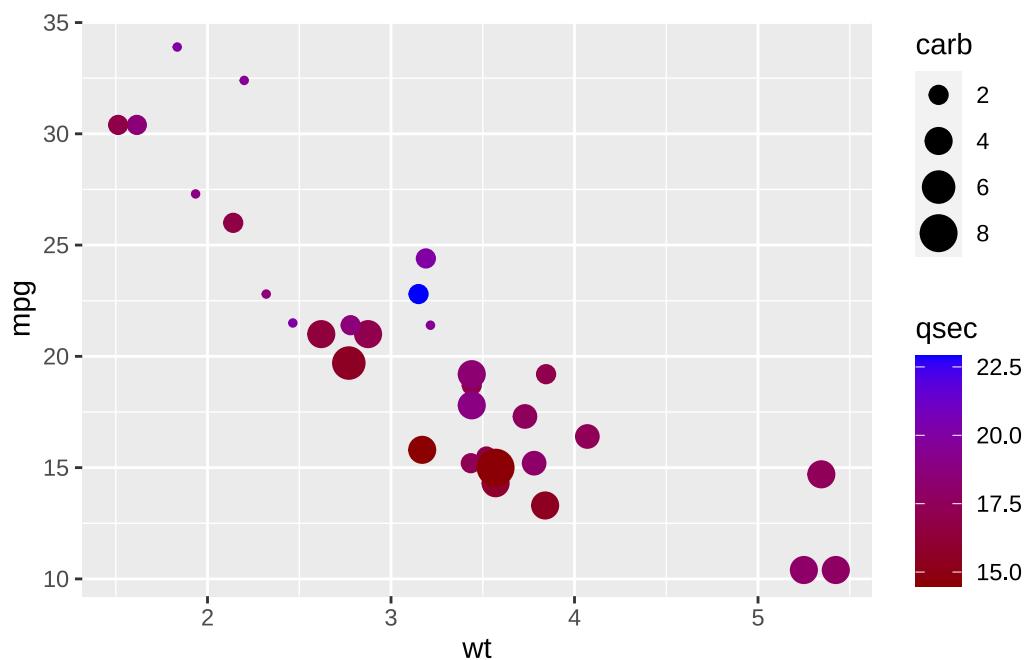
```
ggplot(data=mtcars,aes(wt, mpg,color=qsec))+  
  geom_point(size=4) #outside aes!
```



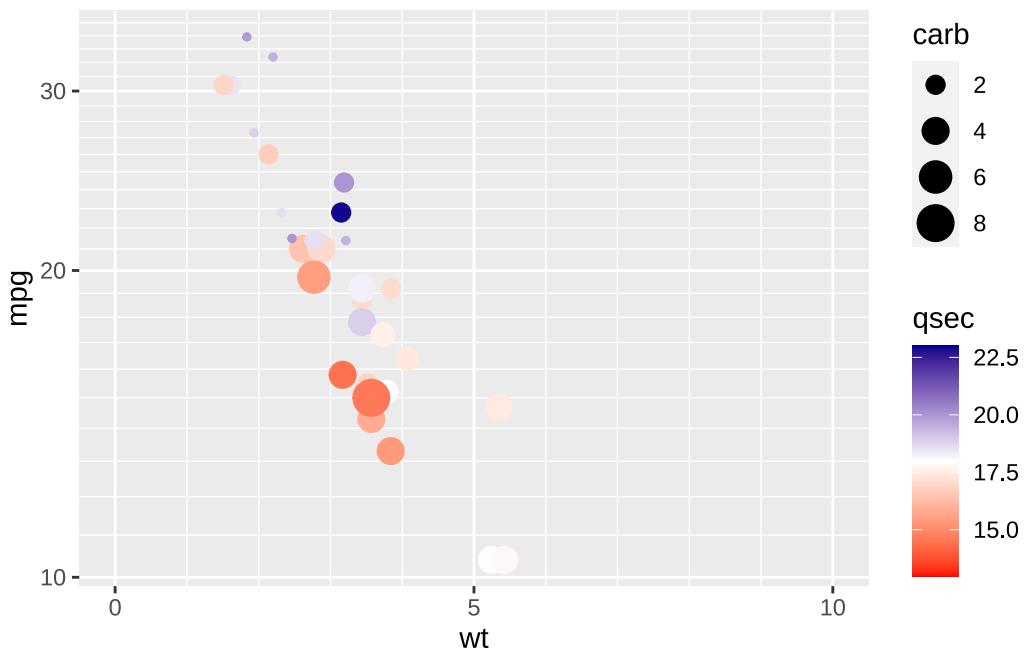
```
ggplot(data=mtcars,aes(wt, mpg,color=qsec, size=carb))+  
  geom_point()
```



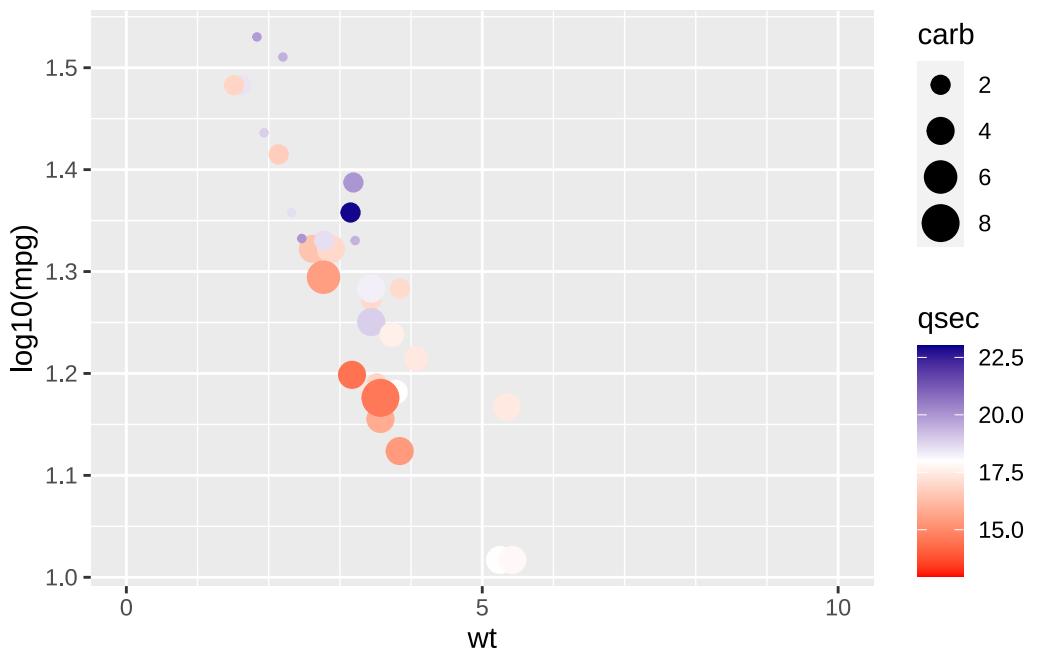
```
ggplot(data=mtcars,aes(wt, mpg,color=qsec, size=carb))+  
  scale_color_gradient(low="darkred",high="blue") +  
  geom_point()
```



```
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(0,100,5),  
    minor_breaks=seq(0,100,1),  
    limits = c(0,10))+  
  scale_y_log10(minor_breaks=seq(0,100,1))
```

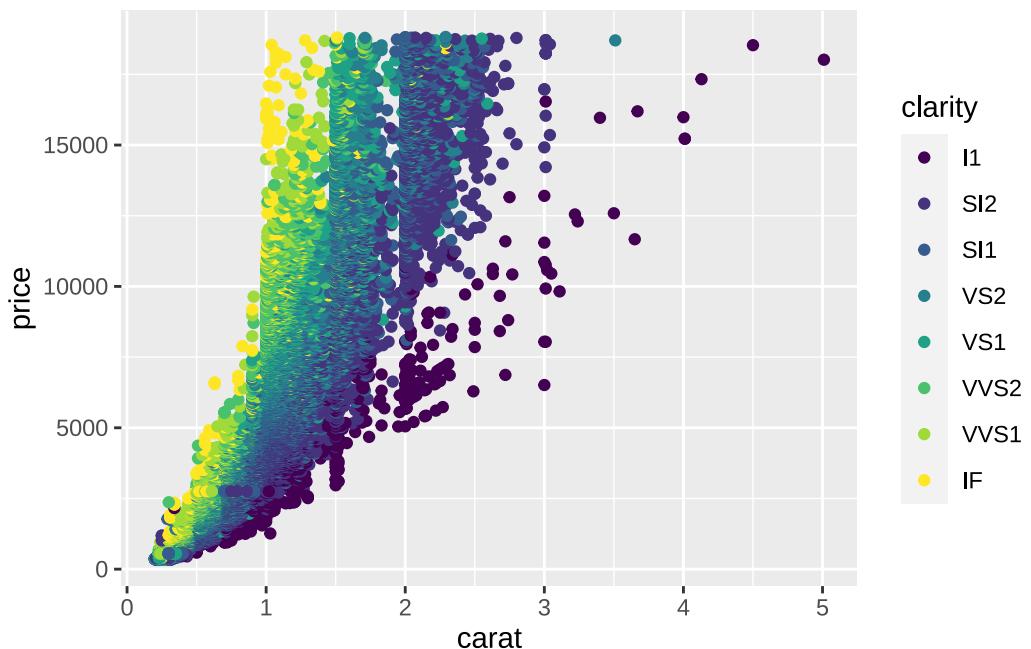


```
ggplot(data=mtcars,aes(wt, log10(mpg),color=qsec, size=carb))+  
  scale_color_gradient2(low="red",high="darkblue",  
    mid="white",  
    limits=c(13,23),midpoint=18)+  
  geom_point() +  
  scale_x_continuous(breaks = seq(0,100,5),  
    minor_breaks=seq(0,100,1),  
    limits = c(0,10))#+
```

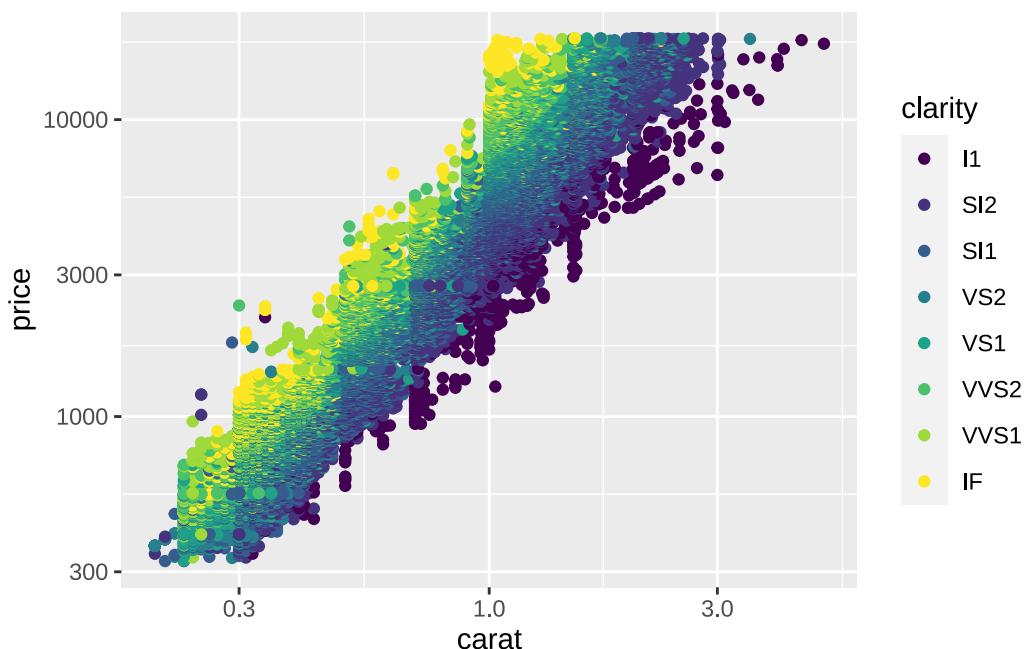


```
# scale_y_log10(minor_breaks=seq(0,100,1))

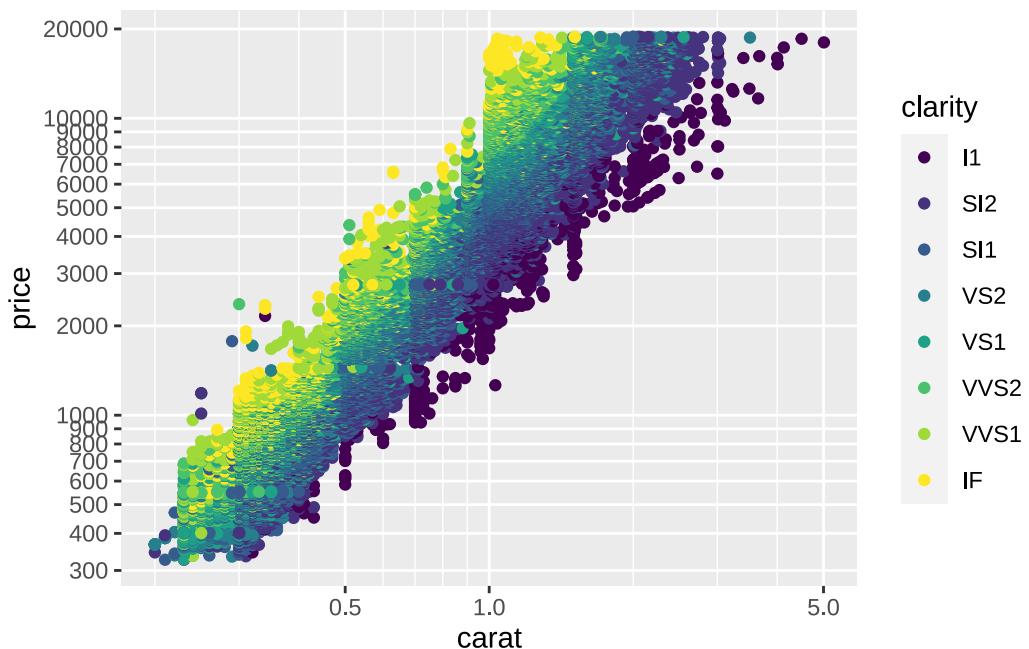
ggplot(diamonds,aes(carat,price,color=clarity))+  
  geom_point()
```



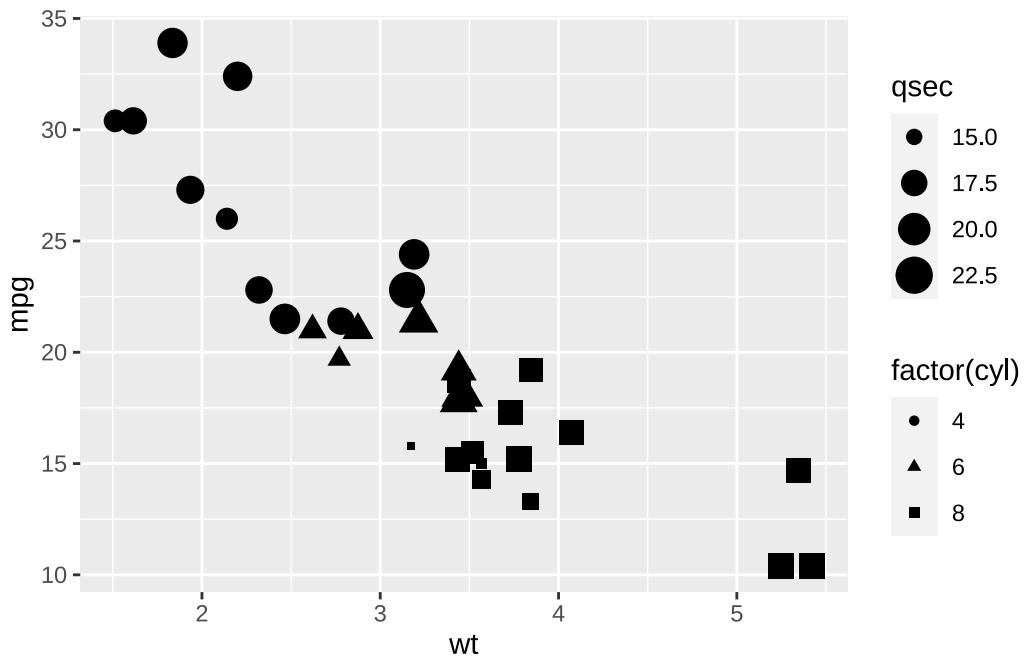
```
ggplot(diamonds,aes(carat,price,color=clarity))+  
  geom_point() +  
  scale_x_log10() +  
  scale_y_log10()
```



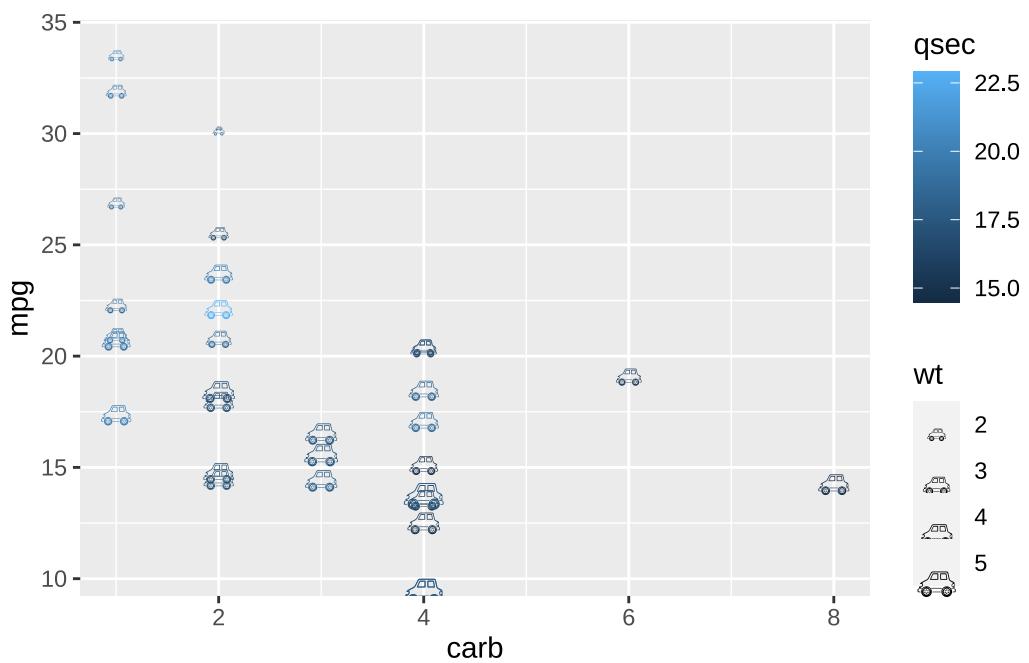
```
ggplot(diamonds,aes(carat,price,color=clarity))+  
  geom_point() +  
  scale_x_log10(  
    breaks=logrange_15,  
    minor_breaks=logrange_123456789) +  
  scale_y_log10(  
    breaks=logrange_123456789,  
    minor_breaks=logrange_123456789)
```



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



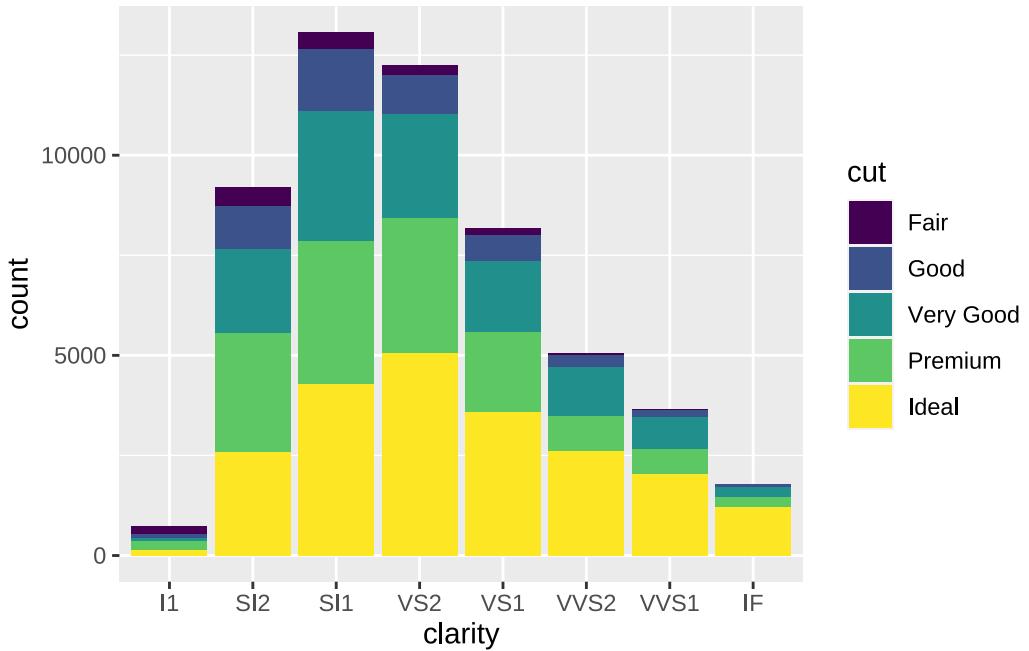
```
ggplot(data=mtcars,aes(carb, mpg,color=qsec, size=wt))+  
  geom_text(family="EmojiOne",label="\U1F697")
```



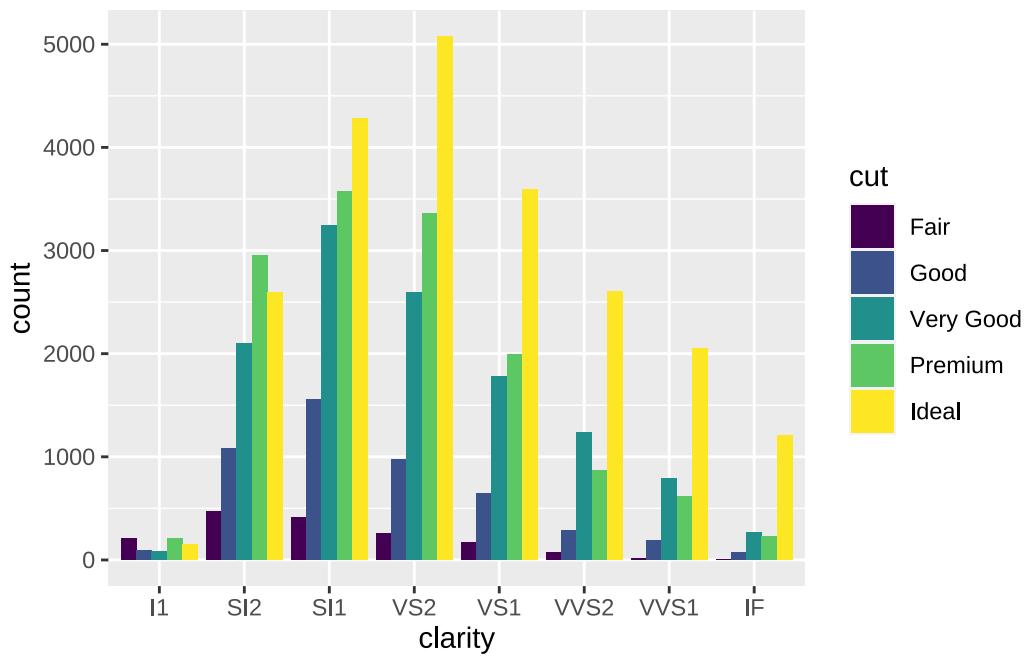
6.8 Positioning elements

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.

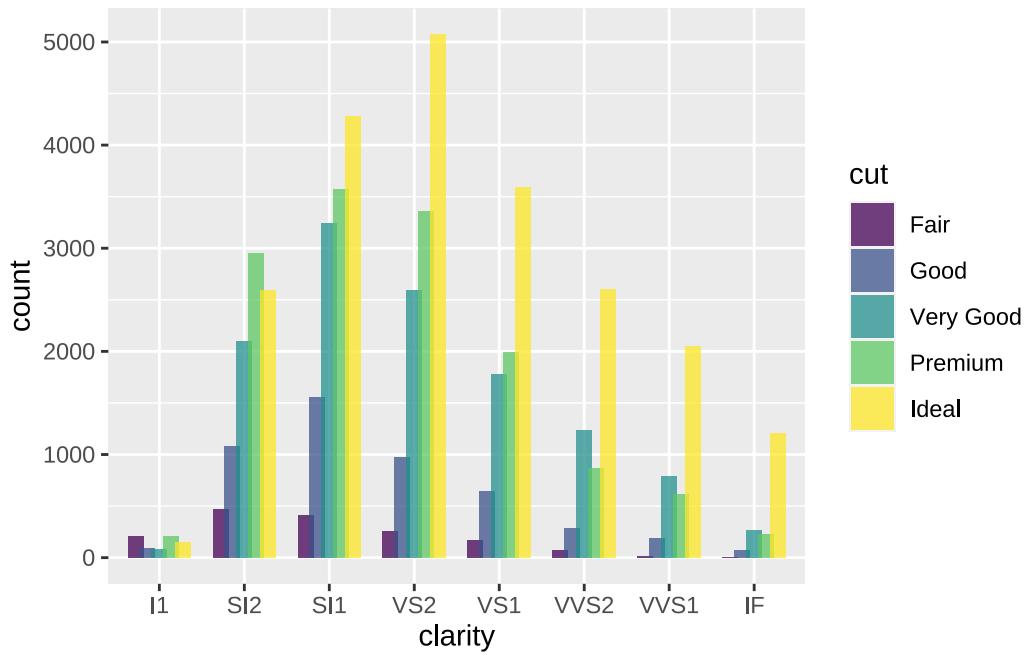
```
p<-ggplot(data=diamonds,aes(clarity,fill=cut))  
p+geom_bar(position="stack") # default for bar
```



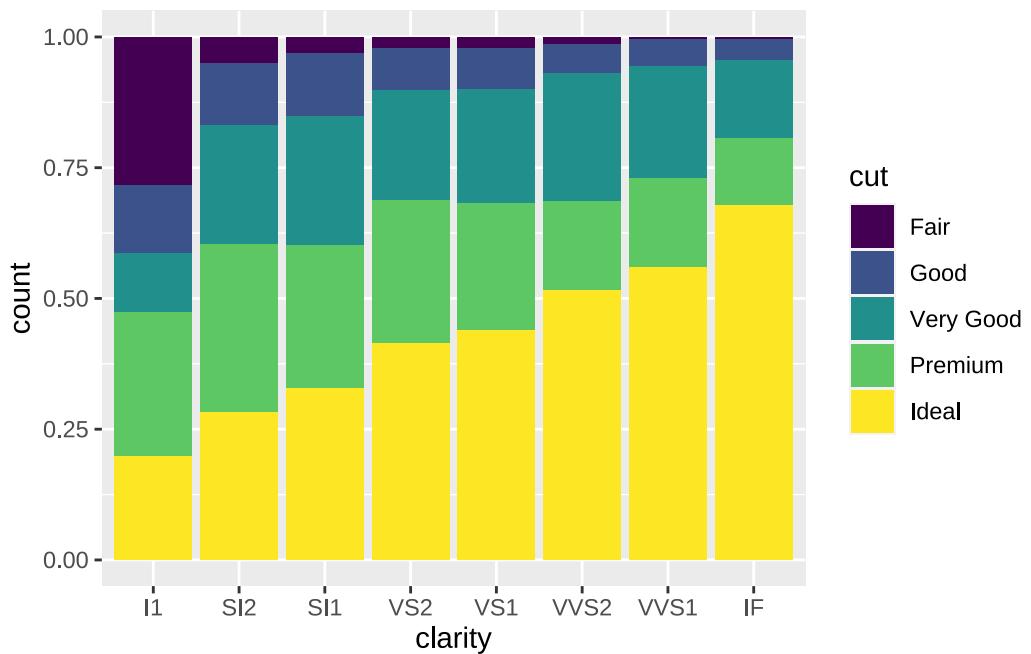
```
p+geom_bar(position="dodge")
```



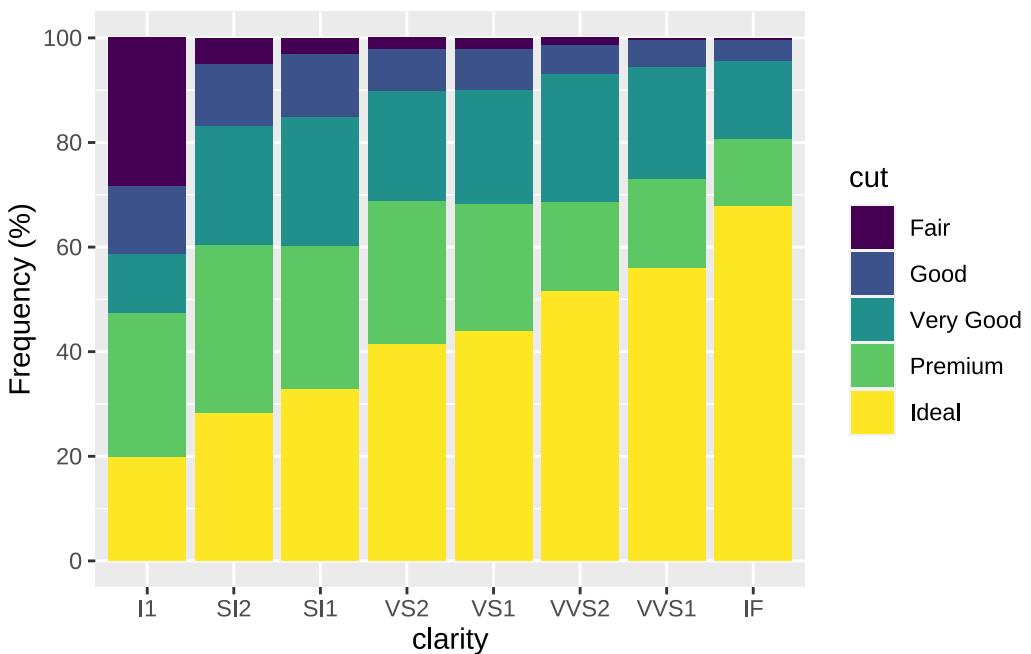
```
p+geom_bar(position=position_dodge(width = 0.7), alpha=.75)
```



```
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))
```

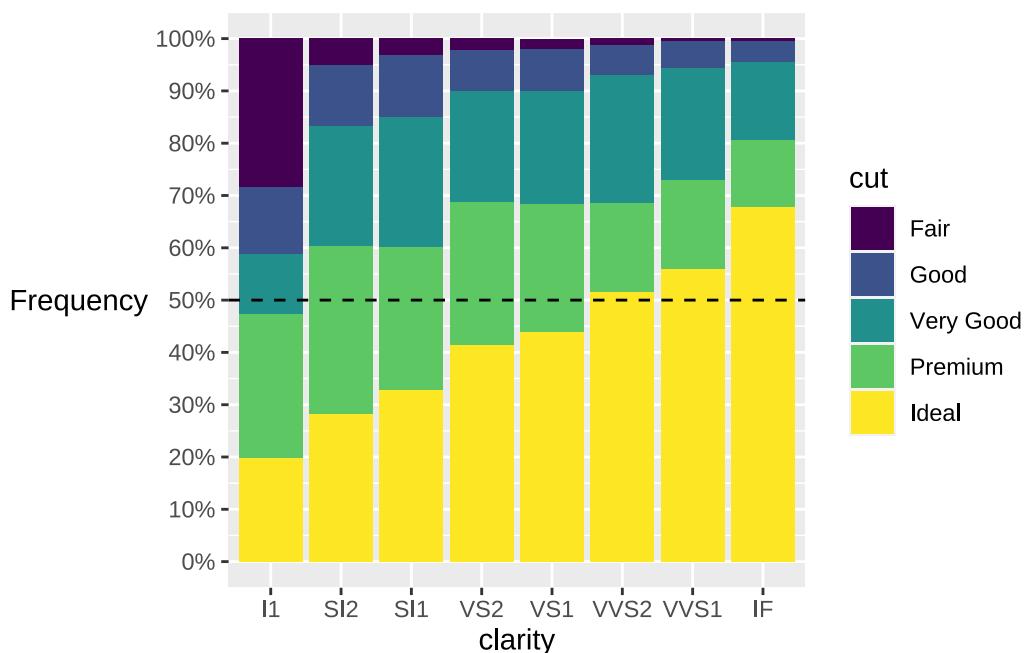


```

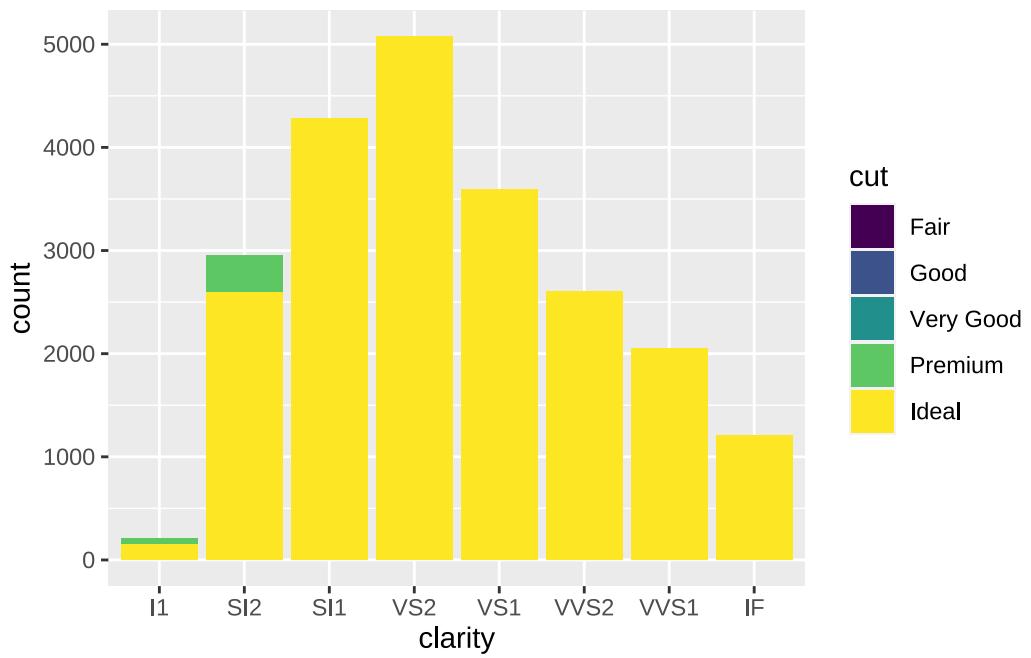
p+geom_bar(position="fill")+
  scale_y_continuous("Frequency",
                     breaks=seq(0,1,.1),
                     labels=scales::percent)+
  theme(axis.title.y = element_text(angle = 0,
                                    vjust = .5))+  

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

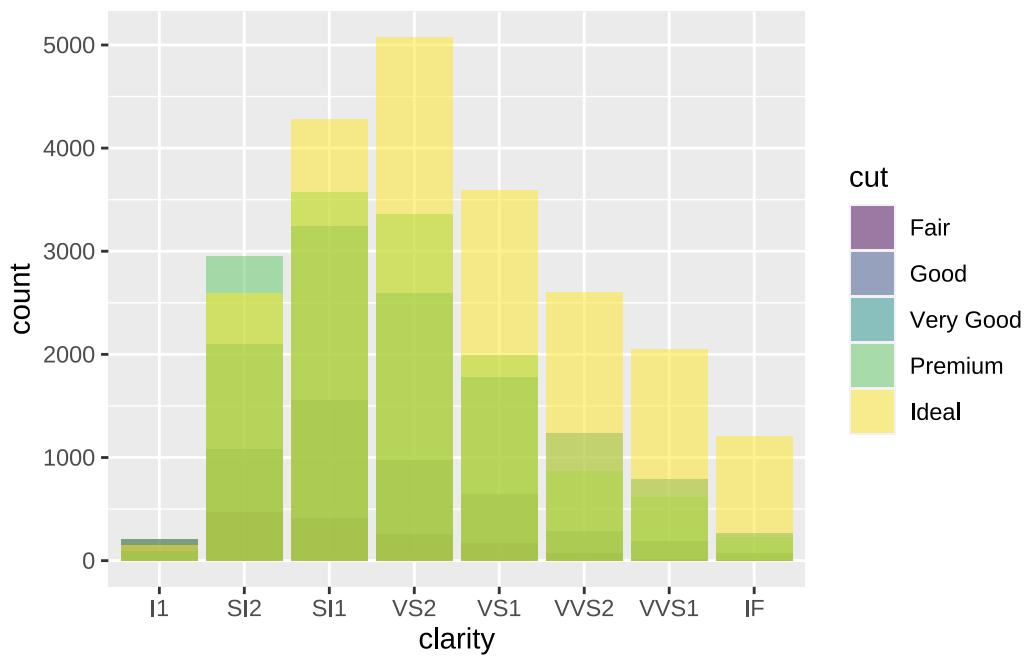
```



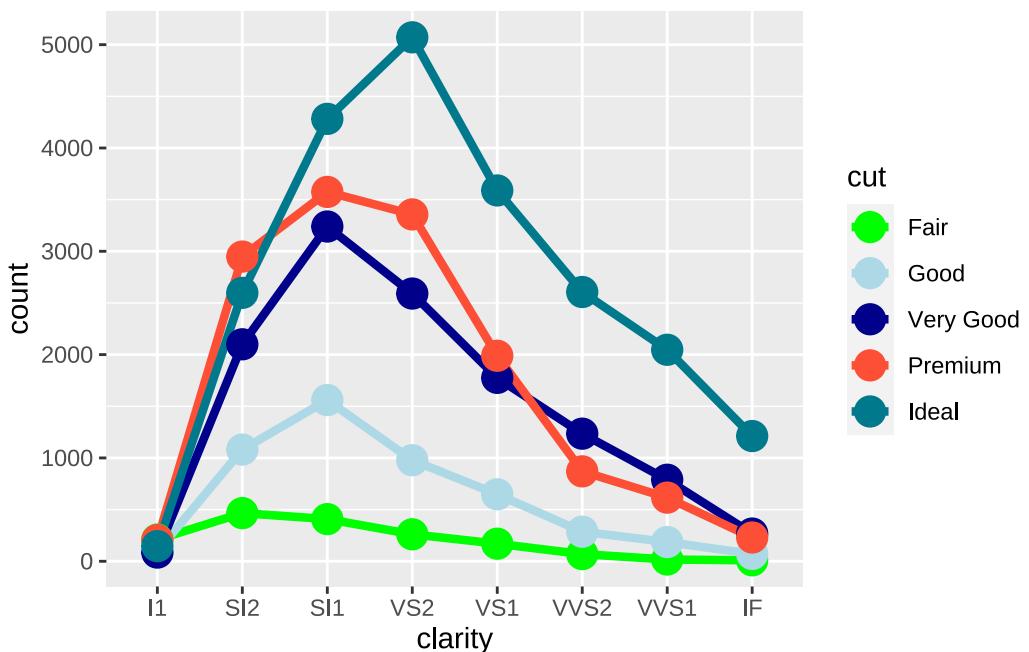
```
p+geom_bar(position="identity") # bad idea!
```



```
p+geom_bar(position="identity",alpha=.5)
```



```
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"))
```



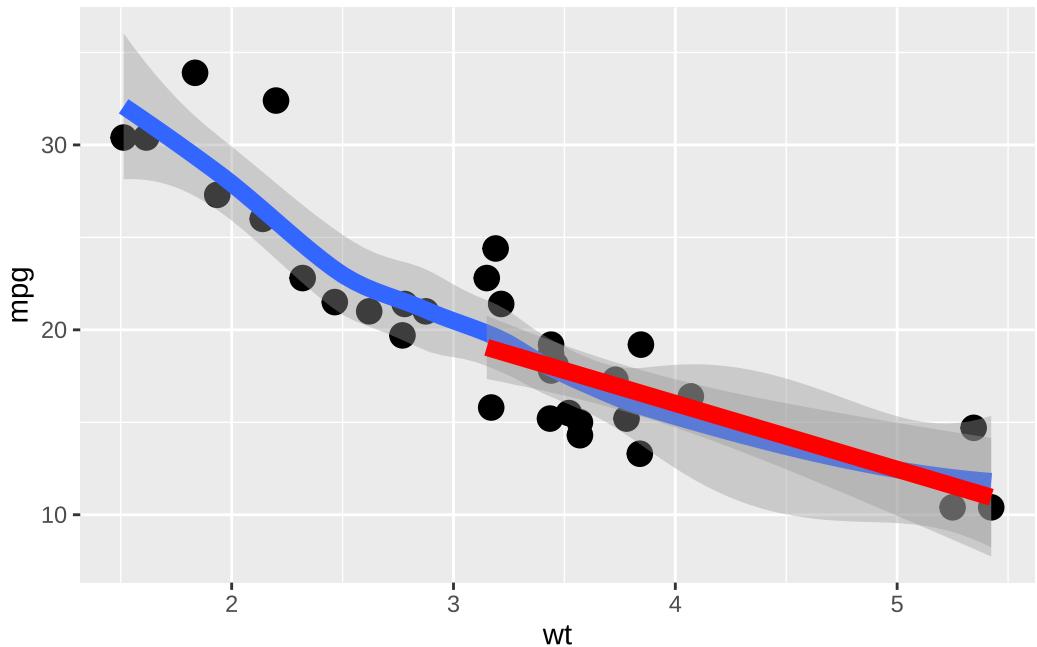
6.9 Order of layers

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

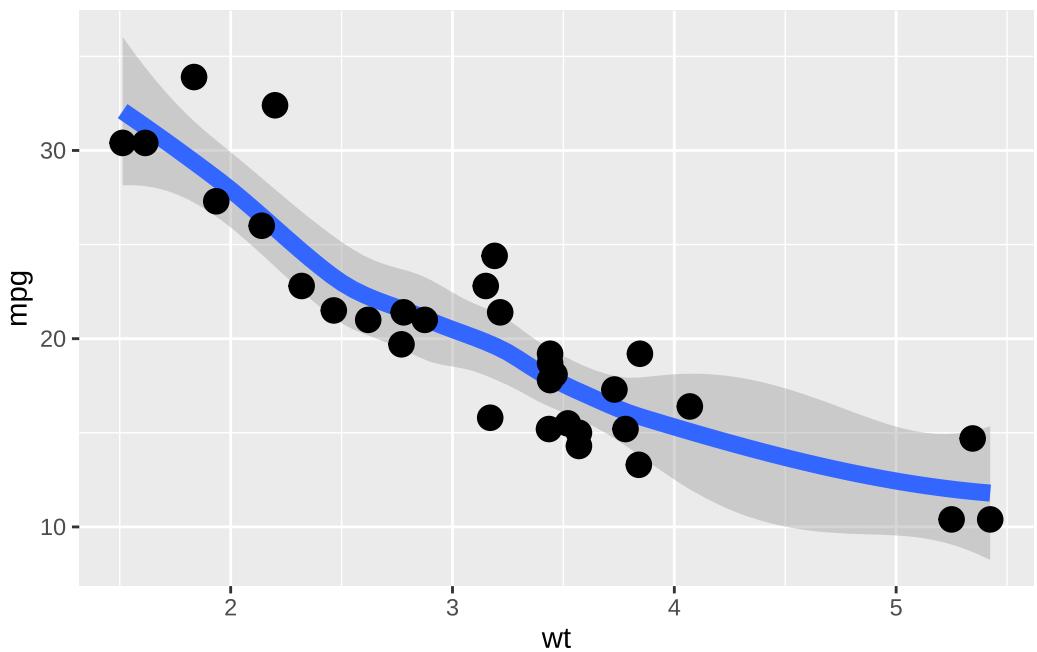
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.

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



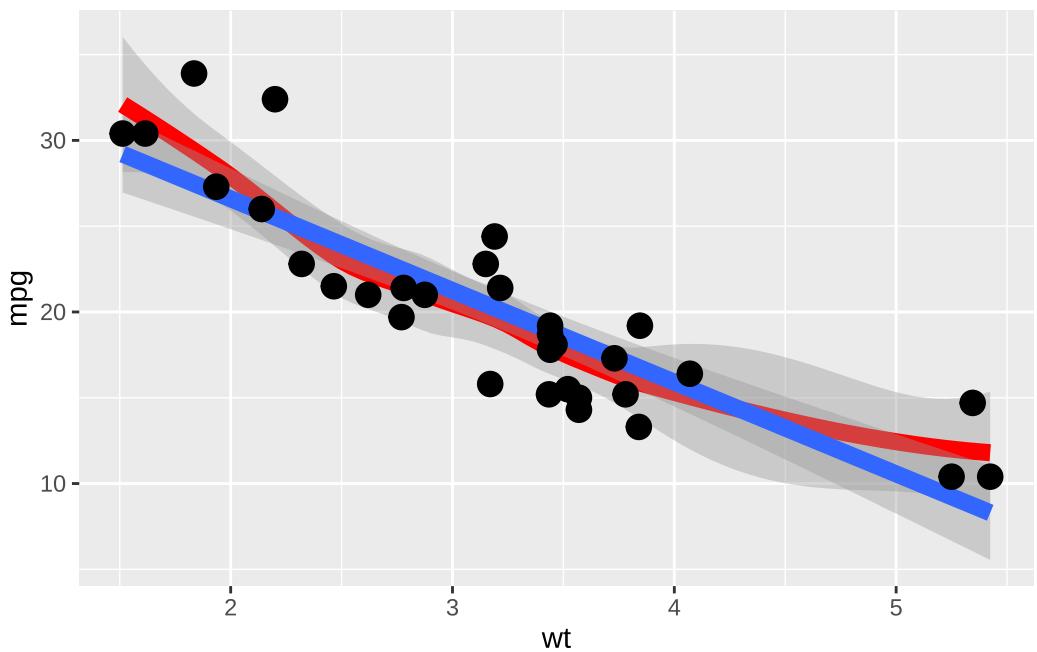
```
ggplot(data=mtcars,aes(wt, mpg))+  
  geom_smooth(linewidth=3)+  
  geom_point(size=4)
```

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



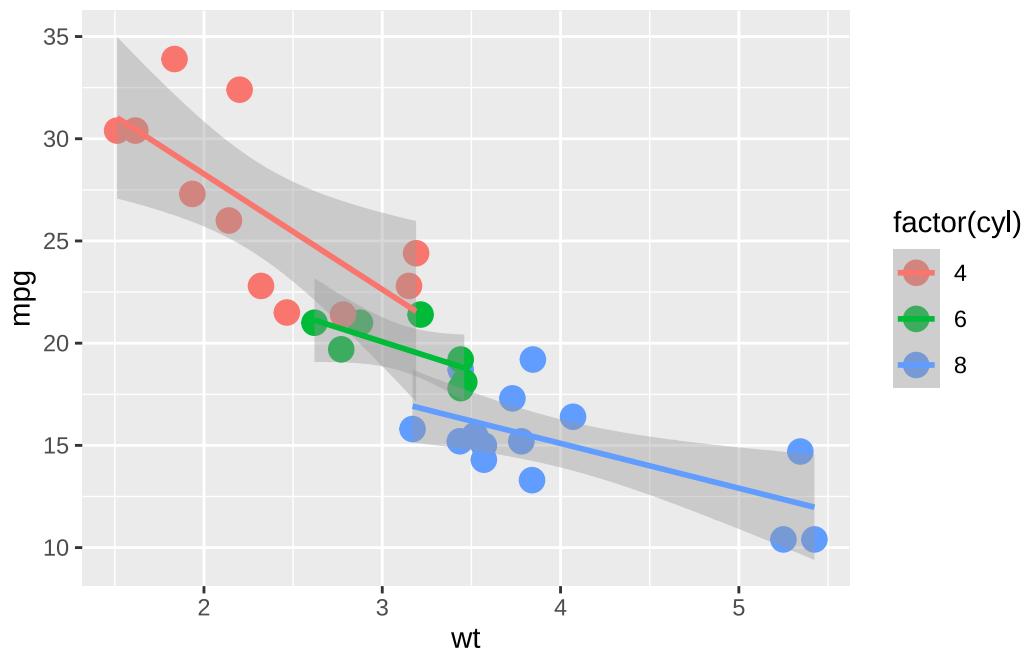
```
ggplot(data=mtcars,aes(wt, mpg))+  
  geom_smooth(linewidth=3,color="red") +  
  geom_smooth(method="lm", linewidth=3) +  
  geom_point(size=4)
```

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



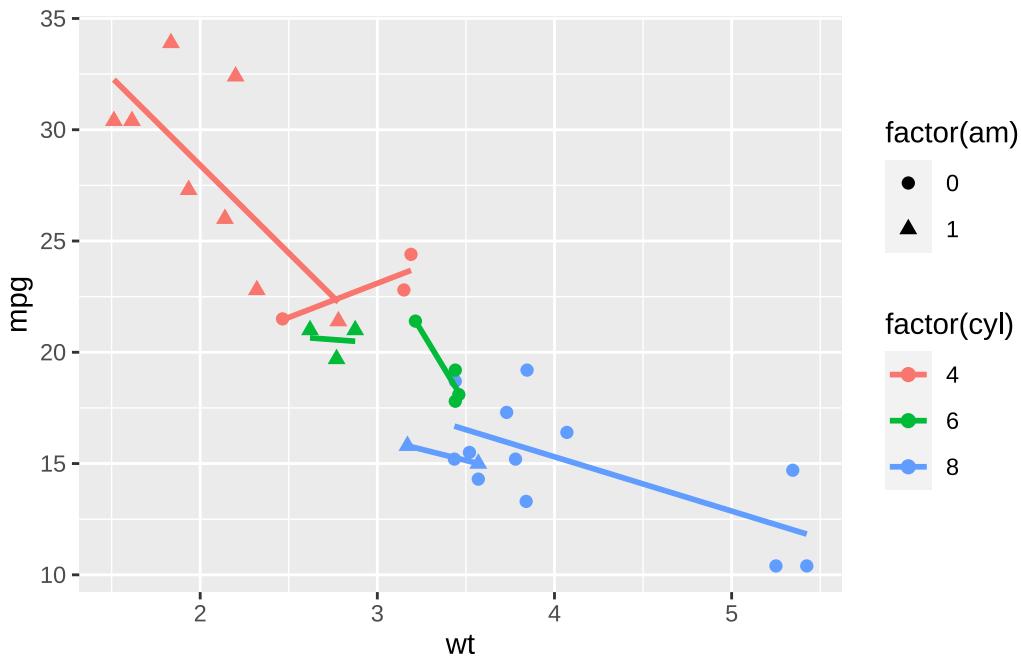
```
ggplot(data=mtcars,aes(wt, mpg,
                        color=factor(cyl)))+
  geom_point(size=4)+
  geom_smooth(method="lm", linewidth=1)
```

```
`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)
```

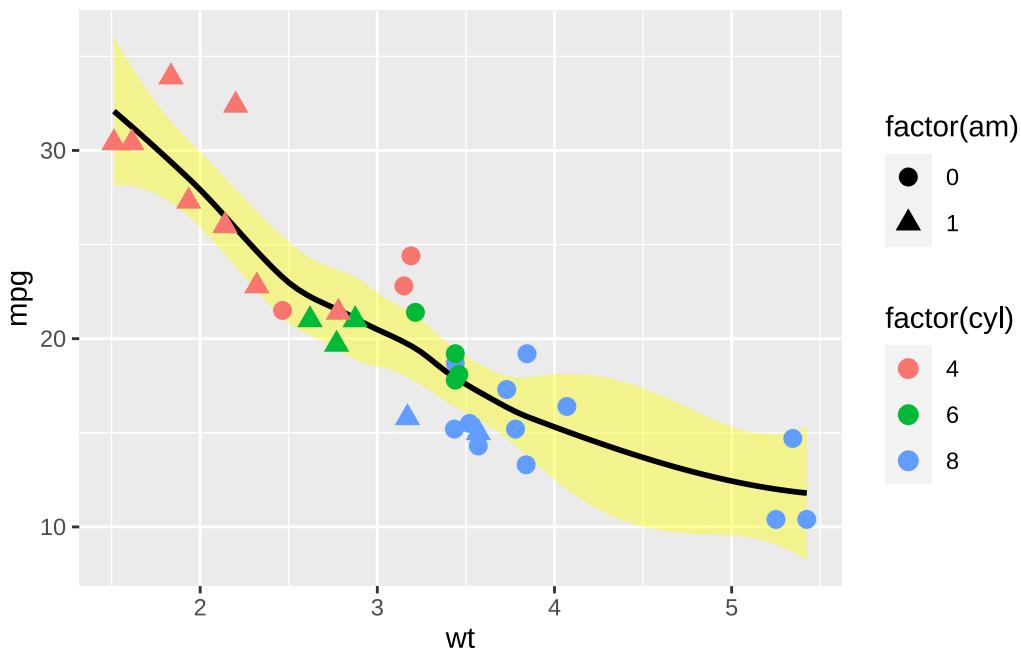
```
`geom_smooth()` using formula = 'y ~ x'
```



6.10 Local aesthetics for layers

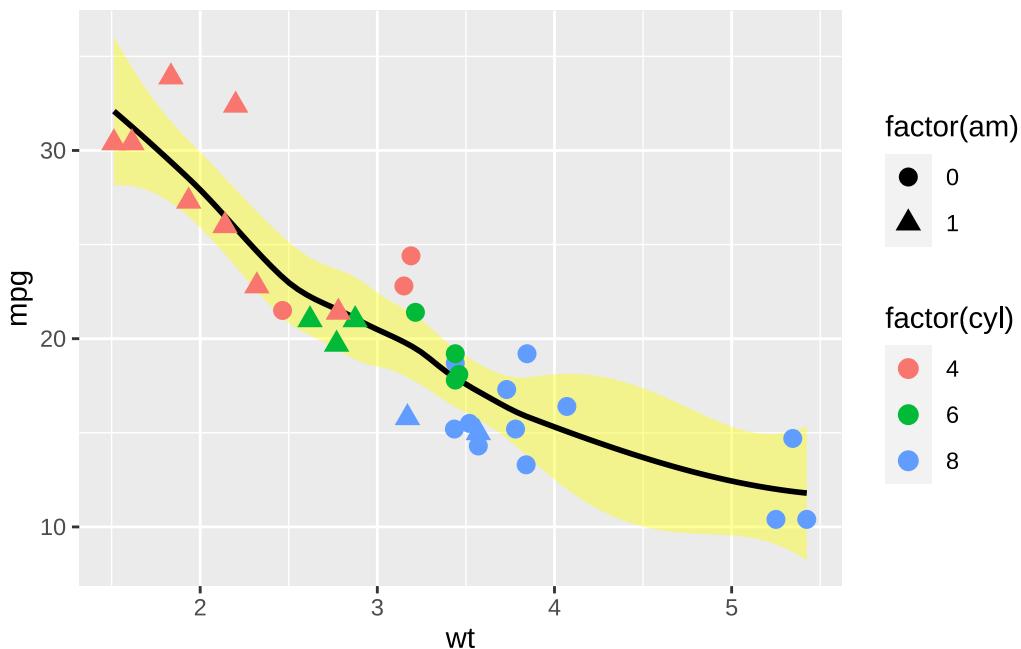
```
#? lm for all?
ggplot(data=mtcars,aes(wt, mpg))+
  geom_smooth(size=1,color="black",fill="yellow")+
  geom_point(size=3,aes(color=factor(cyl),shape=factor(am))) #aes for geom only
```

```
`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'
```



6.11 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`).

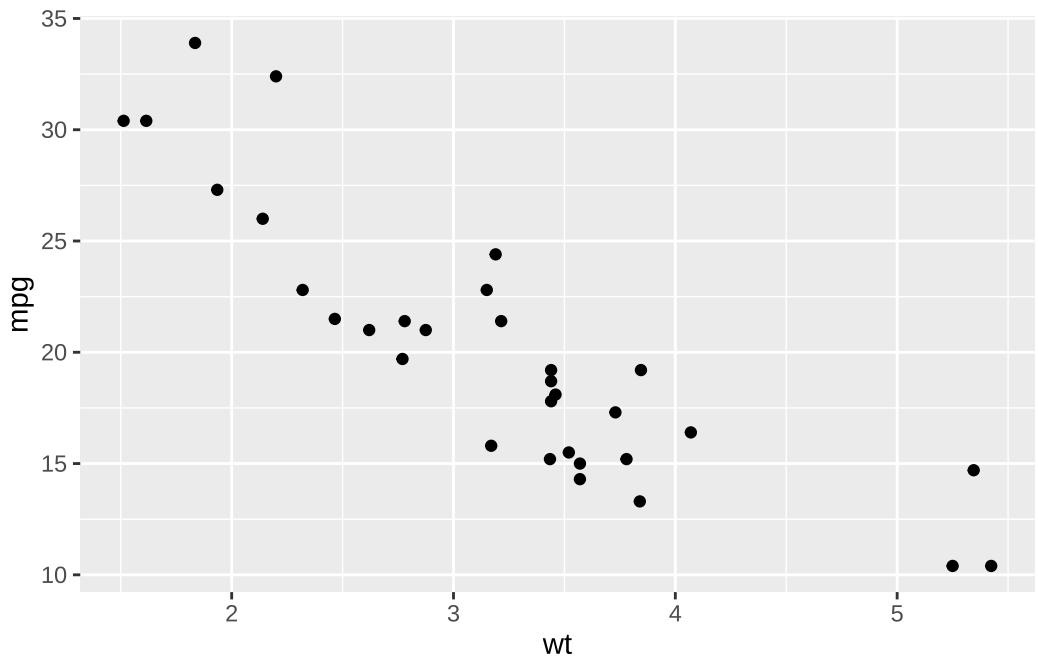
6.11.1 `facet_grid`

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

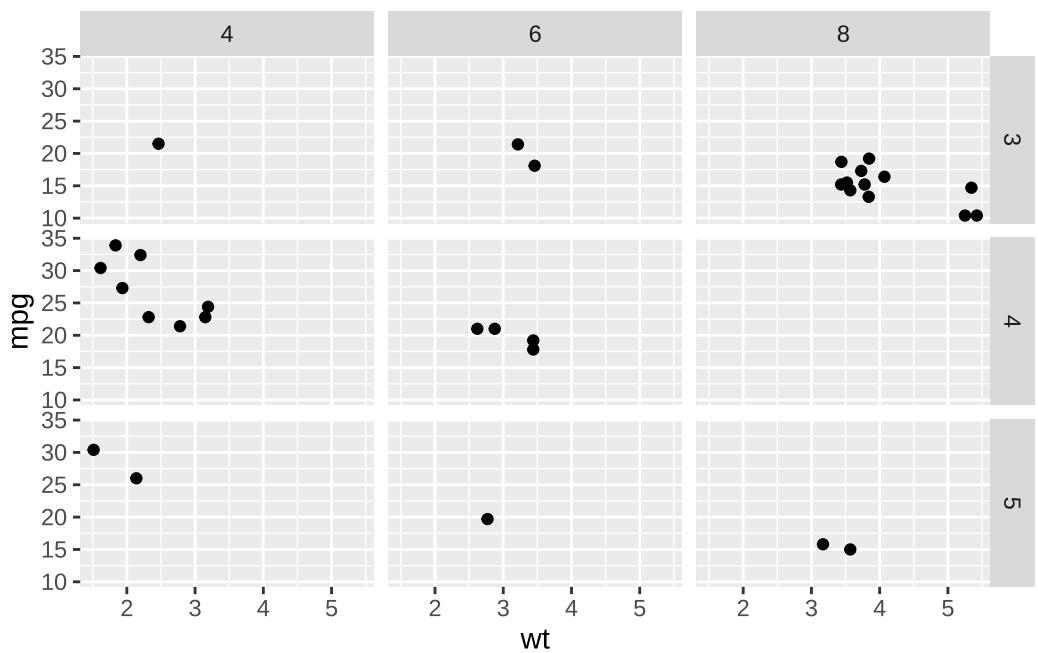
Labeling of facets often requires name and content to be informative.

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

```
(p.tmp <- ggplot(mtcars, aes(wt, mpg)) +
  geom_point())
```



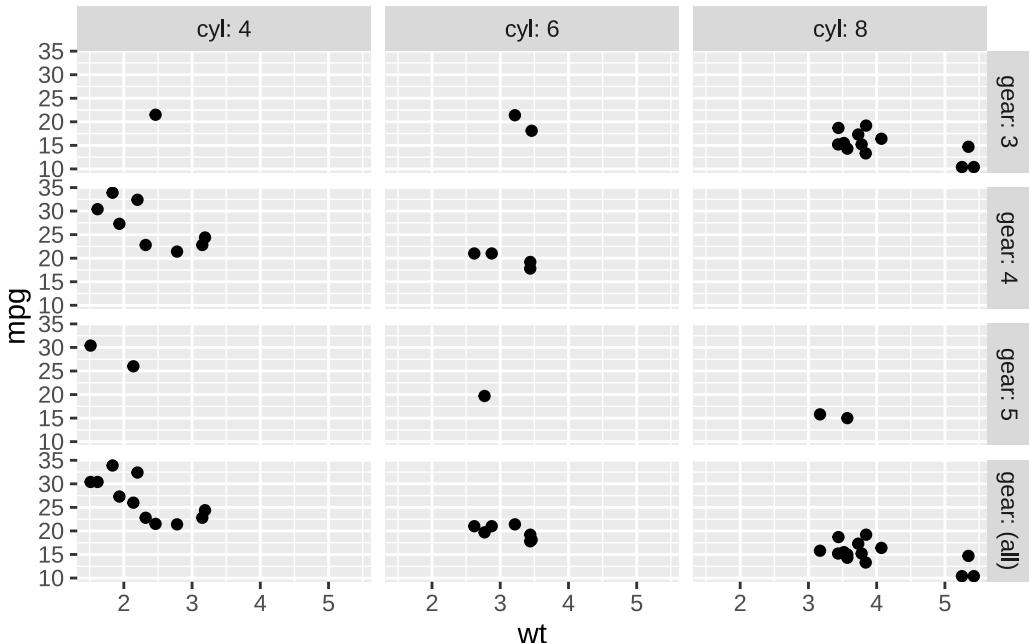
```
p.tmp + facet_grid(rows = vars(gear),  
                    cols = vars(cyl))
```



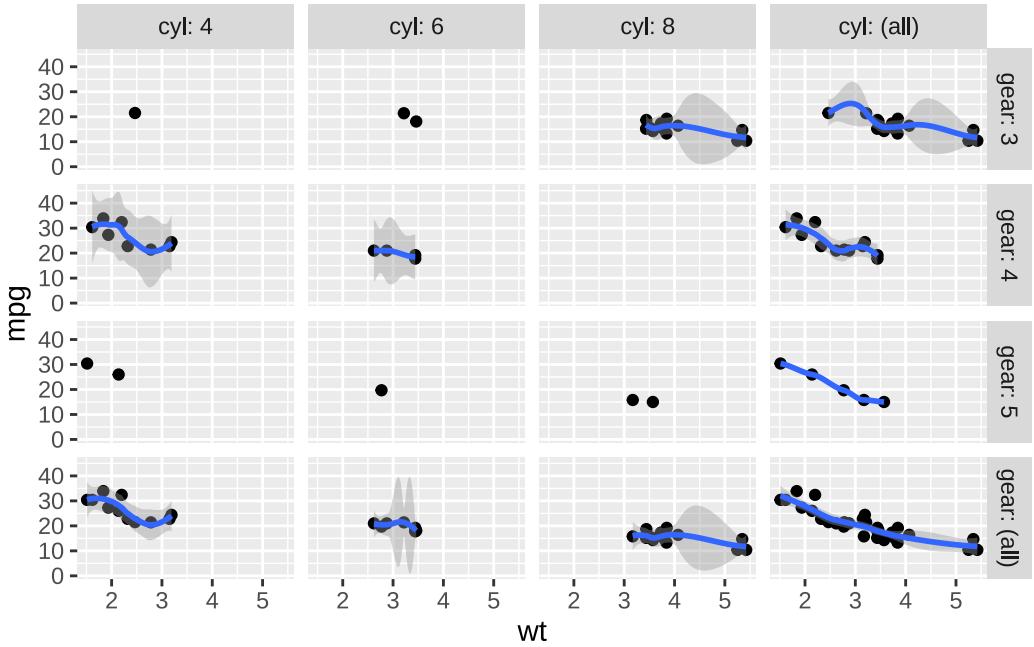
```
cat("facet labeling improved:\n")
```

facet labeling improved:

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



```
# options(warn=-1)  
p.tmp + geom_smooth() +  
  facet_grid(rows = vars(gear),  
             cols = vars(cyl),  
             labeller=label_both, margins=TRUE)
```

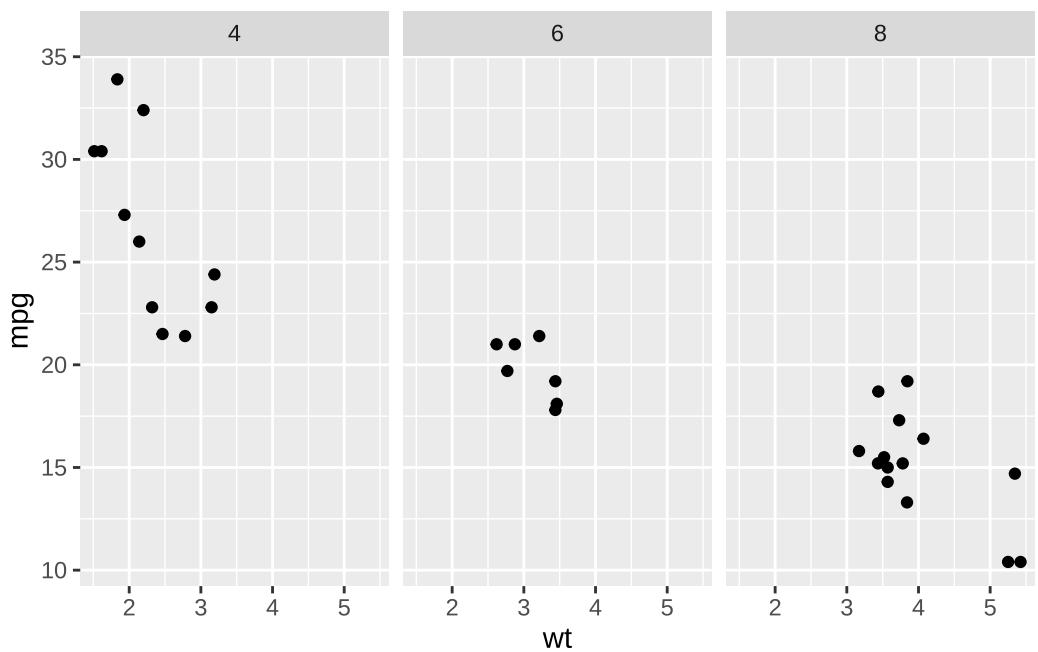


```
# options(warn=0)
```

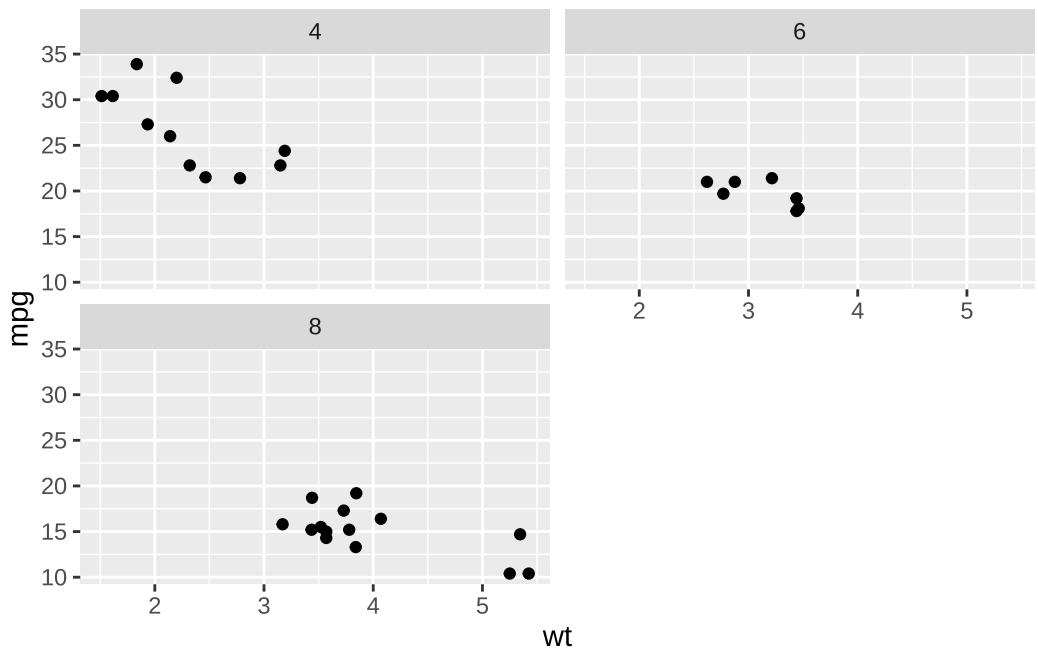
6.11.2 facet_wrap

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

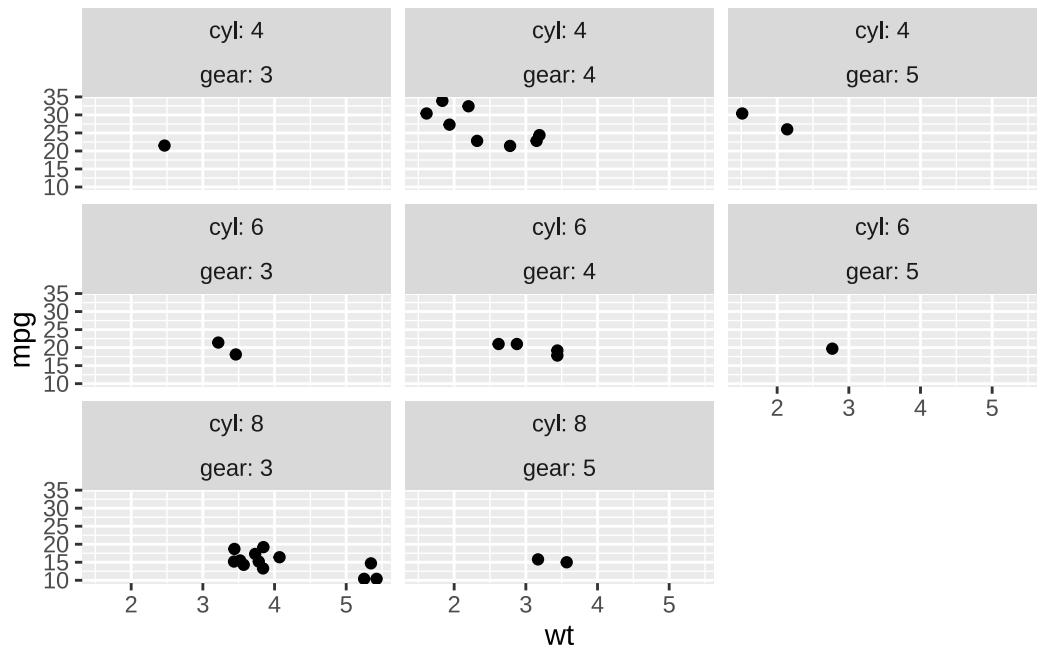
```
p.tmp + facet_wrap(facets = vars(cyl))
```



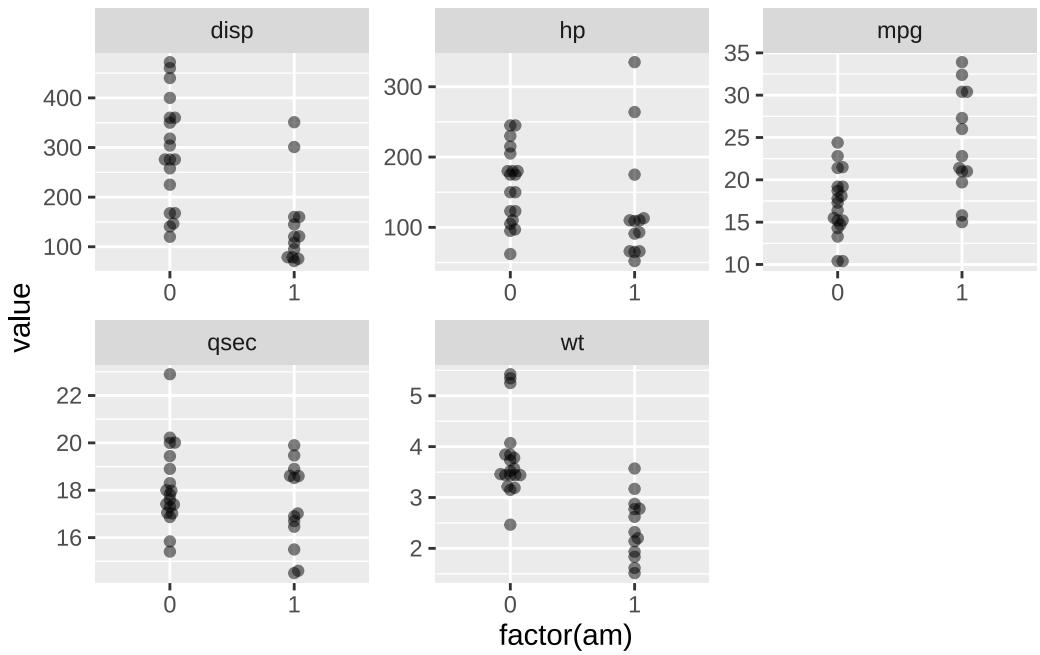
```
p.tmp + facet_wrap(facets = vars(cyl), ncol=2)
```



```
# empty combination is dropped
p.tmp + facet_wrap(facets=vars(cyl,gear),labeller=label_both)
```

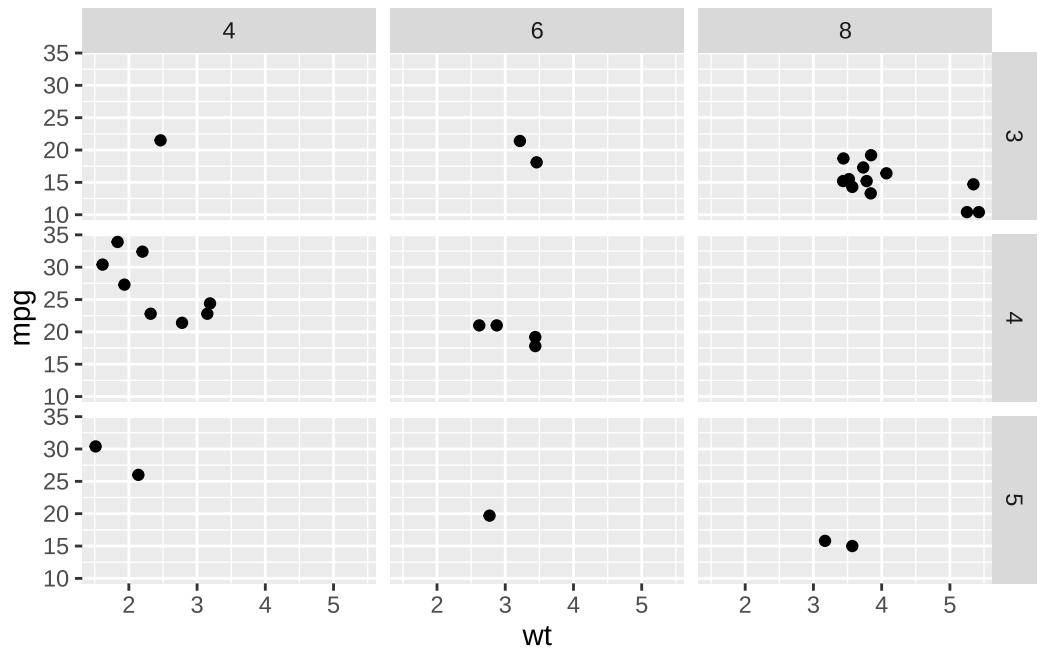


```
#combining variables
mtcars |>
  pivot_longer(cols = c(wt, mpg, hp, disp, qsec)) |>
  ggplot(aes(x=factor(am), y=value))+
  geom_beeswarm(alpha=.5, cex=2)+
  facet_wrap(facets = vars(name), scales="free")
```

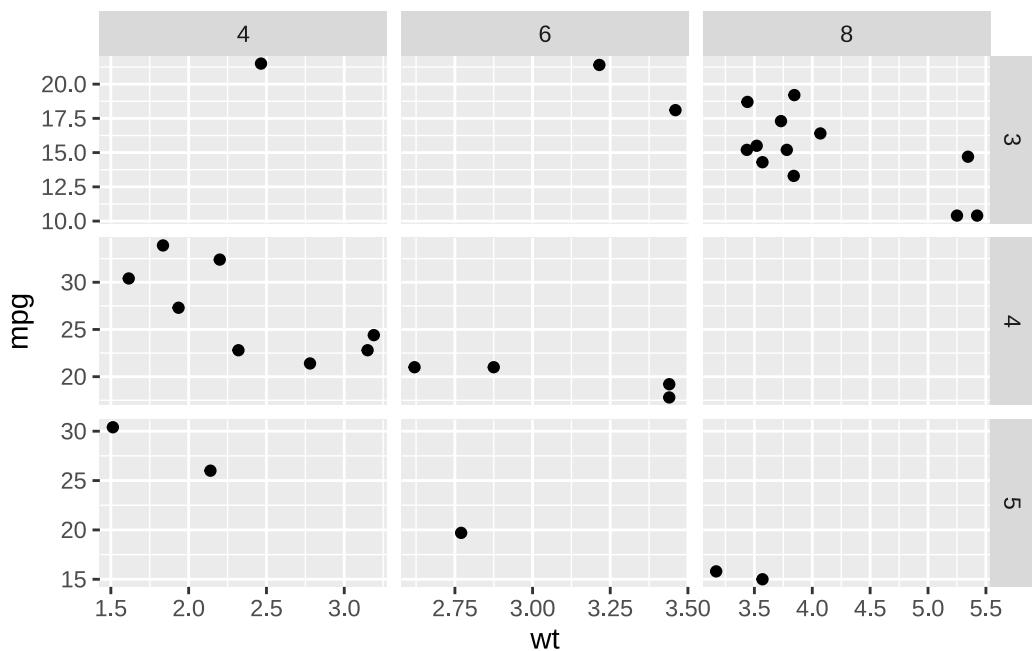


6.11.3 Controlling scales in facets (default: `scales="fixed"`)

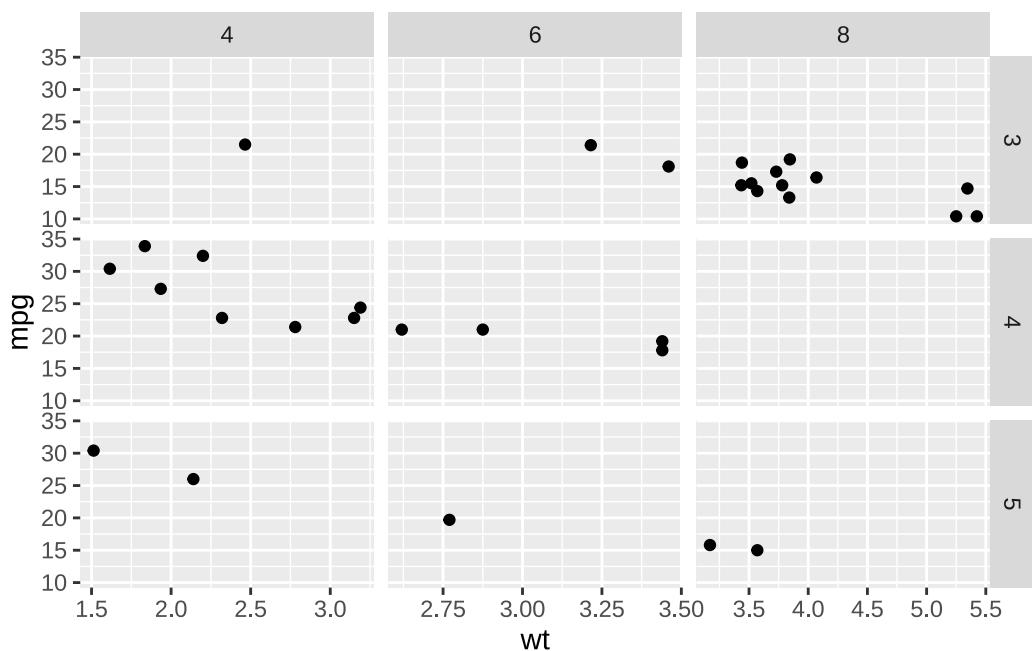
```
p.tmp + facet_grid(gear~cyl, scales="fixed")
```



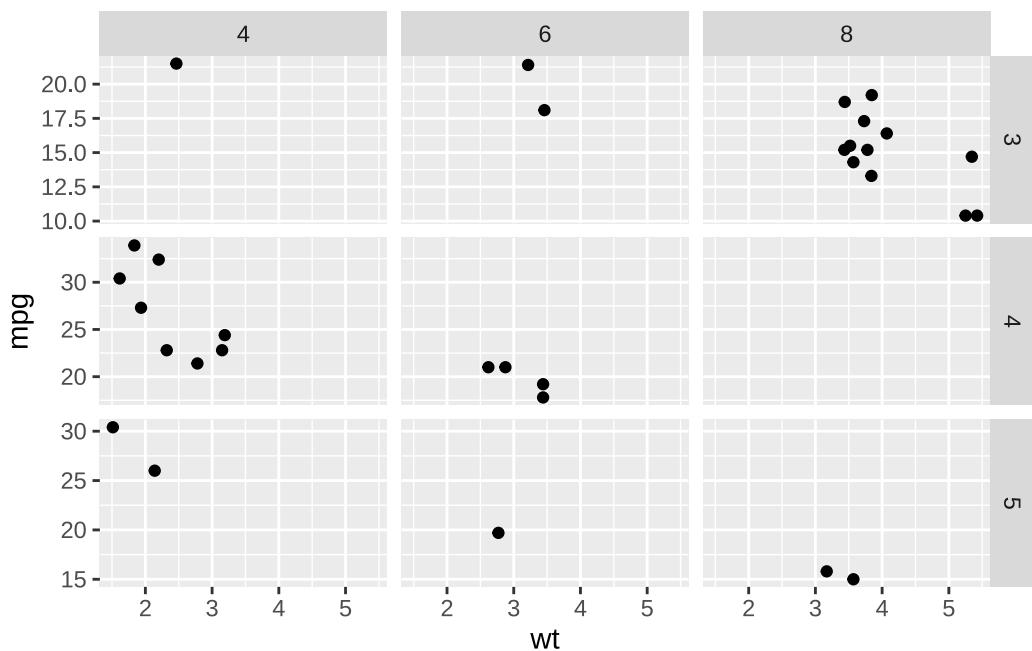
```
p.tmp + facet_grid(gear~cyl, scales="free")
```



```
p.tmp + facet_grid(gear~cyl, scales="free_x")
```



```
p.tmp + facet_grid(gear~cyl, scales="free_y")
```

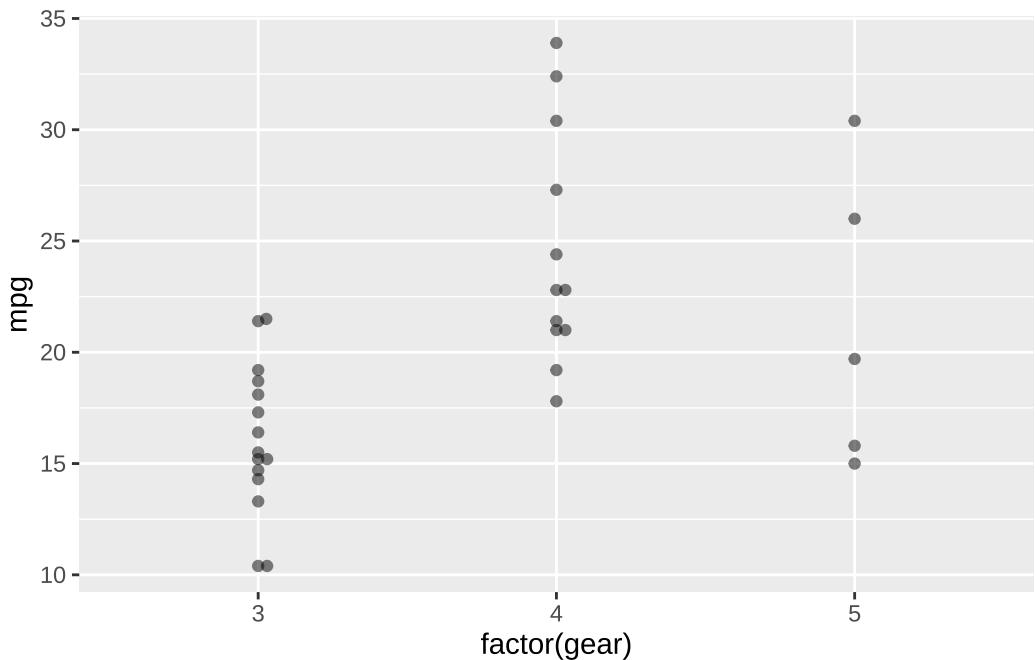


6.12 Showing summaries

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!!

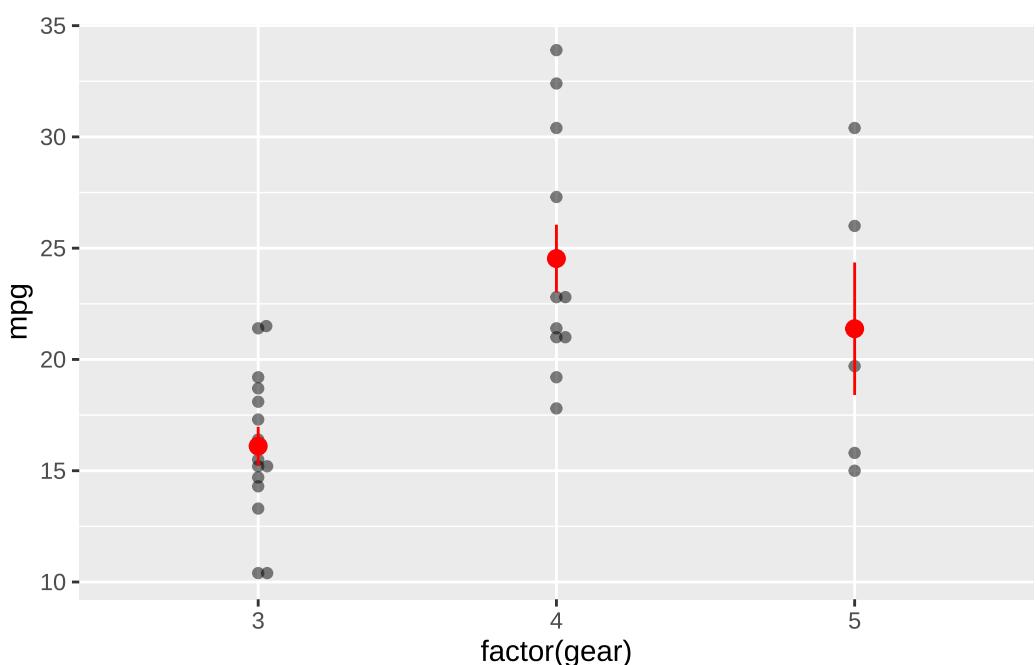
Functions for summary statistics (mean_se, mean_cl_normal, mean_cl_boot etc.) are build on top of Hmisc functions. So this package is needed but not automatically installed with ggplot2.

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

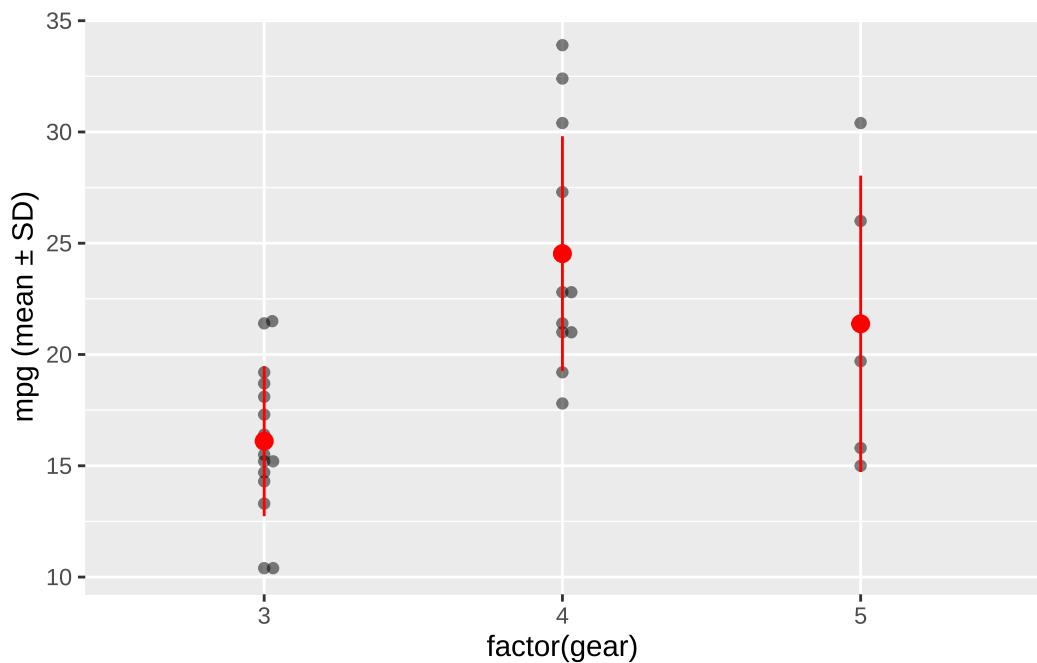


```
plottemp+stat_summary(color="red")
```

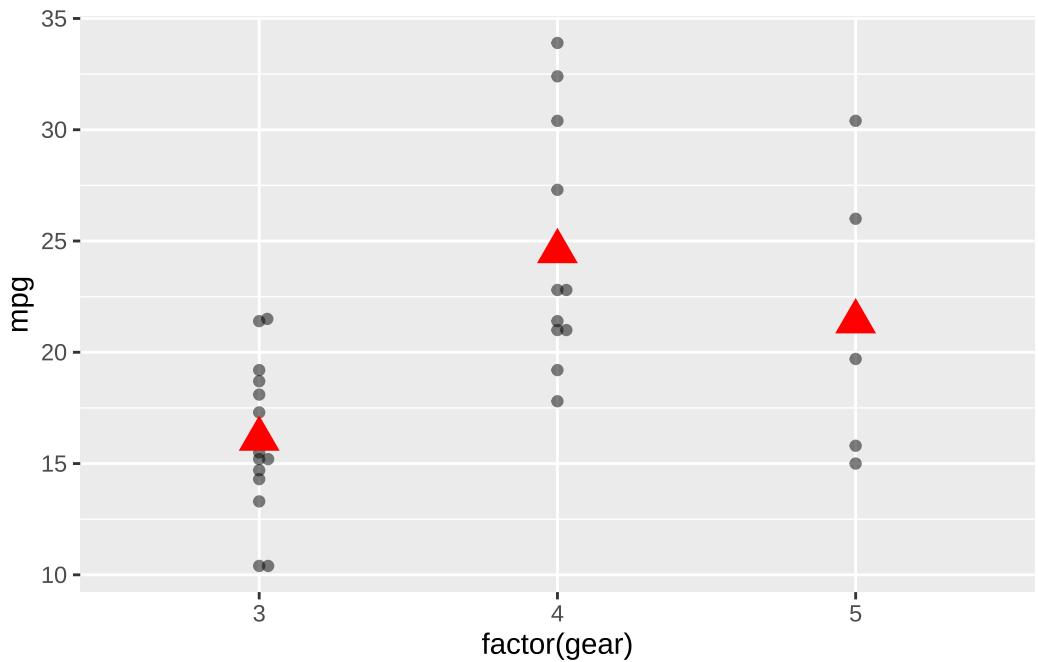
No summary function supplied, defaulting to `mean_se()`



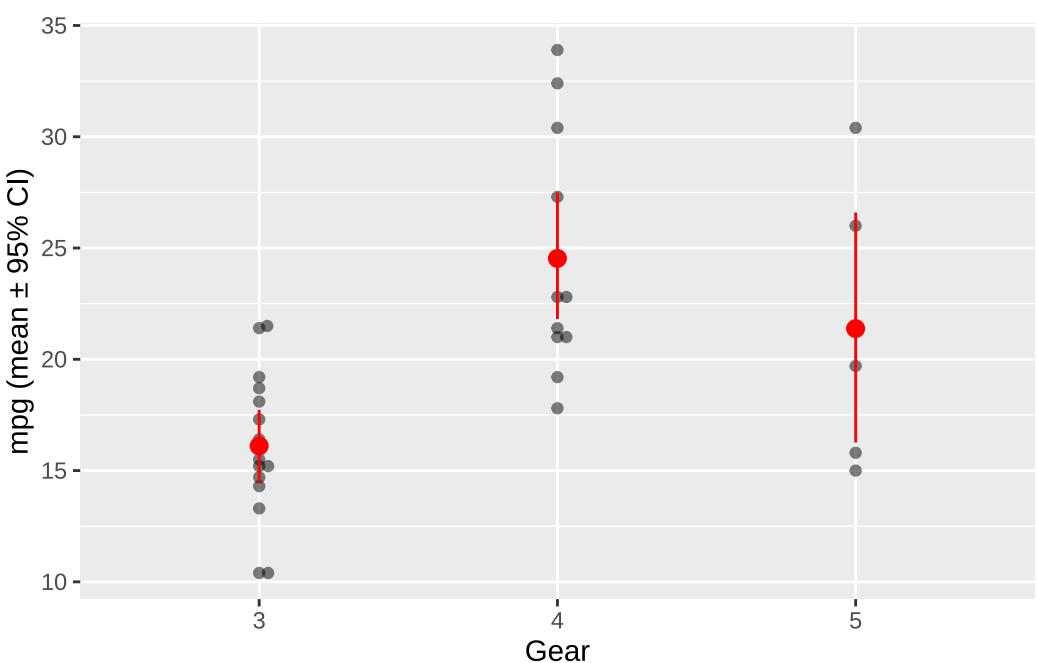
```
plottemp+stat_summary(fun.data="mean_sdl",
                      fun.args=list(mult=1),
                      color="red")+
  ylab("mpg (mean \u00b1 SD)")
```



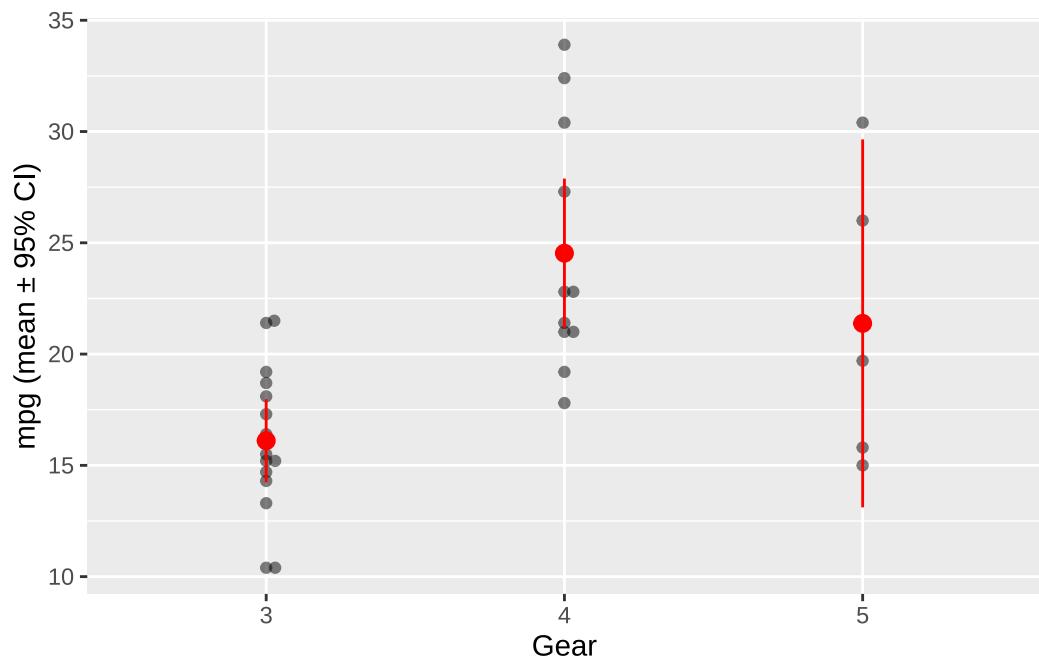
```
plottemp+stat_summary(geom = "point", shape=17, size=5,
                      fun = "mean", color="red")
```



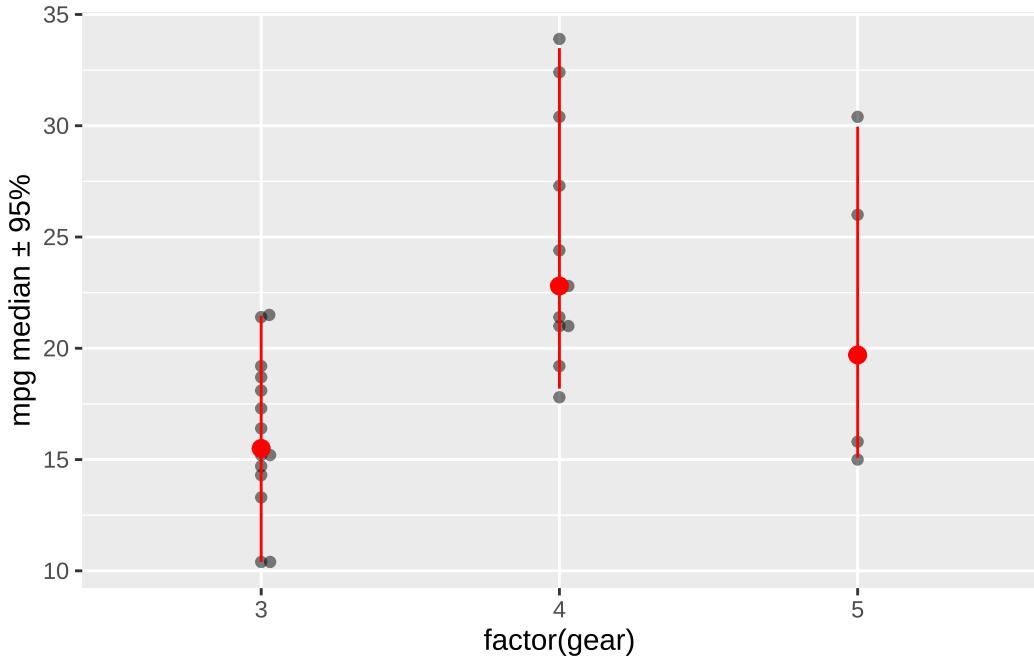
```
plottemp+stat_summary(fun.data="mean_cl_boot",
                      fun.args=list(B=10^4),
                      color="red")+
  ylab("mpg (mean \u00b1 95% CI)")+
  xlab("Gear")
```



```
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()
```

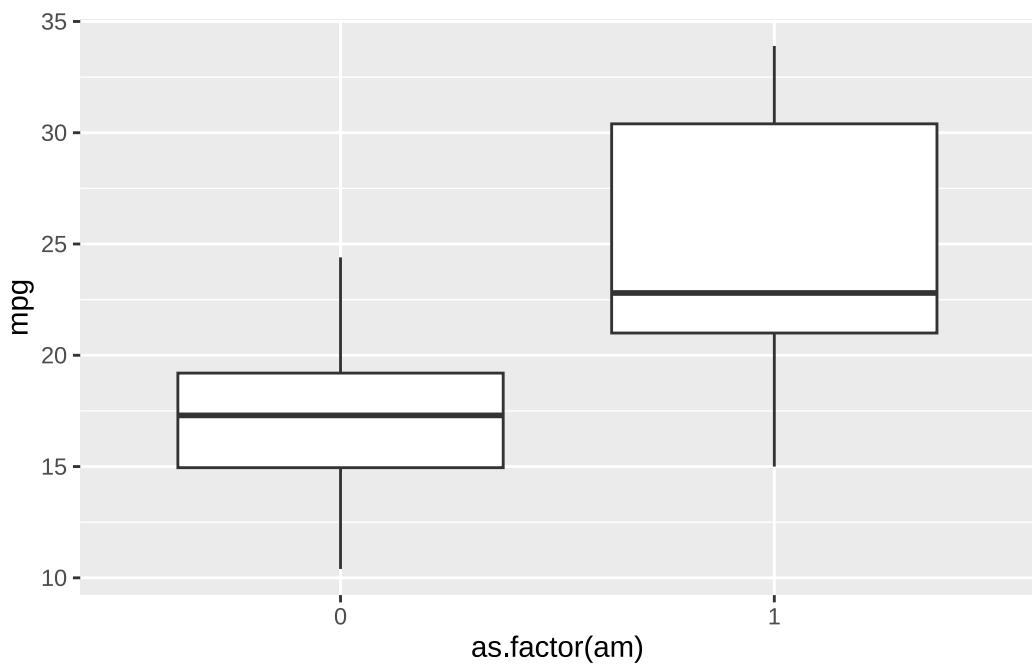
6.13 Indicating significances

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!).

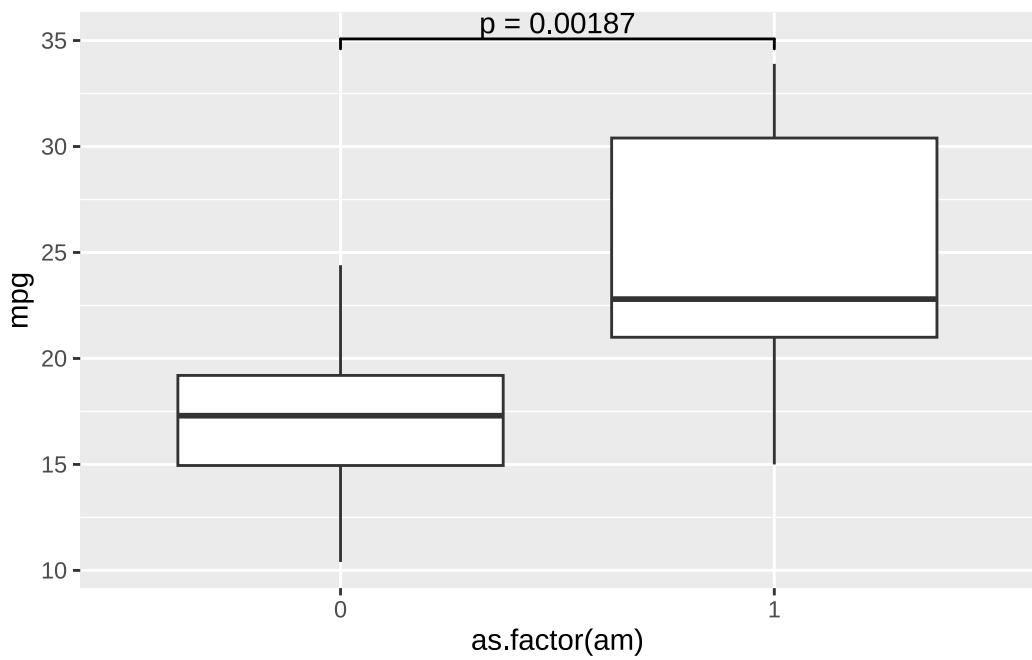
```
# ggsign ####
p <- round(wilcox.test(mtcars$mpg~mtcars$am)$p.value,5)
```

```
Warning in wilcox.test.default(x = DATA[[1L]], y = DATA[[2L]], ...): kann bei Bindungen keinen exakten p-Wert Berechnen
```

```
(plottemp <- ggplot(mtcars,aes(as.factor(am),mpg))+  
  geom_boxplot())
```

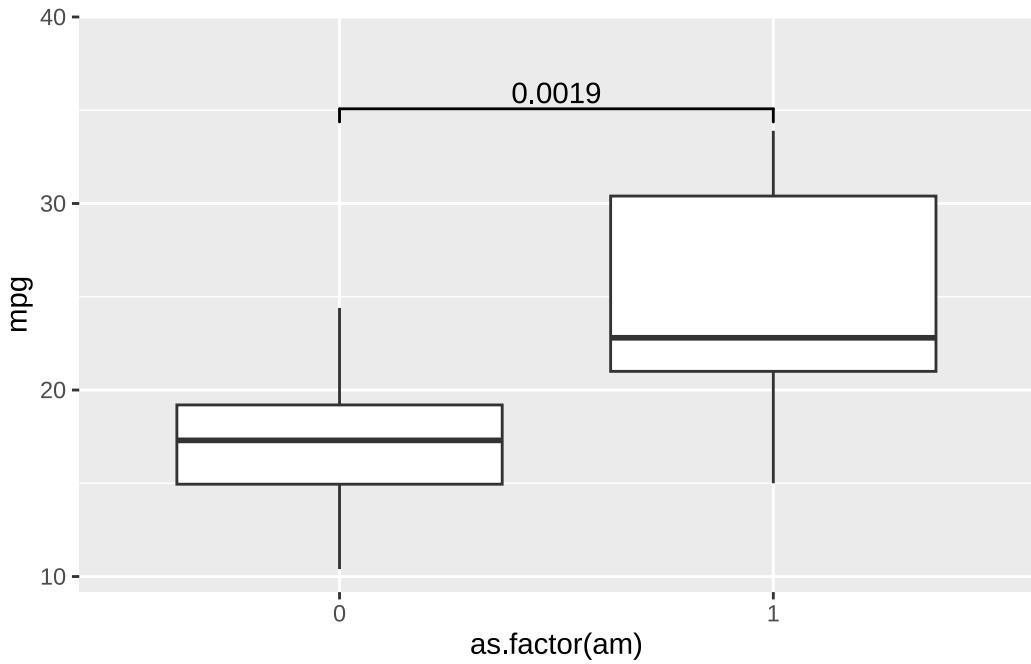


```
plottemp+geom_signif(  
  comparisons=list(c(1,2)),  
  # aes(y=0),  
  # textsize = rel(3), vjust = .0,  
  #y_position=max(mtcars$mpg)+3),  
  annotations=paste0("p = ", p),  
  # annotations=p,  
  tip_length=.02)
```



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

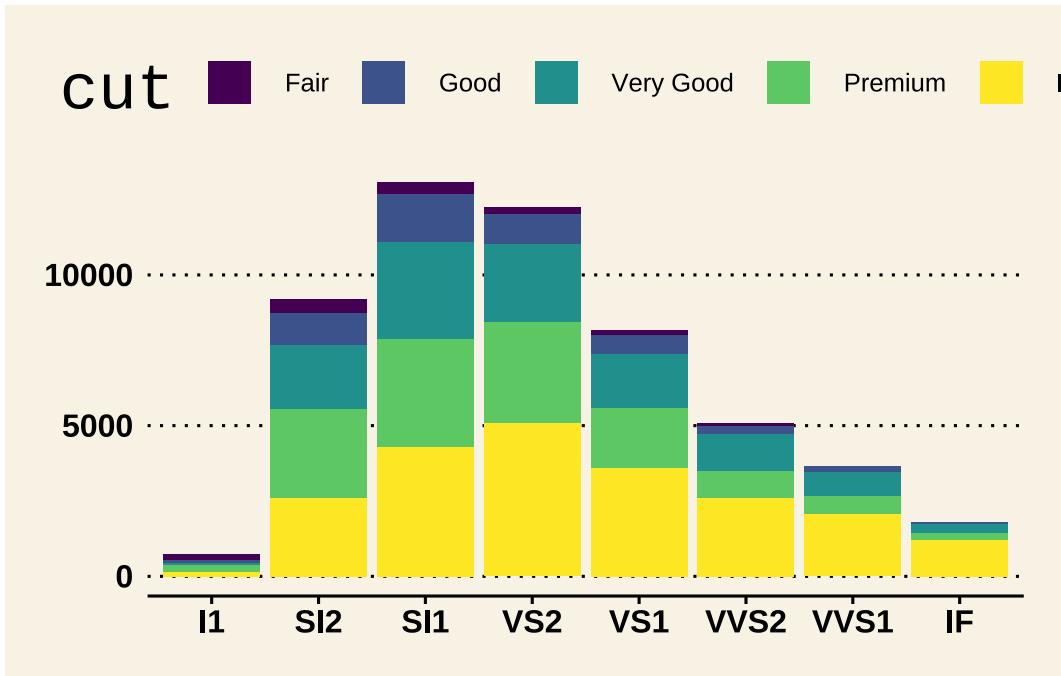
Warning in wilcox.test.default(c(21.4, 18.7, 18.1, 14.3, 24.4, 22.8, 19.2, :
kann bei Bindungen keinen exakten p-Wert Berechnen



6.14 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 are customizable. 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()
```

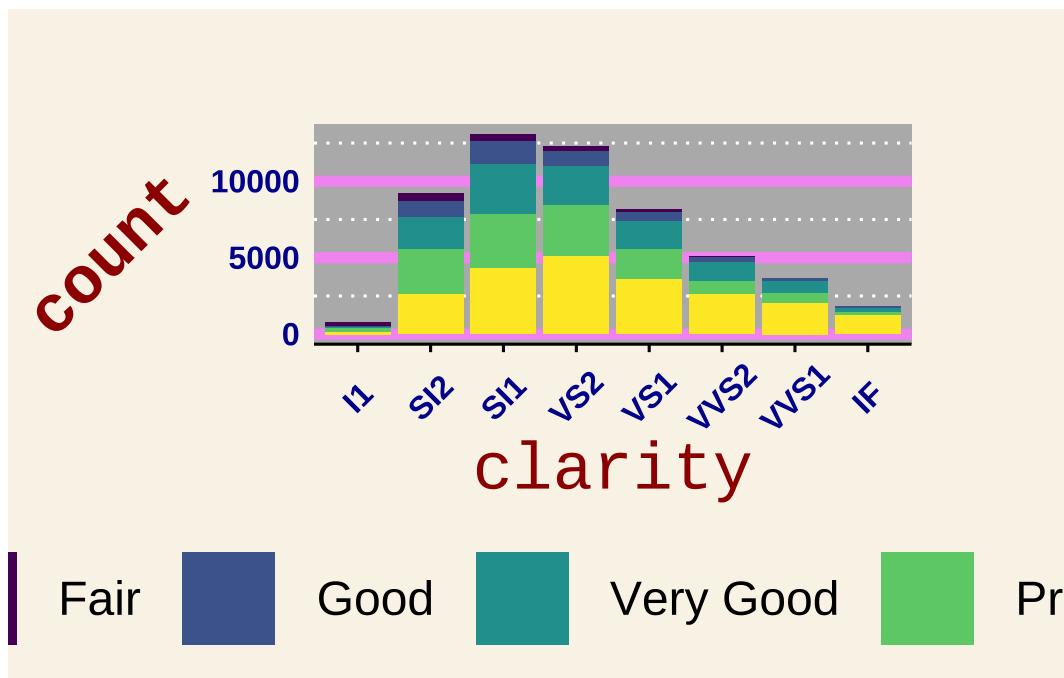


```

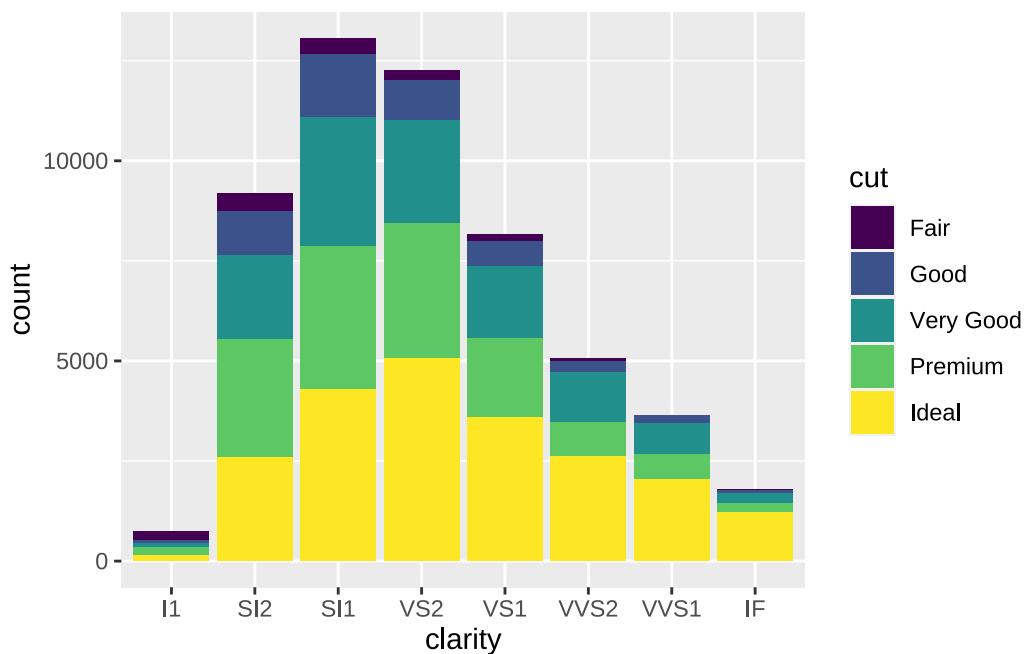
theme_update(legend.position="bottom",
            axis.text=element_text(colour = "darkblue",
                                   size=12),
            axis.text.x=element_text(vjust=0.5,angle=45,
                                    family="sans",
                                    face = "bold"),
            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,
                colour="violet", linewidth = 2),
            legend.text = element_text(size = 18),
            legend.title=element_text(size=30, colour="pink"))

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

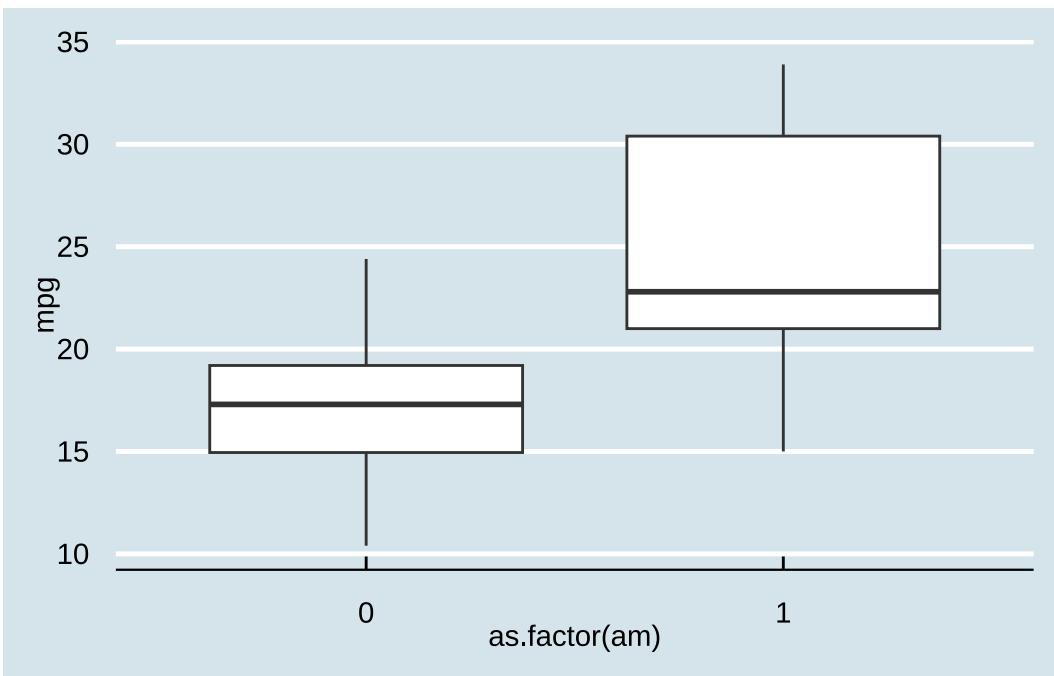
```



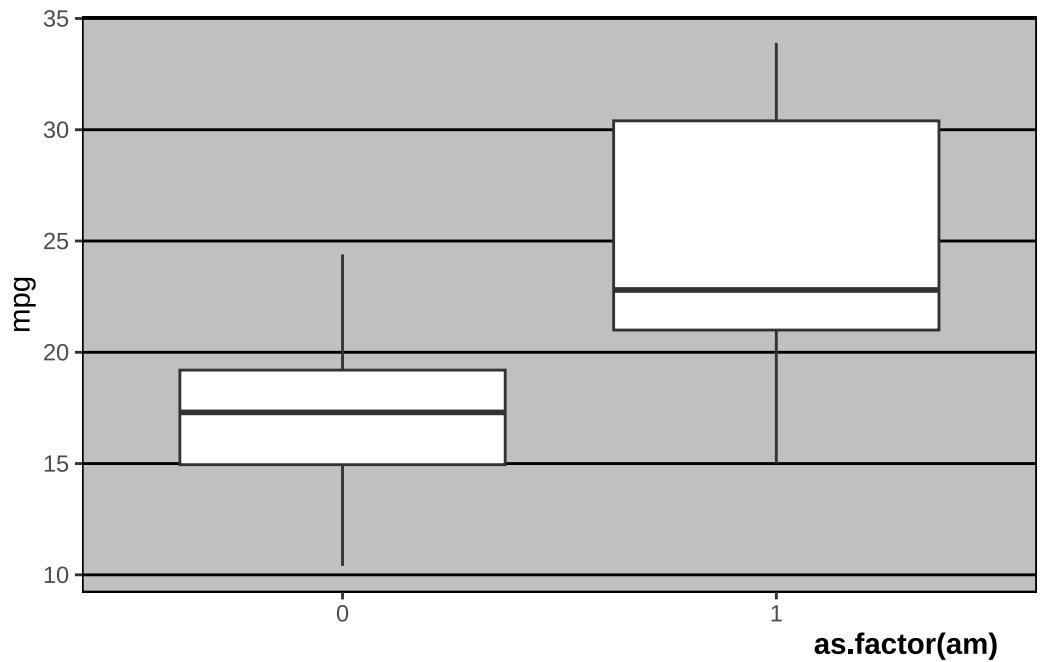
```
theme_set(theme_grey())
#theme_set(old)
ggplot(data=diamonds,aes(x=clarity,fill=cut))+  
  geom_bar()
```



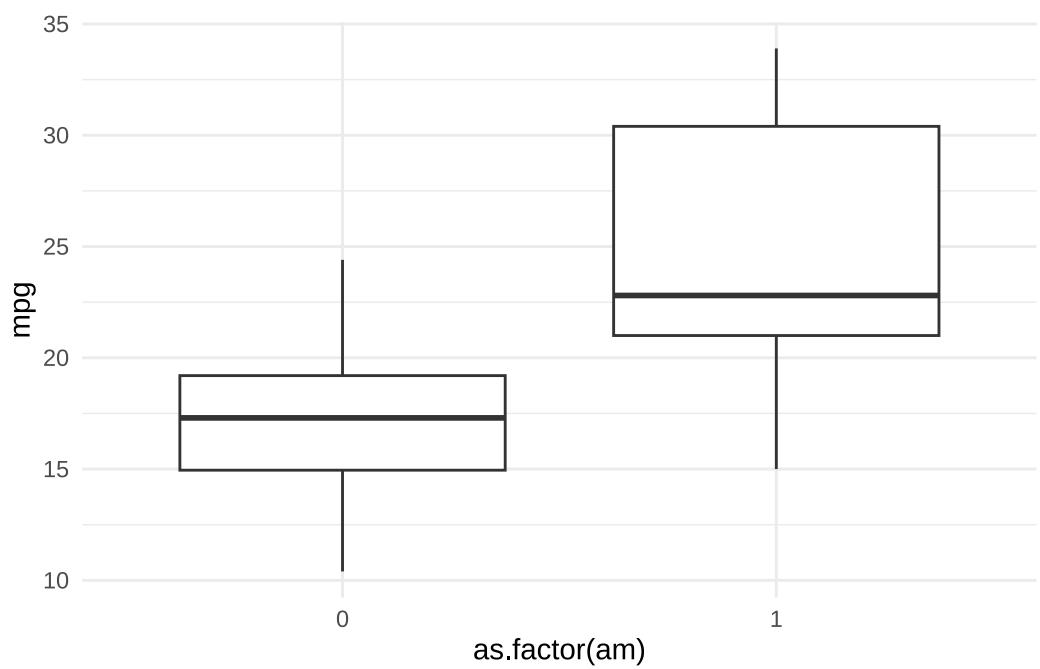
```
# ggthemes #####
plottemp+theme_economist()
```



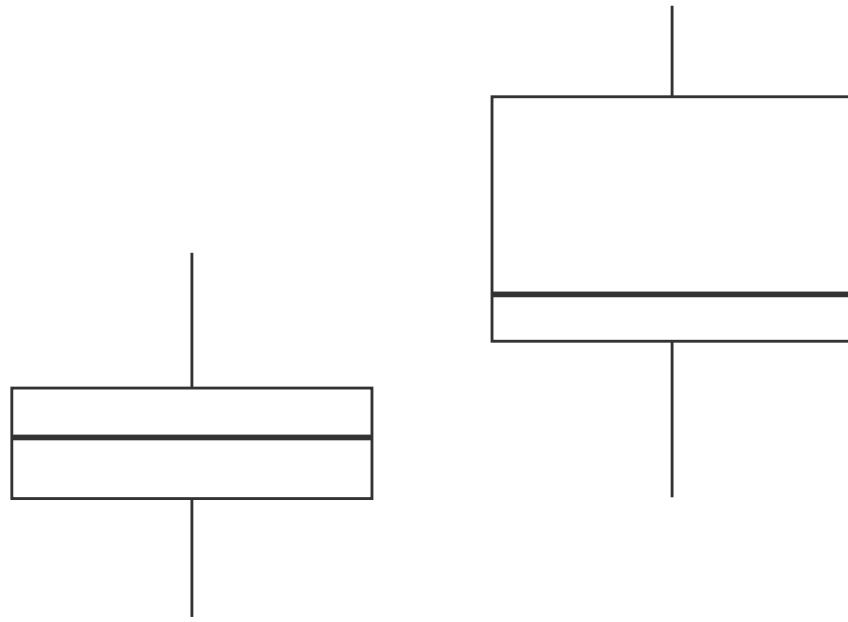
```
plottemp+theme_excel()+
  theme(axis.title.x = element_text(face="bold", hjust=0.95))
```



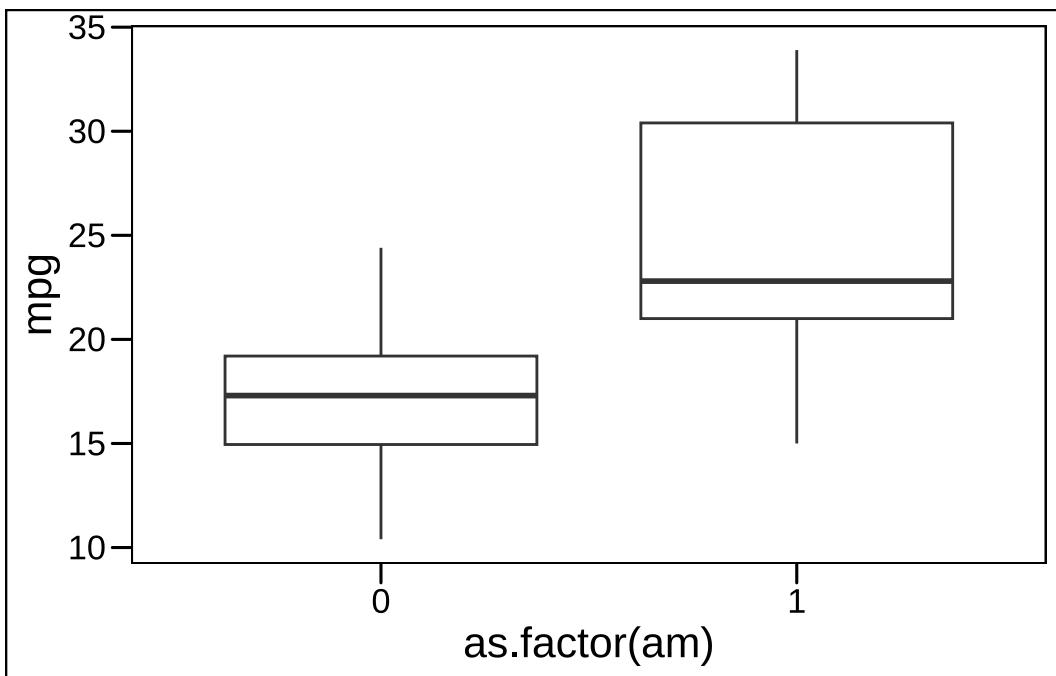
```
plottemp+theme_minimal()
```



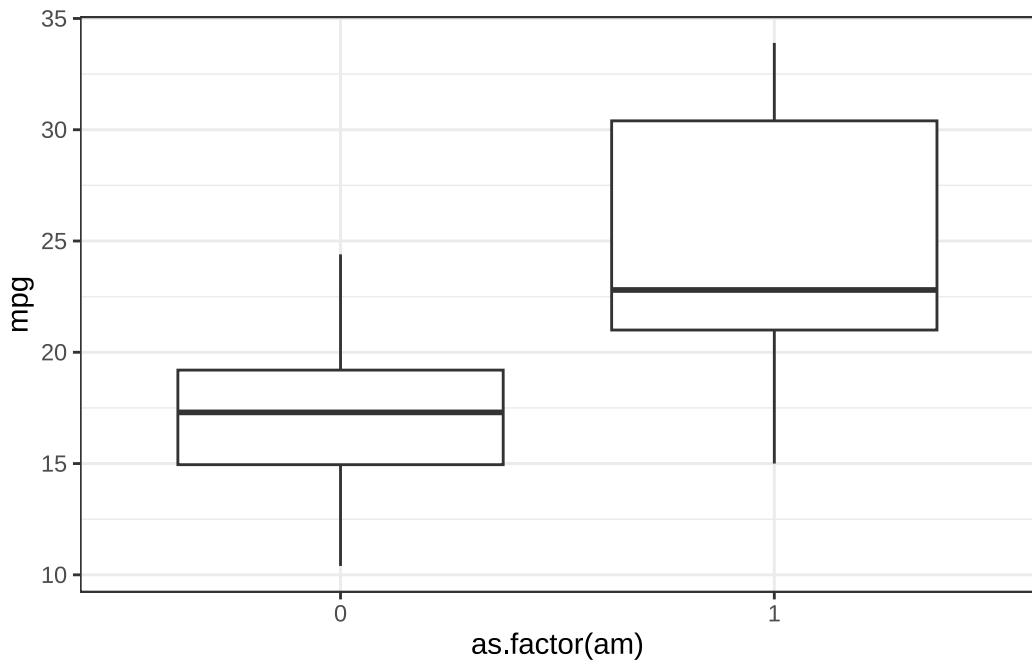
```
plottemp+theme_void()
```



```
plottemp+theme_base()
```



```
plottemp+theme_bw()
```



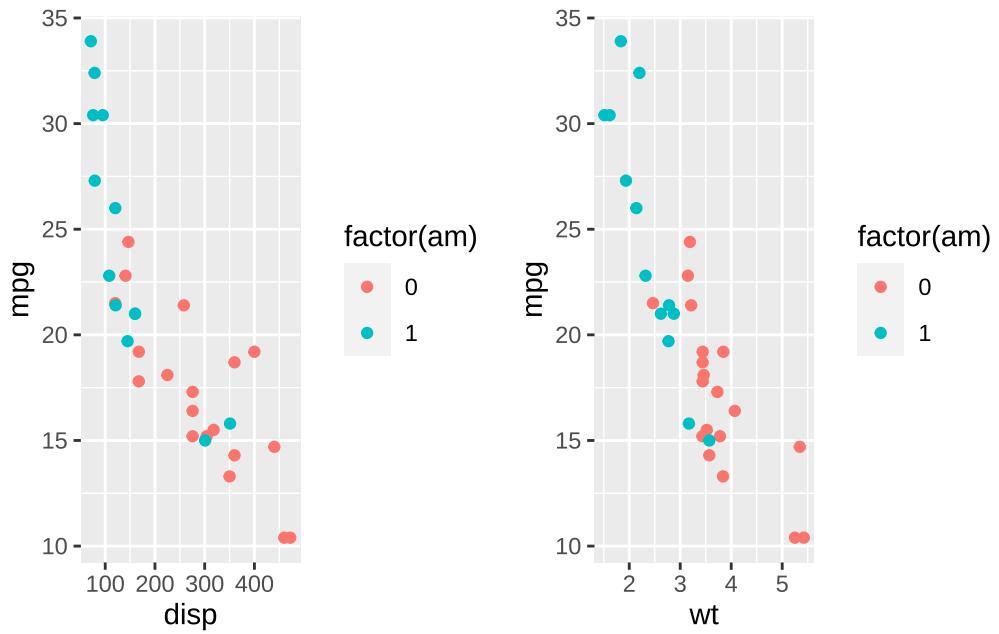
```
# https://github.com/thomasp85/patchwork

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

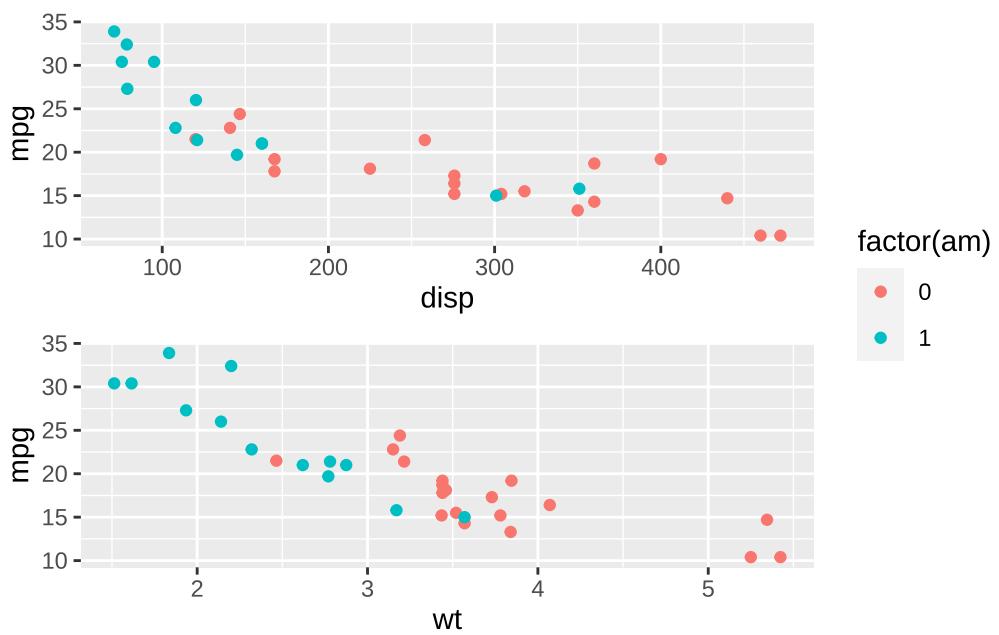
6.15 Combining figures with patchwork

More details on [github](#)

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



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



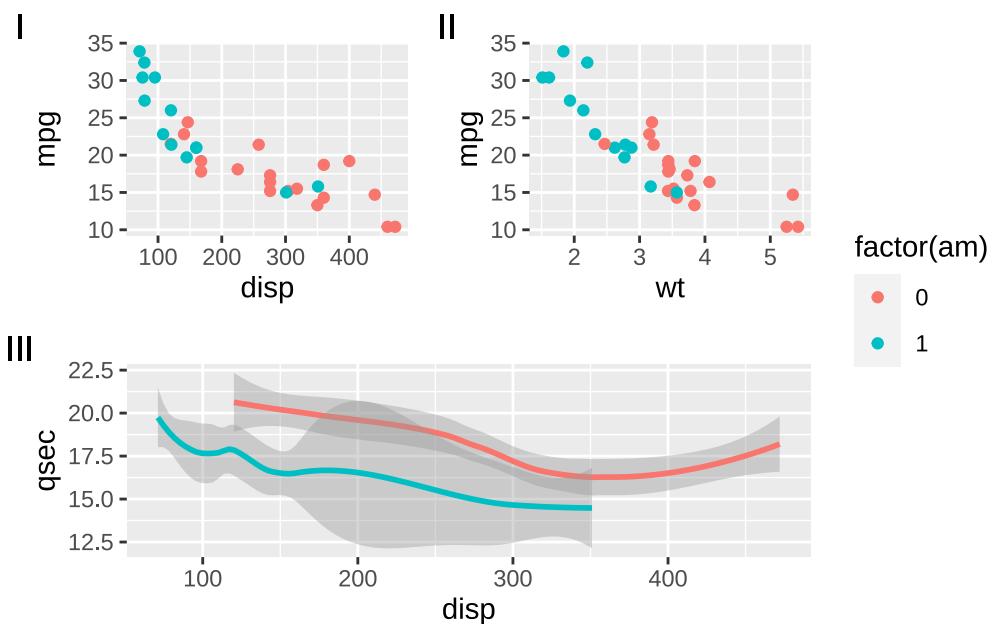
```

p3 <- ggplot(mtcars) +
  geom_smooth(aes(disp, qsec, color=factor(am)))
# p4 <- ggplot(mtcars) +
#   geom_bar(aes(factor(carb), fill=factor(am)))

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

```

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



7 Descriptive statistics

```
pacman::p_load(conflicted,tidyverse,wrappedtools,
                flextable)
#dir("Data/")
load("data/bookdata1.RData")

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

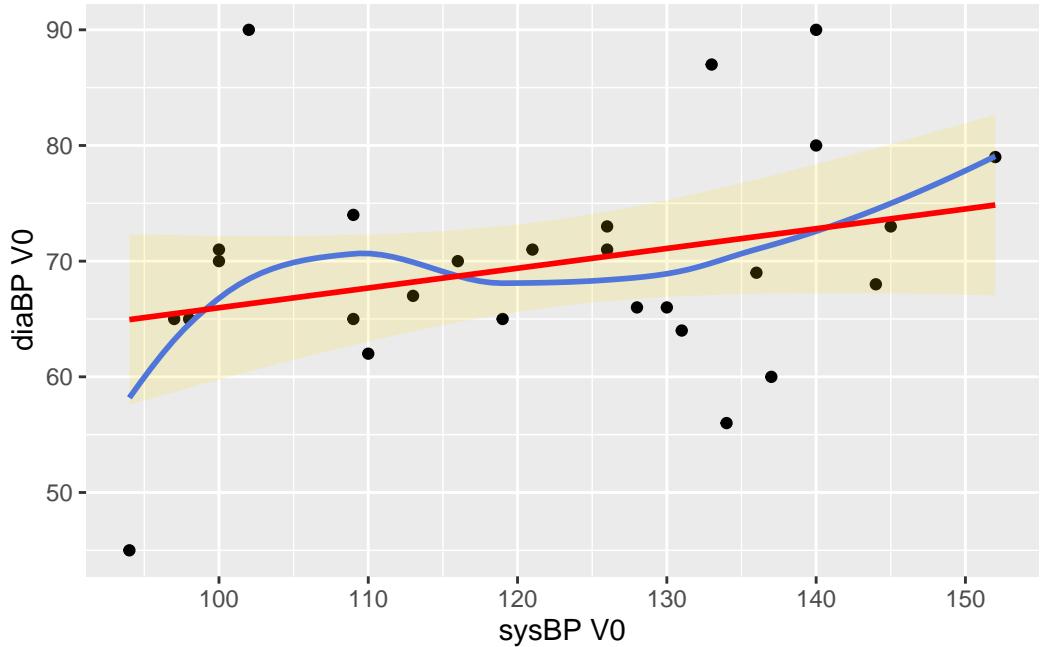
```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
Warning: Removed 1 rows containing non-finite values (`stat_smooth()`).
```

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

```
Warning: Removed 1 rows containing non-finite values (`stat_smooth()`).
```

```
Warning: Removed 1 rows containing missing values (`geom_point()`).
```



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

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

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

```
[1] 160
```

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

```
[1] 174.11
```

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

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

```
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]"
```

```
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)"
```

```
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%)
```

```
cat_desc_table(data = rawdata,  
               desc_vars = factvars$names) |>  
  rename(`n (%)` = desc_all) |>  
  flextable() |>  
  align(~`n (%)` != " ", j = 1, align = "right") |>  
  width(j = c(1,2), width = c(3,4), unit = "cm")
```

Variable	n (%)
Testmedication	
	014 (50%)
	114 (50%)
Sex	
	f4 (14.29%)
	m24 (85.71%)
NYHA V1	
	02 (11.76%)
	19 (52.94%)
	23 (17.65%)
	33 (17.65%)
NYHA V2	
	01 (9.09%)
	16 (54.55%)
	22 (18.18%)

Variable	n (%)
NYHA V3	32 (18.18%)
	16 (50%)
	23 (25%)
	33 (25%)

```
rawdata |>
  group_by(Sex,Testmedication) |>
  summarise(WeightSummary=meansd(`Weight (kg)`))
```

`summarise()` has grouped output by 'Sex'. You can override using the ` `.groups` argument.

```
# A tibble: 4 x 3
# Groups:   Sex [2]
  Sex   Testmedication WeightSummary
  <fct> <fct>           <chr>
1 f      0               86 ± 9
2 f      1               66 ± 3
3 m      0               91 ± 14
4 m      1               90 ± 14
```

```
compare2numvars(rawdata,
                 dep_vars = c( "Size (cm)", "Weight (kg)",
                               "sysBP V0", "diaBP V0"),
                 indep_var = "Sex",
                 gaussian = FALSE)
```

```
# A tibble: 4 x 5
  Variable desc_all    `Sex f`    `Sex m`     p
  <fct>    <chr>       <chr>       <chr>       <chr>
1 Size (cm) 174 (168/179) 166 (162/168) 174 (172/180) 0.011
2 Weight (kg) 84 (77/96) 74 (66/88) 86 (80/104) 0.107
3 sysBP V0 126 (109/136) 119 (105/129) 126 (109/137) 0.609
4 diaBP V0 69 (65/73) 66 (64/80) 70 (65/73) 0.784
```

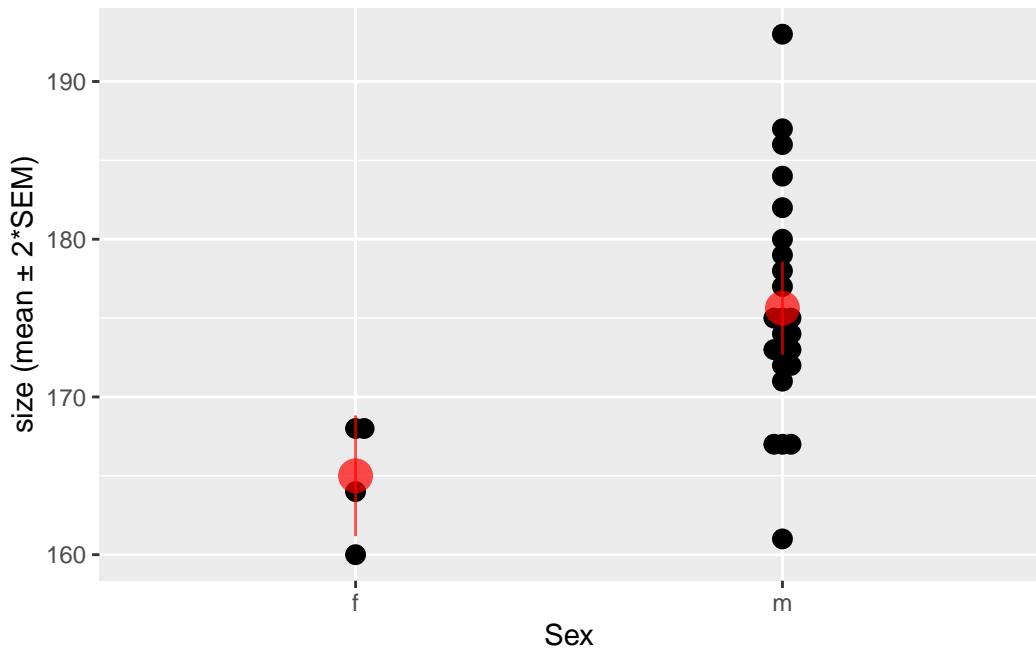
8 Better tables with `flextable`

9 Simple test statistics

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

9.1 Quantitative measures with Gaussian distribution

```
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))+
```



```
by(data = rawdata`Size (cm)`, INDICES = rawdata$Sex,  
   FUN = meanse)
```

```
rawdata$Sex: f  
[1] "165±2"
```

```
-----  
rawdata$Sex: m  
[1] "176±1"
```

```
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))
```

Welch Two Sample t-test

```
data: rawdata`Size (cm)` by rawdata$Sex  
t = -4.3967, df = 7.2767, p-value = 0.002887  
alternative hypothesis: true difference in means between group f and group m is not equal  
95 percent confidence interval:  
-16.295575 -4.954425  
sample estimates:  
mean in group f mean in group m  
165.000 175.625
```

```
tOut$p.value
```

```
[1] 0.00288704
```

```
# equal variances assumption?  
(vartestOut<-var.test(rawdata$`Size (cm)`~rawdata$Sex))
```

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
```

```
(t0Out<-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
```

```
(t0Out<-  
  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
```

```
t_var_test(data = rawdata, formula = ``Size (cm)`~Sex")
```

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=T) #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.ttest(rawdata$`sysBP V0`,
        rawdata$`sysBP V2`,
        # alternative="greater", # x>y
        paired=T)$p.value/2 #pairwise t-test, within subject
```

[1] 0.1935805

```
t.ttest(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)
```

	Variable	desc_all	`Sex f`	`Sex m`	p
	<fct>	<chr>	<chr>	<chr>	
1	Size (cm)	174 ± 8	165 ± 4	176 ± 7	0.009
2	Weight (kg)	88 ± 15	76 ± 13	90 ± 14	0.072
3	sysBP V0	122 ± 17	118 ± 14	123 ± 18	0.587
4	diaBP V0	70 ± 10	71 ± 13	69 ± 9	0.780
5	Lv Edv Mri	206 ± 70	235 ± 72	203 ± 71	0.559
6	Lv Esv Mri	110 ± 70	158 ± 56	105 ± 71	0.322
7	Lv Ef Mri	50 ± 15	33 ± 3	52 ± 15	0.095
8	Lv Ef Biplan Mri	49 ± 14	26 ± 11	51 ± 13	0.018
9	sysBP V2	119 ± 13	114 ± 18	120 ± 12	0.438
10	diaBP V2	66 ± 9	58 ± 12	67 ± 8	0.097
11	BMI	29 ± 4	28 ± 5	29 ± 4	0.585

```
compare2numvars(data = rawdata,dep_vars = gaussvars$names,
indep_var = "Testmedication",gaussian = T)
```

```
# A tibble: 11 x 5
  Variable      desc_all `Testmedication 0` `Testmedication 1` p
  <fct>        <chr>    <chr>            <chr>            <chr>
1 Size (cm)   174 ± 8  173 ± 6          175 ± 9          0.653
2 Weight (kg) 88 ± 15  90 ± 14          86 ± 16          0.448
3 sysBP V0    122 ± 17 122 ± 15         122 ± 19          0.966
4 diaBP V0    70 ± 10  70 ± 7           70 ± 12          0.943
5 Lv Edv Mri  206 ± 70 243 ± 77        173 ± 45          0.019
6 Lv Esv Mri  110 ± 70 138 ± 87        86 ± 41           0.108
7 Lv Ef Mri   50 ± 15  48 ± 18          52 ± 12           0.532
8 Lv Ef Biplan Mri 49 ± 14 47 ± 17        50 ± 13           0.747
9 sysBP V2    119 ± 13 122 ± 11         117 ± 14           0.297
10 diaBP V2   66 ± 9   66 ± 8            66 ± 9            0.966
11 BMI        29 ± 4   30 ± 5           28 ± 3            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)
}
```

```
# A tibble: 11 x 5
  Variable      desc_all `Testmedication 0` `Testmedication 1` p
  <fct>        <chr>    <chr>            <chr>            <chr>
1 Size (cm)   174 ± 8  173 ± 6          175 ± 9          0.653
2 Weight (kg) 88 ± 15  90 ± 14          86 ± 16          0.448
3 sysBP V0    122 ± 17 122 ± 15         122 ± 19          0.966
4 diaBP V0    70 ± 10  70 ± 7           70 ± 12          0.943
5 Lv Edv Mri  206 ± 70 243 ± 77        173 ± 45          0.019
6 Lv Esv Mri  110 ± 70 138 ± 87        86 ± 41           0.108
7 Lv Ef Mri   50 ± 15  48 ± 18          52 ± 12           0.532
8 Lv Ef Biplan Mri 49 ± 14 47 ± 17        50 ± 13           0.747
9 sysBP V2    119 ± 13 122 ± 11         117 ± 14           0.297
10 diaBP V2   66 ± 9   66 ± 8            66 ± 9            0.966
11 BMI        29 ± 4   30 ± 5           28 ± 3            0.180
# A tibble: 11 x 5
  Variable      desc_all `Sex f`   `Sex m`   p
  <fct>        <chr>    <chr>    <chr>    <chr>
1 Size (cm)   174 ± 8  165 ± 4  176 ± 7  0.009
2 Weight (kg) 88 ± 15  76 ± 13  90 ± 14  0.072
```

3 sysBP V0	122	± 17	118	± 14	123	± 18	0.587
4 diaBP V0	70	± 10	71	± 13	69	± 9	0.780
5 Lv Edv Mri	206	± 70	235	± 72	203	± 71	0.559
6 Lv Esv Mri	110	± 70	158	± 56	105	± 71	0.322
7 Lv Ef Mri	50	± 15	33	± 3	52	± 15	0.095
8 Lv Ef Biplan Mri	49	± 14	26	± 11	51	± 13	0.018
9 sysBP V2	119	± 13	114	± 18	120	± 12	0.438
10 diaBP V2	66	± 9	58	± 12	67	± 8	0.097
11 BMI	29	± 4	28	± 5	29	± 4	0.585

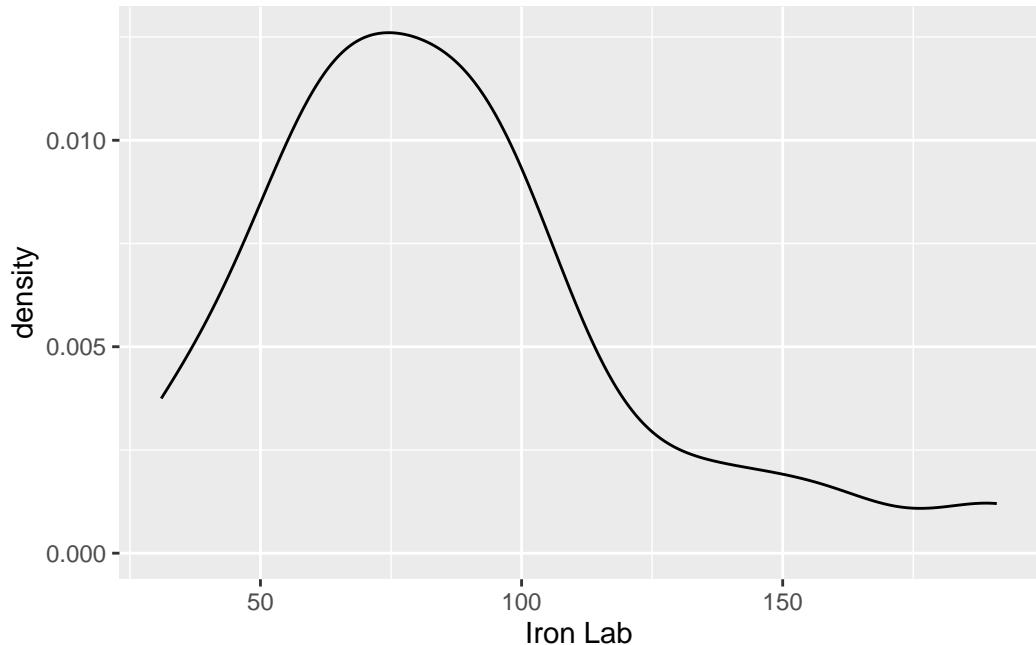
9.2 Ordinal data

```
ordvars$names
```

```
[1] "Ptt Lab"           "Ferritin Lab"      "Iron Lab"          "Transferrin Lab"
[5] "Age"
```

```
ggplot(rawdata,aes(`Iron Lab`))+  
  geom_density()
```

Warning: Removed 1 rows containing non-finite values (`stat_density()`).

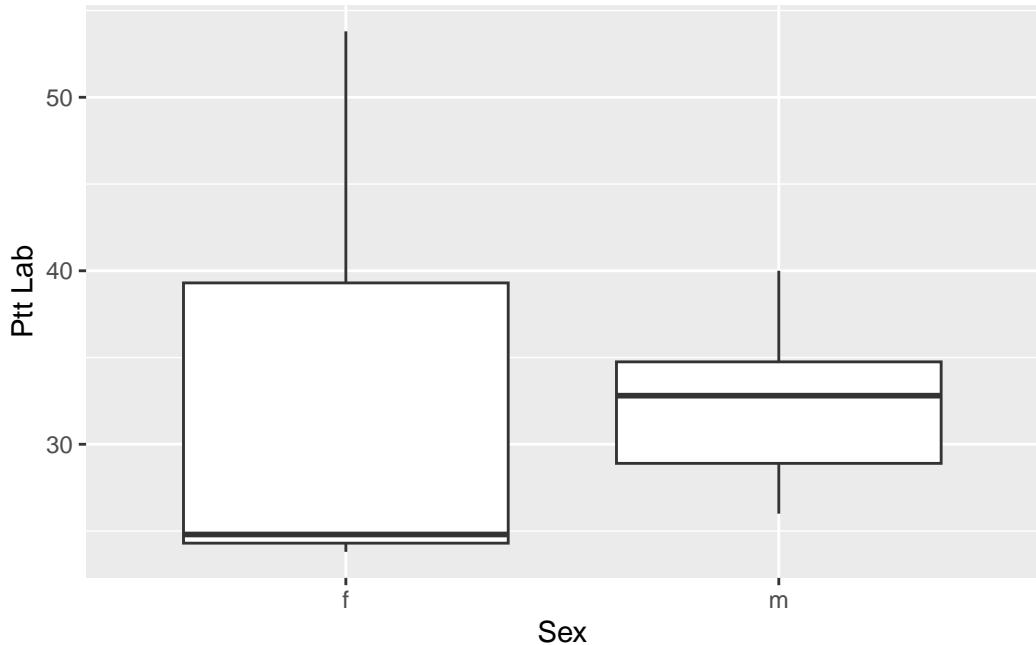


```
by(data = rawdata[[ordvars$index[1]]],  
  INDICES = rawdata$Sex, FUN = median_quart)
```

```
rawdata$Sex: f  
[1] "25 (24/49)"  
-----  
rawdata$Sex: m  
[1] "33 (29/35)"
```

```
ggplot(rawdata,aes(Sex,`Ptt Lab`))+  
  geom_boxplot()
```

Warning: Removed 1 rows containing non-finite values (`stat_boxplot()`).



```
(u0Out<-wilcox.test(  
  rawdata[[ordvars$index[1]]] ~ rawdata$Sex, exact=F))
```

Wilcoxon rank sum test with continuity correction

```
data: rawdata[[ordvars$index[1]]] by rawdata$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`~as.factor(Sex),  
data=rawdata))
```

```
Asymptotic Wilcoxon-Mann-Whitney Test
```

```
data: Ptt Lab by as.factor(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,
                 indep_var = "Sex",gaussian = F)
```

	# A tibble: 5 x 5	Variable	desc_all	`Sex f`	`Sex m`	p
		<fct>	<chr>	<chr>	<chr>	<chr>
1	Ptt Lab	33 (28/35)	25 (24/49)	33 (29/35)	0.375	
2	Ferritin Lab	222 (162/339)	138 (90/591)	224 (172/316)	0.616	
3	Iron Lab	80 (61/102)	95 (92/103)	75 (60/99)	0.165	
4	Transferrin Lab	261 (233/276)	260 (227/299)	262 (235/276)	1.000	
5	Age	62 (53/67)	66 (58/69)	60 (53/66)	0.250	

9.3 Categorial data

```
factvars$names
```

```
[1] "Testmedication" "Sex"           "NYHA V1"          "NYHA V2"
[5] "NYHA V3"
```

```
(crosstab<-table(rawdata$Sex,rawdata$Testmedication))
```

	0	1
f	2	2
m	12	12

```
chisq.test(crosstab,simulate.p.value=T,B=10^5) #empirical p-value
```

```
Pearson's Chi-squared test with simulated p-value (based on 1e+05  
replicates)
```

```
data: crosstab  
X-squared = 0, df = NA, p-value = 1
```

```
chisq.test(table(rawdata$Sex, rawdata$`NYHA V1`)) #based on table
```

```
Warning in chisq.test(table(rawdata$Sex, rawdata$`NYHA V1`)):  
Chi-Quadrat-Approximation kann inkorrekt sein
```

```
Pearson's Chi-squared test
```

```
data: table(rawdata$Sex, rawdata$`NYHA V1`)  
X-squared = 4.3849, df = 3, p-value = 0.2228
```

```
chisq.test(x=rawdata$Sex, y=rawdata$`NYHA V1`,  
simulate.p.value=T, B=10^5) #based on rawdata
```

```
Pearson's Chi-squared test with simulated p-value (based on 1e+05  
replicates)
```

```
data: rawdata$Sex and rawdata$`NYHA V1`  
X-squared = 4.3849, df = NA, p-value = 0.1741
```

```
(crosstab1<-table(rawdata$Sex,  
rawdata$`Weight (kg)`<=  
median(rawdata$`Weight (kg)`)))
```

	FALSE	TRUE
f	1	3
m	13	11

```
(tablettestOut<-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 = 0.5977
```

```
tabletestOut$p.value
```

```
[1] 0.597734
```

```
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 <- "Sex"

compare2qualvars(rawdata,dep_vars = factvars$names[-2],
                 indep_var = groupvar,spacer = " ")
```

	Variable	desc_all	`Sex f`	`Sex m`	p
	<chr>	<chr>	<chr>	<chr>	<chr>
1	"Testmedication"	" "	" "	" "	"1.000"
2	"0"	"14 (50%)"	"2 (50%)"	"12 (50%)"	" "
3	"1"	"14 (50%)"	"2 (50%)"	"12 (50%)"	" "
4	"NYHA V1"	" "	" "	" "	"0.095"
5	"0"	"2 (11.76%)"	"1 (33.33%)"	"1 (7.14%)"	" "
6	"1"	"9 (52.94%)"	"0 (0%)"	"9 (64.29%)"	" "
7	"2"	"3 (17.65%)"	"1 (33.33%)"	"2 (14.29%)"	" "
8	"3"	"3 (17.65%)"	"1 (33.33%)"	"2 (14.29%)"	" "
9	"NYHA V2"	" "	" "	" "	"1.000"
10	"0"	"1 (9.09%)"	"0 (0%)"	"1 (12.5%)"	" "
11	"1"	"6 (54.55%)"	"2 (66.67%)"	"4 (50%)"	" "
12	"2"	"2 (18.18%)"	"1 (33.33%)"	"1 (12.5%)"	" "
13	"3"	"2 (18.18%)"	"0 (0%)"	"2 (25%)"	" "
14	"NYHA V3"	" "	" "	" "	"0.091"
15	"1"	"6 (50%)"	"0 (0%)"	"6 (66.67%)"	" "
16	"2"	"3 (25%)"	"1 (33.33%)"	"2 (22.22%)"	" "
17	"3"	"3 (25%)"	"2 (66.67%)"	"1 (11.11%)"	" "

10 Intro to lm

In this chapter, linear models (including linear regression and ANOVA) will be introduced. Output is not optimized for print, but rather for interactive use.

10.1 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()
```

10.2 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"))

```

re-encoding from CP1252

10.3 Graphical exploration

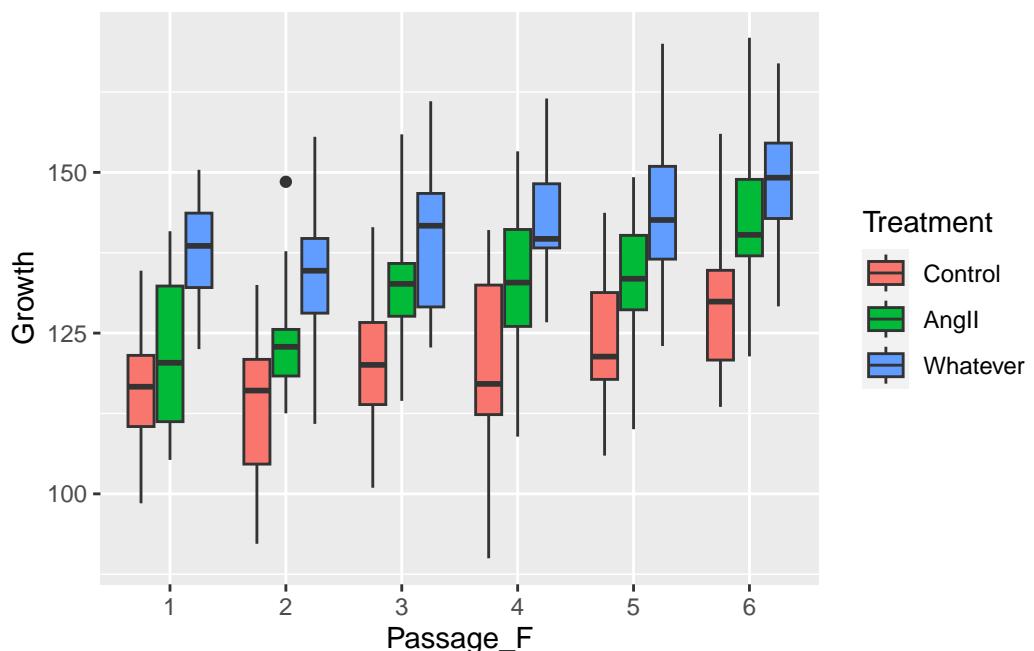
First impression of the data will be attempted by grouped boxplot, followed by interaction plots, both as basic and ggplot with variations.

```

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

  geom_boxplot(coef=3)

```



```

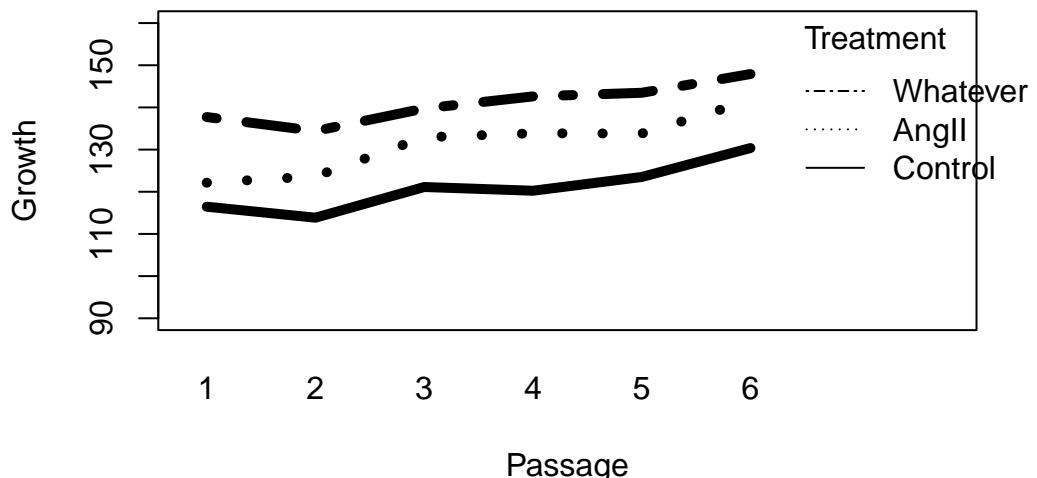
with(rawdata, interaction.plot(  

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

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

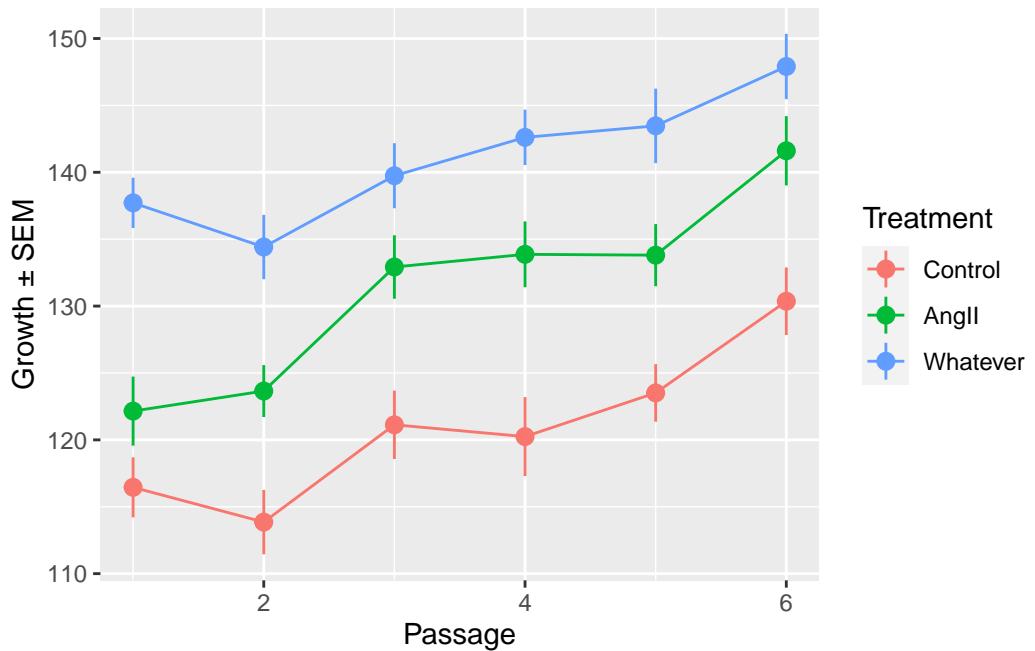
```

```
ylab = "Growth", xlab = "Passage",
trace.label = "Treatment"))
```



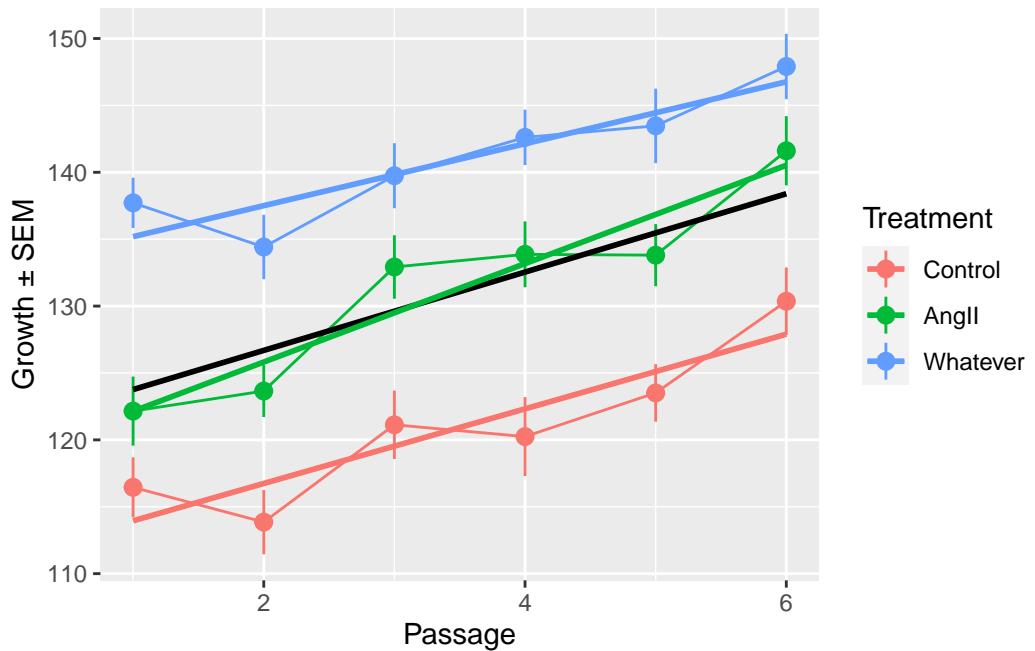
```
# p1<-ggplot(rawdata,aes(x=Passage,y=Growth))+
#   stat_summary(geom='line',fun='mean',aes(color=Treatment))+  
#   stat_summary(geom='line',fun='mean')  
p1<-ggplot(rawdata,aes(x=Passage,y=Growth))+  
  stat_summary(geom='line',fun='mean',aes(color=Treatment))+  
  stat_summary(aes(color=Treatment))+  
  ylab('Growth \u00b1 SEM')  
p1
```

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



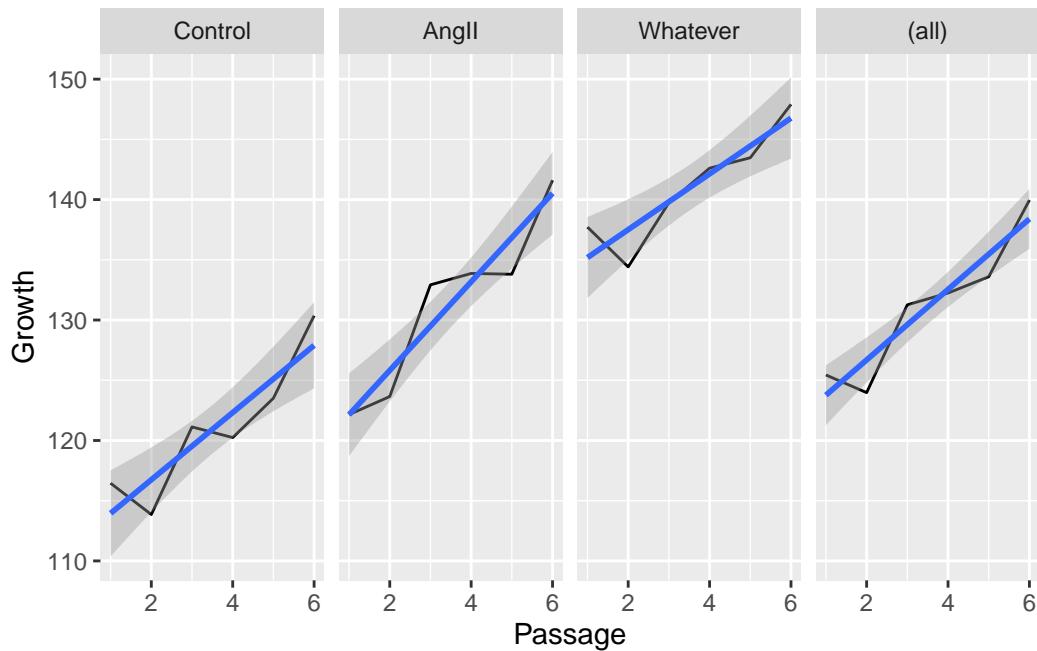
```
p1+geom_smooth(method='lm',color='black',se=F)+  
  geom_smooth(method='lm',aes(color=Treatment),se=F)
```

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



10.4 Linear Models

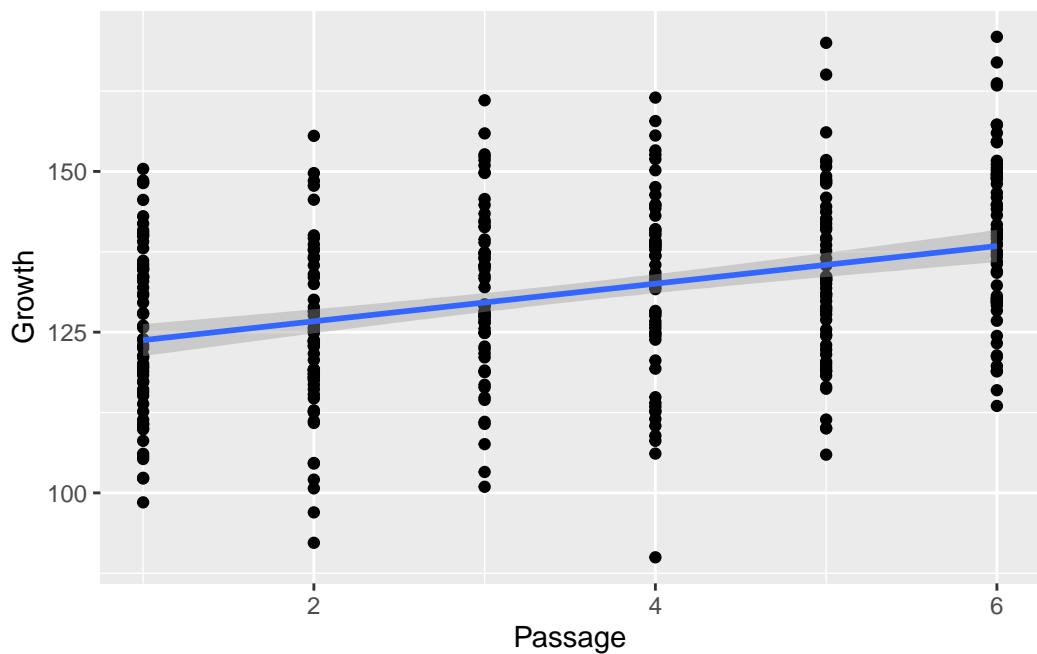
10.4.1 Linear regression

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

10.4.1.1 Graphical exploration

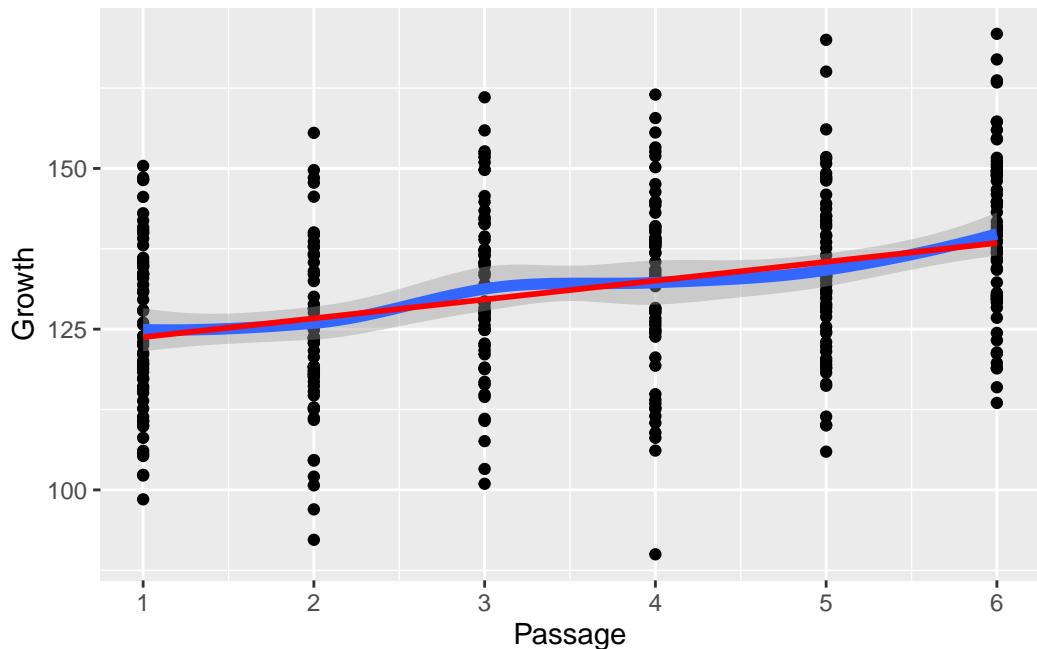
```
ggplot(rawdata,aes(Passage,Growth))+  
  geom_point() +  
  geom_smooth(method=lm)
```

```
`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'
```



10.4.1.2 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
```

```
# 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)
```

10.4.1.3 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))

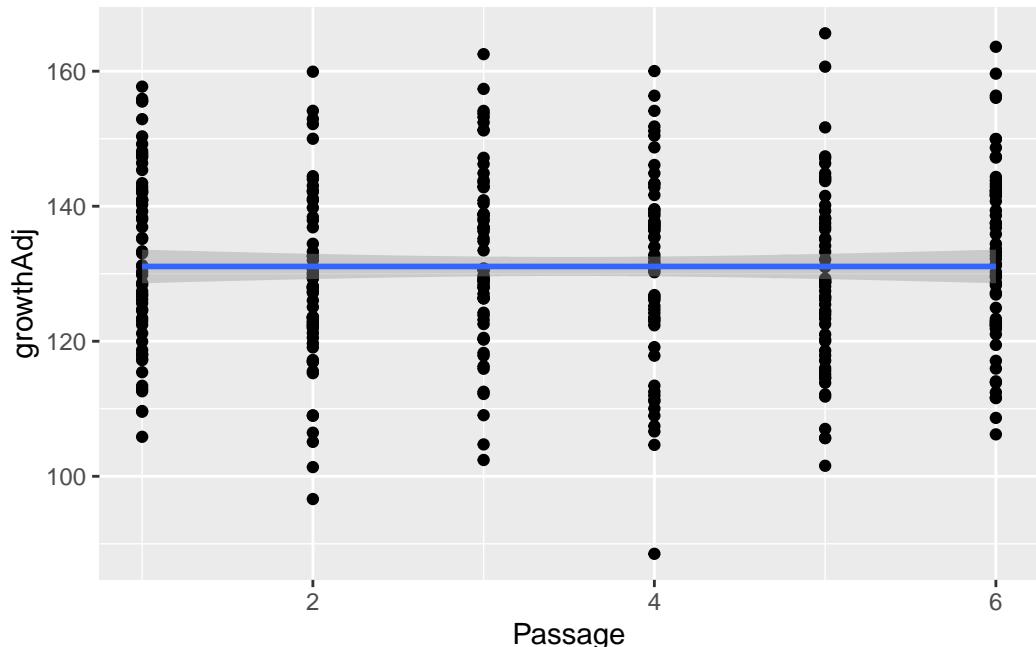
  summarise(rawdata,
            across(contains('growth'),
```

```
~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()
```

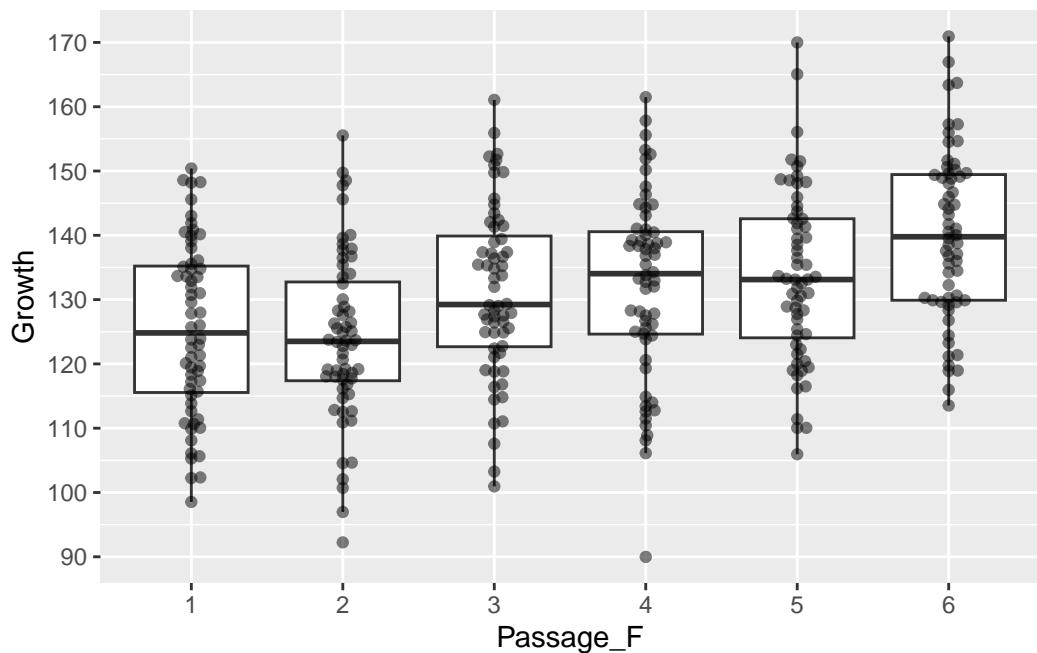
```
# A tibble: 2 x 5
  term      estimate std.error statistic   p.value
  <chr>        <dbl>     <dbl>      <dbl>     <dbl>
1 (Intercept) 1.31e+ 2      1.63    8.05e+ 1 2.04e-231
2 Passage      6.80e-15     0.418   1.63e-14 1.00e+ 0
```

10.4.2 ANOVA

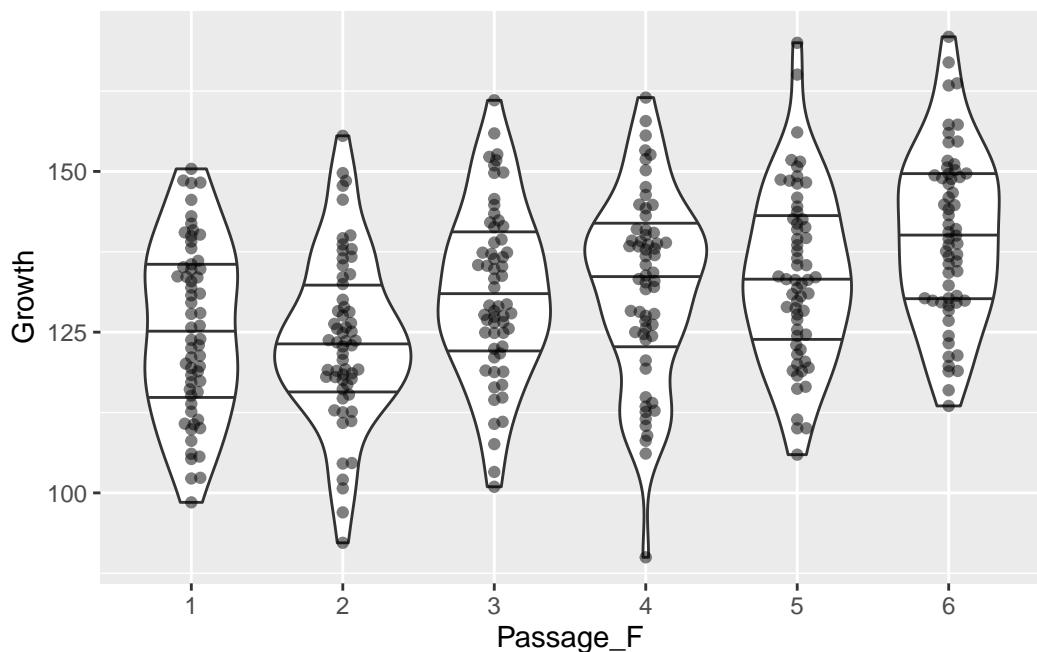
In the linear regression, we had Passage as a continuous IV, estimating a global ‘universal’ effect supposed to be constant. Now we look at Passage_F and model a discrete IV, allowing for specific effects, and thereby comparing means between groups.

10.4.2.1 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)
```



10.4.2.2 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
```

10.4.2.3 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$Growth[which(rawdata$Passage == 2)]
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,
                 p.adjust.method='fdr')
```

Pairwise comparisons using t tests with pooled SD

```
data: rawdata$Growth and rawdata$Passage
```

	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,
                 p.adjust.method='bonferroni')
```

Pairwise comparisons using t tests with pooled SD

data: rawdata\$Growth and rawdata\$Passage

	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'))
```

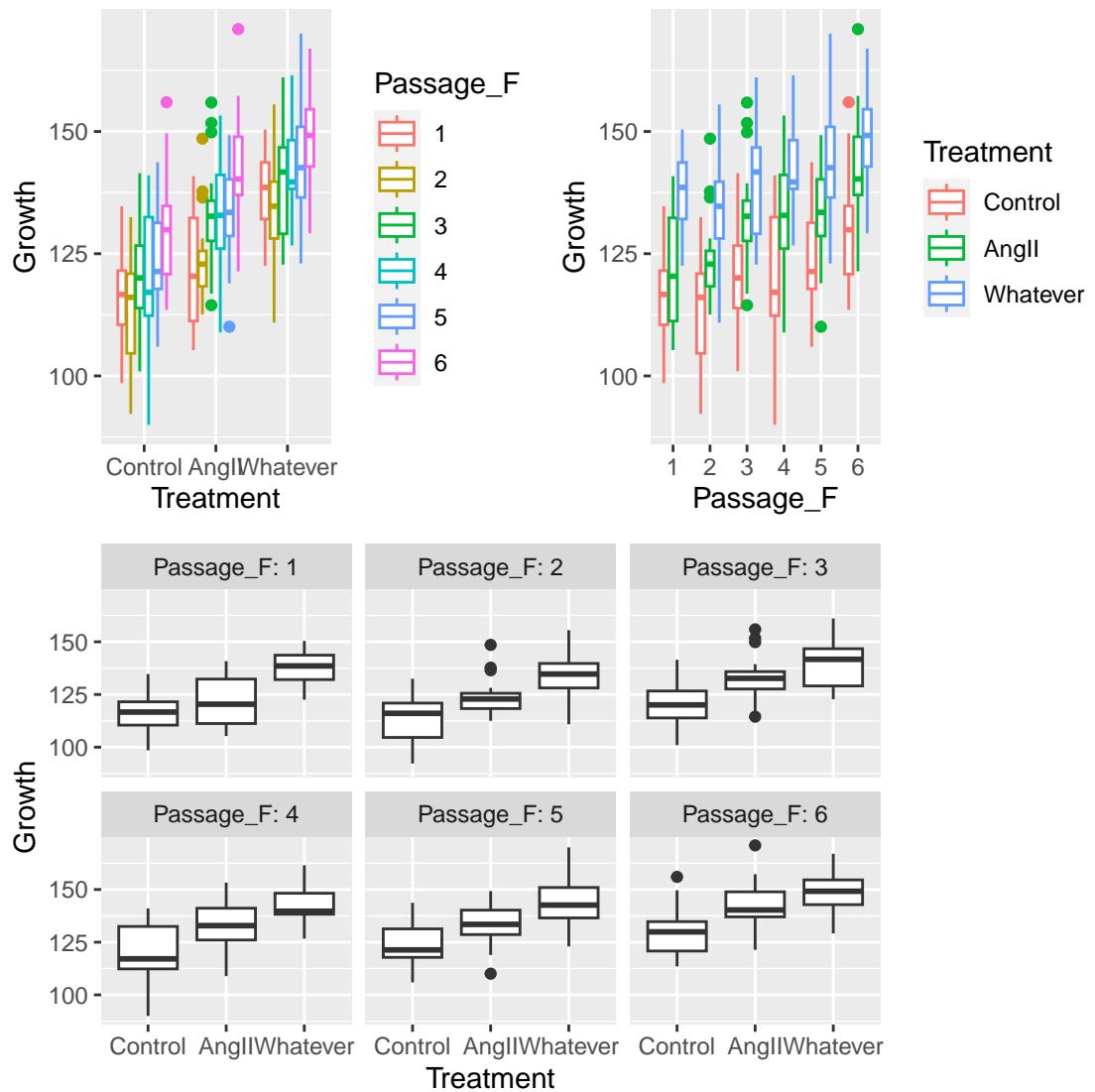
```
[1] "0.552" "0.023" "0.010" "0.003" "0.001"
```

10.4.3 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 $IV1*IV2$.

10.4.3.1 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
```



10.4.3.2 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(lmOut_additive,type=2)

```

Anova Table (Type II tests)

```

Response: Growth
  Sum Sq Df F value    Pr(>F)
Passage    8996    1 77.058 < 2.2e-16 ***
Treatment  24137    2 103.372 < 2.2e-16 ***
Residuals  41562  356
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

```

# 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
```

10.4.3.3 Post-hoc analyses

For multivariable models, pairwise.t.test() is not appropriate, Dunnet or Tukey tests (depending on hypothesis) are typical solutions.

```
summary(glht(model=lmOut_additive,linfct=mcp(Treatment='Dunnett')))
```

Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Dunnett 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 ***

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

```
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 0.0048 **
6-1 14.520106 8.2975011 20.742712 6.9e-08 ***

---
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'))]
```

```
[1] 116.4531
```

```
aOut$'Pr(>F)'
```

```
[1] 2.909117e-279 5.549803e-31 NA
```

```
aOut$`Sum Sq`
```

```
[1] 1754742.53 24136.66 50557.95
```

```
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    "     NA"
```

```
broom::tidy(aOut)
```

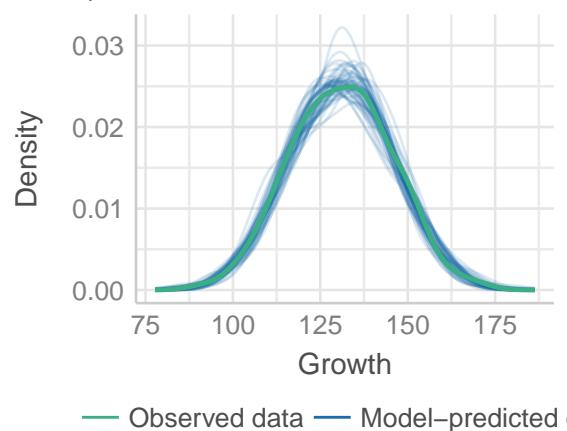
```
# A tibble: 3 x 5
  term        sumsq    df statistic    p.value
  <chr>      <dbl> <dbl>    <dbl>      <dbl>
1 (Intercept) 1754743.     1    12391.  2.91e-279
2 Treatment    24137.      2      85.2  5.55e- 31
3 Residuals    50558.     357      NA      NA
```

10.4.4 Model exploration with package performance

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

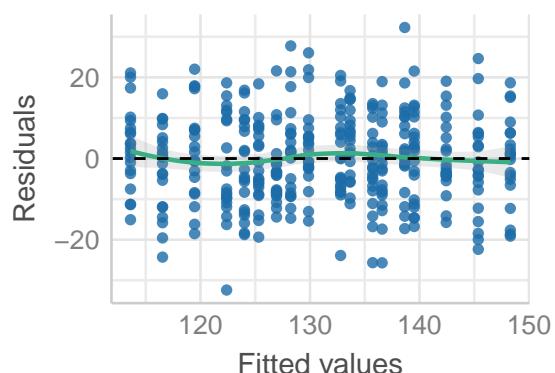
Posterior Predictive Check

Model-predicted lines should resemble observed data



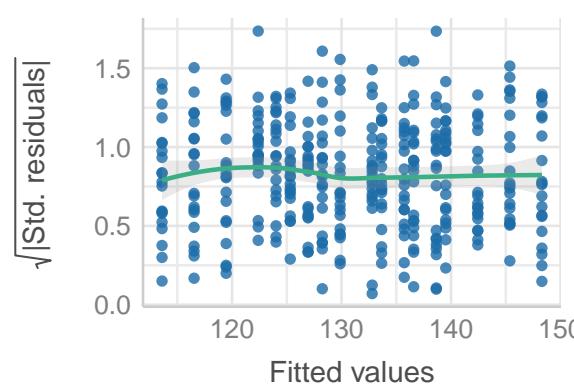
Linearity

Reference line should be flat and horizontal



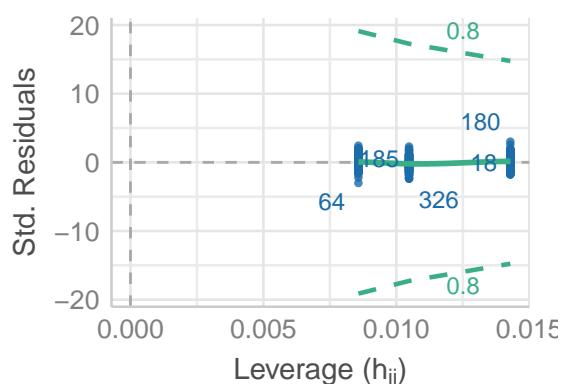
Homogeneity of Variance

Reference line should be flat and horizontal



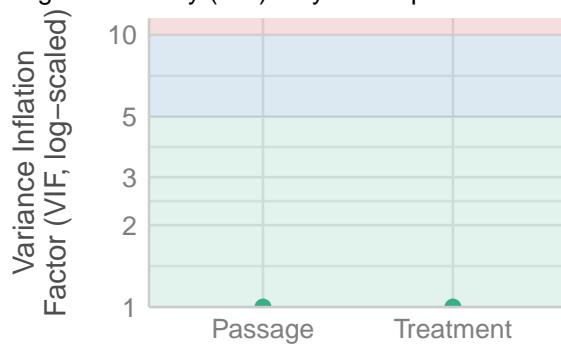
Influential Observations

Points should be inside the contour lines



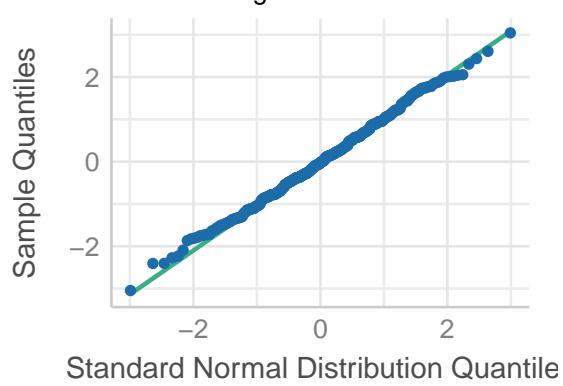
Collinearity

High collinearity (VIF) may inflate parameter uncertainty



Normality of Residuals

Dots should fall along the line



● Low (< 5)

11 Summary

In summary, this book has no content whatsoever.

1 + 1

[1] 2

References