RStatsbook

Andreas Busjahn

2024-07-11

Table of contents

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

The big book of R

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.*

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

```
2 conflicts

* `filter()`: dplyr and stats

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

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

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

```
[conflicted] Will prefer stats::lag 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"
```

```
str(float_num)
```

```
 num 123
```

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

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

```
typeof(int_num)
```

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

```
str(int_num)
```

```
 int 123
```

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

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

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

```
[1] 3
```

```
print(result)
```

```
[1] 3
```

```
cat(result)
```

```
3
```

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

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

```
typeof(char_var)
```

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

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

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

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

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

```
typeof(logical_var)
```

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

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

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

```r
# logicals usualy are defined by conditions:
int_num < float_num
```

```
[1] TRUE
```

```r
# 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. Package forcats provides nice tools for factors!

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

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

```r
class(factor_var)
```

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

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

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

```r
levels(factor_var)
```

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

```r
# 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:

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

```
[1] "2024-08-01"
```

```r
class(date_var)
```

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

```r
typeof(date_var)
```

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

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

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

Mixed classes:

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

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

```r
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 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 NA
```

```
  class((test_NA2))
```

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

```
  class(NA_integer_)
```

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

```
  class(NA_character_)
```

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

```
  class(NA_Date_) # from package lubridate
```

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

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
```

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

```
[1] 2
```

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"
```

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

```
[1] NA
```

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

```
[1] 5
```

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

```
[1] 5
```

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

```
[1] 5
```

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

```
[1] 2
```

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

```
[1] 50.5
```

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

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

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

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

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

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

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

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

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

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

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

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

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

```
[1] "mean ± 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:

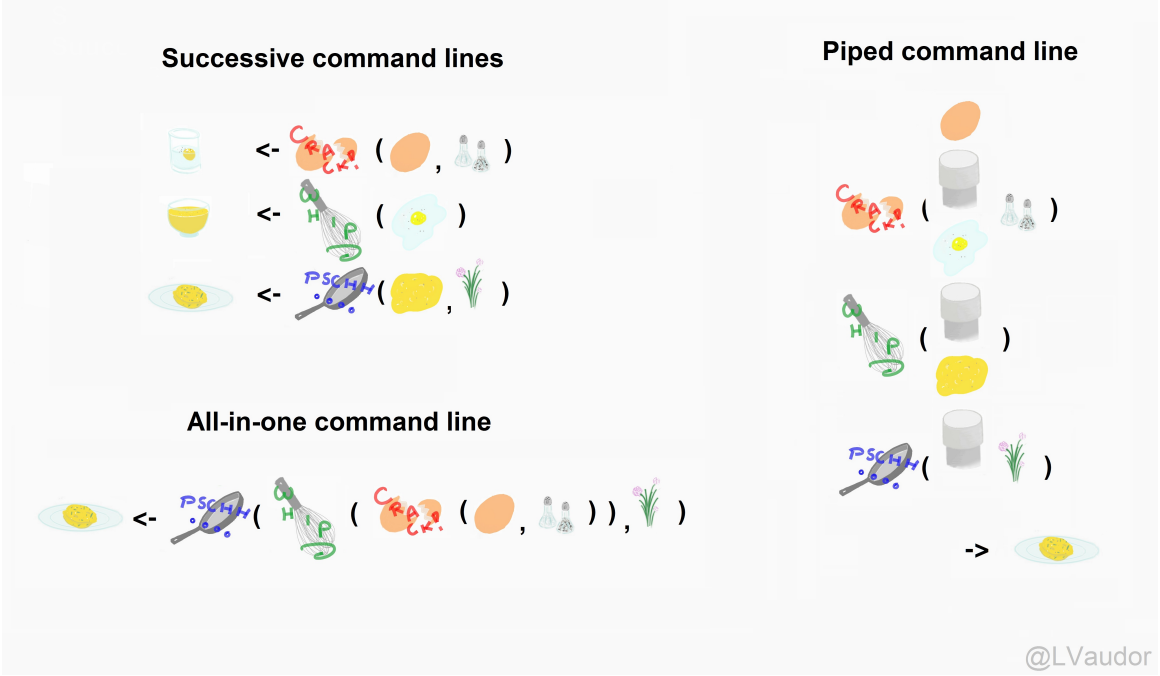


Figure 2.1: *Piping functions*

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

```
[1] 38
```

```
# or (usually better) piped:
mtcars |>   # inbuild example data, use F1!
  mutate(am=factor(am,
                   levels=c(0,1),
                   labels=c("automatic",
                            "manual")))  |> # %>% # #change into better class
  filter(vs==1) |> #filter out V-shaped
  group_by(am) |> # ask for grouped analysis
  summarize(across(.cols = c(wt, mpg, qsec, disp),
                   .fns = meansd))  |>
  pivot_longer(cols = -am, names_to = "Measure")  |> # put variables in rows
  pivot_wider(id_cols = Measure, names_from = am, #put groups in cols
              values_from = value)
```

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

If a sequence of functions is used often, combining them into a new function is advisable, e.g. this 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(y, x){
  return(x/y)
}
(Sys.Date()-as.Date("1958/12/10")) |>
  as.numeric() |>
  division(y = 365.25, x=_) |>
  floor()
```

```
[1] 65
```

```
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
set.seed(3105)
rawdata <- tibble(
  PatID=paste("P",1:patientN), # as in  data.frame
  Sex=sample(x = c("male","female"),  # random generator
             size = patientN,replace = TRUE,
             prob = c(.7,.3)),
  Ethnicity=sample(x = 1:6,
                   size = patientN,
                   replace = TRUE,
                   prob = c(.01,.01,.05,.03,.75,.15)),
  # random assignments
  `Given name`=randomNames(n = patientN,
                           gender = Sex,
                           # this is a reference to column Sex
                           ethnicity = Ethnicity,
                           which.names = "first"),
  `Family name`=randomNames(n = patientN,
                            ethnicity = Ethnicity,
                            which.names = "last"),
  Treatment=sample(x = c("Placebo","Verum"),
                   size = patientN,
                   replace = TRUE),
  `sysRR (mmHg)`=round(rnorm(n=patientN,mean=140,sd=10))-
    (Treatment=="Verum")*15,
  `diaRR (mmHg)`=round(rnorm(n=patientN,mean=80,sd=10))-
    (Treatment=="Verum")*10,
  HR=round(rnorm(n=patientN,mean=90,sd=7)))
rawdata
```

```
# A tibble: 25 x 9
   PatID Sex    Ethnicity `Given name` `Family name` Treatment `sysRR (mmHg)`
   <chr> <chr>      <int> <chr>        <chr>         <chr>                <dbl>
```

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

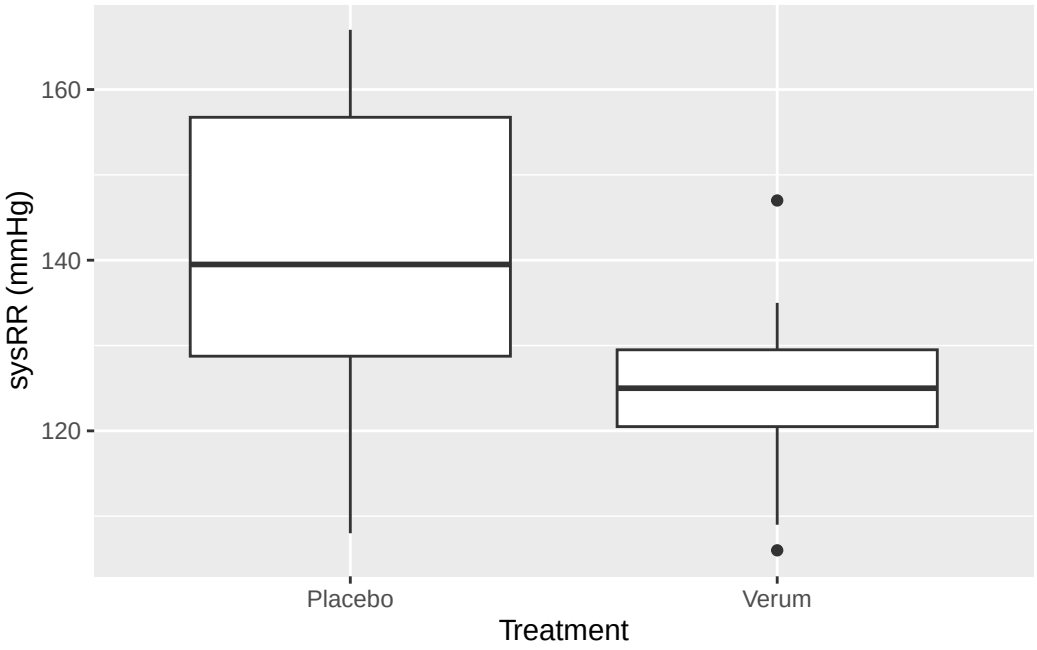
```
colnames(rawdata)
```

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

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

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

```
# example of data management for a tibble, recoding ethnicity:
rawdata <- rawdata  |>
  mutate(Ethnicity=factor(
    Ethnicity,levels = 1:6,
    labels= c(
      "American Indian or Native Alaskan",
      "Asian or Pacific Islander",
      "Black (not Hispanic)",
      "Hispanic",
      "White (not Hispanic)",
      "Middle-Eastern, Arabic")))
# quick visual inspection
ggplot(rawdata,aes(x = Treatment,y = `sysRR (mmHg)`))+
  geom_boxplot()
```



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

```
  rawdata[,6]
```

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

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

```
  rawdata[6]
```

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

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

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

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

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

Differences in addressing data frames and tibbles:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
# A tibble: 25 x 7
   PatID Sex    Ethnicity          Treatment `sysRR (mmHg)` `diaRR (mmHg)`    HR
   <chr> <chr>  <fct>              <chr>              <dbl>          <dbl> <dbl>
 1 P 1   male   White (not Hispan~ Verum                135             82    90
 2 P 2   male   White (not Hispan~ Verum                129             66    80
 3 P 3   male   White (not Hispan~ Verum                128             47    94
 4 P 4   male   American Indian o~ Verum                133             67    92
 5 P 5   female White (not Hispan~ Placebo              159             68    95
 6 P 6   male   White (not Hispan~ Placebo              153             89   100
```

```
 7 P 7   male   White (not Hispan~ Verum              126            86    86
 8 P 8   male   White (not Hispan~ Placebo            158            64    90
 9 P 9   male   White (not Hispan~ Placebo            121            70    91
10 P 10  male   White (not Hispan~ Verum              106            57    90
# i 15 more rows
```

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

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

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

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

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

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

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

```
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"
```

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

```
List of 10
 $ statistic  : Named num -5.73
  ..- attr(*, "names")= chr "t"
 $ parameter  : Named num 37.3
  ..- attr(*, "names")= chr "df"
 $ p.value    : num 1.43e-06
 $ conf.int   : num [1:2] -2.66 -1.27
  ..- attr(*, "conf.level")= num 0.95
 $ estimate   : Named num [1:2] 10.2 12.1
  ..- attr(*, "names")= chr [1:2] "mean of x" "mean of y"
 $ null.value : Named num 0
  ..- attr(*, "names")= chr "difference in means"
 $ stderr     : num 0.343
 $ alternative: chr "two.sided"
 $ method     : chr "Welch Two Sample t-test"
 $ data.name  : chr "rnorm(n = 20, mean = 10, sd = 1) and rnorm(20, 12, 1)"
 - attr(*, "class")= chr "htest"
```

```r
t_out$p.value
```

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

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

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

```r
  # 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"
```

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

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

```r
  # 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"
```

```r
# 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.

```r
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  # AND
```

```
[1] FALSE
```

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

```
[1] FALSE
```

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

```
[1] FALSE
```

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

```
[1] FALSE
```

```r
TRUE | FALSE  # OR
```

```
[1] TRUE
```

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

```
[1] TRUE
```

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

```
[1] TRUE
```

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

```
[1] TRUE
```

```r
any(all(sex=="female",age<=12),
    all(sex=="male", age <= 14))
```

```
[1] TRUE
```

2.6.2.2 *ifelse*

We can get text conditionally:

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

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

```
p <- .012
paste0("That is ",
       ifelse(test = p<=.05,yes = "",no = "not "),
       "significant")
```

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

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

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

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 for simple comparisons:

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

```
# A tibble: 25 x 4
   `sysRR (mmHg)` `diaRR (mmHg)` Hypertension `prescribe something?`
            <dbl>          <dbl> <chr>        <chr>
 1            135             82 hypertensive yes
```

```
 2            129            66 borderline   possibly
 3            128            47 borderline   possibly
 4            133            67 borderline   possibly
 5            159            68 borderline   possibly
 6            153            89 hypertensive yes
 7            126            86 hypertensive yes
 8            158            64 borderline   possibly
 9            121            70 borderline   possibly
10            106            57 normotensive no
# i 15 more rows
```

3 Regular expressions

3.1 Intro

Regex cheatsheet

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

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

Key symbols and their meanings:

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

3.1.1 Some basic examples:

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

```
[1] TRUE
```

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

```
[1] FALSE
```

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

```
[1] TRUE
```

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

```
[1] TRUE
```

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

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

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

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

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

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

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

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

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

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

3.1.2 An example for their use in renaming variables:

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

```
[1] "cup_weigh"        "CAPSIZE_cm"       "height_of_cup_cm"
```

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

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

3.2 Exercise

```
  testset1 <- c("Meier","Mayer","Maier","Meyer","Mayr","Maya","Mayor")
  # find all variations of the name "Meier"

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

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

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

4 Importing data

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

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

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

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

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

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

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

4.2 Import from Excel

4.2.1 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,
              .cols = !contains(c("BP","BMI","NY"))) |>
  rename(`Size (cm)`=Size, #newname=oldname
         `Weight (kg)`=Weight) |>
```

```
  select(-`Sex M`)
saveRDS(rawdata,file = "data/rawdata.rds")
```

4.2.2 ODS files

Package readODS provides similar functionality for OpenOffice/LibreOffice files.

4.2.3 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 |> 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,]
```

4.3 Import from SPSS/SAS

Import from SPSS generic files is implemented in various packages:

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

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

```
Zurückkodierung von CP1252
```

```
str(import1)
```

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

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

dbl+lbl [1:360] 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1... @ format.spss: chr "F8.0" @ labels : Named num [1:3] 1 2 3 ..- attr(*, "names")= chr [1:3] "Kontrolle" "AngII" "Whatever"

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

Kontrolle AngII Whatever 1 2 3

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

Factor w/ 3 levels “Kontrolle”,“AngII”,..: 1 1 1 1 1 1 1 1 1 1 ...

5 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 tidyr provides the functions pivot_wider and pivot_longer for conversations 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://tidyr.tidyverse.org/articles/pivot.html

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

5.1 Example 1: single repeated measure

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

```
# A tibble: 3 x 4
  ID    Var1  Var2Time1 Var2Time2
  <chr> <chr>     <dbl>     <dbl>
1 P 1   A          96.7     115.
2 P 2   B          91.7     105.
3 P 3   C          91.5      96.6
```

```
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  96.7
2 P 1   A     Var2Time2 115.
3 P 2   B     Var2Time1  91.7
4 P 2   B     Var2Time2 105.
5 P 3   C     Var2Time1  91.5
6 P 3   C     Var2Time2  96.6
```

5.2 Example 2: several repeated measures

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

```
  wide_data
```

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

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

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

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

```
# A tibble: 6 x 5
  ID    Var3  Time   Var1  Var2
```

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

```r
# alternatively in 1 step:
long_data <- pivot_longer(
  data=wide_data,
  cols=contains("Time"),
  names_to = c(".value","Time"), # .value will be replaced dynamically
  # names_sep = "Time"
  names_pattern = "(Var\\d+)(Time\\d+)"
)
long_data
```

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

5.3 Example 3: long to wide

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

```
# A tibble: 3 x 6
  ID    Var3  `Var1@Time1` `Var1@Time2` `Var2@Time1` `Var2@Time2`
  <chr> <chr>        <dbl>        <dbl>        <dbl>        <dbl>
1 P 1   A            121.          134.        13.0          13.0
```

```
2 P 2   B             91.5         104.         9.81         11.1
3 P 3   C            105.          115.        14.0          16.3
```

5.4 More examples

5.4.1 Step 1: Create example data:

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

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

head(rawdata)
```

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

5.4.2 Step 2: Transform that data to a long form:

- 1 column for weight
- 1 column for length
- 1 column for measurement time named "Visit"

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

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

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

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

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

5.4.3 Step 3 Transform long to wide

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

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

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

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

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

6 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')
```

6.1 Test for Normal distribution

6.1.1 Testing a single variable

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

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

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

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

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

```
    Asymptotic one-sample Kolmogorov-Smirnov test

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

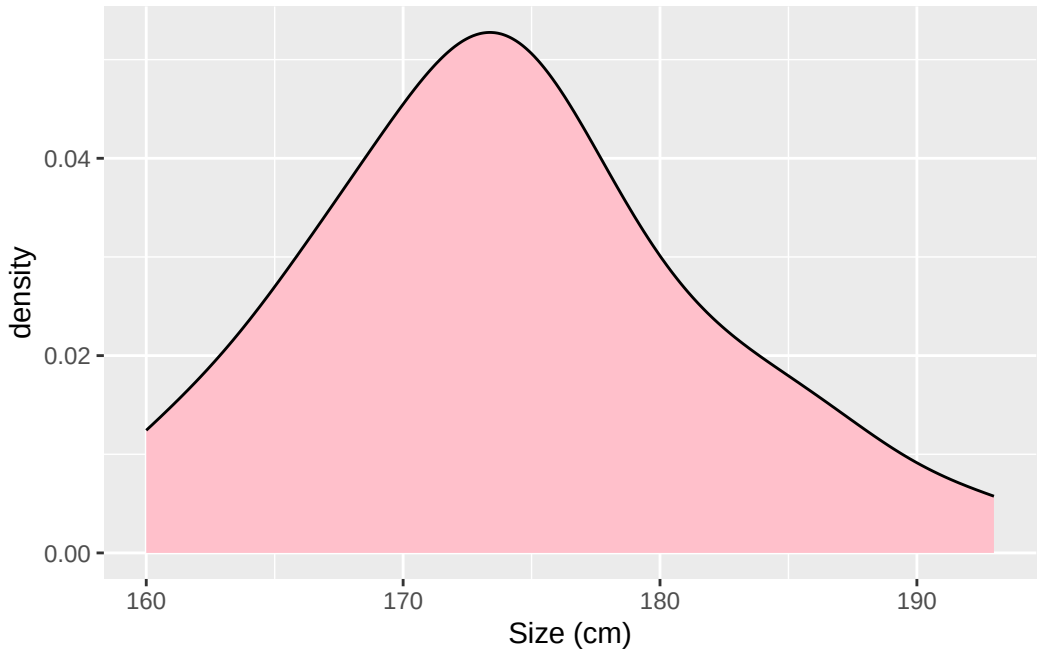
```
ksnormal(rawdata$`Size (cm)`)
```

```
p_Normal_Lilliefors
          0.2330547
```

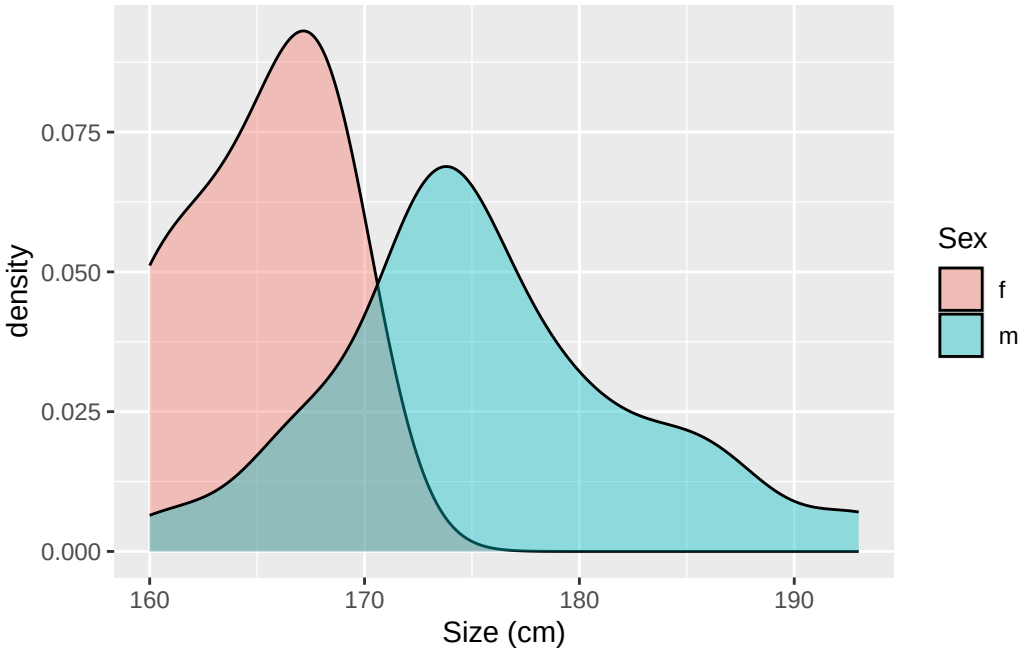
```
shapiro.test(rawdata$`Size (cm)`)
```

```
    Shapiro-Wilk normality test

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

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



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



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

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

```
p_Normal_Lilliefors
          0.1180981
```

```
rawdata |>
  group_by(Sex) |>
  summarize(p_KS = ksnormal(`Size (cm)`,lillie = FALSE),
            `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
```

6.1.2 Testing several variables

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

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

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

```
head(numvars$names)
```

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

```
numvars$count
```

```
[1] 23
```

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

Loop Version 1:

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

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

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

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

Loop Version 2:

```
  ## result table v2, just structure
  resulttable2 <- tibble(Measures=NA_character_,
                         pKS_Placebo=NA_character_,
                         pKS_Verum=NA_character_,
                         .rows = 0)
  for(var_i in numvars$names){
    ks_tmp <- by(data = rawdata[[var_i]],
                 INDICES=rawdata$Testmedication,
                 FUN=ksnormal,
                 lillie=FALSE)
    resulttable2 <-add_row(resulttable2,
                           Measures=var_i,
                           pKS_Placebo=ks_tmp[[1]] |>
```

```
                          formatP(), # added rounding/formatting
                        pKS_Verum=ks_tmp[[2]] |>
                          formatP())
}
head(resulttable2)
```

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

across() - Version:

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

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

```
  head(resulttable1)
```

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

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

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

```
  head(resulttable2)
```

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

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

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

```
rm(numvars)
```

6.2 Picking column names and positions

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

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

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

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

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

7 Visualize data with ggplot

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

```
pacman::p_load(conflicted, tidyverse,
               grid,gridExtra,car,
               ggsci,ggsignif, ggthemes, ggridges,
               # gganimate,
               ggforce,
               ggbeeswarm,
               wrappedtools,
               emojifont,
               patchwork)
conflicts_prefer(dplyr::filter) # solves name conflict
```

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

7.1 Example data

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

```
head(diamonds)
```

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

```
head(mtcars)
```

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

7.2 Basic structure of a ggplot call

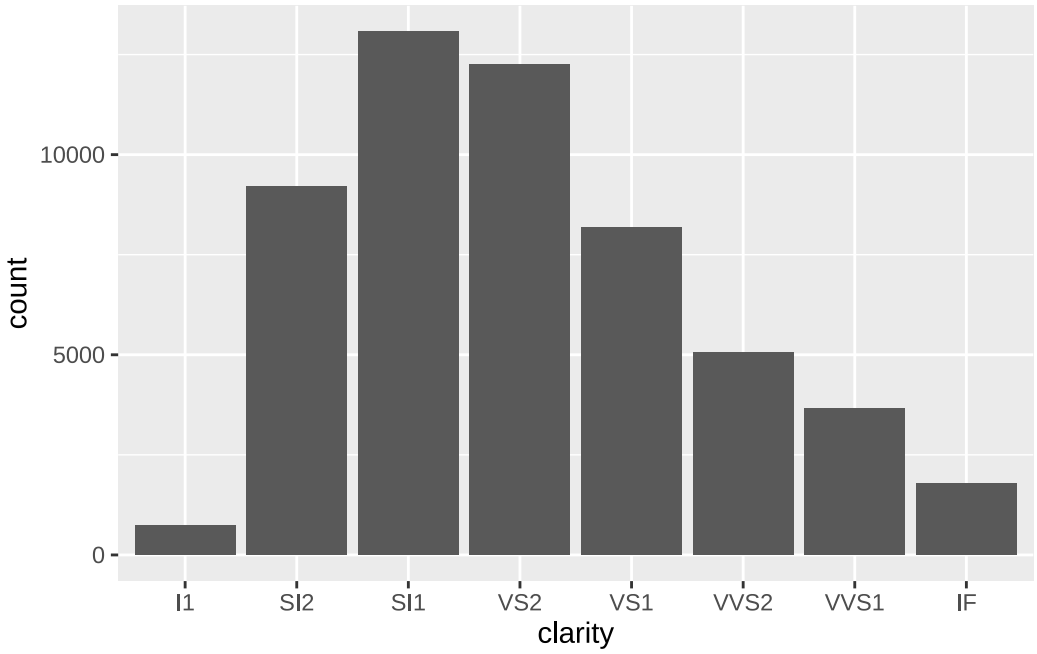
To create a figure, we at least use 2 function calls:

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

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

Minimal example:

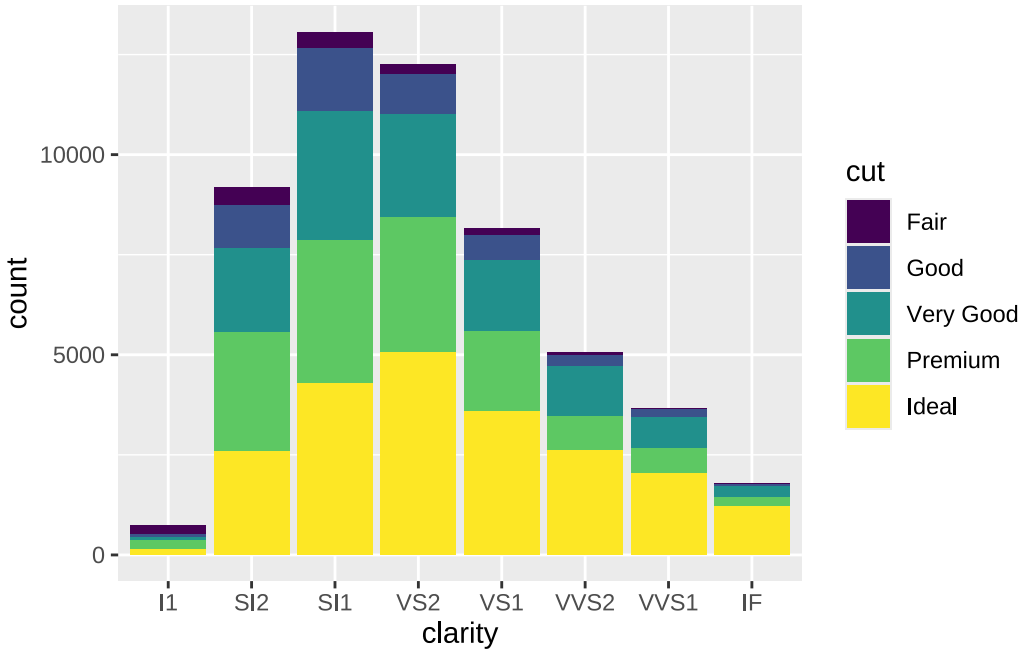
```
ggplot(data=diamonds,mapping = aes(x=clarity))+
  geom_bar()
```



geom_bar() inherits the global aestetic 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-color. This automatically creates subgroups for counting:

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



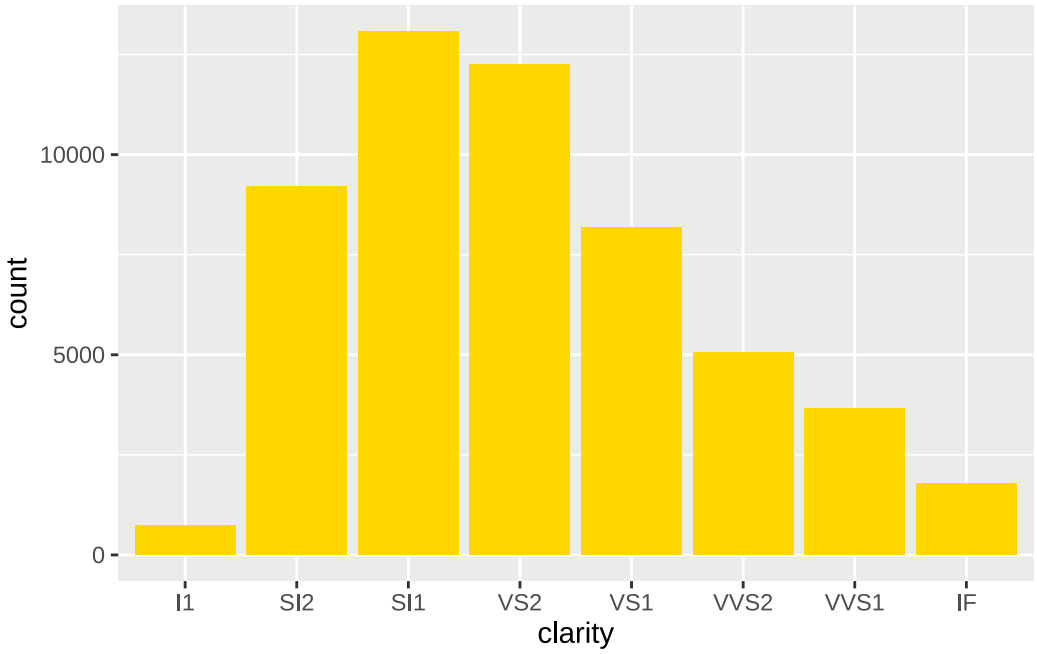
This graph represents this count table:

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

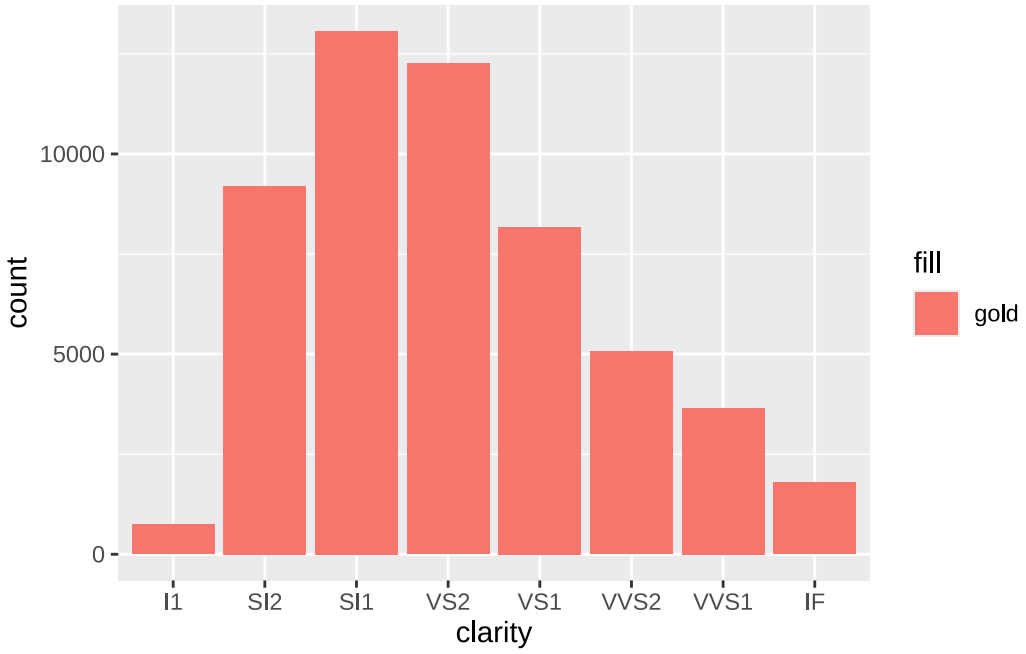
```
# A tibble: 5 x 9
# Groups:   cut [5]
  cut          I1   SI2   SI1   VS2   VS1  VVS2  VVS1    IF
  <ord>     <int> <int> <int> <int> <int> <int> <int> <int>
1 Fair        210   466   408   261   170    69    17     9
2 Good         96  1081  1560   978   648   286   186    71
3 Very Good    84 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

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

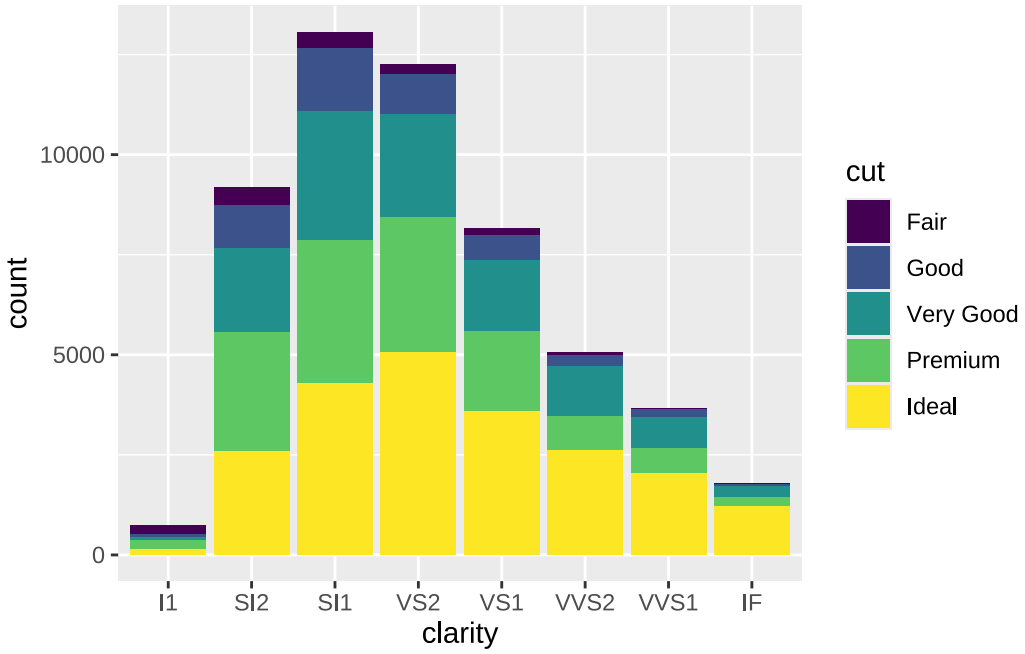
```
#aesthetics outside aes
ggplot(data=diamonds,aes(x=clarity))+
  geom_bar(fill="gold")
```




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




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



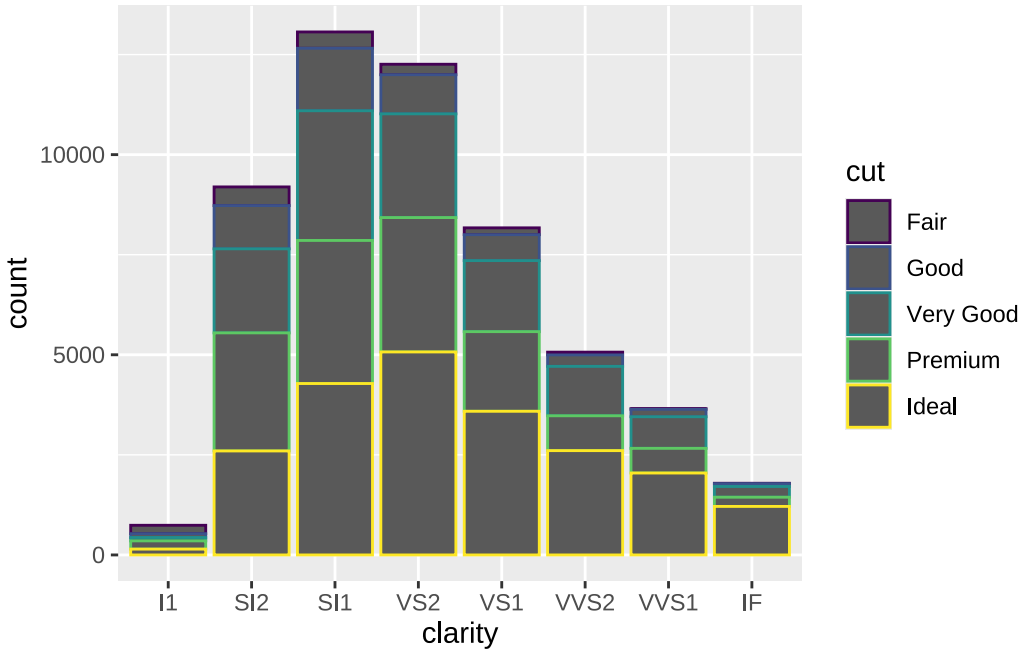
7.3 fill vs. color

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

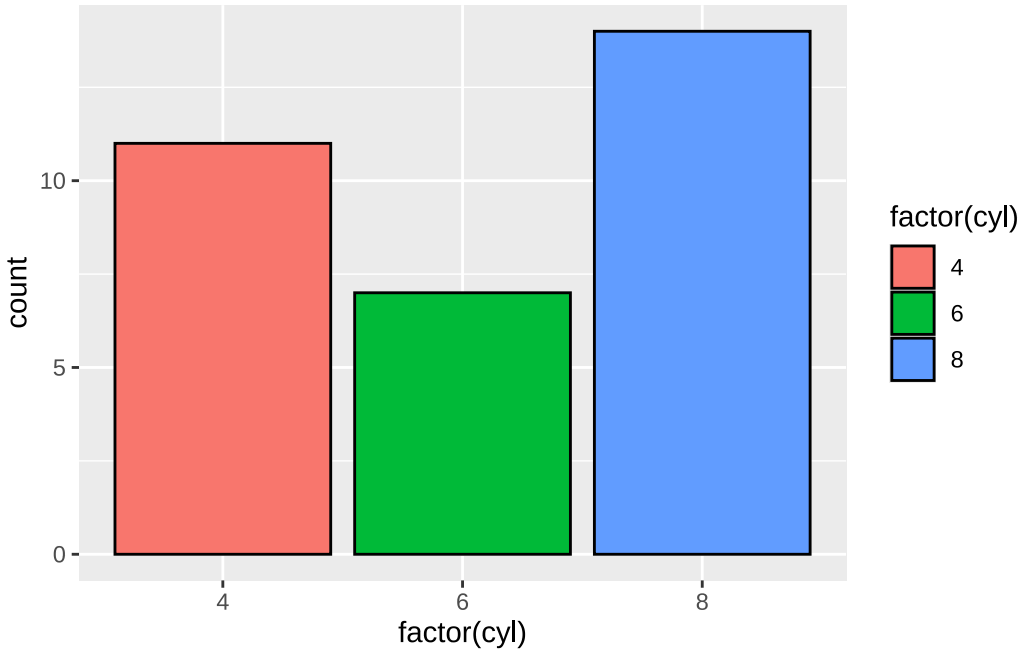
- inner color, defined by `fill`
- outer frame color, defined by `color`

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

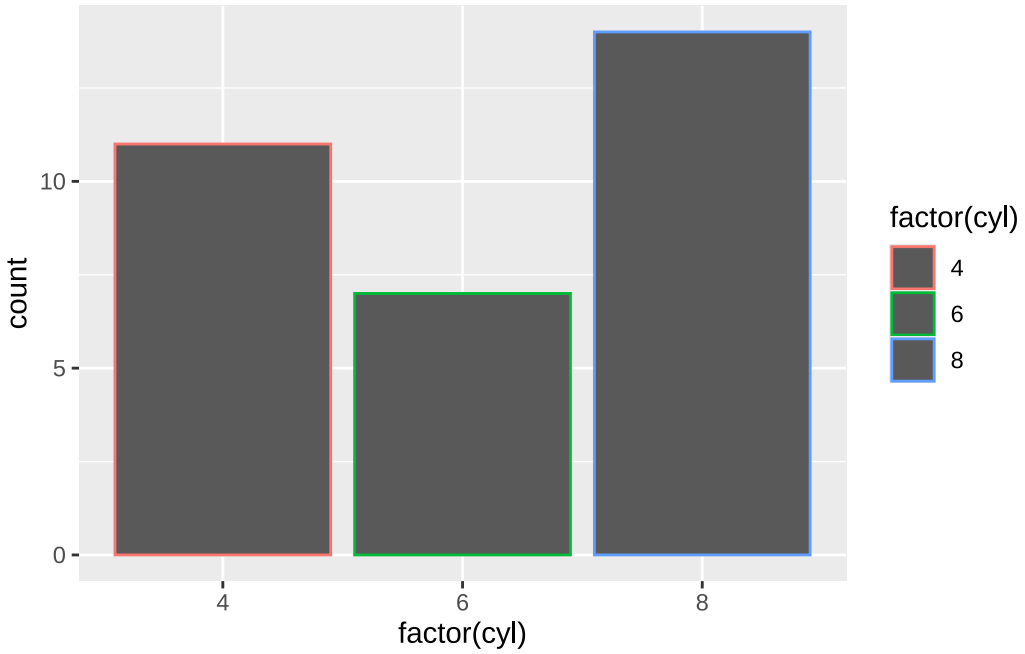
```
ggplot(data=diamonds,aes(x=clarity,color=cut))+
  geom_bar()
```




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




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

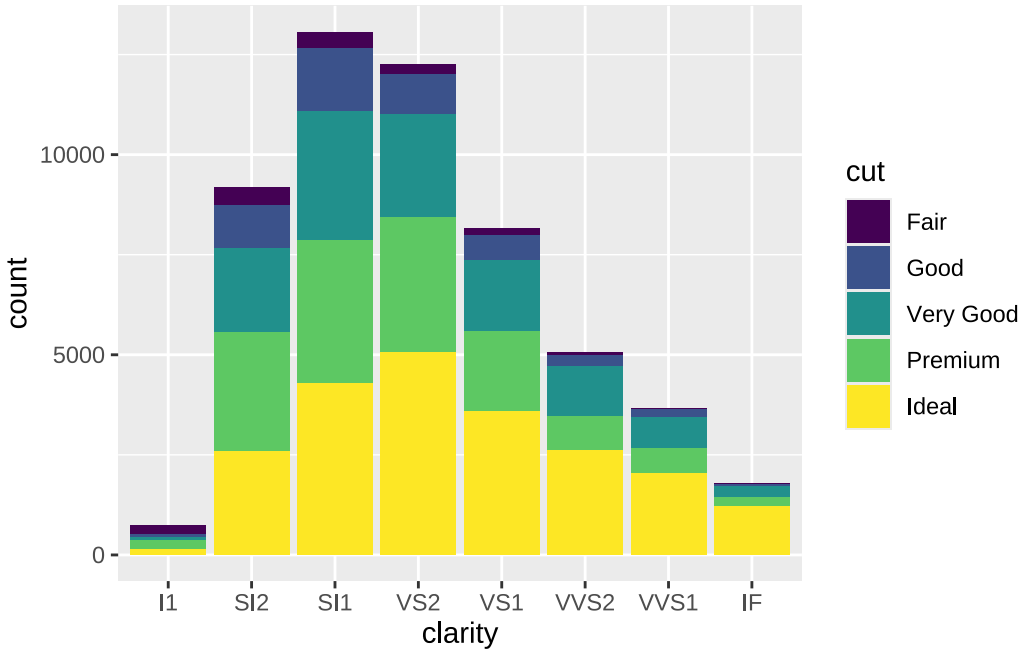


7.4 Color systems

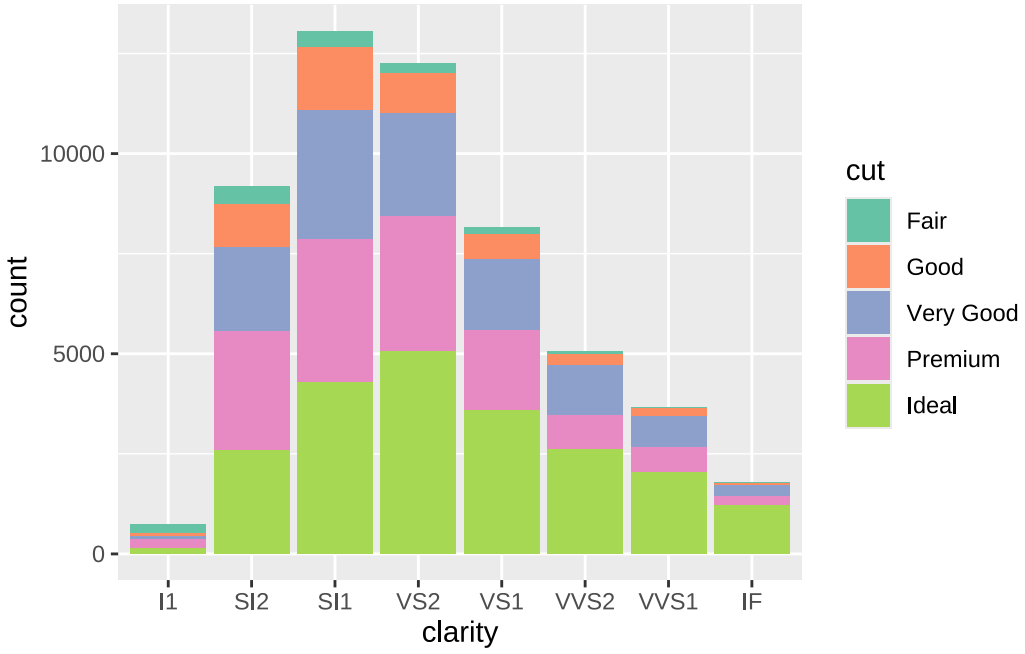
ggplot2 comes with various color definitions, many external packages extend that. Manual definition of colors is possible as well. Redefining the mapping between data and aesthetics can be done with scale_... functions

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

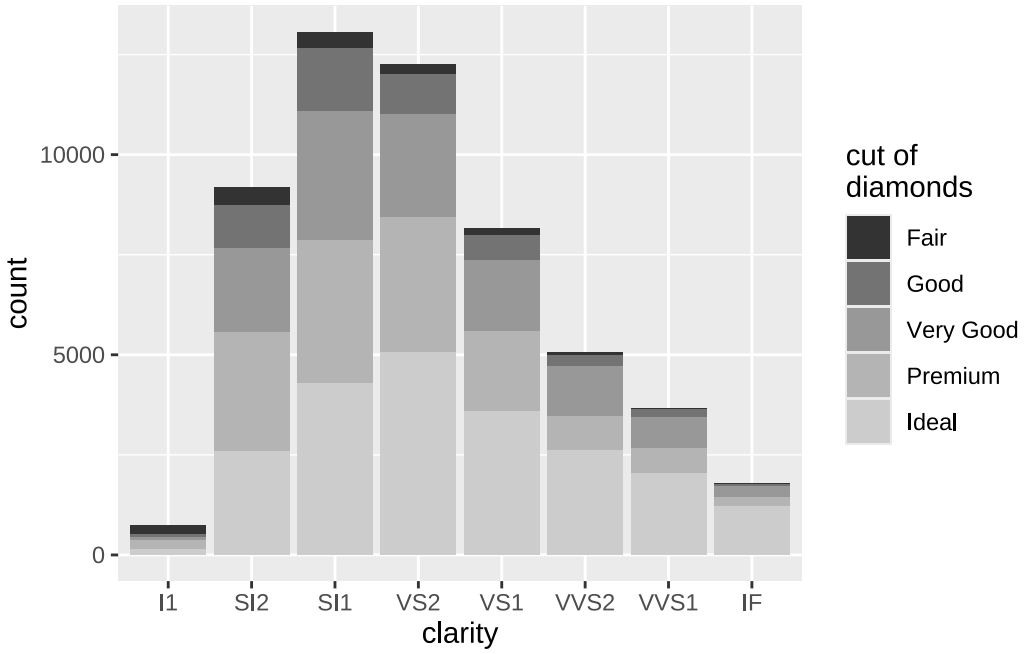
```
(plottemp <- ggplot(data=diamonds,aes(x=clarity,fill=cut))+
    geom_bar())
```



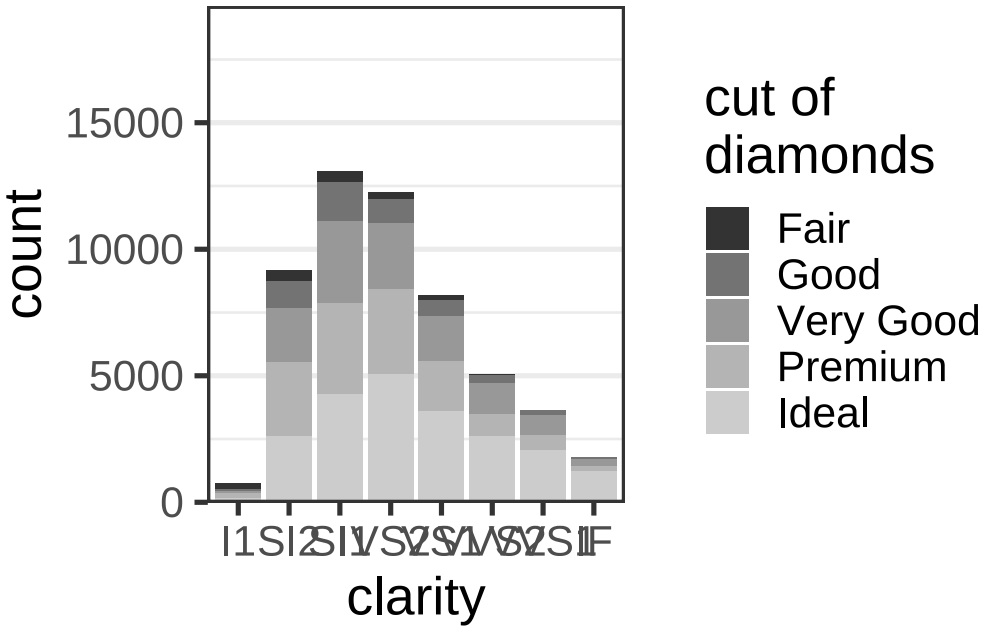

```
plottemp + scale_fill_brewer(palette="Set2") #in-built "scale" for fill
```




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

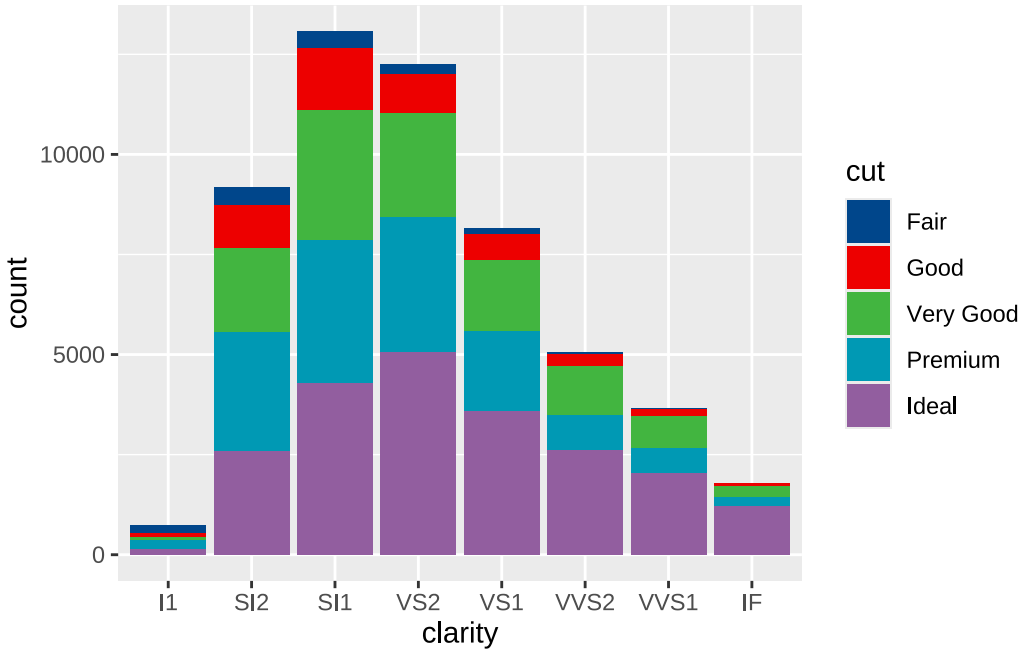



```
plottemp + scale_fill_grey(name = "cut of\ndiamonds") +
  scale_y_continuous(expand = expansion(mult = c(0,.5)))+ # rescaling y
  theme_bw(base_size = 20)+
  theme(panel.grid.major.x = element_blank())
```

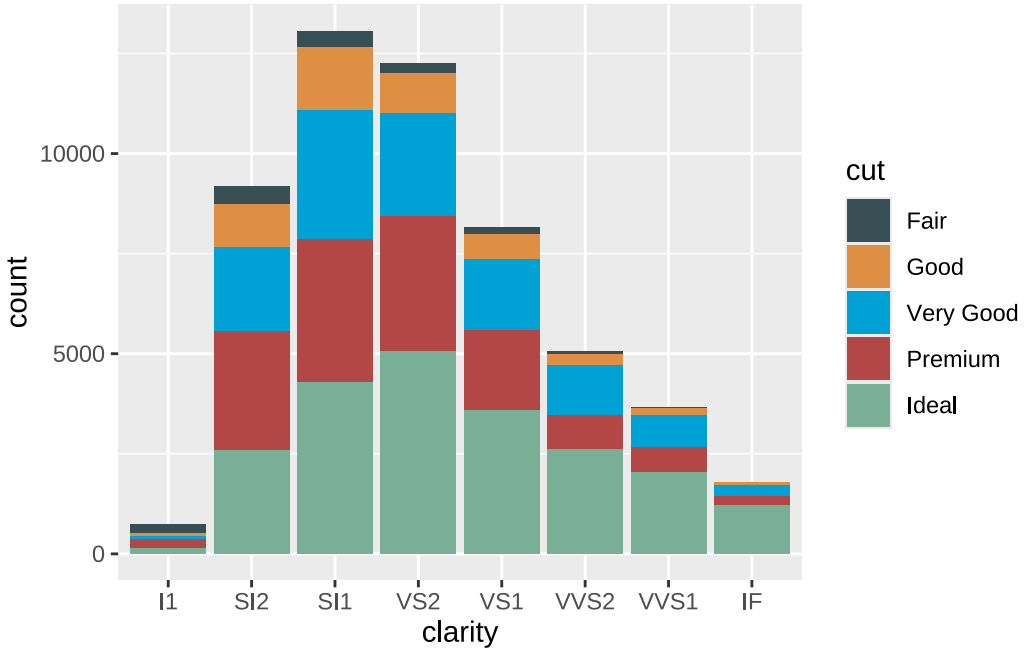


7.4.1 External color definitions from ggsci

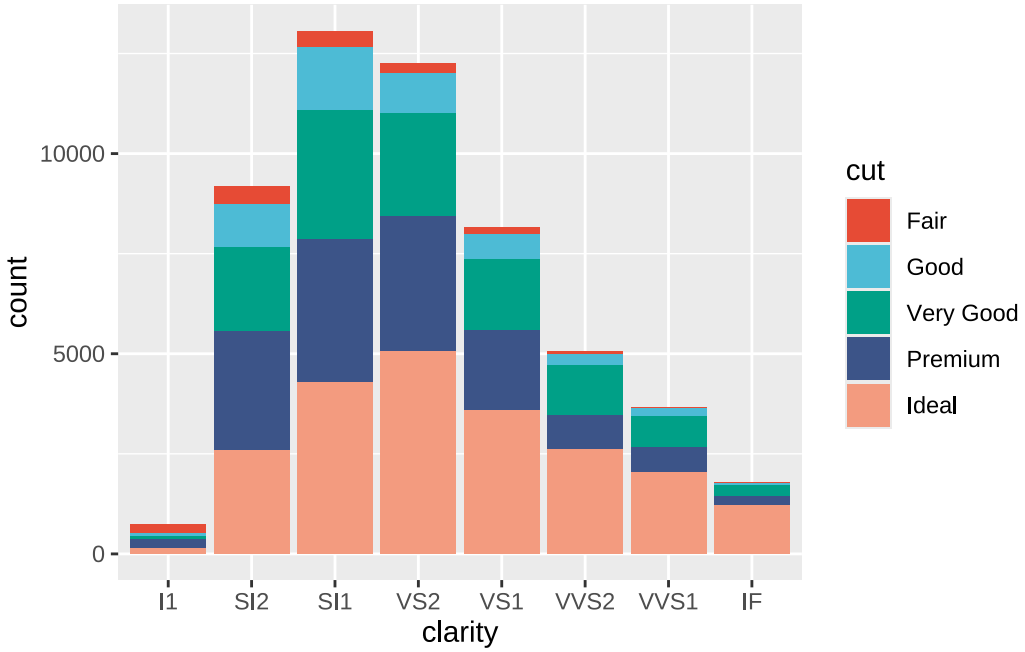
```
plottemp+scale_fill_lancet()
```



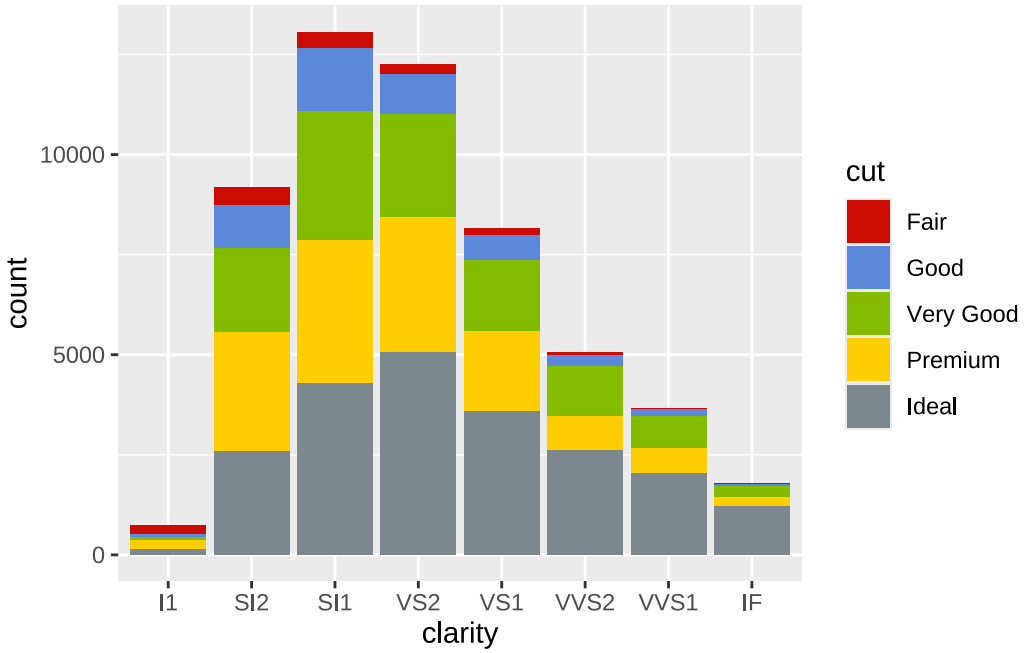

```
plottemp+scale_fill_jama()
```



```
plottemp+scale_fill_npg()
```




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



7.5 Exporting ggplots

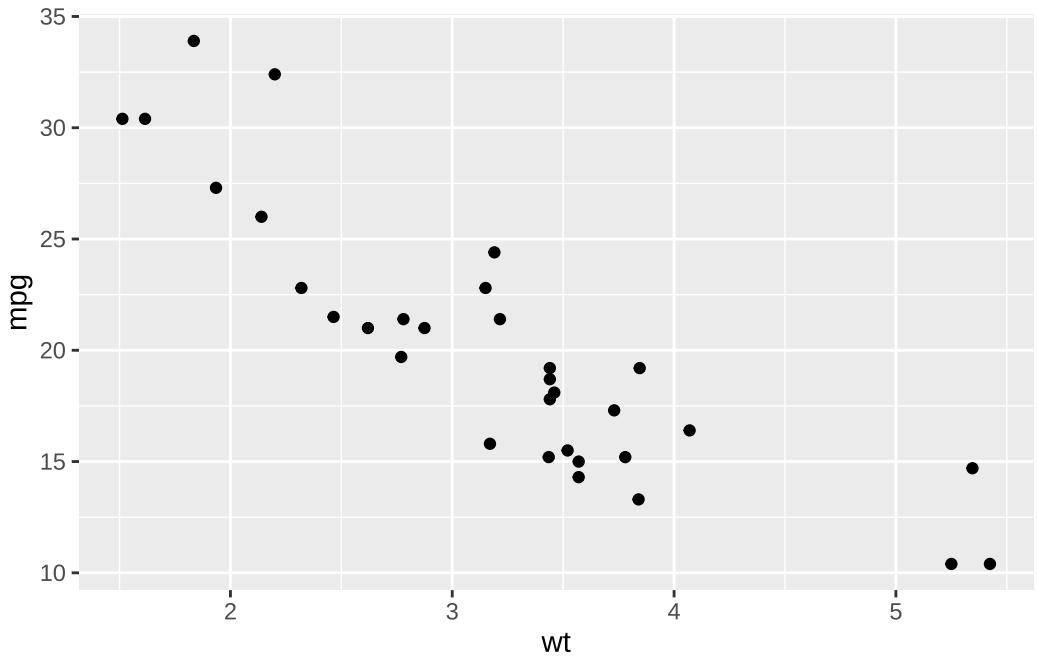
There are two distinct ways, using ggsave() or more generally creating an external graphic device with e.g. png() / tiff() / pdf():

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

7.6 Other geoms

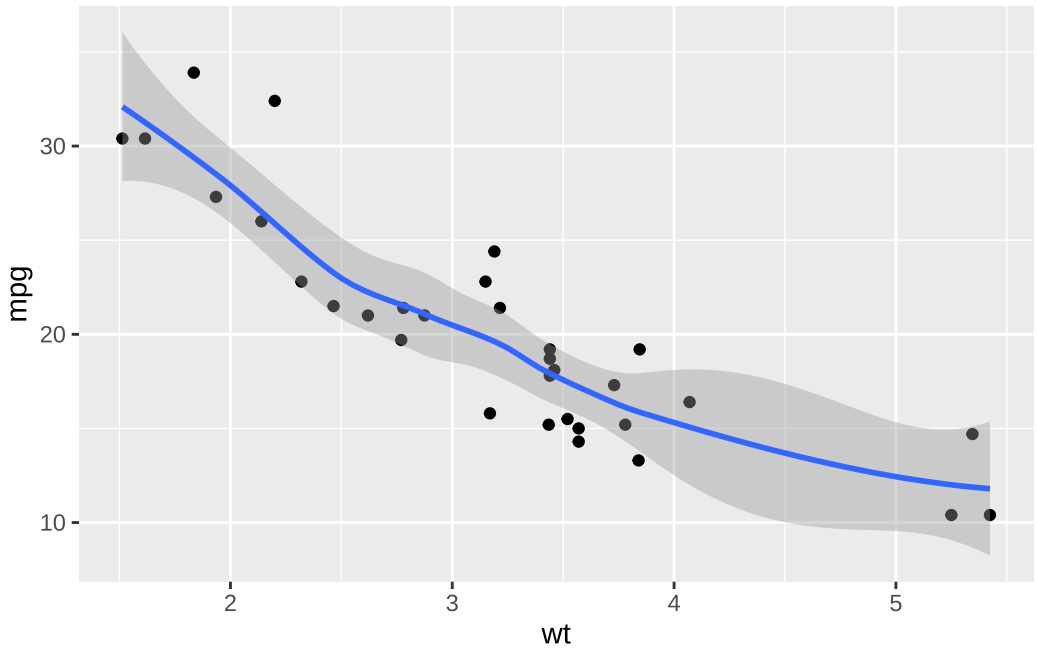
Common forms of plots are barplots, boxplots, scatterplots (possibly with regression line), and density-plots. For plotting dots for groups, there are various options to avoid over-plotting of repeated data, with the beeswarm as my preference.

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



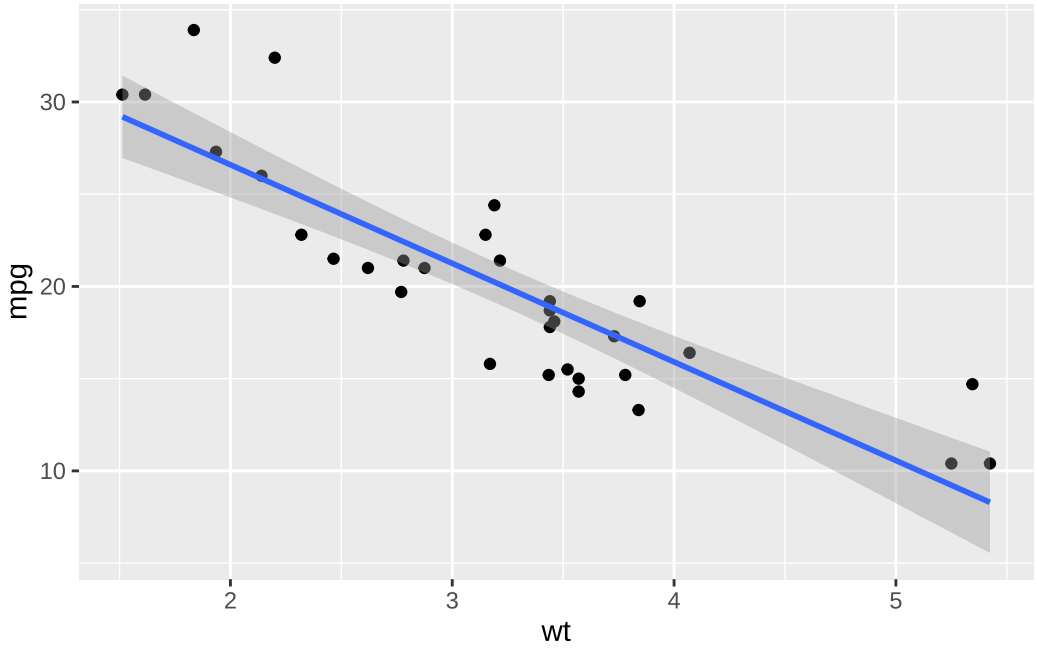

```
ggplot(data=mtcars,aes(x = wt,y = mpg))+
  geom_point()+
  geom_smooth()
```

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

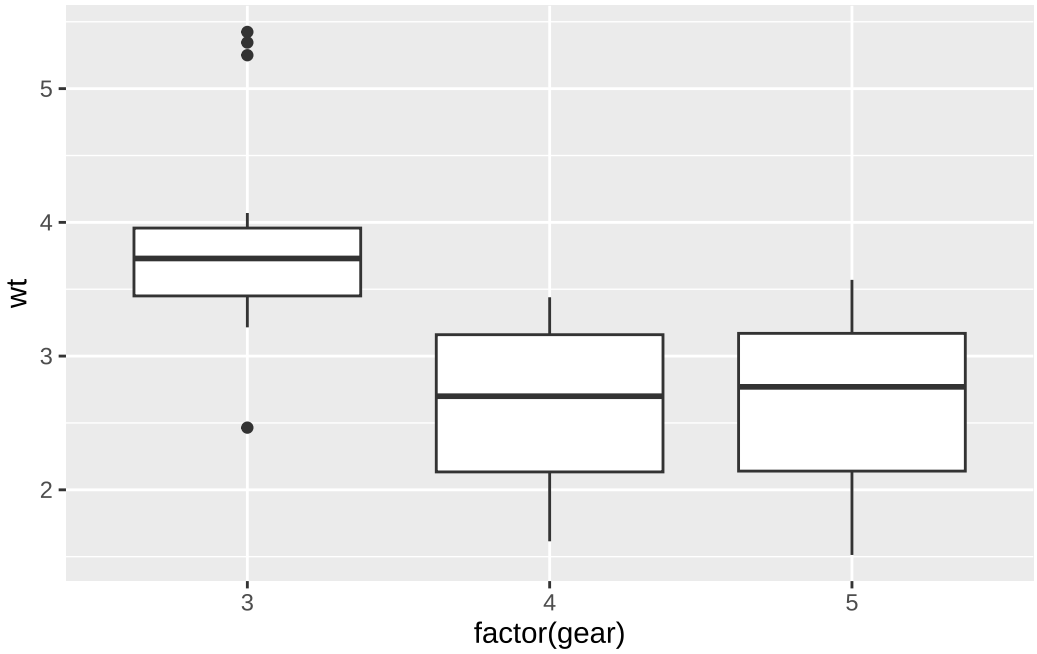


```
ggplot(data=mtcars,aes(x = wt,y = mpg))+
  geom_point()+
  geom_smooth(method="lm")
```

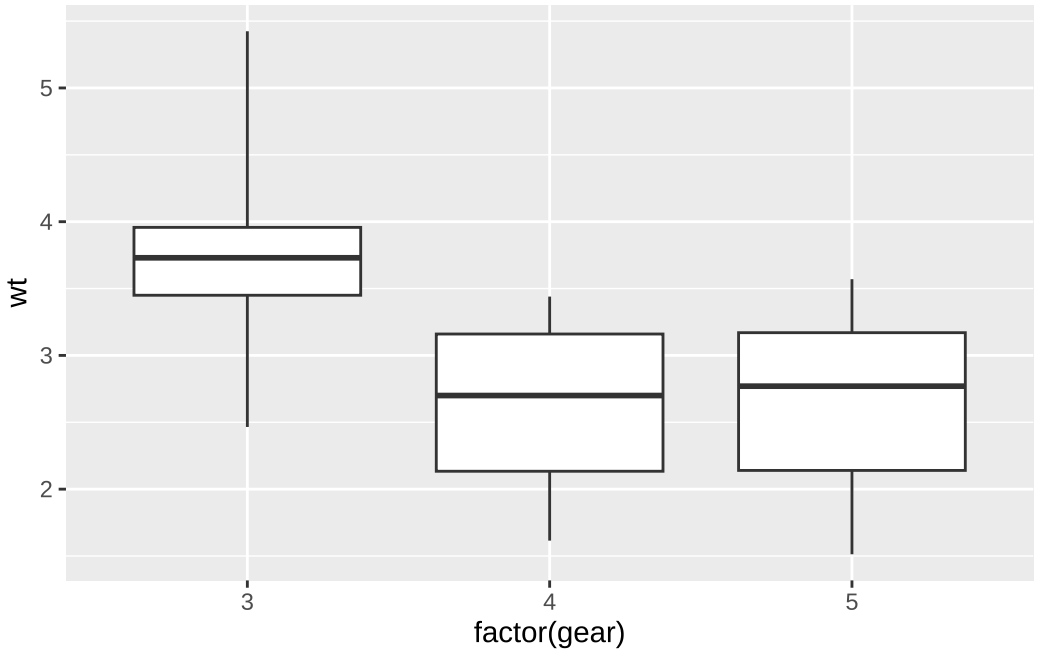
```
`geom_smooth()` using formula = 'y ~ x'
```




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

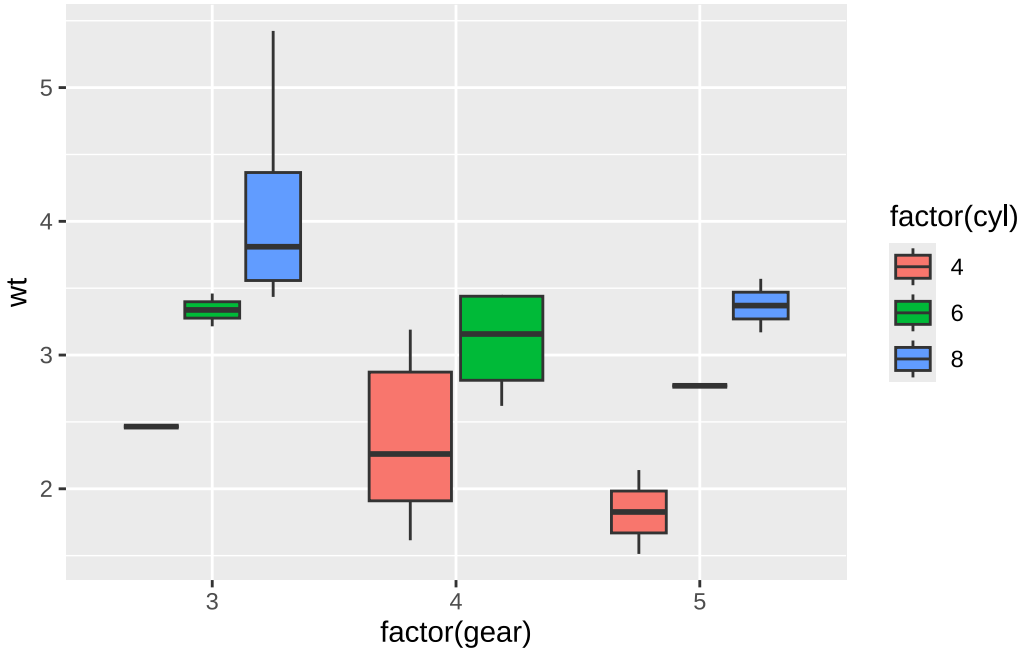



```
ggplot(mtcars,aes(x = factor(gear),y = wt))+
  geom_boxplot(coef=3) # this extends range of expected values
```

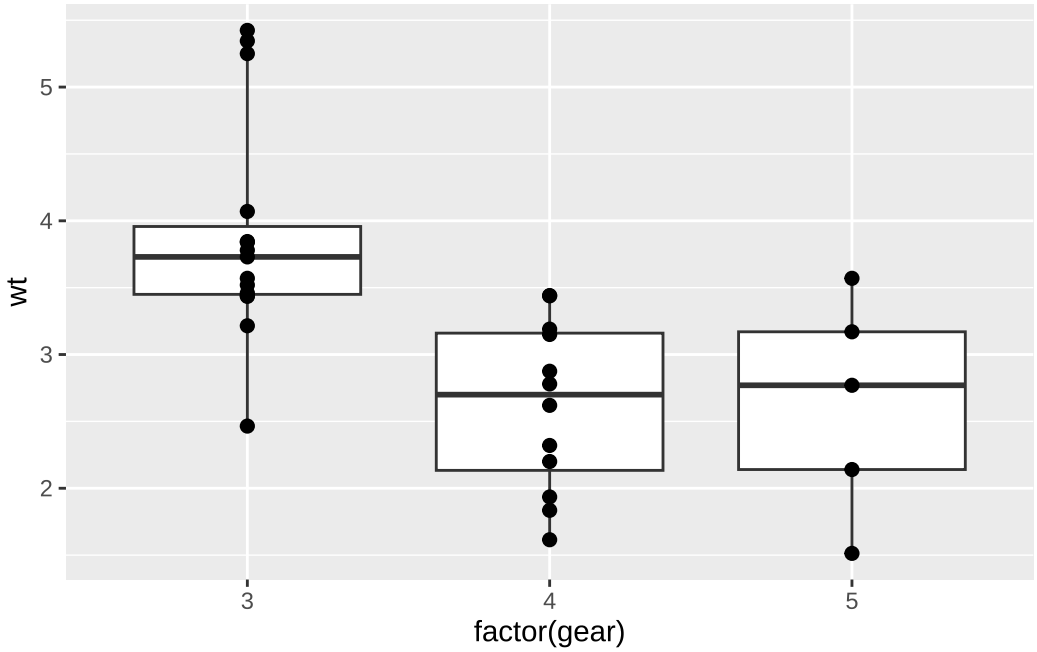



```
ggplot(mtcars,aes(x = factor(gear),y = wt,
                  fill=factor(cyl)))+
```

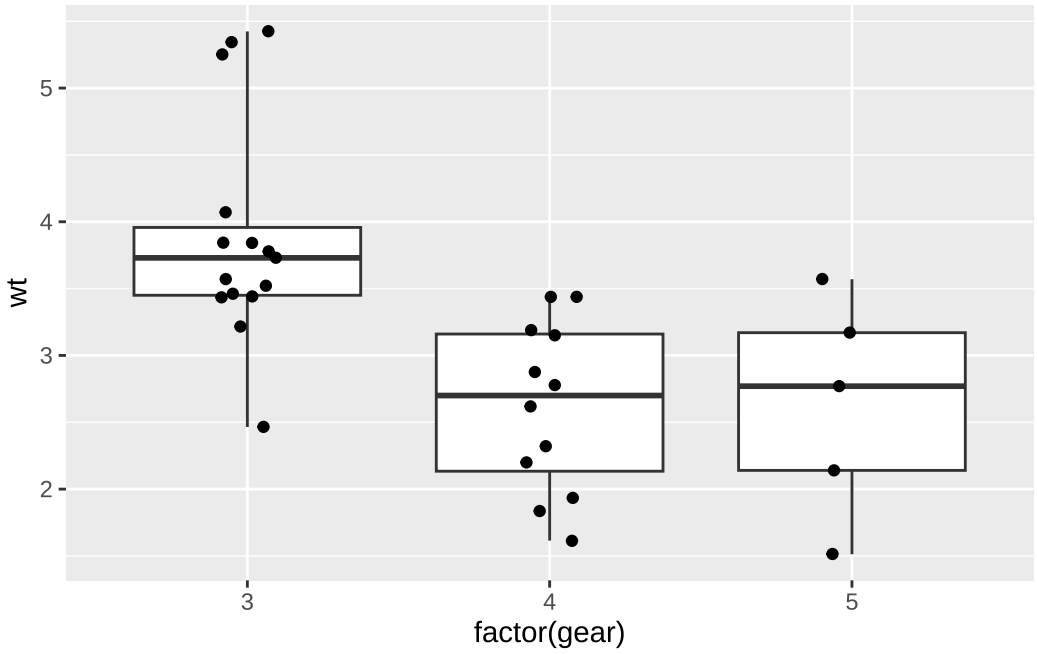
```
  geom_boxplot(coef=3) # group by cyl, as it is mapped to fill
```




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

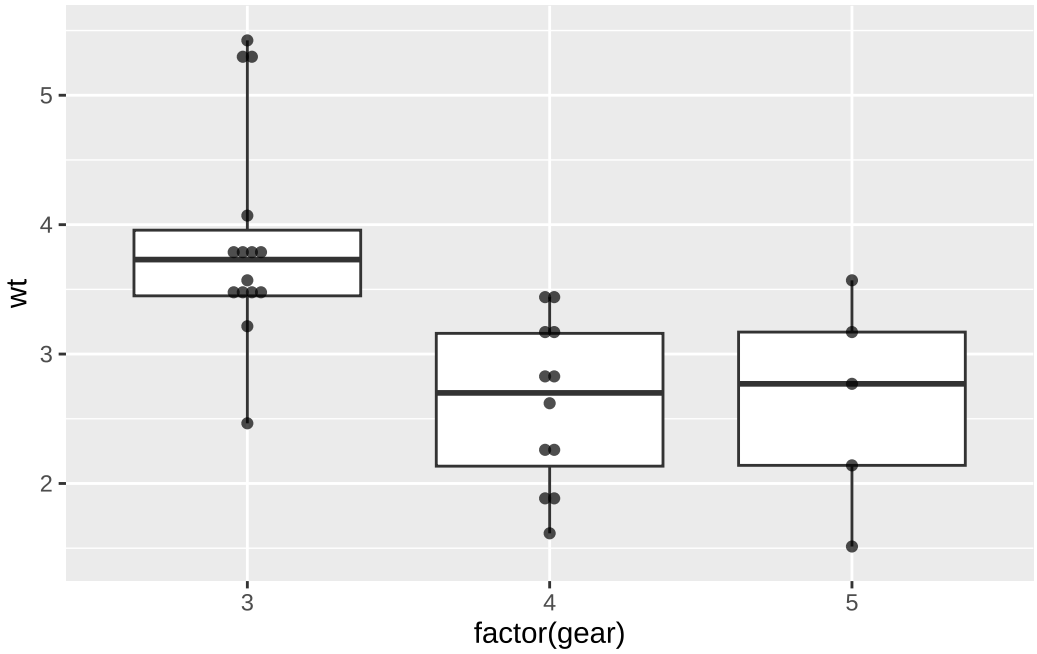


```
ggplot(mtcars,aes(x = factor(gear),y = wt))+
  geom_boxplot(coef=3)+
  geom_point(position = position_jitter(width = .1))
```

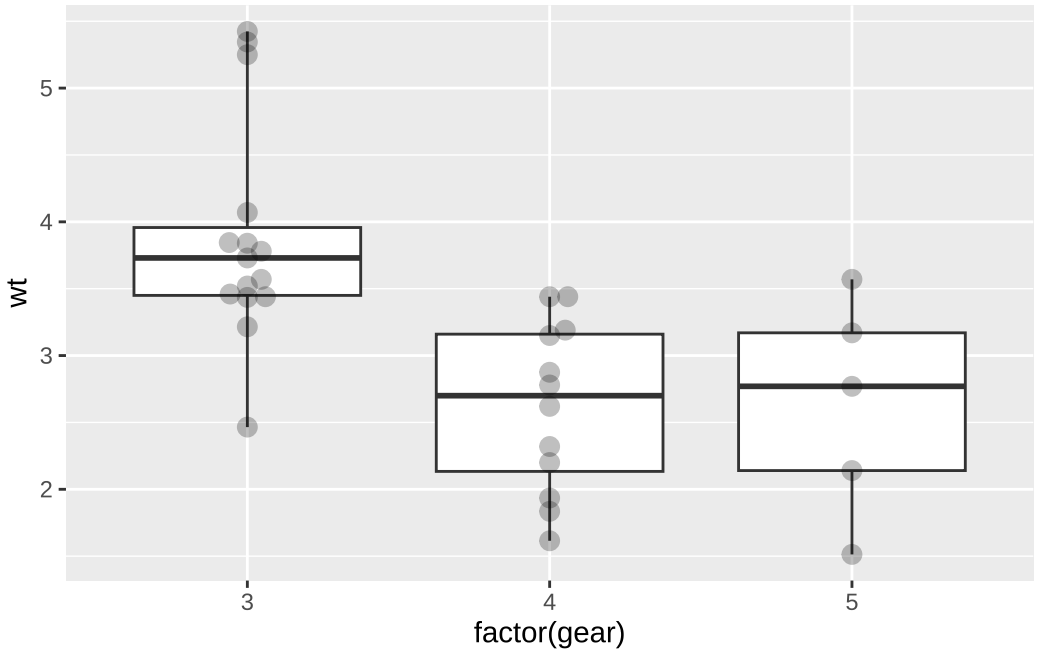



```
ggplot(mtcars,aes(x = factor(gear),y = wt))+
  geom_boxplot(coef=3)+
  geom_dotplot(alpha=.7, # group similar(ish) data on a line
               binaxis = "y",stackdir = "center",
               stackratio = .9,dotsize = .6)
```

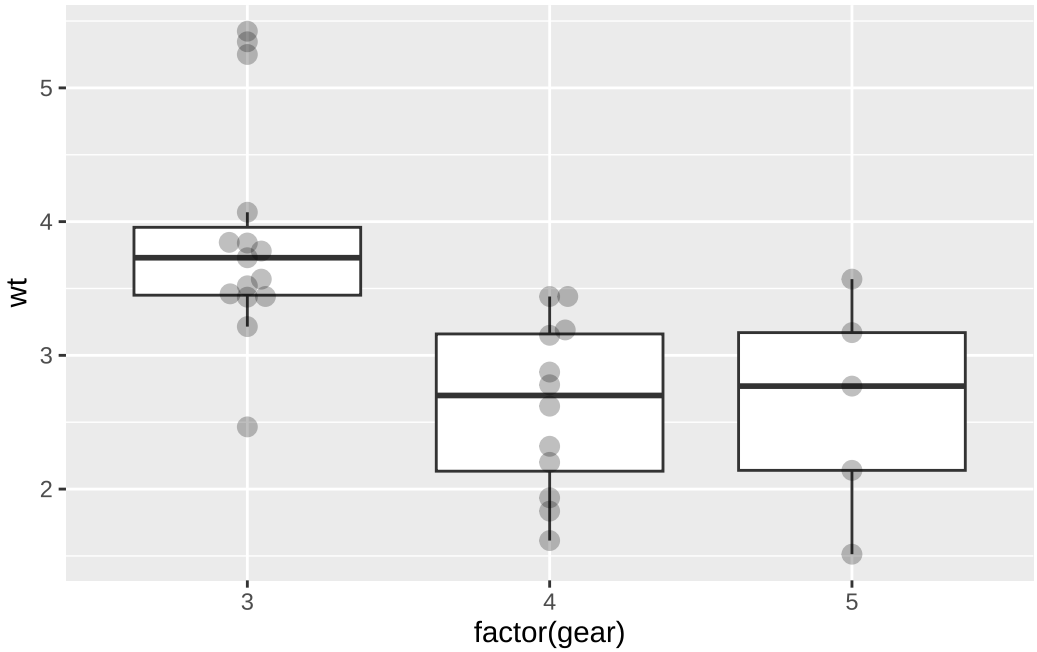
```
Bin width defaults to 1/30 of the range of the data. Pick better value with
`binwidth`.
```



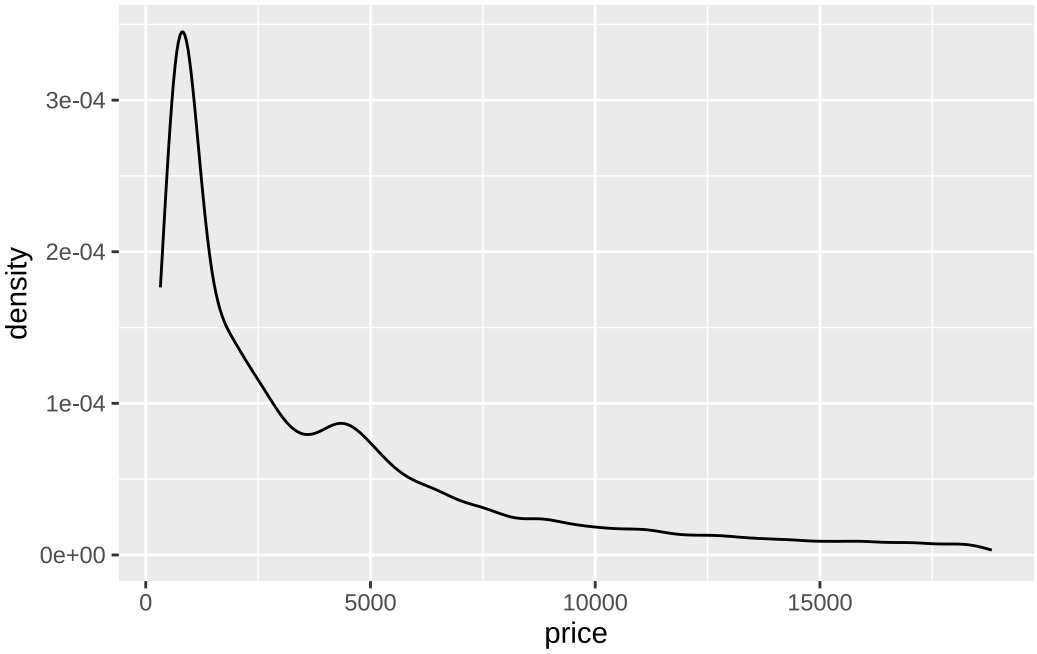

```
ggplot(mtcars,aes(x = factor(gear),y = wt))+
  geom_boxplot(coef=3)+
  ggbeeswarm::geom_beeswarm(cex = 2,size=3,alpha=.25)
```



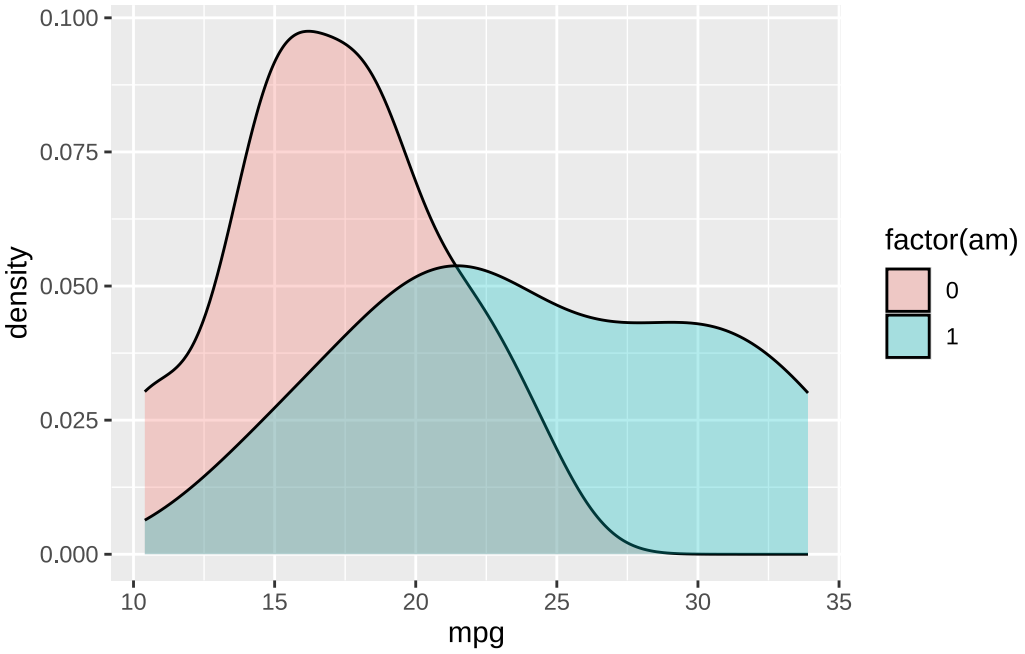
```
ggplot(mtcars,aes(x = factor(gear),y = wt))+
  geom_boxplot(outlier.alpha = 0)+ # to avoid plotting outliers twice
  geom_beeswarm(cex = 2,size=3,alpha=.25)
```




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

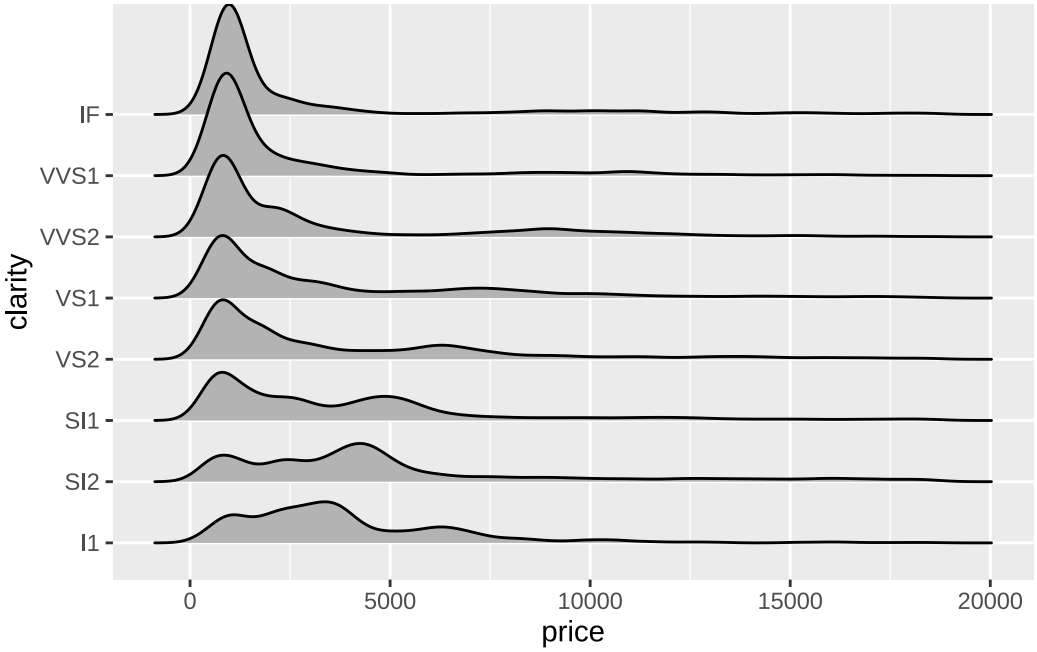



```
ggplot(mtcars,aes(mpg, fill=factor(am)))+
  geom_density(alpha=.3)
```



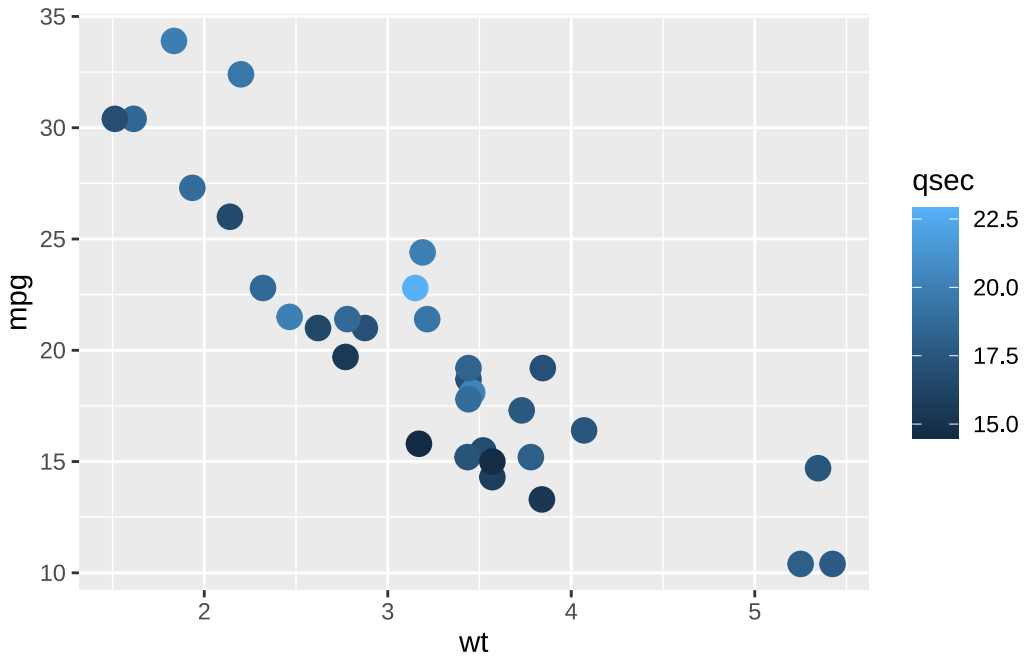

```
ggplot(diamonds,aes(price,y=clarity))+
  geom_density_ridges()
```

```
Picking joint bandwidth of 403
```

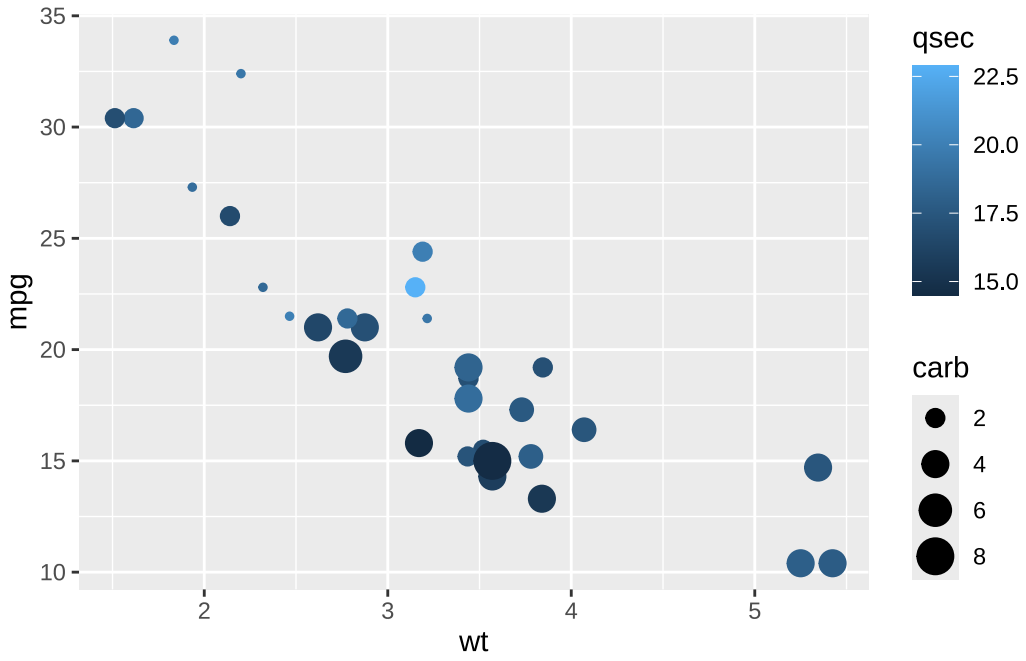


7.7 Combining and finetuning aesthetics

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

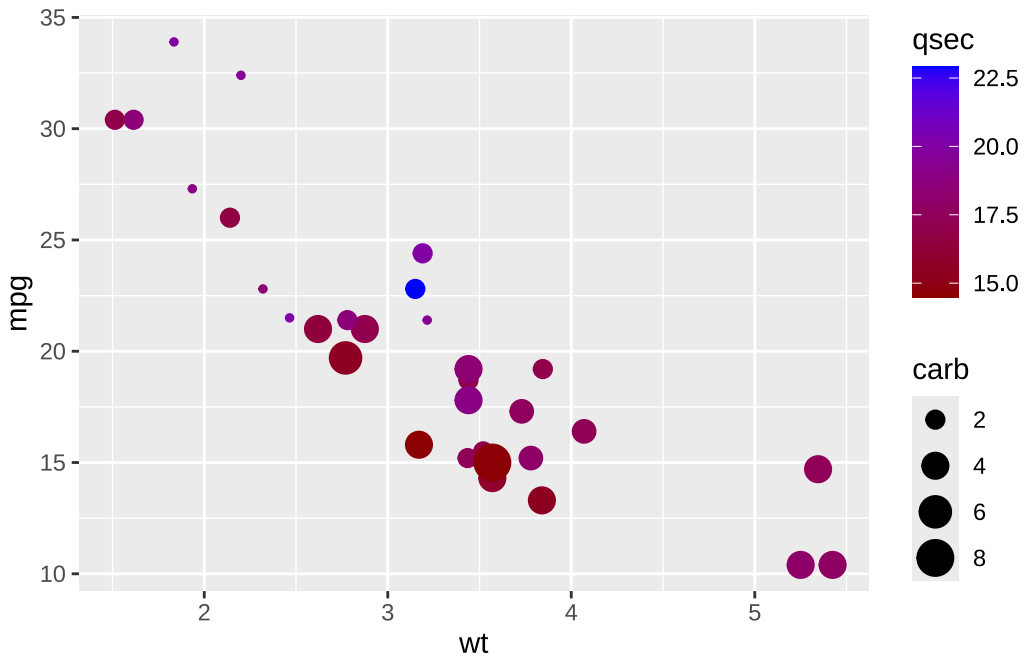



```
ggplot(data=mtcars,aes(wt, mpg,color=qsec, size=carb))+
  geom_point()
```

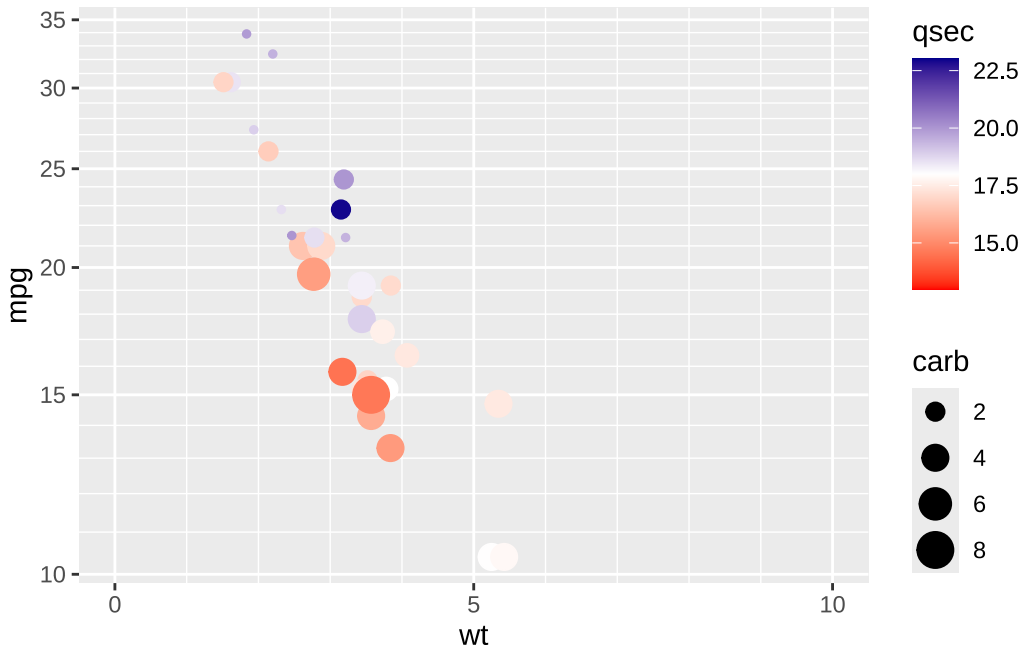



```
ggplot(data=mtcars,aes(wt, mpg,color=qsec, size=carb))+
  scale_color_gradient(low="darkred",high="blue")+
```

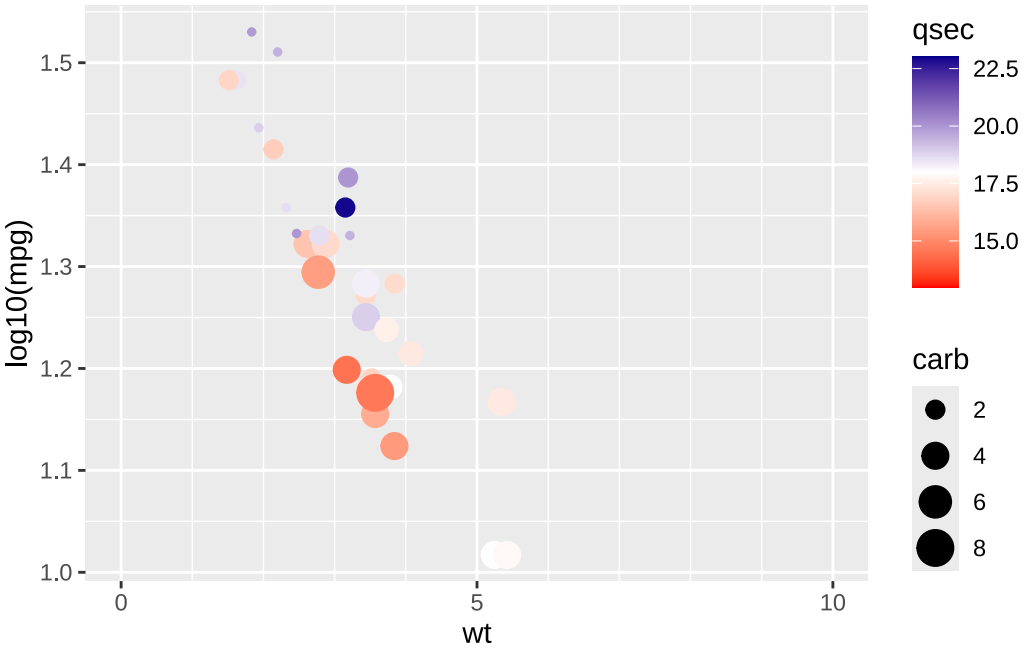
```
  geom_point()
```



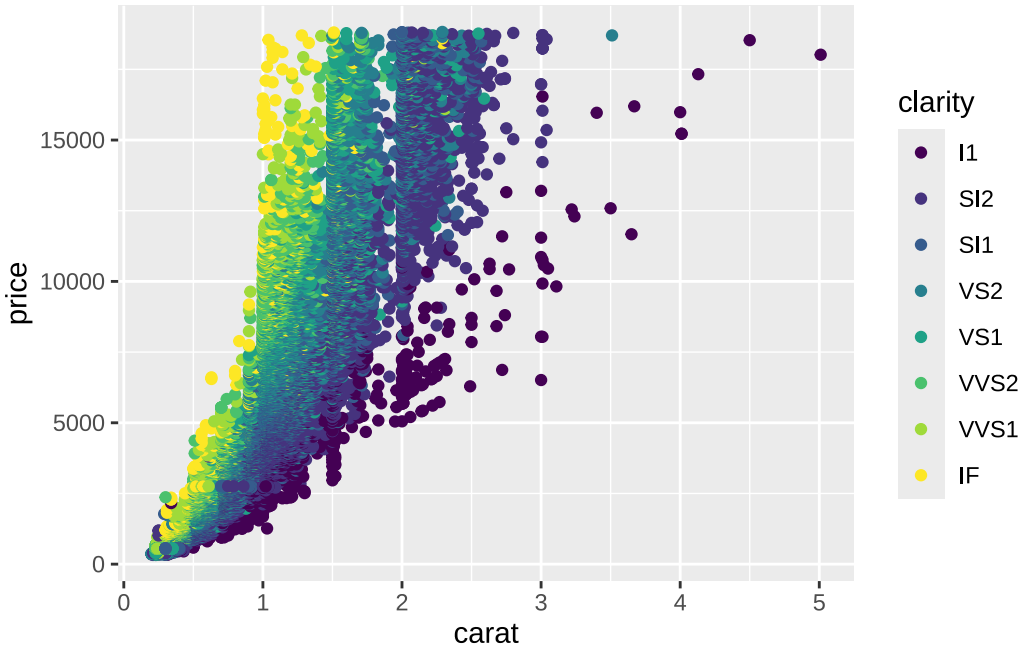

```
ggplot(data=mtcars,aes(wt, mpg,color=qsec, size=carb))+
  scale_color_gradient2(low="red",high="darkblue",
                        mid="white",
                        limits=c(13,23),midpoint=18)+
  geom_point()+
  scale_x_continuous(breaks = seq(-10^3,10^3,5),
                     minor_breaks=seq(-10^3,10^3,1),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



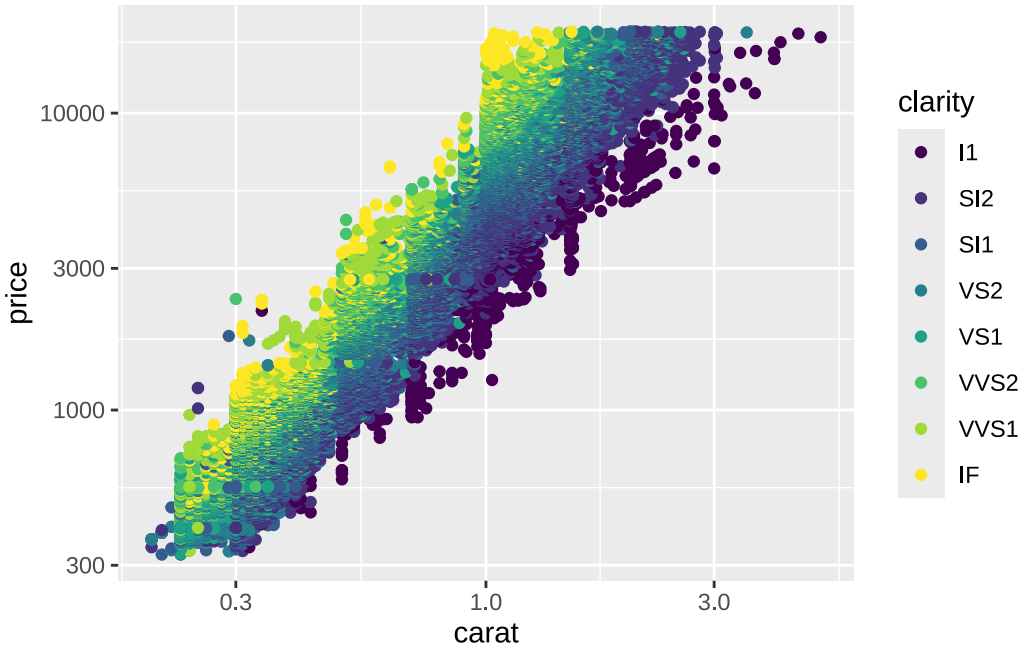

```
ggplot(data=mtcars,aes(wt, log10(mpg),color=qsec, size=carb))+
  scale_color_gradient2(low="red",high="darkblue",
                        mid="white",
                        limits=c(13,23),midpoint=18)+
  geom_point()+
  scale_x_continuous(breaks = seq(0,100,5),
                     minor_breaks=seq(0,100,1),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



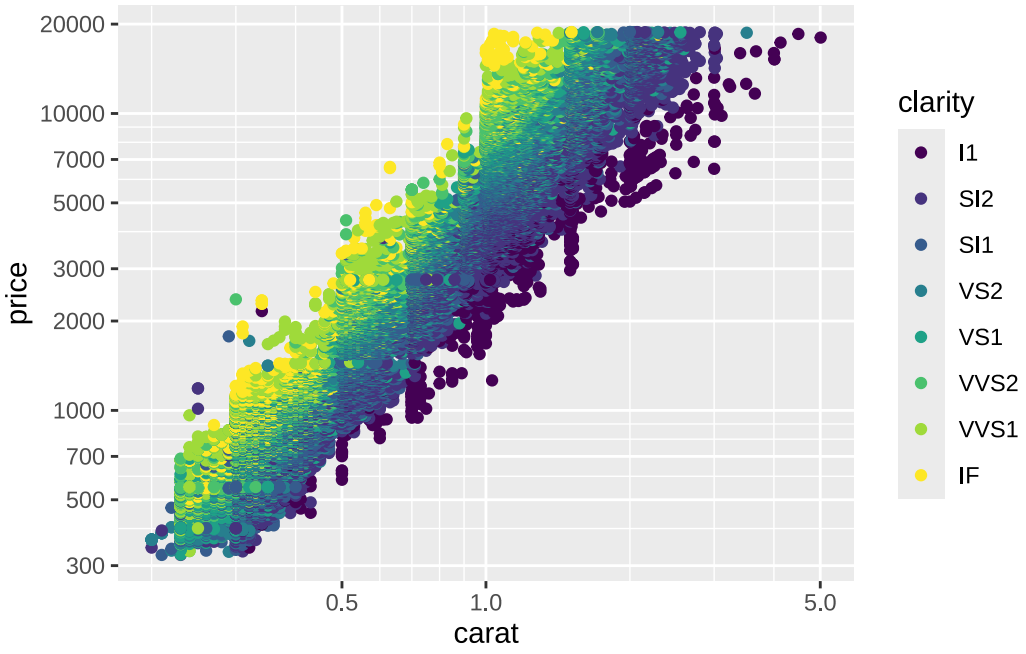

```
  # scale_y_log10(minor_breaks=seq(0,100,1))



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



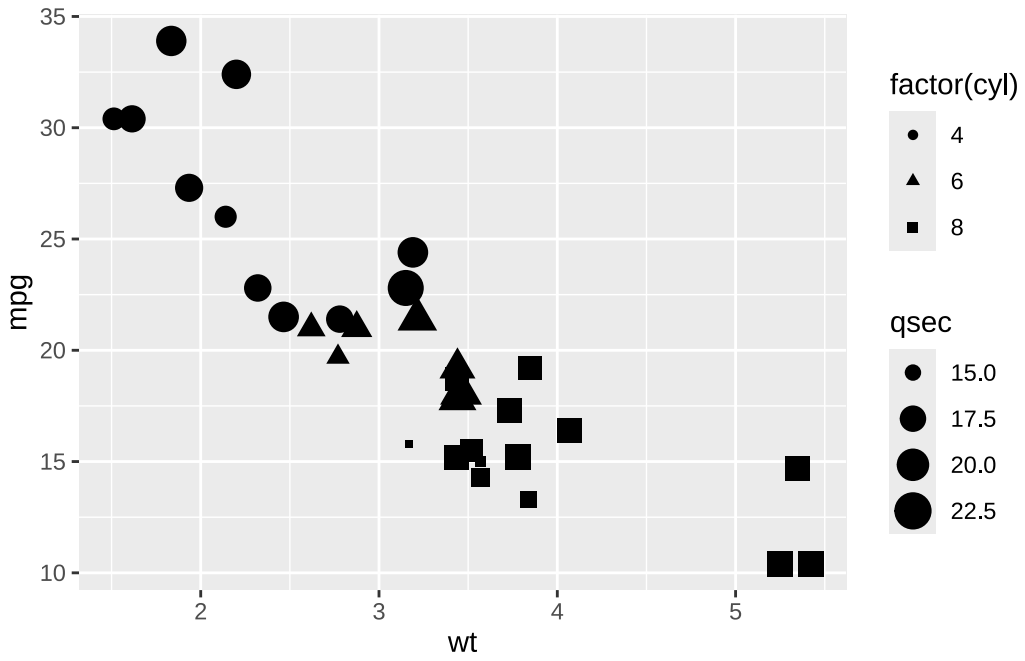

```
ggplot(diamonds,aes(carat,price,color=clarity))+
  geom_point()+
  scale_x_log10()+
  scale_y_log10()
```



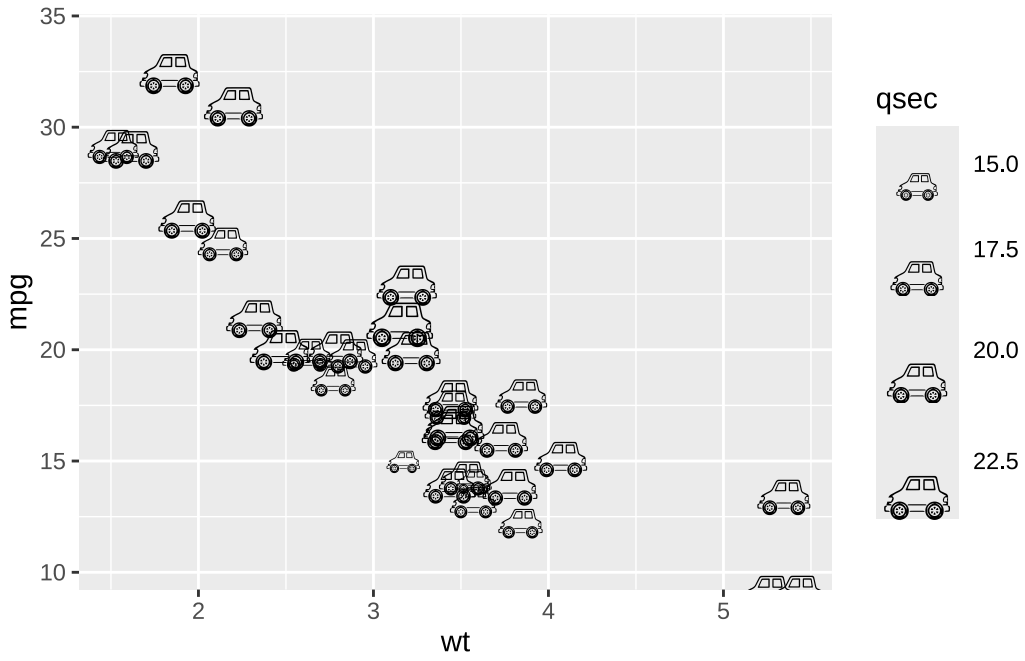
```
ggplot(diamonds,aes(carat,price,color=clarity))+
  geom_point()+
  scale_x_log10(
    breaks=lograngе_15,
    minor_breaks=lograngе_123456789)+
  scale_y_log10(
    breaks=lograngе_12357,
    minor_breaks=lograngе_123456789)
```




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



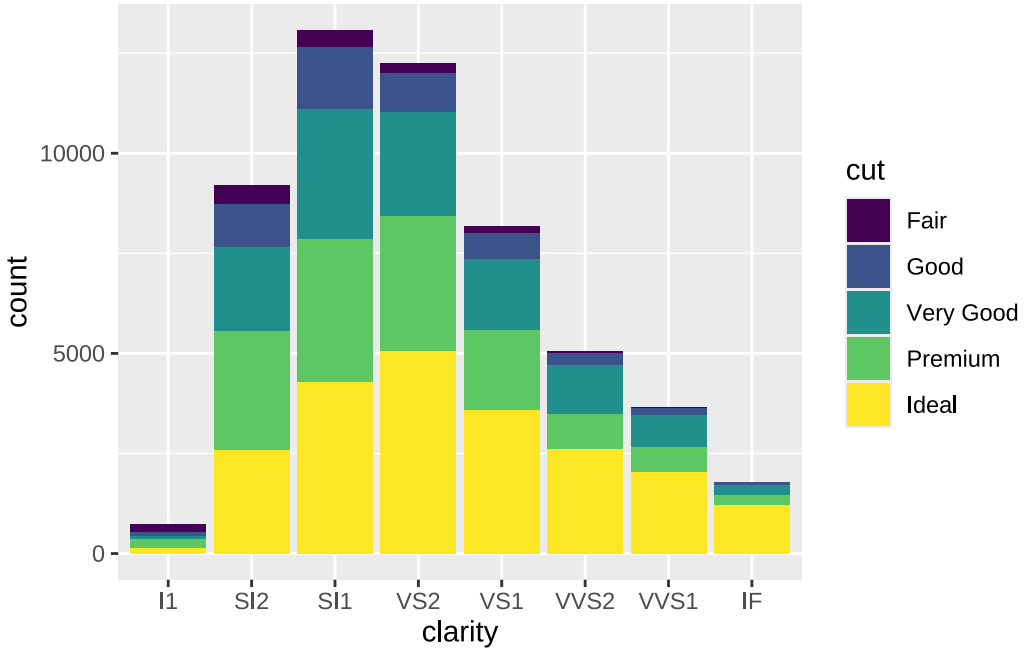

```
ggplot(data=mtcars,aes(wt, mpg, size=qsec))+
  geom_text(family="EmojiOne",label="\U1F697")+
  scale_size_continuous(range = c(5,10))
```



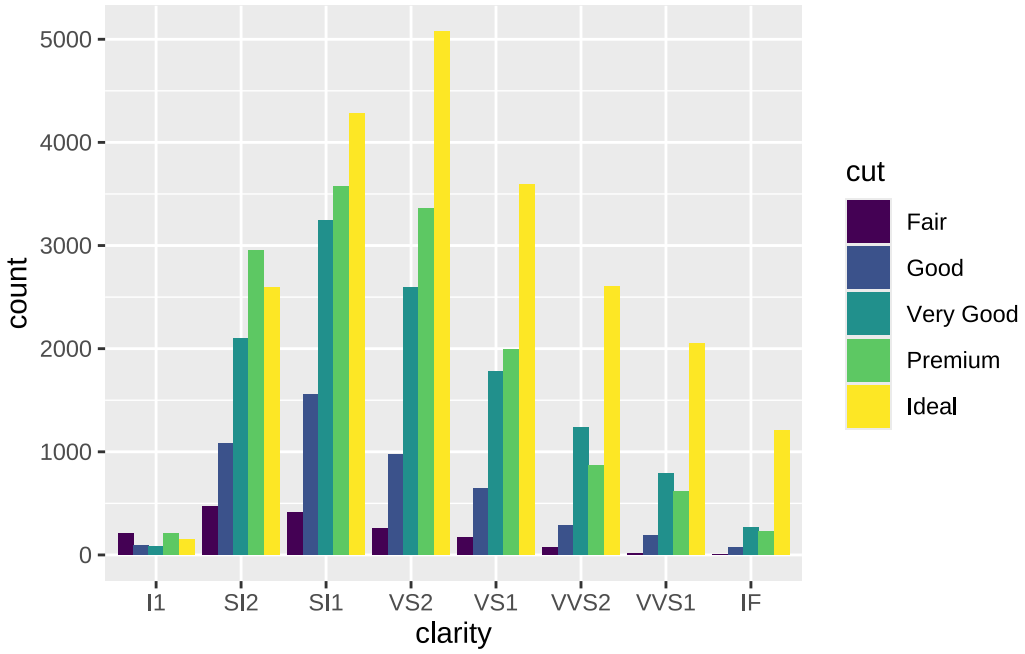
7.8 Positioning elements

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.

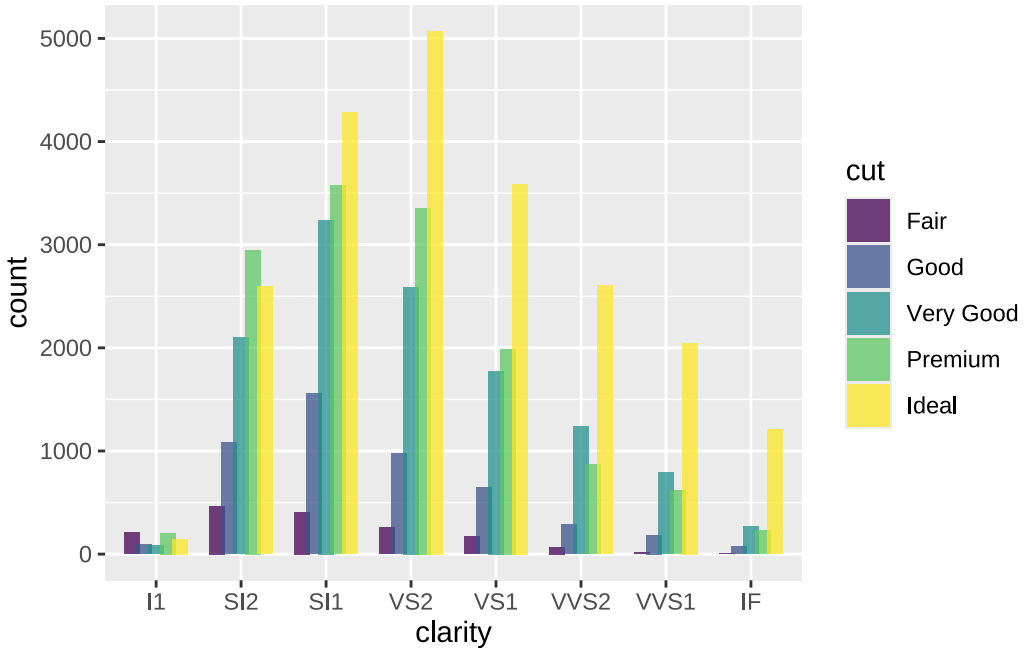
```
p<-ggplot(data=diamonds,aes(clarity,fill=cut))
p+geom_bar(position="stack") # default for bar
```



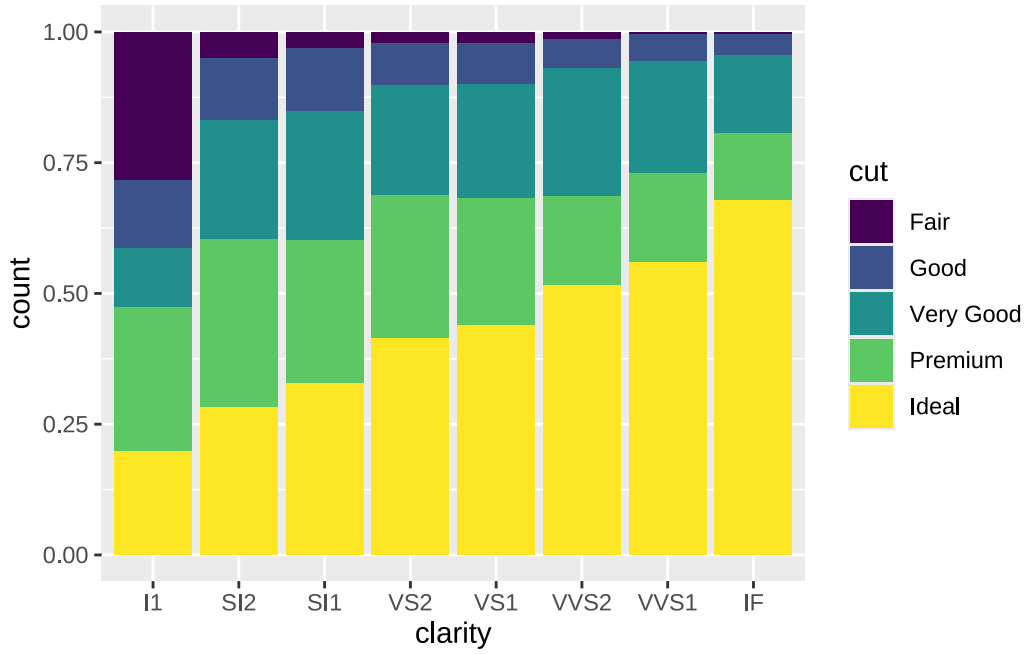

```
p+geom_bar(position="dodge")
```



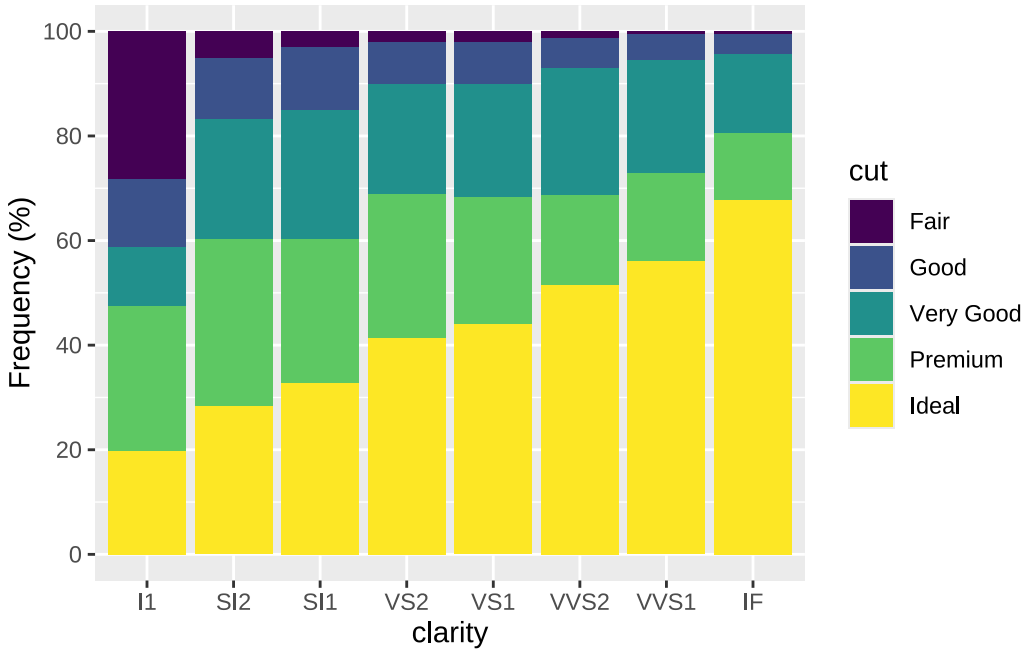

```
p+geom_bar(position=position_dodge(width = 0.7), alpha=.75)
```



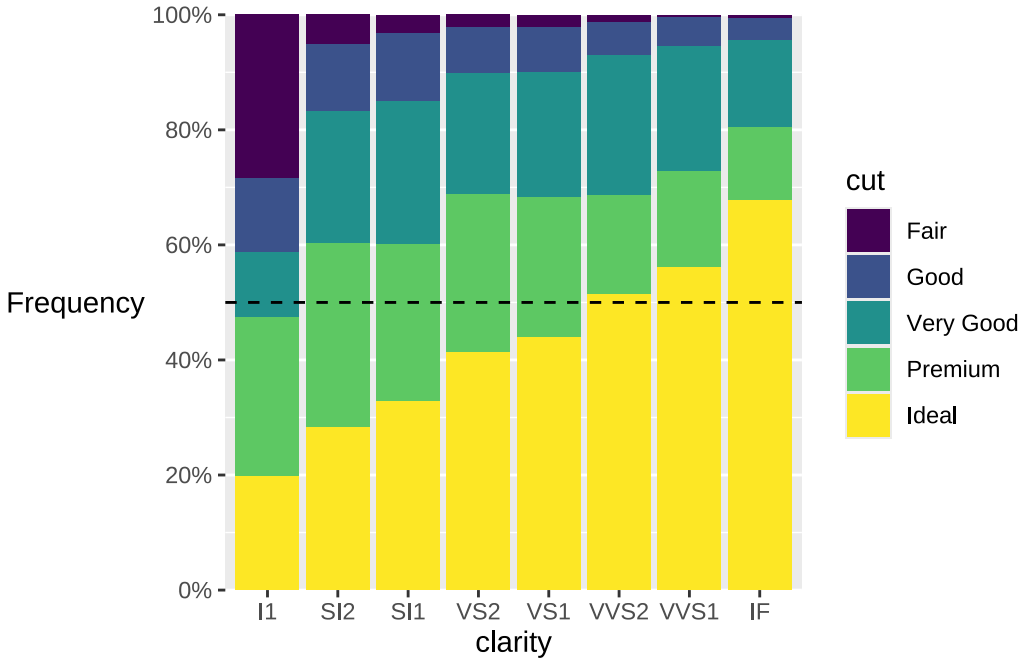

```
p+geom_bar(position="fill") #y-axis labeling needs tuning
```



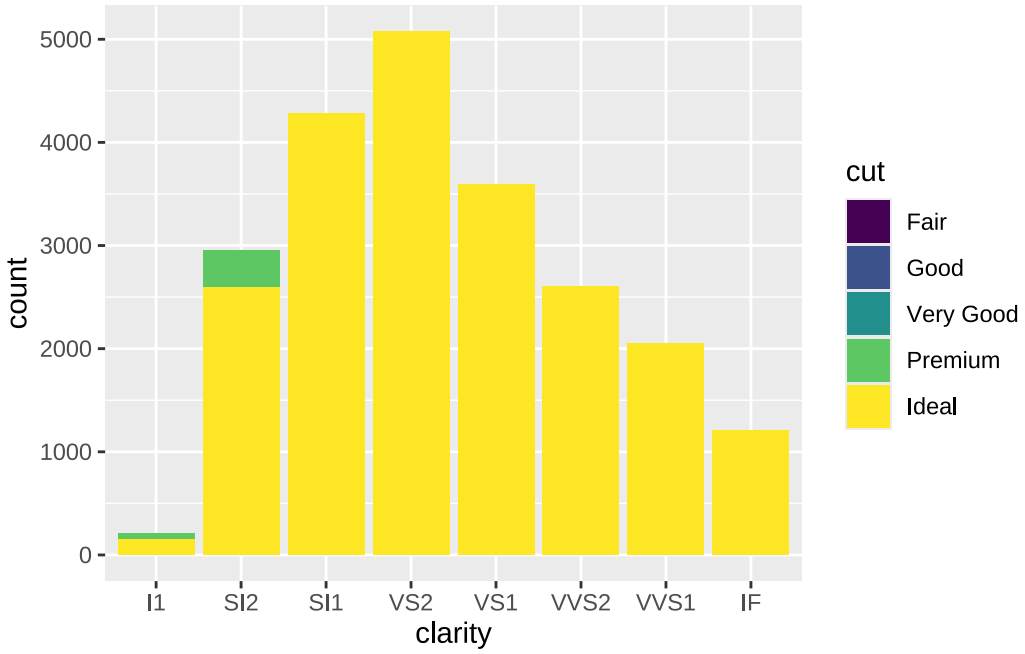

```
p+geom_bar(position="fill")+
  scale_y_continuous(name = "Frequency (%)",
                     breaks=seq(0,1,.2), #steps
                     labels=seq(0,100,20))
```



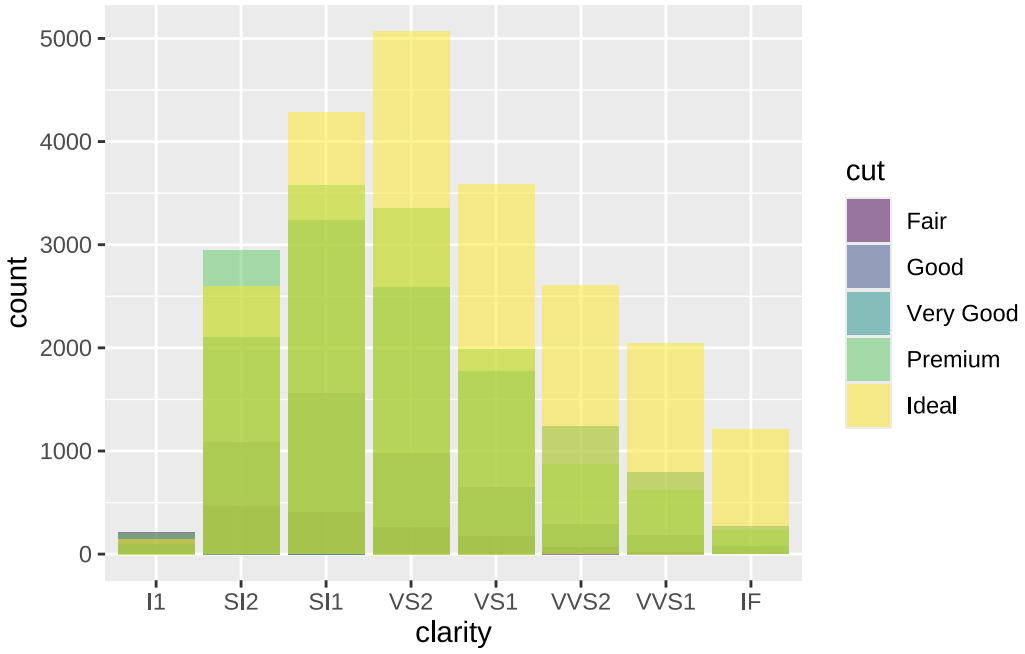
```
p+geom_bar(position="fill")+
  scale_y_continuous("Frequency",
                     breaks=seq(0,1,.2),
                     labels=scales::percent,
                     expand=expansion(mult = c(0,0)))+
  theme(axis.title.y = element_text(angle = 0,
                                    vjust = .5))+
  geom_hline(yintercept = .5, linetype=2) # e.g. for reference lines
```




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

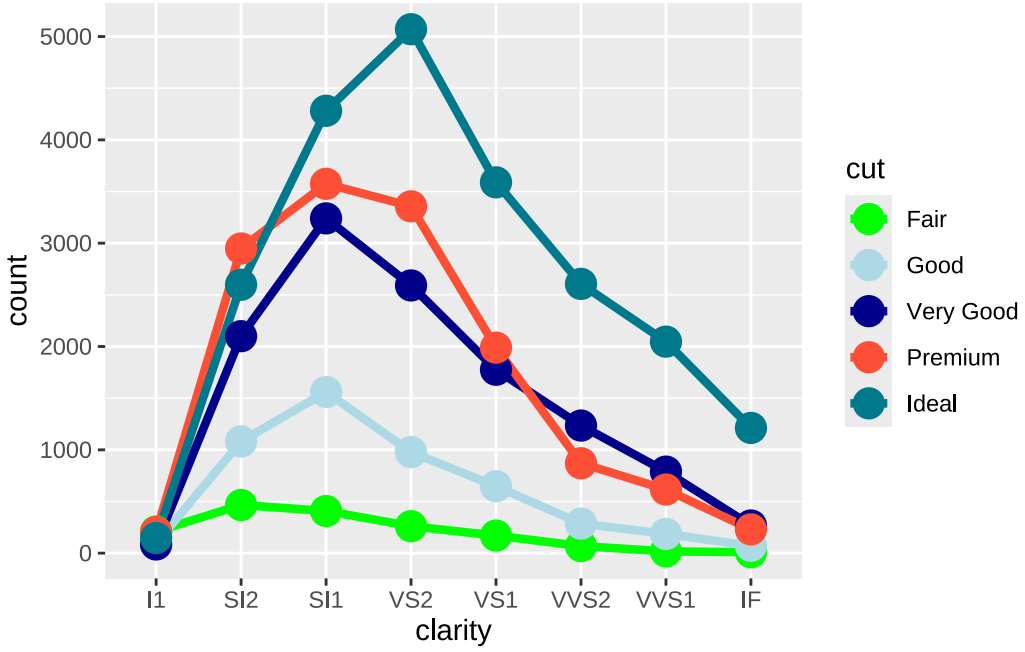



```
p+geom_bar(position="identity",alpha=.5) # even worse!!
```




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

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

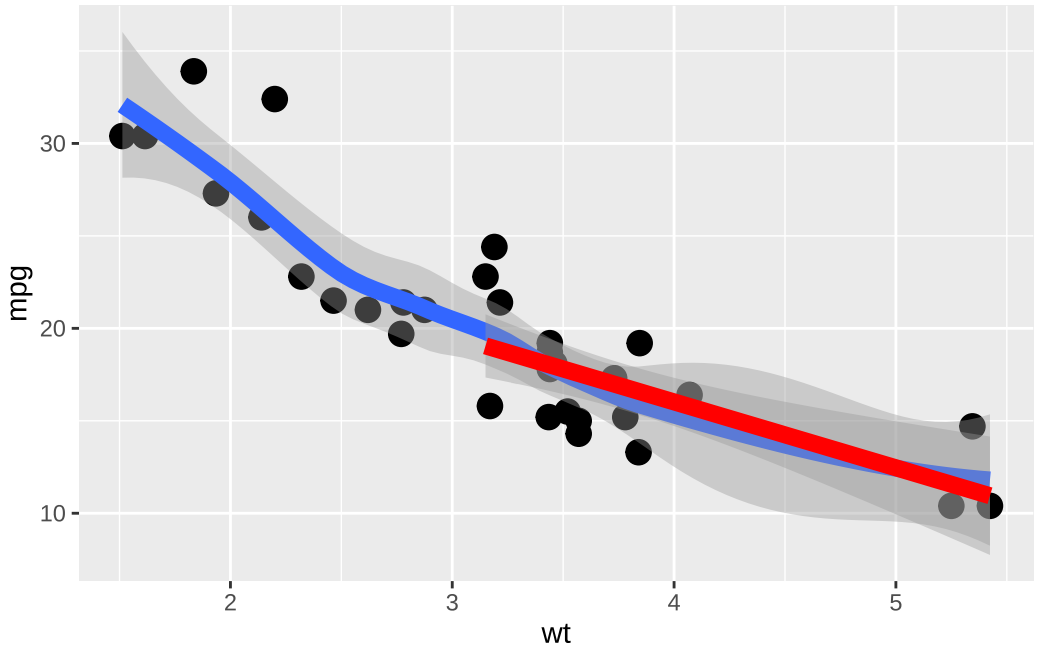


7.9 Order of layers

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

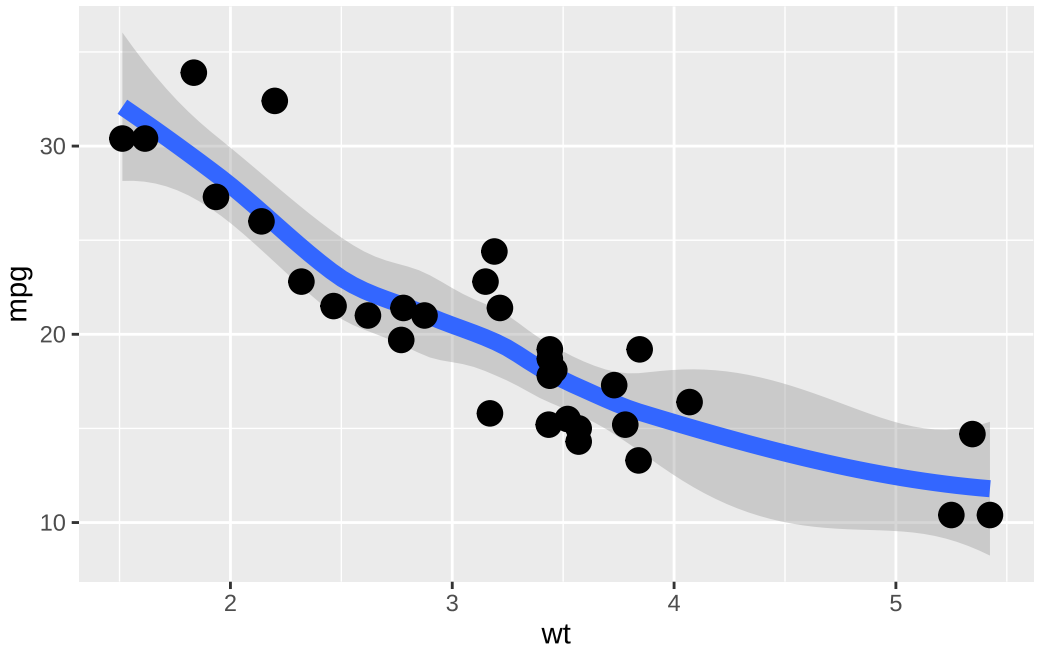
```
ggplot(data=mtcars,aes(wt, mpg))+
  geom_point(size=4)+
  geom_smooth(linewidth=3)+ # line overlaps points
  geom_smooth(data=mtcars |> filter(wt>3), #picks a sub-sample
              method="lm",linewidth=3, color="red")
```

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



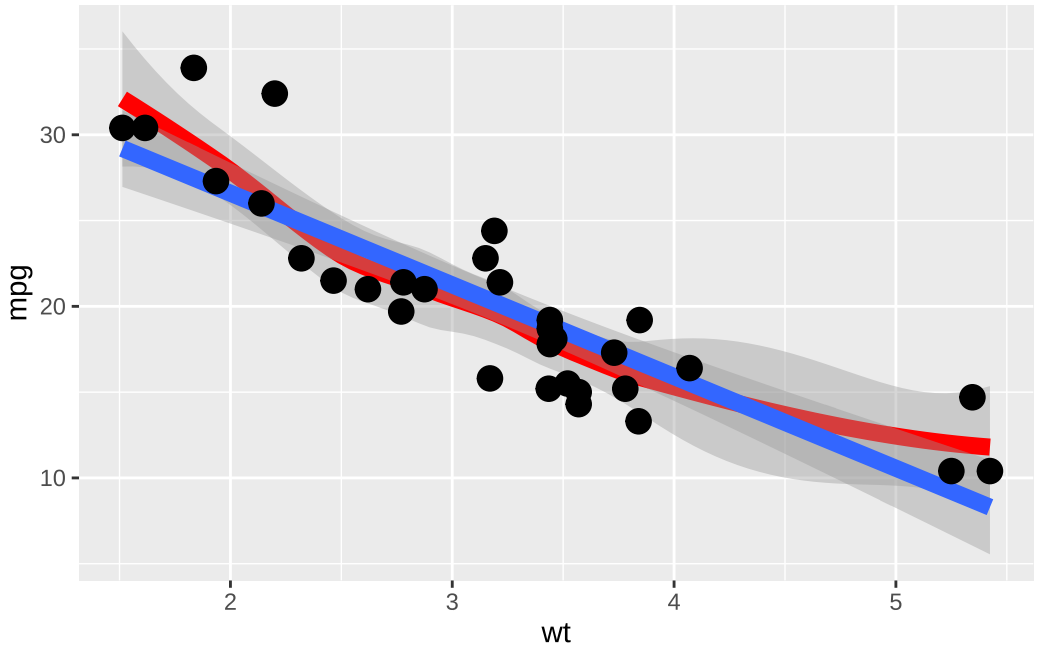

```
ggplot(data=mtcars,aes(wt, mpg))+
  geom_smooth(linewidth=3)+
  geom_point(size=4)
```

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



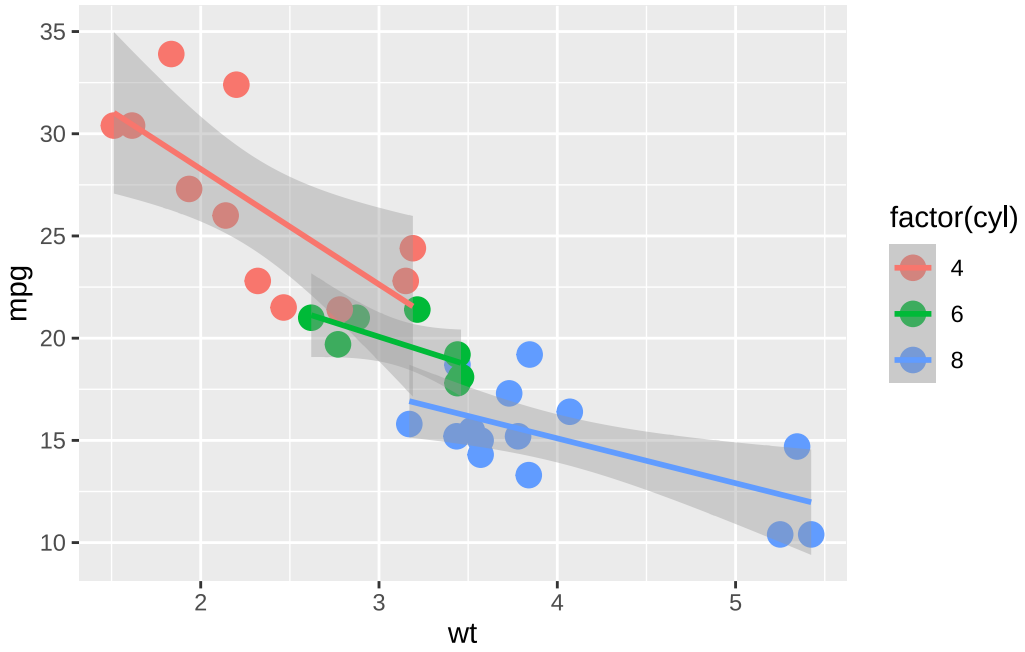
```
  ggplot(data=mtcars,aes(wt, mpg))+
    geom_smooth(linewidth=3,color="red")+
    geom_smooth(method="lm",linewidth=3)+
    geom_point(size=4)
```

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



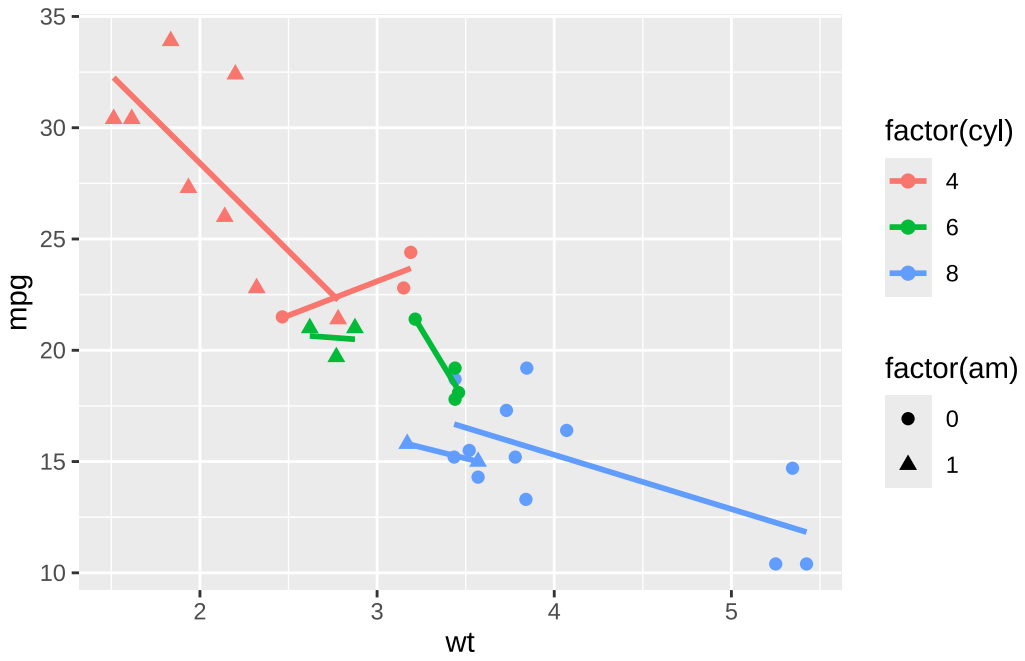

```
  ggplot(data=mtcars,aes(wt, mpg,
                         color=factor(cyl)))+
    geom_point(size=4)+
    geom_smooth(method="lm",linewidth=1)
```

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




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

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

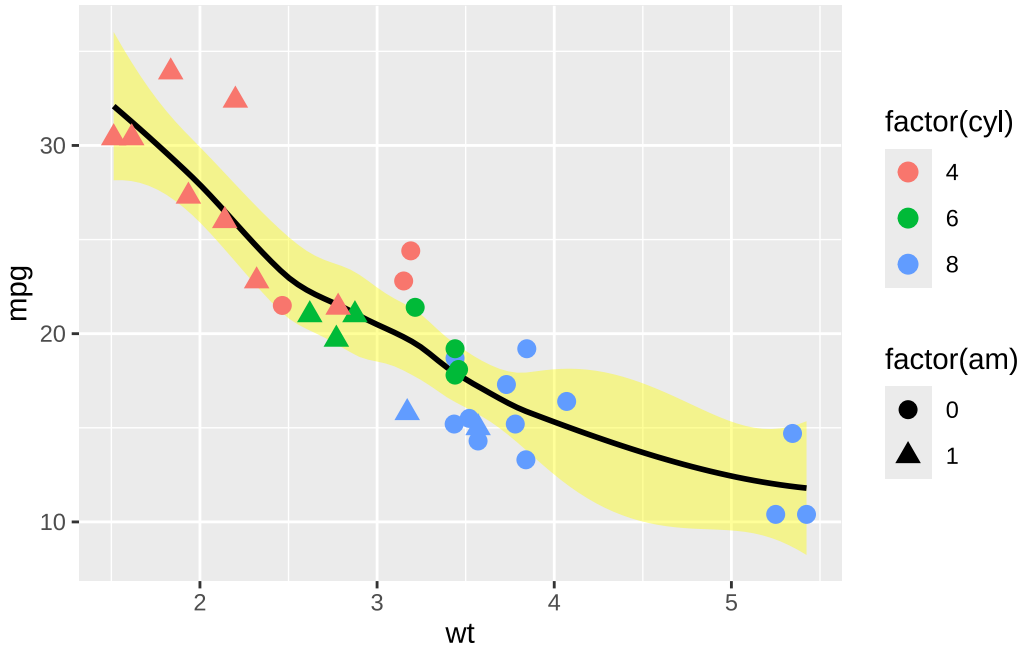


7.10 Local aesthetics for layers

```
#? 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
```

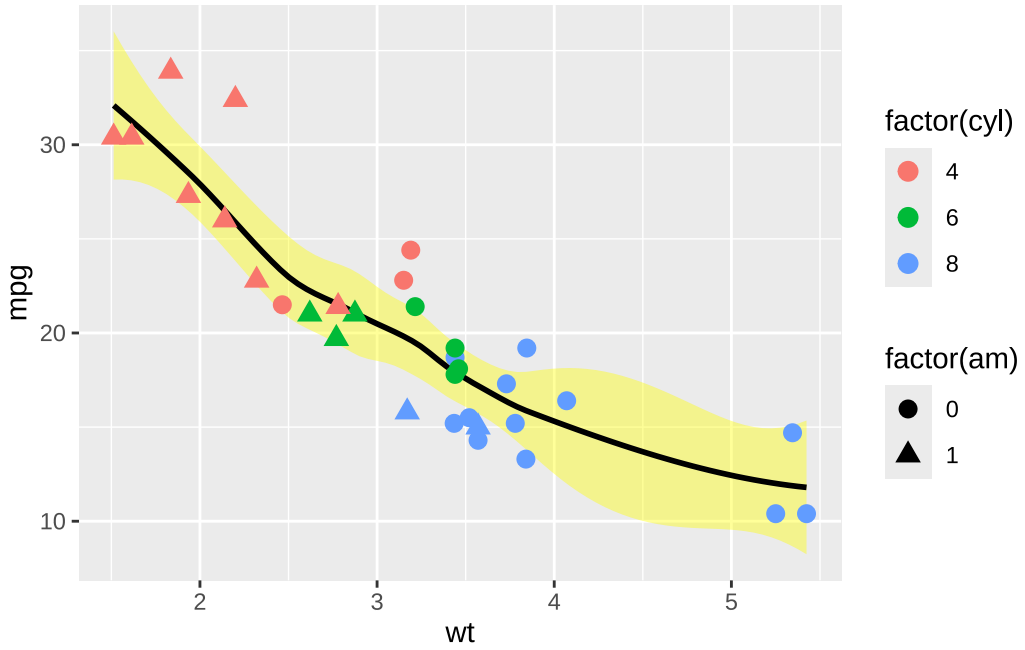
```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```

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




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

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



7.11 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).

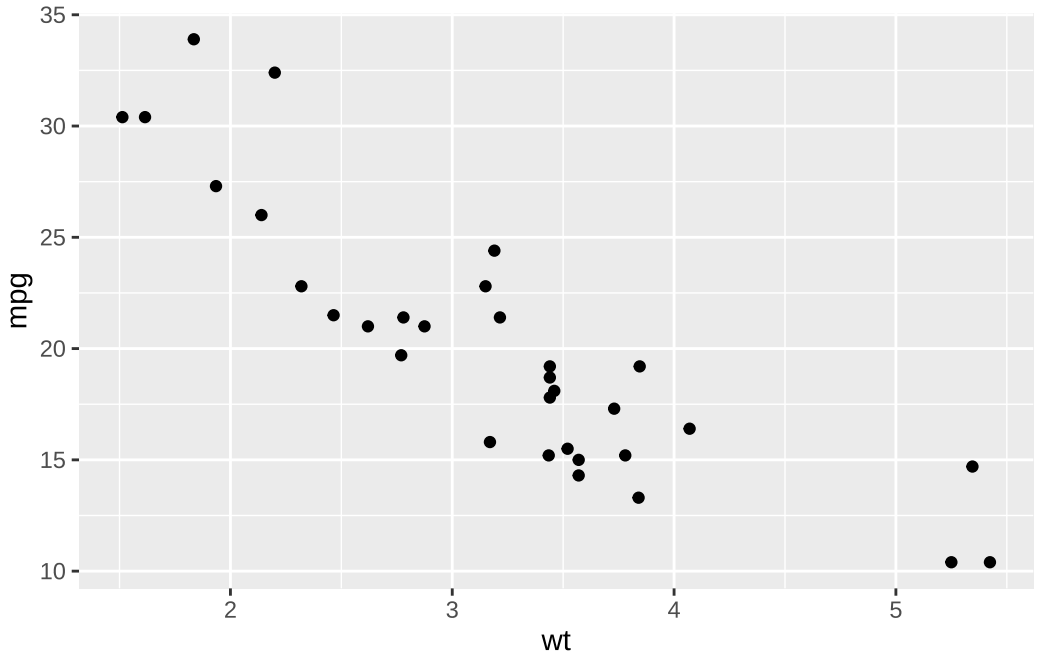
7.11.1 facet_grid

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

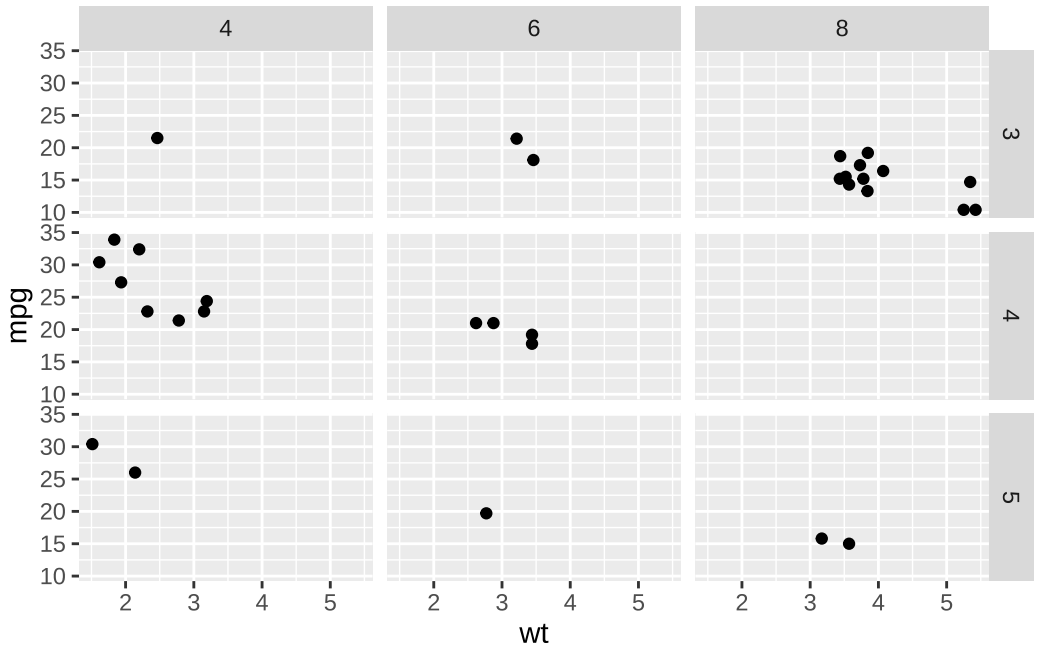
Labeling of facets often requires name and content to be informative.

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

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



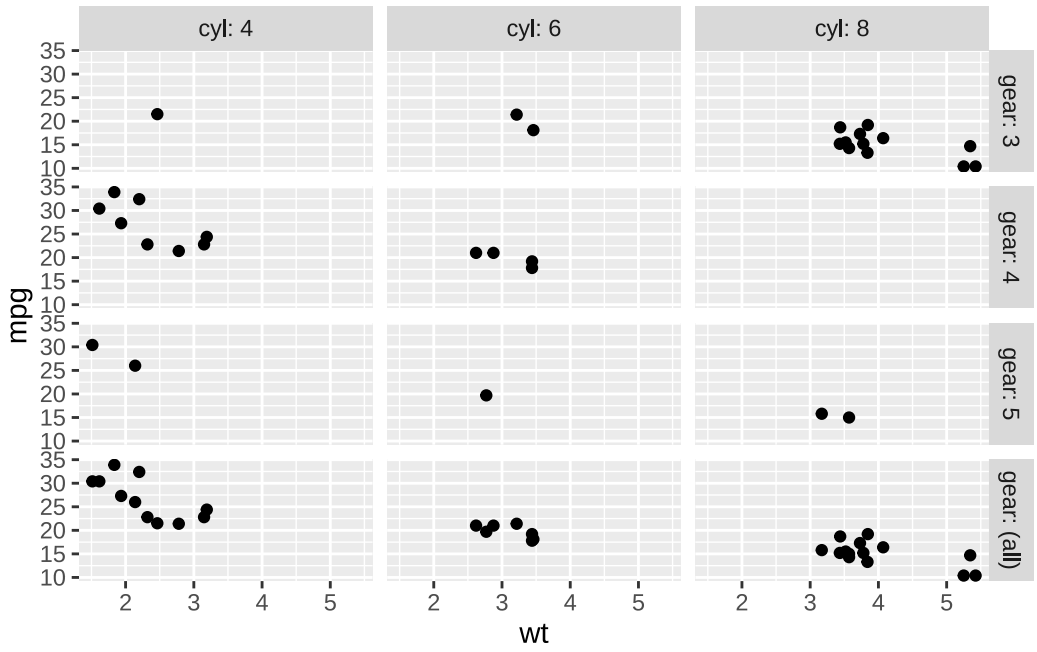

```
p.tmp + facet_grid(rows = vars(gear),
                   cols = vars(cyl))
```



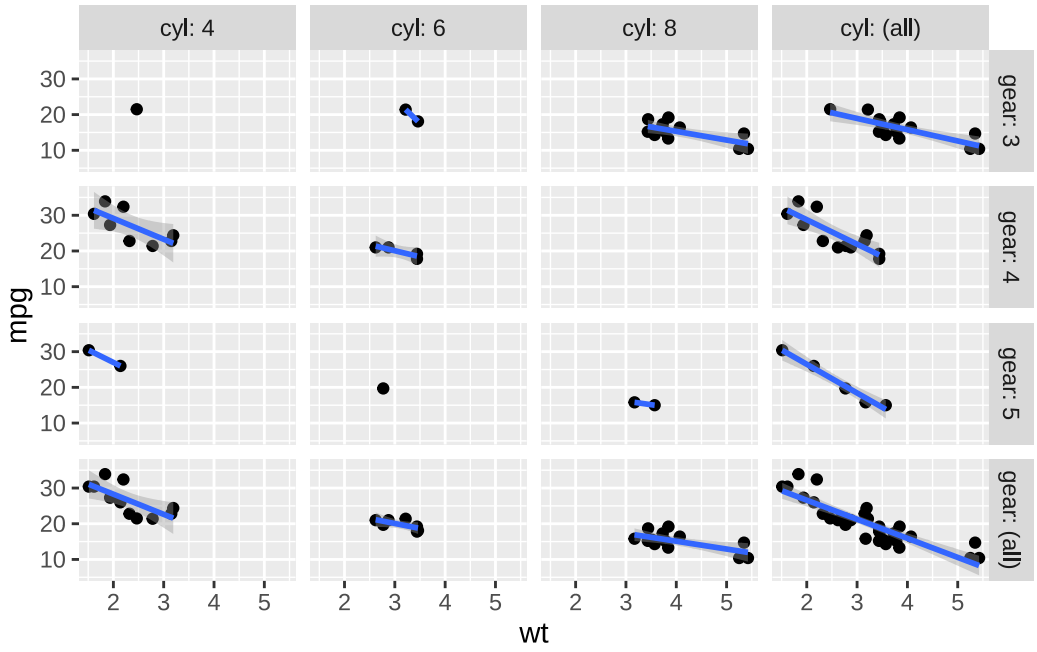

```
cat("facet labeling improved:\n")
```

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

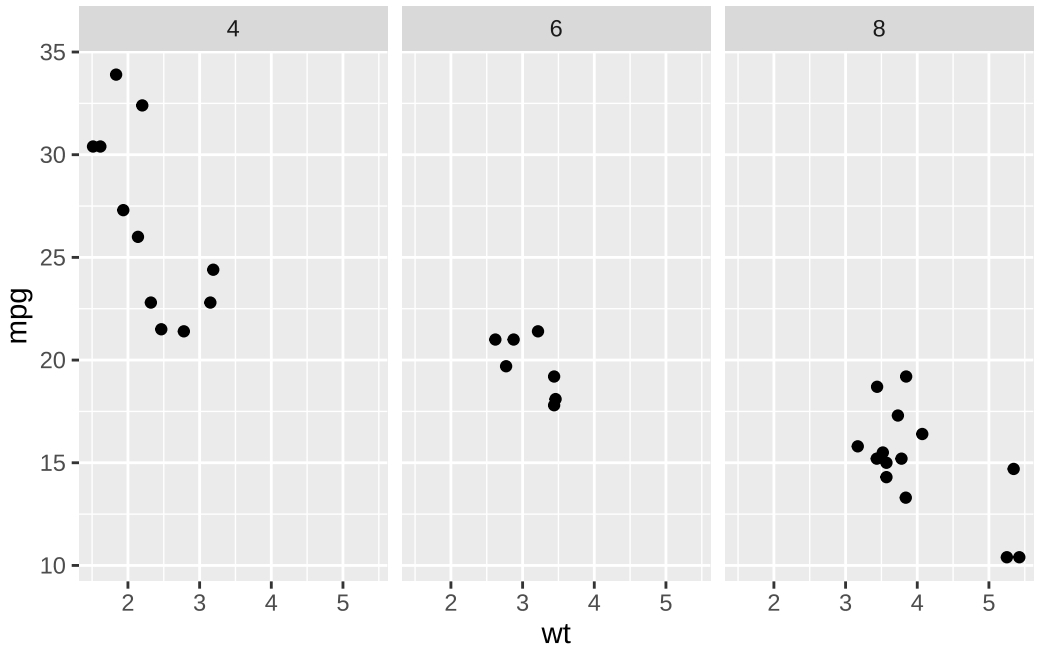


```
# options(warn=0)
```

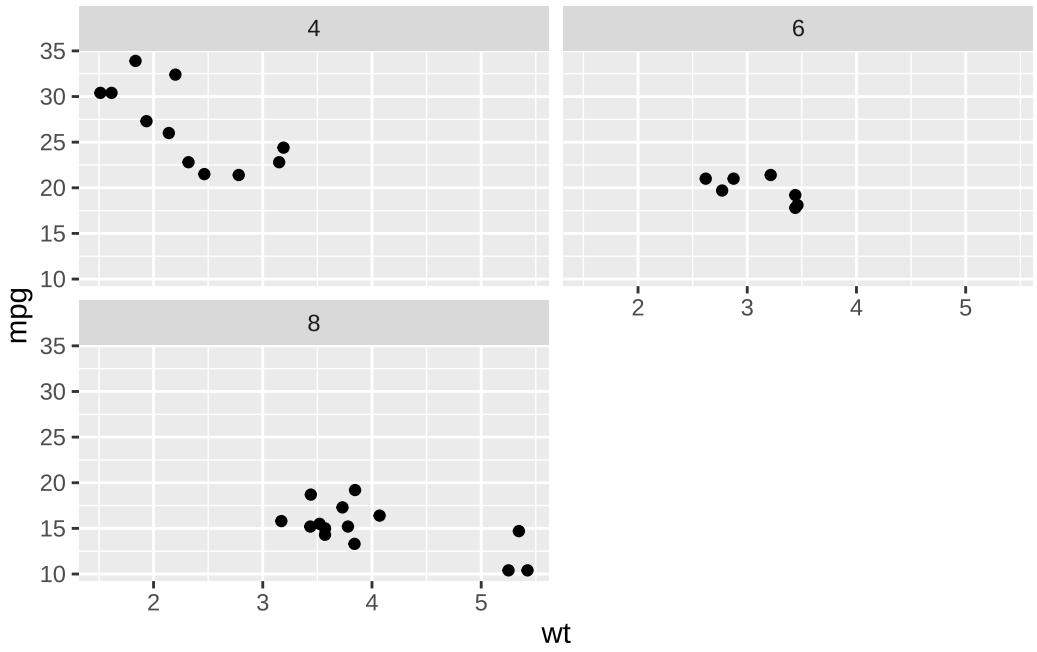
7.11.2 facet_wrap

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

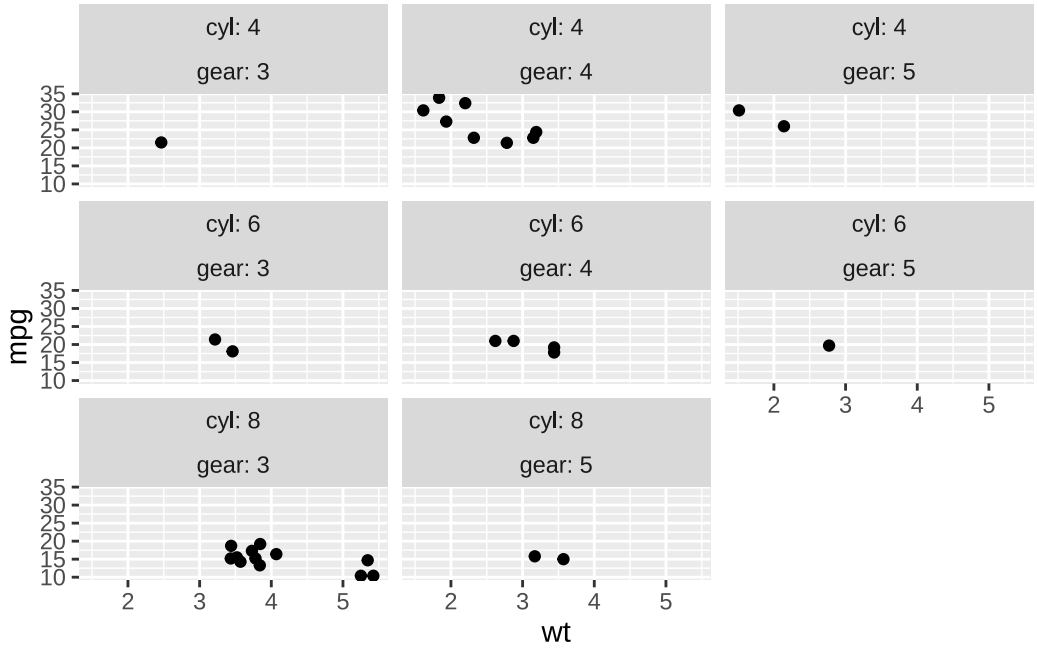
```
p.tmp + facet_wrap(facets = vars(cyl))
```




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

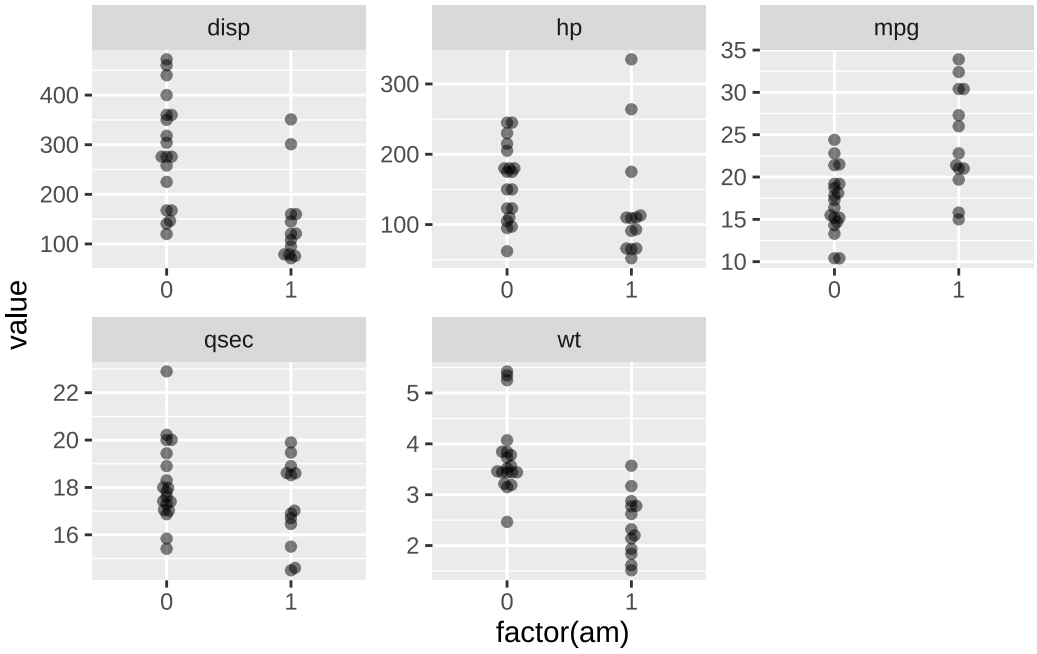



```
# empty combination is dropped
p.tmp + facet_wrap(facets=vars(cyl,gear),labeller=label_both)
```



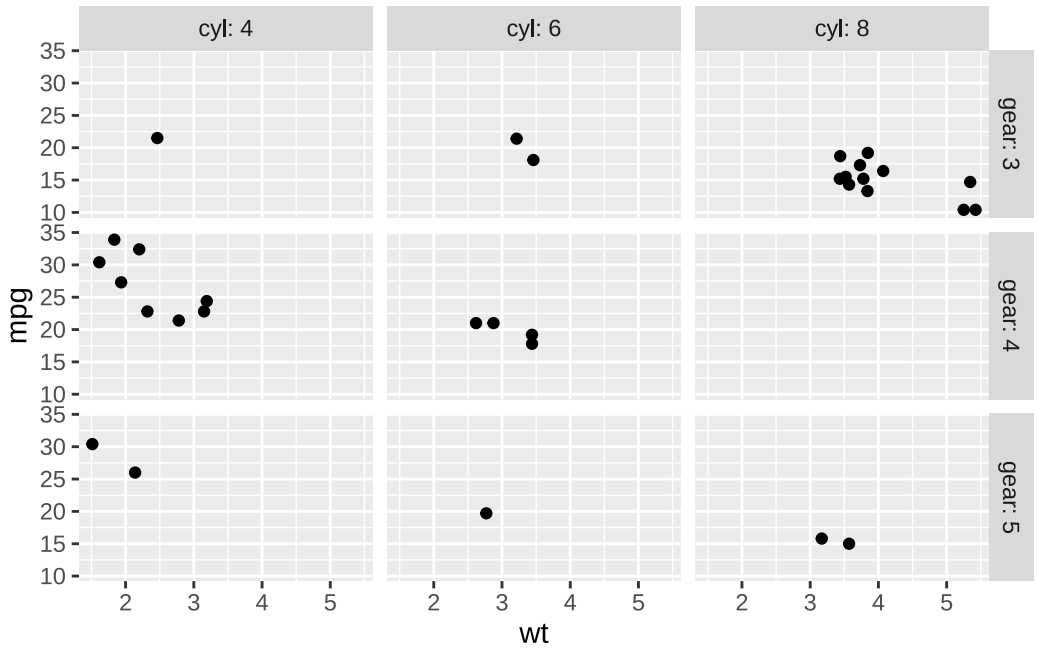

```
#combining variables
mtcars |>
```

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

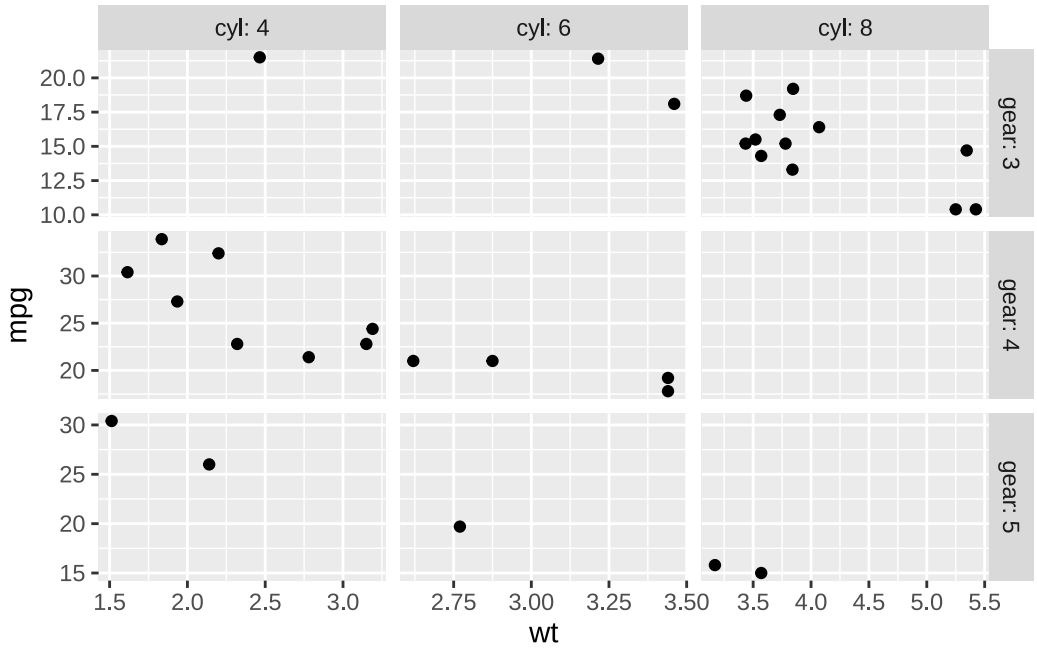


7.11.3 Controlling scales in facets (default: scales="fixed")

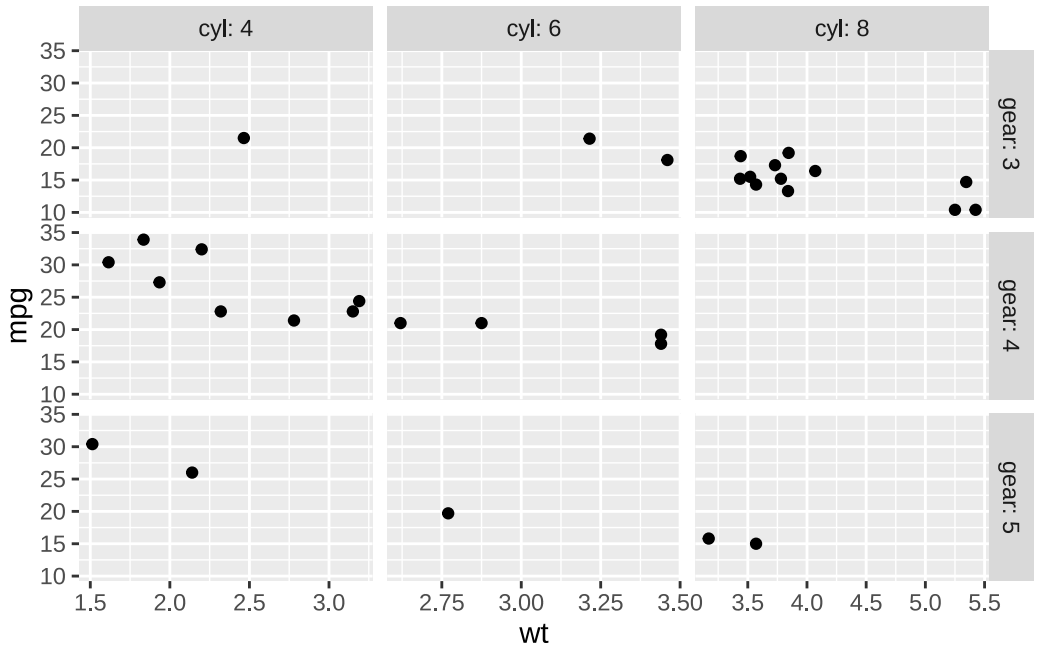
```
p.tmp + facet_grid(rows=vars(gear),cols=vars(cyl),
                   labeller=label_both, scales="fixed")
```



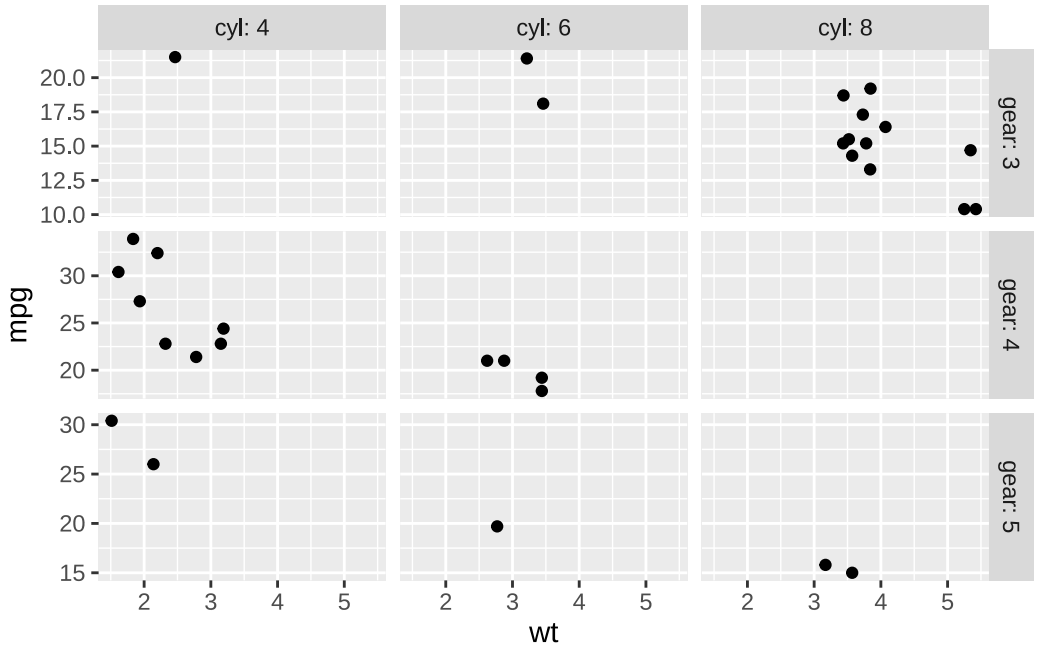

```
p.tmp + facet_grid(rows=vars(gear),cols=vars(cyl),
                   labeller=label_both, scales="free")
```




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




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

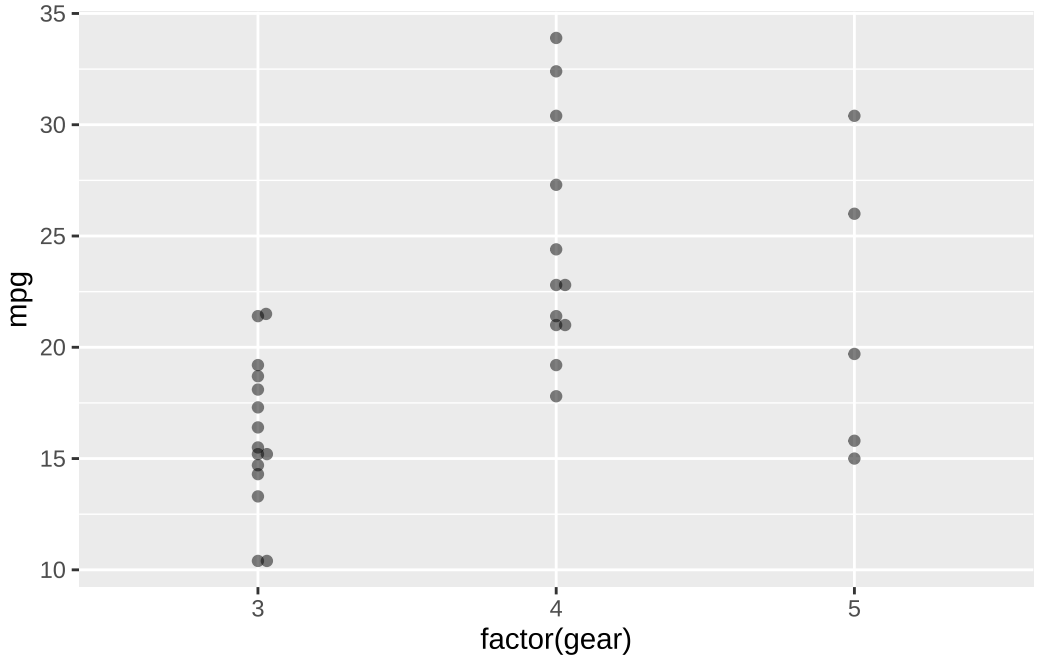


7.12 Showing summaries

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!!

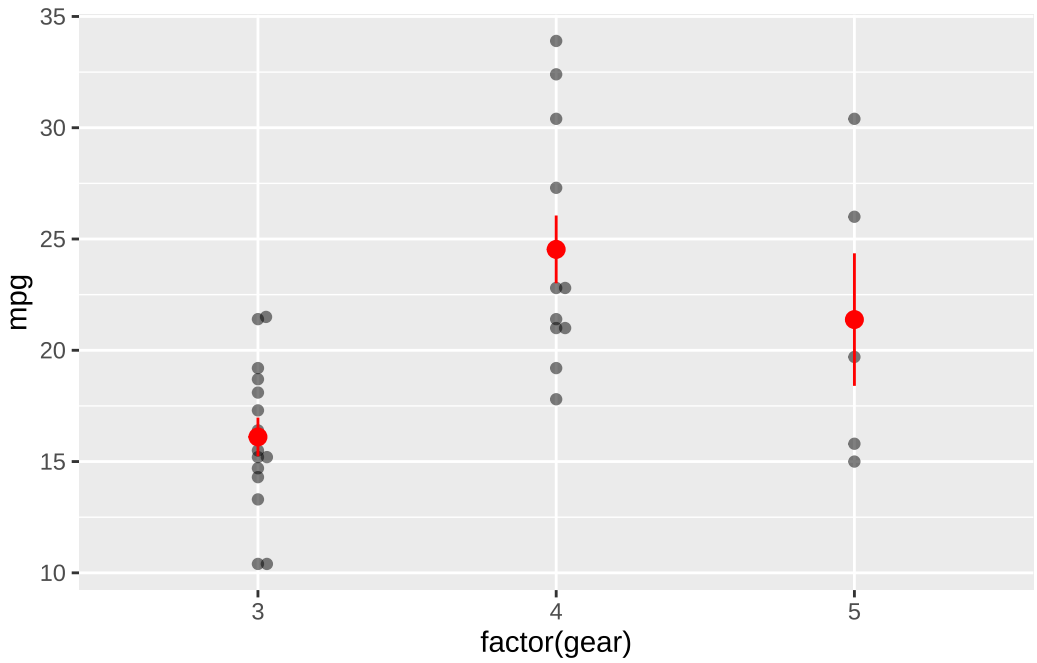
Functions for summary statistics (mean_se, mean_cl_normal, mean_cl_boot etc.) are build on top of Hmisc functions. So this package is needed but not automatically installed with ggplot2.

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

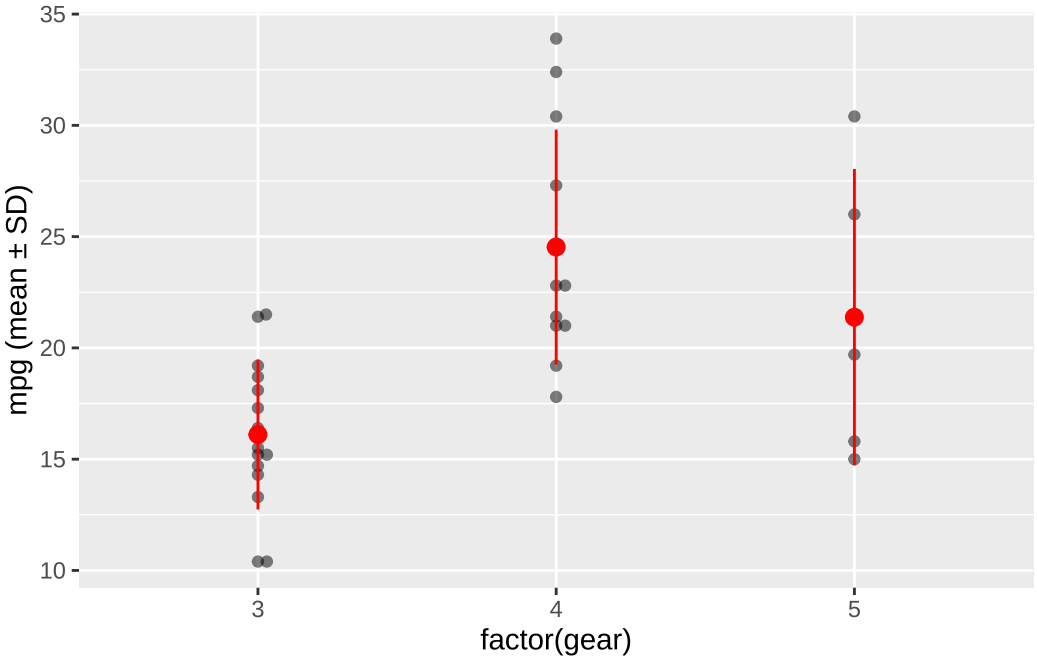



```
plottemp+stat_summary(color="red")
```

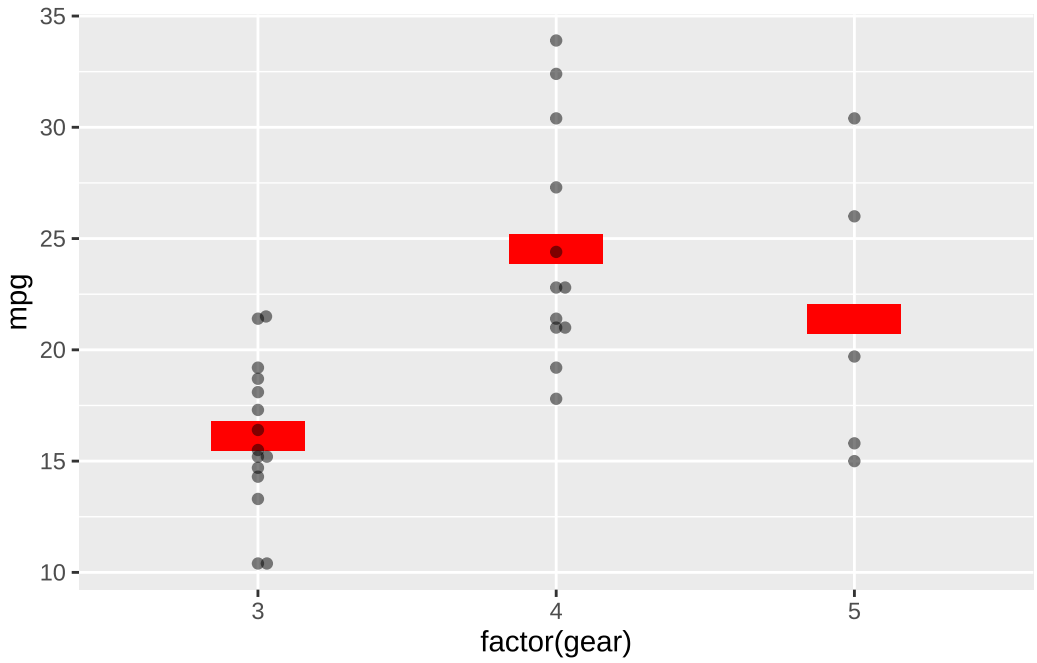
```
No summary function supplied, defaulting to `mean_se()`
```



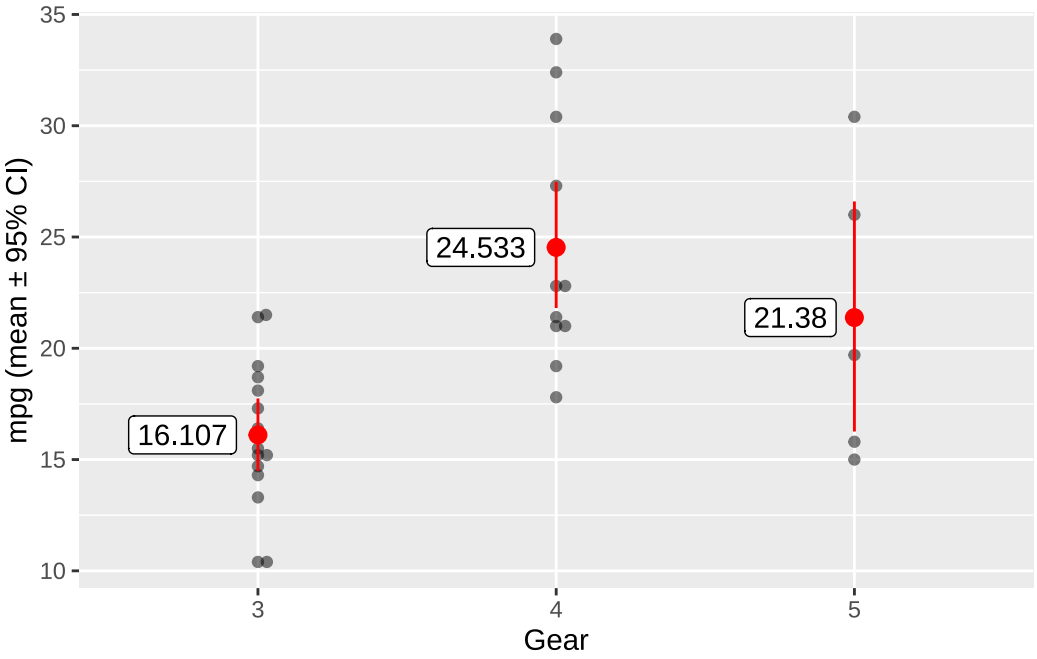

```
plottemp+stat_summary(fun.data="mean_sdl",
                      fun.args=list(mult=1),
                      color="red")+
  ylab("mpg (mean \u00b1 SD)")
```



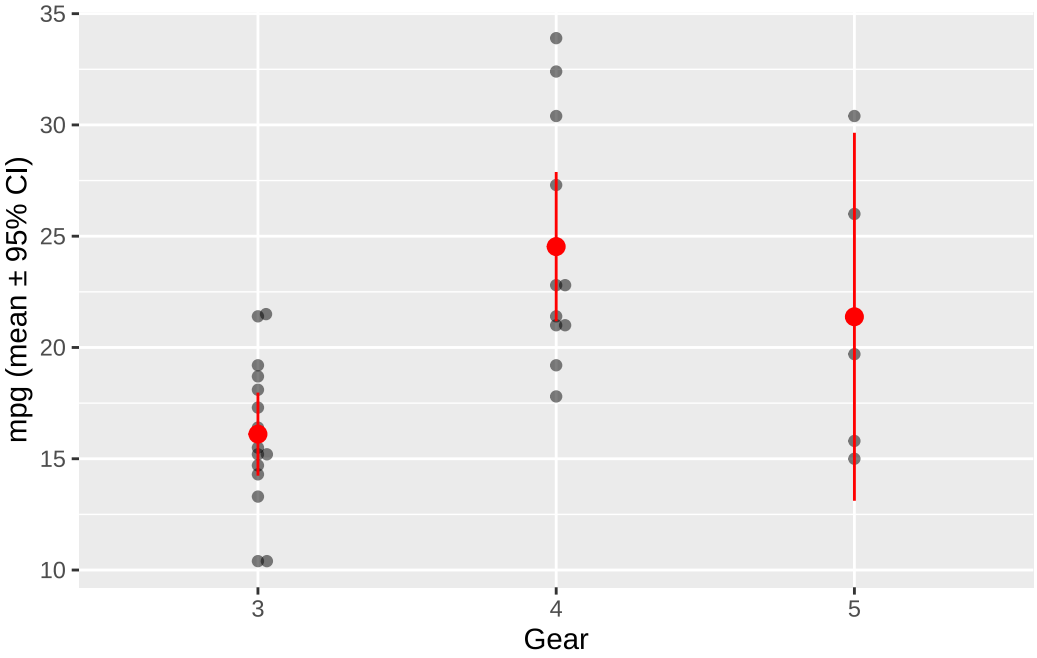
```
ggplot(mtcars,aes(factor(gear),mpg))+
  stat_summary(geom = "point", shape="-",
               size=50,
               fun = "mean",color="red")+
  geom_beeswarm(alpha=.5)
```



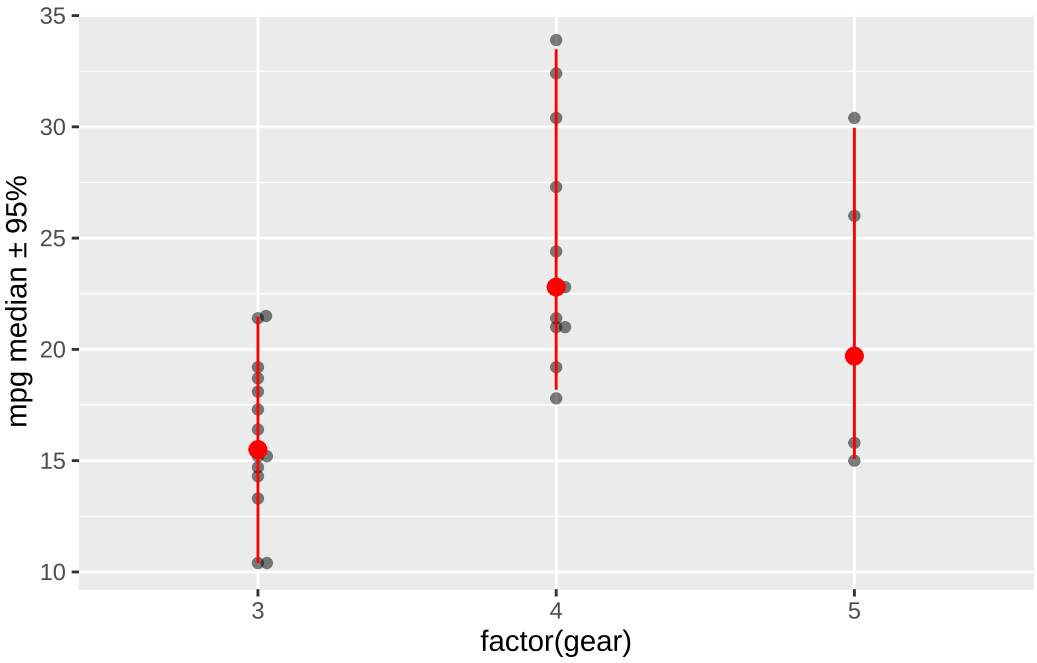

```
means <- mtcars |>
  group_by(gear) |>
  summarise(mean=round(mean(mpg),3),sd=sd(mpg))
plottemp+stat_summary(fun.data="mean_cl_boot",
                      fun.args=list(B=10^4),
                      color="red")+
  geom_label(data=means,
             aes(factor(gear),mean,label=mean),
        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



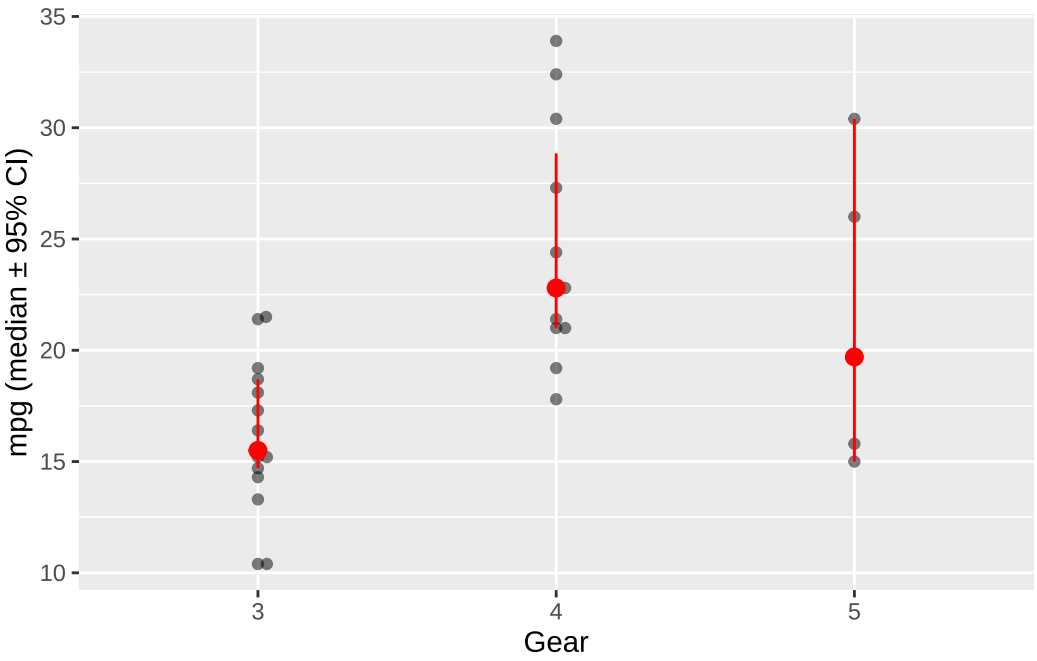

```
plottemp+stat_summary(fun.data="mean_cl_normal",color="red")+
  ylab("mpg (mean \u00b1 95% CI)")+
  xlab("Gear")
```



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




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



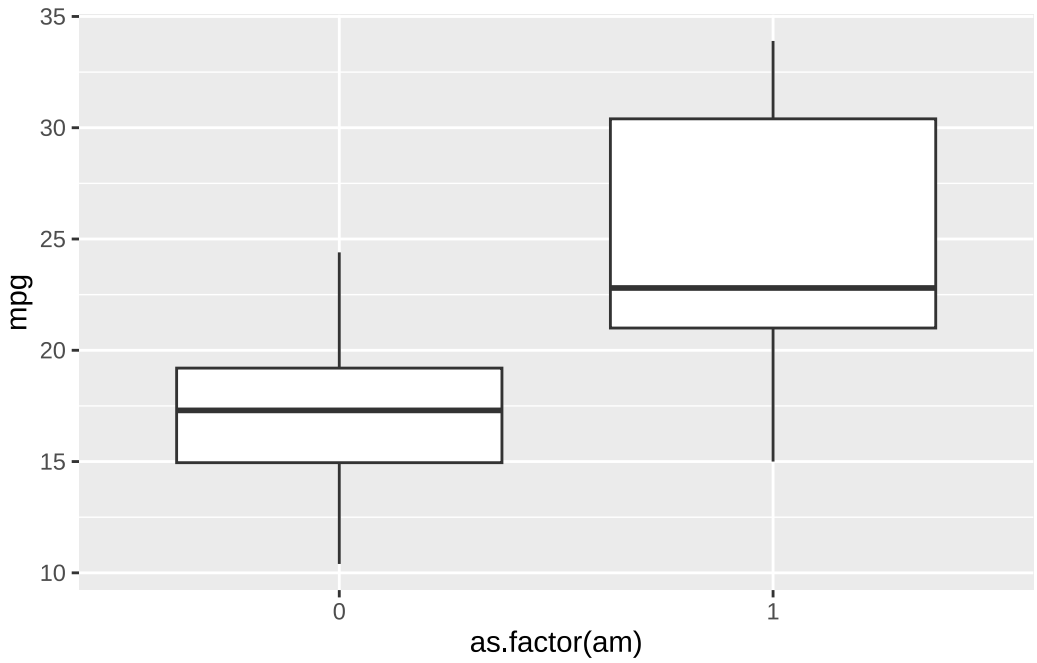
7.13 Indicating significances

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!).

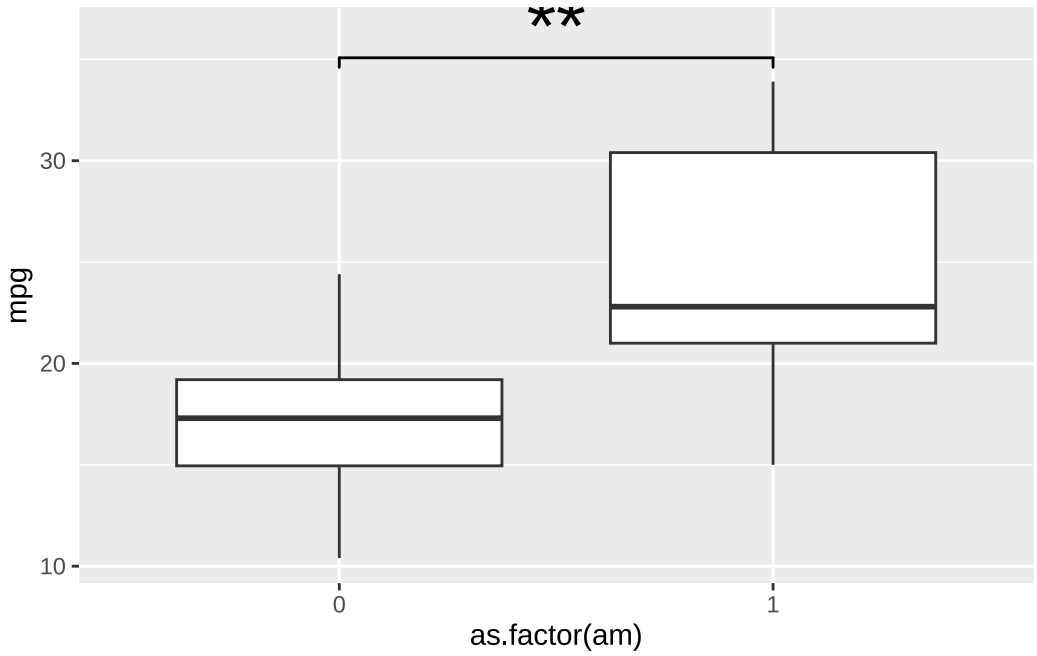
```
# ggsign ####
p <- round(
  wilcox.test(mtcars$mpg~mtcars$am)$p.value,
  5)
```

```
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())
```

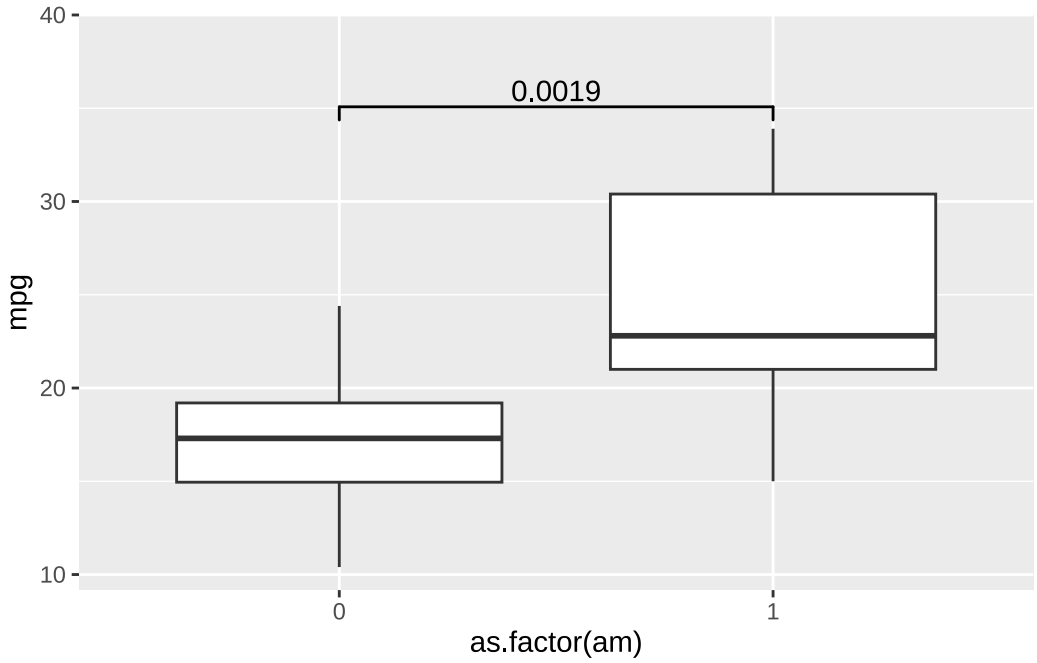



```
plottemp+geom_signif(
  comparisons=list(c(1,2)),
  # aes(y=0),
  textsize = rel(10), vjust = .0,
  #y_position=max(mtcars$mpg+3),
  # annotations=paste0("p = ", p),
  annotations=markSign(p),
  # annotations=p,
  tip_length=.02)+
  scale_y_continuous(expand = expansion(mult=c(0.05,.1)))
```

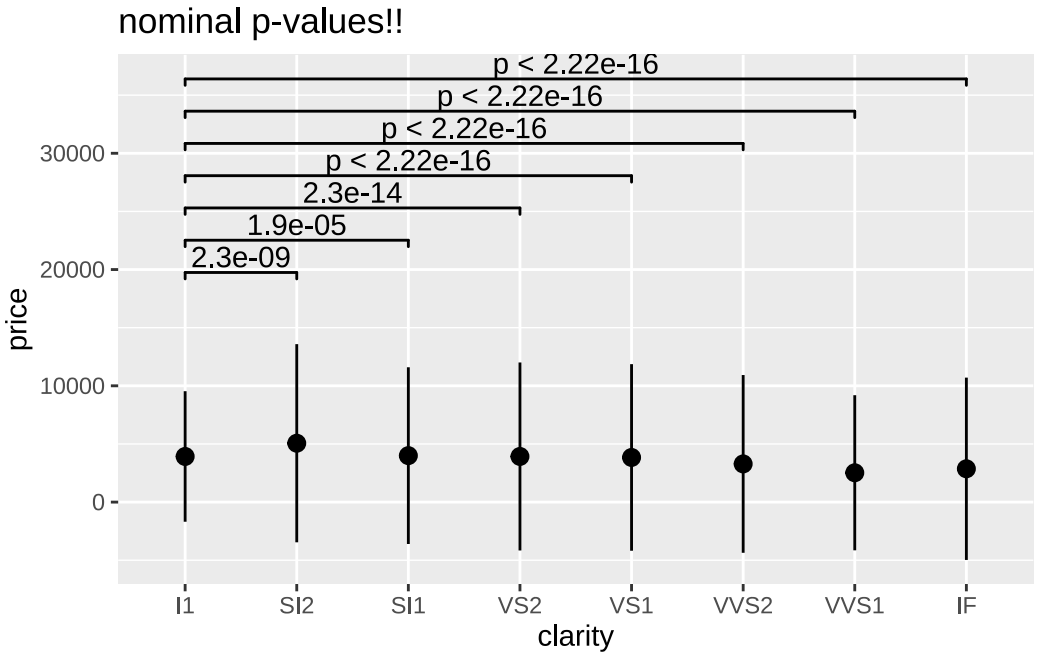



```
plottemp + geom_signif(
  comparisons=list(1:2))+
  scale_y_continuous(expand = expansion(mult=c(0.05,.2)))
```

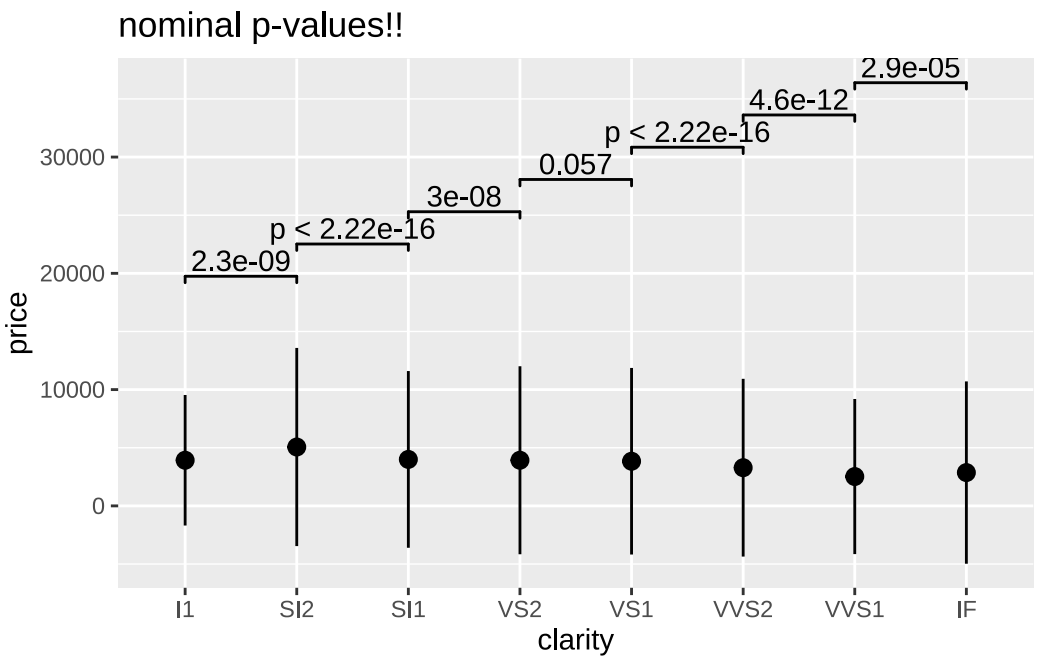
```
Warning in wilcox.test.default(c(21.4, 18.7, 18.1, 14.3, 24.4, 22.8, 19.2, :
kann bei Bindungen keinen exakten p-Wert Berechnen
```



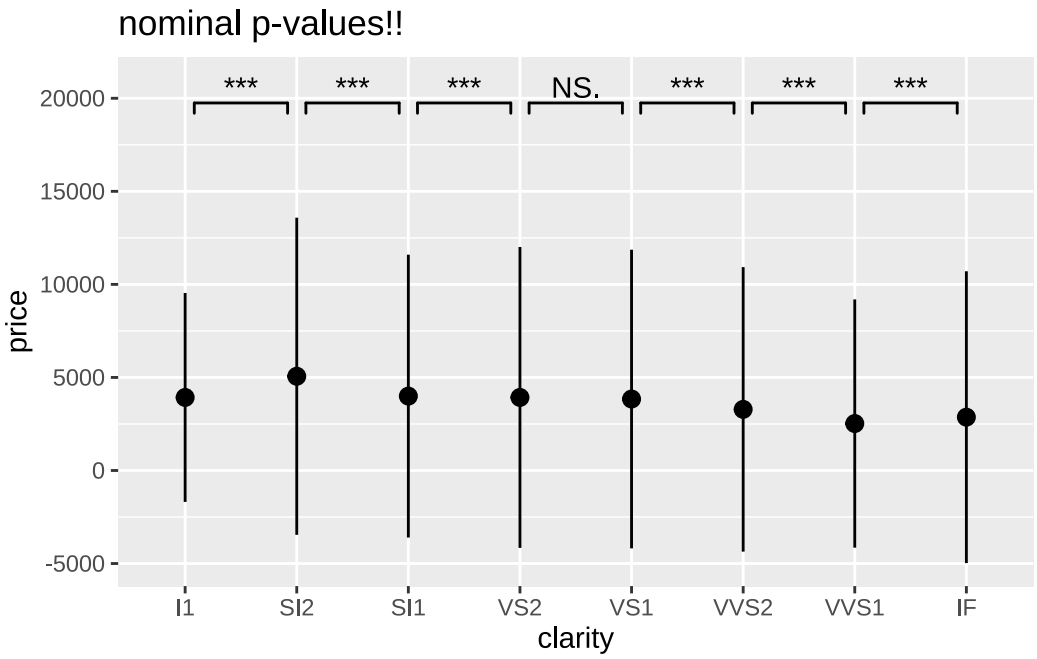

```
ggplot(diamonds,aes(clarity, price))+
  stat_summary(fun.data=mean_sdl)+
  geom_signif(comparisons=list(c(1,2),c(1,3),c(1,4),c(1,5),
                               c(1,6),c(1,7),c(1,8)),
              step_increase = 0.15)+
  ggtitle("nominal p-values!!")
```



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



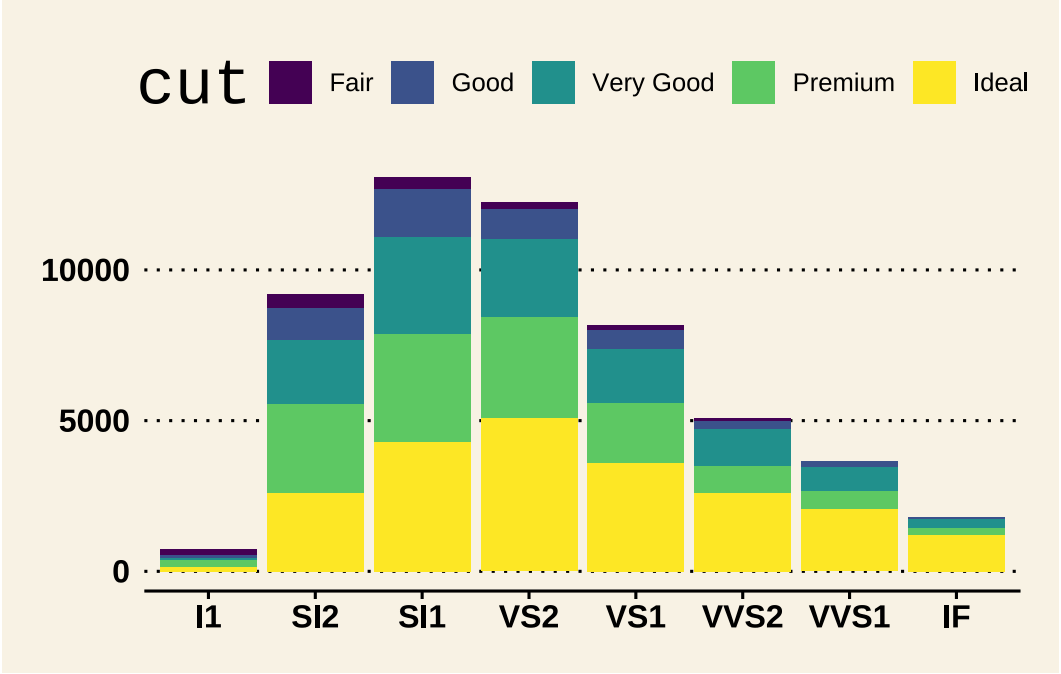

```
ggplot(diamonds,aes(clarity, price))+
  stat_summary(fun.data=mean_sdl)+
  geom_signif(comparisons=list(c(1,2),c(2,3),c(3,4),c(4,5),
                               c(5,6),c(6,7),c(7,8)),
              map_signif_level = TRUE,
              extend_line = -.01)+
  ggtitle("nominal p-values!!")+
  scale_y_continuous(expand = expansion(mult=c(0.05,.1)))
```



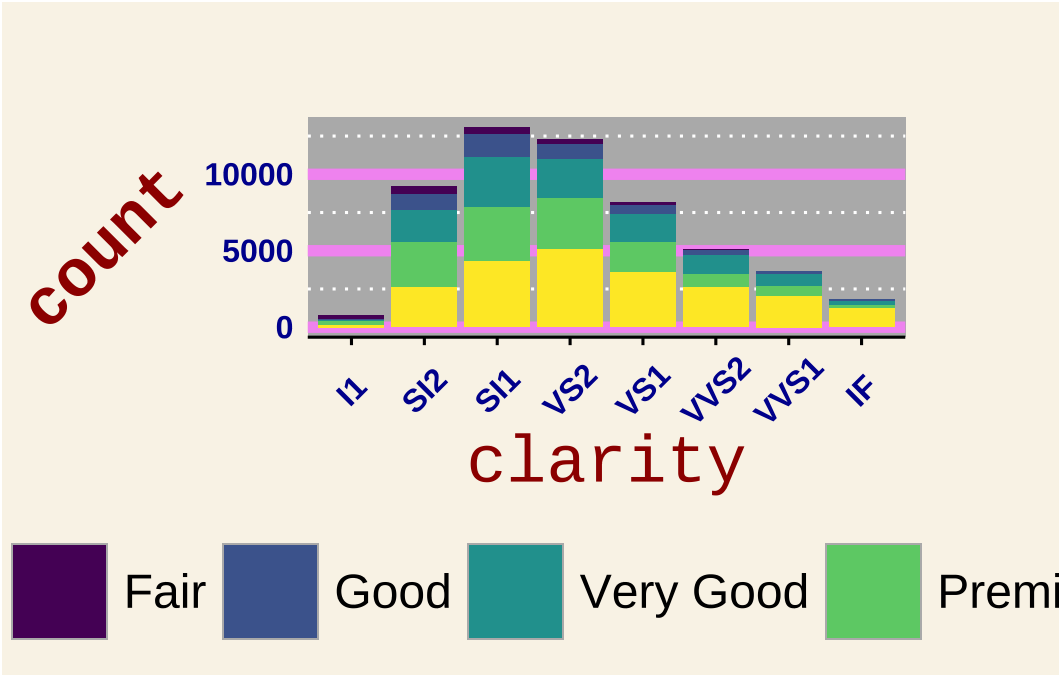
7.14 Theme definitions / changes

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

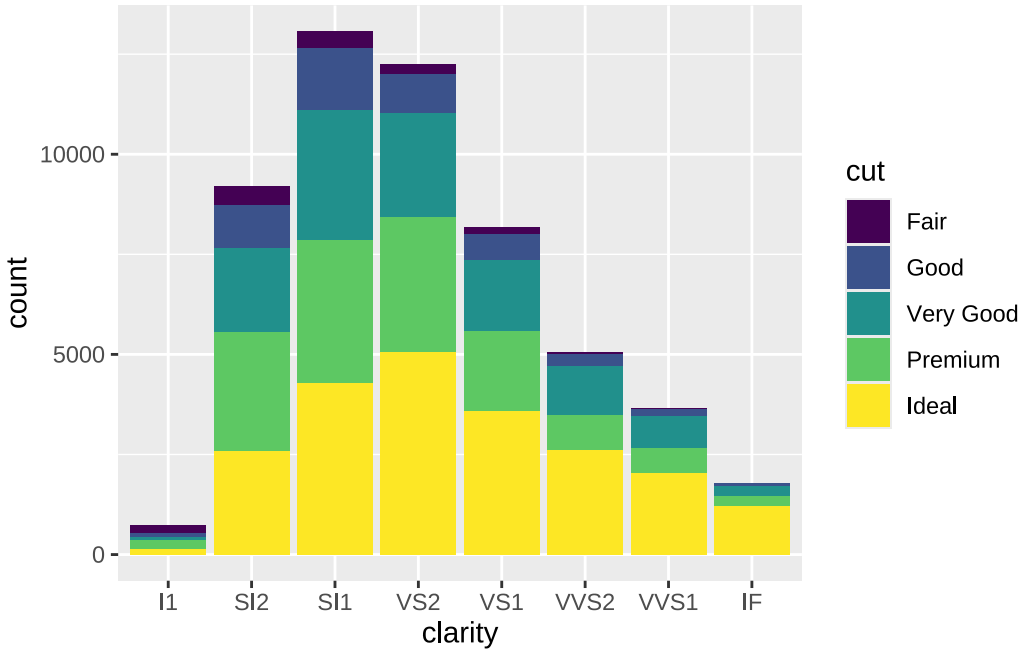
```
old <- theme_set(theme_wsj())
ggplot(data=diamonds,aes(x=clarity,fill=cut))+
  geom_bar()
```



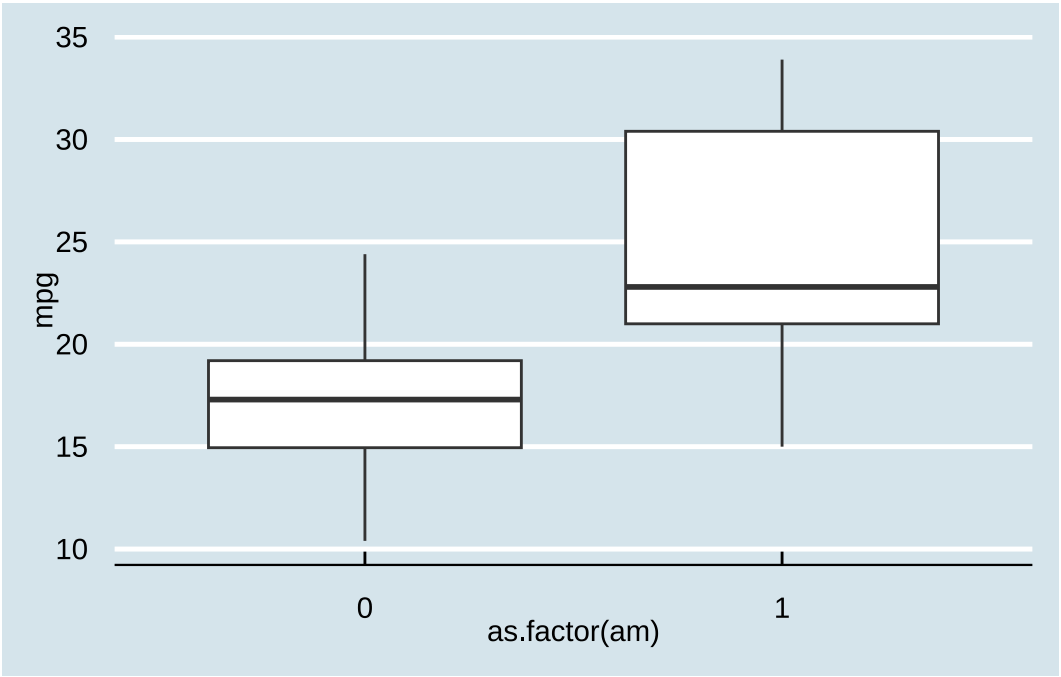

```
theme_update(legend.position="bottom",
             axis.text=element_text(colour = "darkblue",
                                    size=12),
             axis.text.x=element_text(vjust=0.5,angle=45,
                                      family="sans",
                                      face = "bold"),
             axis.title=element_text(size=25,
                                     color="darkred"),
             plot.margin=unit(c(3,4,.5,.3),"lines"),    #N,E,S,W
             axis.title.y=element_text(vjust=0.4,angle=45,
                                       face="bold"),
             legend.key.size=unit(2.5, "lines"),
             panel.background=element_rect(fill="darkgrey"),
             panel.grid.minor = element_line(colour="white"),
             panel.grid.major = element_line(
               linetype=1,
               color="violet",linewidth = 2),
             legend.text = element_text(size = 18),
             legend.title=element_text(size=30, color="pink"))
ggplot(data=diamonds,aes(x=clarity,fill=cut))+
  geom_bar()
```



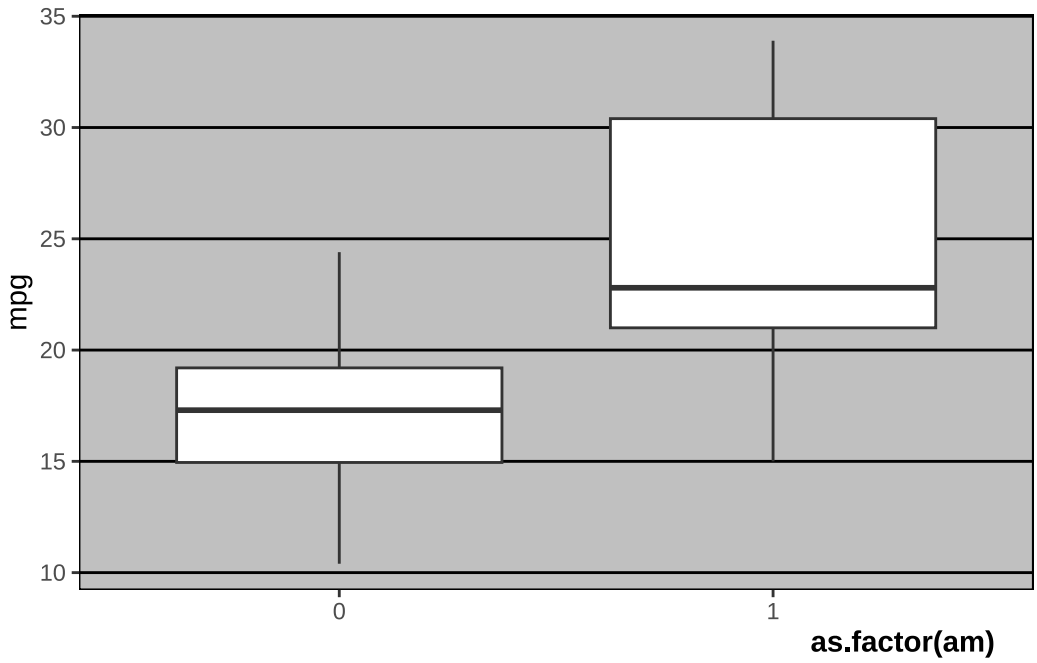

```
theme_set(theme_grey())
#theme_set(old)
ggplot(data=diamonds,aes(x=clarity,fill=cut))+
  geom_bar()
```



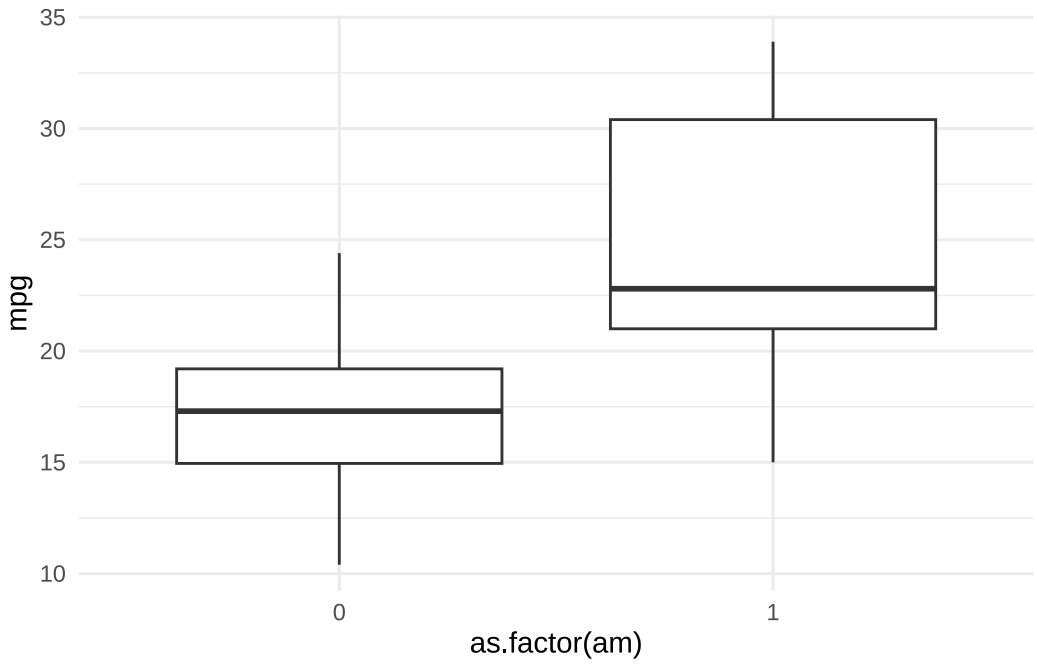
```
# ggthemes ####
plottemp+theme_economist()
```



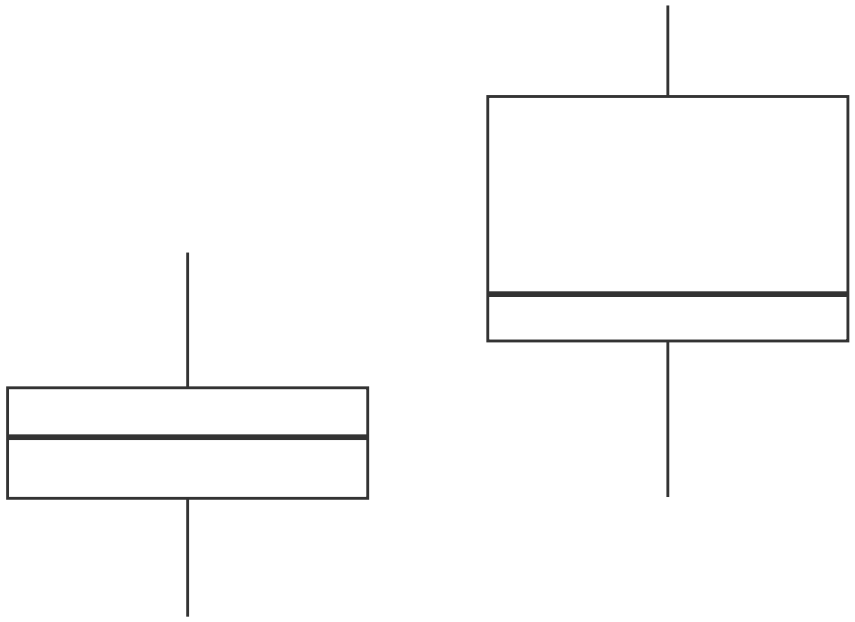

```
plottemp+theme_excel()+
  theme(axis.title.x =  element_text(face="bold", hjust=0.95))
```



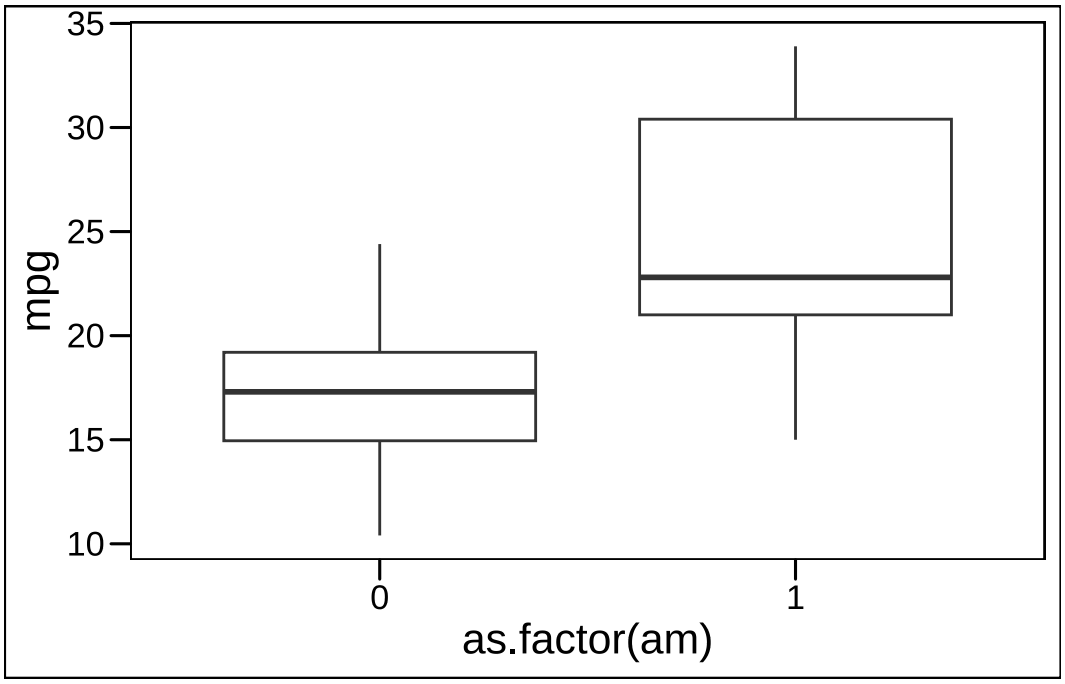
```
plottemp+theme_minimal()
```



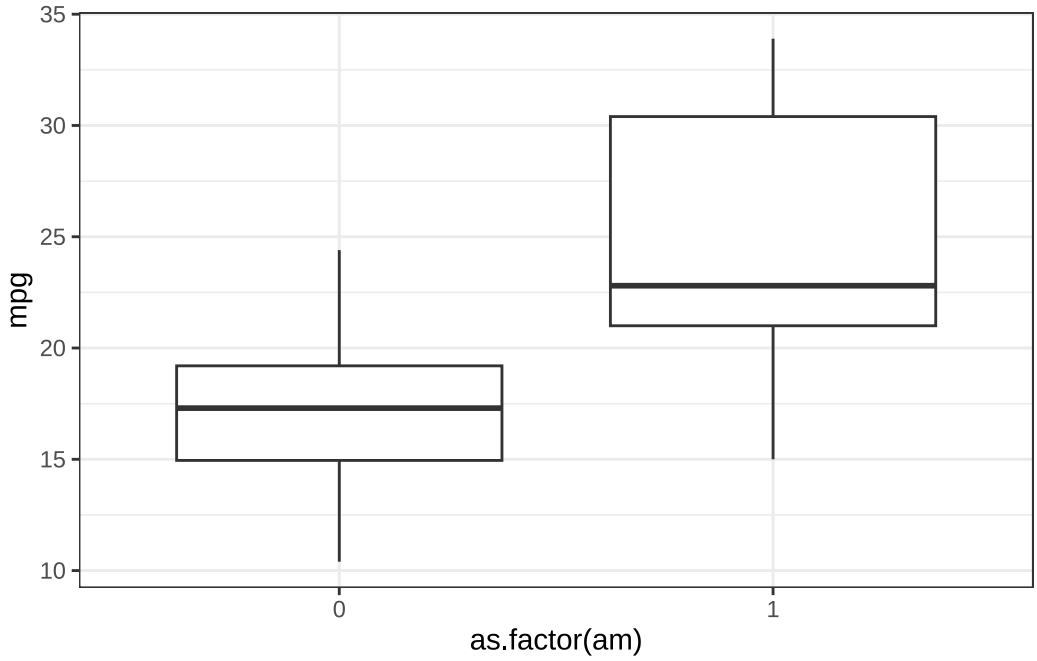

```
plottemp+theme_void()
```



```
plottemp+theme_base()
```




```
plottemp+theme_bw()
```

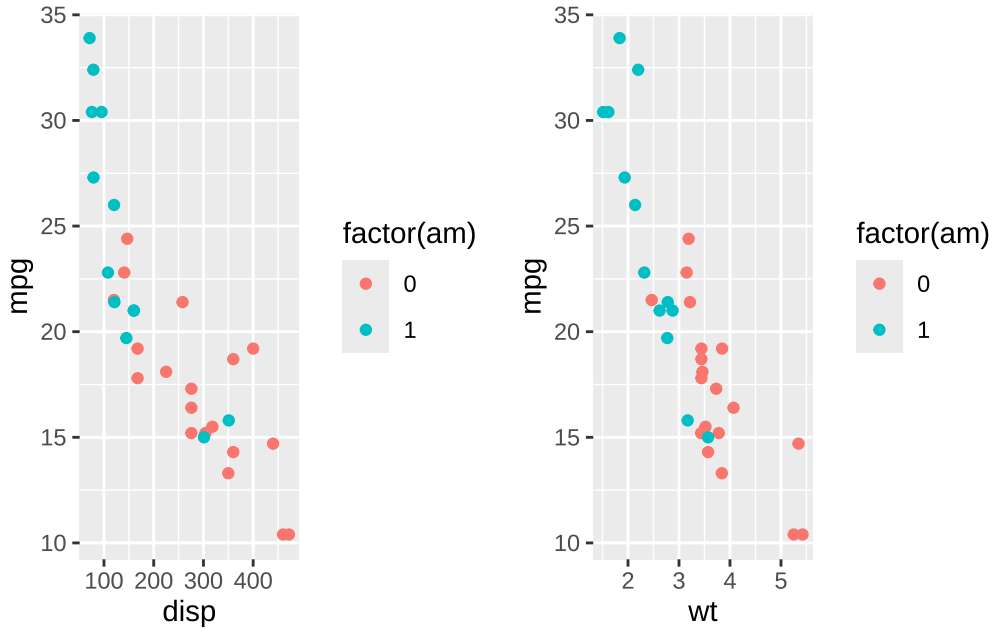


```
# https://www.data-imaginist.com/2019/a-flurry-of-facets/

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

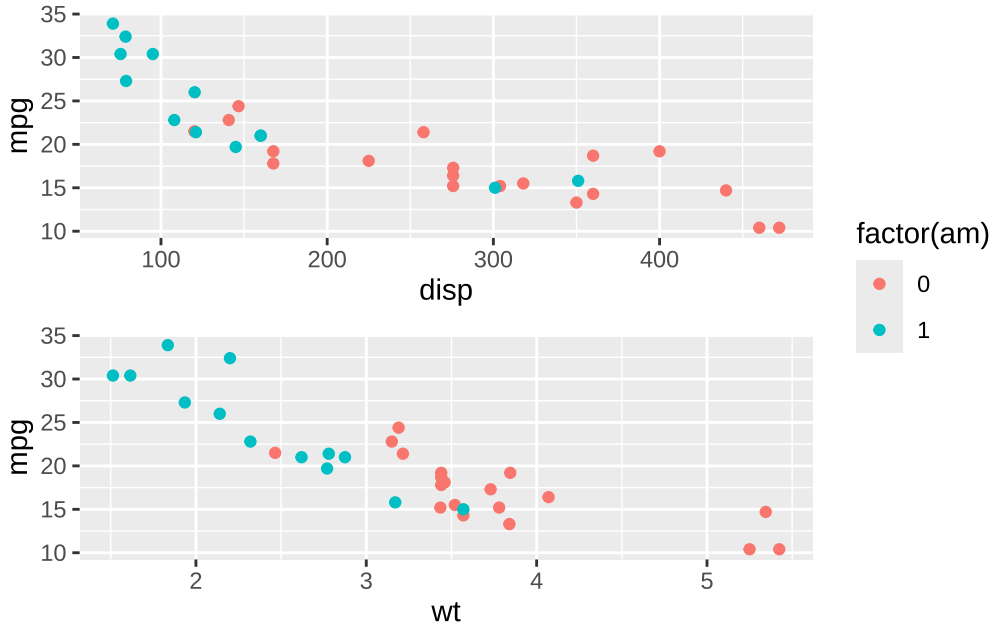
7.15 Combining figures with patchwork

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
```

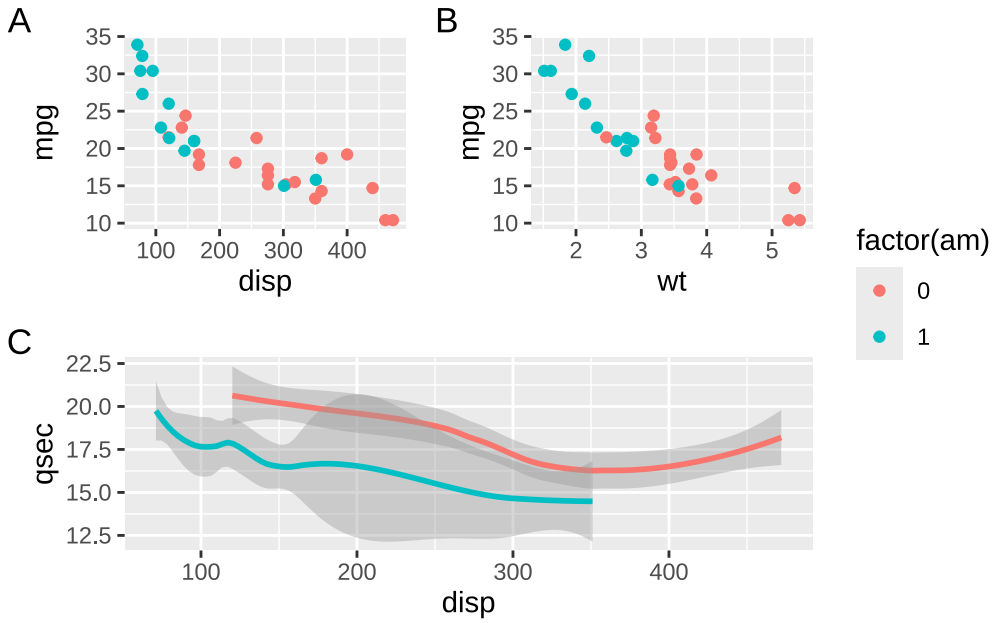



```
(p1 / p2) +
  plot_layout(guides = "collect")
```




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

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

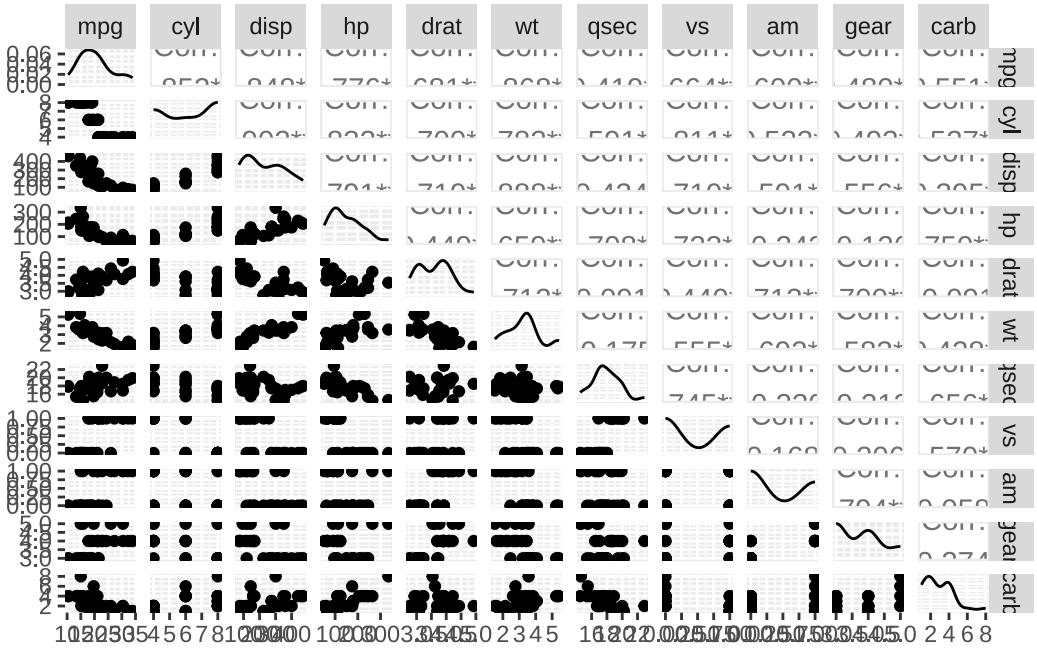


more facets...

animations...

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

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


mpg
cyl
disp
hp
drat
wt
qsec
vs
am
gear
carb

8 Descriptive statistics

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

8.1 Reading in data

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

8.2 Graphical exploration should start before descriptive statistics

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

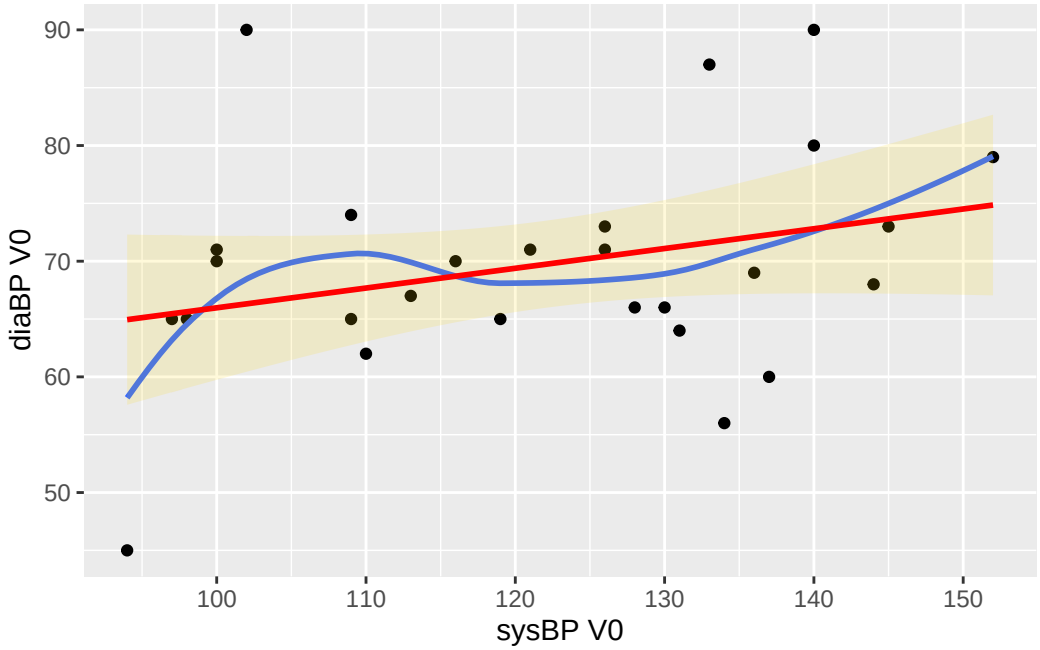
```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

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

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

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

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



8.3 Gaussian variables

8.3.1 Simple function calls

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

```
[1] 174.1071
```

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

```
[1] 7.771454
```

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

```
[1] 160
```

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

```
[1] 1.468667
```

8.3.2 Combined reporting

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

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

```
[1] 174.11
```

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

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

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

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

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

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

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

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

```
[1] "174.1 ± 1.5"
```

8.4 Ordinal variables

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

```
[1] 173.5
```

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

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

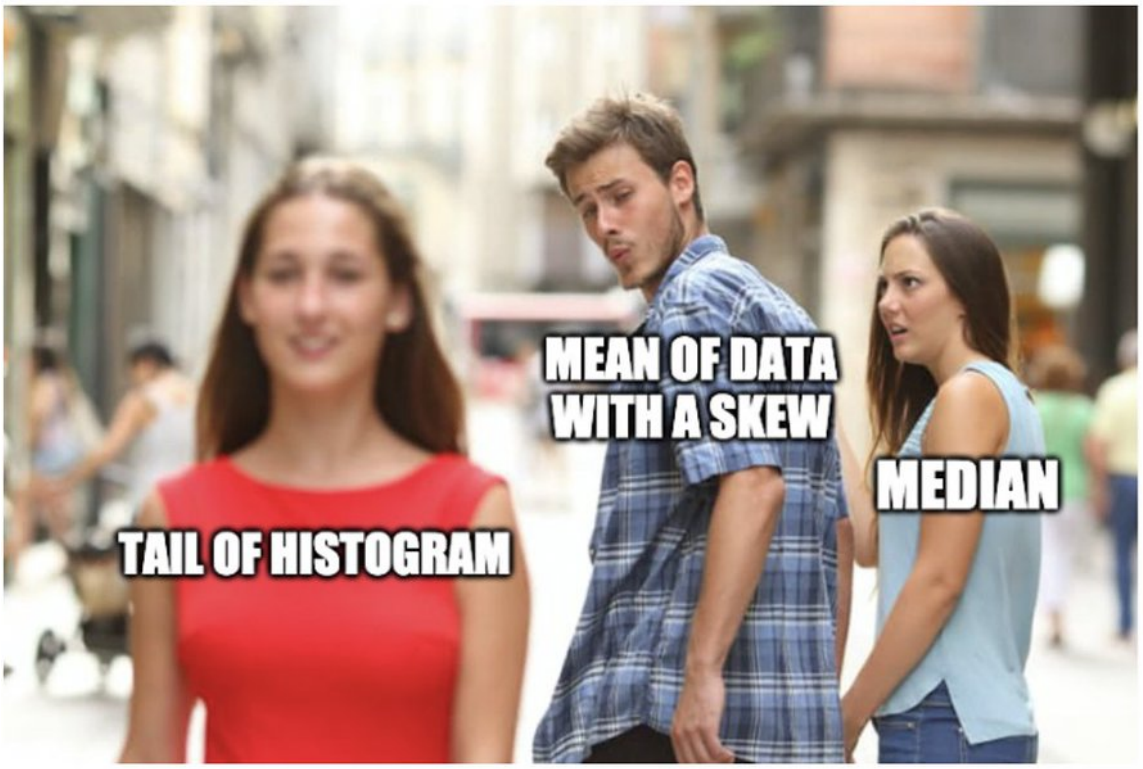
```
median_quart(rawdata$`Size (cm)`)
```

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

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

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

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


MEAN OF DATA
WITH A SKEW
MEDIAN
TAIL OF HISTOGRAM

8.5 Categorical variables

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

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

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

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

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

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

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

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

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

prop.table(sex_count)
```

```
        f         m
0.1428571 0.8571429
```

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

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

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

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

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

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

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

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

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

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

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

```
# cat(" \n\n")
```

8.6 Summarize data

When creating tables with descriptive statistics, you usually report on more than just 1 variable in more than just 1 subgroup, and there may be more than 1 statistics to report. `Summarize()`, often in combination with `across()`, makes that task easier:

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

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

```
# cat("<br>\n\n")

rawdata |>
  group_by(Sex) |>
  summarize(across(gaussvars$names,
                   .fns=~meansd(.x,range=T))) |>
  pivot_longer(cols=-Sex,
               names_to="Variable") |>
  pivot_wider(names_from=Sex) |>
  flextable()
```

Variable	f	m
Size (cm)	165 ± 4 [160 -> 168]	176 ± 7 [161 -> 193]
Weight (kg)	76 ± 13 [64 -> 93]	90 ± 14 [74 -> 120]
sysBP V0	118 ± 14 [102 -> 130]	123 ± 18 [94 -> 152]
diaBP V0	71 ± 13 [62 -> 90]	69 ± 9 [45 -> 90]
Lv Edv Mri	235 ± 72 [184 -> 286]	203 ± 71 [118 -> 379]
Lv Esv Mri	158 ± 56 [119 -> 198]	105 ± 71 [50 -> 294]

Variable	f	m
Lv Ef Mri	33 ± 3 [31 -> 35]	52 ± 15 [17 -> 74]
Lv Ef Biplan Mri	26 ± 11 [19 -> 34]	51 ± 13 [21 -> 69]
sysBP V2	114 ± 18 [96 -> 132]	120 ± 12 [100 -> 145]
diaBP V2	58 ± 12 [51 -> 72]	67 ± 8 [55 -> 80]
BMI	28 ± 5 [24 -> 35]	29 ± 4 [23 -> 41]

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

Variable	n (f)	Mean (f)	Median (f)	SD (f)	n (m)	Mean (m)	Median (m)	SD (m)
Size (cm)	4	165.00000	166.00000	3.829708	24	175.62500	174.50000	7.222022
Weight (kg)	4	76.25000	74.00000	13.073510	24	90.37500	86.00000	14.070668
sysBP V0	4	117.50000	119.00000	13.699148	24	122.60870	126.00000	17.541481
diaBP V0	4	71.00000	66.00000	12.806248	24	69.47826	70.00000	9.467022
Lv Edv Mri	4	235.00000	235.00000	72.124892	24	203.36842	193.00000	71.436227
Lv Esv Mri	4	158.50000	158.50000	55.861436	24	105.42105	71.00000	70.948585

Variable	n (f)	Mean (f)	Median (f)	SD (f)	n (m)	Mean (m)	Median (m)	SD (m)
Lv Ef Mri	4	33.00000	33.00000	2.828427	24	51.57895	56.00000	14.599527
Lv Ef Biplan Mri	4	26.50000	26.50000	10.606602	24	51.00000	54.00000	12.888410
sysBP V2	4	114.00000	114.00000	18.000000	24	120.23810	120.00000	12.140448
diaBP V2	4	58.33333	52.00000	11.846237	24	67.19048	68.00000	7.833384
BMI	4	28.00382	26.67234	4.748644	24	29.31779	28.73469	4.355720

```
# cat("<br>\n\n")

compare2numvars(rawdata,
                dep_vars = c( "Size (cm)","Weight (kg)",
                              "sysBP V0","diaBP V0"),
                indep_var = "Sex",
                gaussian = TRUE) |>
  flextable() |>
  set_table_properties(width=1, layout="autofit")
```

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

```
cat("<br>\n\n")
```


9 Summarize / across

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

1 variable / 1 function / no groups

```
summarize(.data = mtcars,
          MeanSD=meansd(mpg)) |>
  flextable()
```

MeanSD
20 ± 6

1 variable / 1 function / subgroups

```
mtcars |>
  group_by(am) |>
summarize(MeanSD=meansd(mpg),
          .groups = 'drop') |>
  flextable()
```

am	MeanSD
0	17 ± 4
1	24 ± 6

```
# groups in columns
mtcars |>
  group_by(am) |>
summarize(MeanSD=meansd(mpg),
          .groups = 'drop') |>
  pivot_wider(names_from = am,
              values_from = MeanSD,
              names_glue = "am: {am}\n{.value}") |>
  flextable()
```

am: 0 MeanSD	am: 1 MeanSD
17 ± 4	24 ± 6

1 variable / 2 functions / no groups

```
summarize(.data = mtcars,
          `MeanSD mpg`=meansd(mpg),
          `MedianQuartiles mpg`=median_quart(mpg)) |>
  flextable()
```

MeanSD mpg	MedianQuartiles mpg
20 ± 6	19 (15/23)

1 variable / 2 functions / subgroups

```
mtcars |>
  group_by(am) |>
summarize(`MeanSD mpg`=meansd(mpg),
          `MedianQuartiles mpg`=median_quart(mpg),
          .groups = 'drop') |>
  flextable()
```

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

```
# groups in columns
mtcars |>
  group_by(am) |>
summarize(`MeanSD mpg`=meansd(mpg),
          `MedianQuartiles mpg`=median_quart(mpg),
          .groups = 'drop') |>
  pivot_longer(
    cols = starts_with("M"),
    names_to = "Statistics",
    values_to = "estimate") |>
  pivot_wider(names_from = am,
```

```
              values_from = estimate,
              names_glue = "am: {am}") |>
  flextable()
```

Statistics	am: 0	am: 1
MeanSD mpg	17 ± 4	24 ± 6
MedianQuartiles mpg	17 (15/19)	23 (21/30)

2 variables / 1 function / no groups

```
# no function arguments
mtcars |>
  summarize(across(c(mpg,disp),
                   .fns=meansd)) |>
  flextable()
```

mpg	disp
20 ± 6	231 ± 124

```
# with function arguments
mtcars |>
  summarize(across(c(mpg,disp),
                   .fns=~meansd(.x,add_n = TRUE))) |>
  flextable()
```

mpg	disp
20 ± 6 [n=32]	231 ± 124 [n=32]

```
# Variables in rows
mtcars |>
  summarize(across(c(mpg,disp),
                   .fns=~meansd(.x,add_n = TRUE))) |>
  pivot_longer(
    cols = everything(),
    names_to = "Variable",
    values_to = "MeanSD") |>
```

```
  flextable() |>
  set_table_properties(width=.7, layout="autofit")
```

Variable	MeanSD
mpg	20 ± 6 [n=32]
disp	231 ± 124 [n=32]

2 variables / 1 function / subgroups

```
mtcars |>
  group_by(am) |>
  summarize(across(c(mpg,disp),
                   ~meansd(.x,add_n = TRUE)),
            .groups = 'drop') |>
  flextable()
```

am	mpg	disp
0	17 ± 4 [n=19]	290 ± 110 [n=19]
1	24 ± 6 [n=13]	144 ± 87 [n=13]

```
# Variables in rows
mtcars |>
  group_by(am) |>
  summarize(across(c(mpg,disp),
                   ~meansd(.x,add_n = TRUE)),
            .groups = 'drop') |>
  pivot_longer(
    cols = -am,
    names_to = "Variable",
    values_to = "MeanSD") |>
  pivot_wider(names_from = am,
              values_from = MeanSD,
              names_prefix = "am: ") |>
  flextable() |>
  set_table_properties(width=.7, layout="autofit")
```

Variable	am: 0	am: 1
mpg	17 ± 4 [n=19]	24 ± 6 [n=13]
disp	290 ± 110 [n=19]	144 ± 87 [n=13]

2 variables / 2 function / no groups

```
# with/without function arguments
mtcars |>
  summarize(across(
    c(mpg,disp),
    .fns=list(
      MeanSD=~meansd(.x,
                     add_n = TRUE),
      MedianQuart=median_quart))) |>
  flextable() |>
  set_table_properties(width=1, layout="autofit")
```

mpg_MeanSD	mpg_MedianQuart	disp_MeanSD	disp_MedianQuart
20 ± 6 [n=32]	19 (15/23)	231 ± 124 [n=32]	196 (121/337)

```
# Variables in rows
mtcars |>
  summarize(across(
    c(mpg,disp),
    .fns=list(
      MeanSD=~meansd(.x,
                     add_n = TRUE),
      MedianQuart=median_quart))) |>
  pivot_longer(
    cols = everything(),
    names_to = c("Variable",".value"),
    names_sep="_",
    values_to = "MeanSD") |>
  flextable() |>
  set_table_properties(width=1, layout="autofit")
```

Variable	MeanSD	MedianQuart
mpg	20 ± 6 [n=32]	19 (15/23)
disp	231 ± 124 [n=32]	196 (121/337)

2 variables / 2 function / subgroups

```
# with/without function arguments
mtcars |>
  group_by(am) |>
  summarize(across(
    c(mpg,disp),
    .fns=list(
      MeanSD=~meansd(.x,
                     add_n = TRUE),
      MedianQuart=median_quart)),
    .groups="drop") |>
  flextable() |>
  set_table_properties(width=1, layout="autofit")
```

am	mpg_MeanSD	mpg_MedianQuart	disp_MeanSD	disp_MedianQuart
0	17 ± 4 [n=19]	17 (15/19)	290 ± 110 [n=19]	276 (177/360)
1	24 ± 6 [n=13]	23 (21/30)	144 ± 87 [n=13]	120 (79/160)

```
# Variables in rows, groups in columns
mtcars |>
  group_by(am) |>
  summarize(across(
    c(mpg,disp),
    .fns=list(
      MeanSD=~meansd(.x,
                     add_n = TRUE),
      MedianQuart=median_quart)),
    .groups="drop") |>
  pivot_longer(
    cols = -am,
    names_to = c("Variable",".value"),
    names_sep="_",
    values_to = "MeanSD") |>
  pivot_wider(names_from = am,
              values_from = starts_with("M"),
              names_glue = "am: {am}_{.value}",
              names_vary="slowest") |>
  flextable() |>
  separate_header(split="[:_]") |>
  set_table_properties(width=1, layout="autofit")
```

	am			
	0		1	
Variable	MeanSD	MedianQuart	MeanSD	MedianQuart
mpg	17 ± 4 [n=19]	17 (15/19)	24 ± 6 [n=13]	23 (21/30)
disp	290 ± 110 [n=19]	276 (177/360)	144 ± 87 [n=13]	120 (79/160)

```
# pivoting to have variables in rows V2

# with/without function arguments
result_long <-
  mtcars |>
  group_by(am) |>
  summarize(across(
    c(mpg,disp),
    .fns=list(
      MeanSD=~meansd(.x,
                     add_n = TRUE),
      MedianQuart=median_quart))) |>
  pivot_longer(cols = -c(am),
               names_to = c('Variable','.value'),
               names_sep="_",
               values_to = 'Value')
result_long |>
  flextable() |>
  merge_v(j=1) |>
  set_table_properties(width=1, layout="autofit")
```

am	Variable	MeanSD	MedianQuart
0	mpg	17 ± 4 [n=19]	17 (15/19)
	disp	290 ± 110 [n=19]	276 (177/360)
1	mpg	24 ± 6 [n=13]	23 (21/30)
	disp	144 ± 87 [n=13]	120 (79/160)

```
result <-
  result_long |>
  pivot_wider(names_from=am,
              names_prefix="am:",
              names_sep=" ",
              values_from=c(MeanSD, MedianQuart))
```

```
result |>
  flextable() |>
  separate_header(split="[ ]") |>
  set_table_properties(width=1, layout="autofit")
```

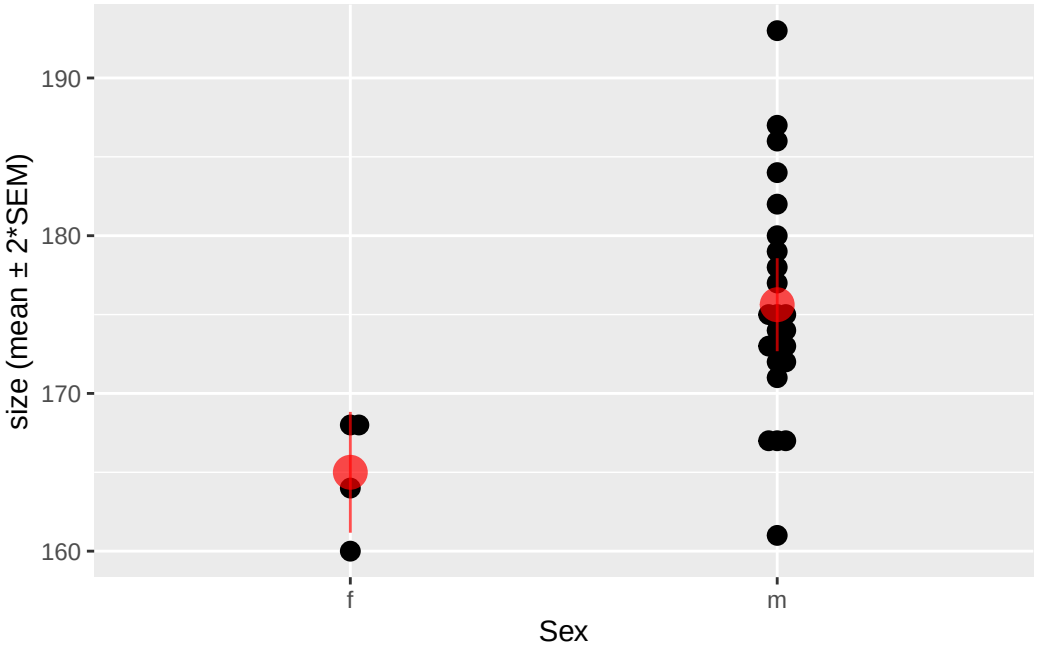
	MeanSD		MedianQuart	
Variable	am:0	am:1	am:0	am:1
mpg	17 ± 4 [n=19]	24 ± 6 [n=13]	17 (15/19)	23 (21/30)
disp	290 ± 110 [n=19]	144 ± 87 [n=13]	276 (177/360)	120 (79/160)

10 Simple test statistics

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

10.1 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))+
  ylab("size (mean \u00B1 2*SEM)")
```



```
  rawdata |>
    group_by(Sex) |>
  summarize(MeanSE=meanse(`Size (cm)`))
```

```
# A tibble: 2 x 2
  Sex   MeanSE
  <fct> <chr>
1 f     165 ± 2
2 m     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)
  tOut$p.value
```

```
[1] 0.00288704
```

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

```
    F test to compare two variances

data:  rawdata$`Size (cm)` by rawdata$Sex
F = 0.2812, num df = 3, denom df = 23, p-value = 0.3232
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
```

```
 0.07497668 3.97434769
sample estimates:
ratio of variances
          0.281199
```

```r
# manual entry
t.test(rawdata$`Size (cm)`~rawdata$Sex,
               var.equal = vartestOut$p.value>.05)
```

```
    Two Sample t-test

data:  rawdata$`Size (cm)` by rawdata$Sex
t = -2.8446, df = 26, p-value = 0.008552
alternative hypothesis: true difference in means between group f and group m is not equal
95 percent confidence interval:
 -18.302594  -2.947406
sample estimates:
mean in group f mean in group m
        165.000         175.625
```

```r
# picked from test
t.test(rawdata$`Size (cm)`~rawdata$Sex,
       var.equal=var.test(
         rawdata$`Size (cm)`~rawdata$Sex)$p.value>.05)
```

```
    Two Sample t-test

data:  rawdata$`Size (cm)` by rawdata$Sex
t = -2.8446, df = 26, p-value = 0.008552
alternative hypothesis: true difference in means between group f and group m is not equal
95 percent confidence interval:
 -18.302594  -2.947406
sample estimates:
mean in group f mean in group m
        165.000         175.625
```

```r
#combined function
t_var_test(data = rawdata,
           formula = "`Size (cm)`~Sex",
           cutoff = .1)
```

```
    Two Sample t-test

data:  Size (cm) by Sex
t = -2.8446, df = 26, p-value = 0.008552
alternative hypothesis: true difference in means between group f and group m is not equal
95 percent confidence interval:
 -18.302594  -2.947406
sample estimates:
mean in group f mean in group m
        165.000         175.625
```

```r
print(c(mean(rawdata$`sysBP V0`,na.rm=T),
        mean(rawdata$`sysBP V2`,na.rm=T)))
```

```
[1] 121.8519 119.4583
```

```r
t.test(rawdata$`sysBP V0`,
       rawdata$`sysBP V2`,
       alternative="greater", # x>y
       paired=TRUE)  #pairwise t-test, within subject
```

```
    Paired t-test

data:  rawdata$`sysBP V0` and rawdata$`sysBP V2`
t = 0.88151, df = 23, p-value = 0.1936
alternative hypothesis: true mean difference is greater than 0
95 percent confidence interval:
 -2.793386       Inf
sample estimates:
mean difference
       2.958333
```

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

```
[1] 0.1935805
```

```r
t.test(rawdata$`Size (cm)`,mu = 173)
```

```
    One Sample t-test

data:  rawdata$`Size (cm)`
t = 0.75384, df = 27, p-value = 0.4575
alternative hypothesis: true mean is not equal to 173
95 percent confidence interval:
 171.0937 177.1206
sample estimates:
mean of x
 174.1071
```

```r
groupvars <- ColSeeker(namepattern = c("Sex","Test"))

compare2numvars(data = rawdata,dep_vars = gaussvars$names,
                indep_var = "Sex",gaussian = T) |>
  flextable() |>
  set_table_properties(width=1, layout="autofit")
```

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

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

```
  set_table_properties(width=1, layout="autofit")
```

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

```
for(group_i in seq_len(groupvars$count)){
  resulttmp <-
    compare2numvars(data = rawdata,
                    dep_vars = gaussvars$names,
                    indep_var = groupvars$names[group_i],gaussian = T)
  # print(resulttmp)
  flextable(resulttmp)  |>
  set_table_properties(width=1, layout="autofit") |>
  flex2rmd() #|> print()
  cat("<br>\n\n")
  cat("\\newpage\n\n")
}
```

Variable	desc_all	Testmedication 0	Testmedication 1	p
Size (cm)	174 ± 8	173 ± 6	175 ± 9	0.653
Weight (kg)	88 ± 15	90 ± 14	86 ± 16	0.448
sysBP V0	122 ± 17	122 ± 15	122 ± 19	0.966
diaBP V0	70 ± 10	70 ± 7	70 ± 12	0.943
Lv Edv Mri	206 ± 70	243 ± 77	173 ± 45	0.019

Variable	desc_all	Testmedication 0	Testmedication 1	p
Lv Esv Mri	110 ± 70	138 ± 87	86 ± 41	0.108
Lv Ef Mri	50 ± 15	48 ± 18	52 ± 12	0.532
Lv Ef Biplan Mri	49 ± 14	47 ± 17	50 ± 13	0.747
sysBP V2	119 ± 13	122 ± 11	117 ± 14	0.297
diaBP V2	66 ± 9	66 ± 8	66 ± 9	0.966
BMI	29 ± 4	30 ± 5	28 ± 3	0.180

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

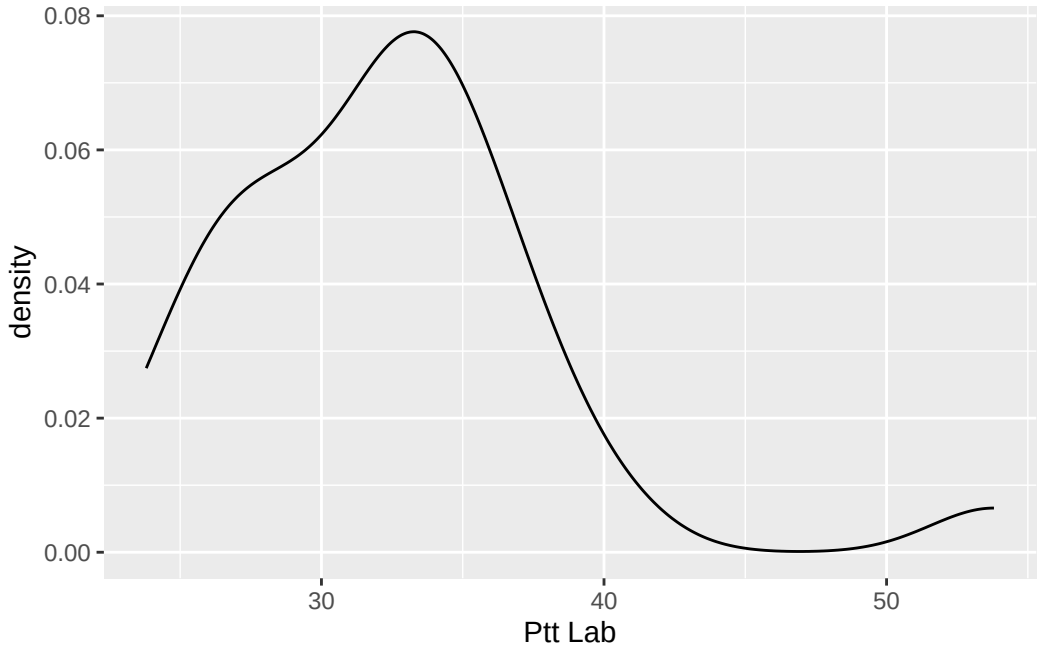
10.2 Ordinal data

```
ordvars$names
```

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

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

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



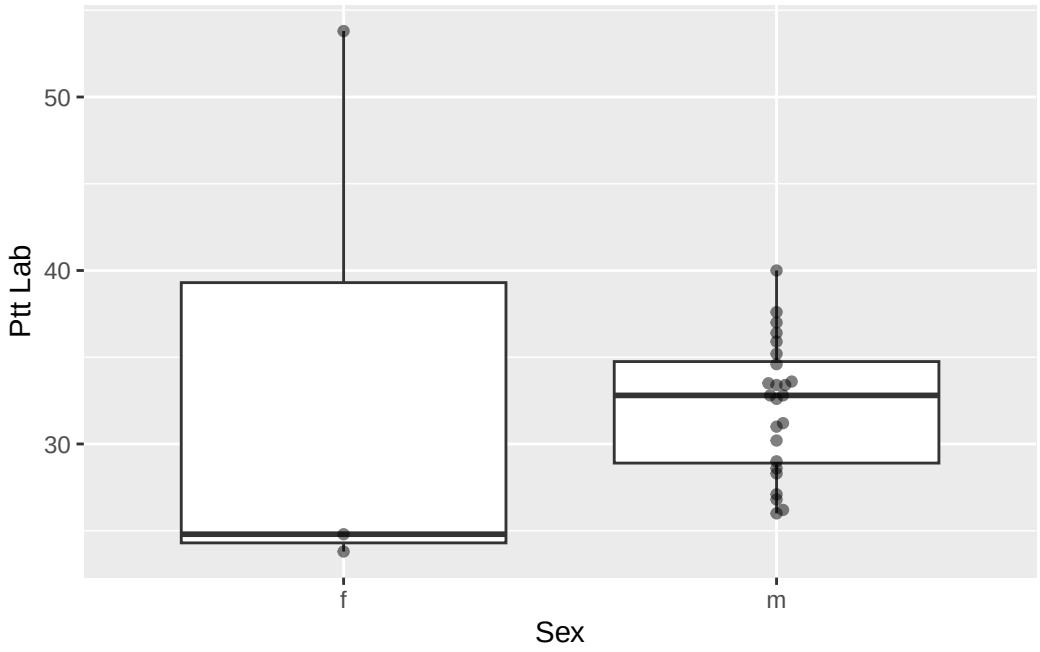

```
by(data = rawdata[[ordvars$index[1]]],
   INDICES = rawdata$Sex,FUN = median_quart)
```

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

```
ggplot(rawdata,aes(Sex,`Ptt Lab`))+
  geom_boxplot()+
  geom_beeswarm(alpha=.5)
```

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

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




```
(uOut<-wilcox.test(
  rawdata[[ordvars$names[3]]]~rawdata$Sex,exact=F))
```

```
    Wilcoxon rank sum test with continuity correction

data:  rawdata[[ordvars$names[3]]] by rawdata$Sex
W = 54.5, p-value = 0.1648
alternative hypothesis: true location shift is not equal to 0
```

```
uOut$p.value
```

```
[1] 0.1647858
```

```
#  coin::wilcox_test
(uOut2<-wilcox_test(`Ptt Lab`~Sex,
                    data=rawdata))
```

```
    Asymptotic Wilcoxon-Mann-Whitney Test

data:  Ptt Lab by Sex (f, m)
Z = -0.9261, p-value = 0.3544
alternative hypothesis: true mu is not equal to 0
```

```
pvalue(uOut2) #no list-object, but methods to extract infos like p
```

```
[1] 0.3543925
```

```
wilcox.test(`Ptt Lab`~Sex,exact=F,correct=F,
            data=rawdata)
```

```
    Wilcoxon rank sum test

data:  Ptt Lab by Sex
W = 24, p-value = 0.3544
alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(x=rawdata$`sysBP V0`,y=rawdata$`sysBP V2`,
            exact=FALSE,
            correct=TRUE,paired=TRUE)
```

```
    Wilcoxon signed rank test with continuity correction

data:  rawdata$`sysBP V0` and rawdata$`sysBP V2`
V = 143, p-value = 0.3478
alternative hypothesis: true location shift is not equal to 0
```

```
compare2numvars(data = rawdata,dep_vars = ordvars$names,n = F,
                range = T,add_n = T,
                indep_var = "Sex",gaussian = F) |>
```

```
    flextable() |>
    set_table_properties(width=1, layout="autofit")
```

Variable	desc_all	Sex f	Sex m
Ptt Lab	33 (28/35) [24 -> 54] [n=27]	25 (24/49) [24 -> 54] [n=3]	33 (29/35) [26 -> 4
Ferritin Lab	222 (162/339) [20 -> 1182] [n=27]	138 (90/591) [81 -> 681] [n=3]	224 (172/316) [20
Iron Lab	80 (61/102) [31 -> 191] [n=27]	95 (92/103) [91 -> 105] [n=3]	75 (60/99) [31 -> 1
Transferrin Lab	261 (233/276) [194 -> 343] [n=27]	260 (227/299) [221 -> 307] [n=3]	262 (235/276) [194
Age	62 (53/67) [43 -> 74] [n=28]	66 (58/69) [53 -> 69] [n=4]	60 (53/66) [43 -> 7

10.3 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 = 0.1735
```

```
fisher_out <- fisher.test(
  table(rawdata$Sex,rawdata$`NYHA V1`))
fisher_out$p.value
```

```
[1] 0.09558824
```

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

```
  FALSE TRUE
f     1    3
m    13   11
```

```
(tabletestOut<-chisq.test(crosstab1,simulate.p.value=T,
                          B=10^5))
```

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

data:  crosstab1
X-squared = 1.1667, df = NA, p-value = 0.5945
```

```
  tabletestOut$p.value
```

```
[1] 0.5945141
```

```
  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
```

```r
# report_cat
groupvar <- "Testmedication"

compare2qualvars(rawdata,dep_vars = factvars$names[-1],
                 indep_var = groupvar,spacer = " ") |>
  flextable()
```

Variable	desc_all	Testmedication 0	Testmedication 1	p
Sex				1.000
f	4 (14.29%)	2 (14.29%)	2 (14.29%)	
m	24 (85.71%)	12 (85.71%)	12 (85.71%)	
NYHA V1				0.724
0	2 (11.76%)	0 (0%)	2 (20%)	
1	9 (52.94%)	4 (57.14%)	5 (50%)	
2	3 (17.65%)	1 (14.29%)	2 (20%)	
3	3 (17.65%)	2 (28.57%)	1 (10%)	
NYHA V2				0.827
0	1 (9.09%)	1 (20%)	0 (0%)	
1	6 (54.55%)	2 (40%)	4 (66.67%)	
2	2 (18.18%)	1 (20%)	1 (16.67%)	
3	2 (18.18%)	1 (20%)	1 (16.67%)	
NYHA V3				0.769
1	6 (50%)	2 (40%)	4 (57.14%)	
2	3 (25%)	1 (20%)	2 (28.57%)	
3	3 (25%)	2 (40%)	1 (14.29%)	

11 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.

11.1 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()
```

11.2 Import / Preparation

Data are read from an SPSS file. Numeric column Passage is mutated into a factor as Passage_F, this is necessary for group comparisons in ANOVA. The call to here() expands the path to a file from the project directory to the full system path.

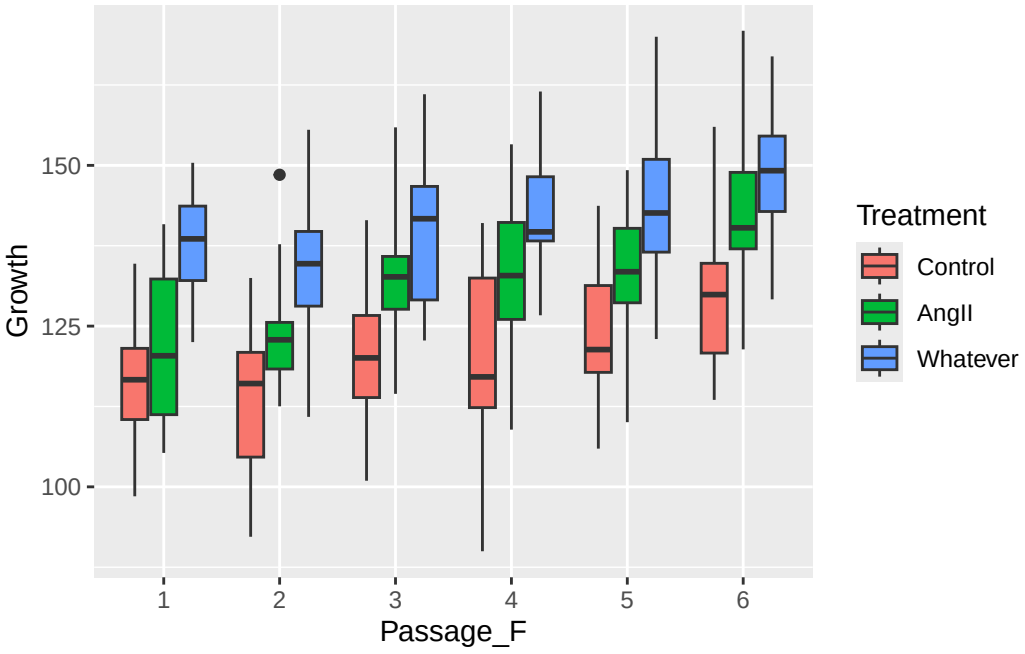
```
rawdata<-foreign::read.spss(file=here('Data/Zellbeads.sav'),
                            use.value.labels=T,to.data.frame=T) |>
  as_tibble() |>
  dplyr::select(-ZahlZellen) |>
  rename(Growth=Wachstum,Treatment=Bedingung) |>
  mutate(Passage_F=factor(Passage),
         Treatment=fct_recode(Treatment,
                              Control="Kontrolle"))
```

```
Zurückkodierung von CP1252
```

11.3 Graphical exploration

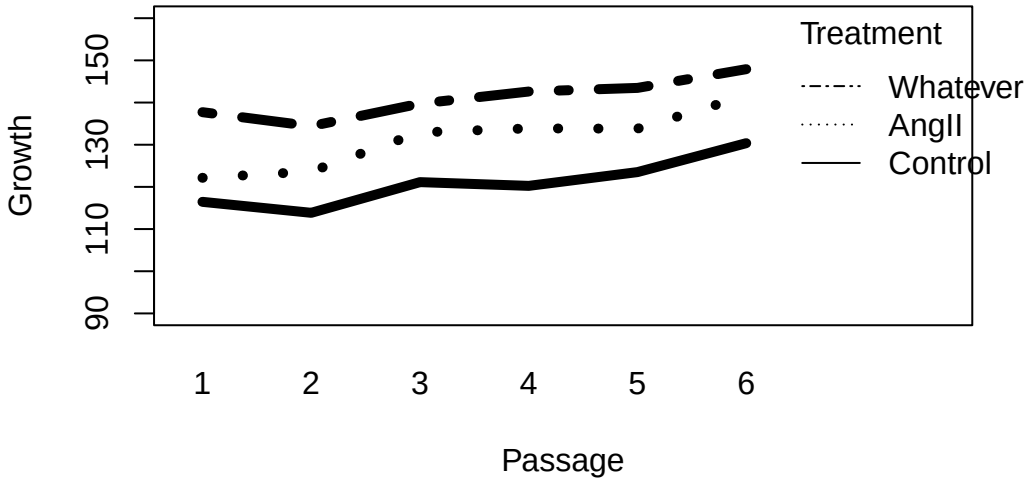
First impression of the data will be attempted by grouped boxplot, followed by interaction plots, both as basic and ggplot with variations.

```
ggplot(rawdata,aes(Passage_F,Growth, fill=Treatment))+
  geom_boxplot(coef=3)
```



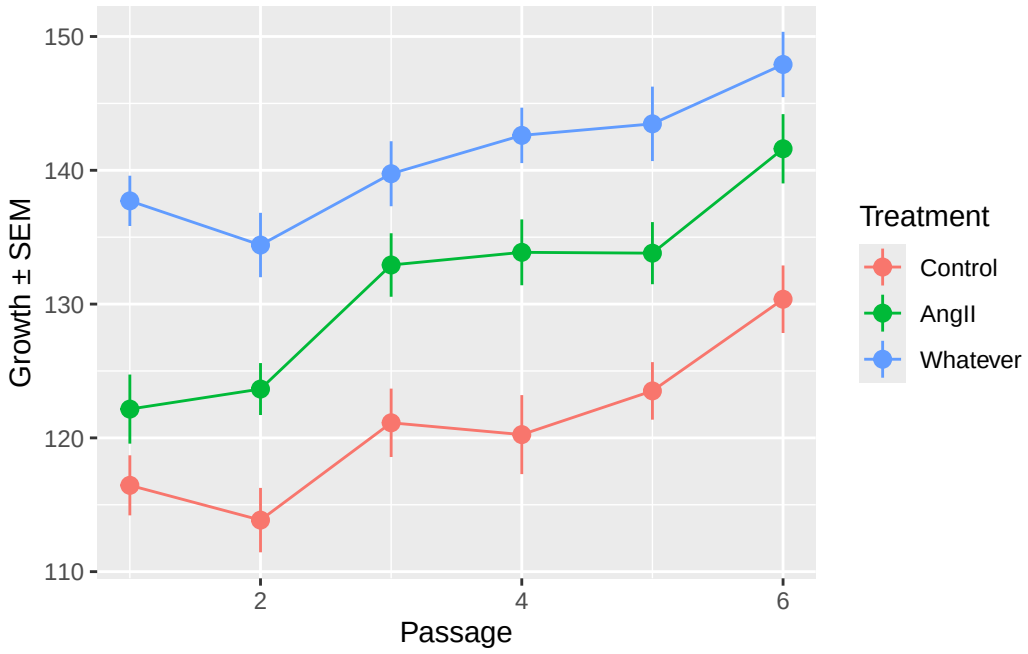

```
with(rawdata, interaction.plot(
  x.factor=Passage, trace.factor=Treatment, response=Growth,
  ylim = c(90, 160), lty = c(1,3,12), lwd = 5,
  ylab = "Growth", xlab = "Passage",
```

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



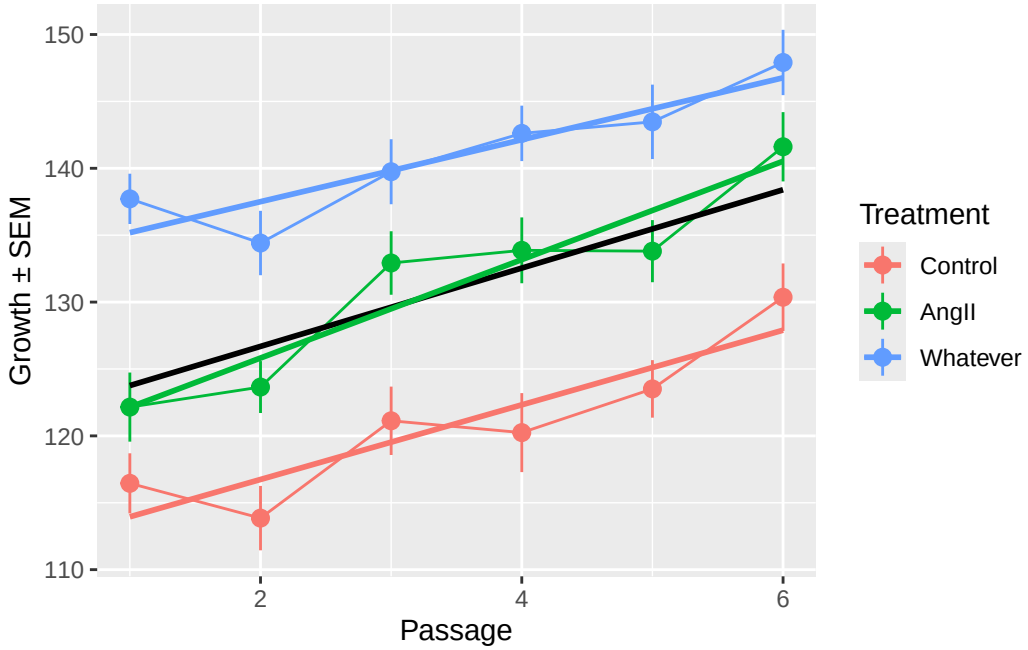

```
# p1<-ggplot(rawdata,aes(x=Passage,y=Growth))+
#    stat_summary(geom='line',fun='mean',aes(color=Treatment))+
#    stat_summary(geom='line',fun='mean')
p1<-ggplot(rawdata,aes(x=Passage,y=Growth))+
  stat_summary(geom='line',fun='mean',aes(color=Treatment))+
  stat_summary(aes(color=Treatment))+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

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



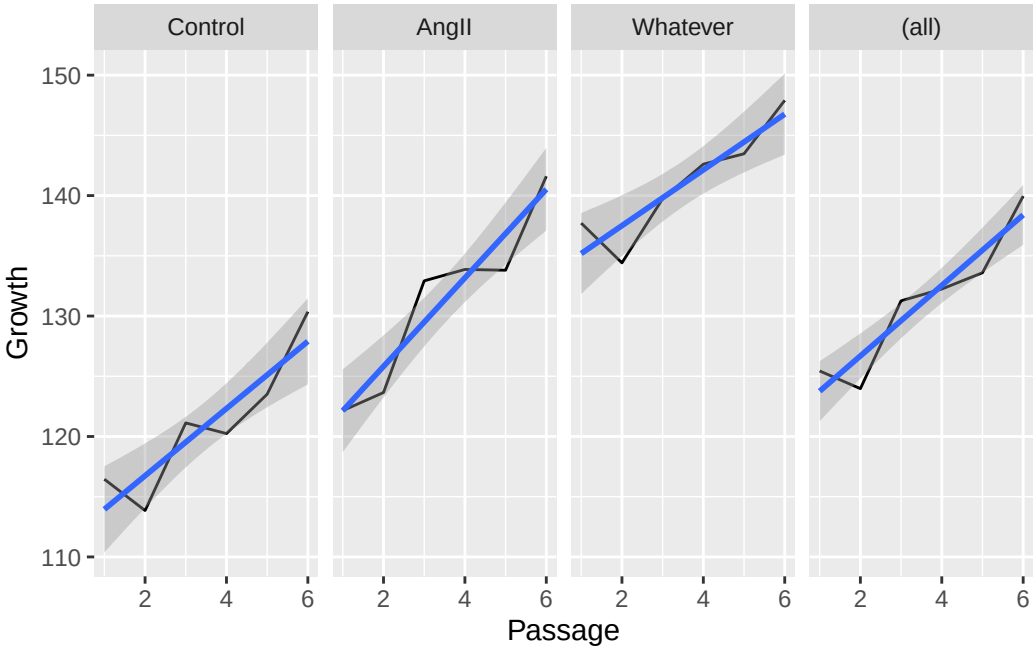

```
p1+geom_smooth(method='lm',color='black',se=F)+
  geom_smooth(method='lm',aes(color=Treatment),se=F)
```

```
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'
```



11.4 Linear Models

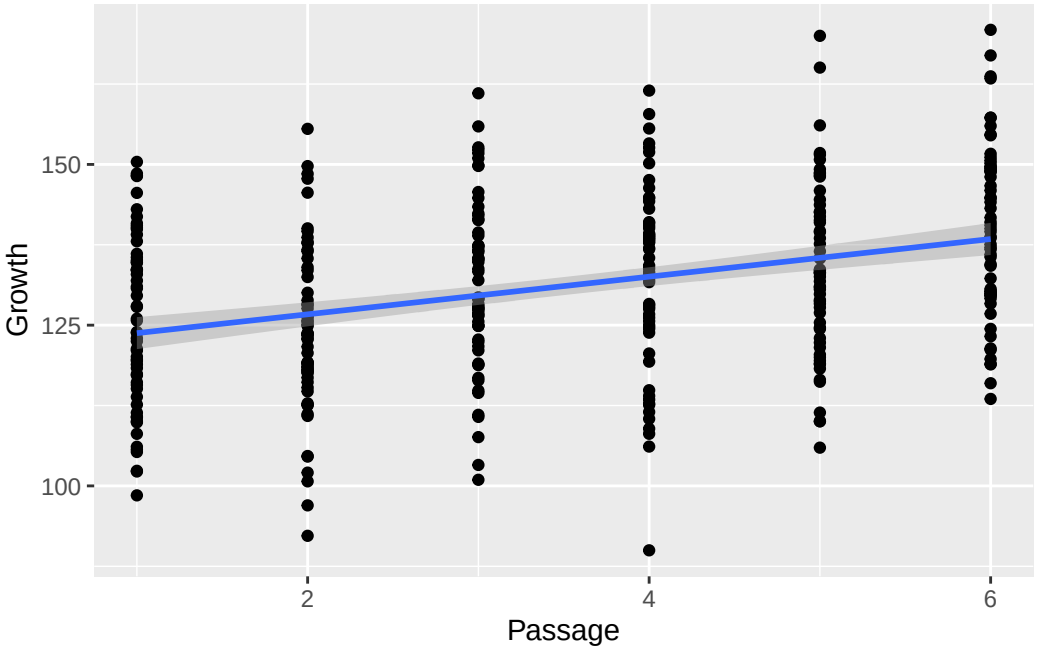
11.4.1 Linear regression

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

11.4.1.1 Graphical exploration

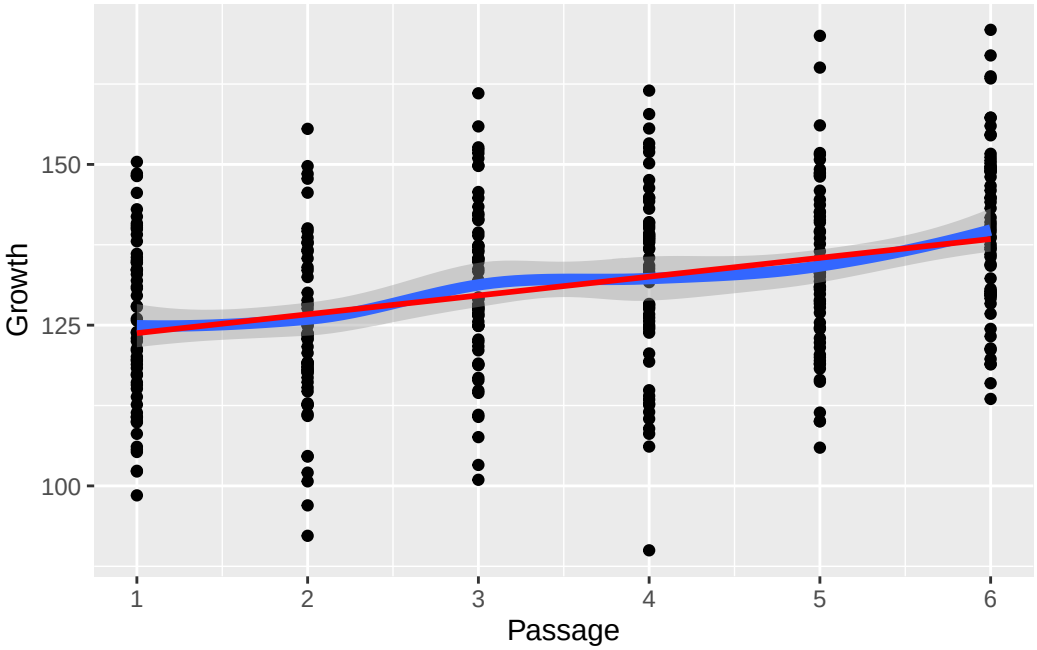
```
ggplot(rawdata,aes(Passage,Growth))+
  geom_point()+
  geom_smooth(method=lm)
```

```
`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'
```



11.4.1.2 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
```

```r
# 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
```

```r
anova_out$`Pr(>F)` #|> na.omit()
```

```
[1] 1.257266e-11           NA
```

```r
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
```

```r
# summary(regressionOut)
# str(regressionOut)
```

11.4.1.3 Adjusting

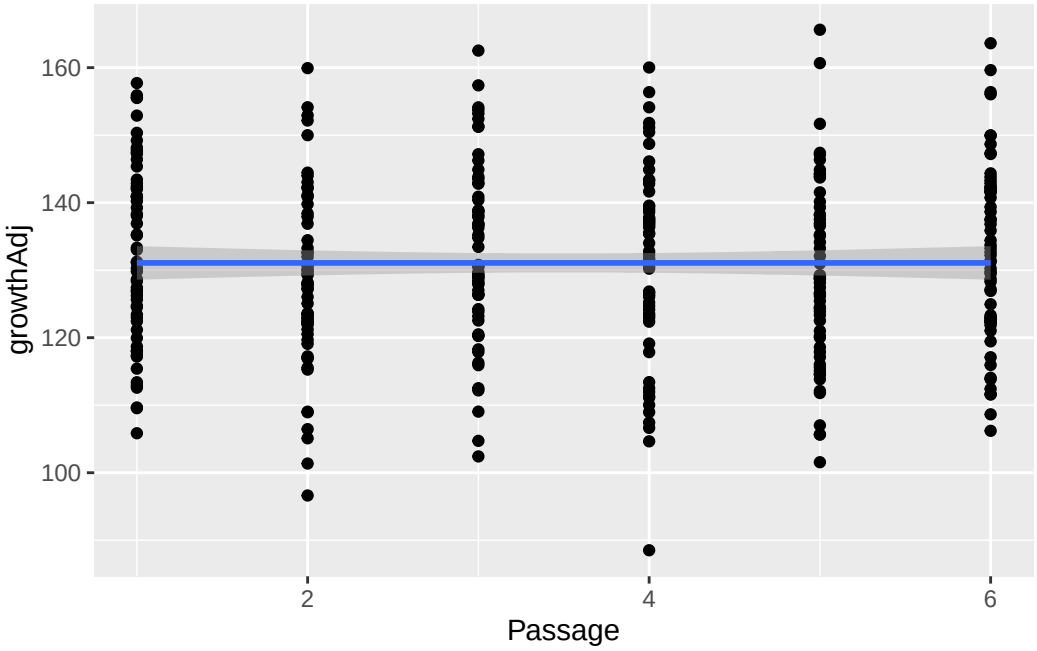
To take out the variance due to Passage effects, we can use the residuals and shift them to the original mean:

```r
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()
```

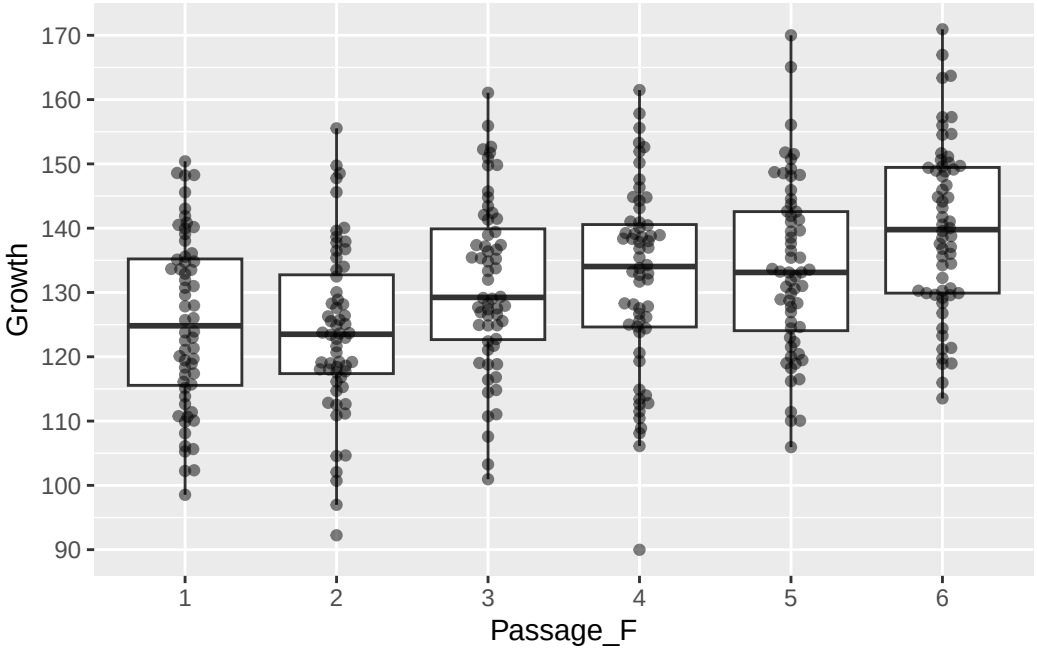
```
# A tibble: 2 x 5
  term        estimate std.error statistic   p.value
  <chr>          <dbl>     <dbl>     <dbl>     <dbl>
1 (Intercept) 1.31e+ 2     1.63   8.05e+ 1 2.04e-231
2 Passage     6.80e-15     0.418  1.63e-14 1.00e+  0
```

11.4.2 ANOVA

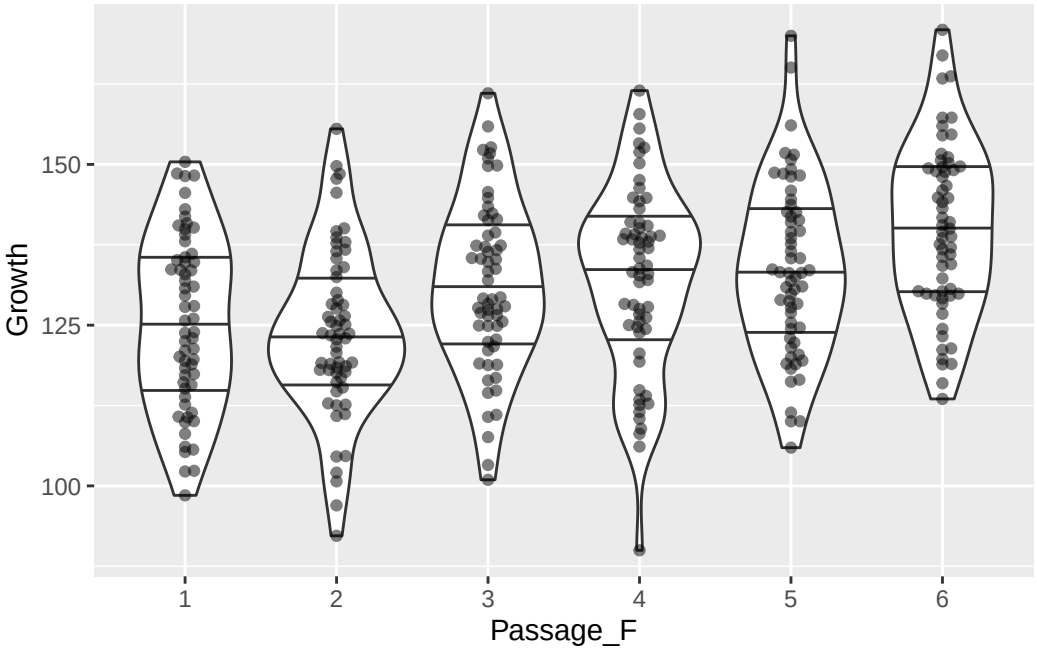
In the linear regression, we had Passage as a continuous IV, estimating a global 'universal' effect supposed to be constant. Now we look at Passage_F and model a discrete IV, allowing for specific effects, and thereby comparing means between groups.

11.4.2.1 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)
```



11.4.2.2 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
```

```r
# 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
```

```r
t$`Pr(>F)`
```

```
[1] 5.851856e-10           NA
```

```r
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
```

11.4.2.3 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!

```r
# possible in a loop, but nominal p
t.test(rawdata$Growth[which(rawdata$Passage==1)],
       rawdata$Growth[which(rawdata$Passage==2)],
       var.equal = T)
```

```
    Two Sample t-test

data:  rawdata$Growth[which(rawdata$Passage == 1)] and rawdata$Growth[which(rawdata$Passag
```

```
t = 0.60679, df = 118, p-value = 0.5452
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -3.321297  6.255936
sample estimates:
mean of x mean of y
 125.4396  123.9723
```

```
  # all pairwise group combinations
  pt_out<-pairwise.t.test(x=rawdata$Growth,
                          g=rawdata$Passage_F,
                          p.adjust.method='none')
  pt_out
```

```
    Pairwise comparisons using t tests with pooled SD

data:  rawdata$Growth and rawdata$Passage_F

  1       2       3       4       5
2 0.55215 -       -       -       -
3 0.01871 0.00331 -       -       -
4 0.00611 0.00088 0.69214 -       -
5 0.00104 0.00011 0.34487 0.58296 -
6 9e-09   3e-10   0.00048 0.00189 0.01025

P value adjustment method: none
```

```
  pairwise.t.test(x=rawdata$Growth,g=rawdata$Passage,
                  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'))
```

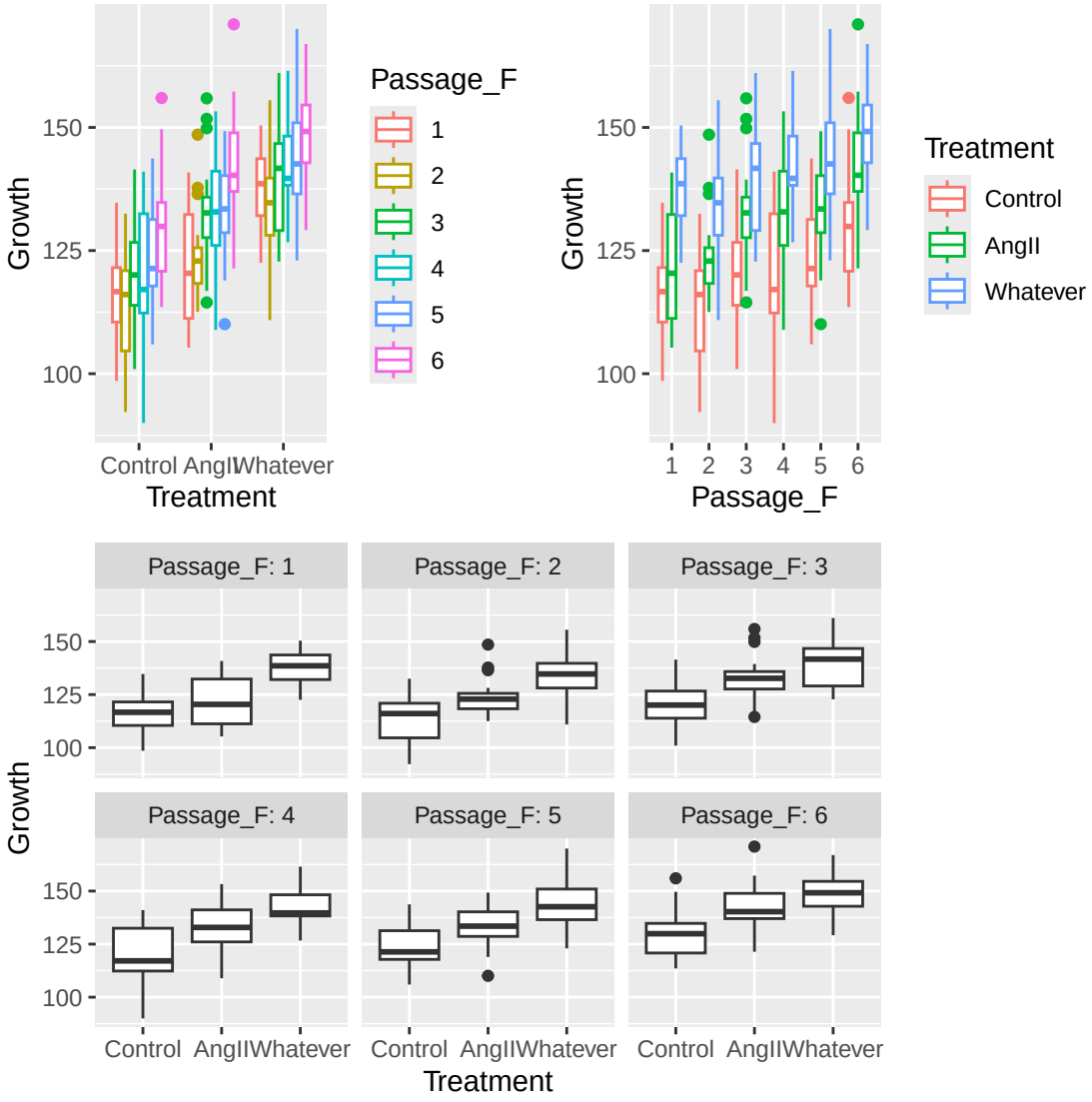
```
[1] "0.552" "0.023" "0.010" "0.003" "0.001"
```

11.4.3 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.

11.4.3.1 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
```



11.4.3.2 Modelling

Models with (*) and without (+) interaction are build and tested.

```
lmOut_interaction<-lm(Growth~Passage*Treatment,data=rawdata)
Anova(lmOut_interaction,type = 3)
```

```
Anova Table (Type III tests)

Response: Growth
                  Sum Sq  Df    F value    Pr(>F)
```

```
(Intercept)       285160   1 2448.5613 < 2.2e-16 ***
Passage             2723   1   23.3855 1.981e-06 ***
Treatment           5635   2   24.1924 1.419e-10 ***
Passage:Treatment    335   2    1.4376    0.2389
Residuals          41227 354
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#
lmOut_additive<-lm(Growth~Passage+Treatment,data=rawdata)
Anova_out <- Anova(lmOut_additive,type=2)
Anova_out$`Pr(>F)`
```

```
[1] 7.051564e-17 4.006053e-36           NA
```

```
tidy(Anova_out)
```

```
# A tibble: 3 x 5
  term       sumsq    df statistic   p.value
  <chr>      <dbl> <dbl>     <dbl>     <dbl>
1 Passage    8996.     1      77.1  7.05e-17
2 Treatment 24137.     2     103.   4.01e-36
3 Residuals 41562.   356      NA   NA
```

```
# for comparison, here is the univariable model
lmOut_uni<-lm(Growth~Treatment,data=rawdata)
aOut<-Anova(lmOut_uni,type=3)
a_uni <- anova(lmOut_uni)
a_uni$`Pr(>F)`
```

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

11.4.3.3 Post-hoc analyses

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

```
glht_out <-
  summary(glht(model=lmOut_additive,
               linfct=mcp(Treatment='Dunnett')))
glht_out$test$pvalues
```

```
[1] 1.316836e-12 5.551115e-16
attr(,"error")
[1] 1e-15
```

```
  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  0.0751 .
4-1  6.801105  0.5784996 13.023710  0.0265 *
5-1  8.156143  1.9335375 14.378748  0.0047 **
6-1 14.520106  8.2975011 20.742712 4.4e-08 ***

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
  DescTools::DunnettTest(Growth~Treatment,data=rawdata)
```

```
  Dunnett's test for comparing several treatments with a control :
    95% family-wise confidence level

$Control
                     diff    lwr.ci   upr.ci   pval
AngII-Control     10.40895  6.996811 13.82109  1e-10 ***
Whatever-Control 20.05199 16.639849 23.46413 <2e-16 ***

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
  pairwise.t.test(rawdata$Growth,rawdata$Treatment,p.adjust.method = 'n')
```

```
    Pairwise comparisons using t tests with pooled SD

data:  rawdata$Growth and rawdata$Treatment

         Control AngII
AngII    5.1e-11 -
Whatever < 2e-16 1.0e-09

P value adjustment method: none
```

```
  # mean(rawdata$Growth[which(rawdata$Passage==1 &
                              # rawdata$Treatment=='Control')])
  anova_out$'Pr(>F)'
```

```
[1] 1.257266e-11           NA
```

```
  #aOut$`Sum Sq`
  summary(lmOut_additive)
```

```

Call:
lm(formula = Growth ~ Passage + Treatment, data = rawdata)

Residuals:
    Min      1Q  Median      3Q     Max
-32.407  -7.793  -0.281   7.255  32.283
```

```
Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       110.6802     1.5280  72.432  < 2e-16 ***
Passage             2.9271     0.3334   8.778  < 2e-16 ***
TreatmentAngII     10.4089     1.3949   7.462 6.59e-13 ***
TreatmentWhatever  20.0520     1.3949  14.375  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.8 on 356 degrees of freedom
Multiple R-squared:  0.4436,    Adjusted R-squared:  0.4389
F-statistic:  94.6 on 3 and 356 DF,  p-value: < 2.2e-16
```

```
  (result<-tibble(predictor=rownames(aOut),
                  p=formatP(aOut$'Pr(>F)',ndigits=5)))
```

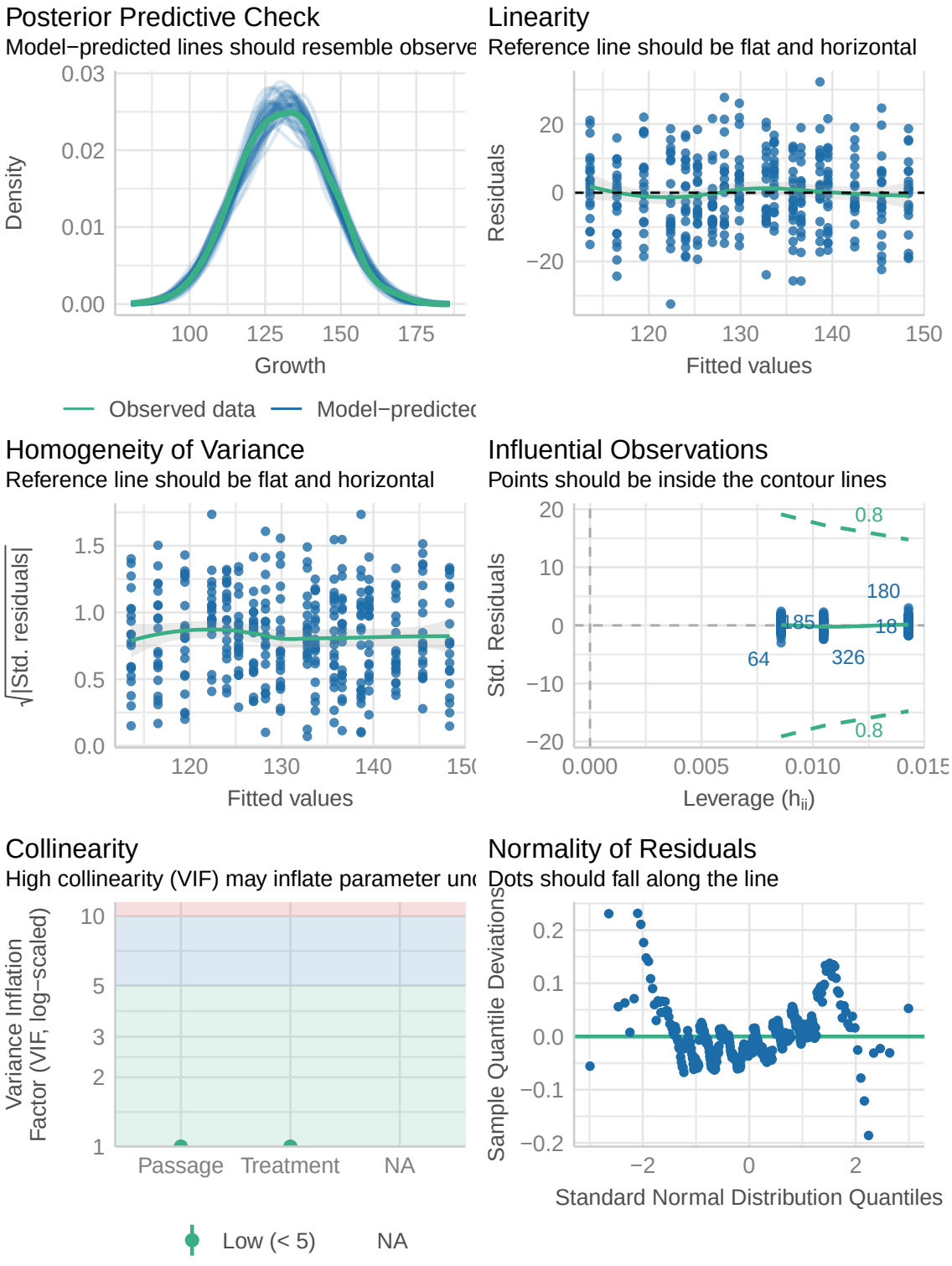
```
# A tibble: 3 x 2
  predictor   p
  <chr>       <chr>
1 (Intercept) "0.00001"
2 Treatment   "0.00001"
3 Residuals   "    NA"
```

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

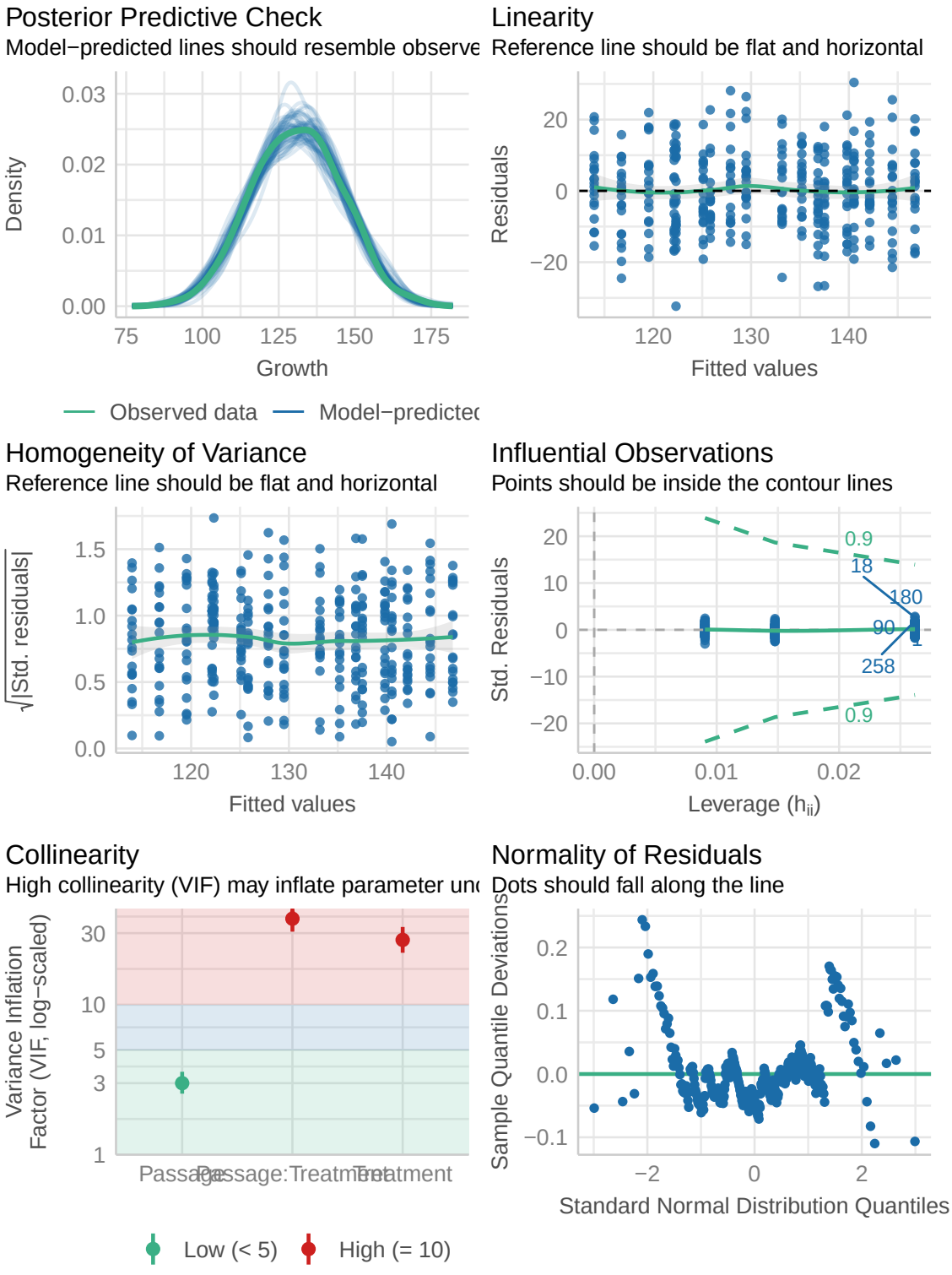
```
# A tibble: 3 x 5
  term           sumsq    df statistic    p.value
  <chr>          <dbl> <dbl>     <dbl>      <dbl>
1 (Intercept) 1754743.     1   12391.   2.91e-279
2 Treatment     24137.     2      85.2  5.55e- 31
3 Residuals     50558.   357      NA    NA
```

11.4.4 Model exploration with package performance

```
# x11() #interactive only!

# from package performance
check_model(lmOut_additive)
```




```
check_model(lmOut_interaction)
```


Posterior Predictive Check
Model−predicted lines should resemble observe
Density
0.03
0.02
0.01
0.00
75
100
125
150
175
Growth
Observed data
Model−predicted
Linearity
Reference line should be flat and horizontal
Residuals
20
0
−20
120
130
140
Fitted values
Homogeneity of Variance
Reference line should be flat and horizontal
√|Std. residuals|
1.5
1.0
0.5
0.0
120
130
140
Fitted values
Influential Observations
Points should be inside the contour lines
Std. Residuals
20
10
0
−10
−20
0.9
18
180
90
1
258
0.9
0.00
0.01
0.02
Leverage (hii)
Collinearity
High collinearity (VIF) may inflate parameter unc
Variance Inflation Factor (VIF, log−scaled)
30
10
5
3
1
Passage
Passage:Treatment
Treatment
Low (< 5)
High (= 10)
Normality of Residuals
Dots should fall along the line
Sample Quantile Deviations
0.2
0.1
0.0
−0.1
−2
0
2
Standard Normal Distribution Quantiles

```
dev.off()
```

```
null device
           1
```

12 Interaction in linear models

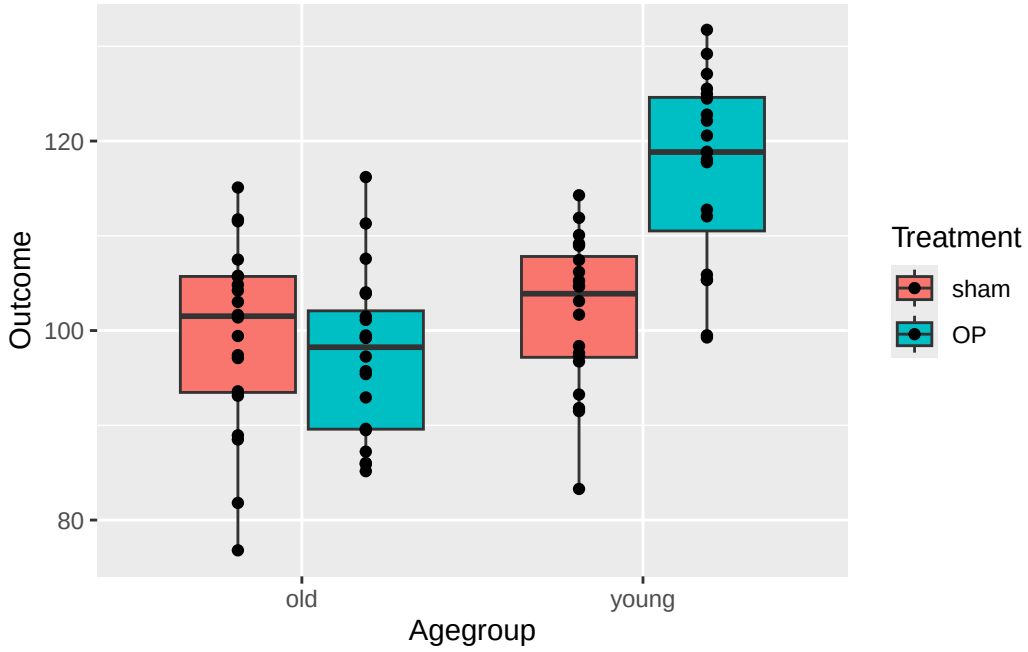
```
pacman::p_load(conflicted,tidyverse,car,multcomp,wrappedtools, broom)
conflicts_prefer(dplyr::select)
```

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

For a better understanding, data with defined effect size and interactions will be simulated and analyzed.

12.1 No age effect, no treatment effect, interaction treatment*agegroup

```
set.seed(101)
rawdata <- tibble(
  Agegroup=factor(
    rep(c('young','old'),each=40),
    levels=c('old','young')),
  Treatment=factor(
    rep(c('sham','OP'),40),
    levels = c('sham','OP'))) |>
  mutate(Outcome=rnorm(n = 80,mean = 100,sd = 10)+
           ((Treatment=='OP')*
              (Agegroup=='young'))*20)
ggplot(rawdata,aes(x=Agegroup,y=Outcome,
                   fill=Treatment))+
  geom_boxplot()+
  geom_point(position=position_dodge(width=.75))
```




```
lmout <- lm(Outcome~Agegroup*Treatment,
            data = rawdata)
tidy(lmout) |>
  select(1:2)
```

```
# A tibble: 4 x 2
  term                      estimate
  <chr>                        <dbl>
1 (Intercept)                  99.5
2 Agegroupyoung                 2.41
3 TreatmentOP                  -2.04
4 Agegroupyoung:TreatmentOP    17.3
```

```
anova(lmout) |>  # this is WRONG!!!
  tidy() |> slice(1:3) |>
  mutate(p.value=formatP(p.value,ndigits=3, mark=TRUE))
```

```
# A tibble: 3 x 6
  term                 df sumsq meansq statistic p.value
  <chr>             <int> <dbl>  <dbl>     <dbl> <chr>
1 Agegroup              1 2443.  2443.      29.2 0.001 ***
2 Treatment             1  872.   872.      10.4 0.002 **
3 Agegroup:Treatment    1 1494.  1494.      17.9 0.001 ***
```

```r
  Anova(lmout,type = 3) |>
    tidy() |> slice(1:4) |>
    mutate(p.value=formatP(p.value,ndigits=3, mark=TRUE))
```

```
# A tibble: 4 x 5
  term                  sumsq    df statistic p.value
  <chr>                 <dbl> <dbl>     <dbl> <chr>
1 (Intercept)        197833.      1  2368.    0.001 ***
2 Agegroup               58.0     1     0.695 0.407 n.s.
3 Treatment              41.7     1     0.499 0.482 n.s.
4 Agegroup:Treatment   1494.      1    17.9   0.001 ***
```

```r
  summary(glht(model=lmout,
               linfct=mcp(Treatment='Tukey')))
```

```
Warning in mcp2matrix(model, linfct = linfct): covariate interactions found --
default contrast might be inappropriate
```

```
     Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Fit: lm(formula = Outcome ~ Agegroup * Treatment, data = rawdata)

Linear Hypotheses:
               Estimate Std. Error t value Pr(>|t|)
OP - sham == 0   -2.042      2.890  -0.707    0.482
(Adjusted p values reported -- single-step method)
```

```r
  summary(glht(model=lmout,
               linfct=mcp(Agegroup='Tukey')))
```

```
Warning in mcp2matrix(model, linfct = linfct): covariate interactions found --
default contrast might be inappropriate
```

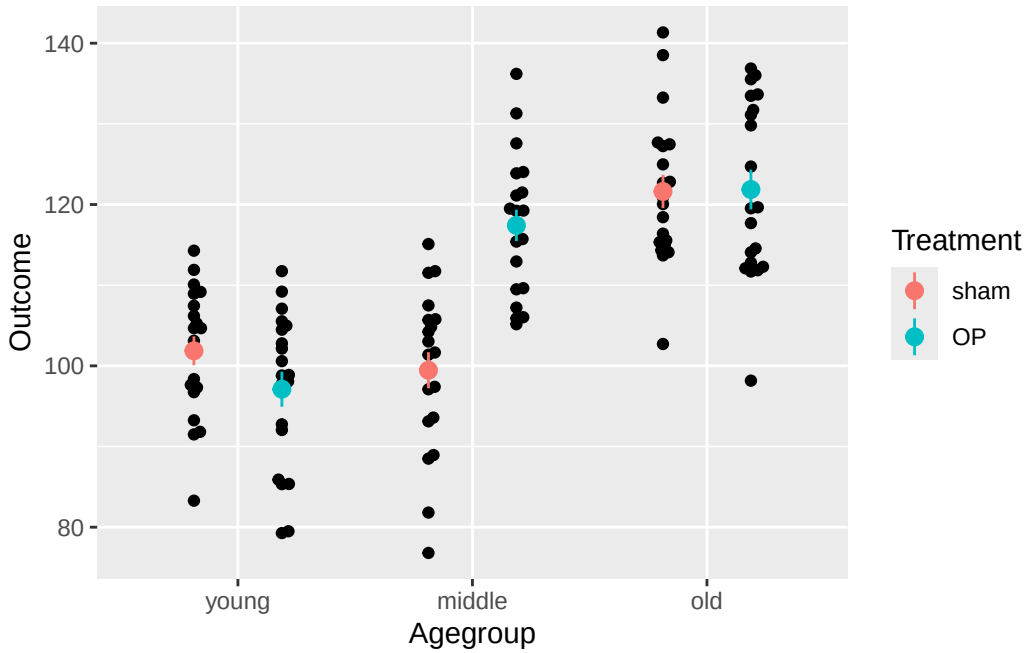
```
     Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts
```

```
Fit: lm(formula = Outcome ~ Agegroup * Treatment, data = rawdata)

Linear Hypotheses:
                 Estimate Std. Error t value Pr(>|t|)
young - old == 0    2.409      2.890   0.834    0.407
(Adjusted p values reported -- single-step method)
```

12.2 Age effect, no treatment effect, interaction treatment*agegroup

```
set.seed(101)
rawdata <- tibble(
  Agegroup=factor(
    rep(c('young','middle','old'),each=40),
    levels=c('young','middle','old')),
  Treatment=factor(
    rep(c('sham','OP'),60),
    levels = c('sham','OP')),
  Outcome=rnorm(120,100,10)+
    (Treatment=='OP')*
    (Agegroup=='middle')*20+
    (Agegroup=='old')*20)
ggplot(rawdata,aes(x=Agegroup,y=Outcome,
                   fill=Treatment))+
  # geom_boxplot()+
  ggbeeswarm::geom_beeswarm(dodge.width = .75)+
  stat_summary(aes(color=Treatment),
               position=position_dodge(width = .75))
```

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



```
lmout <- lm(Outcome~Agegroup*Treatment,
            data = rawdata)
tidy(lmout) |>
  select(1:2)
```

```
# A tibble: 6 x 2
  term                        estimate
  <chr>                          <dbl>
1 (Intercept)                   102.
2 Agegroupmiddle                 -2.41
3 Agegroupold                    19.8
4 TreatmentOP                    -4.76
5 Agegroupmiddle:TreatmentOP     22.7
6 Agegroupold:TreatmentOP         5.01
```

```
anova(lmout) |>  # this is WRONG!!!
  tidy() |> slice(1:3) |>
  mutate(p.value=formatP(p.value,ndigits=3, mark=TRUE))
```

```
# A tibble: 3 x 6
  term                  df  sumsq meansq statistic p.value
  <chr>              <int>  <dbl>  <dbl>     <dbl> <chr>
1 Agegroup               2 10030.  5015.      55.6  0.001 ***
2 Treatment              1   603.   603.       6.69 0.011 *
3 Agegroup:Treatment     2  2848.  1424.      15.8  0.001 ***
```

```
Anova(lmout,type = 3) |>
  tidy() |> slice(1:4) |>
  mutate(p.value=formatP(p.value,ndigits=3, mark=TRUE))
```

```
# A tibble: 4 x 5
  term                  sumsq    df statistic p.value
  <chr>                 <dbl> <dbl>     <dbl> <chr>
1 (Intercept)         207533.     1   2301.   0.001 ***
2 Agegroup              5913.     2     32.8  0.001 ***
3 Treatment              226.     1      2.51 0.116 n.s.
4 Agegroup:Treatment    2848.     2     15.8  0.001 ***
```

```
summary(glht(model=lmout,
             linfct=mcp(Treatment='Tukey')))
```

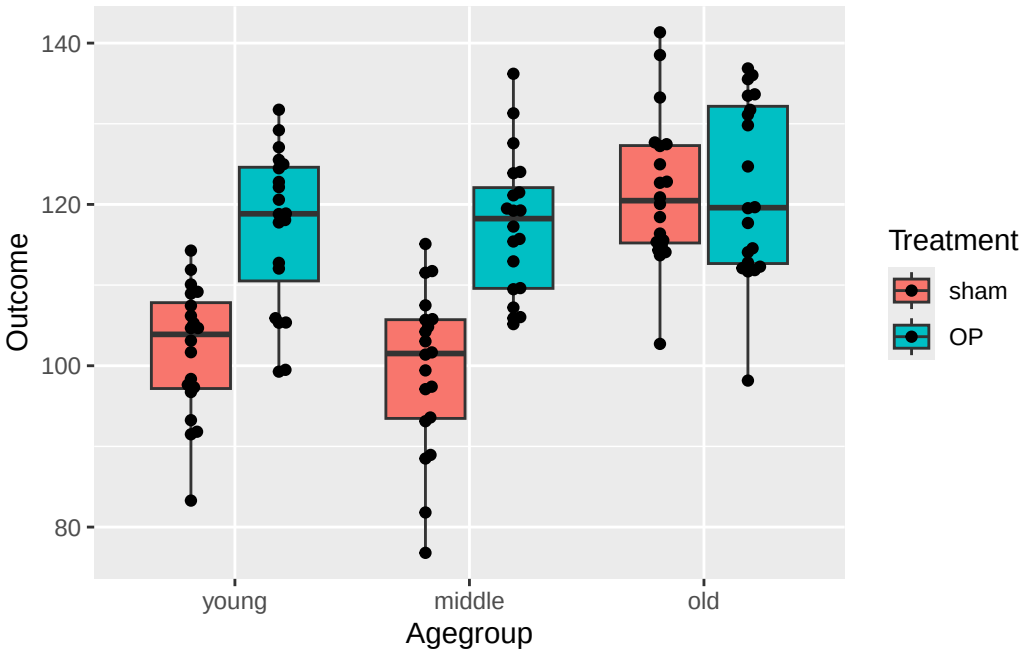
```
Warning in mcp2matrix(model, linfct = linfct): covariate interactions found --
default contrast might be inappropriate


     Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Fit: lm(formula = Outcome ~ Agegroup * Treatment, data = rawdata)

Linear Hypotheses:
               Estimate Std. Error t value Pr(>|t|)
OP - sham == 0   -4.755      3.003  -1.583    0.116
(Adjusted p values reported -- single-step method)
```

```r
summary(glht(model=lmout,
             linfct=mcp(Agegroup='Tukey')))
```

```
Warning in mcp2matrix(model, linfct = linfct): covariate interactions found --
default contrast might be inappropriate


     Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Fit: lm(formula = Outcome ~ Agegroup * Treatment, data = rawdata)

Linear Hypotheses:
                     Estimate Std. Error t value Pr(>|t|)
middle - young == 0   -2.409      3.003  -0.802    0.702
old - young == 0      19.750      3.003   6.576   <1e-05 ***
old - middle == 0     22.159      3.003   7.378   <1e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)
```

12.3 Age effect, treatment effect, interaction treatment*agegroup

```
set.seed(101)
rawdata <- tibble(
  Agegroup=factor(
    rep(c('young','middle','old'),each=40),
    levels=c('young','middle','old')),
  Treatment=factor(
    rep(c('sham','OP'),60),
    levels = c('sham','OP'))) |>
  mutate(Outcome=rnorm(120,100,10)+
           (Treatment=='OP')*
           # (Agegroup %in% c('young','middle'))*
           (Agegroup!='old')*20+
           (Agegroup=='old')*20)
ggplot(rawdata,aes(x=Agegroup,y=Outcome,
                   fill=Treatment))+
  geom_boxplot()+
  ggbeeswarm::geom_beeswarm(dodge.width = .75)
```




```
suppressWarnings(
  ggplot(rawdata,aes(x=as.numeric(Agegroup),y=Outcome,fill=Treatment))+
  ggbeeswarm::geom_beeswarm(aes(shape=Treatment),
```

```
                              alpha=.5, dodge.width = .15)+
    geom_smooth()+
    scale_x_continuous("Agegroup",breaks=1:3,
                       labels=c('young','middle','old')))
```

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

```
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: pseudoinverse used at 0.99
```

```
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: neighborhood radius 2.01
```

```
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: reciprocal condition number 2.0996e-16
```

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

```
Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object)), : pseudoinverse used at
0.99
```

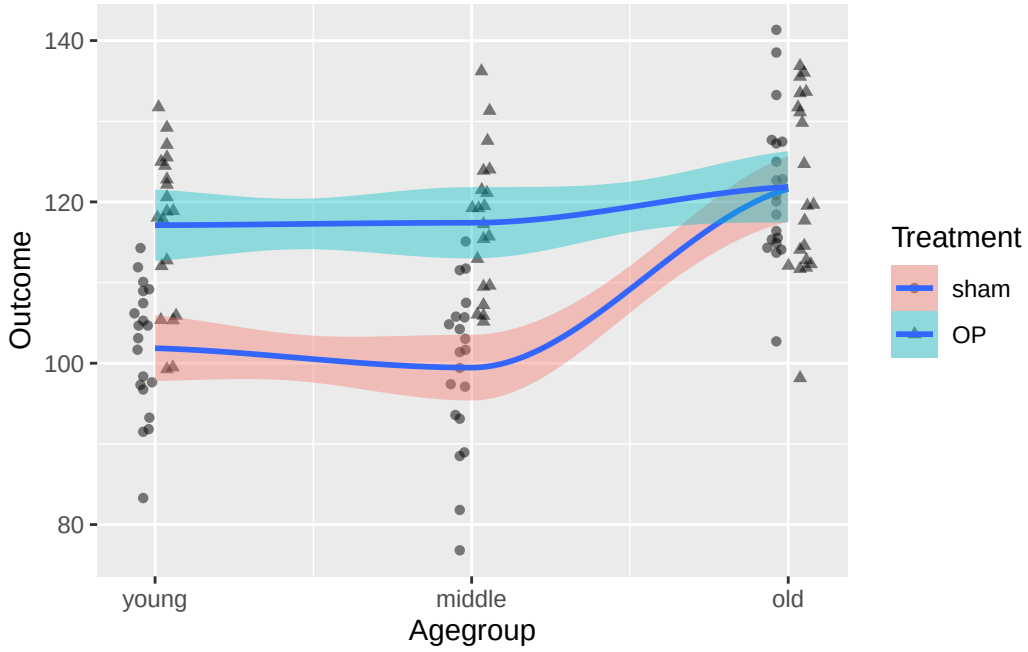
```
Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object)), : neighborhood radius
2.01
```

```
Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object)), : reciprocal condition
number 2.0996e-16
```

```
Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object)), : There are other near
singularities as well. 4.0401
```

```
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: pseudoinverse used at 0.99
```

```
Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: neighborhood radius 2.01

Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametric,
: reciprocal condition number 2.0996e-16

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

Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object)), : pseudoinverse used at
0.99

Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object)), : neighborhood radius
2.01

Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object)), : reciprocal condition
number 2.0996e-16

Warning in predLoess(object$y, object$x, newx = if (is.null(newdata)) object$x
else if (is.data.frame(newdata))
as.matrix(model.frame(delete.response(terms(object)), : There are other near
singularities as well. 4.0401
```




```
lmout <- lm(Outcome~Agegroup*Treatment,
            data = rawdata)
tidy(lmout) |>
  select(1:2)
```

```
# A tibble: 6 x 2
  term                        estimate
  <chr>                          <dbl>
1 (Intercept)                   102.
2 Agegroupmiddle                 -2.41
3 Agegroupold                    19.8
4 TreatmentOP                    15.2
5 Agegroupmiddle:TreatmentOP      2.71
6 Agegroupold:TreatmentOP       -15.0
```

```
anova(lmout) |>  # this is WRONG!!!
  tidy() |> slice(1:3) |>
  mutate(p.value=formatP(p.value,ndigits=3, mark=TRUE))
```

```
# A tibble: 3 x 6
  term                 df sumsq meansq statistic p.value
  <chr>             <int> <dbl>  <dbl>     <dbl> <chr>
1 Agegroup              2 4377.  2188.      24.3 0.001 ***
2 Treatment             1 3730.  3730.      41.4 0.001 ***
```

```
3 Agegroup:Treatment     2 1819.    910.       10.1 0.001 ***
```

```
Anova(lmout,type = 3) |>
  tidy() |> slice(1:4) |>
  mutate(p.value=formatP(p.value,ndigits=3, mark=TRUE))
```

```
# A tibble: 4 x 5
  term                 sumsq    df statistic p.value
  <chr>                <dbl> <dbl>     <dbl> <chr>
1 (Intercept)        207533.     1    2301.  0.001 ***
2 Agegroup             5913.     2      32.8 0.001 ***
3 Treatment            2324.     1      25.8 0.001 ***
4 Agegroup:Treatment   1819.     2      10.1 0.001 ***
```

```
summary(glht(model=lmout,
             linfct=mcp(Treatment='Tukey')))
```

```
Warning in mcp2matrix(model, linfct = linfct): covariate interactions found --
default contrast might be inappropriate


     Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Fit: lm(formula = Outcome ~ Agegroup * Treatment, data = rawdata)

Linear Hypotheses:
               Estimate Std. Error t value Pr(>|t|)
OP - sham == 0   15.245      3.003   5.076 1.52e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)
```

```
summary(glht(model=lmout,
             linfct=mcp(Agegroup='Tukey')))
```

```
Warning in mcp2matrix(model, linfct = linfct): covariate interactions found --
default contrast might be inappropriate
```

```
     Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Fit: lm(formula = Outcome ~ Agegroup * Treatment, data = rawdata)

Linear Hypotheses:
                     Estimate Std. Error t value Pr(>|t|)
middle - young == 0   -2.409      3.003  -0.802    0.702
old - young == 0      19.750      3.003   6.576   <1e-05 ***
old - middle == 0     22.159      3.003   7.378   <1e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)
```

Ignoring the interaction in the model is no solution. Effect sizes will be wrongly estimated:

```
# falsch!!!
lmout <- lm(Outcome~Agegroup+Treatment,
            data = rawdata)
tidy(lmout) |>
  select(1:2)
```

```
# A tibble: 4 x 2
  term           estimate
  <chr>             <dbl>
1 (Intercept)      104.
2 Agegroupmiddle    -1.05
3 Agegroupold       12.3
4 TreatmentOP       11.2
```

```
anova(lmout) |>  # this is WRONG!!!
  tidy() |> slice(1:2) |>
  mutate(p.value=formatP(p.value,ndigits=3, mark=TRUE))
```

```
# A tibble: 2 x 6
  term         df sumsq meansq statistic p.value
  <chr>     <int> <dbl>  <dbl>     <dbl> <chr>
1 Agegroup      2 4377.  2188.      21.0 0.001 ***
2 Treatment     1 3730.  3730.      35.8 0.001 ***
```

```r
  Anova(lmout,type = 2) |>
    tidy() |> slice(1:2) |>
    mutate(p.value=formatP(p.value,ndigits=3, mark=TRUE))
```

```
# A tibble: 2 x 5
  term      sumsq    df statistic p.value
  <chr>     <dbl> <dbl>     <dbl> <chr>
1 Agegroup  4377.     2      21.0 0.001 ***
2 Treatment 3730.     1      35.8 0.001 ***
```

12.4 How to specify interaction in multivariable models

```
# all possible interactions
(lm_out <- lm(mpg~(wt*gear*factor(am)*cyl),
              data=mtcars))
```

```
Call:
lm(formula = mpg ~ (wt * gear * factor(am) * cyl), data = mtcars)

Coefficients:
            (Intercept)                      wt                    gear
                456.687                -158.453                -146.900
           factor(am)1                     cyl                 wt:gear
               -180.550                 -16.129                  53.938
        wt:factor(am)1       gear:factor(am)1                  wt:cyl
                 63.239                 102.915                  10.048
            gear:cyl        factor(am)1:cyl     wt:gear:factor(am)1
                 5.764                 -33.954                 -37.005
          wt:gear:cyl      wt:factor(am)1:cyl    gear:factor(am)1:cyl
                -3.593                   8.905                   3.456
wt:gear:factor(am)1:cyl
                    NA
```

```
# only two-way interactions
(lm_out <- lm(mpg~(wt+gear+factor(am)+cyl)^2,
              data=mtcars))
```

```
Call:
lm(formula = mpg ~ (wt + gear + factor(am) + cyl)^2, data = mtcars)

Coefficients:
     (Intercept)                wt              gear       factor(am)1
        57.58968          -17.65458           -7.80045           13.91506
             cyl           wt:gear     wt:factor(am)1            wt:cyl
         3.13308            4.90387          -11.51010            0.06975
gear:factor(am)1          gear:cyl    factor(am)1:cyl
         2.23288           -1.43230            2.15419
```

```
#some seected interactions
(lm_out <- lm(mpg~wt*(gear+factor(am)+cyl),
```

```
                data=mtcars))
```

```
Call:
lm(formula = mpg ~ wt * (gear + factor(am) + cyl), data = mtcars)

Coefficients:
   (Intercept)              wt            gear     factor(am)1             cyl
      50.53760        -8.43482        -5.80119        21.62110        -1.04500
       wt:gear  wt:factor(am)1          wt:cyl
       2.03447        -7.59198        -0.00499
```

13 Logistic regression

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

```
  pacman::p_load(car, wrappedtools, tidyverse, ggbeeswarm,
                 rpart, rpart.plot, pROC )
  rawdata <- infert |>
    as_tibble()  |>
    select(-contains("stratum"))
  head(rawdata)
```

```
# A tibble: 6 x 6
  education   age parity induced  case spontaneous
  <fct>     <dbl>  <dbl>   <dbl> <dbl>       <dbl>
1 0-5yrs       26      6       1     1           2
2 0-5yrs       42      1       1     1           0
3 0-5yrs       39      6       2     1           0
4 0-5yrs       34      4       2     1           0
5 6-11yrs      35      3       1     1           1
6 6-11yrs      36      4       2     1           1
```

13.1 Data preparation

Before the analysis, the data set is cleaned and prepared for the analysis. The age variable is transformed into pentayears, and the parity variable is lumped into two categories. Education is reversed and transformed into a factor.

```
  rawdata$age-rawdata$age%%5
```

```
  [1] 25 40 35 30 35 35 20 30 20 25 25 35 30 25 30 25 30 25 25 40 40 35 25 35 25
 [26] 40 35 30 25 30 30 30 40 30 35 35 35 30 30 25 35 35 40 35 30 35 25 25 25 35
 [51] 20 35 25 25 25 35 25 25 25 25 35 25 30 30 25 30 20 25 35 25 30 25 30 35 25
 [76] 30 30 25 30 30 35 25 20 25 40 35 30 35 35 20 30 20 25 25 35 30 25 30 25 30
[101] 25 25 40 40 35 25 35 25 40 35 30 25 30 30 30 40 30 35 35 35 30 30 25 35 35
[126] 40 35 30 35 25 25 25 35 20 35 25 25 25 35 25 25 25 25 35 25 30 30 25 30 20
[151] 25 35 25 30 25 30 25 30 30 25 30 30 35 25 20 25 40 35 30 35 35 20 30 20 25
```

```
[176] 25 35 30 25 30 25 30 25 25 40 40 35 25 35 25 40 35 30 25 30 30 30 40 30 35
[201] 35 35 30 30 25 35 35 40 35 30 35 25 25 25 35 20 35 25 25 25 35 25 25 25 25
[226] 35 25 30 30 25 30 20 25 35 25 30 25 30 35 25 30 30 25 30 30 35 25 20
```

```
rawdata$age/5
```

```
 [1] 5.2 8.4 7.8 6.8 7.0 7.2 4.6 6.4 4.2 5.6 5.8 7.4 6.2 5.8 6.2 5.4 6.0 5.2
[19] 5.0 8.8 8.0 7.0 5.6 7.2 5.4 8.0 7.6 6.8 5.6 6.0 6.4 6.8 8.4 6.4 7.8 7.0
[37] 7.2 6.8 6.0 5.6 7.8 7.0 8.2 7.4 6.0 7.4 5.6 5.4 5.2 7.6 4.8 7.2 5.4 5.6
[55] 5.8 7.2 5.6 5.6 5.6 5.4 7.0 5.0 6.8 6.2 5.2 6.4 4.2 5.6 7.4 5.0 6.4 5.0
[73] 6.2 7.6 5.2 6.2 6.2 5.0 6.2 6.8 7.0 5.8 4.6 5.2 8.4 7.8 6.8 7.0 7.2 4.6
[91] 6.4 4.2 5.6 5.8 7.4 6.2 5.8 6.2 5.4 6.0 5.2 5.0 8.8 8.0 7.0 5.6 7.2 5.4
[109] 8.0 7.6 6.8 5.6 6.0 6.4 6.8 8.4 6.4 7.8 7.0 7.2 6.8 6.0 5.6 7.8 7.0 8.2
[127] 7.4 6.0 7.4 5.6 5.4 5.2 7.6 4.8 7.2 5.4 5.6 5.8 7.2 5.6 5.6 5.6 5.4 7.0
[145] 5.0 6.8 6.2 5.2 6.4 4.2 5.6 7.4 5.0 6.4 5.0 6.2 5.2 6.2 6.2 5.0 6.2 6.8
[163] 7.0 5.8 4.6 5.2 8.4 7.8 6.8 7.0 7.2 4.6 6.4 4.2 5.6 5.8 7.4 6.2 5.8 6.2
[181] 5.4 6.0 5.2 5.0 8.8 8.0 7.0 5.6 7.2 5.4 8.0 7.6 6.8 5.6 6.0 6.4 6.8 8.4
[199] 6.4 7.8 7.0 7.2 6.8 6.0 5.6 7.8 7.0 8.2 7.4 6.0 7.4 5.6 5.4 5.2 7.6 4.8
[217] 7.2 5.4 5.6 5.8 7.2 5.6 5.6 5.6 5.4 7.0 5.0 6.8 6.2 5.2 6.4 4.2 5.6 7.4
[235] 5.0 6.4 5.0 6.2 7.6 5.2 6.2 6.2 5.0 6.2 6.8 7.0 5.8 4.6
```

```
table(rawdata$parity)
```

```
 1  2  3  4  5  6
99 81 36 18  6  8
```

```
rawdata <- rawdata |>
  mutate(
    case=factor(case),
    induced_f=factor(induced,
                     levels = c('0','1','2'),
                     labels = (c('none','one','two or more'))),
    spontaneous_f=factor(spontaneous),
    `age [pentayears]`=age/5,
    education=forcats::fct_rev(education),
    parity_grp=forcats::fct_lump(as.character(parity),
                                 n = 2, other_level = '>2') |>
      fct_rev())
```

13.2 Build model

The model is built using glm() and the output is extracted and transformed. The model is tested using Anova() and summary(). The results are then prepared for plotting.

```
logreg_out <-glm(case~`age [pentayears]`+education+parity_grp+ induced_f+spontaneous_f,
                 family=binomial(), data=rawdata)
logreg_out
```

```
Call:  glm(formula = case ~ `age [pentayears]` + education + parity_grp +
    induced_f + spontaneous_f, family = binomial(), data = rawdata)

Coefficients:
         (Intercept)    `age [pentayears]`      education6-11yrs
             -5.8727                0.1820                0.4394
     education0-5yrs           parity_grp2           parity_grp1
              0.6082                1.4562                2.7048
        induced_fone  induced_ftwo or more        spontaneous_f1
              1.3591                2.8292                2.0599
      spontaneous_f2
              4.3296

Degrees of Freedom: 247 Total (i.e. Null);  238 Residual
Null Deviance:      316.2
Residual Deviance: 255.9    AIC: 275.9
```

```
#extract/transform model parameters
(ORs <- exp(logreg_out$coefficients))
```

```
         (Intercept)    `age [pentayears]`      education6-11yrs
         0.002815305          1.199607197          1.551793487
     education0-5yrs           parity_grp2           parity_grp1
         1.837105706          4.289689825         14.951228070
        induced_fone  induced_ftwo or more        spontaneous_f1
         3.892652439         16.931760140          7.844857609
      spontaneous_f2
        75.916231594
```

```
(CIs <- exp(confint(logreg_out)))
```

```
Waiting for profiling to be done...
```

```
                            2.5 %       97.5 %
(Intercept)           1.860835e-04   0.03486434
`age [pentayears]`    8.849570e-01   1.63799477
education6-11yrs      8.051279e-01   3.03038853
education0-5yrs       3.735091e-01   8.28017336
parity_grp2           1.689458e+00  11.53626479
parity_grp1           4.663096e+00  53.09682931
induced_fone          1.726419e+00   9.15964545
induced_ftwo or more  4.809607e+00  65.00363723
spontaneous_f1        3.580230e+00  18.15518387
spontaneous_f2        2.133238e+01 311.72957158
```

```
#test model
(Anova_out <- Anova(logreg_out,type = 2) |>
    broom::tidy() |>
    mutate(p.value=formatP(p.value,ndigits = 5)))
```

```
# A tibble: 5 x 4
  term                statistic    df p.value
  <chr>                   <dbl> <dbl> <chr>
1 `age [pentayears]`       1.37     1 0.24200
2 education                1.89     2 0.38838
3 parity_grp              22.8      2 0.00001
4 induced_f               22.1      2 0.00002
5 spontaneous_f           60.2      2 0.00001
```

```
## test each OR
(sum_out <- summary(logreg_out))
```

```
Call:
glm(formula = case ~ `age [pentayears]` + education + parity_grp +
    induced_f + spontaneous_f, family = binomial(), data = rawdata)

Coefficients:
                    Estimate Std. Error z value Pr(>|z|)
(Intercept)          -5.8727     1.3291  -4.419 9.93e-06 ***
`age [pentayears]`    0.1820     0.1564   1.163  0.24467
education6-11yrs      0.4394     0.3370   1.304  0.19223
education0-5yrs       0.6082     0.7781   0.782  0.43442
parity_grp2           1.4562     0.4880   2.984  0.00284 **
parity_grp1           2.7048     0.6186   4.373 1.23e-05 ***
induced_fone          1.3591     0.4236   3.208  0.00134 **
```

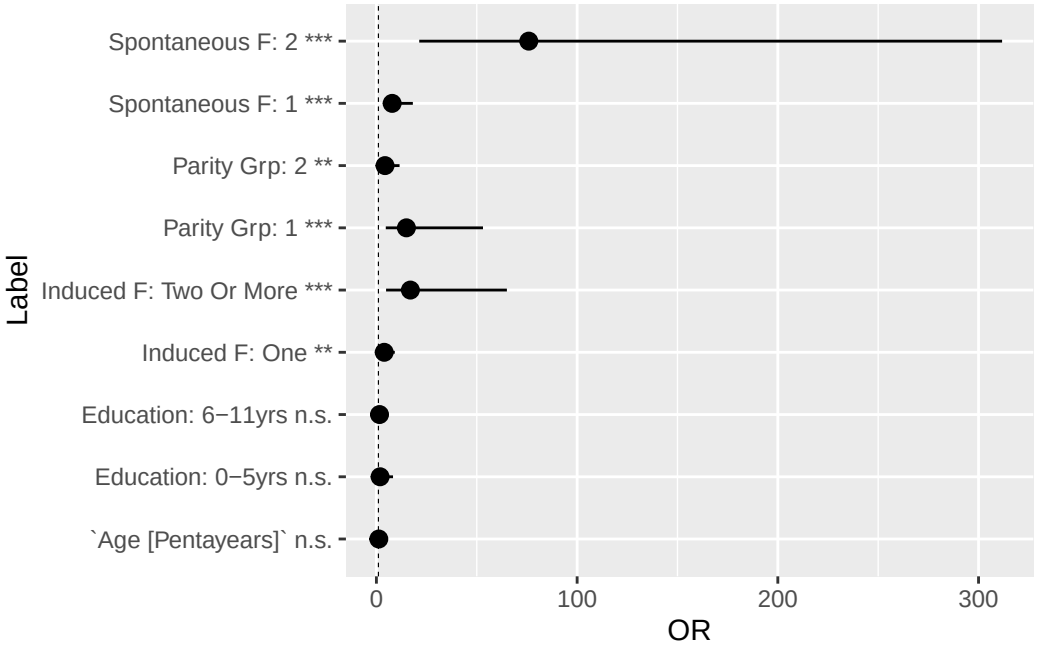
```
induced_ftwo or more   2.8292     0.6610   4.280 1.87e-05 ***
spontaneous_f1         2.0599     0.4124   4.994 5.90e-07 ***
spontaneous_f2         4.3296     0.6814   6.354 2.10e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 316.17  on 247  degrees of freedom
Residual deviance: 255.91  on 238  degrees of freedom
AIC: 275.91

Number of Fisher Scoring iterations: 5
```

13.3 Create structure for ggplot

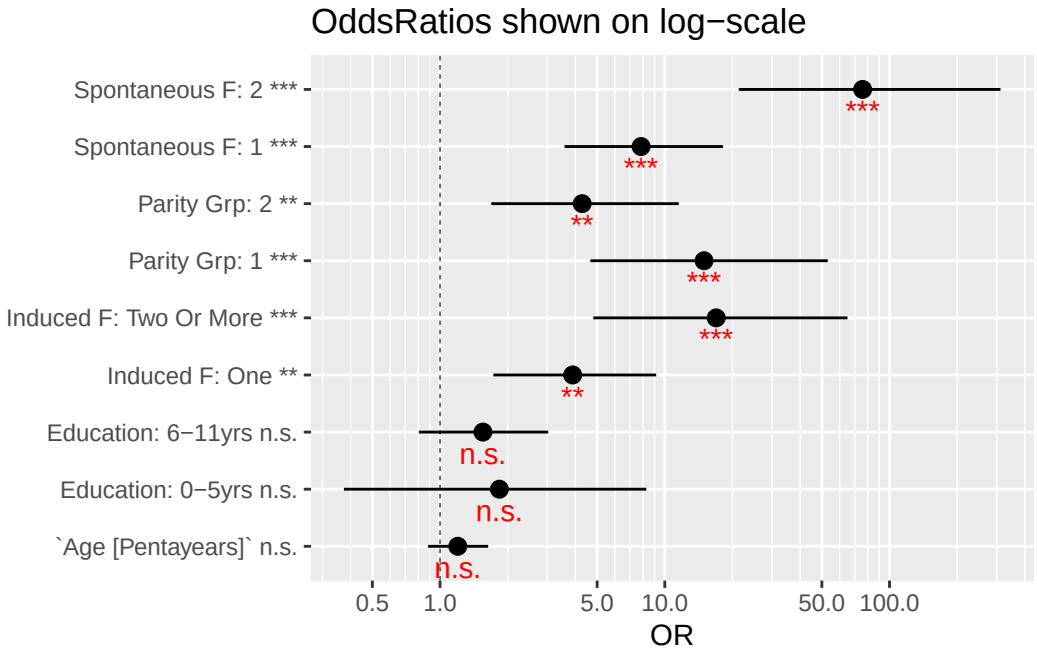
```
OR_plotdata <- tibble(
  Predictor=names(ORs)[-1] |>
    # make names nicer
    str_replace('_',' ') |>
    str_replace_all(c(
      '(grp)(.)'='\\1: \\2',
      '(f)(.)'='\\1: \\2',
      '(n)(\\d)'='\\1: \\2')) |>
    str_to_title(),
  OR=ORs[-1],
  CI_low=CIs[-1,1],
  CI_high=CIs[-1,2],
  p=sum_out$coefficients[-1,4],
  Significance=markSign(p),
  Label=paste(Predictor,Significance))
```

13.4 create forest plot

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




```
baseplot+
  scale_y_log10(breaks=logrange_15,
                minor_breaks=logrange_123456789 )+
  geom_text(aes(label=Significance), vjust=1.5,color='red')+
  ggtitle('OddsRatios shown on log-scale')+
  xlab(NULL)
```



13.5 Create predictions

```
rawdata$pGLM <-
  predict(logreg_out, type = 'response') #predict probability
# run ROC for cutoff
roc_out <- roc(response=rawdata$case,
               predictor=rawdata$pGLM)
```

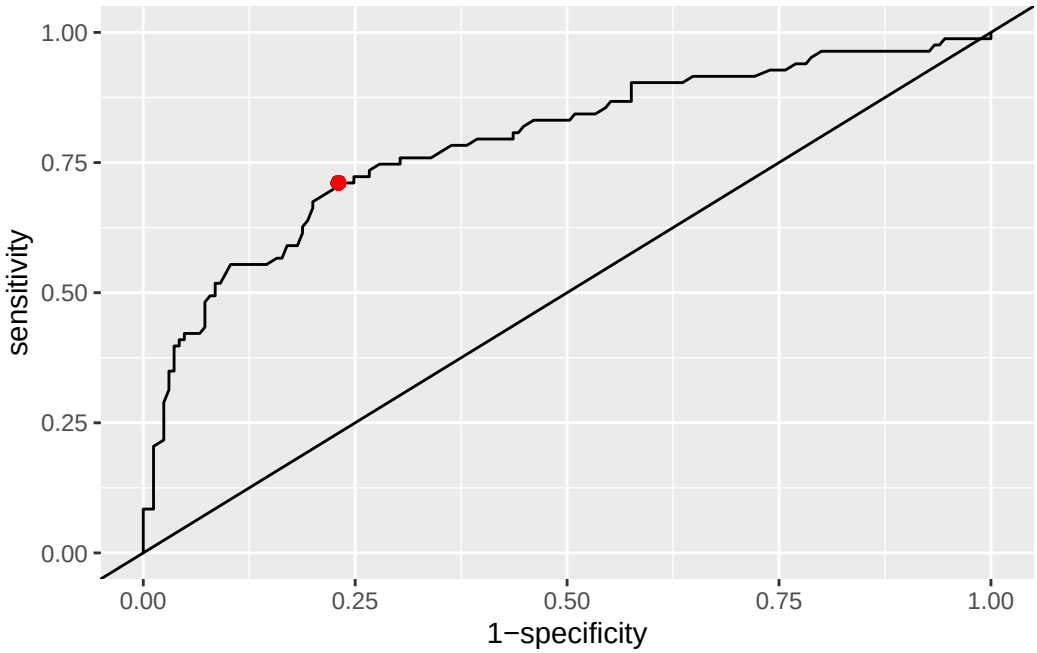
```
Setting levels: control = 0, case = 1
```

```
Setting direction: controls < cases
```

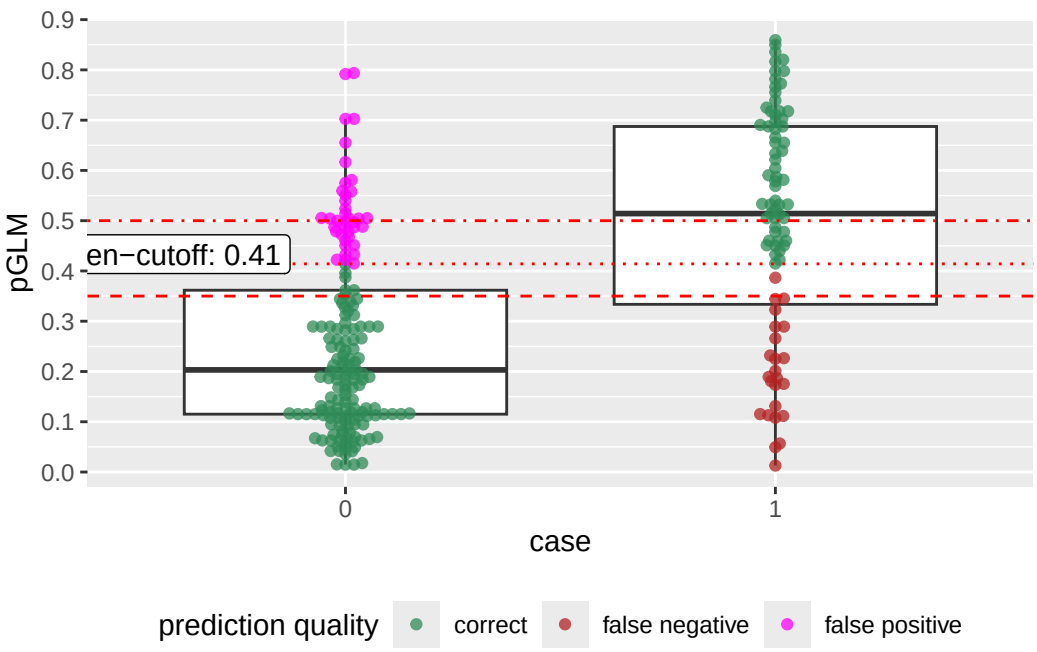
```
youden <- pROC::coords(roc_out,x='best',
                       best.method='youden')
youden
```

```
  threshold specificity sensitivity
1 0.4141249    0.769697   0.7108434
```

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



```
# plot predictions
rawdata |>
  mutate(`prediction quality`=
           case_when(case=="1" &
                       pGLM<youden$threshold ~
                       "false negative",
                     case=="0" &
                       pGLM>=youden$threshold
                     ~ "false positive",
                     .default = 'correct' )) |>
  ggplot(aes(case,pGLM))+
  geom_boxplot(outlier.alpha = 0)+
  scale_y_continuous(breaks=seq(0,1,.1))+
  geom_beeswarm(alpha=.75,
                aes(color=`prediction quality`))+
  scale_color_manual(values=c("seagreen","firebrick","magenta"))+
  geom_hline(yintercept = c(.35, youden$threshold,.5),
             color='red',
             linetype=2:4)+
  annotate(geom = "label",
           x = 1,y=youden$threshold,
           label=paste("Youden-cutoff:",
                       roundR(youden$threshold)),
           hjust=1.2,vjust=0.25)+
  theme(legend.position="bottom")
```



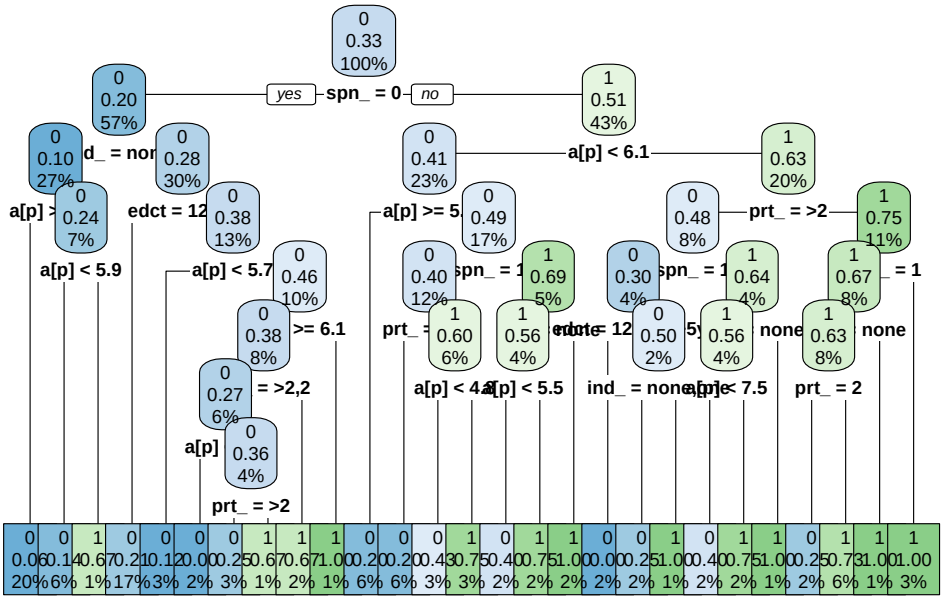
```
ORhuman <-
  paste0(map_chr(ORs,roundR),' (',
         apply(CIs,MARGIN = 1,
               FUN=function(x){
                 paste(roundR(x),collapse=' / ')}), ')')
ORreport <- tibble(Predictor=rownames(CIs)[-1],
                   OR=ORs[-1],
                   low=CIs[-1,1],
                   high=CIs[-1,2],
                   `OR (CI95)`=NA)
ORrounded <- apply(ORreport[,2:4],MARGIN = 1,roundR)
ORreport$`OR (CI95)` <-
  paste0(ORrounded[1,],' (',ORrounded[2,],'/',
         ORrounded[3,],')')
```

13.6 Regression tree as alternative to glm

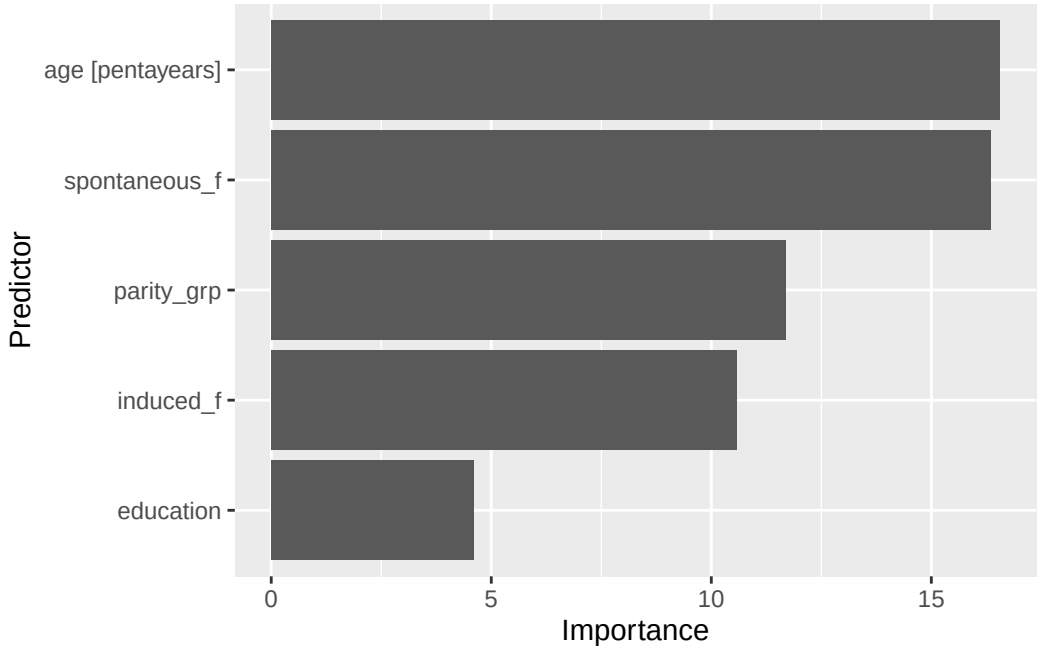
```
cn()
```

```
[1] "education"        "age"              "parity"           "induced"
[5] "case"             "spontaneous"      "induced_f"        "spontaneous_f"
[9] "age [pentayears]" "parity_grp"       "pGLM"
```

```
predvars <- ColSeeker(namepattern =
                        c("penta","edu","_grp","_f"))
rtformula <- paste("case~",
                   paste(predvars$bticked,collapse = "+"))
regtree_out<-rpart(rtformula,
                   minsplit=5,cp=.001,
                   data=rawdata)
rpart.plot(regtree_out,type = 2,tweak=2.0, varlen=4,faclen=5,leaf.round=0)
```




```
importance <-
  as_tibble(regtree_out$variable.importance,
            rownames='Predictor')  |>
  dplyr::rename('Importance'=2)  |>
  mutate(Predictor=fct_reorder(.f = Predictor,
                               .x = Importance,
                               .fun = min))  |>
  arrange(desc(Importance))
importance |>
  ggplot(aes(Predictor,Importance))+
  geom_col()+
  coord_flip()
```




```
rawdata$pRT <- predict(regtree_out)[,2]

#pROC
roc_out_rt <- roc(response=rawdata$case,
  predictor=rawdata$pRT )
```

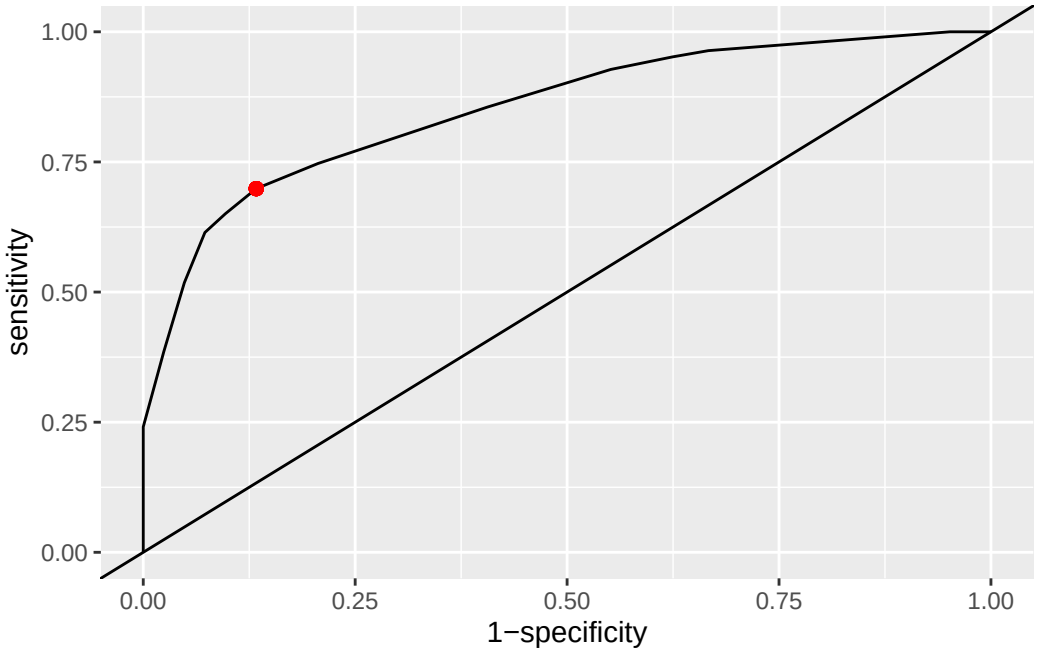
```
Setting levels: control = 0, case = 1
```

```
Setting direction: controls < cases
```

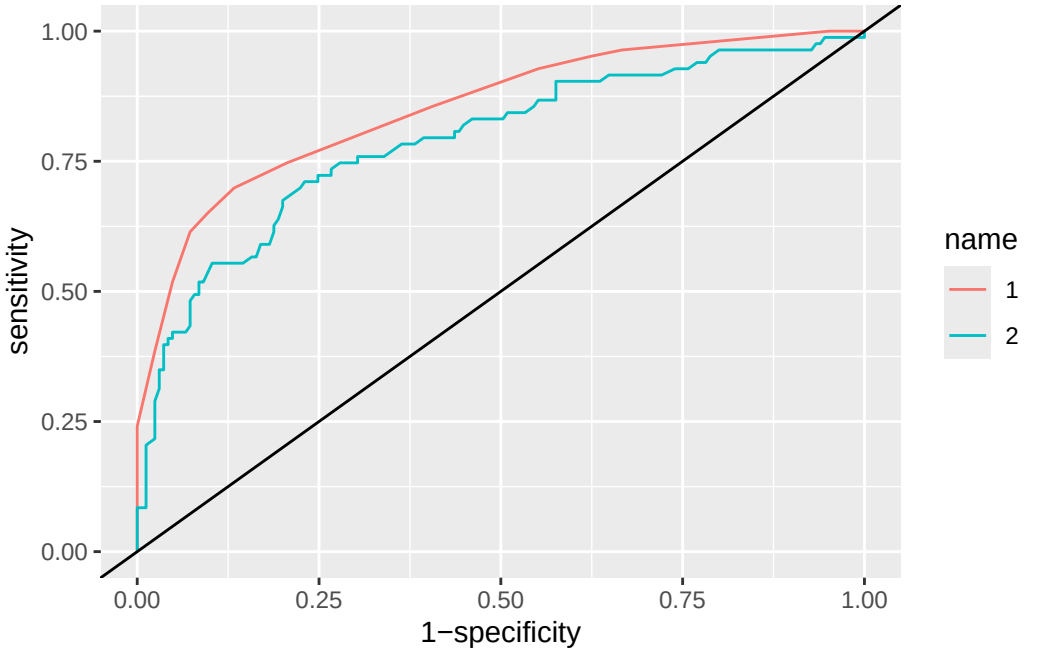
```
youden <- pROC::coords(roc_out_rt,x='best',
                       best.method='youden')
youden
```

```
  threshold specificity sensitivity
1     0.325   0.8666667   0.6987952
```

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

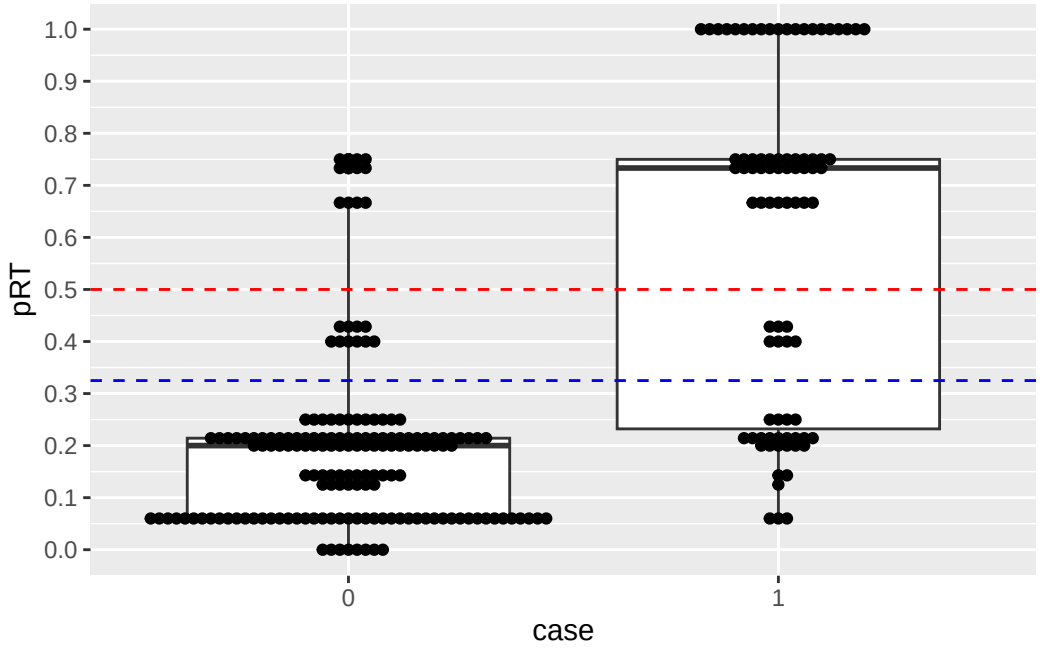



```
ggroc(list(roc_out_rt,roc_out),legacy.axes = T)+
  geom_abline(slope = 1,intercept = 0)
```



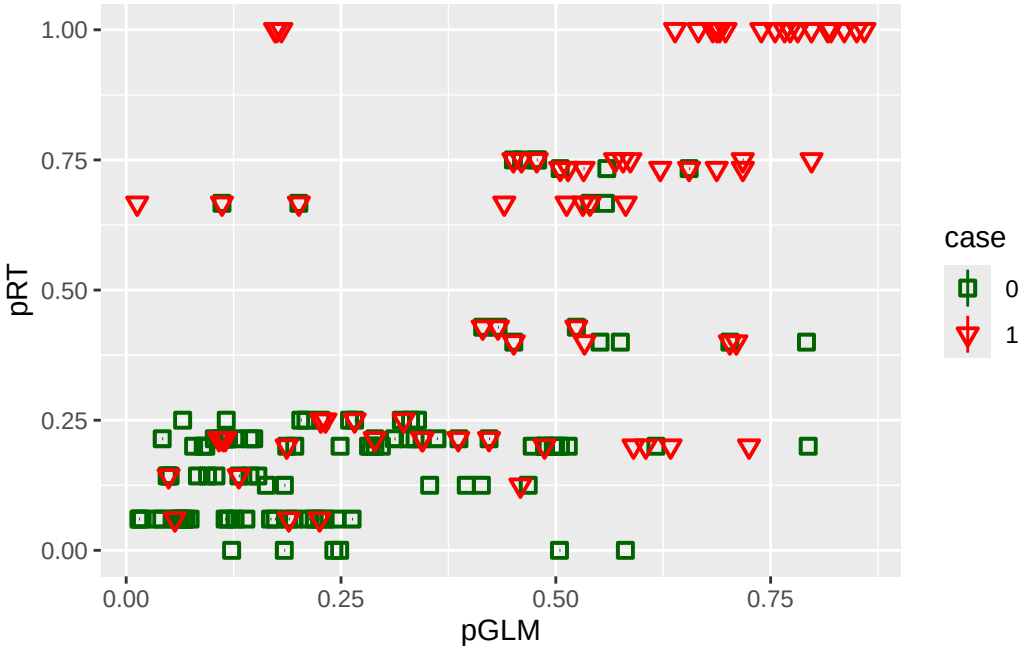

```
ggplot(rawdata,aes(x=case,y=pRT))+
  geom_boxplot(coef=3)+
```

```
  scale_y_continuous(breaks = seq(from = 0,to = 1,by = .1))+
  geom_hline(yintercept = c(.5,youden$threshold),
             color=c('red',"blue"), linetype=2)+
  ggbeeswarm::geom_beeswarm()
```




```
ggplot(rawdata,aes(pGLM,pRT, color=case,shape=case))+
  geom_point(size=2)+
  scale_color_manual(values = c('darkgreen','red'))+
  scale_shape_manual(values = c(0,6))+
  stat_summary(fun.data=mean_cl_boot)
```

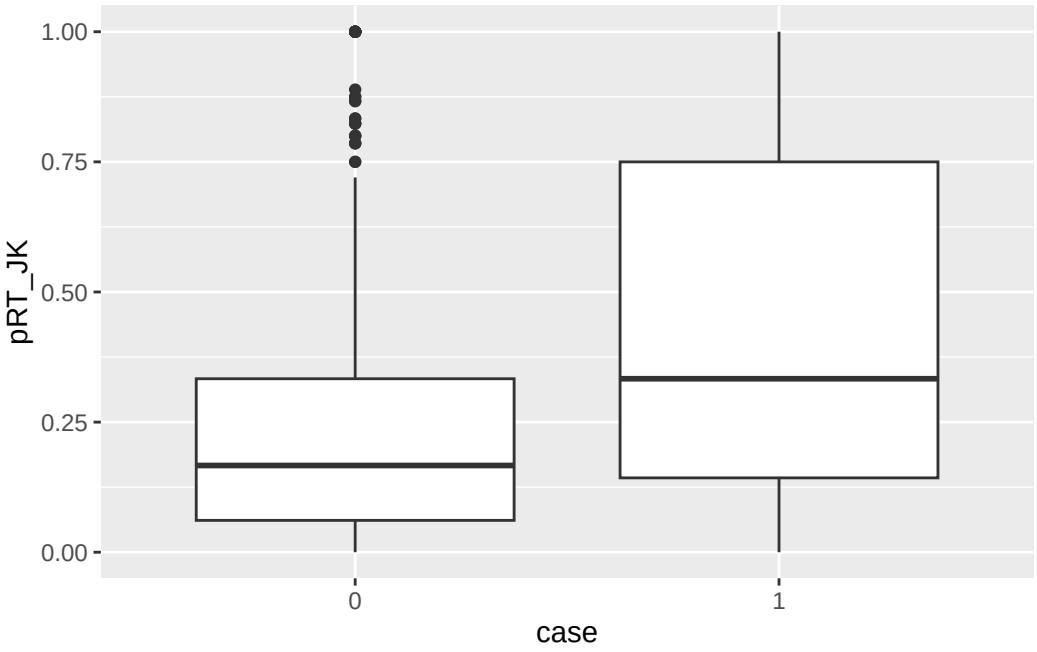
```
Warning: Removed 139 rows containing missing values or values outside the scale range
(`geom_segment()`).
```




```
ggplot(rawdata,aes(x=case,y=pRT))+
  geom_violin()+
  scale_y_continuous(breaks = seq(from = 0,to = 1,by = .1))+
  geom_hline(yintercept = .5,color='red')
```

13.7 Jackknife

```
rawdata$pRT_JK <- NA_real_
rawdata$pGLM_JK <- NA_real_
for(pat_i in 1: nrow(rawdata)){
  tempdata <- rawdata[-pat_i,]
  regtree_out_tmp<-rpart(rtformula,
                         minsplit=5,
                         cp=.001, data=tempdata)
  rawdata$pRT_JK[pat_i] <-
    predict(regtree_out_tmp,
            newdata = rawdata[pat_i,])[,2]

  glm_out_tmp<-glm(rtformula,
                   family = binomial(), data=tempdata)
  rawdata$pGLM_JK[pat_i] <-
    predict(glm_out_tmp,newdata = rawdata[pat_i,],
            type="response")
  }
ggplot(rawdata,aes(case,pRT_JK))+
  geom_boxplot()
```




```
rawdata |>
  dplyr::select(case,pGLM,pGLM_JK, pRT_JK, pRT) |>
```

```
  pivot_longer(cols = c(pGLM,pGLM_JK, pRT_JK,pRT),
               names_to = 'Analysis',
               values_to = 'pAffected') |>
  ggplot(aes(case,pAffected,fill=Analysis))+
  geom_boxplot()
```