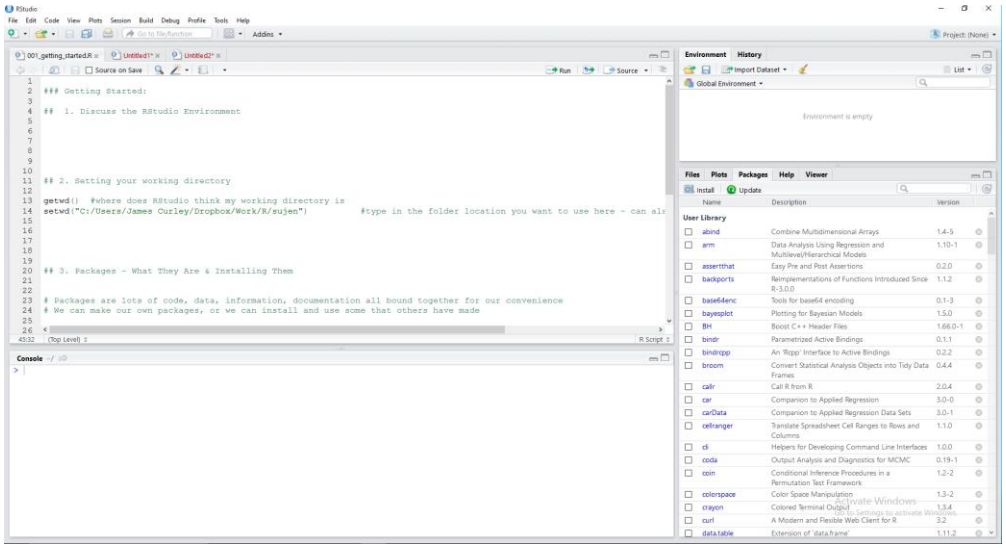
1. Introduction to R and R Studio

Getting Started - RStudio

The Rstudio Environment



- Environment – where data, functions, objects are located
- History – where past code is located
- Import Dataset tab – import data from file
- Files tab – shows current folder location
- Plots tab – shows current graphics plot
- Packages tab – shows currently installed packages
- Help – documentation and help guide
- Viewer – alternative graphics interface e.g. for HTML graphics
- Console –code input, past code and returned output from functions
- Scripts Window – current script files

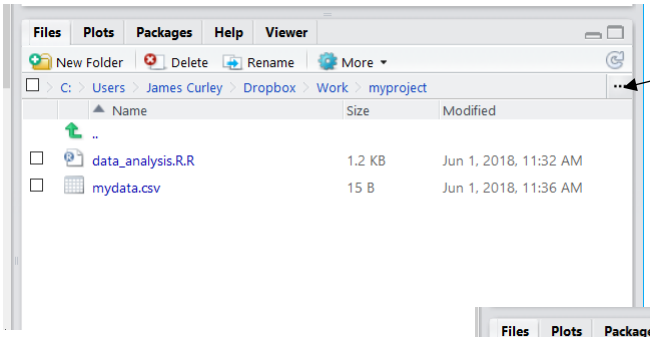
- Other RStudio menu features
- RStudio Addins tab

Working Directories - what are they and how do they help you ?

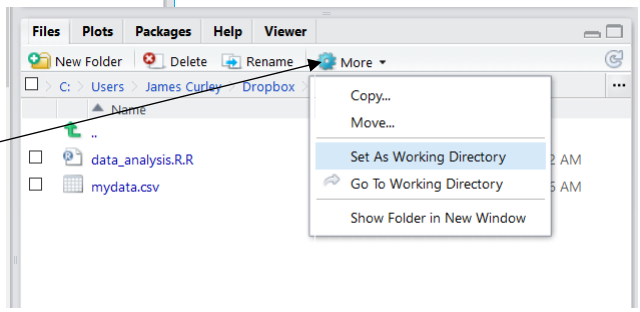
```
## 2. Setting your working directory

getwd()  #where does RStudio think my working directory is
setwd("filename") #type in the folder location you want to use here - can also set using Files panel in RStudio
```

This tells RStudio which folder you consider to be the 'home' location – it will be where it will go searching for files

You can set using the function "setwd()" or you can set using the RStudio interface



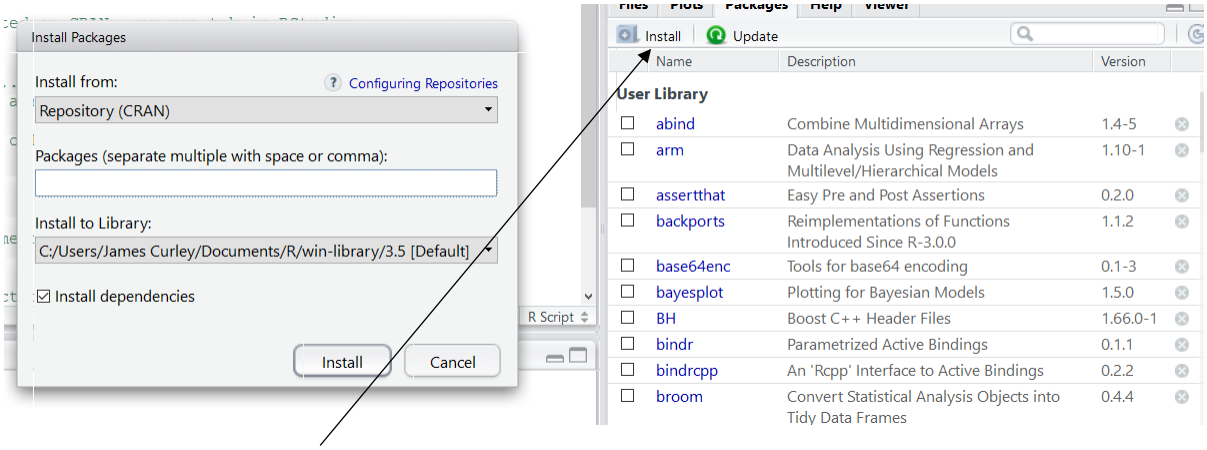
Use the three dots icon "…" to browse for your folder you wish to be the working directory



Use the wheel icon to "Set As Working Directory"

Packages - what are they and how do they help you ?

```
## 3. Packages - What They Are & Installing Them


# Packages are lots of code, data, information, documentation all bound together for our convenience
# We can make our own packages, or we can install and use some that others have made


install.packages("dplyr")      # for example - if package hosted on CRAN - or use tab in RStudio


# Can install the latest development version from github with...
# devtools::install_github("hadley/dplyr")  #make sure you've already installed 'devtools' package

# Packages can be hosted elsewhere occasionally too (i.e. not only on GitHub or CRAN).
```



Use the install icon to open up the Install Packages menu window.

Some useful functions – these are here for reference (you may wish to use them at some point !)

You don't need to remember them - just come back and copy/paste them when needed.

```
### 4. Some useful things you can do to clear console/environment:
# you don't need this now - but you might in the future.

rm(list=ls())  #removes all objects in environment (data, functions, etc)

dev.off() #clears graphic device.

cat("\014") #clears the console

x <- 5
rm(x)  #just removes x from environment.
```

1a. Importing and Exporting Data into R Studio

Importing a CSV file: read.csv()

```
####  Importing and Exporting Data ----


## Getting Data into R - 1.----

# There are ways of doing so from text file, csv file, excel file, other stats packages files,
# webpages (HTML), XML, RData objects, fitbit, google drive, open docs, almost everything....
# special consideration maybe needed if you have really, really large files.
# see refs


# Example - csv file in working directory called pga.csv"
pga <- read.csv("C:/Users/James Curley/Dropbox/dataviz course/data/pga.csv", stringsAsFactors=FALSE)
head(pga)
tail(pga)

pga <- read.csv("pga.csv", stringsAsFactors=FALSE)
# this will overwrite previous pga - be careful naming objects in R....
# ok to use ".", "_", don't start with numbers
```

Importing a CSV file: readr::read_csv()

```
library(tidyverse)
readr::read_csv("pga.csv") # note the underscore, note no "stringsAsFactors=FALSE"

pga <- readr::read_csv("pga.csv") # to save/assign dataset with name "pga"
```

“readr” is a package that is installed with the “tidyverse” suite of packages.

To call a specific function from a specific package we use the double colon between the package and function names.

If you load the library, you do not need to do this. I’m doing it throughout so that you can see which function comes from which library

```r
# Some basic dataframe functions
head(pga) # see top 6 rows
tail(pga) # see bottom 6 rows
dim(pga)  # get rows and columns sizes
nrow(pga) # how many rows in dataset
ncol(pga) # how many columns in dataset

names(pga) # names of variables
colnames(pga) # column names
rownames(pga) # row names

str(pga) # get the structure of the data


View(pga) # See whole dataset in new tab
          # As you get more used to RStudio,
          # you will likely not use this as much (useful for beginners though)
          #or just double click data in Global Environment

pga   # look at data in console (if used read_csv it will be a truncated neat output)
```

```
> head(pga) # see top 6 rows
# A tibble: 6 x 53
  drivepct fairways.hit possible.fairways  year name       driveavg total.distance total.drives rounds girpct greens.hit
     <dbl>        <int>             <int> <int> <chr>         <dbl>          <int>        <int>  <int>  <dbl>      <int>
1     77.2         1031              1335  2004 Fred Funk      272.          52198          192     97   65.5       1132
2     77.1          978              1268  2004 Scott Ver~     278           50314          181     92   68.5       1122
3     76.2          881              1157  2004 Craig Bow~     271.          44959          166     83   62.9        940
4     75.1          886              1180  2004 Joe Durant     287.          48829          170     85   73.3       1121
5     74.7          861              1153  2004 Tom Byrum      273.          45246          166     83   63.6        951
6     74.3          664               894  2004 Jose Coce~     279.          35479          127     68   63.3        729
# ... with 42 more variables: total.holes <int>, goforgreenpct <dbl>, goforgreen.attempts <int>,
#   goforgreen.noattempts <int>, sandsavepct <dbl>, total.sandsaves <int>, total.bunkers <int>, scramblingpct <dbl>,
#   parorbetter <int>, missedgir <int>, threeputtpct <dbl>, threeputt.total <int>, oneputtpct <dbl>,
#   oneputt.total <int>, putts.avg <dbl>, total.putts <int>, rounds1 <int>, birdiepct <dbl>, total.birdies <int>,
#   score.avg <dbl>, strokes <int>, birdiebogieratio <dbl>, total.birdies.better <int>, total.bogeys.worse <int>,
#   par3.birdiepct <dbl>, par3.birdies.better <int>, par3.holes <int>, par4.birdiepct <dbl>, par4.birdies.better <int>,
#   par4.holes <int>, par5.birdiepct <dbl>, par5.birdies.better <int>, par5.holes <int>, total.events <int>,
#   top10s <int>, firsts <int>, seconds <int>, thirds <int>, moneypct <dbl>, totalmoney <int>, country <chr>,
#   label <chr>
```

```
> dim(pga)  # get rows and columns sizes
[1] 2269   53
> nrow(pga) # how many rows in dataset
[1] 2269
> ncol(pga) # how many columns in dataset
[1] 53
> colnames(pga) # column names
 [1] "drivepct"               "fairways.hit"           "possible.fairways"      "year"
 [5] "name"                   "driveavg"               "total.distance"         "total.drives"
 [9] "rounds"                 "girpct"                 "greens.hit"             "total.holes"
[13] "goforgreenpct"          "goforgreen.attempts"    "goforgreen.noattempts" "sandsavepct"
[17] "total.sandsaves"        "total.bunkers"          "scramblingpct"          "parorbetter"
[21] "missedgir"              "threeputtpct"           "threeputt.total"        "oneputtpct"
[25] "oneputt.total"          "putts.avg"              "total.putts"            "rounds1"
[29] "birdiepct"              "total.birdies"          "score.avg"              "strokes"
[33] "birdiebogieratio"       "total.birdies.better"   "total.bogeys.worse"     "par3.birdiepct"
[37] "par3.birdies.better"    "par3.holes"             "par4.birdiepct"         "par4.birdies.better"
[41] "par4.holes"             "par5.birdiepct"         "par5.birdies.better"    "par5.holes"
[45] "total.events"           "top10s"                 "firsts"                 "seconds"
[49] "thirds"                 "moneypct"               "totalmoney"             "country"
[53] "label"
```

```
> str(pga) # get the structure of the data
Classes 'tbl_df', 'tbl' and 'data.frame':       2269 obs. of  53 variables:
 $ drivepct             : num  77.2 77.1 76.2 75.1 74.7 ...
 $ fairways.hit         : int  1031 978 881 886 861 664 711 900 541 868 ...
 $ possible.fairways    : int  1335 1268 1157 1180 1153 894 958 1224 737 1187 ...
 $ year                 : int  2004 2004 2004 2004 2004 2004 2004 2004 2004 2004 ...
 $ name                 : chr  "Fred Funk" "Scott Verplank" "Craig Bowden" "Joe Durant" ...
 $ driveavg             : num  272 278 271 287 273 ...
 $ total.distance       : int  52198 50314 44959 48829 45246 35479 38935 49614 28645 46280 ...
 $ total.drives         : int  192 181 166 170 166 127 138 176 106 170 ...
 $ rounds               : int  97 92 83 85 83 68 73 92 57 85 ...
 $ girpct               : num  65.5 68.5 62.9 73.3 63.6 ...
 $ greens.hit           : int  1132 1122 940 1121 951 729 856 1092 634 958 ...
 $ total.holes          : int  1728 1638 1494 1530 1494 1152 1242 1584 954 1530 ...
 $ goforgreenpct        : num  35 47 28.4 50.4 35.5 ...
 $ goforgreen.attempts  : int  102 116 80 141 97 128 107 113 57 100 ...
 $ goforgreen.noattempts: int  189 131 202 139 176 101 126 191 134 192 ...
 $ sandsavepct          : num  54.6 47.1 53.1 41.7 52.5 ...
 $ total.sandsaves      : int  89 64 78 45 74 79 42 54 39 79 ...
 $ total.bunkers        : int  163 136 147 108 141 128 104 113 78 139 ...
 $ scramblingpct        : num  61.2 61.8 59.2 58.9 63.2 ...
 $ parorbetter          : int  365 319 328 241 343 275 227 292 189 365 ...
 $ missedgir            : int  596 516 554 409 543 423 386 492 320 572 ...
 $ threeputtpct         : num  2.84 2.26 3.08 3.66 2.41 2.78 2.9 2.71 2.2 3.27 ...
 $ threeputt.total      : int  49 37 46 56 36 32 36 43 21 50 ...
 $ oneputtpct           : num  39.2 39.8 39.4 34.7 40.4 ...
 $ oneputt.total        : int  677 652 588 531 604 502 465 599 357 596 ...
 $ putts.avg            : num  28.9 28.8 28.9 30.1 28.7 ...
 $ total.putts          : int  2774 2619 2401 2563 2380 1819 2041 2580 1550 2470 ...
 $ rounds1              : int  96 91 83 85 83 64 69 88 53 85 ...
 $ birdiepct            : num  28.4 30.9 27.1 27 27.9 ...
 $ total.birdies        : int  321 346 254 302 265 225 247 305 161 252 ...
 $ score.avg            : num  70.4 69.9 71.5 70.7 70.5 ...
 $ strokes              : int  6776 6372 5938 5984 5868 4534 4852 6229 3781 6064 ...
 $ birdiebogieratio     : num  1.28 1.6 1.04 1.46 1.22 1.32 1.32 1.34 1.15 1.09 ...
 $ total.birdies.better : int  344 363 273 313 282 233 254 317 171 272 ...
 $ total.bogeys.worse   : int  268 227 263 215 232 176 193 237 149 250 ...
```

Basically data in any form can be imported into Rstudio – see REFS for more information

```
## Getting Data into R - 2. - From Text Files ----
read.delim("pets.txt", header=T)



## Getting Data into R - 3. - From Excel ----

# both of these can be performed using the RStudio Import Dataset Tab
readxl::read_xlsx("mydata.xlsx")
readxl::read_xls("mydata.xls")


## Getting Data into R - 4. - From Other Statistics Packages ----

library(haven)


# SAS
read_sas("mtcars.sas7bdat")

# SPSS
read_sav("mtcars.sav")

# Stata
read_dta("mtcars.dta")
```

Data from websites:

```
## Getting Data into R - 5. - From websites ----

read.csv("https://gist.githubusercontent.com/jalapic/424f6b04a72539e6866044625a9f69e0/raw/ca60926f900324f6137467a09c7bdb7823104afa/lifeExp.csv")
readr::read_csv("https://gist.githubusercontent.com/jalapic/424f6b04a72539e6866044625a9f69e0/raw/ca60926f900324f6137467a09c7bdb7823104afa/lifeExp.csv")
```

CSV, txt, JSON, HTML, etc. – all manner of data hosted on websites can directly be read into Rstudio

Directly entering data into a dataframe structure using function **data.frame()**

```
## Getting Data into R - 4. - Typing it in ----

df <- data.frame(
  country = c("Brazil", "Germany", "Italy", "Argentina", "Uruguay",
              "France", "England", "Spain", "Netherlands", "Czech_",
              "Hungary", "Sweden"),
  wins = c(5,4,4,2,2,1,1,1,0,0,0,0),
  runnerups = c(2,4,2,3,0,1,0,0,3,2,2,1),
  confed = c("COMMEBOL", "UEFA", "UEFA", "COMMEBOL", "COMMEBOL", "UEFA",
             "UEFA", "UEFA", "UEFA", "UEFA", "UEFA", "UEFA")
)

df

df$finals <- df$wins + df$runnerups


colnames(df)
colnames(df)[3]<-"finalists"
colnames(df)
colnames(df)<-c("Country", "Wins", "Finalists", "Confederation", "Finals")
df
```

c() is a function that creates a vector of things – could be numbers, text, both

```
> df
       Country Wins Finalists Confederation Finals
1       Brazil    5         2      COMMEBOL      7
2      Germany    4         4          UEFA      8
3        Italy    4         2          UEFA      6
4    Argentina    2         3      COMMEBOL      5
5      Uruguay    2         0      COMMEBOL      2
6       France    1         1          UEFA      2
7      England    1         0          UEFA      1
8        Spain    1         0          UEFA      1
9  Netherlands    0         3          UEFA      3
10      Czech_    0         2          UEFA      2
11     Hungary    0         2          UEFA      2
12      Sweden    0         1          UEFA      1
```

Exporting Data means writing a file to your working directory

```
#### Exporting Data.


# the most common way will be to write a csv file of your data

write.csv(df, "mydata.csv", row.names = F) #row names prevents an extra column of indices being produced
getwd() # you can find where your working directory is - this is where the file will be saved

## for bigger files, or more complex data structures:

# you may also wish to store data as an RData object
saveRDS(df, file = "mydata.rds")

# these can be read like this:
readRDS(file = "mydata.rds")

## you can also export as other statistics package files using 'haven'
haven::write_sas(mtcars, "mtcars.sas7bdat")
haven::write_sav(mtcars, "mtcars.sav")
haven::write_dta(mtcars, "mtcars.dta")
```

1b. Working with Data in RStudio

Some basic R Syntax - what are vectors ?

```
### Basic Syntax Introduction


## Objects  ----

# Objects are things like data (in any of its forms), functions, etc.

## Here we will learn about 'vectors' and 'dataframes' - the main two data types.



# <-    This thing is the 'assignment operator' - it tells us to store something on the RHS as the name on the LHS
# =     this also works (but don't use it just yet) - it is better to use    <-



# Vectors are one type of object -
mycolors <- c("blue", "red", "yellow")
mycolors #[1] "blue"   "red"    "yellow"

mynums <- 1:100  # the colon means 'to'
mynums #prints 1 to 100 in the console


x  <- c("just", "some", "things", 1, 2, 3, "james") #can combine numbers and characters - all will be characters
x #[1] "just"   "some"   "things" "1"      "2"      "3"      "james"
```

Some basic R Syntax - what are functions ?

```
## Function examples-

# we already used one function above 'c' means to 'concatenate'

print(mycolors)  #print to console  - [1] "blue"   "red"    "yellow"

sample(mycolors)  # R has built in functions - e.g. sample - pick colors without replacement, ... run it a few times


sample(mycolors, size=1)  # functions often have more than one 'argument' , this tells it to pick one

sample(mycolors, 1) #you often don't need to add the argument name - it works it out

sample(mycolors, 2, replace=T)  #another argument in sample is 'replace', here we sample WITH replacement - run a few times


str(mycolors)  # str is a very useful function - it will tell us what type of object we have. "chr" means 'character'
str(mynums)  #int means integers

length(mycolors)  #how many are in the vector?
length(mynums)

rep(mycolors,2) # rep is useful - it repeats the vector like this
rep(mycolors,each=2) # or like this
```

Recap some basic R Syntax - what is a dataframe?

```
### We could manually type things into a datafame like this:

df <- data.frame(
                 numbers = 1:5,
                 pet = c(T,F,T,F,F),
                 animal = c("cat", "pig", "dog", "cow", "armadillo")
                 )


df
```

```
> df
  numbers   pet    animal
1       1  TRUE       cat
2       2 FALSE       pig
3       3  TRUE       dog
4       4 FALSE       cow
5       5 FALSE armadillo
>
```

Adding and removing columns/variables from a dataframe

```
#### What if we want to add in a new variable/column into a pre-existing dataframe.......


df$sizes<- c("small", "big", "small", "big", "unsure")

df

df$age <- 10  #if only give one value, will make whole column this value

df

#can overwrite
df$sizes <-c("small", "big", "medium", "big", "medium")
df

#get rid of a column
df$age <- NULL
```

Working with dataframes in base-R

```
> df<-read.csv("wheels.csv") # read in wheels.csv dataset
>
```

Example dataset is the ‘wheels.csv’ file

```
> head(df)
        id   day1   day2    day3   day4      dob   mother  sex strain wheel startdate
1 637au    4175.0 1975.0  1670.0 1636.0 13.11.04 574ac    male   S129     8  20.09.05
2 637az    4243.5  906.0  1233.0 1270.5 13.11.04 572af    male   S129     4  05.09.05
3 656aa    2644.0 4011.5  5237.0 7404.5 16.12.04 593ar    male     B6     1  12.09.05
4 656af    6906.5 5322.5 10424.5 8468.5 16.12.04 593ar    male     B6     2  13.10.05
5 656al    6517.0 4440.0  5333.5 9291.0 19.12.04 554aa    male     B6     8  09.10.05
6 675ag    4003.5 3053.5  3815.5 3761.0 09.02.05 593ad    male     B6     5  13.10.05
> tail(df)
         id    day1    day2    day3    day4      dob   mother  sex strain wheel startdate
75 703ap        6.5     0.5     2.0      NA 08.06.05 637ai    male   S129     6  14.10.05
76 705ab    13794.5 10868.0 14374.5 12729.0 25.07.05 675aj    male     B6     7  25.10.05
77 705ad     1241.5  1042.0  2240.0  4604.5 25.07.05 675aj    male     B6     6  25.10.05
78 705af    11655.0  7949.5 12400.5 15737.0 25.07.05 675aj    male     B6     5  25.10.05
79 705ah    11594.5 13155.0 13450.0 14176.5 28.07.05 675aa    male     B6     3  25.10.05
80 705ak    12321.0  9602.0 13003.0 13421.5 28.07.05 675aa    male     B6     4  25.10.05
>
> str(df)
'data.frame':   80 obs. of  11 variables:
 $ id       : Factor w/ 80 levels "637au   ","637az   ",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ day1     : num  4175 4244 2644 6906 6517 ...
 $ day2     : num  1975 906 4012 5322 4440 ...
 $ day3     : num  1670 1233 5237 10424 5334 ...
 $ day4     : num  1636 1270 7404 8468 9291 ...
 $ dob      : Factor w/ 29 levels "01.04.05","01.06.05",..: 12 12 14 14 16 11 11 18 20 20 ...
 $ mother   : Factor w/ 51 levels "554aa   ","572af   ",..: 3 2 5 5 1 4 4 13 12 12 ...
 $ sex      : Factor w/ 1 level "male": 1 1 1 1 1 1 1 1 1 1 ...
 $ strain   : Factor w/ 5 levels "B6","F1-129B6",..: 4 4 1 1 1 1 1 3 3 3 ...
 $ wheel    : int  8 4 1 2 8 5 4 2 8 1 ...
 $ startdate: Factor w/ 10 levels "05.09.05","09.10.05",..: 7 1 3 4 2 4 4 6 10 1 ...
>
> ncol(df)
[1] 11
> nrow(df)
[1] 80
>
> dim(df)
[1] 80 11
>
> length(df) #for dataframes same as ncol - i.e. how much across
[1] 11
>
```

```r
#To look at data in individual columns/variables
df$id
df[,1]  #column/variable1 after column
```

```
> df$id
 [1] 637au    637az    656aa    656af    656al    675ag    675ai    686ac    686af    686ag    686ah    686ai    686ar    686as    686at    686az    686ba
[18] 686be    686bf    686bg    686bh    686bj    687aa    687ab    687ag    687aj    687ak    687am    687ao    687ap    687aq    687at    687az    687ba
[35] 687bc    692an    692ao    692at    692au    698aa    698ae    698ag    698ai    698ak    698am    698ao    698ap    698aq    698av    698ay    698bk
[52] 698bm    698ca    699ad    699af    699am    699ap    699at    699bh    700aa    700ab    700af    700ah    700ai    700al    700ap    700ar    700as
[69] 700au    700ba    700be    700bg    703ak    703al    703ap    705ab    705ad    705af    705ah    705ak
80 Levels: 637au    637az    656aa    656af    656al    675ag    675ai    686ac    686af    686ag    686ah    686ai    686ar    686as    ... 705ak
> df[,1]  #column/variable1 after column
 [1] 637au    637az    656aa    656af    656al    675ag    675ai    686ac    686af    686ag    686ah    686ai    686ar    686as    686at    686az    686ba
[18] 686be    686bf    686bg    686bh    686bj    687aa    687ab    687ag    687aj    687ak    687am    687ao    687ap    687aq    687at    687az    687ba
[35] 687bc    692an    692ao    692at    692au    698aa    698ae    698ag    698ai    698ak    698am    698ao    698ap    698aq    698av    698ay    698bk
[52] 698bm    698ca    699ad    699af    699am    699ap    699at    699bh    700aa    700ab    700af    700ah    700ai    700al    700ap    700ar    700as
[69] 700au    700ba    700be    700bg    703ak    703al    703ap    705ab    705ad    705af    705ah    705ak
80 Levels: 637au    637az    656aa    656af    656al    675ag    675ai    686ac    686af    686ag    686ah    686ai    686ar    686as    ... 705ak
>
```

```r
df$day4
df[,5]  #column/variable5

#NA means missing data.
```

```
> df$day4
 [1]  1636.0  1270.5  7404.5  8468.5  9291.0  3761.0 11684.0  9165.5 10187.5 17563.5  1195.5  8190.5 12543.0 15929.5 19913.5 11139.0     3.0 22298.5 14862.0
[20] 23317.5 14799.5 20567.0  9296.5  5153.5    52.0  9573.0  8691.5  9251.5 17913.5 12259.5 28581.5 30463.5 16593.5 14675.0 21031.0 15724.0 12515.5 15790.0
[39] 11379.0  4083.5  3507.0  9653.5  8694.5  4660.5  5340.5  1946.5  2939.5  4116.5  9301.0 14544.5      NA  4520.5 12999.5  5642.5 16331.0  8151.0  5209.0
[58] 14250.5   227.5  6936.5 14132.5 14052.0 11182.0  7686.5 12274.5  9181.0  5925.0 15178.0 18167.0 15682.5 10152.0 10787.5      NA      NA      NA 12729.0
[77]  4604.5 15737.0 14176.5 13421.5
> df[,5]  #column/variable5
 [1]  1636.0  1270.5  7404.5  8468.5  9291.0  3761.0 11684.0  9165.5 10187.5 17563.5  1195.5  8190.5 12543.0 15929.5 19913.5 11139.0     3.0 22298.5 14862.0
[20] 23317.5 14799.5 20567.0  9296.5  5153.5    52.0  9573.0  8691.5  9251.5 17913.5 12259.5 28581.5 30463.5 16593.5 14675.0 21031.0 15724.0 12515.5 15790.0
[39] 11379.0  4083.5  3507.0  9653.5  8694.5  4660.5  5340.5  1946.5  2939.5  4116.5  9301.0 14544.5      NA  4520.5 12999.5  5642.5 16331.0  8151.0  5209.0
[58] 14250.5   227.5  6936.5 14132.5 14052.0 11182.0  7686.5 12274.5  9181.0  5925.0 15178.0 18167.0 15682.5 10152.0 10787.5      NA      NA      NA 12729.0
[77]  4604.5 15737.0 14176.5 13421.5
>
```

```r
#to look at individual observations/rows - before comma
df[15,]  #row15
df[70,]  #row70

#to look at a specific row/col i.e. obs/variable (what you'd consider to be a cell in excel)
df[57,4]  #e.g. observation/row 57,  column 4 which is the day3 score
```

```
> df[15,]  #row15
         id    day1    day2  day3    day4      dob   mother  sex   strain wheel startdate
15 686at    13115.5 11791.5 18054 19913.5 24.03.05 664ae    male F1-B6129     7  30.08.05
> df[70,]  #row70
         id  day1 day2  day3    day4      dob   mother  sex strain wheel startdate
70 700ba    11118 8572 13266 15682.5 01.06.05 674ar    male  Swiss     4  09.10.05
> df[57,4]  #e.g. observation/row 57,  column 4 which is the day3 score
[1] 4690
>
```

table() is a useful function for getting quick summaries of frequencies of data

```
> table(df$strain)

      B6 F1-129B6 F1-B6129     S129    Swiss
      14       22       15       16       13
> table(df$wheel)

1 2 3 4 5 6 7 8 9
9 9 9 8 9 9 9 9 9
> table(df$wheel, df$strain)

    B6 F1-129B6 F1-B6129 S129 Swiss
  1  3        2        2    0     2
  2  2        2        2    1     2
  3  1        3        1    2     2
  4  2        1        1    2     2
  5  2        3        1    1     2
  6  1        1        3    2     2
  7  1        1        3    3     1
  8  1        5        1    2     0
  9  1        4        1    3     0
>
```

Changing column/variable names

```
#### To change column names ####

colnames(df)

#saving the original names as a new object
names_x<-colnames(df)
names_x

# you can change all of them together
colnames(df)<-c("iddddd", "dayyyy1", "dayyyy2", "dayyyy3", "dayyyy4", "dobbb", "mothhhher", "sexxx", "strainnn", "WHEEL", "START")
colnames(df)
head(df)

# you can change just one at a time
colnames(df)[6]
colnames(df)[6] <- "dob"
head(df)

# you can change several at once
colnames(df)[2:5] <-c("day1", "day2", "day3", "day4")
colnames(df)
head(df)

# you can change all this way too
names_x
colnames(df)<-names_x
head(df)
```

Adding removing columns

```
### Adding / Removing a column

df$totalruns <- df$day1 + df$day2 + df$day3 + df$day4
head(df)

df$totalruns <- NULL
head(df)
```

Subsetting data

```
> subset(df, strain=="Swiss")
         id    day1    day2    day3    day4      dob   mother  sex strain wheel startdate
60 700aa     2892.0  4733.0  5314.5  6936.5 05.06.05 674bb    male  Swiss     6  20.09.05
61 700ab     7496.5 10763.5 14837.0 14132.5 05.06.05 674ay    male  Swiss     5  20.09.05
62 700af    11332.0 11000.5 11858.0 14052.0 06.06.05 674ah    male  Swiss     4  20.09.05
63 700ah     9208.0  7453.5  9616.5 11182.0 06.06.05 674be    male  Swiss     3  20.09.05
64 700ai     5780.0  5259.5  6736.0  7686.5 07.06.05 670ba    male  Swiss     2  20.09.05
65 700al     7225.5 11567.0  8804.5 12274.5 03.06.05 674bp    male  Swiss     1  20.09.05
66 700ap     4496.5  5192.5  6253.0  9181.0 03.06.05 674af    male  Swiss     6  09.10.05
67 700ar     4021.0  6878.5  5193.5  5925.0 03.06.05 674bt    male  Swiss     2  09.10.05
68 700as     8680.0  9344.0 13397.5 15178.0 03.06.05 674bo    male  Swiss     1  09.10.05
69 700au     5028.5 11265.0 14573.0 18167.0 02.06.05 674br    male  Swiss     7  09.10.05
70 700ba    11118.0  8572.0 13266.0 15682.5 01.06.05 674ar    male  Swiss     4  09.10.05
71 700be     7856.0  4819.5  9071.0 10152.0 31.05.05 674as    male  Swiss     5  09.10.05
72 700bg     7792.5  7242.0  8181.0 10787.5 08.06.05 670bd    male  Swiss     3  09.10.05
```

Descriptive Statistics – quick summaries

summary() gives a quick snapshot:

```
> summary(df) #some summary stats
       id          day1              day2              day3            day4             dob          mother     sex
 637au   : 1   Min.   :    6.5   Min.   :    0.5   Min.   :    1   Min.   :    3   28.05.05: 8   664af   : 5   male:80
 637az   : 1   1st Qu.: 5748.9   1st Qu.: 4727.0   1st Qu.: 4924   1st Qu.: 5854   01.04.05: 5   665ag   : 5
 656aa   : 1   Median : 8707.2   Median : 7347.8   Median : 8500   Median :10488   07.04.05: 5   664ah   : 4
 656af   : 1   Mean   : 8953.8   Mean   : 7503.7   Mean   : 8756   Mean   :10951   24.03.05: 5   637ai   : 3
 656al   : 1   3rd Qu.:11897.0   3rd Qu.:10789.6   3rd Qu.:12506   3rd Qu.:14815   03.06.05: 4   664ae   : 3
 675ag   : 1   Max.   :21815.5   Max.   :15501.0   Max.   :28515   Max.   :30464   06.06.05: 4   665ab   : 3
 (Other) :74                                                       NA's   :4       (Other) :49   (Other) :57
     strain       wheel          startdate
 B6      :14   Min.   :1.000   05.09.05: 9
 F1-129B6:22   1st Qu.:3.000   09.10.05: 9
 F1-B6129:15   Median :5.000   12.09.05: 9
 S129    :16   Mean   :5.013   15.08.05: 9
 Swiss   :13   3rd Qu.:7.000   20.09.05: 9
               Max.   :9.000   25.10.05: 9
                               (Other) :26
```

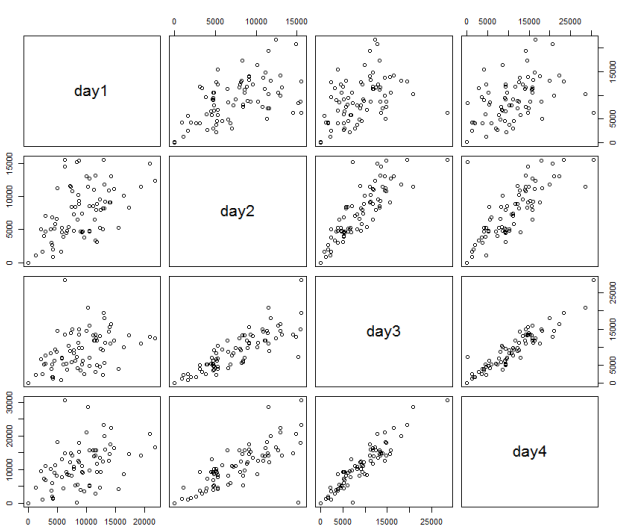
describe() from ‘psych’ gives more info quickly:

```
> psych::describe(df)
           vars  n     mean      sd   median  trimmed     mad min     max   range  skew kurtosis     se
id*           1 80    40.50   23.24    40.50    40.50   29.65 1.0    80.0    79.0  0.00    -1.25   2.60
day1          2 80  8953.76 4668.84  8707.25  8820.04 4528.23 6.5 21815.5 21809.0  0.31    -0.05 521.99
day2          3 80  7503.69 4023.86  7347.75  7438.41 4099.39 0.5 15501.0 15500.5  0.17    -0.77 449.88
day3          4 80  8755.95 5391.03  8499.50  8489.88 5663.53 1.0 28515.0 28514.0  0.68     0.79 602.74
day4          5 76 10950.78 6376.17 10487.50 10641.22 6625.00 3.0 30463.5 30460.5  0.52     0.33 731.40
dob*          6 80    15.06    8.96    15.50    15.20   12.60 1.0    29.0    28.0 -0.07    -1.45   1.00
mother*       7 80    23.18   14.59    18.50    22.41   14.83 1.0    51.0    50.0  0.41    -1.17   1.63
sex*          8 80     1.00    0.00     1.00     1.00    0.00 1.0     1.0     0.0   NaN      NaN   0.00
strain*       9 80     2.90    1.36     3.00     2.88    1.48 1.0     5.0     4.0  0.15    -1.25   0.15
wheel        10 80     5.01    2.61     5.00     5.02    2.97 1.0     9.0     8.0 -0.01    -1.29   0.29
startdate*   11 80     5.58    3.03     6.00     5.59    4.45 1.0    10.0     9.0 -0.07    -1.39   0.34
```

stat.desc() from ‘pastecs’ can be useful too (although not so much here):

```
> pastecs::stat.desc(df)
          id         day1         day2         day3         day4 dob mother sex strain       wheel startdate
nbr.val   NA 8.000000e+01 8.000000e+01 8.000000e+01 7.600000e+01  NA     NA  NA     NA  80.0000000        NA
nbr.null  NA 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00  NA     NA  NA     NA   0.0000000        NA
nbr.na    NA 0.000000e+00 0.000000e+00 0.000000e+00 4.000000e+00  NA     NA  NA     NA   0.0000000        NA
min       NA 6.500000e+00 5.000000e-01 1.000000e+00 3.000000e+00  NA     NA  NA     NA   1.0000000        NA
max       NA 2.181550e+04 1.550100e+04 2.851500e+04 3.046350e+04  NA     NA  NA     NA   9.0000000        NA
range     NA 2.180900e+04 1.550050e+04 2.851400e+04 3.046050e+04  NA     NA  NA     NA   8.0000000        NA
sum       NA 7.163005e+05 6.002950e+05 7.004760e+05 8.322590e+05  NA     NA  NA     NA 401.0000000        NA
median    NA 8.707250e+03 7.347750e+03 8.499500e+03 1.048750e+04  NA     NA  NA     NA   5.0000000        NA
mean      NA 8.953756e+03 7.503688e+03 8.755950e+03 1.095078e+04  NA     NA  NA     NA   5.0125000        NA
SE.mean   NA 5.219926e+02 4.498807e+02 6.027360e+02 7.313967e+02  NA     NA  NA     NA   0.2920322        NA
CI.mean   NA 1.039000e+03 8.954652e+02 1.199716e+03 1.457017e+03  NA     NA  NA     NA   0.5812756        NA
var       NA 2.179810e+07 1.619141e+07 2.906326e+07 4.065552e+07  NA     NA  NA     NA   6.8226266        NA
std.dev   NA 4.668844e+03 4.023856e+03 5.391035e+03 6.376168e+03  NA     NA  NA     NA   2.6120158        NA
coef.var  NA 5.214396e-01 5.362504e-01 6.156996e-01 5.822572e-01  NA     NA  NA     NA   0.5211004        NA
>
```

```
> plot(df)
> plot(df[,2:5])
```



Can use **plot()** to get quick snapshots of the relationships between variables – more on this in the visualization section

Use **by()** to get summary data for each group

```
> by(df, INDICES = df$strain, summary)
df$strain: B6
        id          day1            day2            day3            day4             dob          mother
 656aa   :1   Min.   : 1242   Min.   : 1042   Min.   : 2240   Min.   : 3761   25.07.05:3   675aj   :3
 656af   :1   1st Qu.: 5073   1st Qu.: 4661   1st Qu.: 6257   1st Qu.: 8674   09.02.05:2   593ad   :2
 656al   :1   Median :10926   Median : 7211   Median :11104   Median :12100   16.12.04:2   593ar   :2
 675ag   :1   Mean   : 8695   Mean   : 7243   Mean   : 9857   Mean   :11192   18.04.05:2   633ax   :2
 675ai   :1   3rd Qu.:11730   3rd Qu.: 9417   3rd Qu.:12852   3rd Qu.:13988   20.04.05:2   675aa   :2
 692an   :1   Max.   :13794   Max.   :13155   Max.   :15851   Max.   :15790   28.07.05:2   554aa   :1
 (Other) :8                                                                   (Other) :1   (Other) :2
   sex          strain       wheel          startdate
 male:14   B6       :14   Min.   :1.000   25.10.05:7
           F1-129B6: 0   1st Qu.:2.000   13.10.05:4
           F1-B6129: 0   Median :4.000   09.10.05:2
           S129     : 0   Mean   :4.143   12.09.05:1
           Swiss    : 0   3rd Qu.:5.750   05.09.05:0
                          Max.   :9.000   14.10.05:0
                                          (Other) :0
------------------------------------------------------------------------------
df$strain: F1-129B6
        id          day1            day2            day3            day4             dob
 687aa   : 1   Min.   :   20   Min.   :    3.5   Min.   :    1   Min.   :   52   07.04.05:5
 687ab   : 1   1st Qu.: 7105   1st Qu.: 4791.5   1st Qu.: 5041   1st Qu.: 8286   21.04.05:3
```

describeBy() from 'psych' allows you to do by group:

```
> psych::describeBy(df, group="strain")

 Descriptive statistics by group
group: B6
           vars  n     mean      sd   median  trimmed     mad    min     max range  skew kurtosis      se
id*           1 14    40.36   32.10    37.50    40.17   48.93    3.0    80.0    77  0.10    -1.79    8.58
day1          2 14  8694.82 4226.92 10925.50  8890.96 3555.65 1241.5 13794.5 12553 -0.44    -1.51 1129.69
day2          3 14  7243.21 3589.16  7211.25  7267.33 3826.59 1042.0 13155.0 12113  0.08    -1.16  959.24
day3          4 14  9857.32 4212.96 11104.25  9992.62 3278.40 2240.0 15851.0 13611 -0.42    -1.27 1125.96
day4          5 14 11191.86 3967.15 12099.75 11427.92 4768.78 3761.0 15790.0 12029 -0.51    -1.09 1060.27
dob*          6 14    18.00    5.56    16.50    17.75    5.93   11.0    28.0    17  0.55    -1.04    1.49
mother*       7 14    16.07   15.20     7.00    15.67    4.45    1.0    36.0    35  0.50    -1.82    4.06
sex*          8 14     1.00    0.00     1.00     1.00    0.00    1.0     1.0     0   NaN      NaN    0.00
strain*       9 14     1.00    0.00     1.00     1.00    0.00    1.0     1.0     0   NaN      NaN    0.00
wheel        10 14     4.14    2.66     4.00     4.00    2.97    1.0     9.0     8  0.34    -1.29    0.71
startdate*   11 14     6.14    3.03     6.50     6.25    3.71    2.0     9.0     7 -0.13    -1.96    0.81
------------------------------------------------------------------------------
group: F1-129B6
           vars  n     mean      sd   median  trimmed     mad  min     max   range  skew kurtosis      se
id*           1 22    39.45   13.36    33.50    39.11   13.34 23.0    59.0    36.0  0.28    -1.76    2.85
day1          2 22 10088.11 5203.45  9575.50  9918.72 4205.39 20.0 21815.5 21795.5  0.37    -0.17 1109.38
day2          3 22  8394.89 4335.33  8121.25  8369.50 5007.85  3.5 15501.0 15497.5  0.12    -1.09  924.30
```

```
> swiss <- subset(df, strain=="Swiss")
> swiss[,2:5]
      day1    day2    day3    day4
60  2892.0  4733.0  5314.5  6936.5
61  7496.5 10763.5 14837.0 14132.5
62 11332.0 11000.5 11858.0 14052.0
63  9208.0  7453.5  9616.5 11182.0
64  5780.0  5259.5  6736.0  7686.5
65  7225.5 11567.0  8804.5 12274.5
66  4496.5  5192.5  6253.0  9181.0
67  4021.0  6878.5  5193.5  5925.0
68  8680.0  9344.0 13397.5 15178.0
69  5028.5 11265.0 14573.0 18167.0
70 11118.0  8572.0 13266.0 15682.5
71  7856.0  4819.5  9071.0 10152.0
72  7792.5  7242.0  8181.0 10787.5
>
>
> ## do it by row
> apply(swiss[,2:5], 1, min) #1 indicates rows
     60      61      62      63      64      65      66      67      68      69      70      71      72
 2892.0  7496.5 11000.5  7453.5  5259.5  7225.5  4496.5  4021.0  8680.0  5028.5  8572.0  4819.5  7242.0
> apply(swiss[,2:5], 1, mean)
       60        61        62        63        64        65        66        67        68        69
 4969.000 11807.375 12060.625  9365.000  6365.500  9967.875  6280.750  5504.500 11649.875 12258.375
       70        71        72
12159.625  7974.625  8500.750
>
> ## do it by column
> apply(swiss[,2:5], 2, min) #2 indicates columns
  day1   day2   day3   day4
2892.0 4733.0 5193.5 5925.0
> apply(swiss[,2:5], 2, mean)
     day1      day2      day3      day4
 7148.192  8006.962  9777.038 11641.308
>
```

Can use **apply()** to get summaries across columns or rows

```
> range(df$day1)
[1]     6.5 21815.5
> min(df$day1)
[1] 6.5
> max(df$day1)
[1] 21815.5
>
> mean(df$day1)
[1] 8953.756
> median(df$day1)
[1] 8707.25
> sd(df$day1)
[1] 4668.844
>
> mean(df$day4)
[1] NA
> mean(df$day4, na.rm = T)
[1] 10950.78
> sd(df$day4)
[1] NA
> sd(df$day4, na.rm = T)
[1] 6376.168
>
> quantile(df$day1, .25)
     25%
5748.875
> quantile(df$day1, .75)
  75%
11897
> IQR(df$day1)
[1] 6148.125
>
```

Some of the most common descriptive statistics functions in R

Note that if there are missing data in a variable, then some functions cannot compute this – they require you to add “na.rm=T” to remove missing values from the calculations

Standard Error

This is easy to calculate 'by hand' in R, but for some reason there is no base-r function for it.

We can write our own function – or use the functions available in e.g. psych package.

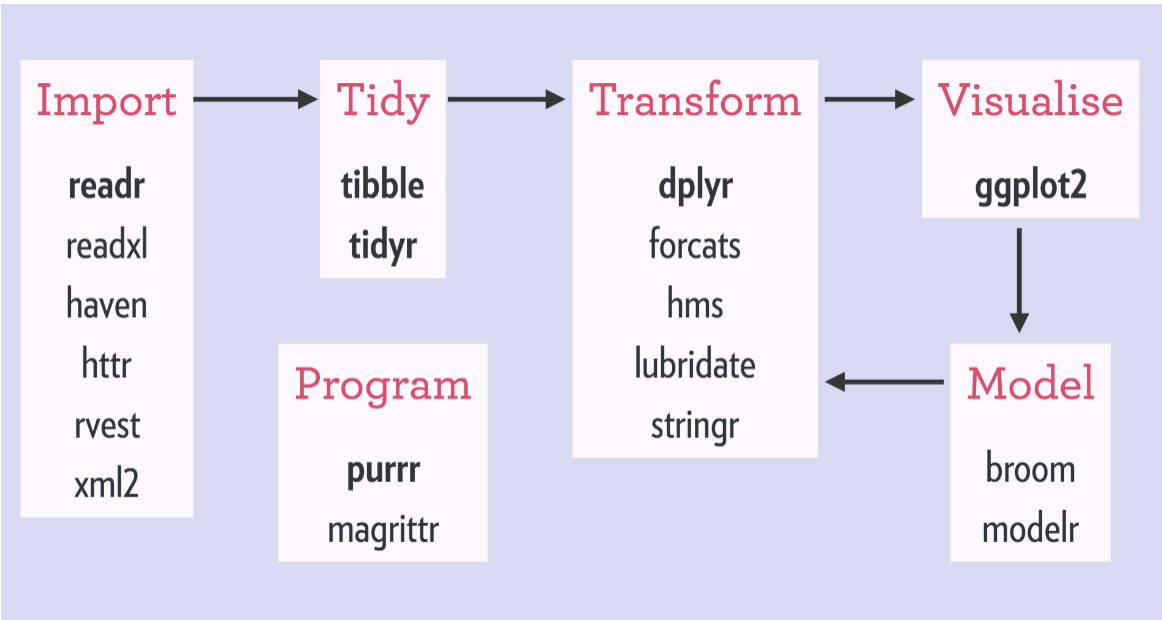
```
> # what about standard error?
> length(df$day1) # we need to know the N to calculate the SEM
[1] 80
> sd(df$day1) / sqrt(length(df$day1)) #SEM by hand
[1] 521.9926
>
> sem <- function(x){sd(x) / sqrt(length(x))} # a function to get SEM
>
> sem(df$day1) #using the function
[1] 521.9926
```

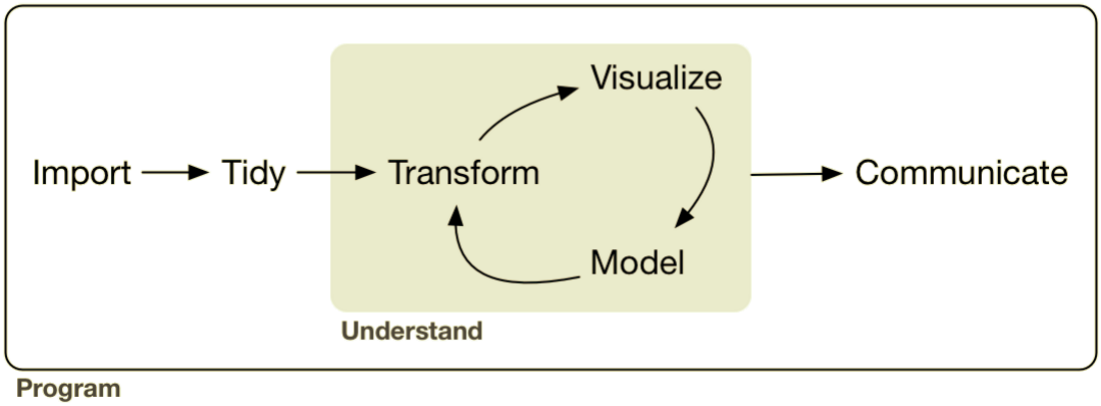
1c. Data Management and Cleaning

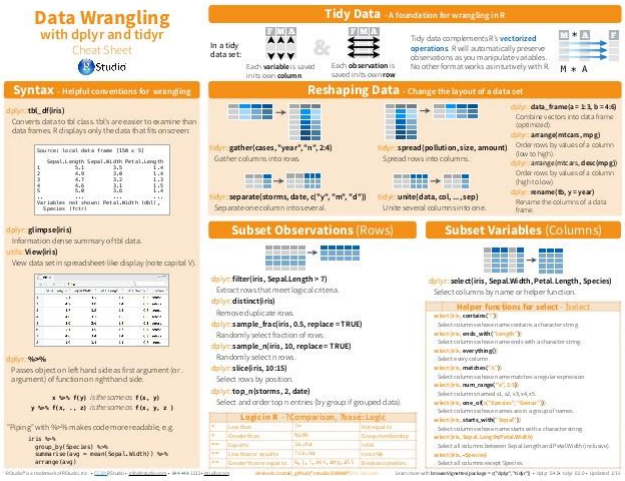
The tidyverse

Components

The tidyverse is a collection of R packages that share common philosophies and are designed to work together. This site is a work-in-progress guide to the tidyverse and its packages.

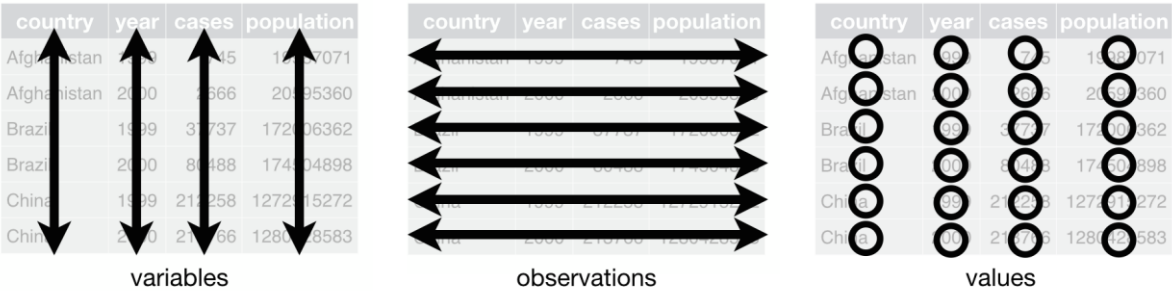






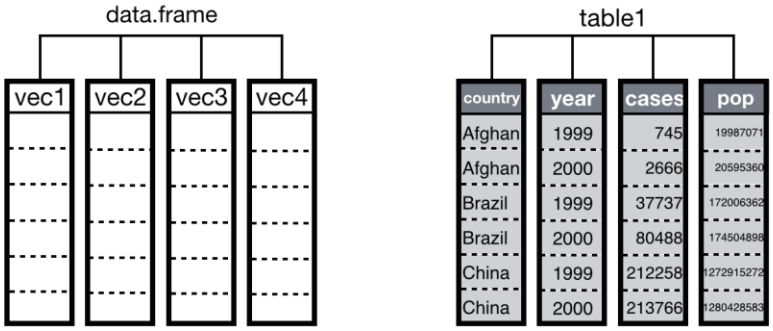
See Rstudio dplyr / tidyr cheatsheet handout

“Tidy” Data



Data Science with R - Garrett Grolemund

“Tidy” Data



Data Science with R - Garrett Grolemund

“Untidy” Data → “Tidy” Data

country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583

table2

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

“Tidy” Data ← “Untidy” Data

country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

table4

An introduction to dplyr

Function	What it does
group_by()	Group data and by a factor variable for subsequent calculations
mutate()	Add a new column / variable
select()	Select specific columns / variables
summarize()	Generate summary statistics
filter()	Keep rows that match specific criteria
arrange()	Order rows by specific criteria
n()	Count number of cases
row_number()	Add a row number (type of ranking method – there are many others)

What is the pipe ?

```
df %>%
   filter(sex == “male”) %>%
   group_by(grade) %>%
   summarize(height_x = mean(height))
```

Read this as “**and then do this**”

Why do it this way? – because it enables us to seam together data, visualization and analysis …… it’s worth the effort !

Example dataset:

```
> weather <- read.csv("lgaweather.csv")
> str(weather)
'data.frame':   23646 obs. of  7 variables:
 $ Date             : Factor w/ 23646 levels "1950-01-01","1950-01-02",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ Max_TemperatureF : int  43 42 57 66 64 57 44 30 45 55 ...
 $ Mean_TemperatureF: int  38 40 50 58 56 50 37 23 34 47 ...
 $ Min_TemperatureF : int  32 39 42 50 46 43 30 16 25 41 ...
 $ Month            : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Year             : int  1950 1950 1950 1950 1950 1950 1950 1950 1950 1950 ...
 $ Day              : int  1 2 3 4 5 6 7 8 9 10 ...
> head(weather)
        Date Max_TemperatureF Mean_TemperatureF Min_TemperatureF Month Year Day
1 1950-01-01               43                38               32     1 1950   1
2 1950-01-02               42                40               39     1 1950   2
3 1950-01-03               57                50               42     1 1950   3
4 1950-01-04               66                58               50     1 1950   4
5 1950-01-05               64                56               46     1 1950   5
6 1950-01-06               57                50               43     1 1950   6
> dim(weather)
[1] 23646     7
```

Use **summarize()** and **group_by()** to get summary data for each group

```
> weather %>%
+   summarise(mean_temp = mean(Max_TemperatureF))
  mean_temp
1  61.60365
> weather %>%
+   group_by(Year) %>%
+   summarise(mean_temp = mean(Max_TemperatureF))
# A tibble: 65 x 2
    Year mean_temp
   <int>     <dbl>
 1  1950      60.6
 2  1951      62.3
 3  1952      62.7
 4  1953      64.2
 5  1954      62.4
 6  1955      62.3
 7  1956      60.7
 8  1957      62.8
 9  1958      59.3
10  1959      62.7
# ... with 55 more rows
```

nb, you can use 'summarise' or 'summarize' interchangeably

```
> # what if you also wanted the standard deviation?
> weather %>%
+   group_by(Year, Month) %>%
+   summarise(mean_temp = mean(Max_TemperatureF), sd_temp = sd(Max_TemperatureF))
# A tibble: 778 x 4
# Groups:   Year [?]
    Year Month mean_temp sd_temp
   <int> <int>     <dbl>   <dbl>
 1  1950     1      47.8   10.7
 2  1950     2      38.0    7.50
 3  1950     3      43.9    9.15
 4  1950     4      55.6    7.76
 5  1950     5      66.0    8.78
 6  1950     6      78.4    8.43
 7  1950     7      82.4    5.52
 8  1950     8      79.9    4.15
 9  1950     9      71.2    7.84
10  1950    10      67.2    8.50
#     with 768 more rows
```

Use **filter()** to subset data

```
> # what if you only cared about July?
> weather %>%
+   filter(Month==7) %>%
+   group_by(Year) %>%
+   summarise(mean_temp = mean(Max_TemperatureF), sd_temp = sd(Max_TemperatureF))
# A tibble: 65 x 3
    Year mean_temp sd_temp
   <int>     <dbl>   <dbl>
 1  1950      82.4    5.52
 2  1951      83.6    5.23
 3  1952      88.2    5.07
 4  1953      85.4    5.92
 5  1954      84.7    6.42
 6  1955      88.9    6.23
 7  1956      81.0    6.56
 8  1957      85.6    6.04
 9  1958      83.1    6.44
10  1959      83.8    4.74
# ... with 55 more rows
>
```

Example dataset:

```
> library(Lahman)
> head(Batting)
   playerID yearID stint teamID lgID  G  AB  R  H X2B X3B HR RBI SB CS BB SO IBB HBP SH SF GIDP
1 abercda01   1871     1    TRO   NA  1   4  0  0   0   0  0   0  0  0  0  0  NA  NA NA NA   NA
2  addybo01   1871     1    RC1   NA 25 118 30 32   6   0  0  13  8  1  4  0  NA  NA NA NA   NA
3 allisar01   1871     1    CL1   NA 29 137 28 40   4   5  0  19  3  1  2  5  NA  NA NA NA   NA
4 allisdo01   1871     1    WS3   NA 27 133 28 44  10   2  2  27  1  1  0  2  NA  NA NA NA   NA
5 ansonca01   1871     1    RC1   NA 25 120 29 39  11   3  0  16  6  2  2  1  NA  NA NA NA   NA
6 armstbo01   1871     1    FW1   NA 12  49  9 11   2   1  0   5  0  1  0  1  NA  NA NA NA   NA
>
```

Use **mutate()** to add a new variable (column)
Use **select()** to select specific columns in the dataset

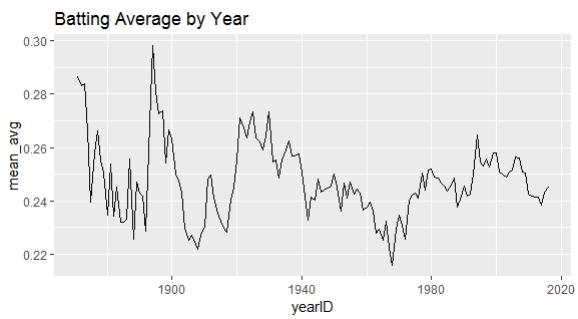
```
> Batting %>%
+   mutate(avg = H / AB) %>%
+   select(playerID, yearID, teamID, lgID, AB, H, avg)
        playerID yearID teamID lgID  AB   H        avg
1      abercda01   1871    TRO   NA   4   0 0.00000000
2       addybo01   1871    RC1   NA 118  32 0.27118644
3      allisar01   1871    CL1   NA 137  40 0.29197080
4      allisdo01   1871    WS3   NA 133  44 0.33082707
5      ansonca01   1871    RC1   NA 120  39 0.32500000
6      armstbo01   1871    FW1   NA  49  11 0.22448980
7      barkeal01   1871    RC1   NA   4   1 0.25000000
8      barnero01   1871    BS1   NA 157  63 0.40127389
9      barrebi01   1871    FW1   NA   5   1 0.20000000
10     barrofr01   1871    BS1   NA  86  13 0.15116279
11      bassjo01   1871    CL1   NA  89  27 0.30337079
12     battijo01   1871    CL1   NA   3   0 0.00000000
13     bealsto01   1871    WS3   NA  36   7 0.19444444
14     beaveed01   1871    TRO   NA  15   6 0.40000000
15     bechtge01   1871    PH1   NA  94  33 0.35106383
16     bellast01   1871    TRO   NA 128  32 0.25000000
```

Many dplyr functions can be chained together in successive rows:

```
> Batting %>%
+   mutate(avg = H / AB) %>%
+   select(playerID, yearID, teamID, lgID, AB, H, avg) %>%
+   filter(AB>50) %>%
+   group_by(yearID) %>%
+   summarise(mean_avg = mean(avg), sd = sd(avg), median = median(avg))
# A tibble: 146 x 4
   yearID mean_avg     sd median
    <int>    <dbl>  <dbl>  <dbl>
 1   1871    0.287 0.0584  0.274
 2   1872    0.283 0.0579  0.281
 3   1873    0.284 0.0558  0.280
 4   1874    0.266 0.0561  0.274
 5   1875    0.239 0.0566  0.244
 6   1876    0.258 0.0531  0.259
 7   1877    0.266 0.0558  0.269
 8   1878    0.255 0.0558  0.248
 9   1879    0.250 0.0506  0.255
10   1880    0.235 0.0516  0.24
# ... with 136 more rows
```

This takes the data, adds an 'avg' column, selects specific columns, then only keeps rows where the individual had at least 50 at bats (AB), and then groups the data by year and calculates the mean, sd, median avg by year

```
> Batting %>%
+   mutate(avg = H / AB) %>%
+   select(playerID, yearID, teamID, lgID, AB, H, avg) %>%
+   filter(AB>50) %>%
+   group_by(yearID) %>%
+   summarise(mean_avg = mean(avg), sd = sd(avg), median = median(avg)) %>%
+   ggplot(aes(x=yearID, y=mean_avg)) + geom_line() +
+   ggtitle("Batting Average by Year")
>
```



The beauty of this dplyr/tidyverse approach is going from raw data to summary data to visualization in a simple chain.

Don't worry about the visualization code right now – we cover this in the next section

Use **n()** to count number of observations per group (or groups):

```
> Batting %>%
+   group_by(yearID, lgID) %>%
+   summarise(total = n())
# A tibble: 276 x 3
# Groups:   yearID [?]
   yearID lgID  total
    <int> <fct> <int>
 1   1871 NA      115
 2   1872 NA      156
 3   1873 NA      125
 4   1874 NA      123
 5   1875 NA      218
 6   1876 NL      124
 7   1877 NL       97
 8   1878 NL       80
 9   1879 NL      127
10   1880 NL      135
# ... with 266 more rows
```

Use **arrange()** to order by a variable:

```
> Batting %>% group_by(yearID) %>% arrange(-H)   # within each year arrange
# A tibble: 102,816 x 22
# Groups:   yearID [146]
   playerID  yearID stint teamID lgID      G    AB     R     H   X2B   X3B    HR   RBI    SB    CS    BB
   <chr>      <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
 1 suzukic01   2004     1 SEA    AL      161   704   101   262    24     5     8    60    36    11    49
 2 sislege01   1920     1 SLA    AL      154   631   137   257    49    18    19   122    42    17    46
 3 odoulle01   1929     1 PHI    NL      154   638   152   254    35     6    32   122     2    NA    76
 4 terrybi01   1930     1 NY1    NL      154   633   139   254    39    15    23   129     8    NA    57
 5 simmoal01   1925     1 PHA    AL      153   654   122   253    43    12    24   129     7    14    35
 6 hornsro01   1922     1 SLN    NL      154   623   141   250    46    14    42   152    17    12    65
 7 kleinch01   1930     1 PHI    NL      156   648   158   250    59     8    40   170     4    NA    54
 8 cobbty01    1911     1 DET    AL      146   591   147   248    47    24     8   127    83    NA    44
 9 sislege01   1922     1 SLA    AL      142   586   134   246    42    18     8   105    51    19    49
10 suzukic01   2001     1 SEA    AL      157   692   127   242    34     8     8    69    56    14    30
# ... with 102,806 more rows, and 6 more variables: SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>,
#   GIDP <int>
```

Use **arrange()** to order by a variable:

```
> Batting %>% group_by(yearID) %>% arrange(-H,.by_group = T)
# A tibble: 102,816 x 22
# Groups:   yearID [146]
   playerID  yearID stint teamID lgID      G    AB     R     H   X2B   X3B    HR   RBI    SB    CS    BB
   <chr>      <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
 1 mcveyca01   1871     1 BS1    NA       29   153    43    66     9     5     0    43     6     0     1
 2 meyerle01   1871     1 PH1    NA       26   130    45    64     9     3     4    40     4     0     2
 3 barnero01   1871     1 BS1    NA       31   157    66    63    10     9     0    34    11     6    13
 4 startjo01   1871     1 NY2    NA       33   161    35    58     5     1     1    34     4     2     3
 5 kingst01    1871     1 TRO    NA       29   144    45    57    10     6     0    34     3     3     1
 6 woltery01   1871     1 NY2    NA       32   138    33    51     6     9     0    44     1     0    10
 7 woodji01    1871     1 CH1    NA       28   135    45    51    10     6     1    29    18     2    11
 8 waterfr01   1871     1 WS3    NA       32   158    46    50     7     4     0    17    11     3    10
 9 pikeli01    1871     1 TRO    NA       28   130    43    49    10     7     4    39     3     2     5
10 flynncl01   1871     1 TRO    NA       29   142    43    48     6     1     0    27     3     3     4
# ... with 102,806 more rows, and 6 more variables: SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>,
#   GIDP <int>
```

Use **summarize_at()** and **summarize_if()** to get summary information over multiple variables

```
> Batting %>% summarise_at(c("AB", "HR", "H"), mean, na.rm = TRUE)
        AB       HR        H
1 141.9055 2.813599 37.13993
> Batting %>% summarise_if(is.numeric, mean, na.rm = TRUE)
    yearID    stint        G       AB        R        H      X2B      X3B       HR      RBI       SB
1 1964.262 1.077838 51.34344 141.9055 18.81554 37.13993 6.289167 1.293252 2.813599 17.00397 2.976821
        CS       BB       SO     IBB      HBP      SH       SF     GIDP
1 1.226008 13.06721 20.52971 1.10587 1.056057 2.29954 1.054101 2.981018
>
```

A very brief introduction to tidyr

So far all the datasets have been in 'tidy' format with observations in rows, variables in columns, and values in columns - what about if data are not in this format ?

- ggplot2 wants data in a long format for plotting
- Several statistical tests need data in a long format to work properly

Function	What it does
gather()	Convert data from wide to long formats
spread()	Convert data from long to wide formats
separate()	Convert one variable into two
unite()	Convert two or more variables into one

Example of an 'untidy' dataset

```
> df <- data.frame(
+   id = c("James", "Patrick", "Steve", "Margaret", "Eileen", "Sheila"),
+   gender = c("Male", "Male", "Male", "Female", "Female", "Female"),
+   before = rnorm(n = 6, mean = 12, sd = 3.5),
+   during = rnorm(n = 6, mean = 7, sd = 3),
+   after = rnorm(n = 6, mean = 10, sd = 4)
+ )
> df
        id gender    before    during     after
1    James   Male 17.327209  4.041748 11.148272
2  Patrick   Male 14.618304 10.577104 13.284113
3    Steve   Male 15.285439 11.465892 11.536081
4 Margaret Female 11.427407  9.470369  9.947773
5   Eileen Female  9.558153  7.549205 11.781887
6   Sheila Female 13.984631  8.398868 11.792134
>
```

Here we are measuring anxiety levels before, during and after a therapy session

All the anxiety level values should just be in one column – this would make it tidy. Other identifying info should be in their own columns too.

Use **gather()** to convert data from a wide format to a long format

```
        id gender   time   anxiety
1     James   Male before 17.327209
2   Patrick   Male before 14.618304
3     Steve   Male before 15.285439
4  Margaret Female before 11.427407
5    Eileen Female before  9.558153
6    Sheila Female before 13.984631
7     James   Male during  4.041748
8   Patrick   Male during 10.577104
9     Steve   Male during 11.465892
10 Margaret Female during  9.470369
11   Eileen Female during  7.549205
12   Sheila Female during  8.398868
13    James   Male  after 11.148272
14  Patrick   Male  after 13.284113
15    Steve   Male  after 11.536081
16 Margaret Female  after  9.947773
17   Eileen Female  after 11.781887
18   Sheila Female  after 11.792134
```

```
## Use gather() to convert from wide to long

df %>%  gather(time, anxiety, c(before, during, after)) # name columns

df %>% gather(time, anxiety, 3:5) # number columns
```

Use **spread()** to convert data from a long format back to a wide format

```
## Use spread() to convert from long to wide

df1 <- df %>% gather(time, anxiety, 3:5) # number columns

df1 %>% spread(time, anxiety)
```

```
> df1 %>% spread(time, anxiety)
        id gender     after    before    during
1   Eileen Female 11.781887  9.558153  7.549205
2    James   Male 11.148272 17.327209  4.041748
3 Margaret Female  9.947773 11.427407  9.470369
4  Patrick   Male 13.284113 14.618304 10.577104
5   Sheila Female 11.792134 13.984631  8.398868
6    Steve   Male 11.536081 15.285439 11.465892
```

Use **unite()** to join variables together

```
## Use unite() to join variables together
# inside unite put: name of new variable, names of variables to join
df %>% gather(time, anxiety, 3:5) %>%
  unite(group, gender, time)
```

```
         id         group   anxiety
1     James   Male_before 17.327209
2   Patrick   Male_before 14.618304
3     Steve   Male_before 15.285439
4  Margaret Female_before 11.427407
5    Eileen Female_before  9.558153
6    Sheila Female_before 13.984631
7     James   Male_during  4.041748
8   Patrick   Male_during 10.577104
9     Steve   Male_during 11.465892
10 Margaret Female_during  9.470369
11   Eileen Female_during  7.549205
12   Sheila Female_during  8.398868
13    James    Male_after 11.148272
14  Patrick    Male_after 13.284113
15    Steve    Male_after 11.536081
16 Margaret  Female_after  9.947773
17   Eileen  Female_after 11.781887
18   Sheila  Female_after 11.792134
```

Use **separate()** to separate variables into different columns

```
## Use separate() to separate joined variables
df %>% gather(time, anxiety, 3:5) %>%
  unite(group, gender, time) %>%
  separate(col = group, into = c("gender", "timeperiod"), sep = "_")
```

```
         id gender timeperiod   anxiety
1     James   Male     before 17.327209
2   Patrick   Male     before 14.618304
3     Steve   Male     before 15.285439
4  Margaret Female     before 11.427407
5    Eileen Female     before  9.558153
6    Sheila Female     before 13.984631
7     James   Male     during  4.041748
8   Patrick   Male     during 10.577104
9     Steve   Male     during 11.465892
10 Margaret Female     during  9.470369
11   Eileen Female     during  7.549205
12   Sheila Female     during  8.398868
13    James   Male      after 11.148272
14  Patrick   Male      after 13.284113
15    Steve   Male      after 11.536081
16 Margaret Female      after  9.947773
17   Eileen Female      after 11.781887
18   Sheila Female      after 11.792134
```

2. Data Visualization & Descriptive Statistics

ggplot2

ggplot2

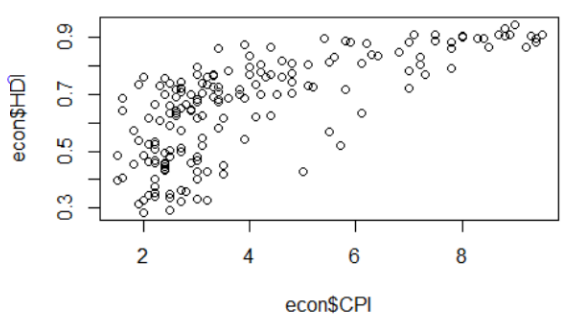
- These slides contain some basic information for creating ggplot2 figures. I am summarizing here but the tutorial code files go through step by step.
- I will demonstrate code for many of the most typical plots that are made in ggplot2.
- Throughout the course we will see several more examples that I do not have space to include in the slides, but all code for these examples will be provided in the additional material,.
- Please see references for many other guides on how to make ggplot2 graphs.
- Remember that ggplot2 figures are highly customizable – every part of the figure can be altered - we will only touch the surface of the vast number of possibilities in graphing
- The focus here will be on charts used commonly when doing statistical analysis.

Example Dataset:

```
> econ <- read.csv("EconomistData.csv")
> head(econ)
      Country HDI.Rank   HDI CPI            Region
1 Afghanistan      172 0.398 1.5      Asia Pacific
2     Albania       70 0.739 3.1 East EU Cemt Asia
3     Algeria       96 0.698 2.9              MENA
4      Angola      148 0.486 2.0               SSA
5   Argentina       45 0.797 3.0          Americas
6     Armenia       86 0.716 2.6 East EU Cemt Asia
```

There are several functions for plotting in base-R and we will use some. An example scatterplot is below using the **plot()** function. These can be made beautiful (with a lot of work) However, ggplot2 gives us far more control.

```
> plot(econ$CPI, econ$HDI)
```

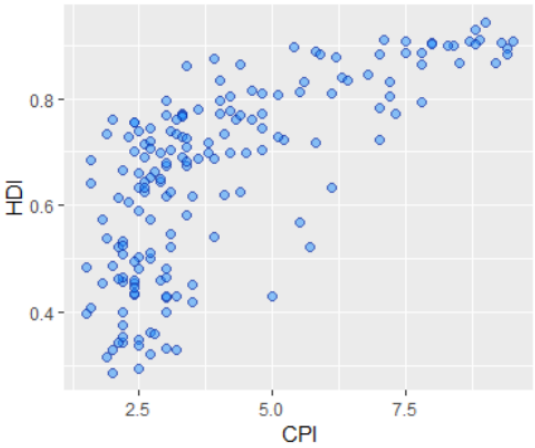


Building up layers of a ggplot2 graph – an illustration

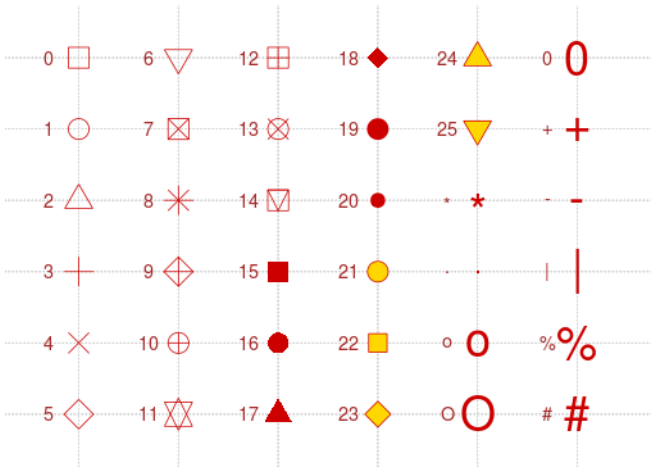
Example 1 - a scatterplot

```
ggplot(econ) #just plots a canvas but knows we are using the 'econ' dataset

ggplot(econ, aes(x=CPI, y=HDI))  # add axes for X and Y

ggplot(econ, aes(x=CPI, y=HDI)) + geom_point() # add dots

ggplot(econ, aes(x=CPI, y=HDI)) + geom_point(shape = 21) #shape the dots

ggplot(econ, aes(x=CPI, y=HDI)) + geom_point(shape = 21, colour = "navy", fill = "dodgerblue") #add colors

ggplot(econ, aes(x=CPI, y=HDI)) + geom_point(size=2) #resize

ggplot(econ, aes(x=CPI, y=HDI)) + geom_point(size=2,
                                             shape = 21, colour = "navy", fill = "dodgerblue",
                                             alpha=.5) #add transparency
```

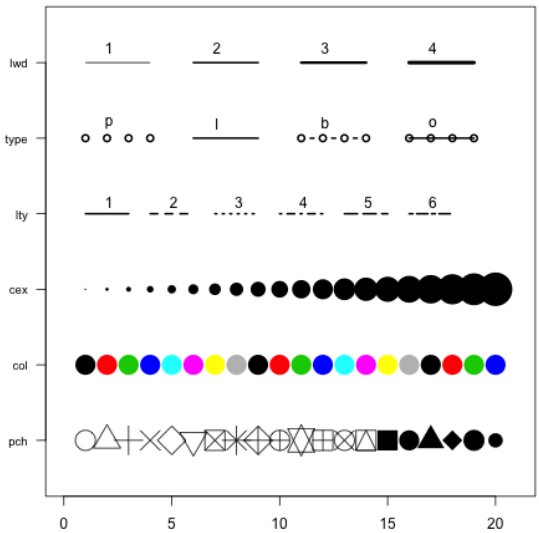
```
> ggplot(econ, aes(x=CPI, y=HDI)) + geom_point(size=2,
+                                              shape = 21, colour = "navy", fill = "dodgerblue",
+                                              alpha=.5) #add transparency
>
```



geom_point() shapes – use “pch=“

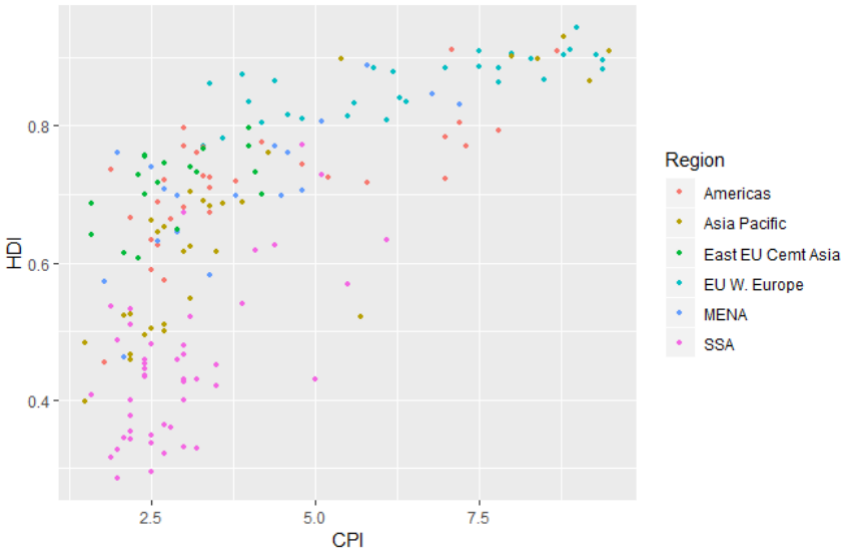


Other common ggplot2 attributes that you may wish to set:

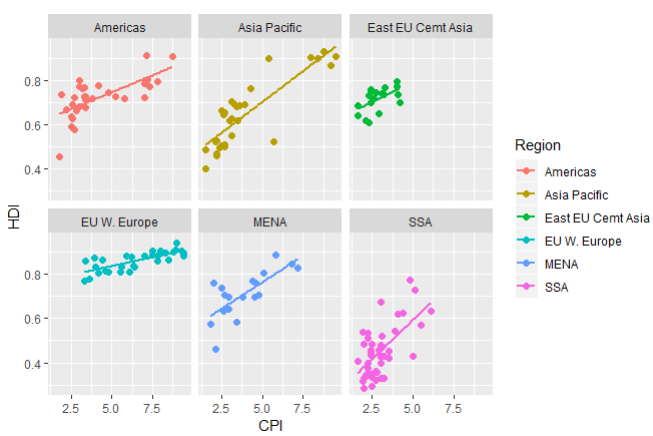


Color by grouping variable

```
> ggplot(econ, aes(x=CPI, y=HDI, color=Region)) + geom_point(size=1) #color by Region
```



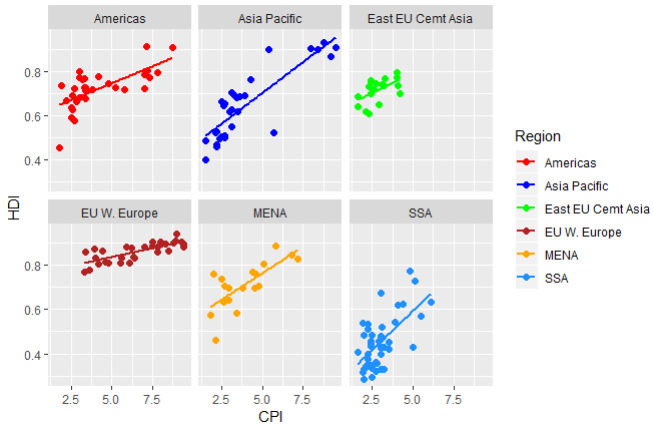

```
ggplot(econ, aes(x=CPI, y=HDI, group=Region, color=Region)) +
  geom_point(size=2) + stat_smooth(se=F, method="lm") + facet_wrap(~Region)
```



To add linear trendline:
stat_smooth(method="lm")

To separate groups into panels:
facet_wrap(~Region)

```
ggplot(econ, aes(x=CPI, y=HDI, group=Region, color=Region)) +
  geom_point(size=2) +
  stat_smooth(se=F, method="lm") +
  facet_wrap(~Region) +
  scale_color_manual(values=c("red", "blue", "green", "firebrick", "orange", "dodgerblue"))
#number of colors must be equal to number of groups
```



There are numerous methods for setting colors. Here we set them manually using

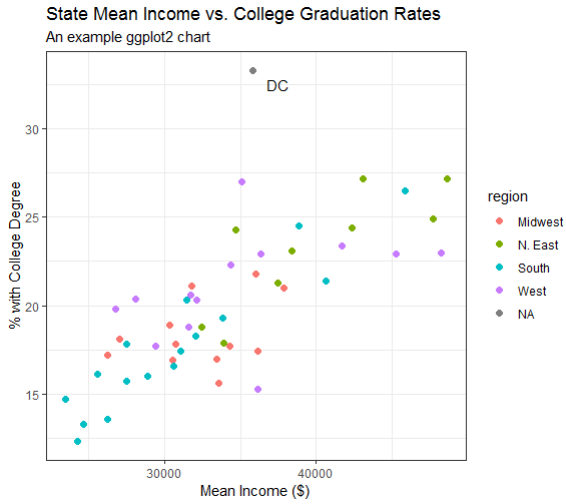
scale_color_manual(values=c(.....))

Other options include generating a gradient of colors or using other predefined color palettes.

```
ggplot(states, aes(income, college, color=region)) +
  geom_point(size=2) +
  xlab("Mean Income ($)") +
  ylab("% with College Degree") +
  ggtitle("State Mean Income vs. College Graduation Rates", subtitle="An example ggplot2 chart") +
  annotate("text", x=37500, y=32.5, label="DC") +
  theme_bw()
```

Examples of additional customization:

- Axes titles
- Main title
- Subtitle
- Annotating text on graph
- Theme of plot



Exporting your ggplot2 figure to your computer

```
myplot <- ggplot(econ, aes(x=CPI, y=HDI, group=Region, color=Region)) +
           geom_point(size=2) +
           stat_smooth(se=F, method="lm") +
           facet_wrap(~Region) +
           scale_color_manual(values=c("red", "blue", "green", "firebrick", "orange", "dodgerblue"))

ggsave("bestplot.png", myplot, width=8, height=8)
#play with width / height to work out what works best for your plot
```

The .png image will be located inside your working directory

Making line graphs in ggplot2

Example Dataset

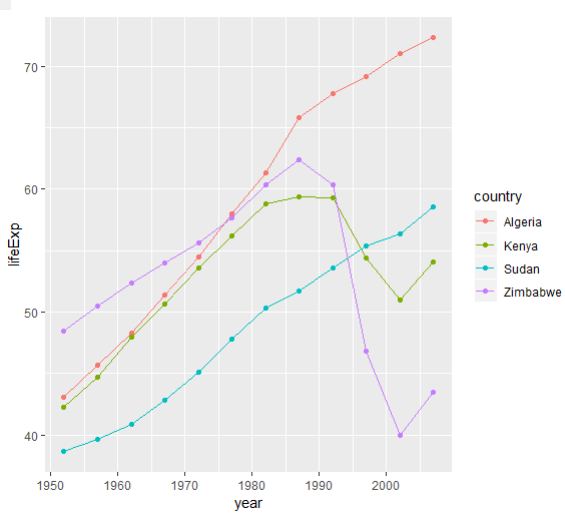
```
> library(gapminder)
> d1 <- c("Algeria", "Sudan", "Kenya", "Zimbabwe")
> d2 <- subset(gapminder, country %in% d1)  #%in% is a function which just means 'in'
```

```
> head(d2)
# A tibble: 6 x 6
  country continent  year lifeExp      pop gdpPercap
  <fct>   <fct>     <int>   <dbl>    <int>     <dbl>
1 Algeria Africa     1952    43.1  9279525     2449.
2 Algeria Africa     1957    45.7 10270856     3014.
3 Algeria Africa     1962    48.3 11000948     2551.
4 Algeria Africa     1967    51.4 12760499     3247.
5 Algeria Africa     1972    54.5 14760787     4183.
6 Algeria Africa     1977    58.0 17152804     4910.
```

Building up a line graph in ggplot2

```
ggplot(d2)

ggplot(d2, aes(x = year, y = lifeExp, color = country)) # tells R to color & group by 'country'

#add point and line by 'country'
ggplot(d2, aes(x = year, y = lifeExp, color = country)) + geom_line() + geom_point()

#alter line width
ggplot(d2, aes(x = year, y = lifeExp, color = country)) + geom_line(lwd=1) + geom_point()

#geom_path() is often equivalent to geom_line() - there are some differences (not here though)
ggplot(d2, aes(x = year, y = lifeExp, color = country)) + geom_path() + geom_point()

p <- ggplot(d2, aes(x = year, y = lifeExp, color = country)) + geom_line() + geom_point()

p

p + theme(legend.position = "none")
```

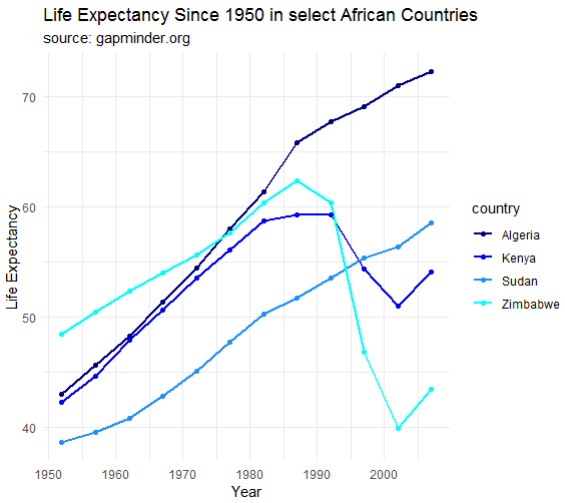
```
ggplot(d2, aes(x = year, y = lifeExp, color = country)) +
  geom_line() +
  geom_point()
```



Basic line graph

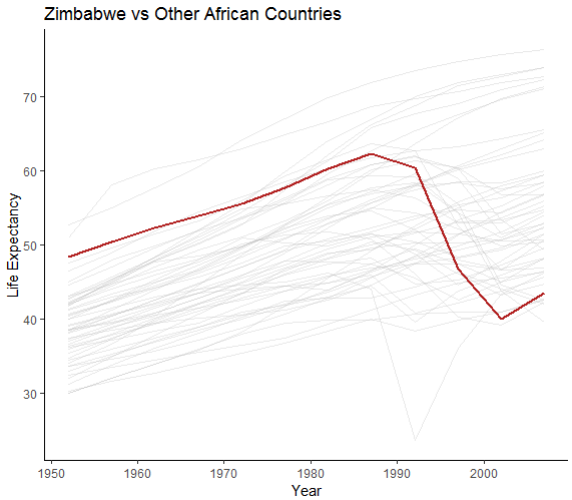
```
ggplot(d2, aes(x = year, y = lifeExp, color = country)) +
  geom_line(lwd=1) +
  geom_point() +
  theme(legend.position = "none") +
  theme_minimal() +
  scale_color_manual(values=c("navy", "blue", "dodgerblue", "cyan")) +
  xlab("Year") +
  ylab("Life Expectancy") +
  ggtitle("Life Expectancy Since 1950 in select African Countries",
          subtitle = "source: gapminder.org")
```

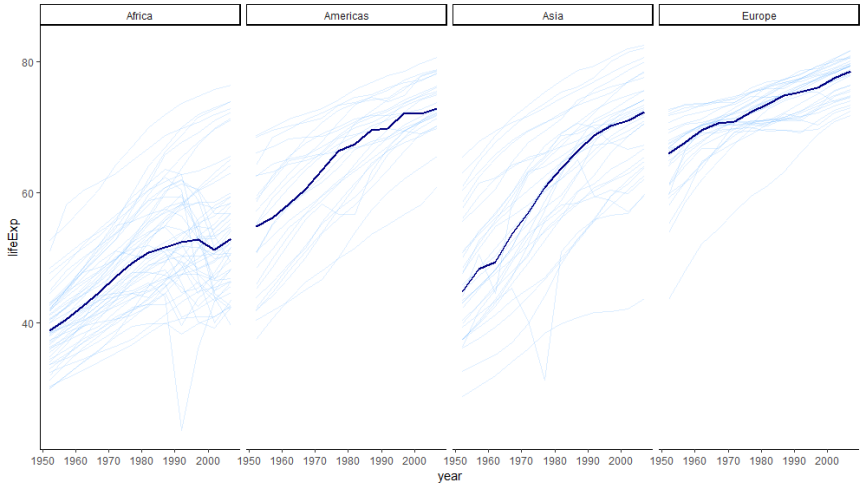
Customized line graph




```
africa <- subset(gapminder, continent=="Africa")
zim <- subset(africa, country=="Zimbabwe")

ggplot(africa, aes(x = year, y = lifeExp,group=country)) +
  geom_line(color="gray66", alpha=.25) +
  geom_line(data=zim, color="firebrick", lwd=1)+
  theme(legend.position="none") +
  theme_classic() +
  ggtitle("Zimbabwe vs Other African Countries") +
  xlab("Year") +
  ylab("Life Expectancy")
```

Overlaying individual observation data with summary data





ggplot2 is very versatile – here we are seeing the median life expectancy of four continents (dark blue lines) against each individual country life expectancy over a range of years (see tutorial for code)

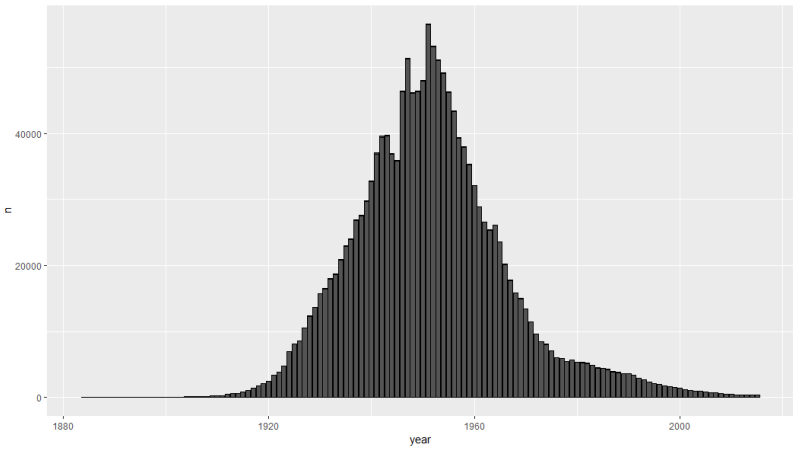
Making histogram plots in ggplot2

```
> patricia <- babynames %>% filter(name=="Patricia")
> patricia
# A tibble: 217 x 5
    year sex   name         n      prop
   <dbl> <chr> <chr>    <int>     <dbl>
 1  1884 F     Patricia     6 0.0000436
 2  1885 F     Patricia     5 0.0000352
 3  1886 F     Patricia     8 0.0000520
 4  1887 F     Patricia    10 0.0000643
 5  1888 F     Patricia    12 0.0000633
 6  1889 F     Patricia    17 0.0000898
 7  1890 F     Patricia    11 0.0000545
 8  1891 F     Patricia    12 0.0000610
 9  1892 F     Patricia    21 0.0000934
10  1893 F     Patricia    28 0.000124
# ... with 207 more rows
```

Example data from the 'babynames' package – contains the total number of children born in each year with a given name and gender 1880-2015

Use **geom_bar()** if data is already in summary format (i.e. every row is a count)

```
ggplot(patricia, aes(x=year, y=n)) +
  geom_bar(stat='identity', color='black', lwd=.5, fill='gray33',width=.9)
```




```
library(Lahman)

head(Batting)

Batting_sum <- Batting %>%
  group_by(playerID) %>%
  summarise(totalH = sum(H),
            totalAB = sum(AB),
            avg = totalH/totalAB
  )

Batting_sum <- Batting_sum %>% filter(totalAB>200)
head(Batting_sum)
```

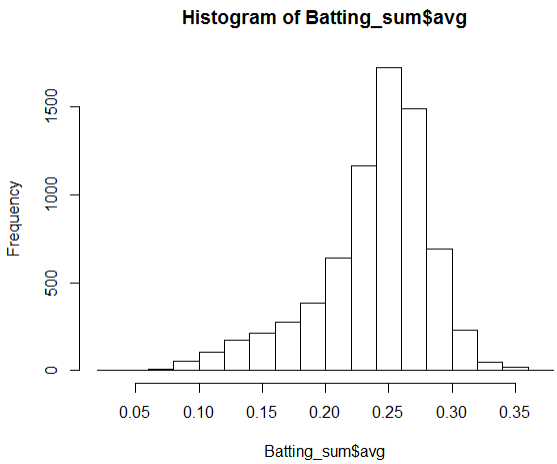
.

Here using the 'Batting' dataset from 'Lahman' package, we have a dataset that has individual observations on each row (i.e. not summary frequency counts).

We can create histograms directly from this.

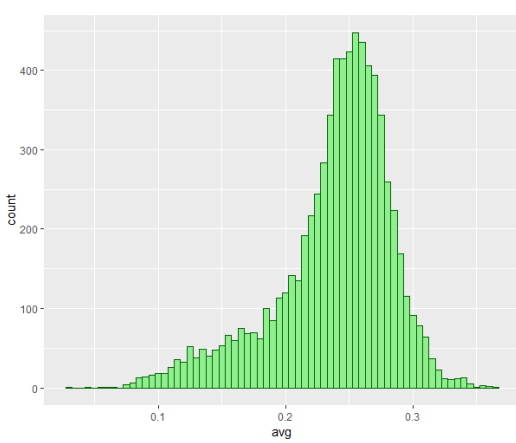
```
# A tibble: 6 x 4
  playerID  totalH totalAB   avg
  <chr>      <int>   <int> <dbl>
1 aaronha01   3771   12364 0.305
2 aaronto01    216     944 0.229
3 abbated01    772    3044 0.254
4 abbeybe01     38     225 0.169
5 abbeych01    492    1751 0.281
6 abbotfr01    107     513 0.209
```

```
hist(Batting_sum$avg) #quick look using base-r
```

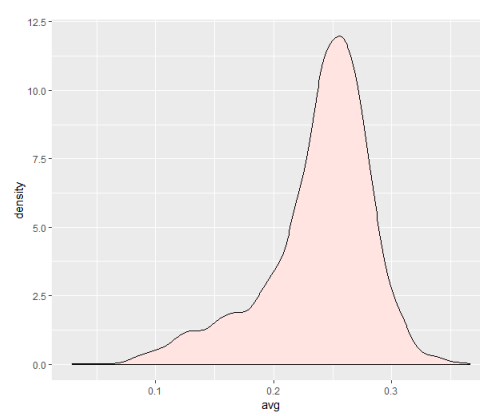


Again, these base-r plots can be modified (we'll have some examples in the tutorial) but to create beautiful plots we should use ggplot2

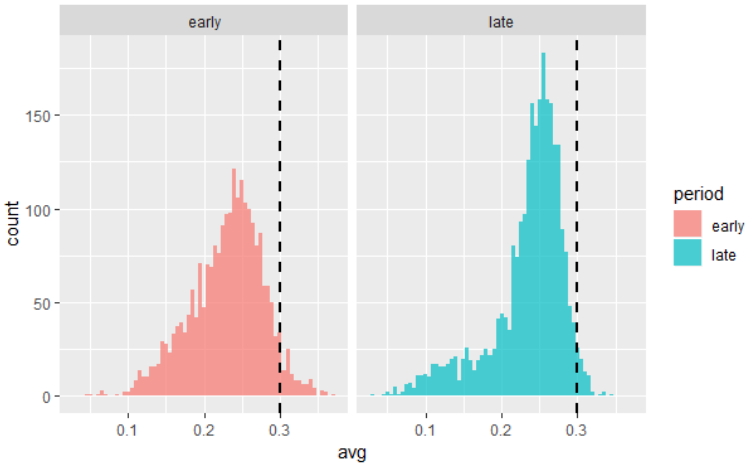
```
ggplot(Batting_sum, aes(x=avg)) +
  geom_histogram(binwidth = .005, color='darkgreen',fill='lightgreen')
```




```
ggplot(Batting_sum, aes(x=avg)) + geom_density(fill='mistyrose')
```



```
ggplot(Batting_all, aes(x=avg, fill=period)) +
  geom_histogram(position="identity", alpha=.7, binwidth=.005) +
  facet_wrap(~period) +
  geom_vline(aes(xintercept=0.3), color="black", linetype="dashed", size=1)
```



Further customization of histograms

facet_wrap() – makes panel

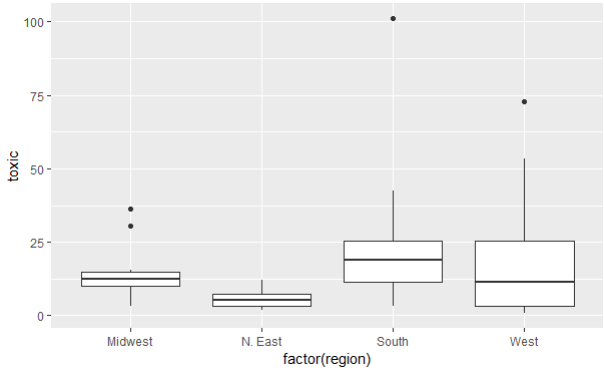
geom_vline() – adds vertical line at .3

postion="identity" – ensures consistency of x-scale

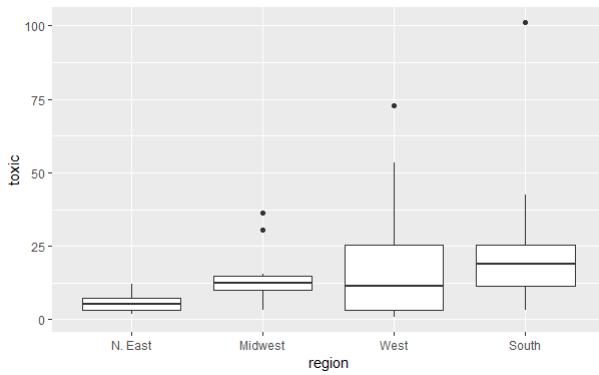
binwidth=.005 - determines size of each bin

Making boxplots in ggplot2

```
ggplot(states, aes(x=factor(region), y=toxic)) + geom_boxplot()
```

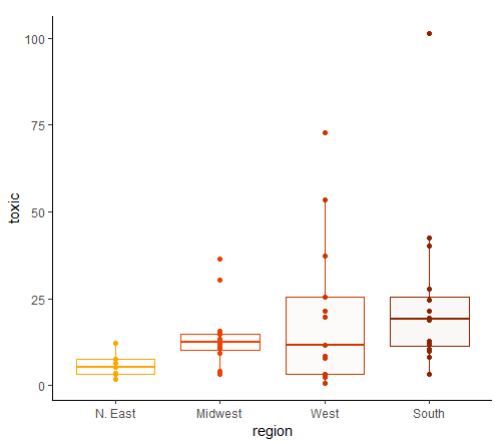


```r
# one way to define the order on the x-axis is as follows:
levels(states$region)
states$region <- factor(states$region, levels=c("N. East", "Midwest", "West", "South"))

ggplot(states, aes(x=region, y=toxic)) + geom_boxplot()
```



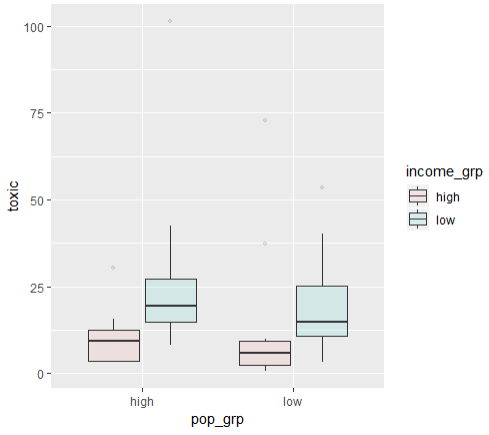
Redefining factor levels to ensure preferred order on x-axis

```r
ggplot(states, aes(x=region, y=toxic, fill=region, color=region)) +
  geom_boxplot(alpha=.1, outlier.size=1) +
  scale_color_manual(values=c("orange","orangered2","orangered3","orangered4"))+
  scale_fill_manual(values=c("lavenderblush","mistyrose1","mistyrose2","mistyrose3")) +
  theme_classic() +
  theme(legend.position='none') +
  geom_point()
```



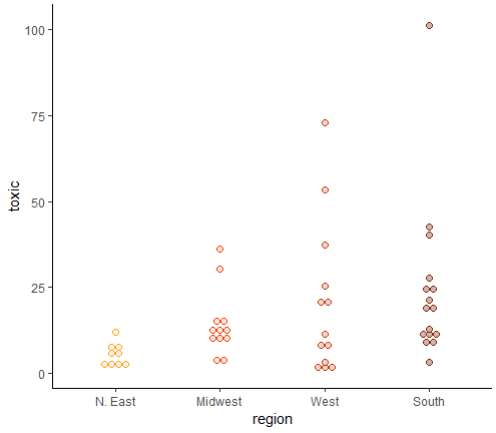
Overlaying raw data over boxplots can be a very useful way of demonstrating the distribution of data within and between groups

```
## Comparing two variables:

head(states)
states$income_grp <- ifelse(states$income>median(states$income), 'high', 'low')
states$pop_grp <- ifelse(states$pop>median(states$pop,na.rm=T), 'high', 'low')
table(states$income_grp, states$pop_grp)


ggplot(states, aes(x=pop_grp, y=toxic, fill=income_grp)) + geom_boxplot(alpha=.1, outlier.size=1)
```



Adding a “fill” term inside “aes” allows us to compare boxplots across two factors

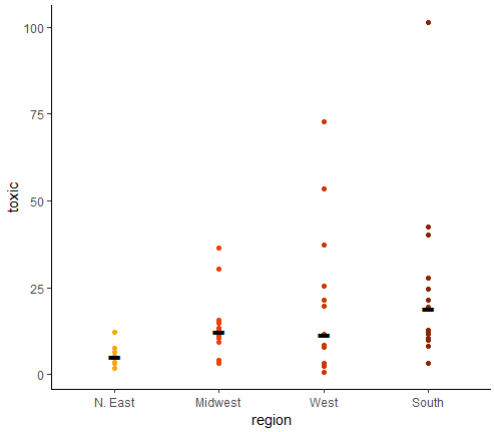
```
ggplot(states, aes(x=region, y=toxic, fill=region, color=region)) +
  scale_color_manual(values=c("orange","orangered2","orangered3","orangered4"))+
  scale_fill_manual(values=c("lavenderblush","mistyrose1","mistyrose2","mistyrose3")) +
  theme_classic() +
  theme(legend.position='none') +
  geom_dotplot(binaxis = "y", stackdir = "center", binwidth = 2)
```



dotplot (beeswarm) plots

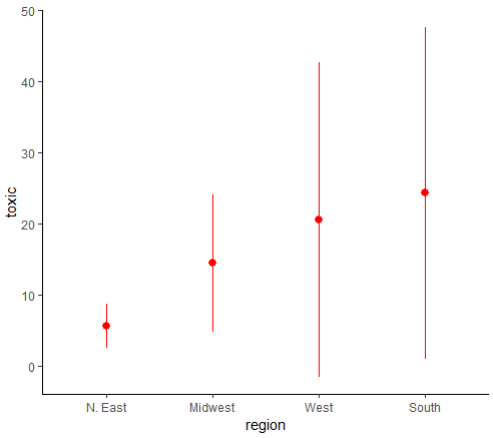
```
p <- ggplot(states, aes(x=region, y=toxic, fill=region, color=region)) +
  scale_color_manual(values=c("orange","orangered2","orangered3","orangered4"))+
  scale_fill_manual(values=c("lavenderblush","mistyrose1","mistyrose2","mistyrose3")) +
  theme_classic() +
  theme(legend.position='none')
```

```
# superimpose summary stat e.g. median.
p + geom_point() +
  stat_summary(fun.y = median, geom = "point", size = 11, color='black', pch="-")
```



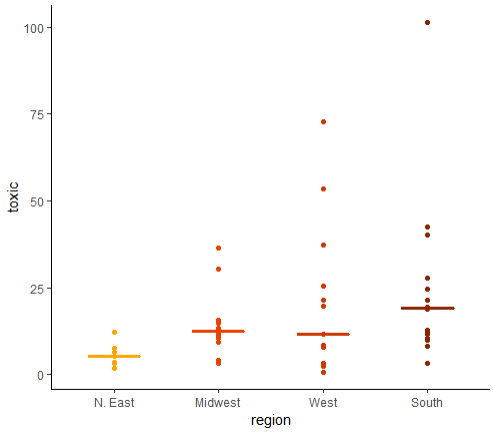
Using “stat_summary()” to overlay summary stats

```
#Mean and Standard Deviation
p + stat_summary(fun.data=mean_sdl, fun.args = list(mult=1),
                 geom="pointrange", color="red")
```



Using “stat_summary()” to overlay summary stats

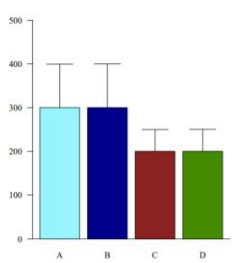
```r
## An alternative way using geom_crossbar:

# first you need another summary datfame of what you want to plot:
library(tidyverse)
states_summary <- states %>% group_by(region) %>% summarise(med = median(toxic, na.rm=T))
states_summary

p + geom_point() +
  geom_crossbar(data=states_summary, aes(x=region, group=region, y=med, ymin=med, ymax=med), width=0.5)
```



Beware of Dynamite

Why Are Dynamite Plots Bad?

The height of the bar represents the average, and the whisker the standard deviation (or standard error). They do resemble the dynamite you may see in cartoons, don't they? Why are they bad?



So little information! This plot presents 4 means and 4 standard deviations (or standard errors). This is a very inefficient use of space.

What do means mean? Averages do not usually convey much information. How spread are the data? Are there outliers? What are the sample sizes? None of these interesting and important questions is answered.

Whiskers get in the way. Whiskers add some information, maybe, but I don't know how to use them. They make the bars look taller, and the little information given by the bars is distorted.

Where are the data? There are data behind the bar, but the bar also covers space where no data exist. In addition, there must be some data above the bar in all that blank space.
Where are the data? You learn very little about the data and their *distribution*.

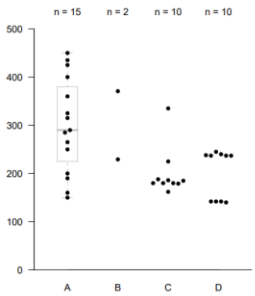
What might we conclude from this dynamite plot?
• Groups A and B are identical. • Groups C and D are identical. • Groups A and B are more spread than C and D.

http://biostat.mc.vanderbilt.edu/wiki/pub/Main/TatsukiRcode/Poster3.pdf

Beware of Dynamite

What Should We Use Instead of Dynamite Plots?

We ought to show every single data point unless the sample size is too big. In that case, some summary measures may be used, but very rarely is a single average sufficient to represent the entire dataset. I like showing every single data point as a dot. Alternatively, a box plot showing the median, quartiles, and outliers may be used for a larger dataset.



Here's the same data set shown in a way that reveals the individual points and their *distributions*.

- Are groups A and B identical?
- Are groups C and D identical?
- Are groups A and B more spread than C and D?

Furthremore, why are there only 2 data in group B? What is special about the one datum in group C that stands alone at the top? Are there two distinct groups within group D? What is going on?

Obviously, this is a made-up example. Mean-spirited, I might add. The fundamental problem with the dynamite plot is that it allows such a malicious example to be constructed. If all data points are shown, this is not possible. Intentionally or not, a dynamite plot hides more than it reveals!

http://biostat.mc.vanderbilt.edu/wiki/pub/Main/TatsukiRcode/Poster3.pdf

Making dynamite plots in ggplot2

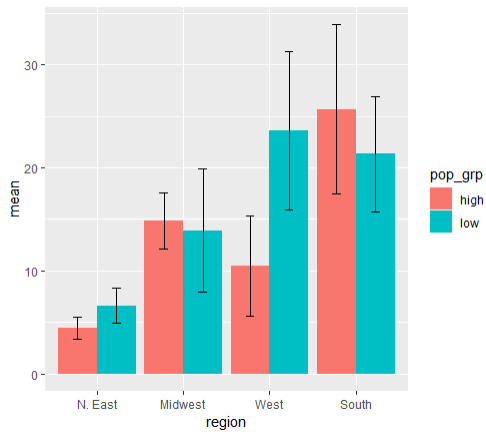
```
states$pop_grp <- ifelse(states$pop>median(states$pop,na.rm=T), 'high', 'low')

# 1. summarise data -  mean, median, sd, se, confindence interval
sem <- function(x) sqrt(var(x,na.rm=T)/length(x))

summary(states$toxic)

states_summary <-
  states %>%
  group_by(region, pop_grp) %>%
  summarise(
    mean=mean(toxic,na.rm=T),
    sd=sd(toxic,na.rm=T),
    se=sem(toxic),
    n = length(toxic)
  )

states_summary
```

```
> states_summary
# A tibble: 8 x 6
# Groups:   region [?]
  region  pop_grp  mean    sd    se     n
  <fct>   <chr>   <dbl> <dbl> <dbl> <int>
1 N. East high     4.42  2.06  1.03     4
2 N. East low      6.62  3.72  1.67     5
3 Midwest high    14.8   7.19  2.72     7
4 Midwest low     13.9  13.4   5.98     5
5 West    high    10.5   8.37  4.83     3
6 West    low     23.6  24.3   7.68    10
7 South   high    25.7  27.3   8.23    11
8 South   low     21.3  12.4   5.56     5
>
```

The easiest way to do this is to summarize data first using e.g. dplyr, and then produce the graph.

Use **geom_bar()** to plot means across groups
Use **geom_errorbar()** to plot error bars

```
ggplot(states_summary, aes(x=region, y=mean, fill=pop_grp)) +
  geom_bar(position=position_dodge(), stat="identity") +
  geom_errorbar(aes(ymin=mean-se, ymax=mean+se),
                width=.2,
                position=position_dodge(.9))
```

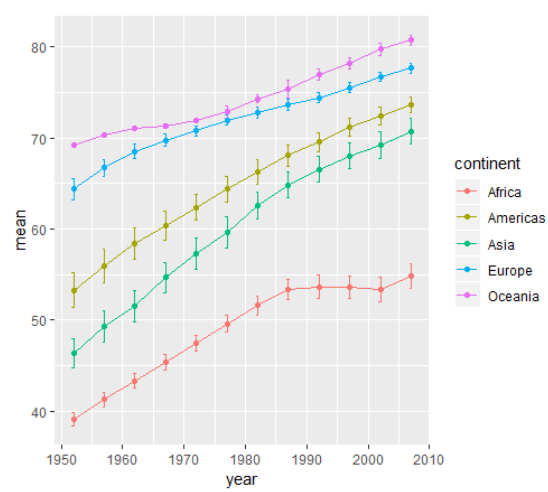


Using fill=pop_grp means we are plotting 2 bars per region (one for 'high' and one for 'low').

We therefore use 'postion=position_dodge()' and stat="identity"
to put them side by side rather than having them stack on top of each other

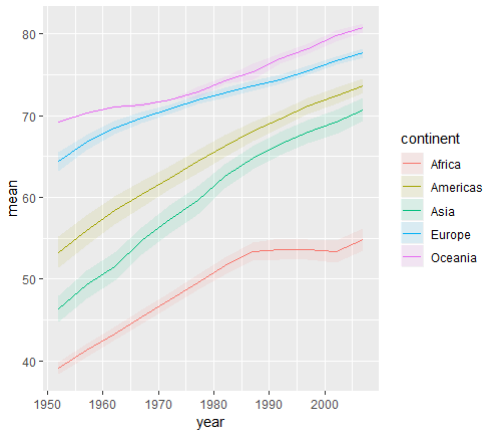
Likewise, for line graphs – we just need to summarize the data first

We can use **geom_errorbar()**




```
### By continent and year:
gapminder_yr <-
  gapminder %>%
  group_by(year, continent) %>%
  summarise(
    mean=mean(lifeExp,na.rm=T),
    sd=sd(lifeExp,na.rm=T),
    se=sem(lifeExp),
    n = length(lifeExp)
  )

gapminder_yr

ggplot(gapminder_yr, aes(x=year, y=mean, color=continent)) +
  geom_errorbar(aes(ymin=mean-se, ymax=mean+se), width=.5, position="identity") +
  geom_line() +
  geom_point()
```

```
ggplot(gapminder_yr, aes(x=year, y=mean, color=continent, fill=continent)) +
  geom_ribbon(aes(ymin = mean-se, ymax = mean+se), alpha=.1, color=NA) +
  geom_line(aes(y = mean))
```



Sometimes using ribbons (shaded regions) rather than error bars are more aesthetically pleasing.

This can be done using **geom_ribbon()**

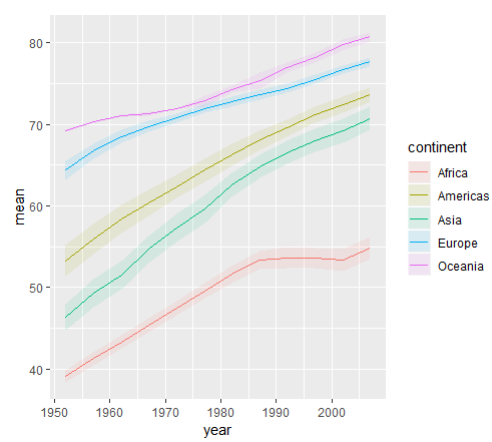
Chaining

One last thing… - the beauty of the pipe is that we don't have to reassign subsetted data to a new object before plotting. We can go directly from the raw data to the plot like this:

```
### Chaining

gapminder %>%
  group_by(year, continent) %>%
  summarise(
    mean=mean(lifeExp,na.rm=T),
    sd=sd(lifeExp,na.rm=T),
    se=sem(lifeExp),
    n = length(lifeExp)
  )%>%
ggplot(aes(x=year, y=mean, color=continent, fill=continent)) +
  geom_ribbon(aes(ymin = mean-se, ymax = mean+se), alpha=.1, color=NA) +
  geom_line(aes(y = mean))
```

Here we just directly chain the **dplyr** operations with the **ggplot2** operations

No need to tell it what data to use – it uses what comes immediately before it in the chain



3. Parametric Statistics

3a. Testing normality and variances of distributions

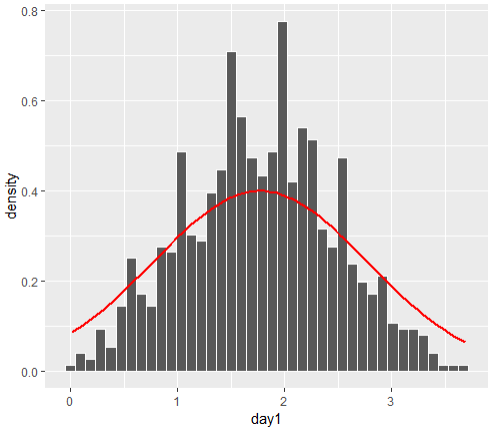
Parametric statistics often have specific assumptions. Two common ones are that data are normally distributed and there is homogeneity of variance across groups.

1. Visual inspection of the normality of the distribution

```
> dlf <- read.csv("dlf.csv")
> dim(dlf)
[1] 809   5
> head(dlf)
  ticknumb gender day1 day2 day3
1     2111   Male 2.64 1.35 1.61
2     2229 Female 0.97 1.41 0.29
3     2338   Male 0.84   NA   NA
4     2384 Female 3.03   NA   NA
5     2401 Female 0.88 0.08   NA
6     2405   Male 0.85   NA   NA
>
```

Example dataset shows the hygiene level (lower scores = worse) for festival goers over 3 days of a festival.

```
ggplot(dlf, aes(x = day1)) + geom_histogram(aes(y = ..density..), color='white', bins=40) +
  stat_function(fun = dnorm,
                args = list(mean = mean(dlf$day1, na.rm=T, sd = sd(dlf$day1, na.rm=T))),
                color="red",
                size = 1)
```

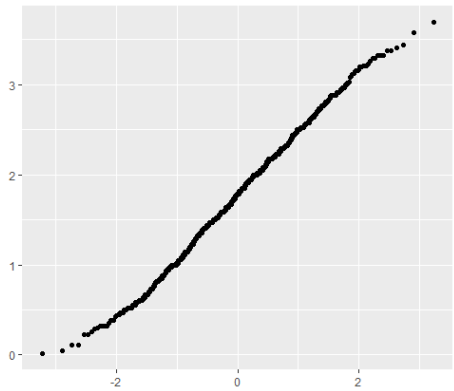


To a histogram plot – we can add a normal distribution curve for a given mean and standard deviation

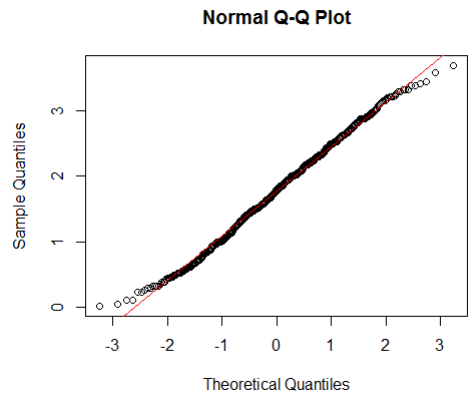
This enables us to make some informed decisions about the likelihood of the data being normally distributed.

2. The QQ plot plots the cumulative values of our data against the theoretical cumulative values from a normal distribution

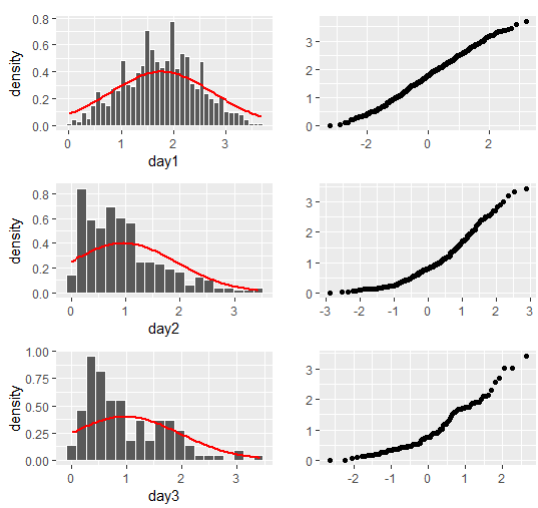
Large deviations from a straight line are indicative of a non-normal distribution (some tailing off at the extremes is typical)




```
qplot(sample = dlf$day1)
```




```
qqnorm(dlf$day1)
qqline(dlf$day1, col="red") # adds a line to existing one
```




```
# Days 1, 2 and 3 together

#devtools::install_github("thomasp85/patchwork")
library(patchwork)
library(ggplot2)


p1a <- ggplot(dlf, aes(x = day1)) + geom_histogram(aes(y = ..density..), color='white', bins=40) +
  stat_function(fun = dnorm, args = list(mean = mean(dlf$day1, na.rm=T), sd = sd(dlf$day1, na.rm=T))),
                color="red",size = 1)

p2a <- ggplot(dlf, aes(x = day2)) + geom_histogram(aes(y = ..density..), color='white', bins=20) +
  stat_function(fun = dnorm, args = list(mean = mean(dlf$day2, na.rm=T), sd = sd(dlf$day2, na.rm=T))),
                color="red",size = 1)

p3a <- ggplot(dlf, aes(x = day3)) + geom_histogram(aes(y = ..density..), color='white', bins=20) +
  stat_function(fun = dnorm, args = list(mean = mean(dlf$day3, na.rm=T), sd = sd(dlf$day3, na.rm=T))),
                color="red",size = 1)


p1b <- qplot(sample = dlf$day1)
p2b <- qplot(sample = dlf$day2)
p3b <- qplot(sample = dlf$day3)

# layout in a 3 x 2 grid
(p1a | p1b) /(p2a | p2b) / (p3a | p3b)
```

The library ‘patchwork’ provides a nice way of putting multiple variables next to each other

Here day2 and day3 appear to have non-normally distributed data

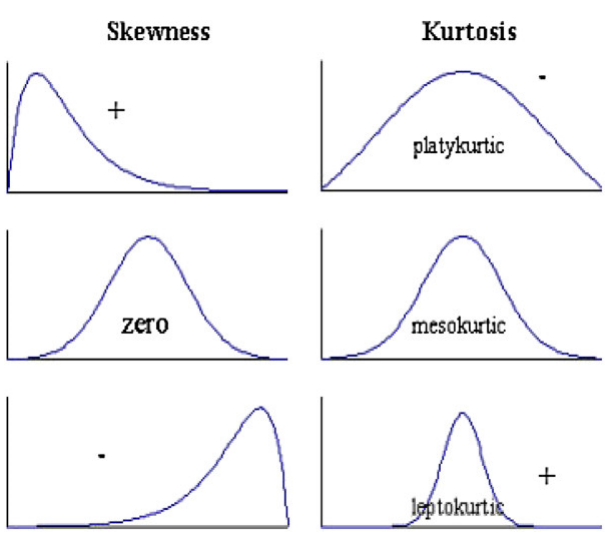
3. We should also examine the kurtosis and skewness of the distributions

These can be done using the following functions:

```
> psych::describe(dlf$day1)
   vars   n mean   sd median trimmed mad  min  max range skew kurtosis   se
X1    1 809 1.77 0.69   1.79    1.77 0.7 0.02 3.69  3.67    0    -0.42 0.02
```

```
> round(pastecs::stat.desc(dlf[,3:5], basic = F, norm = T), 3)
               day1  day2  day3
median        1.790 0.790 0.760
mean          1.771 0.955 0.977
SE.mean       0.024 0.044 0.064
CI.mean.0.95  0.048 0.087 0.127
var           0.482 0.513 0.504
std.dev       0.694 0.716 0.710
coef.var      0.392 0.750 0.727
skewness     -0.003 1.097 1.008
skew.2SE     -0.018 3.653 2.309
kurtosis     -0.424 0.829 0.595
kurt.2SE     -1.235 1.386 0.686
normtest.W    0.996 0.908 0.908
normtest.p    0.032 0.000 0.000
```

```
# skew and kurtosis should be 0 in a truly normal distribution.
# positive skew values mean a long right tail (more scores on left of distribution)
# positive kurtosis values = pointy and heavy-tailed distribution
# negative kurtosis values = flatter and thinner-tailed distribution

# convert skew or kurtosis to a Z score (S-0 / SES, K-0 / SEK) to evaluate significance
# here, if skew.2SE/kurt.2SE > 1 (or < -1) then likely significant
# for alpha = .01 cut-off is 1.29, for alpha = 0.001 cut-off is 1.65
# this holds true only for smaller samples (<200)

# normtest.W   normtest.p  - these are outcomes of Shapiro-Wilk test (discussed in a moment)
```



Example Dataset:

```
> exam <- read.csv('exam.csv')
> head(exam)
  exam computer lectures numeracy   uni
1   18       54     75.0        7 Uni-A
2   30       47      8.5        1 Uni-A
3   40       58     69.5        6 Uni-A
4   30       37     67.0        6 Uni-A
5   40       53     44.5        2 Uni-A
6   15       48     76.5        8 Uni-A
> str(exam)
'data.frame':   100 obs. of  5 variables:
 $ exam    : int  18 30 40 30 40 15 36 40 63 31 ...
 $ computer: int  54 47 58 37 53 48 49 49 45 62 ...
 $ lectures: num  75 8.5 69.5 67 44.5 76.5 70 18.5 43.5 100 ...
 $ numeracy: int  7 1 6 6 2 8 3 7 4 6 ...
 $ uni     : Factor w/ 2 levels "Uni-A","Uni-B": 1 1 1 1 1 1 1 1 1 1 ...
>
```

When using statistics that compare across individual groups, it is important to check that data is normally distributed within each group - this can be done using **by()** as we saw earlier:

```
> by(data = exam$exam, INDICES = exam$uni, FUN = psych::describe)
exam$uni: Uni-A
   vars  n  mean    sd median trimmed  mad min max range skew kurtosis   se
X1    1 50 40.18 12.59     38   39.85 12.6  15  66    51 0.29    -0.72 1.78
------------------------------------------------------------------------------------
exam$uni: Uni-B
   vars  n  mean    sd median trimmed mad min max range skew kurtosis   se
X1    1 50 76.02 10.21     75    75.7 8.9  56  99    43 0.26    -0.46 1.44
> by(data = exam$exam, INDICES = exam$uni, FUN = pastecs::stat.desc, basic = F, norm = T)
exam$uni: Uni-A
      median         mean      SE.mean CI.mean.0.95          var      std.dev     coef.var     skewness
  38.0000000   40.1800000    1.7803210    3.5776890  158.4771429   12.5887705    0.3133094    0.2906760
    skew.2SE     kurtosis     kurt.2SE   normtest.W   normtest.p
   0.4317816   -0.7230849   -0.5462122    0.9721662    0.2828984
------------------------------------------------------------------------------------
exam$uni: Uni-B
      median         mean      SE.mean CI.mean.0.95          var      std.dev     coef.var     skewness
  75.0000000   76.0200000    1.4432079    2.9002348  104.1424490   10.2050208    0.1342413    0.2559866
    skew.2SE     kurtosis     kurt.2SE   normtest.W   normtest.p
   0.3802527   -0.4609644   -0.3482086    0.9837115    0.7151182
```

4. Shapiro-Wilk Test - Testing if a distribution could come from a normal distribution of a given mean and SD.

```
# compare sample to a normal distribution with mean and sd same as the sample
# if p<.05 distributions significantly differ
# can be overly sensitive with large samples
# combine with visual inspection (histograms, qqplot) and evaluation of skew and kurtosis

shapiro.test(exam$exam)
shapiro.test(exam$numeracy)
```

```
> shapiro.test(exam$exam)

        Shapiro-Wilk normality test

data:  exam$exam
W = 0.96131, p-value = 0.004991

> shapiro.test(exam$numeracy)

        Shapiro-Wilk normality test

data:  exam$numeracy
W = 0.92439, p-value = 2.424e-05
```

As with skewness and kurtosis, we should also perform the Shapiro-Wilk test across groups:

```
> by(data = exam$exam,
+    INDICES = exam$uni,
+    FUN = shapiro.test) # when comparing groups, need to know the distribution within each group
exam$uni: Uni-A

        Shapiro-Wilk normality test

data:  dd[x, ]
W = 0.97217, p-value = 0.2829

------------------------------------------------------------------------------------
exam$uni: Uni-B

        Shapiro-Wilk normality test

data:  dd[x, ]
W = 0.98371, p-value = 0.7151

> by(data = exam$numeracy,
+    INDICES = exam$uni,
+    FUN = shapiro.test)
exam$uni: Uni-A

        Shapiro-Wilk normality test

data:  dd[x, ]
W = 0.94082, p-value = 0.01452

------------------------------------------------------------------------------------
exam$uni: Uni-B

        Shapiro-Wilk normality test

data:  dd[x, ]
W = 0.93235, p-value = 0.006787
```

There are many 'normality' tests – and different software packages include different versions. Most people recommend using Shapiro-Wilk, however, here is a quick summary of how to perform some of the other commonly requested normality tests:

```
### Some Normality Test Extensions ----

library(nortest)

# Most statisticians suggest using Shapiro-Wilk (along with visual inspection and skew/kurtosis),
# as it has more power than others - but several others exist:
# e.g. Kolmogorov Smirnov Test (though sensitive to outliers)
# e.g. Lilliefors Test (less conservative than KS)

nortest::lillie.test(exam$exam) #Lillie
nortest::ad.test(exam$exam) #Anderson-Darling


by(data = exam$exam, INDICES = exam$uni, FUN = nortest::lillie.test)
by(data = exam$exam, INDICES = exam$uni, FUN = nortest::ad.test)

# the package 'nortest' has several other normality tests, as does the package 'normtest'

# 'normtest' also includes some more robust skewness/kurtosis tests:


library(normtest)

by(data = exam$exam, INDICES = exam$uni, FUN = nortest::lillie.test)

examA <- subset(exam, uni == "Uni-A")
examB <- subset(exam, uni == "Uni-B")

normtest::kurtosis.norm.test(examA$numeracy, nrepl=2000)  #p>.05,  ok for kurtosis
normtest::skewness.norm.test(examA$numeracy, nrepl=2000)  #p>.05,  ok for skewness
```

5. Levene's Test for Homogeneity of Variance

Use the **leveneTest()** function from the package 'car':

```
# Test for homogeneity of variance across all levels of a variable
# Essentially a one-way anova across groups on deviations of each score from its group mean

car::leveneTest(exam$exam, exam$uni) #1st = continuous outcome variable, 2nd = grouping variable

# if p-value <.05 suggests that there are differences in variances across groups (i.e. heterogeneity)

# also has issues with being overly sensitive for large sample sizes
# (power of test leads to increased sensitivity)
```

```
> car::leveneTest(exam$exam, exam$uni) #1st = continuous outcome variable, 2nd = grouping variable
Levene's Test for Homogeneity of Variance (center = median)
      Df F value Pr(>F)
group  1  2.0886 0.1516
      98
>
```

An alternative is to calculate Hartley's Fmax (see additional material)

3b. Paired and unpaired T-tests

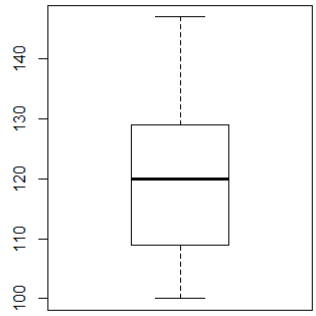
Example Dataset:

```
> df <- read.csv("bloodwork.csv")
> str(df)
'data.frame':   30 obs. of  11 variables:
 $ ids        : Factor w/ 30 levels "AAA","ACC","AGC",..: 28 21 15 1 4 19 26 5 6 25 ...
 $ age        : int  19 20 21 21 24 25 26 28 28 29 ...
 $ sex        : Factor w/ 2 levels "female","male": 1 2 1 1 1 1 1 1 1 1 ...
 $ state      : Factor w/ 3 levels "CT","NJ","NY": 2 3 2 1 2 3 3 1 2 1 ...
 $ children   : int  1 1 0 3 0 1 2 0 0 0 ...
 $ smoker     : Factor w/ 2 levels "no","yes": 1 2 1 1 1 2 1 1 1 2 ...
 $ hrate      : int  63 73 65 66 67 80 71 62 68 74 ...
 $ bpsyst     : int  101 120 100 109 108 118 118 104 118 117 ...
 $ cellcount  : num  0.126 0.169 0.281 0.244 0.092 0.014 0.093 0.082 0.197 0.429 ...
 $ immuncount : num  0.993 1.176 4.336 2.558 6.446 ...
 $ immuncount2: num  0.921 1.194 3.213 4.012 9.128 ...
> head(df)
  ids age    sex state children smoker hrate bpsyst cellcount immuncount immuncount2
1 JEC  19 female    NJ        1     no    63    101     0.126      0.993       0.921
2 GBH  20   male    NY        1    yes    73    120     0.169      1.176       1.194
3 EDH  21 female    NJ        0     no    65    100     0.281      4.336       3.213
4 AAA  21 female    CT        3     no    66    109     0.244      2.558       4.012
5 AJF  24 female    NJ        0     no    67    108     0.092      6.446       9.128
6 FJC  25 female    NY        1    yes    80    118     0.014      3.972       2.848
>
```

The function **t.test()** can be used for all t-tests (one-sample, two-sample, paired and unpaired)

A one-sample t-test:

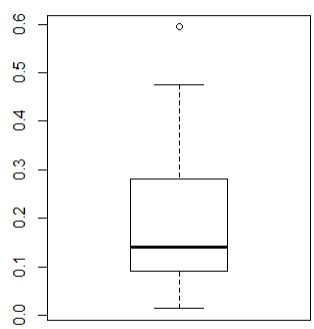
```
> boxplot(df$bpsyst)
> t.test(df$bpsyst, alternative = "less", mu = 130) # is mean < 130 ?

        One Sample t-test

data:  df$bpsyst
t = -4.1518, df = 29, p-value = 0.0001323
alternative hypothesis: true mean is less than 130
95 percent confidence interval:
     -Inf 124.4273
sample estimates:
mean of x
 120.5667
```



When performing quick statistical procedures, it's often easiest to make quick base-r graphics. Here, we are using **boxplot()** to plot a boxplot.

Use **wilcox.test()** for a non-parametric alternative to one-sample t-test:

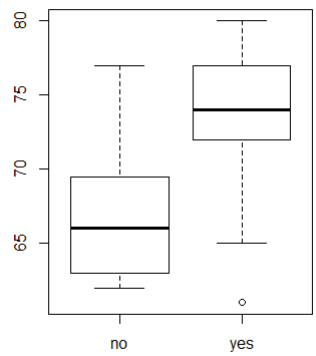
```
> boxplot(df$cellcount)
> wilcox.test(df$cellcount, mu=0.1, alternative = "greater")

        Wilcoxon signed rank test with continuity correction

data:  df$cellcount
V = 366, p-value = 0.003113
alternative hypothesis: true location is greater than 0.1

Warning message:
In wilcox.test.default(df$cellcount, mu = 0.1, alternative = "greater") :
  cannot compute exact p-value with ties
>
```



Independent t-tests:

The tilde can be pronounced as "is predicted by"

```
### Independent t-tests ----

boxplot(df$hrate ~ df$smoker)  # quick simple boxplot


t.test(df$hrate ~ df$smoker)

t.test(hrate ~ smoker, data=df) # alternative way of writing it

t.test(hrate ~ smoker, df) # same as above
```



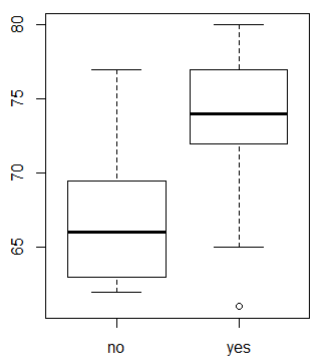

```
        Welch Two Sample t-test

data:  df$hrate by df$smoker
t = -3.8114, df = 26.802, p-value = 0.0007342
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -10.667156  -3.199511
sample estimates:
 mean in group no mean in group yes
         67.00000          73.93333
```

Note: R automatically applies the Welch's correction for unequal variances between groups

R output includes confidence intervals of true mean difference

Independent t-tests:

```
t.test(df$hrate ~ df$smoker, var.equal=T)  #the default is actually for variances not to be equa
#unequal variances assumed is the Welch's correction
```

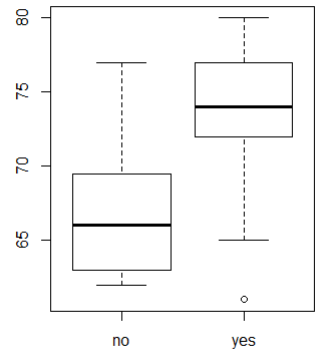


Note: R automatically applies the Welch's correction for unequal variances between groups

You can perform the independent t-test by removing Welch's correction – though in practice this isn't a good idea. There is no real benefit to removing it when variances are equal – so best to keep in place.

Independent t-tests:

```
> t.test(df$hrate ~ df$smoker, var.equal=T, alternative="less")

        Two Sample t-test

data:  df$hrate by df$smoker
t = -3.8114, df = 28, p-value = 0.0003476
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
      -Inf -3.838768
sample estimates:
 mean in group no mean in group yes
         67.00000          73.93333
```



To perform a one-tailed test, you need to add the argument **alternative="less"** or **alternative="greater"**

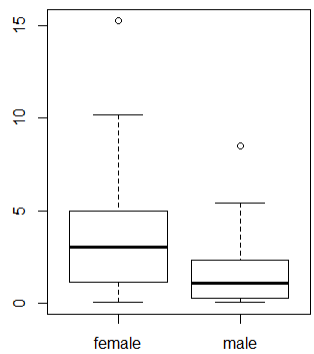
Calculating Effect Sizes

We can calculate the effect size a t-test using the formula $$r = \sqrt{\frac{t^2}{t^2 + df}}$$

```
# Calculate effect sizes
ttest1 <- t.test(hrate ~ smoker, data=df, var.equal=T)
t <- ttest1$statistic[[1]]  #[1] -3.811362
df <- ttest1$parameter[[1]] #[1] 28
r <- sqrt(t^2 / (t^2 + df)) #[1] 0.5844542
```

Use **wilcox.test()** for a non-parametric alternative to independent t-test:

```
> by(df$immuncount2, df$sex, shapiro.test) #not normal in each group
df$sex: female

        Shapiro-Wilk normality test

data:  dd[x, ]
W = 0.8363, p-value = 0.005188

------------------------------------------------------------------------
df$sex: male

        Shapiro-Wilk normality test

data:  dd[x, ]
W = 0.75386, p-value = 0.002933

> boxplot(df$immuncount2~df$sex)
>
> wilcox.test(df$immuncount2~df$sex)

        Wilcoxon rank sum test

data:  df$immuncount2 by df$sex
W = 154, p-value = 0.0531
alternative hypothesis: true location shift is not equal to 0
```




```
> #can use lapply to run multiple t-tests at once (if this is desirable and ok to do)
> lapply(df[,7:11], function(x) t.test(x ~ df$smoker, var.equal = TRUE))
$`hrate`

        Two Sample t-test

data:  x by df$smoker
t = -3.8114, df = 28, p-value = 0.0006952
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -10.65964  -3.20703
sample estimates:
 mean in group no mean in group yes
         67.00000          73.93333


$bpsyst

        Two Sample t-test

data:  x by df$smoker
t = -2.3502, df = 28, p-value = 0.02604
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -18.590972  -1.275695
sample estimates:
 mean in group no mean in group yes
         115.6000          125.5333


$cellcount

        Two Sample t-test
```

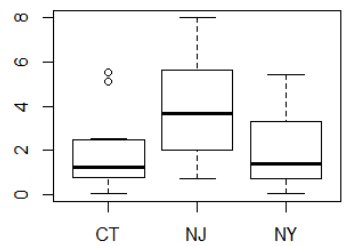
Performing multiple t-tests at once using **lapply()**

Note – this does not apply any correction to the p-value threshold

Whether running multiple tests is satisfactory is a researcher decision

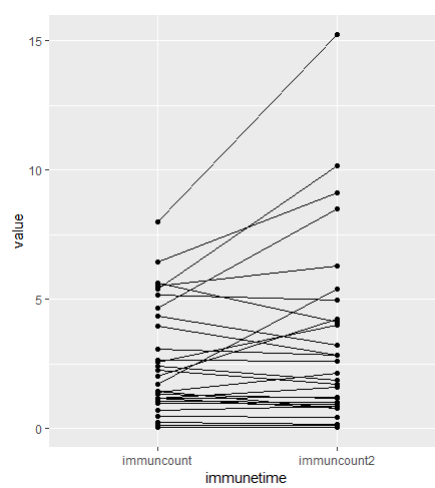
Use **pairwise.t.test()** to perform multiple t-tests between levels of a factor and adjust p-value

```
> boxplot(immuncount ~ state, data=df)
> ## when have more than 2 levels of a factor
> boxplot(immuncount ~ state, data=df)
> pairwise.t.test(df$immuncount,df$state , p.adjust.method="holm")

        Pairwise comparisons using t tests with pooled SD

data:  df$immuncount and df$state

   CT   NJ
NJ 0.11 -
NY 0.88 0.15

P value adjustment method: holm
>
```



Add **paired=TRUE** to perform a paired t-test between two vectors

```
> df %>% gather(immunetime,value,10:11) %>%
+ ggplot(aes(x=immunetime, y=value, group=ids)) +
+          geom_point() + geom_line()
> t.test(df$immuncount, df$immuncount2, alternative=c("two.sided"), paired=TRUE)

        Paired t-test

data:  df$immuncount and df$immuncount2
t = -1.9923, df = 29, p-value = 0.05583
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.43346773  0.01880106
sample estimates:
mean of the differences
              -0.7073333

> wilcox.test(df$immuncount, df$immuncount2, paired=TRUE) # where y1 and y2 are numeric

        Wilcoxon signed rank test

data:  df$immuncount and df$immuncount2
V = 202, p-value = 0.5425
alternative hypothesis: true location shift is not equal to 0
```



Add paired=TRUE to the wilcox.test() for the non-parametric equivalent

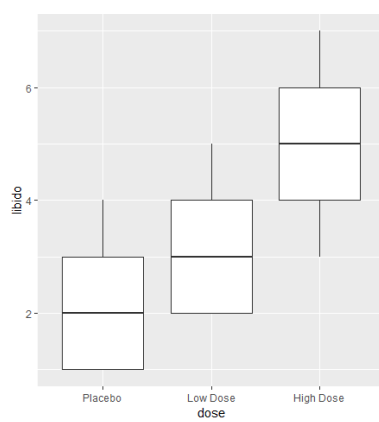
3c. Between-Subjects ANOVA

- One-Way ANOVA
- Analysis of Covariance
- Two-Way ANOVA

Example dataset:

```
> libido <- c(3,2,1,1,4,5,2,4,2,3,7,4,5,3,6)
> dose<-gl(3,5,labels=c("Placebo","Low Dose","High Dose"))
> viagra <- data.frame(dose,libido)
> viagra
        dose libido
1    Placebo      3
2    Placebo      2
3    Placebo      1
4    Placebo      1
5    Placebo      4
6   Low Dose      5
7   Low Dose      2
8   Low Dose      4
9   Low Dose      2
10  Low Dose      3
11 High Dose      7
12 High Dose      4
13 High Dose      5
14 High Dose      3
15 High Dose      6
```

```
# visualize data with plot
ggplot(viagra, aes(x=dose, y = libido)) +
  geom_boxplot()
```



Prior to an ANOVA, you should visualize data and perform checks of assumptions

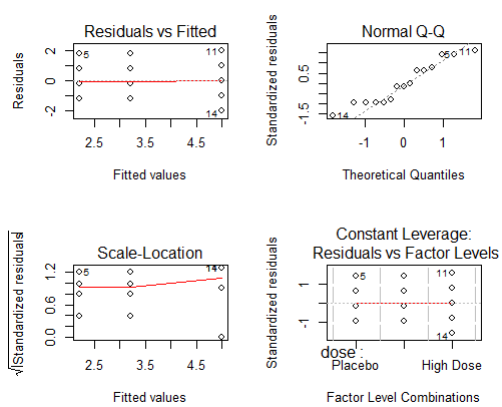
Use **aov()** to produce a model

```
> mod1 <- aov(libido ~dose, data=viagra)
> mod1
Call:
   aov(formula = libido ~ dose, data = viagra)

Terms:
                    dose Residuals
Sum of Squares  20.13333  23.60000
Deg. of Freedom       2        12

Residual standard error: 1.402379
Estimated effects may be unbalanced
> summary(mod1)
            Df Sum Sq Mean Sq F value Pr(>F)
dose         2  20.13  10.067   5.119 0.0247 *
Residuals   12  23.60   1.967
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
>
```

aov() is a simple method of generating the typical statistical output for a one-way ANOVA

Checking diagnostic plots using **plot()**

```
> ## Diagnostics
> par(mfrow=c(2,2))
> plot(mod1)
>
```



For all ANOVAs and regression models we can use plot() to investigate diagnostics.

In brief, the two plots on the left side are examining if there residuals vary across fitted values.

The top-right plot is the typical QQ plot.

The bottom-right is a plot that examines leverage of individual points (we discuss more in linear regression module).

Use **oneway.test()** to produce a model that includes a correction for unequal variances (heterogeneity)

```
> mod2 <- oneway.test(libido ~dose, data=viagra)
> mod2

        One-way analysis of means (not assuming equal variances)

data:  libido and dose
F = 4.3205, num df = 2.0000, denom df = 7.9434, p-value = 0.05374
```

Planned Contrasts -

If you wish to perform some planned contrasts (or comparisons), you can set the contrasts of a variable. Using this method, you would perform a linear regression (which is equivalent to an ANOVA in this case) using the function **lm()**

```
> mod3 <- lm(libido~dose, data=viagra)
> summary(mod3)

Call:
lm(formula = libido ~ dose, data = viagra)

Residuals:
   Min     1Q Median     3Q    Max 
  -2.0   -1.2   -0.2    0.9    2.0 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)   
(Intercept)     2.2000     0.6272   3.508  0.00432 **
doseLow Dose    1.0000     0.8869   1.127  0.28158   
doseHigh Dose   2.8000     0.8869   3.157  0.00827 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.402 on 12 degrees of freedom
Multiple R-squared:  0.4604,	Adjusted R-squared:  0.3704 
F-statistic: 5.119 on 2 and 12 DF,  p-value: 0.02469
```

This is the output using lm()

Notice how the F-statistic, degrees of freedom and p-value of the overall model are identical to that using aov()

aov() is just lm() in disguise !

```
> mod3 <- lm(libido~dose, data=viagra)
> summary(mod3)

Call:
lm(formula = libido ~ dose, data = viagra)

Residuals:
   Min     1Q Median     3Q    Max 
  -2.0   -1.2   -0.2    0.9    2.0 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)   
(Intercept)     2.2000     0.6272   3.508  0.00432 **
doseLow Dose    1.0000     0.8869   1.127  0.28158   
doseHigh Dose   2.8000     0.8869   3.157  0.00827 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.402 on 12 degrees of freedom
Multiple R-squared:  0.4604,	Adjusted R-squared:  0.3704 
F-statistic: 5.119 on 2 and 12 DF,  p-value: 0.02469
```

```
> anova(mod3)
Analysis of Variance Table

Response: libido
          Df Sum Sq Mean Sq F value  Pr(>F)
dose       2 20.133 10.0667  5.1186 0.02469 *
Residuals 12 23.600  1.9667                  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
>
```

Wrap the lm model in anova() to get the typical ANOVA summary

Planned Contrasts -

Use contrasts() to identify the current groupings of a variable. This dictates what comparisons the model makes. If you wish to set your own, there are some important rules.

```
> contrasts(viagra$dose)
          Low Dose High Dose
Placebo           0         0
Low Dose          1         0
High Dose         0         1
> contrast1 <- c(-2,1,1)
> contrast2 <- c(0,-1,1)
> contrasts(viagra$dose) <- cbind(contrast1,contrast2)
> contrasts(viagra$dose)
          contrast1 contrast2
Placebo          -2         0
Low Dose          1        -1
High Dose         1         1
>
```

1. The sum of each contrast must equal 0
2. Assign weights negatively and positively to differentiate groups that you wish to compare.
3. Use a weight of 0 to ignore/exclude a group from comparisons.

Here, contrast1 is comparing placebo (-2) to low (+1) and high (+1) dose combined.

Contrast2 is comparing the group low dose (-1) to the group high dose (+1) whilst ignoring placebo (0).

Planned Contrasts

```
> mod4 <- lm(libido~dose, data=viagra)
> summary(mod4)

Call:
lm(formula = libido ~ dose, data = viagra)

Residuals:
   Min     1Q Median     3Q    Max 
  -2.0   -1.2   -0.2    0.9    2.0 

Coefficients:
               Estimate Std. Error t value Pr(>|t|)    
(Intercept)      3.4667     0.3621   9.574 5.72e-07 ***
dosecontrast1    0.6333     0.2560   2.474   0.0293 *  
dosecontrast2    0.9000     0.4435   2.029   0.0652 .  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.402 on 12 degrees of freedom
Multiple R-squared:  0.4604,	Adjusted R-squared:  0.3704 
F-statistic: 5.119 on 2 and 12 DF,  p-value: 0.02469

> #important note - these p-values are 2-tailed.
> #if we have planned comparisons, we can divide these by 2 to get 1-tailed p-values
>
```

Notice how the output now has our planned contrasts for the individual coefficients.

Note that these p-values are 2-tailed so can be halved to 1-tailed because of our planned comparisons.

The other overall model statistics are still the same.

Planned Contrasts

```
> # linear or quadratic relationship ?
> contrasts(viagra$dose) <- contr.poly(3) #there are 3 groups
> contrasts(viagra$dose)
                .L         .Q
Placebo   -7.071068e-01  0.4082483
Low Dose  -7.850462e-17 -0.8164966
High Dose  7.071068e-01  0.4082483
>
```

```
> mod5 <- lm(libido~dose, data=viagra)
> summary(mod5)

Call:
lm(formula = libido ~ dose, data = viagra)

Residuals:
   Min     1Q Median     3Q    Max 
  -2.0   -1.2   -0.2    0.9    2.0 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)   3.4667     0.3621   9.574 5.72e-07 ***
dose.L        1.9799     0.6272   3.157  0.00827 ** 
dose.Q        0.3266     0.6272   0.521  0.61201    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.402 on 12 degrees of freedom
Multiple R-squared:  0.4604,	Adjusted R-squared:  0.3704 
F-statistic: 5.119 on 2 and 12 DF,  p-value: 0.02469

> # this indicates that there is a linear relationship, but no quadratic relationship
>
```

R has some other built in planned contrast methods.

One useful one is **contr.poly()**

This can be used to determine if there is a linear relationship or quadratic (curvilinear).

Posthoc Testing

If we did not have planned comparisons, we could do posthoc testing.

```
> pairwise.t.test(viagra$libido, viagra$dose, p.adjust.method = "bonferroni")

        Pairwise comparisons using t tests with pooled SD

data:  viagra$libido and viagra$dose

          Placebo Low Dose
Low Dose  0.845   -
High Dose 0.025   0.196

P value adjustment method: bonferroni
>
```

As we saw earlier, we could perform pairwise comparisons adjusting our p-values using **pairwise.t.test()**

Posthoc Testing

Use **TukeyHSD()** to perform the Tukey's Honestly Significantly Different posthoc tests

```
> TukeyHSD(mod1)
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = libido ~ dose, data = viagra)

$`dose`
                   diff        lwr      upr     p adj
Low Dose-Placebo    1.0 -1.3662412 3.366241 0.5162761
High Dose-Placebo   2.8  0.4337588 5.166241 0.0209244
High Dose-Low Dose  1.8 -0.5662412 4.166241 0.1474576

>
```

Posthoc Testing

If you require alternative posthoc testing methods (there are many) then you could use the library 'multcomp'.

The function **glht()** performs several other posthoc tests including Dunnet's posthoc test.

```
> library(multcomp) #need R version > 3.5
> posthocs <- glht(mod1, linfct=mcp(dose="Dunnett"), base=1)
> summary(posthocs)

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Dunnett Contrasts


Fit: aov(formula = libido ~ dose, data = viagra)

Linear Hypotheses:
                        Estimate Std. Error t value Pr(>|t|)
Low Dose - Placebo == 0    1.0000     0.8869   1.127   0.4459
High Dose - Placebo == 0   2.8000     0.8869   3.157   0.0152 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)

> confint(posthocs)

         Simultaneous Confidence Intervals

Multiple Comparisons of Means: Dunnett Contrasts


Fit: aov(formula = libido ~ dose, data = viagra)

Quantile = 2.5026
95% family-wise confidence level


Linear Hypotheses:
                        Estimate lwr     upr
Low Dose - Placebo == 0  1.0000  -1.2196  3.2196
High Dose - Placebo == 0 2.8000   0.5804  5.0196

>
```

Effect Sizes and Model Statistics

```
> ##effect size
> library(broom)
> glance(mod1)  #r2 sometimes called eta-squared in one-way ANOVA
  r.squared adj.r.squared    sigma statistic    p.value df    logLik     AIC     BIC deviance df.residual
1 0.4603659     0.3704268 1.402379  5.118644 0.02469429  3 -24.68305 57.3661 60.1983     23.6          12
> #effect size is the sqrt(R2)
>
> tidy(mod1)
       term df    sumsq    meansq statistic    p.value
1      dose  2 20.13333 10.066667  5.118644 0.02469429
2 Residuals 12 23.60000  1.966667        NA         NA
>
```

The package **broom** has several useful helper functions to get model summaries including effect sizes

glance() will give us effect sizes (r.squared) as well as sigma, AIC, BIC, deviance. These can all be used to assess model fit. We discuss this more later.

tidy() gives us information about the ANOVA model in a tidy dataframe format.

Effect Sizes and Model Statistics

```
> library(sjstats)
> anova_stats(mod1)
# A tibble: 2 x 12
  term         df sumsq meansq statistic p.value etasq partial.etasq omegasq partial.omegasq cohens.f power
  <chr>     <dbl> <dbl>  <dbl>     <dbl>   <dbl> <dbl>         <dbl>   <dbl>           <dbl>    <dbl> <dbl>
1 dose          2  20.1  10.1       5.12   0.025  0.46          0.46   0.354           0.354    0.924  0.81
2 Residuals    12  23.6   1.97     NA     NA     NA            NA     NA              NA       NA     NA
>
```

Other packages also exist to give us some of these same model effect sizes and more.

Coefficient Effect Sizes

$$r = \sqrt{\frac{t^2}{t^2 + df}}$$

We can calculate the effect size of individual estimates using the same formula as for the t-test

```
> summary.lm(mod4)

Call:
lm(formula = libido ~ dose, data = viagra)

Residuals:
   Min     1Q Median     3Q    Max
  -2.0   -1.2   -0.2    0.9    2.0

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)     3.4667     0.3621   9.574 5.72e-07 ***
dosecontrast1   0.6333     0.2560   2.474   0.0293 *
dosecontrast2   0.9000     0.4435   2.029   0.0652 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.402 on 12 degrees of freedom
Multiple R-squared:  0.4604,    Adjusted R-squared:  0.3704
F-statistic: 5.119 on 2 and 12 DF,  p-value: 0.02469

> rcontrast <- function(t,DF){sqrt(t^2/(t^2 + DF))}
> rcontrast(2.474, 12) #0.48
[1] 0.5811825
>
```

Here we make a function **rcontrast()** that we can use to get the effect size for contrast1 placebo vs low+high dose

What to do if assumptions are not met ? - 1. you could try non-parametric alternatives

```
> kruskal.test(libido ~ dose, data=viagra)

        Kruskal-Wallis rank sum test

data:  libido by dose
Kruskal-Wallis chi-squared = 6.2, df = 2, p-value = 0.04505

>
> library(PMCMRplus)
> PMCMRplus::kwAllPairsNemenyiTest(libido ~ dose, data = viagra, p.adjust.method="bonferroni")

        Pairwise comparisons using Tukey-Kramer-Nemenyi all-pairs test with Tukey-Dist approximation

data: libido by dose

          Placebo Low Dose
Low Dose  0.606   -
High Dose 0.039   0.298

P value adjustment method: single-step
alternative hypothesis: two.sided
Warning message:
In kwAllPairsNemenyiTest.default(c(3, 2, 1, 1, 4, 5, 2, 4, 2, 3,  :
  Ties are present, p-values are not corrected.
>
```

What to do if assumptions are not met ? - 2. might be better to try e.g. random sampling methods

```
> library(coin)
> oneway_test(libido ~ dose, data = viagra, distribution=approximate(B=10000))

        Approximative K-Sample Fisher-Pitman Permutation Test

data:  libido by dose (Placebo, Low Dose, High Dose)
chi-squared = 6.4451, p-value = 0.0308

>
```

Analysis of Covariance

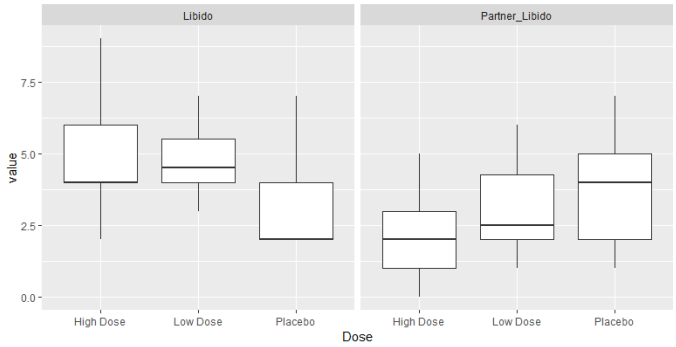
Example Dataset

```
> viagrac <- read.csv("viagrac.csv")
> head(viagrac)
     Dose Libido Partner_Libido
1 Placebo      3              4
2 Placebo      2              1
3 Placebo      5              5
4 Placebo      2              1
5 Placebo      2              2
6 Placebo      2              2
>
```

We may wish to perform an analysis of Covariance when we are interested in the effect of some predictor on a response variable but we wish to control for another potentially confounding variable.

```
> mod0 <- aov(Libido ~ Dose, data = viagrac)
> summary(mod0)  #not significant
            Df Sum Sq Mean Sq F value Pr(>F)
Dose         2  16.84   8.422   2.416  0.108
Residuals   27  94.12   3.486
>
```

For instance here Dose does not predict libido, but we have not accounted for the potentially confounding variable Partner_Libido




```
> modx <- aov(Partner_Libido ~ Dose, data = viagrac)
> summary(modx)
            Df Sum Sq Mean Sq F value Pr(>F)
Dose         2  12.77   6.385   1.979  0.158
Residuals   27  87.10   3.226
> #p>.05, yes - we are ok to proceed
>
```

An important assumption of the ANCOVA (in addition to other assumptions that we have previously discussed) is that the predictor variable (dose) and the covariate (partner_libido) are independent.

```
> # add in the covariate
> mod1 <- aov(Libido ~ Dose + Partner_Libido, data = viagrac)
> mod2 <- aov(Libido ~ Partner_Libido + Dose, data = viagrac)
>
> mod1
Call:
   aov(formula = Libido ~ Dose + Partner_Libido, data = viagrac)

Terms:
                    Dose Partner_Libido Residuals
Sum of Squares  16.84380       15.07575  79.04712
Deg. of Freedom        2              1        26

Residual standard error: 1.743638
Estimated effects may be unbalanced
> mod2
Call:
   aov(formula = Libido ~ Partner_Libido + Dose, data = viagrac)

Terms:
                Partner_Libido     Dose Residuals
Sum of Squares         6.73436 25.18519  79.04712
Deg. of Freedom              1        2        26

Residual standard error: 1.743638
Estimated effects may be unbalanced
>
> summary(mod1)
               Df Sum Sq Mean Sq F value Pr(>F)
Dose            2  16.84   8.422   2.770 0.0812 .
Partner_Libido  1  15.08  15.076   4.959 0.0348 *
Residuals      26  79.05   3.040
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> summary(mod2)
               Df Sum Sq Mean Sq F value Pr(>F)
Partner_Libido  1   6.73   6.734   2.215 0.1487
Dose            2  25.19  12.593   4.142 0.0274 *
Residuals      26  79.05   3.040
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

R uses type I sequential sums of squares - any predictor entered into a model is evaluated only after prior predictors.

There are several types of ANOVA method:

- Type 1: Sequential method.
- Type 2: All predictors are evaluated after taking into account all others except interactions.
- Type 3: All predictors are evaluated after taking into account all others.
- Type 4: (similar to type 3 but accounts for missing data)

Recommendation:

- Type 1: Unless predictors are completely independent (unlikely) not very useful.
- Type 2: If interested in MAIN EFFECTS only.
- **Type 3: Default for most stats packages - better with uneven sample sizes and interactions.**


```
> # add in the covariate
> mod1 <- aov(Libido ~ Dose + Partner_Libido, data = viagrac)
> mod2 <- aov(Libido ~ Partner_Libido + Dose, data = viagrac)
>
> mod1
Call:
   aov(formula = Libido ~ Dose + Partner_Libido, data = viagrac)

Terms:
                    Dose Partner_Libido Residuals
Sum of Squares  16.84380       15.07575  79.04712
Deg. of Freedom        2              1        26

Residual standard error: 1.743638
Estimated effects may be unbalanced
> mod2
Call:
   aov(formula = Libido ~ Partner_Libido + Dose, data = viagrac)

Terms:
                Partner_Libido     Dose Residuals
Sum of Squares         6.73436 25.18519  79.04712
Deg. of Freedom              1        2        26

Residual standard error: 1.743638
Estimated effects may be unbalanced
>
> summary(mod1)
               Df Sum Sq Mean Sq F value Pr(>F)
Dose            2  16.84   8.422   2.770 0.0812 .
Partner_Libido  1  15.08  15.076   4.959 0.0348 *
Residuals      26  79.05   3.040
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> summary(mod2)
               Df Sum Sq Mean Sq F value Pr(>F)
Partner_Libido  1   6.73   6.734   2.215 0.1487
Dose            2  25.19  12.593   4.142 0.0274 *
Residuals      26  79.05   3.040
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```


Adding in the covariate to the model appears to be straightforward but note something very important

The ORDER in which the covariate is added makes a big difference to the model - why ?

R uses type I sequential sums of squares - any predictor entered into a model is evaluated only after prior predictors.

There are several types of ANOVA method:

- Type 1: Sequential method.
- Type 2: All predictors are evaluated after taking into account all others except interactions.
- Type 3: All predictors are evaluated after taking into account all others.
- Type 4: (similar to type 3 but accounts for missing data)

Type 1: Unless predictors are completely independent (unlikely) not very useful.
Type 2: If interested in MAIN EFFECTS only.
Type 3: Default for most stats packages - better with uneven sample sizes and interactions.

Unless you specifically want to calculate your Sums of Squares using an alternative method, the best solution is to convert model results to Type III results using the **Anova()** function from the 'car' package.

```
> ## Use Type III Sums of Squares method:
> car::Anova(mod1, type = "III")
Anova Table (Type III tests)

Response: Libido
                Sum Sq Df F value   Pr(>F)
(Intercept)    131.159  1 43.1406 5.78e-07 ***
Dose            25.185  2  4.1419  0.02745 *
Partner_Libido  15.076  1  4.9587  0.03483 *
Residuals       79.047 26
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> car::Anova(mod2, type = "III")
Anova Table (Type III tests)

Response: Libido
                Sum Sq Df F value   Pr(>F)
(Intercept)    131.159  1 43.1406 5.78e-07 ***
Partner_Libido  15.076  1  4.9587  0.03483 *
Dose            25.185  2  4.1419  0.02745 *
Residuals       79.047 26
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
>
```

Notice how both of these models now give the same results

Planned Contrasts

We can perform planned contrasts as before

```
> # set our own planned comparisons
> contrast1 <- c(1,1,-2)
> contrast2 <- c(1,-1,0)
> contrasts(viagrac$Dose) <- cbind(contrast1,contrast2)
>
> contrasts(viagrac$Dose) #check
          contrast1 contrast2
High Dose         1         1
Low Dose          1        -1
Placebo          -2         0
>
> # Run the model
> mod3 <- aov(Libido ~ Partner_Libido + Dose, data = viagrac)
> car::Anova(mod3, type="III")
Anova Table (Type III tests)

Response: Libido
               Sum Sq Df F value    Pr(>F)
(Intercept)    76.069  1 25.0205 3.342e-05 ***
Partner_Libido 15.076  1  4.9587   0.03483 *
Dose           25.185  2  4.1419   0.02745 *
Residuals      79.047 26
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
>
```

Adjusted Means

```
> library(effects)
> adjmeans <- effect("Dose", mod3, se=T)
>
> adjmeans #means

 Dose effect
Dose
High Dose  Low Dose   Placebo
 5.151251  4.712050  2.926370
> summary(adjmeans) #confidence limits

 Dose effect
Dose
High Dose  Low Dose   Placebo
 5.151251  4.712050  2.926370

 Lower 95 Percent Confidence Limits
Dose
High Dose  Low Dose   Placebo
 4.118076  3.435984  1.700854

 Upper 95 Percent Confidence Limits
Dose
High Dose  Low Dose   Placebo
 6.184427  5.988117  4.151886
> adjmeans$se #standard errors
[1] 0.5026323 0.6207971 0.5962045
>
```

Using the function effect() from the 'effects' package we can calculated the adjusted means

These are means that are adjusted for the influence of the covariate.

Identifying which Adjusted Means differ using Planned Contrasts

```
> # Planned Contrasts
> summary.lm(mod3)

Call:
aov(formula = Libido ~ Partner_Libido + Dose, data = viagrac)

Residuals:
    Min      1Q  Median      3Q     Max
-3.2622 -0.7899 -0.3230  0.8811  4.5699

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)       3.1260     0.6250   5.002 3.34e-05 ***
Partner_Libido    0.4160     0.1868   2.227  0.03483 *
Dosecontrast1     0.6684     0.2400   2.785  0.00985 **
Dosecontrast2     0.2196     0.4056   0.541  0.59284
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.744 on 26 degrees of freedom
Multiple R-squared:  0.2876,    Adjusted R-squared:  0.2055
F-statistic:   3.5 on 3 and 26 DF,  p-value: 0.02954

> # contrast 1 compares the adjusted mean of placebo (2.93) to mean of adjusted means of other two groups
> # contrast 2 comapres adjusted means of low and high dose groups
> # clearly, the covariate (partner's libido) & outcome (libido) have a positive relationship also.
>
```

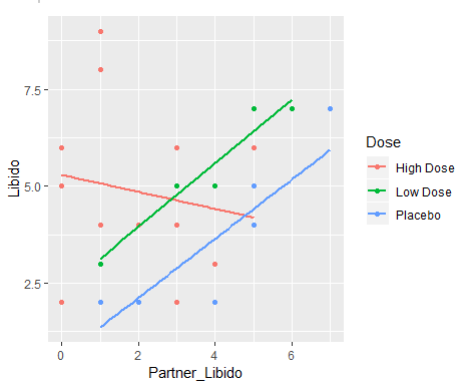
Identifying which Adjusted Means differ using Posthoc tests

```
> # Posthocs tests (has to be Tukey or Dunnet as need to compare adjusted means)
> posthocs1 <- multcomp::glht(mod3, linfct=multcomp::mcp(Dose="Tukey"))
> summary(posthocs1)

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Fit: aov(formula = Libido ~ Partner_Libido + Dose, data = viagrac)

Linear Hypotheses:
                        Estimate Std. Error t value Pr(>|t|)
Low Dose - High Dose == 0  -0.4392     0.8112  -0.541   0.8516
Placebo - High Dose == 0   -2.2249     0.8028  -2.771   0.0265 *
Placebo - Low Dose == 0    -1.7857     0.8494  -2.102   0.1090
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)

> confint(posthocs1)

         Simultaneous Confidence Intervals

Multiple Comparisons of Means: Tukey Contrasts


Fit: aov(formula = Libido ~ Partner_Libido + Dose, data = viagrac)

Quantile = 2.4844
95% family-wise confidence level


Linear Hypotheses:
                        Estimate lwr     upr
Low Dose - High Dose == 0 -0.4392  -2.4546  1.5762
Placebo - High Dose == 0  -2.2249  -4.2194 -0.2304
Placebo - Low Dose == 0   -1.7857  -3.8958  0.3245
```

Homogeneity of Regression Slopes

- In addition to the standard diagnostic plots we should also check homogeneity of regression slopes
- This means that the relationship between the covariate and the outcome variable should be consistent across predictor groups

```
> ggplot(viagrac, aes(x = Partner_Libido, y = Libido, color=Dose)) + geom_point() +
+   stat_smooth(method="lm", se=F) # this plot seems problematic
>
```



This plot is clearly problematic – the relationship for 'high-dose' is different to the other two

Homogeneity of Regression Slopes

```
> # test assumption - include interaction term
> mod4 <- aov(Libido ~ Partner_Libido * Dose, data = viagrac)
> summary(mod4)
                    Df Sum Sq Mean Sq F value Pr(>F)
Partner_Libido       1   6.73   6.734   2.757 0.1098
Dose                 2  25.19  12.593   5.156 0.0137 *
Partner_Libido:Dose  2  20.43  10.213   4.181 0.0277 *
Residuals           24  58.62   2.443
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> # to use type III
> car::Anova(mod4, type="III")
Anova Table (Type III tests)

Response: Libido
                    Sum Sq Df F value    Pr(>F)
(Intercept)         53.542  1 21.9207 9.323e-05 ***
Partner_Libido      17.182  1  7.0346   0.01395 *
Dose                36.558  2  7.4836   0.00298 **
Partner_Libido:Dose 20.427  2  4.1815   0.02767 *
Residuals           58.621 24
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
>
```

We can identify this lack of homogeneity of regression slopes by identifying a significant interaction term in this model

Effect Sizes for the Model

```
> ## Effect Sizes for ANCOVA
> library(broom)
> glance(mod3)
  r.squared adj.r.squared    sigma statistic    p.value df    logLik      AIC      BIC deviance df.residual
1 0.2876499     0.2054557 1.743638  3.499636 0.02954491  4 -57.10086 124.2017 131.2077 79.04712          26
> tidy(mod3)
            term df     sumsq    meansq statistic    p.value
1 Partner_Libido  1  6.734357  6.734357  2.215050 0.14870037
2           Dose  2 25.185194 12.592597  4.141929 0.02744654
3      Residuals 26 79.047116  3.040274        NA         NA
> library(sjstats)
> sjstats::eta_sq(mod3, partial=T)
# A tibble: 2 x 2
  term           partial.etasq
  <chr>                  <dbl>
1 Partner_Libido        0.0785
2 Dose                  0.242
> sjstats::anova_stats(mod3)
# A tibble: 3 x 12
  term             df sumsq meansq statistic p.value  etasq partial.etasq omegasq partial.omegasq cohens.f  power
  <chr>         <dbl> <dbl>  <dbl>     <dbl>   <dbl>  <dbl>         <dbl>   <dbl>           <dbl>    <dbl>  <dbl>
1 Partner_Libi~     1  6.73   6.73      2.22   0.149  0.061         0.079   0.032           0.039    0.292  0.319
2 Dose              2 25.2   12.6       4.14   0.027  0.227         0.242   0.168           0.173    0.564  0.73
3 Residuals        26 79.0    3.04     NA     NA     NA            NA      NA              NA       NA     NA
>
```

As we saw earlier, effect size for the model can be calculated in the same fashion

Effect Sizes for the Coefficient Estimates

```
> # for individual estimates
> summary.lm(mod3)

Call:
aov(formula = Libido ~ Partner_Libido + Dose, data = viagrac)

Residuals:
    Min      1Q  Median      3Q     Max 
-3.2622 -0.7899 -0.3230  0.8811  4.5699 

Coefficients:
               Estimate Std. Error t value Pr(>|t|)    
(Intercept)      3.1260     0.6250   5.002 3.34e-05 ***
Partner_Libido   0.4160     0.1868   2.227  0.03483 *  
Dosecontrast1    0.6684     0.2400   2.785  0.00985 ** 
Dosecontrast2    0.2196     0.4056   0.541  0.59284    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.744 on 26 degrees of freedom
Multiple R-squared:  0.2876,	Adjusted R-squared:  0.2055 
F-statistic:   3.5 on 3 and 26 DF,  p-value: 0.02954

> rcontrast <- function(t,DF){sqrt(t^2/(t^2 + DF))}
> rcontrast(2.785, 26) #0.48
[1] 0.4793451
>
```

These can also be calculated in the same manner as we saw earlier by converting the t-statistics to an r-statistic.

Two-Way ANOVA

```
> df <- read.csv("drink.csv")
> head(df)
  gender alcohol extraversion
1 Female    None           35
2 Female    None           30
3 Female    None           40
4 Female    None           40
5 Female    None           40
6 Female    None           45
> dim(df)
[1] 48  3
>
> table(df$alcohol)

2 Pints 4 Pints    None 
     16      16      16 
> table(df$gender)

Female   Male 
    24     24 
>
```

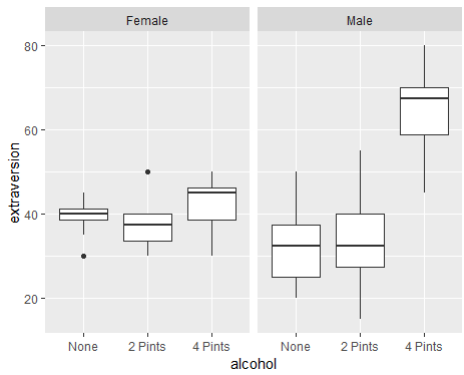
Factorial ANOVAs are when there are more than one predictors (independent variables) in the model.

This example dataset looks at the extraversion of males and females after no alcohol, 2 or 4 pints of alcohol.

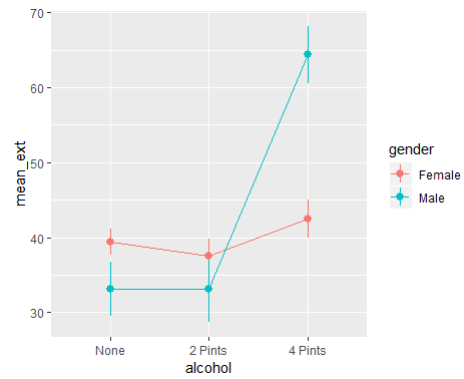
The predictors are gender and alcohol group. The response or outcome variable is extraversion.

Visualization of Data

Visualization of the data suggests that there is an effect of alcohol on extraversion, but only for males and only for the 4 pints group




```
> #boxplot
> ggplot(df, aes(alcohol, extraversion)) + geom_boxplot() + facet_wrap(~gender)
>
```




```
> #line plot of means
> df %>% group_by(gender, alcohol) %>%
+   summarize(mean_ext = mean(extraversion), sd_ext = sd(extraversion),
+             n_ext = n(), serr_ext = sd_ext/sqrt(n_ext)) %>%
+   ggplot(., aes(x= alcohol, y=mean_ext, color=gender, group=gender))  +
+   geom_point() + geom_line() +
+   geom_pointrange(aes(ymin = mean_ext - serr_ext, ymax = mean_ext + serr_ext))
>
```

As with other ANOVA we have to check assumptions

```
> car::leveneTest(df$extraversion, interaction(df$gender, df$alcohol)) # ok, assumption met.
Levene's Test for Homogeneity of Variance (center = median)
      Df F value Pr(>F)
group  5  1.4252 0.2351
      42
>
```

For the Levene Test we should examine homogeneity of variances across all potential interactions.

We do this by adding in the term ‘interaction’ into the function.

Other diagnostic tests should be performed as described earlier (also see tutorial scripts).

Setting planned contrasts - following same rules as outlined earlier

```
> contrasts(df$alcohol)
        2 Pints 4 Pints
None          0       0
2 Pints       1       0
4 Pints       0       1
> contrasts(df$alcohol) <- cbind(c(-2,1,1), c(0,-1,1))
> contrasts(df$alcohol)
        [,1] [,2]
None      -2    0
2 Pints    1   -1
4 Pints    1    1
>
```

An interaction term can be added in to the **aov()** function using the * notation

```
> mod1 <- aov(extraversion ~ gender * alcohol, data = df)
> summary(mod1)
               Df Sum Sq Mean Sq F value   Pr(>F)
gender          1    169   168.8   2.032    0.161
alcohol         2   3332  1666.1  20.065 7.65e-07 ***
gender:alcohol  2   1978   989.1  11.911 7.99e-05 ***
Residuals      42   3488    83.0
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
>
> # Type III
> car::Anova(mod1, type="III")
Anova Table (Type III tests)

Response: extraversion
               Sum Sq Df   F value    Pr(>F)
(Intercept)     83333  1 1003.5842 < 2.2e-16 ***
gender            169  1    2.0323    0.1614
alcohol          3332  2   20.0654 7.649e-07 ***
gender:alcohol   1978  2   11.9113 7.987e-05 ***
Residuals        3488 42
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
>
> # Looking at the line plot and these results, clear that for females alcohol has no
> # real effect on extraversion of who they talk with, but for males
> # there is a large effect with the highest alcohol value.
>
```

We can evaluate model statistics and effect sizes using the same methods as before (see earlier)

```
> # Other summary methods
> library(broom)
> glance(mod1)
  r.squared adj.r.squared    sigma statistic      p.value df    logLik      AIC     BIC deviance df.residual
1 0.6110595      0.564757 9.112393  13.19713 9.608579e-08  6 -170.9668 355.9336 369.032   3487.5          42
> tidy(mod1)
            term df    sumsq     meansq statistic      p.value
1         gender  1  168.750  168.75000  2.032258 1.613818e-01
2        alcohol  2 3332.292 1666.14583 20.065412 7.649011e-07
3 gender:alcohol  2 1978.125  989.06250 11.911290 7.986603e-05
4      Residuals 42 3487.500   83.03571        NA           NA
> tidy(car::Anova(mod1, type="III"))
            term     sumsq df   statistic      p.value
1    (Intercept) 83333.333  1 1003.584229 5.996759e-31
2         gender   168.750  1    2.032258 1.613818e-01
3        alcohol  3332.292  2   20.065412 7.649011e-07
4 gender:alcohol  1978.125  2   11.911290 7.986603e-05
5      Residuals  3487.500 42          NA           NA
>
```

The **anova_stats()** function from the 'sjstats' package gives lots of useful output too:

```
> library(sjstats)
> anova_stats(mod1)
# A tibble: 4 x 12
  term             df sumsq meansq statistic p.value  etasq partial.etasq omegasq partial.omegasq cohens.f  power
  <chr>         <dbl> <dbl>  <dbl>     <dbl>   <dbl>  <dbl>         <dbl>   <dbl>           <dbl>    <dbl>  <dbl>
1 gender            1  169.  169.       2.03   0.161  0.019         0.046   0.009           0.021    0.22   0.297
2 alcohol           2 3332. 1666.      20.1    0      0.372         0.489   0.35            0.443    0.977  1
3 gender:alcoh~     2 1978.  989.      11.9    0      0.221         0.362   0.2             0.313    0.753  0.995
4 Residuals        42 3488.   83.0     NA     NA     NA            NA      NA              NA       NA     NA
>
```

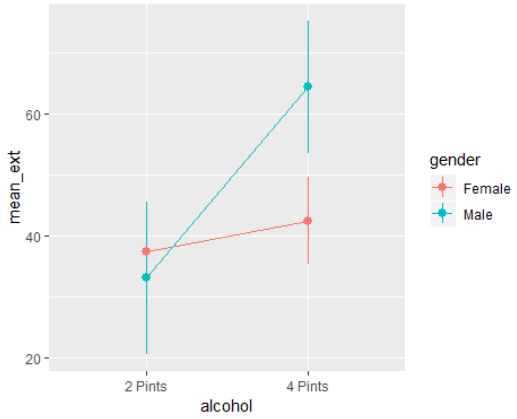
We can also evaluate planned contrasts in the same way as before (see earlier)

```
> # Summary of contrast results
> summary.lm(mod1)

Call:
aov(formula = extraversion ~ gender * alcohol, data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-19.375  -5.156   0.625   5.625  21.875 

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)    
(Intercept)        41.667      1.315  31.679  < 2e-16 ***
gender1             1.875      1.315   1.426 0.161382    
alcohol1            2.708      0.930   2.912 0.005727 ** 
alcohol2            9.062      1.611   5.626 1.37e-06 ***
gender1:alcohol1    2.500      0.930   2.688 0.010258 *  
gender1:alcohol2    6.562      1.611   4.074 0.000201 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.112 on 42 degrees of freedom
Multiple R-squared:  0.6111,	Adjusted R-squared:  0.5648 
F-statistic:  13.2 on 5 and 42 DF,  p-value: 9.609e-08

>
> # this summarizes test for differences in means between the following groups:
>
> # gender1 - 1st contrast for gender (female vs male)
> # alcohol1 - 1st contrast for alcohol (any alcohol vs no-alcohol)
> # alcohol2 - 2nd contrst for alcohol (2 pints vs 4 pints)
> # gender1:alcohol1 - is the effect of alcohol1 different by gender?
> # gender1:alcohol2 - is the effect of alcohol2 different by gender?
>
```

It is clear here that there are significant interactions between alcohol and gender – these should take priority in our evaluation over any main effects.

(It's also possible to run posthoc tests similar to those described before – see tutorial scripts)

```
#2 pints vs 4 pints
df %>% filter(alcohol != "None") %>%
  group_by(gender,alcohol) %>%
  summarize(mean_ext = mean(extraversion), sd_ext = sd(extraversion)) %>%
  ggplot(., aes(x= alcohol, y=mean_ext, color=gender, group=gender))  +
  geom_point() + geom_line() +
  geom_pointrange(aes(ymin = mean_ext - sd_ext, ymax = mean_ext + sd_ext))
```



Visualizing planned contrast interactions

Individual estimate effect sizes are calculated as before (see earlier)

```
> # for individual estimates
> summary.lm(mod1)

Call:
aov(formula = extraversion ~ gender * alcohol, data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-19.375  -5.156   0.625   5.625  21.875 

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)    
(Intercept)        41.667      1.315  31.679  < 2e-16 ***
gender1             1.875      1.315   1.426 0.161382    
alcohol1            2.708      0.930   2.912 0.005727 ** 
alcohol2            9.062      1.611   5.626 1.37e-06 ***
gender1:alcohol1    2.500      0.930   2.688 0.010258 *  
gender1:alcohol2    6.562      1.611   4.074 0.000201 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.112 on 42 degrees of freedom
Multiple R-squared:  0.6111,	Adjusted R-squared:  0.5648 
F-statistic:  13.2 on 5 and 42 DF,  p-value: 9.609e-08

> rcontrast <- function(t,DF){sqrt(t^2/(t^2 + DF))}
> rcontrast(4.074, 42) #0.53
[1] 0.5322082
> 
```

3d. Repeated-Measures ANOVA

For a repeated-measures ANOVA (also called within-subjects ANOVA), we have several observations on the same subjects.

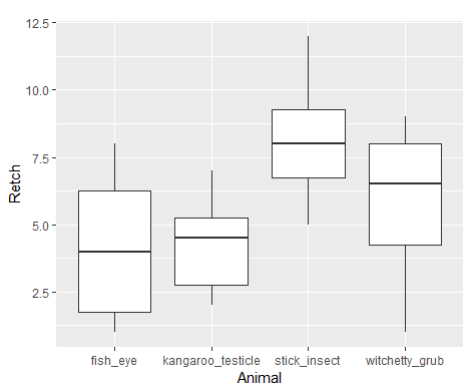
Conceptually data for repeated-measures ANOVA often are in wide format.

```
> ### Import Data
> df <-read.csv("tucker.csv")
> head(df)
  participant stick_insect kangaroo_testicle fish_eye witchetty_grub
1          P1            8                 7        1              6
2          P2            9                 5        2              5
3          P3            6                 2        3              8
4          P4            5                 3        1              9
5          P5            8                 4        5              8
6          P6            7                 5        6              7
>
> df %>% gather(Animal, Retch, 2:5) -> df
> df$Animal <- factor(df$Animal)
> head(df)
  participant       Animal Retch
1          P1 stick_insect     8
2          P2 stick_insect     9
3          P3 stick_insect     6
4          P4 stick_insect     5
5          P5 stick_insect     8
6          P6 stick_insect     7
>
```

In R, we like to have 'tidy' data, so convert to long format using tidyr.

These data show the length of time (in seconds) it took to retch for different participants eating four different foods.

```
> ggplot(df, aes(Animal, Retch)) + geom_boxplot()
```



We should visualize our data, as well as examine our data for normality, skewnewss, kurtosis

```
> mod1 <- aov(Retch ~ Animal + Error(participant/Animal), data=df)
> summary(mod1)

Error: participant
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals  7  17.38   2.482

Error: participant:Animal
          Df Sum Sq Mean Sq F value Pr(>F)
Animal     3  83.13  27.708   3.794 0.0256 *
Residuals 21 153.38   7.304
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The basic approach is to use **aov()** and include an Error term – this tells the model that the variable 'Animal' is nested within the variable 'participant' - this will ensure that aov() performs a repeated-measures ANOVA.

```
> library(ez)
> ezANOVA(data = df, dv = .(Retch), wid = .(participant), within = .(Animal),
+         detailed = T, type = 3)
$`ANOVA`
       Effect DFn DFd     SSn     SSd          F            p p<.05       ges
1 (Intercept)   1   7 990.125  17.375 398.899281 1.973536e-07     * 0.8529127
2      Animal   3  21  83.125 153.375   3.793806 2.557030e-02     * 0.3274249

$`Mauchly's Test for Sphericity`
  Effect        W          p p<.05
2 Animal 0.136248 0.04684581     *

$`Sphericity Corrections`
  Effect       GGe      p[GG] p[GG]<.05       HFe      p[HF] p[HF]<.05
2 Animal 0.5328456 0.06258412           0.6657636 0.04833061         *

>
```

Effect sizes: "generalized eta-squared"

This model can also be run using the ezANOVA() function in the 'ez' package. The advantage of this is that it also assesses sphericity

– an assumption of the model is that the variances of the *differences* between all groups are equal

Here sphericity cannot be assumed as p<.05 for the Mauchly's test – but it also provides corrected values using the Greenhouse-Geisser and Huynd-Feldt methods. A good rule of thumb is to take the average of these p-values as to whether the model is significant.

pairwise-t-test can be used for posthoc tests

```
> pairwise.t.test(df$Retch, df$Animal, p.adj="bonferroni", paired=T)

        Pairwise comparisons using paired t tests

data:  df$Retch and df$Animal

                  fish_eye kangaroo_testicle stick_insect
kangaroo_testicle 1.0000   -                 -
stick_insect      0.0056   0.0121            -
witchetty_grub    1.0000   1.0000            1.0000

P value adjustment method: bonferroni
>
```

Multilevel Approach to Repeated-Measures ANOVA

Alternatively, we an use a multilevel approach – this means that we can ignore sphericity.

As before – we can assign planned contrasts using the same rules:

```
> PartvsWhole <- c(-1,-1,1,1)
> TesticlevsEye <- c(1,-1,0,0)
> StickvsGrub <- c(0,0,-1,1)
>
> # positive compared against negatives and zeros are ignored
>
> contrasts(df$Animal) <- cbind(PartvsWhole, TesticlevsEye, StickvsGrub)
> contrasts(df$Animal)
                  PartvsWhole TesticlevsEye StickvsGrub
fish_eye                   -1             1           0
kangaroo_testicle          -1            -1           0
stick_insect                1             0          -1
witchetty_grub              1             0           1
>
```

```
> # run the model
> library(nlme)
> mod2 <- nlme::lme(Retch ~ Animal, random = ~1|participant/Animal, data = df, method = "ML")
> summary(mod2)
Linear mixed-effects model fit by maximum likelihood
 Data: df
       AIC      BIC    logLik
  158.3949 168.6551 -72.19747

Random effects:
 Formula: ~1 | participant
         (Intercept)
StdDev: 7.756885e-05

 Formula: ~1 | Animal %in% participant
        (Intercept)  Residual
StdDev:    2.309935 0.0117607

Fixed effects: Retch ~ Animal
                     Value Std.Error DF   t-value p-value
(Intercept)         5.5625 0.4365423 21 12.742178  0.0000
AnimalPartvsWhole   1.3750 0.4365423 21  3.149752  0.0048
AnimalTesticlevsEye -0.0625 0.6173641 21 -0.101237  0.9203
AnimalStickvsGrub  -1.1875 0.6173641 21 -1.923500  0.0681
 Correlation:
                   (Intr) AnmlPW AnmlTE
AnimalPartvsWhole  0
AnimalTesticlevsEye 0      0
AnimalStickvsGrub  0      0      0

Standardized Within-Group Residuals:
         Min           Q1           Med           Q3          Max
-0.0104692609 -0.0046836167  0.0001377534  0.0041326030  0.0085407128

Number of Observations: 32
Number of Groups:
           participant Animal %in% participant
                     8                     32
```

We use **lme()** from the package 'nlme' to run this approach.

Importantly, we are setting a random effect parameter in the model using:
random = ~1|participant/animal

This in effect tells the model that individuals may vary in their propensity to retch and that scores across animals are connected by participant

The summary output gives us a lot of information. Of note, are the fixed effects – these are our planned contrasts. These can be interpreted similarly to those we've seen in other models.

We can also use this information to calculate estimate effect sizes as before

```
> baseline <- lme(Retch ~ 1, random = ~1|participant/Animal, data = df, method = "ML")
>
> anova(baseline,mod2)
         Model df      AIC      BIC    logLik   Test  L.Ratio p-value
baseline     1  4 165.0875 170.9504 -78.54373
mod2         2  7 158.3949 168.6551 -72.19747 1 vs 2 12.69253  0.0054
>
```

To investigate whether there is a main effect of "animal"

– i.e. does including the group means of 'animal' in the model make a better fit of the data?

Here we create a baseline model by excluding the 'animal' predictor – in its place we put '1' indicating keeping the intercept.

We use **anova()** to compare the models directly. Of note, we see that the AIC and BIC are lower for the model including 'animal'. (two measures of the fit with smaller values being a better fit – see regression section).

The likelihood ratio test is also significantly different between the two models, so including 'animal' does improve fit – indicating that 'animal' has a significant effect on the outcome variable Retch.

Posthoc tests for repeated-measures ANOVA

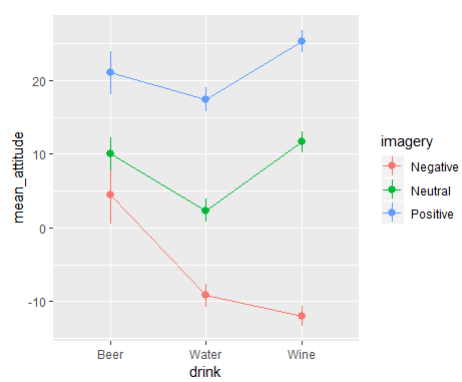
```
> library(multcomp)
> summary(multcomp::glht(mod2, linfct=multcomp::mcp(Animal="Tukey")))

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Fit: lme.formula(fixed = Retch ~ Animal, data = df, random = ~1 |
    participant/Animal, method = "ML")

Linear Hypotheses:
                                      Estimate Std. Error z value Pr(>|z|)
kangaroo_testicle - fish_eye == 0        0.125      1.155   0.108  0.99955
stick_insect - fish_eye == 0             4.000      1.155   3.463  0.00291 **
witchetty_grub - fish_eye == 0           1.625      1.155   1.407  0.49495
stick_insect - kangaroo_testicle == 0    3.875      1.155   3.355  0.00440 **
witchetty_grub - kangaroo_testicle == 0  1.500      1.155   1.299  0.56372
witchetty_grub - stick_insect == 0      -2.375      1.155  -2.056  0.16774
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)

>
```

We can compute Tukey post-hoc tests (among others) using the 'multcomp' package.

Repeated-Measures for Factorial Designs

```
> head(attitude)
  participant   groups attitude  imagery drink
1          P1  beerneg        6 Negative  Beer
2          P1 beerneut        5  Neutral  Beer
3          P1  beerpos        1 Positive  Beer
4          P1 waterneg      -14 Negative Water
5          P1 waterneu       -2  Neutral Water
6          P1 waterpos       10 Positive Water
>
```

The same principles we saw in the one-way ANOVA can be used in a factorial design



Using **ezANOVA()**

```
> library(ez)
> mod5 <- ezANOVA(data = attitude, dv = .(attitude), wid = .(participant), within = .(imagery, drink),
+         type=3, detailed=T)
> mod5
$`ANOVA`
         Effect DFn DFd       SSn      SSd          F            p p<.05       ges
1   (Intercept)   1  19 11218.006 1920.106 111.005411 2.255322e-09     * 0.4126762
2       imagery   2  38 21628.678 3352.878 122.564825 2.680197e-17     * 0.5753191
3         drink   2  38  2092.344 7785.878   5.105981 1.086293e-02     * 0.1158687
4 imagery:drink   4  76  2624.422 2906.689  17.154922 4.589040e-10     * 0.1411741

$`Mauchly's Test for Sphericity`
         Effect         W            p p<.05
2       imagery 0.6621013 2.445230e-02     *
3         drink 0.2672411 6.952302e-06     *
4 imagery:drink 0.5950440 4.356587e-01

$`Sphericity Corrections`
         Effect       GGe        p[GG] p[GG]<.05       HFe        p[HF] p[HF]<.05
2       imagery 0.7474407 1.757286e-13         * 0.7968420 3.142804e-14         *
3         drink 0.5771143 2.976868e-02         * 0.5907442 2.881391e-02         *
4 imagery:drink 0.7983979 1.900249e-08         * 0.9785878 6.809640e-10         *

>
> # need to correct for sphericity for main effects but not interaction term
> ## posthoc tests
>
```

Posthoc testing from **exANOVA()** model

```
> pairwise.t.test(attitude$attitude, attitude$groups, paired = T, p.adjust.method = 'bonferroni')

        Pairwise comparisons using paired t tests

data:  attitude$attitude and attitude$groups

         beerneg beerneut beerpos waterneg waterneu waterpos wineneg wineneut
beerneut 1.00000 -        -       -        -        -        -       -
beerpos  0.00217 0.01982  -       -        -        -        -       -
waterneg 0.18860 0.00010  2.9e-06 -        -        -        -       -
waterneu 1.00000 0.74838  0.00212 0.00068  -        -        -       -
waterpos 0.47584 1.00000  1.00000 4.9e-11  9.0e-07  -        -       -
wineneg  0.00265 2.0e-07  5.6e-08 1.00000  0.00041  1.3e-09  -       -
wineneut 1.00000 1.00000  0.39905 1.1e-07  8.1e-05  0.10547  2.3e-07 -
winepos  0.01105 0.00310  1.00000 3.2e-10  4.3e-10  0.07300  1.9e-10 2.2e-05

P value adjustment method: bonferroni
>
```

Multilevel Approach

```
> ## baseline model
> baseline <- nlme::lme(attitude ~ 1, random = ~1|participant/drink/imagery, data = attitude, method="ML")
>
> # update models adding one variable at a time
> drinkModel <- update(baseline, .~. + drink)
> imageryModel <- update(drinkModel, .~. + imagery)
> attitudeModel <- update(imageryModel, .~. + drink:imagery)
>
> # compare models
> anova(baseline, drinkModel, imageryModel, attitudeModel)
              Model df      AIC      BIC    logLik   Test   L.Ratio p-value
baseline          1  5 1503.590 1519.555 -746.7950
drinkModel        2  7 1498.461 1520.812 -742.2306 1 vs 2   9.12891  0.0104
imageryModel      3  9 1350.529 1379.265 -666.2644 2 vs 3 151.93237  <.0001
attitudeModel     4 13 1316.512 1358.020 -645.2560 3 vs 4  42.01676  <.0001
>
```

Here we build up the model one step at a time using **update()** – this allows us to compare models to each other to find the best fit.

In the random effect, we have to specify that both drink & imagery are nested within participant.

Planned contrasts can be performed in the same manner as before (see additional tutorial code for more information)

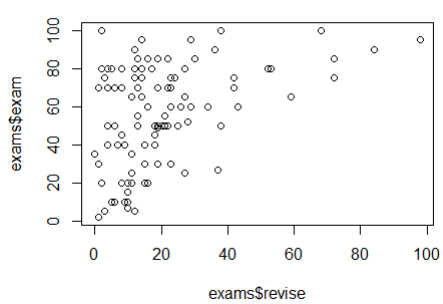
4. Correlation & Regression

4a. Pearson and Spearman Correlation

Example dataset

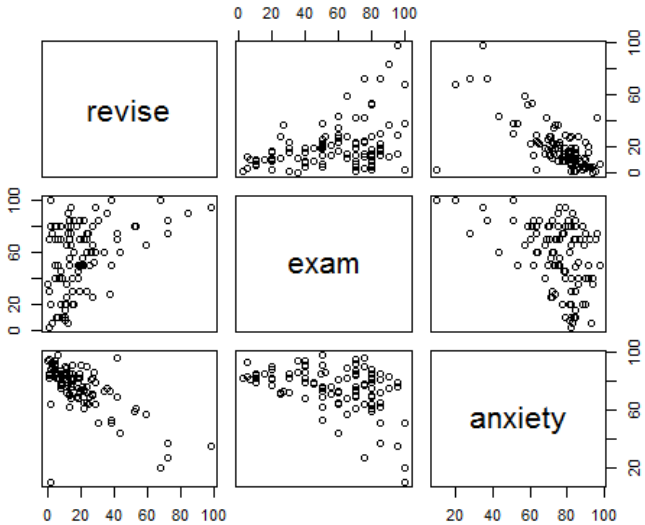
```
> dim(exams)
[1] 103   5
> head(exams)
# A tibble: 6 x 5
   code revise  exam anxiety gender
  <int>  <int> <int>   <dbl> <chr>
1     1      4    40    86.3 Male
2     2     11    65    88.7 Female
3     3     27    80    70.2 Male
4     4     53    80    61.3 Male
5     5      4    40    89.5 Male
6     6     22    70    60.5 Female
>
```

```
> plot(exams$revise, exams$exam)
```

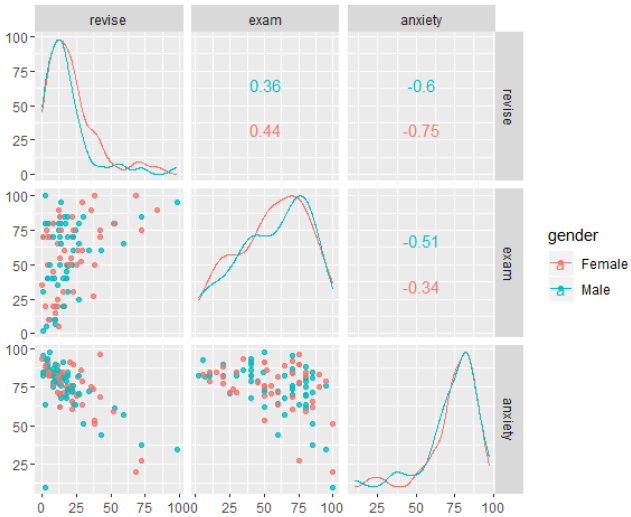


Get overview of variables of interest using base-r

```
> plot(exams[,2:4])
```

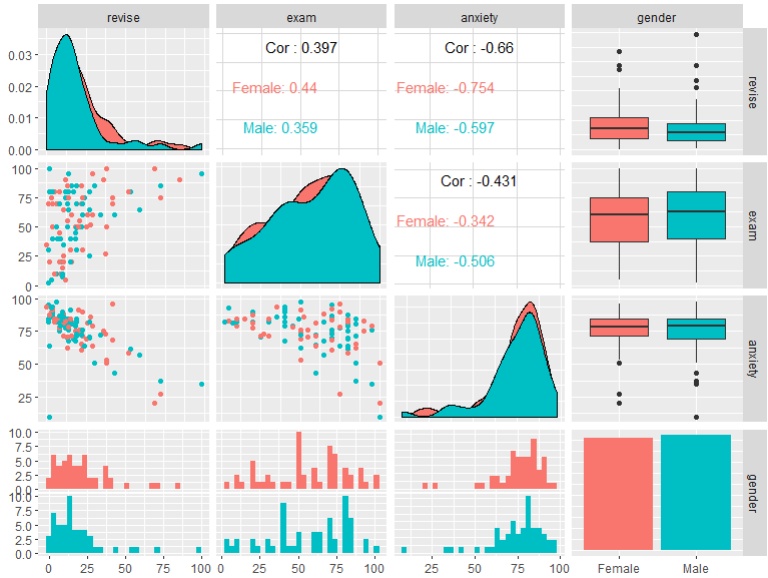



```
> GGally::ggscatmat(exams, columns = 2:4, color="gender", alpha=0.8)
```



Use the **ggscatmat()** function from the GGally package to get an overview in a ggplot2 style – this includes Pearson correlation coefficients by group

```
> ggpairs(exams, columns = 2:5, mapping = aes(color = gender), alpha=0.8)
```



The ggpairs() function from the same package allows you to compare across continuous and discrete variables

Use **cor()** to get correlation matrices or specific correlations between variables

```
> cor(exams[,2:4],use = "complete.obs")
             revise       exam    anxiety
revise    1.0000000  0.3787481 -0.6603021
exam      0.3787481  1.0000000 -0.4309643
anxiety  -0.6603021 -0.4309643  1.0000000
```

```
> cor(exams$revise, exams$anxiety, use = "complete.obs",  method = "pearson")
[1] -0.6603021
> cor(exams$revise, exams$anxiety, use = "complete.obs",  method = "spearman")
vailable
[1] -0.6107712
>
```

Pearson, Spearman and Kendall (Tau-B) methods are available.

Use **cor.test()** to run a significance test

```
> cor.test(exams$revise, exams$anxiety)  #'cor.test' doesn't mind NA data

        Pearson's product-moment correlation

data:  exams$revise and exams$anxiety
t = -8.7923, df = 100, p-value = 4.363e-14
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.7575042 -0.5344582
sample estimates:
       cor
-0.6603021

>
```

Adding in alternative = “less” etc. dictates a one-tailed test as **cor.test()** is 2-tailed by default.

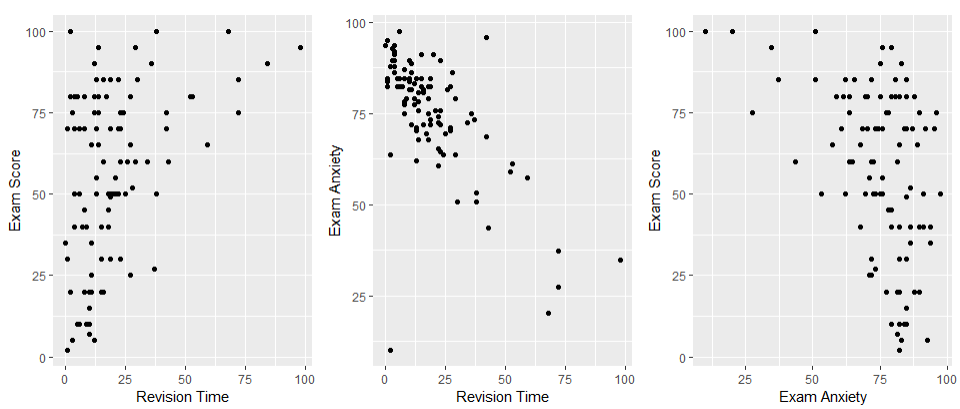
Use **rcorr()** from the ‘Hmisc’ package to get a matrix of correlation tests:

```
> examMatrix <- as.matrix(exams[, 2:4])  # converts data into a 'matrix' format
> Hmisc::rcorr(examMatrix) # use 'rcorr' from package Hmisc
        revise  exam anxiety
revise    1.00  0.40   -0.66
exam      0.40  1.00   -0.43
anxiety  -0.66 -0.43    1.00

n
        revise exam anxiety
revise     103  103     102
exam       103  103     102
anxiety    102  102     102

P
        revise exam anxiety
revise          0    0
exam     0           0
anxiety  0      0
>
```

Use **corr.test()** from the 'psych' package to get a matrix of correlation tests with adjusted p-values:

```
> psych::corr.test(examMatrix, method="spearman", adjust="holm")
Call:psych::corr.test(x = examMatrix, method = "spearman", adjust = "holm")
Correlation matrix
        revise  exam anxiety
revise    1.00  0.35   -0.61
exam      0.35  1.00   -0.39
anxiety  -0.61 -0.39    1.00
Sample Size
        revise exam anxiety
revise     103  103     102
exam       103  103     102
anxiety    102  102     102
Probability values (Entries above the diagonal are adjusted for multiple tests.)
        revise exam anxiety
revise       0    0       0
exam         0    0       0
anxiety      0    0       0

 To see confidence intervals of the correlations, print with the short=FALSE option
>
```

Partial Correlations



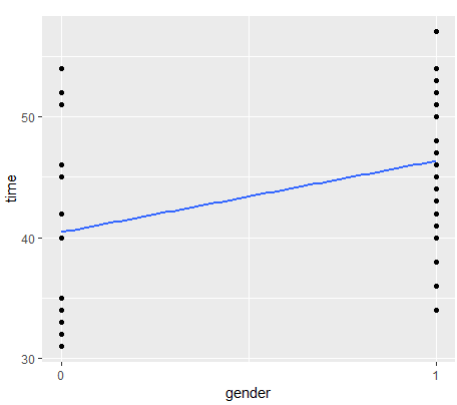
When we have 3 variables that are clearly interrelated, but we are interested in the relationship between 2 of them, we can use a partial correlation to get and adjusted correlation coefficient accounting for their relationship with a third variable.

```
> exams1 <- exams[!is.na(exams$anxiety),] # removing the one line with an NA in it.
> # get a matrix of partial correlations
> ppcor::pcor(exams1[,2:4]) #default is Pearson's
$`estimate`
            revise       exam    anxiety
revise   1.0000000  0.1389764 -0.5951982
exam     0.1389764  1.0000000 -0.2602346
anxiety -0.5951982 -0.2602346  1.0000000

$p.value
              revise        exam      anxiety
revise  0.000000e+00 0.165732624 5.232035e-11
exam    1.657326e-01 0.000000000 8.583321e-03
anxiety 5.232035e-11 0.008583321 0.000000e+00

$statistic
           revise      exam   anxiety
revise   0.000000  1.396348 -7.369715
exam     1.396348  0.000000 -2.681698
anxiety -7.369715 -2.681698  0.000000

$n
[1] 102

$gp
[1] 1

$method
[1] "pearson"

>
```

Use **pcor()** from the 'ppcor' package to get a matrix of partial correlation tests

Partial correlation tests need complete observations across variables.

Point-Biserial Correlations



Point-biserial correlations can be used to determine whether there is a correlation between a discrete dichotomous variable (e.g. gender) and a continuous variable (e.g. time).

Here we are investigating whether male cats (1) spend more hours per week (time) away from home than female cats (2)

Point-Biserial Correlations

```
> # running correlation  (x=continuous_variable, y=dichotomous_variable)
> # dichotomous variable can be numbers (0/1) or character string (male/female)
> polycor::polyserial(cats$time,cats$gender, std.err = T)

Polyserial Correlation, 2-step est. = 0.46 (0.1245)
Test of bivariate normality: Chisquare = 15.64, df = 5, p = 0.007967

> # running correlation  (x=continuous_variable, y=dichotomous_variable)
> # dichotomous variable can be numbers (0/1) or character string (male/female)
> polycor::polyserial(cats$time,cats$gender, std.err = T)

Polyserial Correlation, 2-step est. = 0.46 (0.1245)
Test of bivariate normality: Chisquare = 15.64, df = 5, p = 0.007967

>
> # Polyserial Correlation, 2-step est. = 0.46 (0.1245)
> # Test of bivariate normality: Chisquare = 15.64, df = 5, p = 0.007967
>
> ltm::biserial.cor(cats$time,cats$gender, level = 2) #[1] 0.3784542
[1] 0.3784542
>
> # using pearson (works if dichotomous variable is coded 0/1)
> cor.test(cats$time, cats$gender) #0.3784542

        Pearson's product-moment correlation

data:  cats$time and cats$gender
t = 3.1138, df = 58, p-value = 0.002868
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.137769 0.576936
sample estimates:
      cor
0.3784542

>
```

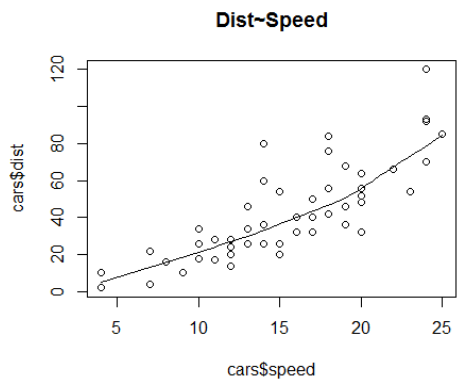
There are several options for running point-biserial correlations in R, each using slightly different algorithms.

Probably best to use the **polyserial()** function from the 'polycor' pacakge.

4b. Linear Regression

```
> # we can use built in dataset of 'cars'
> dim(cars)
[1] 50  2
> head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
```

```
> scatter.smooth(x=cars$speed, y=cars$dist, main="Dist~Speed")
>
```

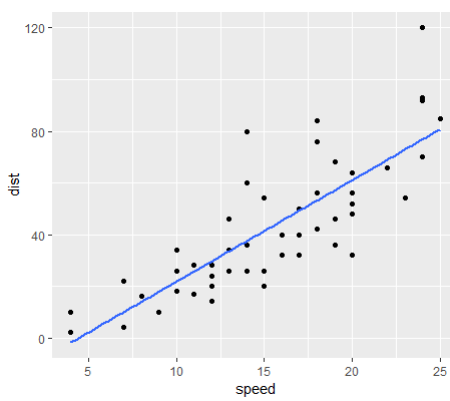



```
> ggplot(cars, aes(x = speed, y = dist)) +
+   geom_point() +
+   stat_smooth(method='lm', se=F) # remove shaded area
>
```

We can use a linear regression model to determine the linear relationship between two continuous variables.

Our model will determine a) the intercept, b) the coefficient of the slope of the trend line, and e) have some error

Y = a + bX + e



Use **lm()** to build the basic linear regression model

```
> mod1 <- lm(dist ~ speed, data=cars)  # build regression model
>
> mod1

Call:
lm(formula = dist ~ speed, data = cars)

Coefficients:
(Intercept)        speed
    -17.579        3.932

> summary(mod1)

Call:
lm(formula = dist ~ speed, data = cars)

Residuals:
    Min      1Q  Median      3Q     Max
-29.069  -9.525  -2.272   9.215  43.201

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -17.5791     6.7584  -2.601   0.0123 *
speed         3.9324     0.4155   9.464 1.49e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 15.38 on 48 degrees of freedom
Multiple R-squared:  0.6511,    Adjusted R-squared:  0.6438
F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12

>
```

R^2 and the Residual Standard Error are useful measures of goodness-of-fit of the model

```
> ## 95% confidence intervals around estimates
> confint(mod1)
                 2.5 %    97.5 %
(Intercept) -31.167850 -3.990340
speed         3.096964  4.767853
> confint(mod1, level = 0.95)
                 2.5 %    97.5 %
(Intercept) -31.167850 -3.990340
speed         3.096964  4.767853
```

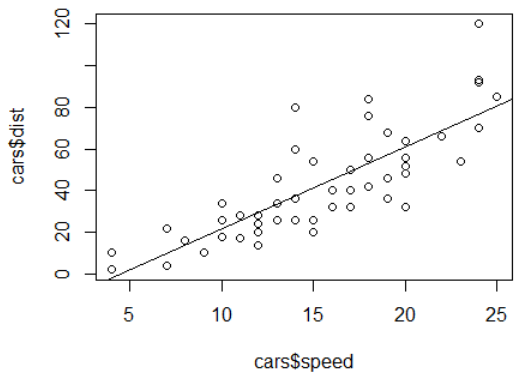
Use **anova()** to test if model fits data better than zero slope model

```
> anova(mod1) # test null hypothesis of zero slope.
Analysis of Variance Table

Response: dist
          Df Sum Sq Mean Sq F value   Pr(>F)
speed      1  21186 21185.5  89.567 1.49e-12 ***
Residuals 48  11354   236.5
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
>
> # AIC (Akaike's information criterion) and BIC (Bayesian information criterion)
> # Further measures of goodness of fit.
> # For model comparison, the model with the lowest AIC and BIC score is preferred.
>
> AIC(mod1) # 419.1569
[1] 419.1569
> BIC(mod1) # 424.8929
[1] 424.8929
>
```

Other measures of goodness-of-fit include AIC and BIC

Use **predict()** to calculate predicted values of Y based on values of X

```
> ## Make a simple regression plot:
> plot(cars$speed, cars$dist)
> abline(mod1)
>
>
>
> ## predicting values
>
> newdata <-data.frame(speed=15.5) # wrap the parameter
> predict(mod1, newdata)     # apply predict
       1
43.37324
>
> #        1
> # 43.37324
>
> predict(mod1, newdata, interval = "confidence")
       fit      lwr      upr
1 43.37324 38.99931 47.74717
>
> #        fit      lwr      upr
> # 1 43.37324 38.99931 47.74717
>
```




```
## Add predictions
library(modelr)

cars %>%
  modelr::add_predictions(mod1)
```

Use **add_predictions()** from the 'modelr' package to add predictions to a dataset

```
   speed dist      pred
1      4    2 -1.849460
2      4   10 -1.849460
3      7    4  9.947766
4      7   22  9.947766
5      8   16 13.880175
6      9   10 17.812584
7     10   18 21.744993
8     10   26 21.744993
9     10   34 21.744993
10    11   17 25.677401
11    11   28 25.677401
```

Residuals

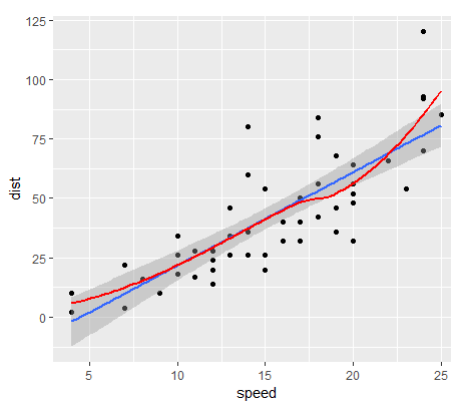
Raw residuals as well as standardized residuals and studentized residuals can be obtained from the model object.

```
mod1$residuals  # these are the residuals:  observed - predicted.

mean(mod1$residuals)  # should be 0.  [1] 8.65974e-17

resid(mod1) #residuals
rstandard(mod1) #standardized residuals (residuals / standard deviation of residuals)
rstudent(mod1) # studentized residuals  (residuals controlled for leverage)
```

Checking assumptions

Assumption: Linear relationship between X and Y

```
# Assumption:  Linear relationship between X and Y

ggplot(cars, aes(speed, dist)) +
  geom_point() +
  geom_smooth(method = "lm") +
  geom_smooth(se = FALSE, color = "red")
```



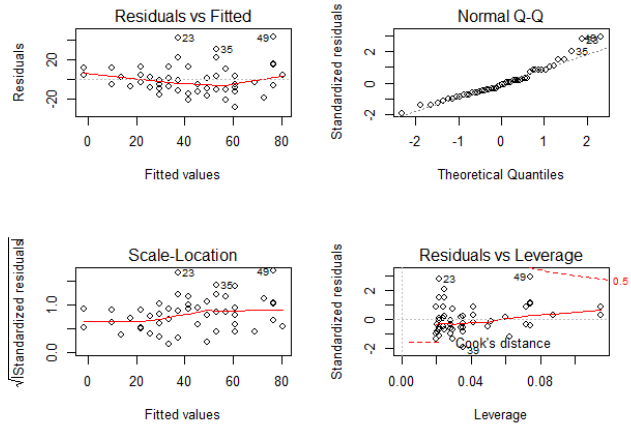
```
# Assumption: Homoscedasticity of residuals

par(mfrow=c(2,2))  # set 2 rows and 2 column plot layout
plot(mod1)
```

Assumption: Homoscedasticity of residuals

Top-left: fitted values (x) against residuals (y) - also tests for non-linearity

Bottom-left: fitted values (x) against square-root of standardized residuals (y)

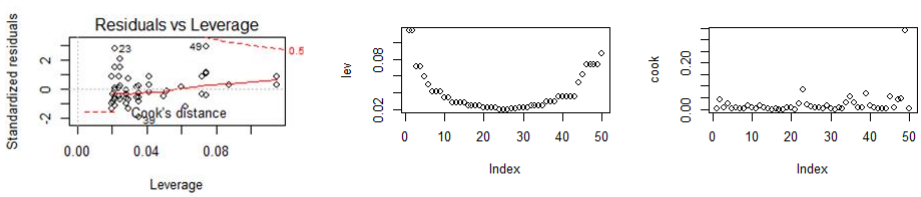
Expect an approximately flat red line (no overall pattern) for both if homoscedasticity



Assumption: Normality of residuals

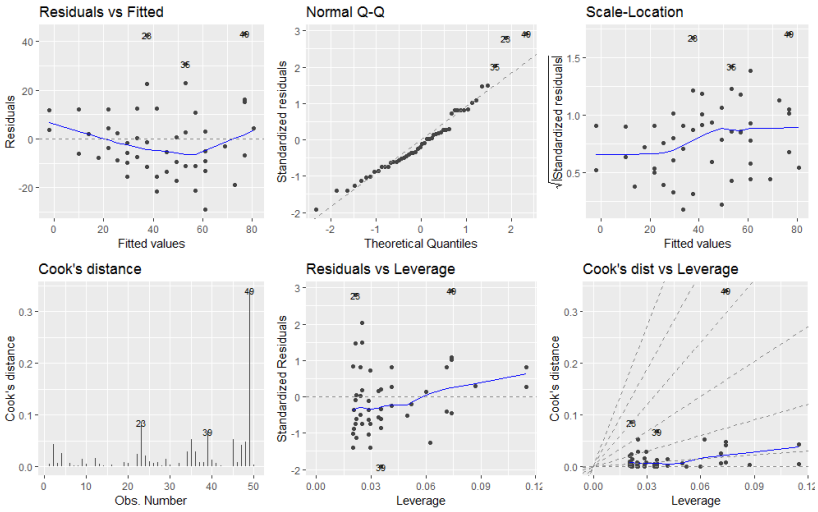
Top-right plot = Normal QQ plot
If points lie exactly on the line then they come from a perfectly normal distribution. Some deviation is expected, particularly at the tails. Can also test using **shapiro.test()**

Leverage - a measure of how much the predicted value would change if the observation is moved one unit in the y-direction




```
lev <- hat(model.matrix(mod1)) #this gets the leverage of each data point
lev
plot(lev)

# Studentized residuals are residuals standardized for their leverage
rstudent(mod1)

# An overly influential point is one with a disproportionately large leverage or outlier or both
# Cook's Distance is a measure of influence  >1 generally indicates overly high influence

cook <- cooks.distance(mod1)
cook
plot(cook)
```

```r
## ggplot style diagnostic plots
library(ggfortify)
autoplot(mod1, label.size = 3)

# can actually plot more than base-r basic diagnostic plots

autoplot(mod1, which = 1:6, ncol = 3, label.size = 3)
```



Use the 'broom' package to get model summary information including for observations

```
> broom::tidy(mod1) # coefficients in data frame
         term   estimate std.error statistic      p.value
1 (Intercept) -17.579095 6.7584402 -2.601058 1.231882e-02
2       speed   3.932409 0.4155128  9.463990 1.489836e-12
>
> broom::glance(mod1) # model level statistics in a dataframe
  r.squared adj.r.squared    sigma statistic      p.value df    logLik      AIC      BIC deviance df.residual
1 0.6510794     0.6438102 15.37959  89.56711 1.489836e-12  2 -206.5784 419.1569 424.8929 11353.52          48
>
> broom::augment(mod1) # observation level statistics in a dataframe
   dist speed   .fitted  .se.fit     .resid       .hat   .sigma      .cooksd  .std.resid
1     2     4 -1.849460 5.212326   3.849460 0.11486131 15.53088 4.592312e-03  0.26604155
2    10     4 -1.849460 5.212326  11.849460 0.11486131 15.43338 4.351399e-02  0.81893273
3     4     7  9.947766 4.112527  -5.947766 0.07150365 15.51624 6.202350e-03 -0.40134618
4    22     7  9.947766 4.112527  12.052234 0.07150365 15.43489 2.546734e-02  0.81326629
5    16     8 13.880175 3.766297   2.119825 0.05997080 15.53907 6.446705e-04  0.14216236
6    10     9 17.812584 3.435464  -7.812584 0.04989781 15.49830 7.131993e-03 -0.52115255
7    18    10 21.744993 3.124921  -3.744993 0.04128467 15.53232 1.331656e-03 -0.24869180
8    26    10 21.744993 3.124921   4.255007 0.04128467 15.52941 1.719059e-03  0.28256008
9    34    10 21.744993 3.124921  12.255007 0.04128467 15.43474 1.425993e-02  0.81381197
10   17    11 25.677401 2.841330  -8.677401 0.03413139 15.48889 5.823417e-03 -0.57409795
11   28    11 25.677401 2.841330   2.322599 0.03413139 15.53851 4.172024e-04  0.15366341
12   14    12 29.609810 2.593545 -15.609810 0.02843796 15.36972 1.551791e-02 -1.02971654
13   20    12 29.609810 2.593545  -9.609810 0.02843796 15.47714 5.881222e-03 -0.63392061
14   24    12 29.609810 2.593545  -5.609810 0.02843796 15.52015 2.004169e-03 -0.37005667
```

Validating models using Training/Testing Data

A common approach to validating models is to develop a model using a proportion (e.g. 80%) of the original data and to then test it on the remaining out-of-sample data.

By comparing the mean square errors of the out-of-sample data to the in-sample (training) data, it's possible to evaluate the goodness of fit of a model.

In more complex models, this avoids over-fitting a model to the data.

Procedure for validation

```
## Testing prediction accuracy of models ----

# Create Training and Test data -
set.seed(17)  #Set random number generator so can replicate results

# randomly select 80% of rownumbers
training.rows <- sample(1:nrow(cars), 0.8*nrow(cars))

#training data - 80% of dataset
trainingData <- cars[training.rows, ]

#test data - 20% of dataset
testData  <- cars[-training.rows, ]

# Model based on training data only
fit.train <- lm(dist ~ speed, data=trainingData)
predicted.values <- predict(fit.train, testData)

# Check model fit as before
summary(fit.train)
AIC(fit.train)

# we can correlate actual values against predicted values
cor(testData$dist, predicted.values) #.85

# we can add in predictions as a column into our data
library(tidyverse)
library(modelr)

trainingData <- trainingData %>%  add_predictions(fit.train)
trainingData
```

```r
# We should assess if the out-of-sample (test data) mean squared error (MSE) (mean squared prediction error),
# is notably greater than the MSE for the in-sample (training data).

trainingData %>% summarise(MSE = mean((dist - pred)^2))

#       MSE
# 1 201.565

testData <- testData %>% add_predictions(mod1)
testData
testData %>% summarise(MSE = mean((dist - pred)^2))

#        MSE
# 1 325.9917
```

Here we can see that the mean square error appears higher in the test out-of-sample data compared to the training data, which might be cause for concern about the fit of our model.

Multiple Regression

```r
> advertising <- read.csv("Advertising.csv")
> head(as.data.frame(advertising))
     TV radio newspaper sales
1 230.1  37.8      69.2  22.1
2  44.5  39.3      45.1  10.4
3  17.2  45.9      69.3   9.3
4 151.5  41.3      58.5  18.5
5 180.8  10.8      58.4  12.9
6   8.7  48.9      75.0   7.2
```

In multiple regression we include multiple predictors in the model. This requires each predictor to have it's own estimate.

Here we are trying to predict sales based on money spent on TV, radio or newspaper advertising.

```
> # Generate model
> mod2 <- lm(sales ~ TV + radio + newspaper, data = advertising)
> summary(mod2)

Call:
lm(formula = sales ~ TV + radio + newspaper, data = advertising)

Residuals:
    Min      1Q  Median      3Q     Max 
-8.8277 -0.8908  0.2418  1.1893  2.8292 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)  2.938889   0.311908   9.422   <2e-16 ***
TV           0.045765   0.001395  32.809   <2e-16 ***
radio        0.188530   0.008611  21.893   <2e-16 ***
newspaper   -0.001037   0.005871  -0.177     0.86    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.686 on 196 degrees of freedom
Multiple R-squared:  0.8972,	Adjusted R-squared:  0.8956 
F-statistic: 570.3 on 3 and 196 DF,  p-value: < 2.2e-16

>
> #confidence intervals
> confint(mod2)
                  2.5 %     97.5 %
(Intercept)  2.32376228 3.55401646
TV           0.04301371 0.04851558
radio        0.17154745 0.20551259
newspaper   -0.01261595 0.01054097
>
```

Models can be made using **lm()** and summary information can be derived as before.

Each predictor is included on the right hand side of the ~ in the model formula.

Compare multiple regression to simple linear regression:

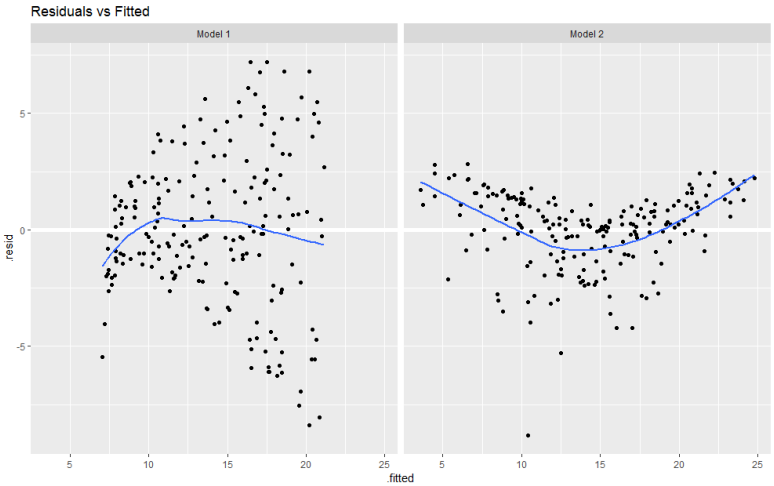
```
> mod1 <- lm(sales ~ TV, data = advertising)
> list(mod1 = broom::glance(model1), mod2 = broom::glance(model2))
$`mod1`
  r.squared adj.r.squared    sigma statistic     p.value df    logLik      AIC      BIC deviance df.residual
1 0.6118751     0.6099148 3.258656   312.145 1.46739e-42  2 -519.0457 1044.091 1053.986 2102.531         198

$mod2
  r.squared adj.r.squared   sigma statistic      p.value df    logLik      AIC      BIC deviance df.residual
1 0.8972106     0.8956373 1.68551  570.2707 1.575227e-96  4 -386.1811 782.3622 798.8538 556.8253         196

>
```

Model 2's R^2=.90 is much higher than that for model1 indicating that model2 explains more of the variance in sales. More importantly the adjusted R^2 is higher (this is adjusted for the number of predictors in the model).

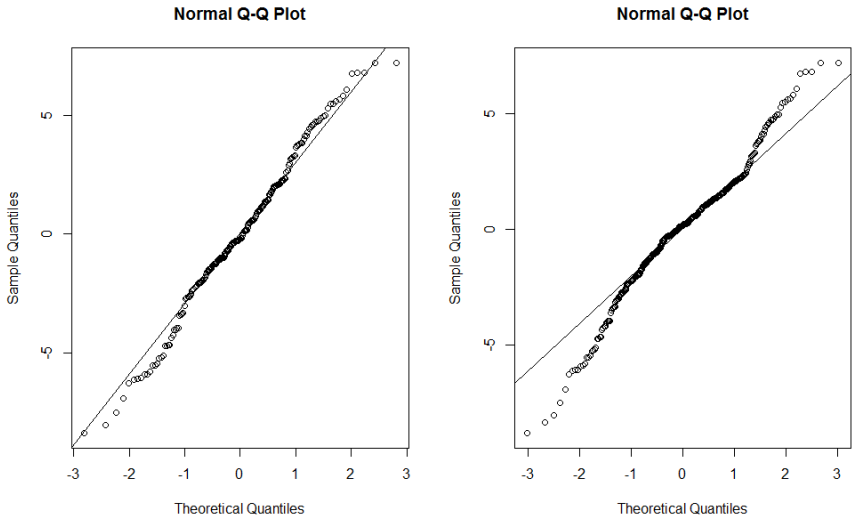
The RSE (sigma) of model 2 is lower than model 1 showing that we have reduced the variance of our parameters.

The AIC and BIC are also substantially lower in model 2.

Diagnostic plots should be made as described before – here model 2 appears to have issues with heteroscedasticity



Diagnostic plots should be made as described before – here model 2 appears to have issues with residuals not being normally distributed



Including interactions in the model

```
> mod3 <- lm(sales ~ TV * radio, data = advertising)
>
>
> tidy(mod3)
         term    estimate    std.error statistic      p.value
1 (Intercept) 6.750220203 0.2478713699 27.232755 1.541461e-68
2          TV 0.019101074 0.0015041455 12.698953 2.363605e-27
3       radio 0.028860340 0.0089052729  3.240815 1.400461e-03
4    TV:radio 0.001086495 0.0000524204 20.726564 2.757681e-51
>
> list(model1 = broom::glance(mod1),
+      model2 = broom::glance(mod2),
+      model3 = broom::glance(mod3))
$`model1`
  r.squared adj.r.squared    sigma statistic     p.value df    logLik      AIC      BIC deviance df.residual
1 0.6118751     0.6099148 3.258656   312.145 1.46739e-42  2 -519.0457 1044.091 1053.986 2102.531         198

$model2
  r.squared adj.r.squared   sigma statistic      p.value df    logLik      AIC      BIC deviance df.residual
1 0.8972106     0.8956373 1.68551  570.2707 1.575227e-96  4 -386.1811 782.3622 798.8538 556.8253         196

$model3
  r.squared adj.r.squared     sigma statistic       p.value df    logLik      AIC      BIC deviance df.residual
1 0.9677905     0.9672975 0.9435154  1963.057 6.684247e-146  4 -270.1389 550.2778 566.7694 174.4834         196
```

Collinearity

When one or more predictor variables in a multiple regression model can be linearly predicted from the others then we have collinearity

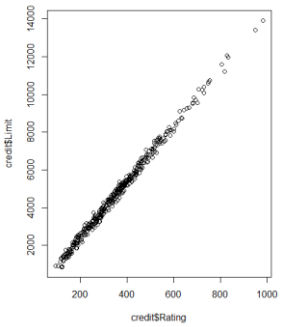
Example dataset:

```
> credit<-read.csv("credit.csv")
> head(credit)
   Income Limit Rating Cards Age Education Gender Student Married Ethnicity Balance
1  14.891  3606    283     2  34        11   Male      No     Yes Caucasian     333
2 106.025  6645    483     3  82        15 Female     Yes     Yes     Asian     903
3 104.593  7075    514     4  71        11   Male      No      No     Asian     580
4 148.924  9504    681     3  36        11 Female      No      No     Asian     964
5  55.882  4897    357     2  68        16   Male      No     Yes Caucasian     331
6  80.180  8047    569     4  77        10   Male      No      No Caucasian    1151
>
```

Collinearity

```
> mod4 <- lm(Balance ~ Age + Limit, data = credit)
> mod5 <- lm(Balance ~ Rating + Limit, data = credit)
>
> list(`Model 1` = tidy(mod4),
+      `Model 2` = tidy(mod5))
$`Model 1`
         term    estimate    std.error statistic       p.value
1 (Intercept) -173.410901 43.828387048 -3.956589  9.005366e-05
2         Age   -2.291486  0.672484540 -3.407492  7.226468e-04
3       Limit    0.173365  0.005025662 34.495944 1.627198e-121

$`Model 2`
         term      estimate   std.error  statistic      p.value
1 (Intercept) -377.53679536 45.25417619 -8.3425846 1.213565e-15
2      Rating    2.20167217  0.95229387  2.3119672 2.129053e-02
3       Limit    0.02451438  0.06383456  0.3840298 7.011619e-01

>
```

In the second model we have collinearity between limit and rating - this has caused the standard error of the estimate to increase dramatically meaning the result is no longer significant



Identify collinearity by examining correlation matrix of predictors

Multicollinearity

Also possible that no correlation will exist between any two predictors, but three or more variables may be highly interrelated – this is multicollinearity.

Can be tested in R using the **vif()** function from the 'car' package.

```
car::vif(mod4)
#      Age    Limit
# 1.010283 1.010283

car::vif(mod5)
#   Rating    Limit
# 160.4933 160.4933
```

VIF is the variance inflation factor – if it is close to 1 then it suggests no collinearity. If it is >5 then there is high collinearity. If it is between 1 and 5 then there is some collinearity.

Partial F-test to test if variables add substantially to a model

```
> head(housing)
# A tibble: 6 x 5
   sqft  price City       bedrooms baths
  <int>  <int> <chr>         <int> <dbl>
1  3392 339000 Dublin            3   2.1
2  4100 899900 pleasanton        4   3
3  3200 448641 Clayton           5   4
4  1436 239999 Moraga            4   3
5  1944 377500 Antioch           3   2
6  1500 299900 Danville          3   2.5
>
```

```
simple <-lm(price ~ sqft, data=housing) #Simple model
reduced = lm(price ~ sqft + bedrooms, data=housing) # Reduced model
full = lm(price ~ sqft + bedrooms + baths, data=housing) # Full model


anova(simple,reduced, full)

# Analysis of Variance Table
#
# Model 1: price ~ sqft
# Model 2: price ~ sqft + bedrooms
# Model 3: price ~ sqft + bedrooms + baths
#   Res.Df        RSS Df  Sum of Sq      F Pr(>F)
# 1     81 3.9969e+12
# 2     80 3.8991e+12  1 9.7842e+10 1.9913 0.1621
# 3     79 3.8817e+12  1 1.7399e+10 0.3541 0.5535
```

Use **anova()** to compare models.

Appears 'sqft' is the only variable needed to predict price

Model Selection

Deciding which predictors/variables to include in a model can be difficult.

There are however some automatic model selection methods that can assist with this.

The package 'leaps' has some of these – we will discuss two: Best Subset Selection and Stepwise Selection.

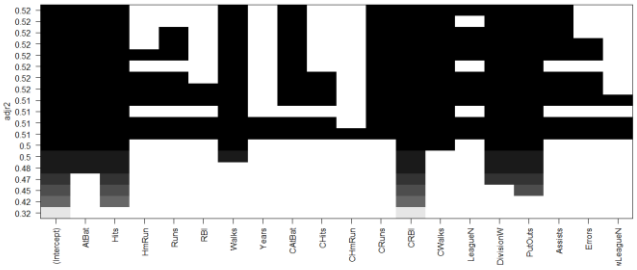
```
> hitters<-read.csv("hitters.csv")
> dim(hitters)
[1] 263  21
> head(hitters)
                  X AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI CWalks
1       -Alan Ashby   315   81     7   24  38    39    14   3449   835     69   321  414    375
2      -Alvin Davis   479  130    18   66  72    76     3   1624   457     63   224  266    263
3     -Andre Dawson   496  141    20   65  78    37    11   5628  1575    225   828  838    354
4 -Andres Galarraga   321   87    10   39  42    30     2    396   101     12    48   46     33
5  -Alfredo Griffin   594  169     4   74  51    35    11   4408  1133     19   501  336    194
6        -Al Newman   185   37     1   23   8    21     2    214    42      1    30    9     24
  League Division PutOuts Assists Errors Salary NewLeague
1      N        W     632      43     10  475.0         N
2      A        W     880      82     14  480.0         A
3      N        E     200      11      3  500.0         N
4      N        E     805      40      4   91.5         N
5      A        W     282     421     25  750.0         A
6      N        E      76     127      7   70.0         A
>
```

Example Dataset

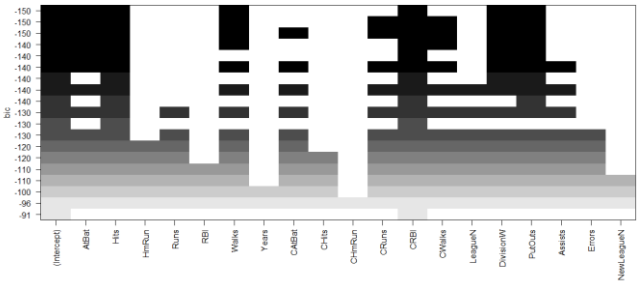
An example – forward stepwise selection

```
# Forward stepwise:
# 1. select the null model with no predictors
# 2. find best model after adding each of the predictors
# 3. to that model add each predictor in turn and find the best.
# 4. repeat 2&3 until added all predictors.
# 5. select best model from those kept.

forward <- regsubsets(Salary ~ ., hitters, nvmax = 19, method = "forward")
```

Plots to evaluate model fit



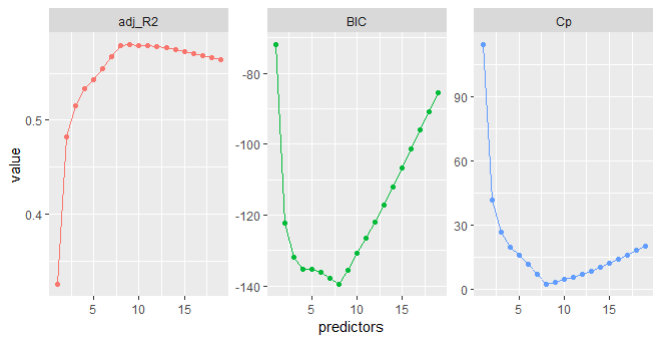

```
plot(forward, scale = "adjr2")
```




```
plot(forward, scale = "bic")
```

Using an out-of-sample dataset to evaluate model selection.

```
# create training - testing data
set.seed(17)
sample1 <- sample(c(TRUE, FALSE), nrow(hitters), replace = T, prob = c(0.75,0.25))
train <- hitters[sample1, ]
test <- hitters[!sample1, ]

# Forward Stepwise Selection
best_subset <- regsubsets(Salary ~ ., train, nvmax = 19, method = "forward")
res <- summary(best_subset)
res

# extract and plot results
data.frame(predictors = 1:19,
       adj_R2 = res$adjr2,
       Cp = res$cp,
       BIC = res$bic) %>%
  gather(statistic, value, -predictors) %>%
  ggplot(aes(predictors, value, color = statistic)) +
  geom_line(show.legend = F) +
  geom_point(show.legend = F) +
  facet_wrap(~ statistic, scales = "free")

which.max(res$adjr2)
## [1] 9
which.min(res$bic)
## [1] 8
which.min(res$cp)
## [1] 8
```



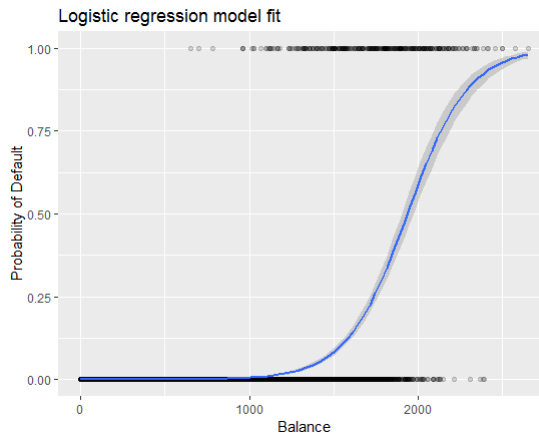
4c. Logisitic Regression

We use the logistic regression when the response (outcome) variable is discrete and error terms do not follow the normal distribution.

Plotting data and adding a logistic regression fit curve

```
> head(default)
# A tibble: 6 x 4
  default student balance income
  <chr>   <chr>     <dbl>  <dbl>
1 No      No         730. 44362.
2 No      Yes        817. 12106.
3 No      No        1074. 31767.
4 No      No         529. 35704.
5 No      No         786. 38463.
6 No      Yes        920.  7492.
>
```

```
default %>%
  mutate(prob = ifelse(default == "Yes", 1, 0)) %>%
  ggplot(aes(balance, prob)) +
  geom_point(alpha = .15) +
  geom_smooth(method = "glm", method.args = list(family = "binomial")) +
  ggtitle("Logistic regression model fit") +
  xlab("Balance") +
  ylab("Probability of Default")
```



Use **glm()** to perform a generalized linear model and **family="binomial"** to set as a logistic regression

```
> mod1 <- glm(default01 ~ balance, family = "binomial", data = train)
> mod1

Call:  glm(formula = default01 ~ balance, family = "binomial", data = train)

Coefficients:
(Intercept)      balance
 -11.006278     0.005669

Degrees of Freedom: 6046 Total (i.e. Null);  6045 Residual
Null Deviance:      1723
Residual Deviance: 908.7        AIC: 912.7
```

```
> summary(mod1)  #remember estimates are on log-likelihood scale.

Call:
glm(formula = default01 ~ balance, family = "binomial", data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.2905  -0.1395  -0.0528  -0.0189   3.3346

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.101e+01  4.887e-01  -22.52   <2e-16 ***
balance      5.669e-03  2.949e-04   19.22   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1723.03  on 6046  degrees of freedom
Residual deviance:  908.69  on 6045  degrees of freedom
AIC: 912.69

Number of Fisher Scoring iterations: 8

> glance(mod1)
  null.deviance df.null    logLik      AIC      BIC deviance df.residual
1      1723.027    6046 -454.3437 912.6874 926.1021 908.6874        6045
>
```

If the (residual) deviance is substantially reduced compared to the null deviance then suggests model is a good fit

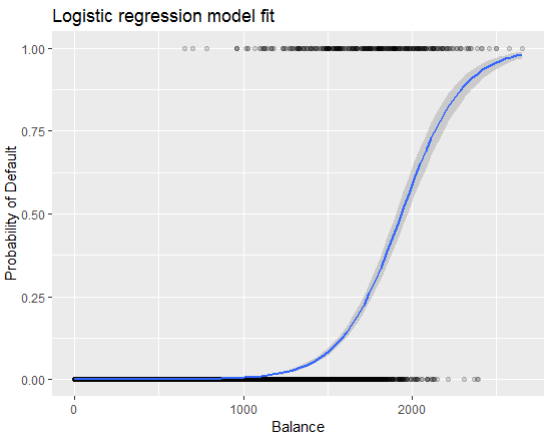
Estimates can be hard to interpret in logistic regression as they are log odds.

```
> tidy(mod1)
         term      estimate   std.error statistic       p.value
1 (Intercept) -11.006277528 0.488739437 -22.51972 2.660162e-112
2     balance   0.005668817 0.000294946  19.21985  2.525157e-82
> coef(mod1)
  (Intercept)       balance
-11.006277528   0.005668817
> coef(summary(mod1))
                 Estimate  Std. Error   z value      Pr(>|z|)
(Intercept) -11.006277528 0.488739437 -22.51972 2.660162e-112
balance       0.005668817 0.000294946  19.21985  2.525157e-82
> exp(coef(mod1))
 (Intercept)      balance
1.659718e-05 1.005685e+00
```

Here, a one unit increase in balance is associated with an increase in the log odds of defaulting of 0.0057 units.

However, exponentiating the coefficients – a one dollar increase in monthly balance is associated with a 1.0057 increase in risk of defaulting.

```
> predict(mod1, data.frame(balance = c(1000, 2000)), type = "response")
          1           2
0.004785057 0.582089269
>
```

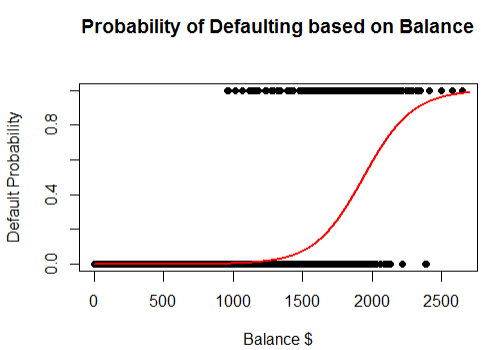


Can use predict() to make predictions.

Here the results are provided in probabilities –

So a monthly balance of $2000 leads to a 58.2% likelihood of defaulting all other things being equal.

```
> range.x <- seq(0, 2700, 0.1)  #create a range of x values between min/max of x
> ypredict <- predict(mod1, list(balance = range.x),type="response")  #predict y values for range of x using model
> default01 <- ifelse(train$default=="No", 0 ,1) #making sure default is 0or1 (as no/yes, plot converts to 1or2)
> plot(train$balance, default01, pch = 16,
+      xlab = "Balance $", ylab = "Default Probability",
+      main = "Probability of Defaulting based on Balance") # plot
> lines(range.x, ypredict,  col = "red", lwd = 2)
>
```



Generating the logistic regression plot in base-r

Example including categorical variables in a logistic regression

```
> mod2 <- glm(default01 ~ student, family = "binomial", data = train)
> tidy(mod2)
         term   estimate  std.error statistic     p.value
1 (Intercept) -3.5534091 0.09336545 -38.05914 0.000000000
2  studentYes  0.4413379 0.14927208   2.95660 0.003110511
> summary(mod2)

Call:
glm(formula = default01 ~ student, family = "binomial", data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-0.2951  -0.2951  -0.2376  -0.2376   2.6764

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -3.55341    0.09337 -38.059  < 2e-16 ***
studentYes   0.44134    0.14927   2.957  0.00311 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1723.0  on 6046  degrees of freedom
Residual deviance: 1714.6  on 6045  degrees of freedom
AIC: 1718.6

Number of Fisher Scoring iterations: 6

>
```

```
predict(mod2, data.frame(student = factor(c("Yes", "No"))), type = "response")

#          1          2
# 0.04261206 0.02783019
```

Notably the increase in probability for defaulting if a student is very small even though it's significant.

```
> mod3 <- glm(default01 ~ balance + income + student, family = "binomial", data = train)
> summary(mod3)  #The student estimate has turned to negative.

Call:
glm(formula = default01 ~ balance + income + student, family = "binomial",
    data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.4556  -0.1344  -0.0499  -0.0174   3.4155

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.091e+01  6.481e-01 -16.830  < 2e-16 ***
balance      5.907e-03  3.102e-04  19.040  < 2e-16 ***
income      -5.013e-06  1.079e-05  -0.465  0.64212
studentYes  -8.095e-01  3.133e-01  -2.584  0.00978 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1723.03  on 6046  degrees of freedom
Residual deviance:  895.02  on 6043  degrees of freedom
AIC: 903.02

Number of Fisher Scoring iterations: 8

>
```

Multiple logisitic regression

Potentially some collinearity:

```
> car::vif(mod3) #test for collinearity
 balance   income  student
1.078081 2.697636 2.801508
```

```
> # predictions for multiple logistic regression
>
> predictdf <- data.frame(balance = 1500, income = 40, student = c("Yes", "No"))
> predict(mod3, predictdf, type = "response")
         1          2
0.05437124 0.11440288
```

Using predict() to make predictions for multiple logistic regression.

Here, students are about half as likely to default as non-student for given income and balance.

Logisitic Regression Model Diagnostics

Likelihood Ratio Test to Evaluate if our models are improving the fit of the data:

```
> mod0 <- glm(default01 ~ 1, family = "binomial", data = train)
> anova(mod0, mod1, mod3, test = "Chisq") #p<.05 can reject null.
Analysis of Deviance Table

Model 1: default01 ~ 1
Model 2: default01 ~ balance
Model 3: default01 ~ balance + income + student
  Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
1      6046    1723.03
2      6045     908.69  1   814.34 < 2.2e-16 ***
3      6043     895.02  2    13.67  0.001076 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
>
```

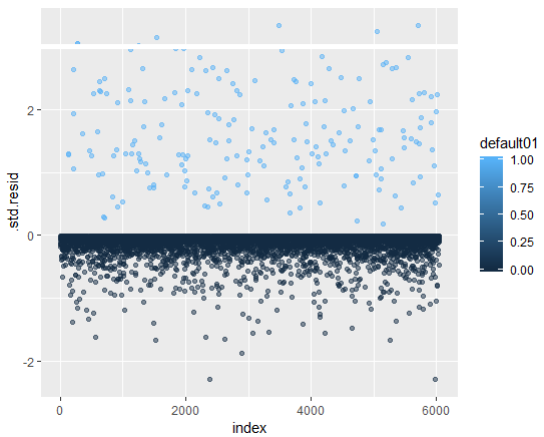
Although there is no R^2 statistic for a logistic regression, we can calculate McFadden's R^2 which can be used as an approximation. We use the function **pR2()** from the 'pscl' package.

McFadden's R^2 ranges from 0-1 but is highly unlikely to ever get to 1. Indeed, a McFadden's R^2 around .40 indicates a very good fit.

```
> list(mod1 = pscl::pR2(mod1)["McFadden"],
+      mod2 = pscl::pR2(mod2)["McFadden"],
+      mod3 = pscl::pR2(mod3)["McFadden"])
$`mod1`
 McFadden
0.4726215

$mod2
   McFadden
0.004898314

$mod3
 McFadden
0.4805543

>
```

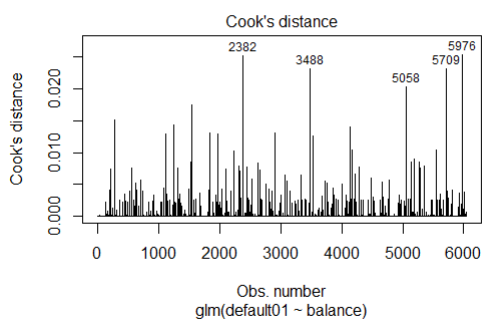
Residuals

```
mod1.r <- augment(mod1) %>%  mutate(index = 1:n())

ggplot(mod1.r, aes(index, .std.resid, color = default01)) +
  geom_point(alpha = .5) +
  geom_ref_line(h = 3)
```



Although residuals do not have to be normally distributed and variances does not have to be constant across groups for logistic regression – it is worth checking standardized residuals. Those that are above 3 (or below -3) may be possible outliers.

```
# 4. Cook's Distances
plot(mod1, which = 4, id.n = 5)
```



As with other models, we can look at Cook's distances to identify points with high leverage

Evaluating classification success

```
> m1 <- predict(mod1, newdata = test, type = "response")
> mod1 = table(test$default, m1 > 0.5) %>% prop.table() %>% round(3)
> mod1

      FALSE  TRUE
  No  0.962 0.003
  Yes 0.025 0.010
>
```

If we use a validation method (training/testing) we can identify the success of our model on the out-of-sample test data.

```
> table(test$default, m1 > 0.5)

      FALSE TRUE
  No   3803   12
  Yes    98   40
>
```

Here, for model1, 96.2% of cases are true negatives. 1% of cases are true positives. 2.5% are false negatives, and 0.3% are false positives.

We have a lower error rate (only 12/3815) were wrongly classified as not defaulting.

But low precision – as only 40/138 were correctly identified as defaulters.

5. Generalized Linear Models

Poisson Regression

In a Poisson Regression, the response variable are the number of occurrences (frequency) of an event with no upper bound. The response variable must allow for counts of 0 however.

e.g. number of aggressive acts per animal
e.g. number of awards earned by each student in a high school
e.g. number of robberies per month

Typically these types of data the distribution is such that often the mode value is 0, the next most common is 1, then 2 etc. Transforming data in any way monotonically will never shift the mode to the middle of the distribution.

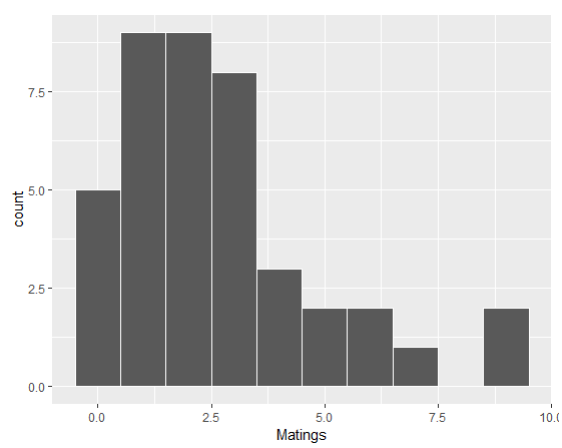
Also count data are bounded by 0 as a lower limit - linear regression assumes a continuous response variable -infinity to +infinity

As count data often fall into a Poisson distribuion, a poisson regression can be used. As with logistic regression - poisson regression uses maximum likelihood estimation, and we transform the response variable (Y) via a link function. The link function in natural log (so need to transform coefficients back for interpretation).

```
> elephants <- read.csv("elephants.csv")
> dim(elephants)
[1] 41  2
> head(elephants)
  Age Matings
1  27       0
2  28       1
3  28       1
4  28       1
5  28       3
6  29       0
```

```
> table(elephants$Matings)

0 1 2 3 4 5 6 7 9
5 9 9 8 3 2 2 1 2
>
```

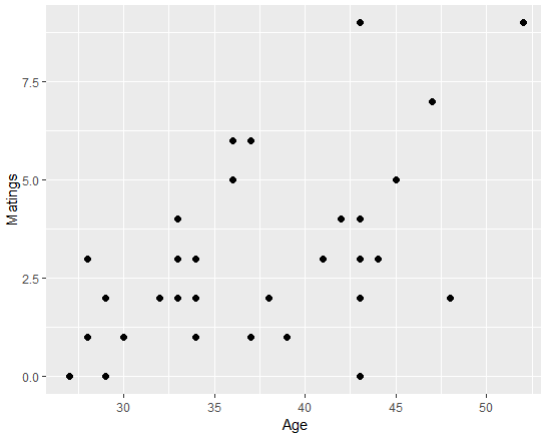
```
> ggplot(elephants, aes(x = Matings)) + geom_histogram(color='white', bins=10)
>
```



Example dataset – number of matings by elephants by age.

The histogram illustrates the skewed distribution of the data.

```r
# scatterplot
ggplot(elephants, aes(x = Age, y = Matings)) + geom_point(size = 2)
```



We can use a Poisson regression to investigate the relationship between Age and Matings.

Use **glm()** to build the model

```
> mod1 <- glm(Matings ~ Age, data = elephants, family = "poisson")
> summary(mod1)

Call:
glm(formula = Matings ~ Age, family = "poisson", data = elephants)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.80798  -0.86137  -0.08629   0.60087   2.17777

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.58201    0.54462  -2.905  0.00368 **
Age          0.06869    0.01375   4.997 5.81e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 75.372  on 40  degrees of freedom
Residual deviance: 51.012  on 39  degrees of freedom
AIC: 156.46

Number of Fisher Scoring iterations: 5
```

One important thing to note is the relationship between residual deviance and degrees of freedom. If the residual deviance is much higher than the d.f. then suggests over-dispersion (variance not being accounted for by the model).

Here 51.01 > 39 - this is a ratio of 1.3. Some suggest up to 1.5 is ok, others suggest it needs to be close to 1.0

```r
# If variance > mean - also indicative of over-dispersion
mean(elephants$Matings) # 2.682927
var(elephants$Matings)  # 5.071951
```

```
> glance(mod1)
  null.deviance df.null   logLik      AIC      BIC deviance df.residual
1      75.37174      40 -76.2289 156.4578 159.8849 51.01163          39
> tidy(mod1)
         term    estimate  std.error statistic      p.value
1 (Intercept) -1.58200796 0.54462132 -2.904785 3.675052e-03
2         Age  0.06869281 0.01374578  4.997375 5.811590e-07
>
```

We can use the usual functions in the package 'broom' to retrieve model summary information.

```
> coef(mod1)
(Intercept)         Age
-1.58200796  0.06869281
> exp(coef(mod1))
(Intercept)         Age
  0.2055619   1.0711071
> exp(confint.default(mod1))
                 2.5 %     97.5 %
(Intercept) 0.07069036 0.5977577
Age         1.04263544 1.1003563
>
```

Positive coefficient for Age indicates that as Ages increases so does Matings.

Exponentiating the coefficients shows that for a 1 unit increase in Agem there is a 7.1% increase in matings, or there will be an increase in matings of 1.07 times.

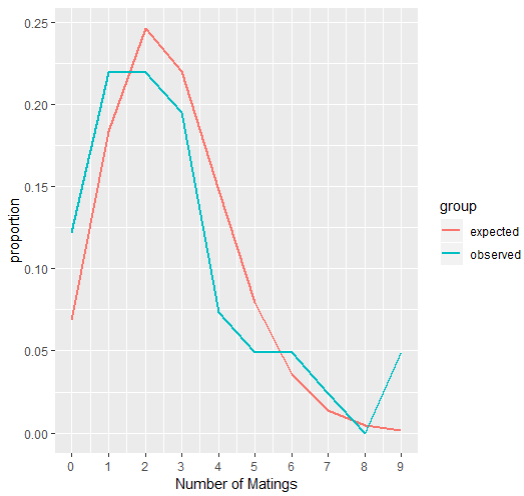
Use **predict()** to obtain predicted values to plot trendline

```
# Get predicted values ----
data.frame(elephants,pred=mod1$fitted)


range(elephants$Age) #[1]    27 52
range.x <- seq(27, 52, .1)  #create a range of x values between min/max of x
ypredict <- predict(mod1, list(Age = range.x),type="response")  #predict y values for range of x using model

plot(elephants$Age, elephants$Matings, pch = 16,
     xlab = "Age", ylab = "Matings",
     main = "Poisson Regression Model") # plot
lines(range.x, ypredict,  col = "red", lwd = 2)
```

Another assumption of a Poisson regression is that no observed counts should be substantially more or less than those expected from a Poisson distribution. That is, there should not be substantially more 0s (or 1s, 2s) than expected by a Poisson distribution with a given mean.

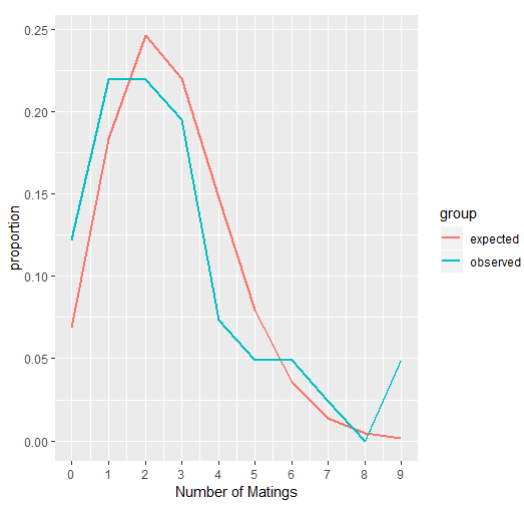
```
#observed mean
obs.mean <- mean(elephants$Matings)
obs.mean

# Generate Y values expected from a Poisson distribution
dpois(0:9, lambda = obs.mean)
```

Use **dpois()** to generate observed values for our mean, and then ggplot2 to plot



Visually this can be helpful, though can also perform a Chi-squared test to test whether the distributions (observed vs expected) are similar:

```
totalN <- sum(elephants$Matings)

df %>% mutate(count=proportion * totalN) %>%
  dplyr::select(group,number,count) %>%
    spread(group,count) -> df1

df1

chisq.test(df1$observed, df1$expected)

# Pearson's Chi-squared test
#
# data:  df1$observed and df1$expected
# X-squared = 60, df = 54, p-value = 0.2673

## The chi-square test here says that expected and observed
## are likely to come from the same population
```



Pearson's X^2 dispersion statistic (Hilbe 2011)

```
> N <- length(elephants$Age)
> p <- length(coef(mod1))
> sum(mod1.resids^2) / (N-p) #1.157334
[1] 1.157334
>
> #function
> msme::P__disp(mod1)
pearson.chi2   dispersion
   45.136039     1.157334
>
> # pearson.chi2   dispersion
> #    45.136039     1.157334
>
```

An additional check is the Pearson X^2 dispersion statistic as described by Hilbe.

A value of 1 indicates that the variances of the observed and predicted variables are the same, and a value over 1 indicates that there is over-dispersion.

Hilbe recommends overdispersion if value is >1.25 for moderate sized models, and >1.05 for large numbers of observations (Hilbe p82 Modeling Count Data 2016).

The msme package has a function **P_disp()** that performs this test.

Deviance Goodness of Fit Test

We can also test how much variation in observed vs predicted outcomes we should expect if our model is a good fit of the data.

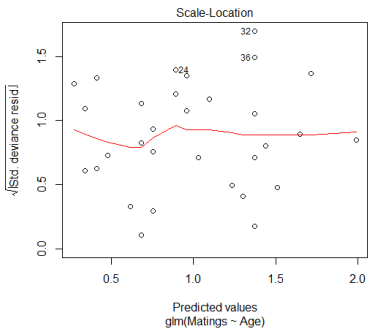
The deviance follows a chi-squared distribution, with df = n - p (n is number of observations, p is number of parameters)

```
> pchisq(mod1$deviance, mod1$df.residual, lower.tail = F) #0.094
[1] 0.09426231
> 1-pchisq(mod1$deviance, mod1$df.residual) #0.094 #sometimes written like this
[1] 0.09426231
>
```

Here we have reason to believe that we cannot reject the null hypothesis that our model is correctly specified – so this suggests our model is a good fit.

Scale-Location Plot

```
# Scale-Location plot
plot(mod1,which=3)
```



This plot examines whether the model is correctly scaling the variance with the mean. It should be an approximately even scatter of points, with a fairly straight line through it.

Quasi-Poisson Regression

```
> mod2 <- glm(Matings ~ Age, data = elephants, family = "quasipoisson")
> summary(mod2)

Call:
glm(formula = Matings ~ Age, family = "quasipoisson", data = elephants)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.80798  -0.86137  -0.08629   0.60087   2.17777

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.58201    0.58590  -2.700   0.0102 *
Age          0.06869    0.01479   4.645 3.81e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for quasipoisson family taken to be 1.157334)

    Null deviance: 75.372  on 40  degrees of freedom
Residual deviance: 51.012  on 39  degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 5

> coef(mod2)
(Intercept)         Age
-1.58200796  0.06869281
>
```

If our Poisson Regression is not suitable for our data – one option is to use a quasi-Poisson regression that adjusts for overdispersion.

This can be undertaken by adding to the model: family = “quasipoisson”

Negative Binomial Regression

An alternative approach for over-dispersed data may be to use a negative binomial regression.
The variance of a negative binomial distribution is a function of its mean and an additional parameter k (dispersion parameter).

```
> mod3 <- MASS::glm.nb(Matings ~ Age, data = elephants)
> summary(mod3)

Call:
MASS::glm.nb(formula = Matings ~ Age, data = elephants, init.theta = 16.48629005,
    link = log)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.66032  -0.81966  -0.08079   0.52865   1.89021

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.58837    0.59393  -2.674  0.00749 **
Age          0.06887    0.01519   4.534  5.8e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Negative Binomial(16.4863) family taken to be 1)

    Null deviance: 65.199  on 40  degrees of freedom
Residual deviance: 44.498  on 39  degrees of freedom
AIC: 157.92

Number of Fisher Scoring iterations: 1


              Theta:  16.5
          Std. Err.:  25.7

 2 x log-likelihood:  -151.923
>
```

Use the function glm.nb() from the package 'MASS' to run a negative binomial regression.

```
> coef(mod3)
(Intercept)         Age
-1.58837248  0.06886884
> coef(summary(mod3))
               Estimate Std. Error   z value     Pr(>|z|)
(Intercept) -1.58837248 0.59392961 -2.674345 7.487546e-03
Age          0.06886884 0.01519092  4.533552 5.800006e-06
>
```