University of Zurich UZH

Makerere University

# Data Analysis with R:
## Lecture Slides (all)

Sonja Hartnack, Terence Odoch & Muriel Buri

October 2018

# Lecture Slides for Day 1

# Goals of the course

To be able to...

- import data sets to R
- describe data with R
- apply basic statistical tests in R
- some ideas for more advanced statistical tools ...
- simulate a data set similar to own research

# General remarks

**Course schedule:**

- Starting at 9:00am / 9:30am (?)
- Tea breaks in between
- Lunch break
- Teaching until 4.30pm ($\sim$ 5pm)

**Optaining a certificate is conditional on:**

- active participation in class
- attending at least 75 % of the course (lecture & exercises)
- short final exam (format to be defined)

# Getting to know each other

- My name is ...
- I am doing a Master / a PhD in ...
- I hope to learn in this course how to ....
- My personal goal for this course is ...

# How do we reach these goals

- hands on exercises with R:
  - `chickwts`
  - `ToothGrowth`
  - `bacteria`
  - `perulung`
  - ... and others.
- interactive discussions & student's present their own solutions
- ask us a lot of questions but also ask google for help!
- group work
- short motivational lectures

# Get started with data set: chickwts

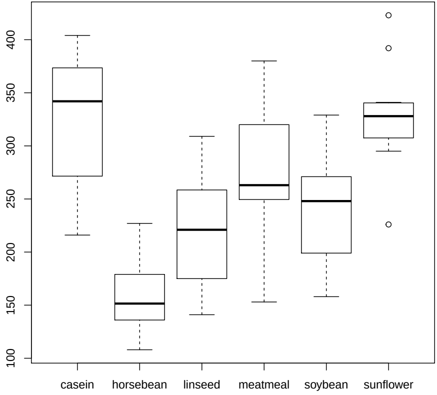
An experiment was conducted to measure and compare the effectiveness of various feed supplements on the growth rate of chickens.

```
# load data set "chickwts"
data(chickwts)
# the head(...) function shows the first 6 observations
head(chickwts)

##   weight      feed
## 1    179 horsebean
## 2    160 horsebean
## 3    136 horsebean
## 4    227 horsebean
## 5    217 horsebean
## 6    168 horsebean

# FUNCTION - open bracket - DATA SET / VARIABLE - close bracket
```

# Ideas for plotting the data

# Ideas for plotting the data

```
# use x axis to show the categorical variable (feed),
# y axis to represent the continuous variable (weight)
# boxplot (y.cont.variable ~ x.cat.variable, data = dataset)
# ?boxplot
boxplot(weight ~ feed, data = chickwts)
```



# Ideas for analysing the data

# Ideas for analysing the data

```
anova <- aov(weight ~ feed, data = chickwts)
summary(anova)

##              Df Sum Sq Mean Sq F value   Pr(>F)
## feed          5 231129   46226   15.37 5.94e-10 ***
## Residuals    65 195556    3009
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

summary(aov(weight ~ feed, data = chickwts))

##              Df Sum Sq Mean Sq F value   Pr(>F)
## feed          5 231129   46226   15.37 5.94e-10 ***
## Residuals    65 195556    3009
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

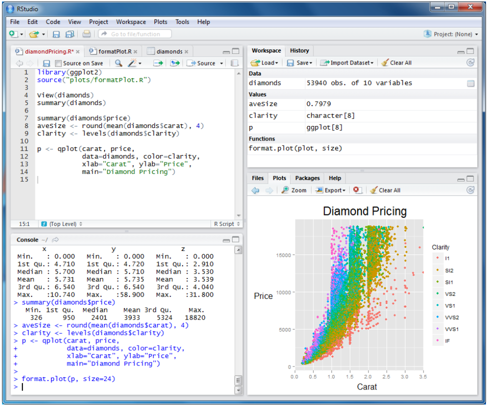
# Functionalities in R and RStudio

## A hands on example

```
x <- c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
y <- c(20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30)
plot(x, y)
```

# Functionalities in R and RStudio

- Source
- Console
- Environment, History, Files
- Files, Plots, Packages, Help



# Good housekeeping!

- Define manually a new folder called **RCourse** in your personal documents on your personal computer

- Know in which directory you are

```
getwd()

## [1] "/home/mburi/ownCloud/git/DataAnalysisWithR/Lectures"
```

- Set directory path

```
# back- and forslash is dependent on the system
setwd("C:/Users/muriel/Documents/RCourse/")
setwd("C:\\Users\\muriel\\Documents\\RCourse\\")
```

- Always clean up before starting with new R-Script

```
rm(list=ls()) # empty workspace, delete previously saved variables
```

# How to get help in R

```
?chickwts
?boxplot
```

Also, have a look at the examples at the end of the help pages.

# Exercise 1

# A data frame in R: `chickwts`

**chickwts[ ROWS , COLUMNS ]**

| | weight | feed |
|---|---|---|
| 1 | 179 | horsebean |
| 2 | 160 | horsebean |
| 3 | 136 | horsebean |
| 4 | 227 | horsebean |
| 5 | 217 | horsebean |
| 6 | 168 | horsebean |
| 7 | 108 | horsebean |
| 8 | 124 | horsebean |
| 9 | 143 | horsebean |
| 10 | 140 | horsebean |
| 11 | 309 | linseed |
| 12 | 229 | linseed |
| 13 | 181 | linseed |

**chickwts[ 6, 1 ]**

| | weight | feed |
|---|---|---|
| 1 | 179 | horsebean |
| 2 | 160 | horsebean |
| 3 | 136 | horsebean |
| 4 | 227 | horsebean |
| 5 | 217 | horsebean |
| 6 | 168 | horsebean |
| 7 | 108 | horsebean |
| 8 | 124 | horsebean |
| 9 | 143 | horsebean |
| 10 | 140 | horsebean |
| 11 | 309 | linseed |
| 12 | 229 | linseed |
| 13 | 181 | linseed |

**chickwts[ 11, 2]**

# Rows and columns of a data frame: `chickwts`

Values of ...

```
# ... all columns of sixth observation:
chickwts[6, ]

# ... all columns of sixth to eleventh observation:
chickwts[c(6:11), ]

# ... all columns of sixth, eleventh and twentieth observation:
chickwts[c(6, 11, 20), ]

# ... all rows of first column (weight):
chickwts[ , 1]

# ... all rows of second column (feed):
chickwts[ , 2]

# or use the "$" sign as a reference to column "feed":
chickwts$feed
```

# Lecture Slides for Day 2

# What is a data frame in R?

A data frame is used for storing a list of vectors of equal length. For example, the following variable `df` is a data frame containing three vectors `n`, `s`, `b`.

```
n <- c(2, 3, 5)
s <- c("aa", "bb", "cc")
b <- c(TRUE, FALSE, TRUE)
df <- data.frame(n, s, b) # df is a data frame
```

The characteristics of a data frame are:

- The column names should be non-empty.
- The row names should be unique.
- Each column should contain same number of data items.

# Data frame in R

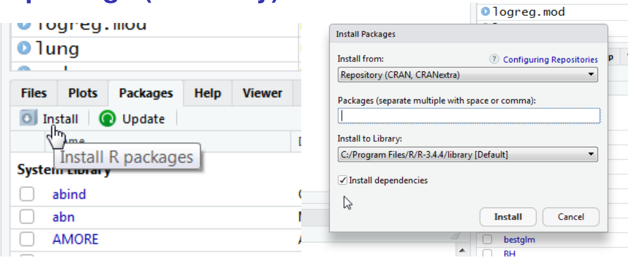
```r
a <- c(1, 2, 3, 4)
a

## [1] 1 2 3 4

data.frame(a)

##   a
## 1 1
## 2 2
## 3 3
## 4 4

b <- c("d", "h", "h", "d")
dat <- data.frame(a, b)
dat

##   a b
## 1 1 d
## 2 2 h
## 3 3 h
## 4 4 d
```

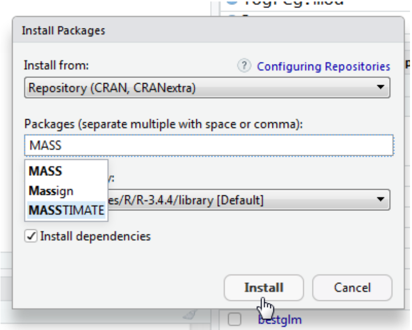
# Data frame in R: How to add a variable (var)

```
my.var <- c(1.3, 1.5, 1.8, 2.4)
# use "$" to refer to the additional vector variable
dat$my.var1 <- my.var
dat$my.var2 <- my.var
dat

##   a b my.var1 my.var2
## 1 1 d     1.3     1.3
## 2 2 h     1.5     1.5
## 3 3 h     1.8     1.8
## 4 4 d     2.4     2.4

# What is the dimension (number of rows and columns) of our data frame?
dim(dat) # 4 rows and 4 columns

## [1] 4 4
```

# Exercise 2

# How to install a package (manually) in R



# Using R is like cooking ...

| | |
|---|---|
| Get into the kitchen | Change working directory |
| Get specialist electric tools into your kitchen (e.g. blender, ice-cream maker, etc.) | Install packages |
| Switch on your specialist electric tools | Load packages using the "library" function |
| Bring in your ingredients | Import data and save to R data frames |
| Check your ingredients | Use the function "summary" and basic tables to check your data for missing or implausible values (e.g. a number in a variable where "yes" or "no" are expected |
| Chop things up (if required) | Split or filter data |
| Cook, using general and specialist tools | Carry out further descriptive and test statistics |

# How to install a package in R

```r
# INSTALL package (only done ONCE!)
install.packages("MASS")
# LOAD package (whenever you use something from it!)
library("MASS")
data(bacteria)
?bacteria
```

# Exercise 3

# How to google for getting help in R

- Google for **select observations in R**.

# Creating and assigning objects in R

Objects are assigned values using $<-$ ,
an arrow formed out of $<$ and $-$. For example, the following command assigns the value 1 to the object `a`.

```
a <- 1 # ALWAYS use "gets" assignment operator!
# a = 1 # DO NOT USE the equal sign as the assignment operator!
```

After this assignment, the object `a` contains the value 1. Another assignment to the same object will change its value.

```
a <- 5
```

# Examples of assigned objects: single number

```r
a <- 1
b <- 2
c <- a + b # c = 3
c

## [1] 3
```

# Examples of assigned objects: vector

```
a <- c(1, 2, 3, 4, 5)
b <- 1
c <- a + b
c

## [1] 2 3 4 5 6
```

# Examples of assigned objects: model

```
anova_model <- aov(weight ~ feed, data = chickwts)
summary(anova_model)

##             Df Sum Sq Mean Sq F value   Pr(>F)
## feed         5 231129   46226   15.37 5.94e-10 ***
## Residuals   65 195556    3009
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

# Examples of assigned objects: data frame

```
bac <- bacteria
str(bac) # $ week: int  0 2 4 11 0 2 6 11 0 2 ...

## 'data.frame': 220 obs. of  6 variables:
##  $ y   : Factor w/ 2 levels "n","y": 2 2 2 2 2 2 1 2 2 2 ...
##  $ ap  : Factor w/ 2 levels "a","p": 2 2 2 2 1 1 1 1 1 1 ...
##  $ hilo: Factor w/ 2 levels "hi","lo": 1 1 1 1 1 1 1 1 2 2 ...
##  $ week: int  0 2 4 11 0 2 6 11 0 2 ...
##  $ ID  : Factor w/ 50 levels "X01","X02","X03",..: 1 1 1 1 2 2 2 2 3 3 ...
##  $ trt : Factor w/ 3 levels "placebo","drug",..: 1 1 1 1 3 3 3 3 2 2 ...

bac_sub <- subset(bac, week == 2)
str(bac_sub) # $ week: int  2 2 2 2 2 2 2 2 2 2 ...

## 'data.frame': 44 obs. of  6 variables:
##  $ y   : Factor w/ 2 levels "n","y": 2 2 2 2 2 2 1 2 2 2 ...
##  $ ap  : Factor w/ 2 levels "a","p": 2 1 1 2 2 1 1 2 2 2 ...
##  $ hilo: Factor w/ 2 levels "hi","lo": 1 1 2 2 2 2 1 1 2 1 ...
##  $ week: int  2 2 2 2 2 2 2 2 2 2 ...
##  $ ID  : Factor w/ 50 levels "X01","X02","X03",..: 1 2 3 4 5 6 7 8 9 11 ...
##  $ trt : Factor w/ 3 levels "placebo","drug",..: 1 3 2 1 1 2 3 1 1 1 ...
```

# Structure of a R objects

The `str` function displays the structure of an R object. One line for each "basic" structure is displayed.

```
## 'data.frame': 44 obs. of  6 variables:
##  $ y   : Factor w/ 2 levels "n","y": 2 2 2 2 2 1 2 2 2 ...
##  $ ap  : Factor w/ 2 levels "a","p": 2 1 1 2 2 1 1 2 2 2 ...
##  $ hilo: Factor w/ 2 levels "hi","lo": 1 1 2 2 2 2 1 1 2 1 ...
##  $ week: int  2 2 2 2 2 2 2 2 2 2 ...
##  $ ID  : Factor w/ 50 levels "X01","X02","X03",..: 1 2 3 4 5 6 7 8 9 11 ...
##  $ trt : Factor w/ 3 levels "placebo","drug",..: 1 3 2 1 1 2 3 1 1 1 ...
```

# Exercise 4

# Data types in R

- numeric

```
data(ToothGrowth)
ToothGrowth$len[1:6]

## [1]  4.2 11.5  7.3  5.8  6.4 10.0

class(ToothGrowth$len[1:6])

## [1] "numeric"
```

- integers

```
bacteria$week[1:6]

## [1]  0  2  4 11  0  2

class(bacteria$week[1:6])

## [1] "integer"
```

- (un/ordered) factor

```
chickwts$feed[1:6]

## [1] horsebean horsebean horsebean horsebean horsebean horsebean
## Levels: casein horsebean linseed meatmeal soybean sunflower

levels(chickwts$feed)[1:3]

## [1] "casein"    "horsebean" "linseed"
```

# Data types in R: Ordered Factors

Ordinal variables are represented as ordered factors:

```
bac_growth <- c("none", "+", "++", "+", "+++", "+", "none") # vector
bac_growth <- factor(bac_growth, levels = c("none", "+", "++", "+++"),
                     order = TRUE)
bac_growth

## [1] none +    ++   +    +++  +    none
## Levels: none < + < ++ < +++

#
mood <- c("OK", "Well", "Super", "Super", "Don't ask", "OK") # vector
mood <- factor(mood, levels = c("Don't ask", "Well", "OK", "Super"),
               order = TRUE)
mood

## [1] OK        Well      Super     Super     Don't ask OK
## Levels: Don't ask < Well < OK < Super
```

# Exercise 5

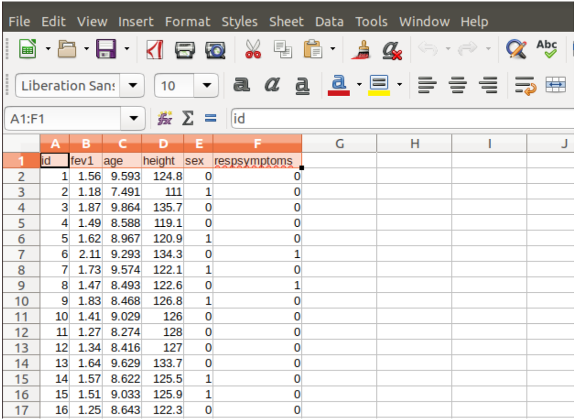
# Exercise 6

# Examples of different data types

- numeric variable
- integer variable
- variable with two levels (binary factor)
- ordered variable with **more than** two levels (ordinal)
- unordered variable with **more than** two levels (nominal)

# Rules for importing data into R (from Excel)

- First row of excel sheet contains **variable names**: `y, ap, hilo, week, ID, trt`.
- Columns of excel sheet represent **variables**.
- Rows of excel sheet represent **observations per individual** (except for the first row).




| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | id | fev1 | age | height | sex | respsymptoms |
| 2 | 1 | 1.56 | 9.593 | 124.8 | 0 | 0 |
| 3 | 2 | 1.18 | 7.491 | 111 | 1 | 0 |
| 4 | 3 | 1.87 | 9.864 | 135.7 | 0 | 0 |
| 5 | 4 | 1.49 | 8.588 | 119.1 | 0 | 0 |
| 6 | 5 | 1.62 | 8.967 | 120.9 | 1 | 0 |
| 7 | 6 | 2.11 | 9.293 | 134.3 | 0 | 1 |
| 8 | 7 | 1.73 | 9.574 | 122.1 | 1 | 0 |
| 9 | 8 | 1.47 | 8.493 | 122.6 | 0 | 1 |
| 10 | 9 | 1.83 | 8.468 | 126.8 | 1 | 0 |
| 11 | 10 | 1.41 | 9.029 | 126 | 0 | 0 |
| 12 | 11 | 1.27 | 8.274 | 128 | 0 | 0 |
| 13 | 12 | 1.34 | 8.416 | 127 | 0 | 0 |
| 14 | 13 | 1.64 | 9.629 | 133.7 | 0 | 0 |
| 15 | 14 | 1.57 | 8.622 | 125.5 | 1 | 0 |
| 16 | 15 | 1.51 | 9.033 | 125.9 | 1 | 0 |
| 17 | 16 | 1.25 | 8.643 | 122.3 | 0 | 0 |

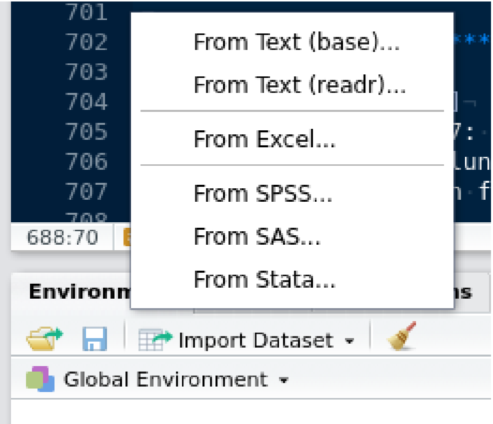
# Rules for naming variables

Variable names should ...

- start with a letter (not a number):
  `y, ap, hilo, week, ID, trt`
- longer variables names should be separated with dots:
  `time.in.weeks`
- do not use special characters, such as /, #, @, &, $\star$, ...

# How to import external data files into R?

- > Import Dataset > **From Text (base)...** > CSV Files (.csv) or
- > Import Dataset > **From Excel...**



## How to import external data files into R?

- Environment (upper right corner)
- > Import Dataset > **From Text (base)...** > CSV Files (.csv)

```
perulung_ems <- read.csv("perulung_ems.csv", row.names = 1,
                          sep = ";")
lung <- data.frame(perulung_ems)
```

- > Import Dataset > **From Text (base)...** > Text Files (.txt)
- > Import Dataset > **From Excel...** > Excel Files (.xlsx)

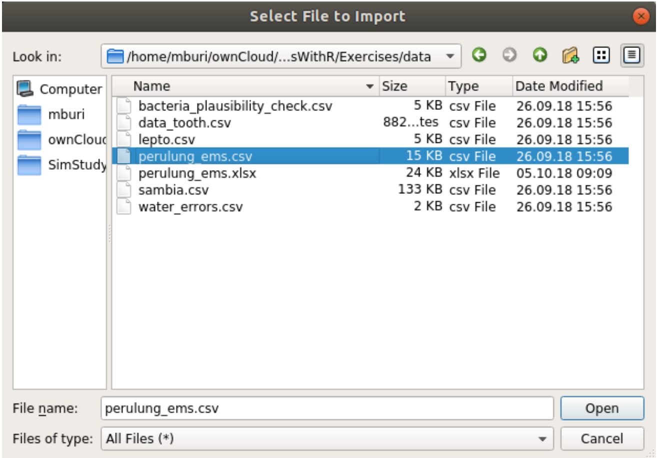
```
install.packages("readxl")
library("readxl")
perulung_ems <- read_excel("perulung_ems.xlsx")
lung <- data.frame(perulung_ems)
head(lung)
```

# How to import .txt and .csv files into R? (1/2)

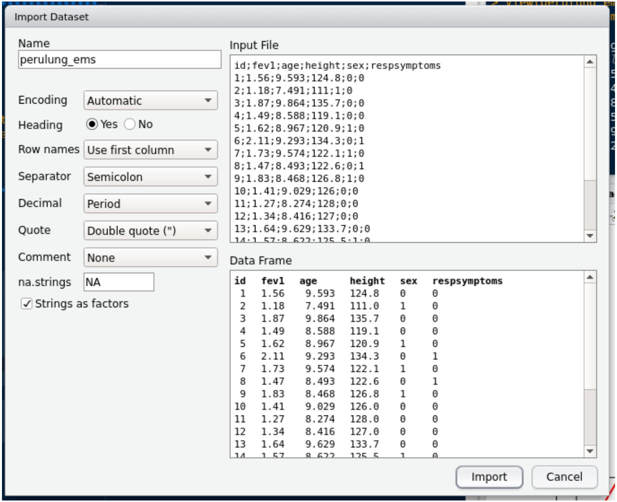
- Environment (upper right corner)
- $>$ Import Dataset $>$ From Text (base)... $>$ CSV Files (.csv)

# How to import .txt and .csv files into R? (1/2)

- Environment (upper right corner)
- > Import Dataset > From Text (base)... > CSV Files (.csv)



# How to import .txt and .csv files into R? (2/2)



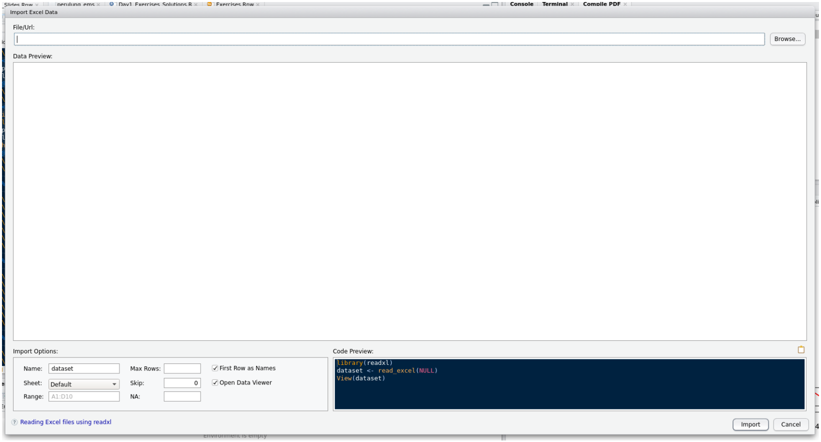

```r
perulung_ems <- read.csv("perulung_ems.csv", row.names = 1,
                         sep = ";")
lung <- data.frame(perulung_ems)
```

## How to import .xlsx files into R? (1/3)

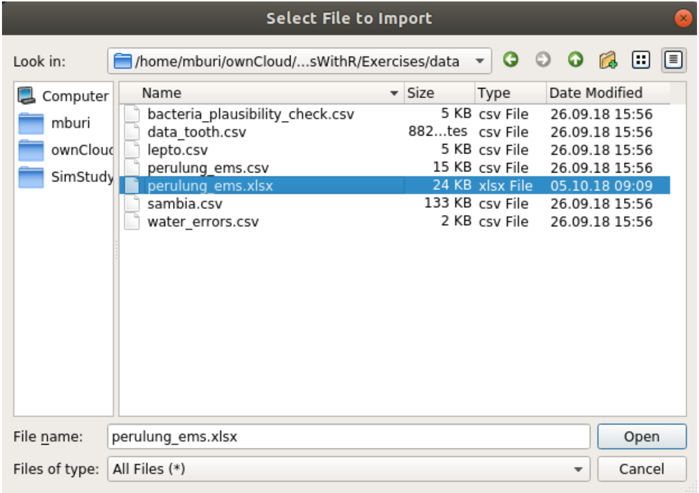
- Environment (upper right corner)
- $>$ Import Dataset $>$ From Excel... $>$ Excel Files (.xlsx)

# How to import .xlsx files into R? (1/3)

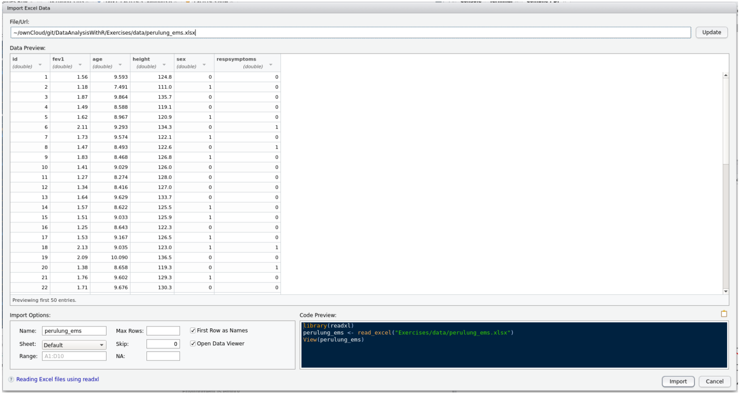
- Environment (upper right corner)
- > Import Dataset > From Excel... > Excel Files (.xlsx)



# How to import .xlsx files into R? (2/3)



# How to import .xlsx files into R? (3/3)




```r
perulung_ems <- read_excel("perulung_ems.xlsx")
lung <- data.frame(perulung_ems)
head(lung)
```

# Exercise 7: perulung

Data from a study of lung function among children living in a deprived suburb of Lima, Peru. Data taken from Kirkwood and Sterne, 2nd edition.
Variables:

- `fev1`: in liter, ”forced expiratory volume in 1 second” measured by a spirometer. This is the maximum volume of air which the children could breath out in 1 second
- `age`: in years
- `height`: in cm
- `sex`: 0 = girl, 1 = boy
- `respsymp`: respiratory symptoms experienced by the child over the previous 12 months

# Lecture Slides for Day 3

## Why do we need Statistics?

**Repeatability of results:**
**Statistical science** allows us to estimate what might happen if an experiment was repeated - but without having to actually repeat it!

# Why do we need Statistics?

- Study results must be shown to be robust, i.e. real and not due to random chance
- Best way to demonstrate this is to repeat the same experiment/study many times each with **different** subjects (animals) drawn from the **same study population** and show that the result is truly repeatable
- It is generally totally impractical, in terms of both time and resources, to repeat an experiment many times!

# Why do we need Statistics?

- Instead of repeating the experiment many times **probability theory i.e. statistics** is used to **estimate** what might have happened if the experiment had been repeated
- A mathematical model is used to fill this “data gap”
- Generally the most difficult task in statistics is to decide what “model” is most appropriate for a given experiment

# What is Statistics? - A definition

A set of analytical tools designed to quantify uncertainty

- If an experiment or procedure is repeated, how likely is it that the new results will be similar to those already observed?
- What is the likely variation in results if the experiment was repeated?

# What is Statistics? - A definition

The key scientific purpose of statistics

- to provide **evidence** of the existence of some “effect” of scientific interest
- i.e. evidence based medicine

# As a reminder: The importance of study design

Even the most sophisticated statistical analyses cannot rescue a poorly designed study

$\rightarrow$ unreliable results

$\rightarrow$ inability to answer the main research question

# Putting Statistics in Context

- The vast majority of analyses can be done in a straightforward fashion - just remember and always use common sense as a guide - be skeptical!
- It is very easy to get “lost” in the statistical software and technical jargon, which differs markedly between different software packages. Terminology can also differ greatly between textbooks...
- Wikipedia is as good a resource as any for finding out about different statistical tests and terminology

# Exploratory Data Analysis

- It is crucially important to explore your data fully before considering any “formal” statistical analyses

# Exploratory Data Analysis

- It is crucially important to explore your data fully before considering any “formal” statistical analyses
- What explorations are done depends on the objective of the study - the research question(s)

# Exploratory Data Analysis

- It is crucially important to explore your data fully before considering any “formal” statistical analyses
- What explorations are done depends on the objective of the study - the research question(s)
- Helps to decide what kind of formal statistical analyses might be most appropriate for the data available

# Exploratory Data Analysis

- It is crucially important to explore your data fully before considering any “formal” statistical analyses
- What explorations are done depends on the objective of the study - the research question(s)
- Helps to decide what kind of formal statistical analyses might be most appropriate for the data available
- What a simple descriptive analysis **does not** provide is evidence of whether the observed treatment effects are large enough to be notable once sampling variation has been accounted - that is the role of formal analyses, e.g. hypothesis testing

# Summary Statistics
## Continuous (Integers / Numeric)

- `Mean` - a measure of location. Always examine the average value of the response variable(s) for the different "treatment" effects in your data
- `Median` - a robust single value summary of a set of data (50% quantile point) - most useful in highly skewed data or data with outliers
- `Standard deviation (sd)` - a measure of spread, how variable the data are
- `Standard error of the mean (se)` - an estimate of how far the sample mean is likely to be from the population mean
- and others: `min, max, range, IQR, ...`

# Continuous (Integers / Numeric) Summary Statistics

```r
mean(x) # mean

median(x) # median

sd(x) # standard deviation

min(x) # minimum

max(x) # maximum

range(x) # range

IQR(x) # interquartile range
```

# Continuous Data Summaries

**standard deviation**

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

**standard error**

$$se = \frac{s}{\sqrt{n}}$$

# Correlation coefficient
## Combination of continuous and continuous

Correlation coefficient a measure association between two continuous variables (common but somewhat limited)

**Pearson's correlation coefficient r**

$$r=\frac{\sum_{i=1}^{n}(X_i-\bar{X})(Y_i-\bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i-\bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i-\bar{Y})^2}}$$

$\bar{X}$: mean of variable x

$\bar{Y}$: mean of variable y

# Correlation of continuous and ordinal variables

```r
# Test for Association/Correlation Between
# Paired Samples
cor.test(data$x, data$y, method = "pearson")
cor.test(data$x, data$y, method = "spearman")

# Scatterplot(s)
pairs(data$x ~ data$y)
pairs(data)
```

# Summary Statistics
## Continuous and ordinal variables

```r
tapply(data$x.cont, data$y.fac, mean)

tapply(data$x.cont, data$y.fac, median)

tapply(data$x.cont, data$y.fac, sd)
```

# Summary Statistics
## Ordinal

- `Median` - a robust single value summary of a set of data (50% quantile point) - most useful in highly skewed data or data with outliers

- `e.g.10th and 90th percentile` - a measure of spread, how variable the data are

```
quantile(x, probs = c(0.1, 0.9))
```

- `proportions` - e.g. percentage per grade

```
prop.table(table(data$x.fac))
prop.table(table(data$x.fac, data$y.fac))
```

# Summary Statistics
## Nominal

- `proportions` - percentage within the different categories
- `contingency tables` e. g. 2 x 2

```
table(data$x.fac)
table(data$x.fac, data$y.fac)
prop.table(table(data$x.fac))
prop.table(table(data$x.fac, data$y.fac))
```

# Exercise 8

# How to deal with missing values in R? (1/3)

- In R, missing values are represented by the symbol **NA** (not available).
- Impossible values (e. g., dividing by zero) are represented by the symbol **NaN** (not a number).
- Ask yourself why a **NA** and / or **NaN** occurs!

# How to deal with missing values in R? (2/3)

- Testing for Missing Values

```
vec1 <- c(1, 2, 3, NA)
is.na(vec1) # returns a vector (FALSE, FALSE, FALSE, TRUE)
# The TRUE indicates the position of the NA in vec1.
```

- Recoding Values to Missing

```
# recode specific values (e. g. 0.001) to missing for variable x
# select rows where x is 0.001 and recode value in column x with NA
dat$x[dat$x == 0.001] <- NA
```

# How to deal with missing values in R? (3/3)

- Excluding Missing Values from specific function calls

```
a <- c(1, 2, NA, 3)
mean(a) # returns NA
mean(a, na.rm=TRUE) # returns 2
```

- Check for complete cases with function `complete.cases(...)`

```
# list rows of data that have missing values
dat[!complete.cases(dat),]
subdat <- dat[complete.cases(dat),]
```

- Create new dataset without missing data with function `na.omit(...)`

```
new.dat <- na.omit(dat)
```

# How to check your data for plausibility?

- Ask yourself what can go wrong?
- Implausible values?
- Impossible values?
- Logical errors?

# Exercise 9A: Plausibility Checks

# Exercise 9B: Missing Values

# Exercise 10