Introduction to R

Statistical Consulting Group

Seminar for Statistics, ETH Zürich

Consulting in Statistics and Statistics Software

Seminar for Statistics:
Statistical Consulting Group
HG G13, Tel. 044 632 22 23
E-mail: beratung@stat.math.ethz.ch

Further information:
https://www.math.ethz.ch/sfs

Scripts, Slides, Exercises and Datasets in
ftp://stat.ethz.ch/U/sfs/RKurs/R.Intro/

Table of Contents

1. R Basics
   - R and RStudio
   - Import Data
   - R Objects and Indexing
   - Function Calls
   - Missing values
2. Graphics
3. Hypothesis Tests
4. Linear Regression
   - Correlation
   - Linear Regression
5. Going further

R Basics

In this section we will have a look at . . .

. . . R & RStudio

. . . some shortcuts

. . . basic syntax

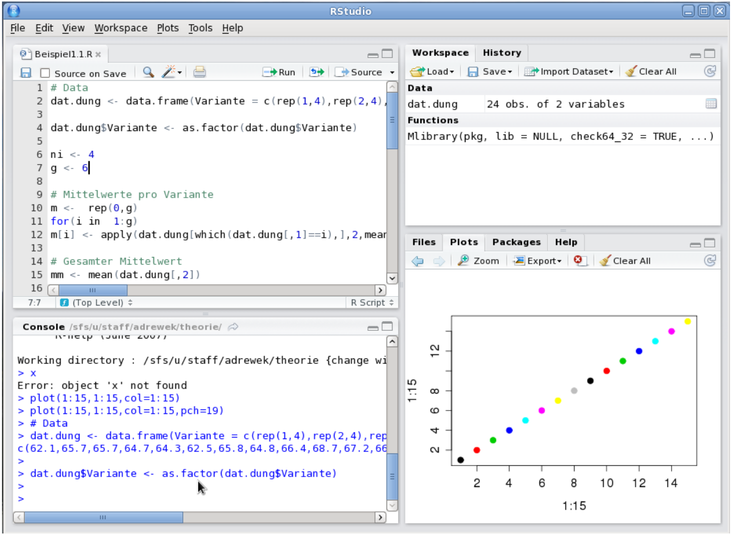
. . . where to find help

What is R?

R is...

- **free**
- **open-source**
- **flexible**
- **expandable** with thousands of **add-on packages**: http://cran.r-project.org/web/views/
- **widely used** both in academia and industry
- teaser: http://shiny.rstudio.com/gallery/

RStudio: Integrated Development Environment (IDE) for R



```
# Data
dat.dung <- data.frame(Variante = c(rep(1,4),rep(2,4),

dat.dung$Variante <- as.factor(dat.dung$Variante)

ni <- 4
g <- 6

# Mittelwerte pro Variante
m <-  rep(0,g)
for(i in  1:g)
m[i] <- apply(dat.dung[which(dat.dung[,1]==i),],2,mear

# Gesamter Mittelwert
mm <- mean(dat.dung[,2])
```

Console /sfs/u/staff/adrewek/theorie/

```
  R-help (June 2007)

Working directory : /sfs/u/staff/adrewek/theorie {change wi
> x
Error: object 'x' not found
> plot(1:15,1:15,col=1:15)
> plot(1:15,1:15,col=1:15,pch=19)
> # Data
> dat.dung <- data.frame(Variante = c(rep(1,4),rep(2,4),rep
c(62.1,65.7,65.7,64.7,64.3,62.5,65.8,64.8,66.4,68.7,67.2,66
>
> dat.dung$Variante <- as.factor(dat.dung$Variante)
>
>
```

Data

dat.dung 24 obs. of 2 variables

Functions

Mlibrary(pkg, lib = NULL, check64_32 = TRUE, ...)

R and RStudio: the basics

- Write your code in a script file inside the source code editor
- Execute R code by sending lines of code from editor to R console
- Add comments using the hashtag # (enables automatic sectioning of code in RStudio)
  - No lines of code are lost when R is shut down or crashes
  - Assures reproducible code/coding
  - Makes it easy to share your code with colleagues or reviewers

RStudio

- Send Code to Console: `Ctrl+Enter` (Mac: `Cmd+Enter`) or Menu Button `Run`
- Code Completion: RStudio supports the automatic completion of code using the `Tab` key.
- Previous Commands: recall previous commands using the arrow key `Up` or `Ctrl+Up`
- Console Title Bar: current working directory and key to interrupt R during a long computation
- Clear Console with `Ctrl+L`
- Assignments (<-) with `Alt+Minus`

R and RStudio: Basics

R as a calculator

```
> sqrt(2)

[1] 1.4142

> x <- 3
> y <- x^2
> x + y

[1] 12

> sin(2 * pi)

[1] -2.4493e-16
```

R and RStudio: Basics

Creating vectors

```
> c(1, 5, 80)

[1]  1  5 80

> 2:11

 [1]  2  3  4  5  6  7  8  9 10 11

> a <- c(1, 6, 10, 22, 7, 13)
> mean(a)

[1] 9.8333

> sum(a)

[1] 59
```

R and RStudio: Basics

Creating matrices and data frames

```
> matrix()

     [,1]
[1,]   NA

> m <- matrix(1:6, nrow=3, ncol=2, byrow = TRUE)
> m

     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6

> data.frame()

data frame with 0 columns and 0 rows

> df <- data.frame(Name = c("Ich","Du","Er"),
+         Geschlecht = as.factor(c(0,1,0)), Alter = c(21,47,33))
> df

  Name Geschlecht Alter
1  Ich          0    21
2   Du          1    47
3   Er          0    33
```

R and RStudio: details

In the R console, you see the prompt: >

You type a command, get a result and a new prompt.

```
> 3 + 4
```

```
[1] 7
```

An incomplete statement can be continued on the next line

```
> 3 * (4 +
+       2)
```

```
[1] 18
```

An incomplete statement is indicated by the prompt: +

- Check that prompt is > after error message before further code execution

R and RStudio: details

An R statement may consist of

- an assignment:

  ```
  > t.a <- 3 * (4 + 2)
  > t.b <- t.a + 2.5
  ```

  stores the result of the calculation under `t.a` resp. `t.b`.

- a name of an object: displays object

  ```
  > t.a
  
  [1] 18
  ```

- a call to a function: numerical or graphical result

  ```
  > mean(c(t.a,t.b))
  
  [1] 19.25
  
  > mn <- mean(c(t.a,t.b))
  ```

  A function is called by its name followed by `()`.

Assignment and Workspace

- Everything in R is an object and has a certain name like
  `t.a, mean, mn`.
- R stores objects in your workspace

```
> t.a <- 3 * (4 + 2)
```

- ATTENTION: Overwriting an object in R throughs no warning

```
> t.a <- t.b^2
> t.a

[1] 420.25
```

Where to Find Help?

- Help about any function `?foo`
- If you have any question, google ‘how do I . . . with R‘.
  - huge community
  - already asked by somebody else
- Very useful and helpful Q&A website: http://stackexchange.com/
- Cheat Sheet for Base R
  https://www.rstudio.com/resources/cheatsheets/
- R Reference Card
  https://cran.r-project.org/doc/contrib/Baggott-refcard-v2.pdf
- Learning by doing is particularly true for programming

Import Data Sets

In this section we will learn how to ...

...import data sets

...change the working directory

Import Data: Website

We have a look at a data set with results of several disciplines.

Importing data from a website

```
> url <- "http://stat.ethz.ch/Teaching/Datasets/WBL/sport.dat"
> d.sport <- read.table(url, header = TRUE)
> head(d.sport)

           weit kugel hoch  disc stab speer punkte
OBRIEN     7.57 15.66  207 48.78  500 66.90   8824
BUSEMANN   8.07 13.60  204 45.04  480 66.86   8706
DVORAK     7.60 15.82  198 46.28  470 70.16   8664
FRITZ      7.77 15.31  204 49.84  510 65.70   8644
HAMALAINEN 7.48 16.32  198 49.62  500 57.66   8613
NOOL       7.88 14.01  201 42.98  540 65.48   8543
```

Import Data: Working Directory

Let's set the working directory - the folder from which to operate (e.g., save and load from). Use:

```
> getwd()  ## print the current working directory
> setwd("/userdata/Documents/Rcourse/")
```

Or alternatively in RStudio use 'Session' → 'Set Working Directory' → 'Choose Directory...'

Import Data: File

Importing data in R is easy

- Different ways depending on the format (csv, txt, xlsx, etc.).
- **Alternative**: use the ‘Import Dataset‘ tool in RStudio (upper-right panel)

```
> d.sport <- read.table(file = "sport.dat", header = TRUE)
> head(d.sport)
```

Save Data or Write Data to a File

To save or write data to a file:

- Text-files:

  ```
  > write.table(x, file = "xy.txt", sep = " ")
  ```

  where `x` is the data object to be stored and `xy.txt`

- Excel-files: Use CSV

  ```
  > write.csv(...)
  > write.csv2(...)
  ```

R Objects & Indexing

In this section we will have a look at . . .

. . . R objects

. . . how indexing works

Introductory Example: R Objects

R Objects:

- data frame: most essential data structure in R

```
> str(d.sport)
```

```
'data.frame':        15 obs. of  7 variables:
 $ weit  : num  7.57 8.07 7.6 7.77 7.48 7.88 7.64 7.61 7.27 7.49 ...
 $ kugel : num  15.7 13.6 15.8 15.3 16.3 ...
 $ hoch  : int  207 204 198 204 198 201 195 213 207 204 ...
 $ disc  : num  48.8 45 46.3 49.8 49.6 ...
 $ stab  : int  500 480 470 510 500 540 540 520 470 470 ...
 $ speer : num  66.9 66.9 70.2 65.7 57.7 ...
 $ punkte: int  8824 8706 8664 8644 8613 8543 8422 8318 8307 8300 ...
```

- vector, e.g. a column of the data set `d.sport`

```
> kugel <- d.sport$kugel
> str(kugel)
```

```
 num [1:15] 15.7 13.6 15.8 15.3 16.3 ...
```

```
> participant <- rownames(d.sport)
> str(participant)
```

```
 chr [1:15] "OBRIEN" "BUSEMANN" "DVORAK" "FRITZ" "HAMALAINEN" "NOOL" ...
```

Introductory Example: Select Elements I

Select elements:

```
> participant[4]

[1] "FRITZ"

> d.sport[c(3, 6, 4), c(1:3, 7)]

       weit kugel hoch punkte
DVORAK 7.60 15.82  198   8664
NOOL   7.88 14.01  201   8543
FRITZ  7.77 15.31  204   8644

> d.sport["FRITZ", ]

      weit kugel hoch  disc stab speer punkte
FRITZ 7.77 15.31  204 49.84  510  65.7   8644
```

Accessing Parts of an Object

Goal is to look at or use a part of your object

To access only part of an object, use `[ ]`:

- for vectors: `myvector[x]`
- for two-dimensional objects, e.g. data frames or matrices: `mydata.frame[x, y]`

Specify the indices by a vector (e.g. `c(1, 2, 6)`) and separate the indices of different dimensions by commas

Introductory Example: Select Elements II

Let's play around with the indexing of a data frame: two-dimensional object!

```
> d.sport[ , ]
> c(1, 3, 7)
> 1:10
> d.sport[1:10 , ]
> d.sport[-c( 1, 3, 7), ] # negative indices are excluded
> d.sport[ , 2:3]
> d.sport[c(1, 3, 6), 2:3]
```

Function Calls

In this section we will have a look at . . .

. . . basic function calls

. . . mandatory and optional arguments

. . . R packages

Introductory Example: Functions

Example function calls:

```
> mean(d.sport$kugel)
```

```
[1] 15.199
```

```
> quantile(d.sport$kugel)
```

```
   0%   25%   50%   75%  100% 
13.53 14.60 15.31 15.74 16.97 
```

```
> quantile(d.sport$kugel, probs = c(0.75, 0.9))
```

```
   75%    90% 
15.740 16.674 
```

Check out the function's help file:

```
> ?mean
```

Function Call

- Functions consist of mandatory and optional arguments:

```
mean(x, trim = 0, na.rm = FALSE, ...)
```

`x`: mandatory argument
`trim`: optional argument, default is 0
`na.rm`: optional argument, default is FALSE

Function Call

The arguments of a function have a defined order and each argument has its own unique name.

```
> mean(x = d.sport$kugel, na.rm = TRUE)
```

You can either use the names of the arguments, or place the values in the correct order (or a mix of both):

```
> mean(d.sport$kugel, ,TRUE)
```

Example functions with no mandatory arguments: matrix(), vector(), array(), list()

```
> ?matrix
```

Useful Functions

Useful functions (look for help by typing `?str`):

- `str()`
- `nrow()` and `ncol()`
- `dim()`
- `summary()`
- `apply()`
- `head()` and `tail()`

see also R Reference card

Packages

By default, R only provides a basic set of functions. Additional functions (and datasets) are obtained by loading additional **add-on packages**:

- Install and load:

```
> install.packages("MASS")  # install onto computer once
> require(MASS) # for every R session. Alternatively:
> library(MASS)
```

- Online resources:
  - list of all packages: `http://cran.r-project.org/web/packages/`
  - by topic: `http://cran.r-project.org/web/views/`
  - ask Google

Missing values

In this section we will have a look at . . .

. . . missing values

Missing Values

Very common with real data. Let's fake the situation for the d.sport data.

```
> d.sport.NA <- d.sport
> d.sport.NA[2, "kugel"] <- NA
> d.sport.NA[3, "hoch"] <- -999
```

In R, missing values are coded as NA (not available) and are treated in a special way, e.g. is.na():

```
> is.na(d.sport.NA) # one logical value per element

> sum(is.na(d.sport.NA)) # adds up the TRUE elements

[1] 1

> which(is.na(d.sport.NA), arr.ind = TRUE) # where are the NA's?

         row col
BUSEMANN   2   2
```

Missing Values

Specify missing values while reading in the data, or afterwards:

```
> d.sport.NA[d.sport.NA == -999] <- NA
> ## alternatively, while reading in the data:
> d.sport <- read.csv("sport.dat", header = TRUE, na.strings = c("NA", "-999"))
```

Many functions have an argument to handle missing values, e.g. `na.rm`, `na.omit`:

```
> sum(d.sport.NA$kugel)
> sum(d.sport.NA$kugel, na.rm = TRUE)
```

Missing Values

Drop observations (rows) that contain an NA:

```
> na.omit(d.sport.NA)
```

Basic Graphics

In this section we will have a look at . . .

. . . basic plot functions

. . . alternative R packages for graphics

Iris Data

R comes with built-in data sets, e.g. `iris`:

```
> data(iris) # now it shows up in the RStudio environment
> iris
```

It has the following structure

```
> str(iris)
```

```
'data.frame':        150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1
```

A `factor` is another class of values in R, representing categorical values with different “levels”. Internally it is coded with numbers (that are used to build dummy-variables in statistical models).

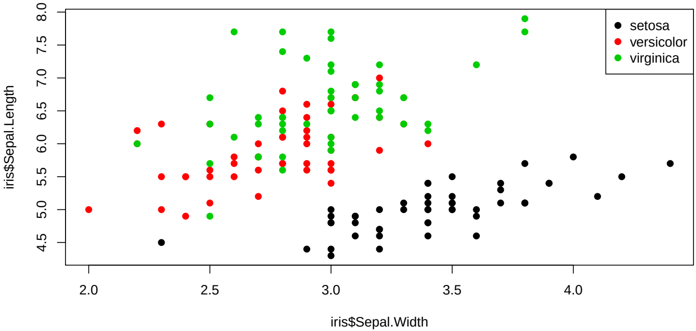
The `plot` Function

The `plot` function has only one mandatory argument which is `x`.

- second most important argument: `y`
- many optional arguments (`col`, `pch`, `main`, `cex`,...)
- check `?plot` and google!
- use function `par` (`?par`) to set or query graphical parameters

The plot Function

```
> # High-level plotting function: opens a plot
> plot(x = iris$Sepal.Width, y = iris$Sepal.Length,
+      col = iris[ , "Species"], pch = 19)
> # Low-level plotting functions: add to an existing plot
> legend("topright", legend = levels(iris[ ,"Species"]),
+        pch = 19, col = 1:3) # add the legend
```



The plot Function

Let's look at the previous function call step-by-step:

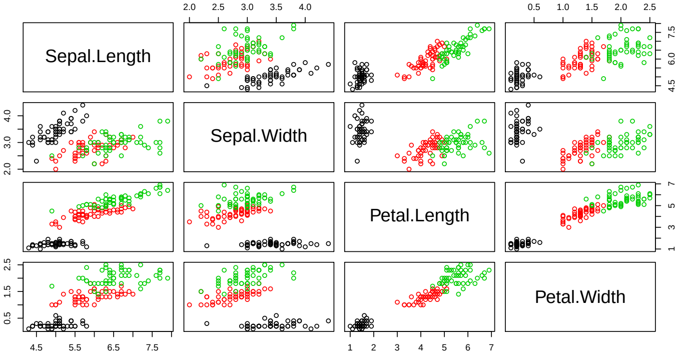
- execute only parts
- omit arguments one-by-one

Alternatively one could use formula notation:

```
> # same plot as on the previous slide without a legend
> plot(Sepal.Length ~ Sepal.Width,
+      col = Species, pch = 19, data = iris)
```

Which plot did we look at just before?

```
> pairs(iris[ , -5], col = iris[ , 5])
```



Arguments of `plot`

Statement	Meaning
`type`	Style of drawing (single points, lines etc.)
`log`	logarithmic scale
`xlim`	range of *x*-coordinates
`ylim`	range of *y*-coordinates
`pch`	Plotting character
`col`	Coloring points
`lty`	line type
`lwd`	line width
`main`	main title (appears above the plot)
`xlab`	label of *x*-axis
`ylab`	label of *y*-axis

The three categories of `graphics` functions

Three categories of *R graphics functions*:

- High-level plotting functions such as `plot()`
  to generate a new graphical display.
- Low-level plotting functions such as `legend()`
  to add further graphical elements to an existing plot.
- Interactive functions such as `identify()`
  to amend or collect information interactively from a plot.

Low-level Plotting Functions

Statement	Meaning
`points(x, y, pch = 1)`	Draws points pictured as `pch`.
`text(x, y, text)`	Writes `text` at coordinate (x, y).
`lines(x, y, lty = 1)`	Adds a line to graph.
`abline(a, b)`	Adds a line with intercept `a` and slope `b`.
`abline(h = y, v = x)`	Horizontal and vertical lines.
`legend(x, y, text, lty, pch)`	Creates a legend.

Useful Plot Functions

Useful functions (look for help by typing `?foo`):

- `plot`, `pairs`, `interaction.plot`
- `boxplot`, `hist`
- `plot3d`

Graphical Output

It is easy to export R graphics. Remember to specify your working directory!

```
> pdf(file = "iris_plot.pdf") #  open the graphics device
> plot(Sepal.Length ~ Sepal.Width, data = iris)
> ## add anything else you want in your plots, and when you are done, use:
> dev.off() # close the graphics device
```

Several plots in one graphical window

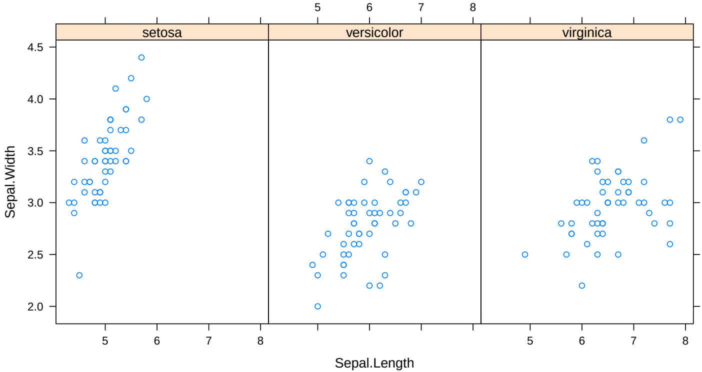
```
> par(mfrow=c(3, 2))
```

splits the graphical window into 3 rows and 2 columns.

Newer Graphics: `lattice`

`lattice` package functions: excel at repeating graphs for various groups

```
> library(lattice)
> xyplot(Sepal.Width ~ Sepal.Length | Species, data = iris)
```

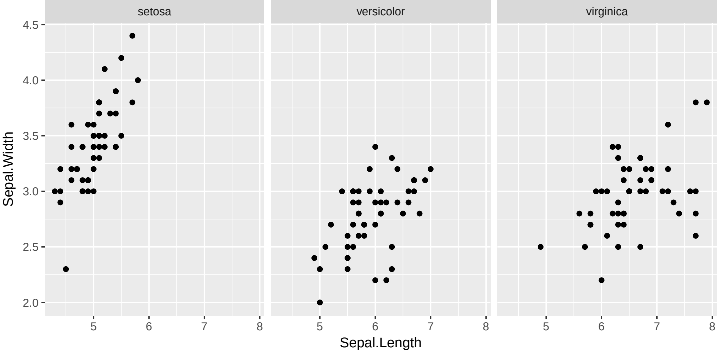


See `http://www.statmethods.net/advgraphs/trellis.html` for more information.

Newer Graphics: ggplot2

ggplot2 package: very flexible, based on grammar of graphics

```
> library(ggplot2)
> ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width)) +
+   facet_grid(facets = ~ Species) + geom_point()
```



More information can be found on `http://ggplot2.org`.

Hypothesis Tests

In this section we will have a look at . . .

. . . t-Test and Wilcoxon rank-sum test

. . . Chi-square test

Hypothesis Tests - Approach

Hypothesis testing in 6 steps:

1. Declare model by which data were generate (e.g. population is normally distributed, large sample size and σ not known)
2. Define **null hypothesis** H_0***** and **alternative hypothesis** H_A******
   - **The statement being tested in a test of [statistical] significance. Test the strength of the evidence against the null hypothesis.*
   - ** *The statement that is hoped or expected to be true instead of the null hypothesis.*
3. Choose the level of significance α
4. Determine critical values for the level of significance α and degrees of freedom $df = (n - 1)$
5. Define and calculate test statistic, e.g. one-sample test, $t = \frac{\bar{x}-\mu_0}{s_x/\sqrt{n}}$
6. Compare the test statistic to the critical values and make decision: **reject or fail to reject** H_0

Hypothesis Tests - an Example

Is the sepal length of versicolor different to that of virginica? Let's use a t-Test and a Wilcoxon rank-sum test.

We first subset the iris data set:

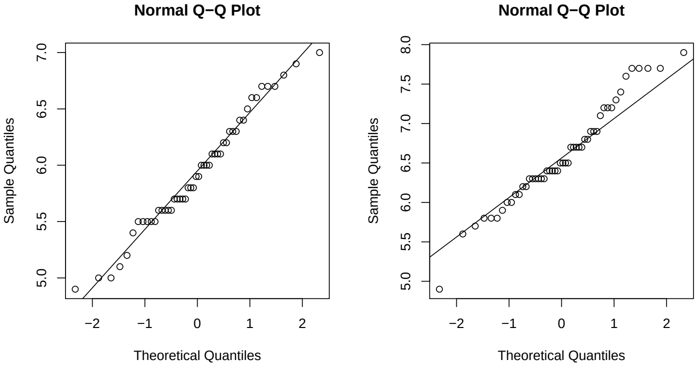
```
> testdata <- iris[iris$Species != "setosa", c("Sepal.Length", "Species")]
> testdata$Species <- droplevels(testdata$Species)
> str(testdata) ## prepare and check data
```

```
'data.frame':        100 obs. of  2 variables:
 $ Sepal.Length: num  7 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 ...
 $ Species     : Factor w/ 2 levels "versicolor","virginica": 1 1 1 1 1 1 1 1 1 1
```

Hypothesis Tests - an Example

Check normality assumption of t-Test using QQ-Plot:

```
> versi.id <- testdata$Species == "versicolor"
> par(mfrow=c(1,2))
> qqnorm(testdata$Sepal.Length[versi.id]); qqline(testdata$Sepal.Length[versi.id])
> qqnorm(testdata$Sepal.Length[!versi.id]); qqline(testdata$Sepal.Length[!versi.id
```



Hypothesis Tests - an Example

Two sample t-test:

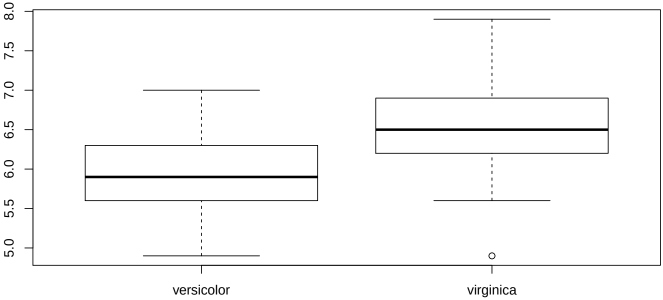
```
> versi.id <- testdata$Species == "versicolor"
> t.test(x = testdata$Sepal.Length[versi.id],
+        y = testdata$Sepal.Length[!versi.id],
+        var.equal = TRUE )

        Two Sample t-test

data:  testdata$Sepal.Length[versi.id] and testdata$Sepal.Length[!versi.id]
t = -5.63, df = 98, p-value = 1.7e-07
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.88185 -0.42215
sample estimates:
mean of x mean of y
    5.936     6.588
```

t-test rejects the null hypothesis at 5% significance level. Don't forget to visually check the normality assumptions (QQ-Plot).

Hypothesis Tests - an Example

Check assumption of Wilcoxon Rank Test:
same distribution, just a location shift

```
> boxplot(testdata$Sepal.Length ~ testdata$Species)
```



Hypothesis Tests - an Example

Now, perform the test

```
> versi.id <- testdata$Species == "versicolor"
> wilcox.test(x = testdata$Sepal.Length[versi.id],
+             y = testdata$Sepal.Length[!versi.id])

        Wilcoxon rank sum test with continuity correction

data:  testdata$Sepal.Length[versi.id] and testdata$Sepal.Length[!versi.id]
W = 526, p-value = 5.9e-07
alternative hypothesis: true location shift is not equal to 0
```

Wilcoxon Rank Test also rejects the null hypothesis at 5% significance level. Wilcoxon Rank Test is the prefered test for a two-samples statistical test!

Hypothesis Tests

How to proceed:

- Formulate null and alternative hypotheses
- Choose appropriate test
- Collect data, i.e. do an experiment
- Look at data:
  `plot(), pairs(), hist(), boxplot()`
- Validate assumptions for test
- Carry out the test and interpret result

Hypothesis Tests

	1 sample / 2 dep. samples	2 indep. samples
parametric	t-Test → normality	t-Test → normality (& equal variance)
non-param.	Wilcoxon Test (signed rank) → symmetric distribution	Wilcoxon Test (rank sum) → location shift

Hypothesis Tests - Chi-Squared Test of Independence

Hypothesis:
H_0 : Independence of education and marriage status
H_A : Dependence of education and marriage status

Example:

Education	Married once	Married > 1	Total
College	550	61	611
No College	681	144	825
Total	1231	205	1436

Chi-Squared Test of Independence

```
> url <- "http://stat.ethz.ch/Teaching/Datasets/edu.txt"
> d.edu <- read.table(url, header = TRUE)
```

Cross-tables in R:

- Count number of cases with same value:

```
> table(d.edu[, "Married"])

Married more Married once
         205         1231
```

- Cross-table

```
> table(d.edu[, "Education"], d.edu[, "Married"])

             Married more Married once
  College              61          550
  No College          144          681
```

Chi-Squared Test of Independence

Now we perform the Chi-squared test:

```
> chisq.test(d.edu[, "Education"], d.edu[, "Married"])

	Pearson's Chi-squared test with Yates' continuity correction

data:  d.edu[, "Education"] and d.edu[, "Married"]
X-squared = 15.4, df = 1, p-value = 8.7e-05
```

Result: Reject H_0, i.e. education and marriage are dependent.

Correlation

In this section we will have a look at . . .

. . . correlation

. . . why you should plot your data

Correlation

Load a new data set:

```
> url <- "http://stat.ethz.ch/Teaching/Datasets/basischOhneNA.dat"
> d.basisch <- read.table(url, header = TRUE)
```

Look at the structure of the object:

```
> str(d.basisch)
```

```
'data.frame':        123 obs. of  4 variables:
 $ ph    : num  7.33 7.69 7.9 8.14 7.62 ...
 $ l.sar : num  0.0969 0.4393 1 1.316 0.0607 ...
 $ height: num  5.91 5.2 4.4 4.5 6.05 6 5.35 5.55 4.95 5.2 ...
 $ h.quad: num  34.9 27 19.4 20.2 36.6 ...
```

- ph: pH value of soil around the tree
- l.sar: log(sodium absorption ratio)
- height: height of tree
- h.quad: square of height of tree

Correlation

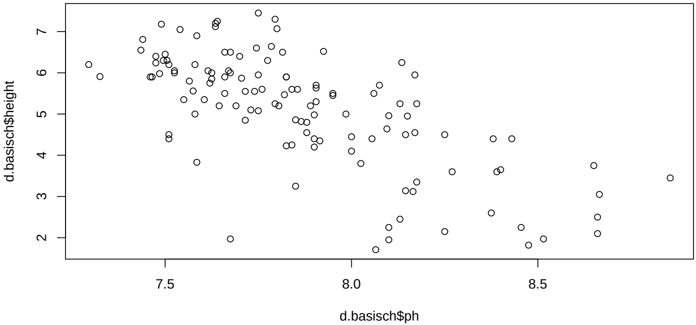
Calculate the (Pearson) correlation of `ph` and `height`

```
> cor(d.basisch$ph, d.basisch$height)
```

```
[1] -0.69257
```

Corresponding plot:

```
> plot(d.basisch$ph, d.basisch$height)
```



Correlation

All plots show two variables with a correlation of 0.7!

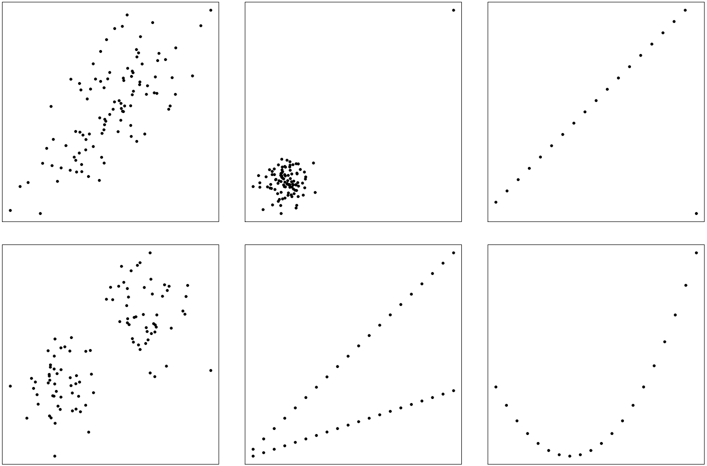


Figure: from the script Regression by Prof. H.R. Künsch

Regression

In this section we will have a look at . . .

. . . fitting a simple linear regression model

. . . checking model assumptions

. . . fitting a multiple linear regression model

Data

We continue working with the data set `d.basisch` from the correlation section.

Response variable:

- `height` or `h.quad`: Height of trees or squared height, respectively.

Possible explanatory variables:

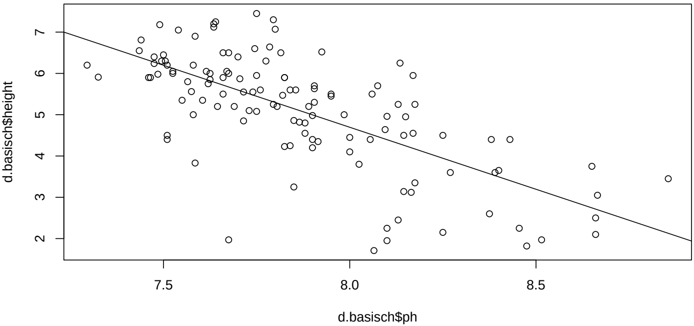
- `ph`: PH-Values of soil,
- `l.sar`: log(sodium absorption ratio)

Simple Linear Regression

The simple linear regression model is

$$Y_i = \alpha + \beta x_i + E_i, \qquad E_i \overset{i.i.d}{\sim} \mathcal{N}(0, \sigma^2)$$

Let us pick the variable `ph` as the explanatory variable.



Simple Linear Regression

Fit to data using `lm`:

```
> fit <- lm(formula = height ~ ph, data = d.basisch)
```

Show a summary of `fit`:

```
> summary(fit)
```

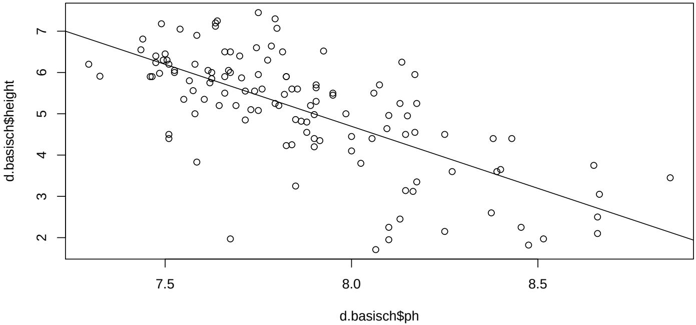
```
Call:
lm(formula = height ~ ph, data = d.basisch)

Residuals:
    Min      1Q  Median      3Q     Max
-3.7020 -0.5471  0.0874  0.6663  2.0033

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  28.7227     2.2395   12.82   <2e-16 ***
ph           -3.0034     0.2844  -10.56   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.008 on 121 degrees of freedom
Multiple R-squared:  0.4797,        Adjusted R-squared:  0.4754
F-statistic: 111.5 on 1 and 121 DF,  p-value: < 2.2e-16
```

Simple Linear Regression

Estimated Equation: `height` = 28.7 - 3.0 `pH`

Drawing line into scatterplot:

```
> plot(d.basisch$ph, d.basisch$height)
> abline(fit)
```



Simple Linear Regression - Residual Analysis

Diagnostics plots are straightforward:

```
> par(mfrow = c(2, 2))
> plot(fit)
```

- Tukey-Anscombe plot (is the variance of the errors E_i constant? Is the regression function correct?)

- Q-Q plot (are the errors E_i normally distributed?)

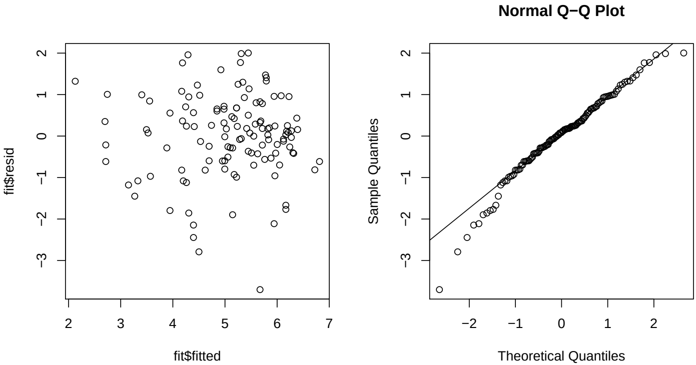
- Scale Location plot (similar to Tukey-Anscombe plot.)

- Leverage plot (what points have a strong influence on the fit?)

Simple Linear Regression

Residual plots by hand:

```
> par(mfrow = c(1,2))
> plot(fit$fitted,fit$resid)    # Tukey-Anscombe
> qqnorm(fit$resid)   # Quantil-Quantil plot
> qqline(fit$resid)   # adds the diagonal line
```



Simple Linear Regression

Some Tukey-Anscombe plots:

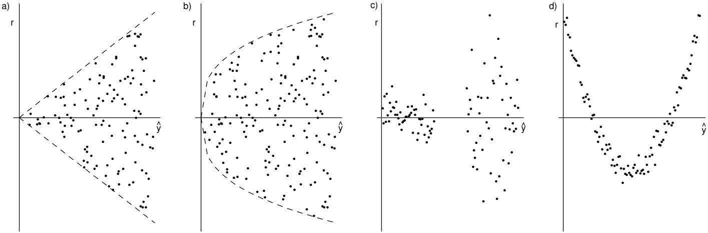


Figure: from the lecture notes "Regression" by Prof. H.R. Künsch

Multiple Linear Regression

Expand the simple linear model to more than one explanatory variable.

$$Y_i = \alpha + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} + E_i, \quad E_i \overset{i.i.d}{\sim} \mathcal{N}(0, \sigma^2)$$

Fit the model with `lm`:

```
> fitm <- lm(height ~  ph + l.sar, data = d.basisch)
```

Multiple Linear Regression

```
> summary(fitm)

Call:
lm(formula = height ~ ph + l.sar, data = d.basisch)

Residuals:
    Min      1Q  Median      3Q     Max 
-4.1314 -0.4911  0.0849  0.6488  2.4754 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  26.9466     2.7445   9.818  < 2e-16 ***
ph           -2.7558     0.3603  -7.649  5.6e-12 ***
l.sar        -0.2519     0.2255  -1.117    0.266    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.007 on 120 degrees of freedom
Multiple R-squared:  0.485,	Adjusted R-squared:  0.4764 
F-statistic: 56.51 on 2 and 120 DF,  p-value: < 2.2e-16
```

Multiple Linear Regression

Residual analysis:

- Look at the same plots as for simple linear regression.

```
> plot(fitm)
```

- It may help to plot the explanatory variables against the residuals.

```
> plot(d.basisch$ph, fitm$resid)
> plot(d.basisch$l.sar, fitm$resid)
```

Going further

In this section we will briefly look at . . .

. . . how to report your results

Report your results

Knitr: awesome tool to embed your R code and results (including tables and plots) in a nicely looking document.
http://yihui.name/knitr/

Rmarkdown can be used to produce reports in any of these formats:

- PDF
- HTML
- slides
- Word document (if your collaborators require it)

http://rmarkdown.rstudio.com

Book about reproducible research:
http://christophergandrud.github.io/RepResR-RStudio/

Useful Cheatsheets

Useful cheatsheets for R can be found here:
https://www.rstudio.com/resources/cheatsheets/