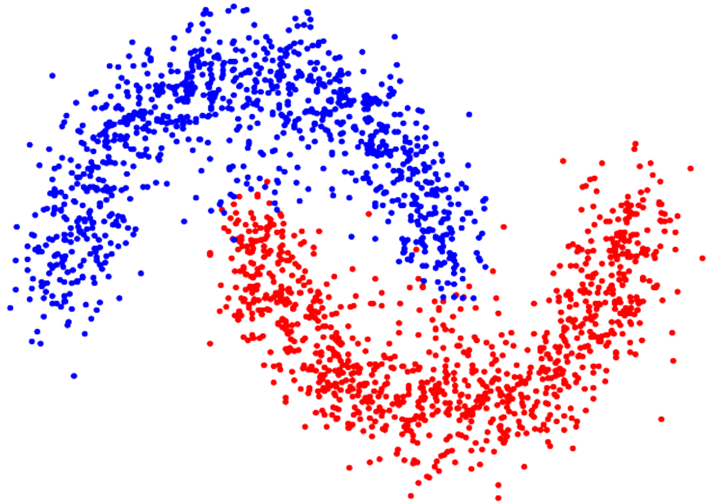
University of Zurich[UZH]

Blockkurs:
Introduction to Machine Learning for Psychologists

Introduction to Machine Learning (and R)



Yannick Rothacher

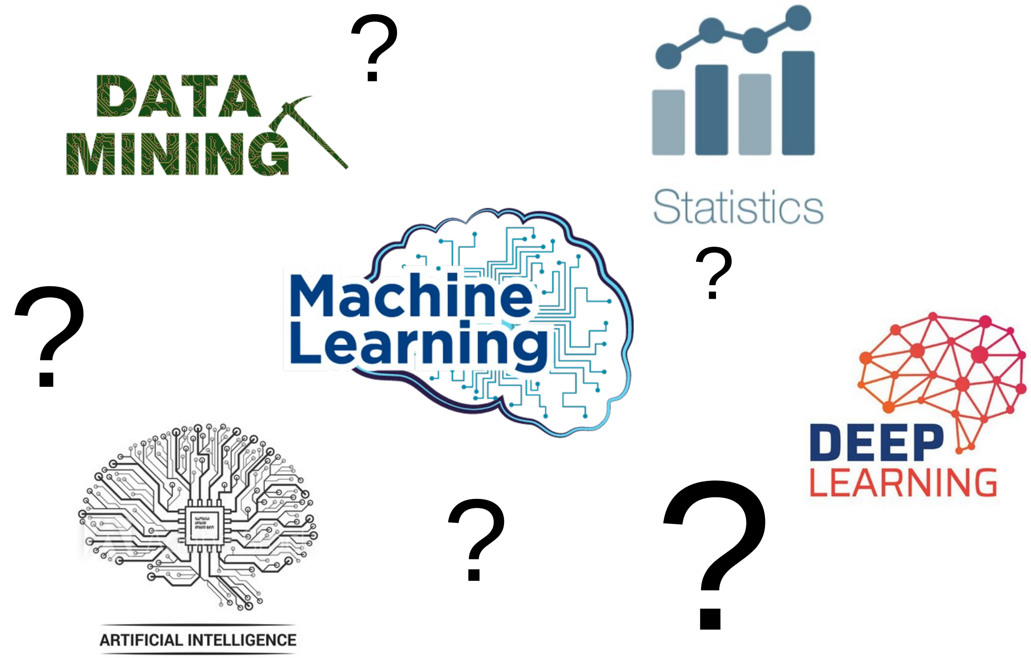
Zürich, FS2025

Who am I?

- **Yannick Rothacher**
- Background: Biology
  (PhD in Neuroscience)
- Further education in "applied statistics" at ETH Zürich
- Post-Doc at the Professorship for Psychological Methods, Evaluation and Statistics (Prof. Carolin Strobl)
- Hired as “Data Scientist” at Swiss Paraplegic Research
- yrothacher@gmail.com

Course organization

- Two day course
- Mixture of lectures and practical exercises in R
- To get the credit point you have to write an **analysis report** after the course
  - The idea is that you take a data set ideally from your research, and apply one or multiple methods from this course to it
  - If you do not have access to a suitable data set I can provide you with one
  - The report has to be **handed in per mail until the 27. April 2025**
- Material is available on: https://github.com/ryannick28/MLCourse2025

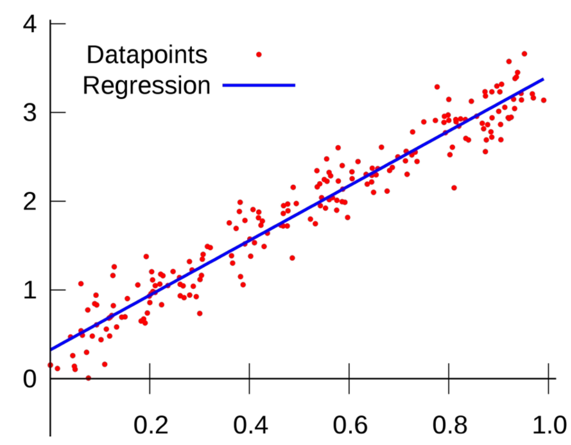


In groups:

- Why are you interested in machine learning?
- What are your expectations of this course?
- What do you associate with the term “machine learning”?

Course goals

- Give an insight into **various methods** in Machine Learning
- Teach the operating principles of the presented algorithms
- Practice the application of Machine Learning methods to data
- Deepen your skills in **R**



Tentative timetable

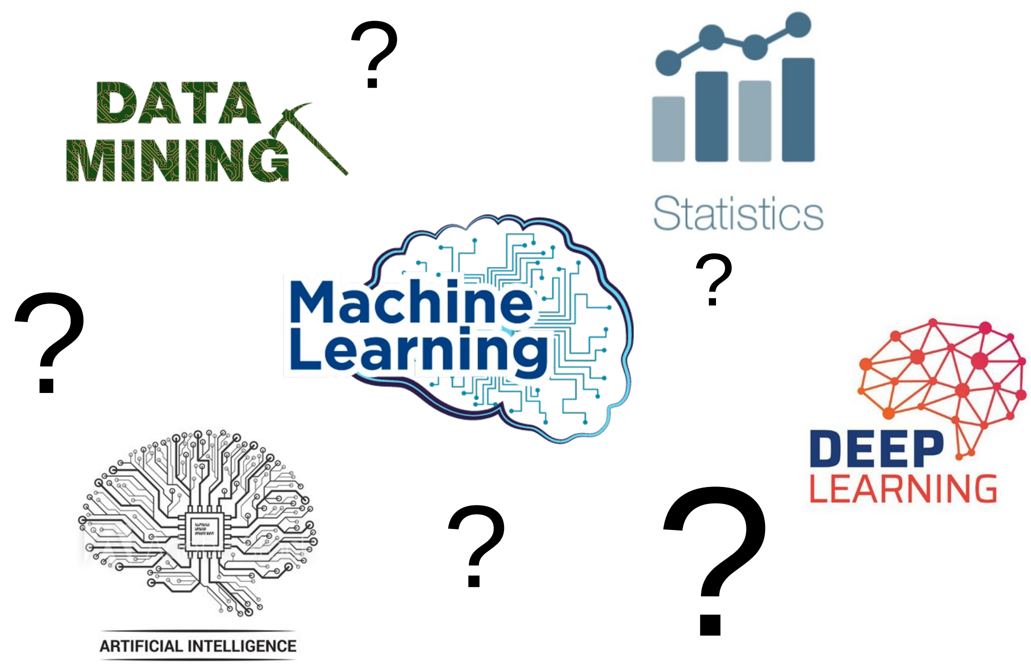
--- DAY 1 (2.4, AND-3-46) ---

09:30 - 11:00	Welcome + RIntro
11:00 - 11:45	PCA
11:45 - 12:30	PCA Exercise
--Lunch--	
14:00 – 14:45	K-Means
14:45 - 15:15	K-Means Exercise
14:15 - 15:45	KNN
15:45 - 16:15	KNN Exercise
16:15 - 17:15	Crossvalidation + Write own function

--- DAY 2 (4.4, AND-2-44) ---

09:30 - 10:30	Decision trees
10:30 - 11:30	Decision trees Exercise
--Lunch--	
13:00 - 14:30	Ensemble methods (+ Interpretability)
14:30 - 15:15	Ensemble methods Exercise
15:15 - 16:15	Neural Networks
16:15 - 17:15	Neural Networks Exercise

What is Machine Learning?



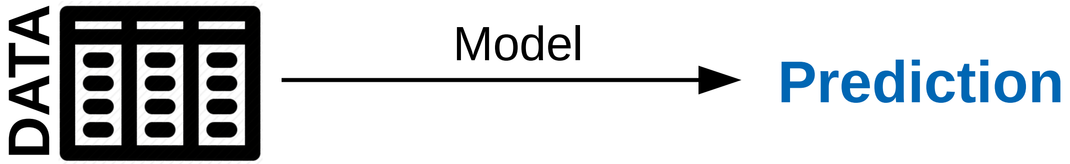
- Distinction from Machine Learning to other statistical methodology not always clear
- When comparing Machine Learning with “classical” statistics:
  - Statistical models are generally designed for **inference**
  - Machine Learning models are generally designed for **prediction**

Application of Machine Learning

Being able to **predict** certain outcomes based on data can be important in many different areas in **research and industry**

Examples:

- Predict the winner of a basketball game
- Predict the weather of tomorrow
- Predict whether a medical scan shows an image of a tumor
- Predict whether an email is spam or not
- Predict how likely a person is about to develop depression

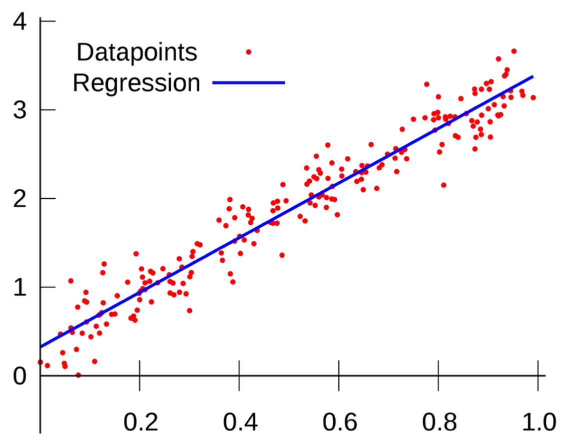


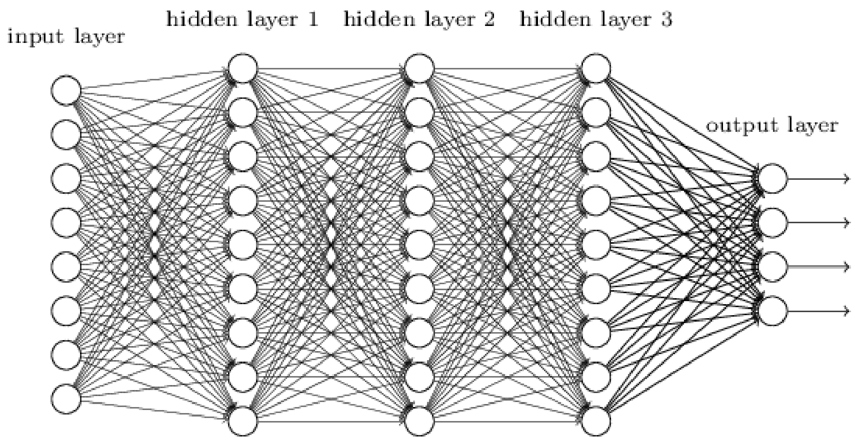
In all cases: **Predictions are based on data !**

University of Zurich[UZH]

Prediction models don't have to be complicated

- Simple linear regression can also be used to predict values of new observations



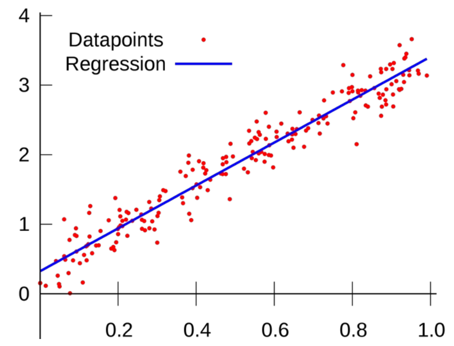


- However, sometimes statistical models have limited prediction accuracy, but allow **inference about the relation** between predictors and target variables (e.g. showing a significant influence of a treatment).
- In many Machine Learning models, the prediction accuracy is very good but it is difficult to interpret the variables' relations (e.g. neural network)

Application of Machine Learning

- Again: In general one tries to predict a target variable based on predictor variables

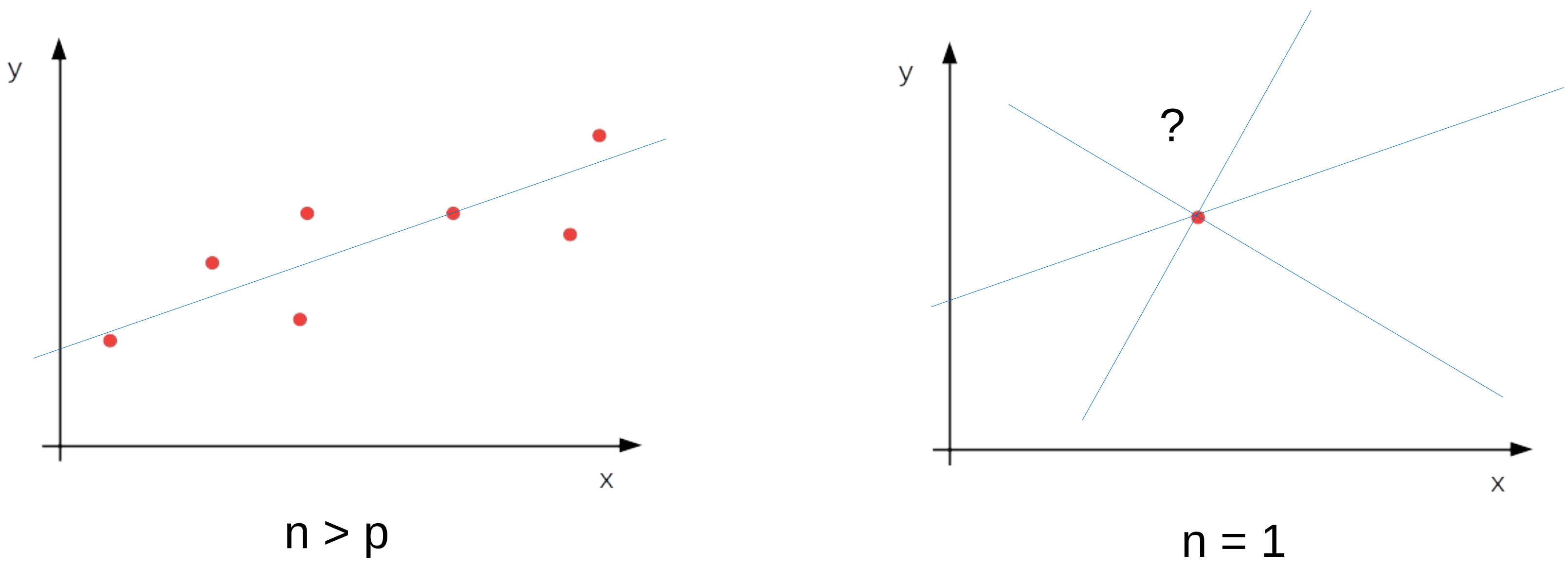
target variable ~ predictor variables
y ~ X



- Target variable is usually a category or a number
  - Y is category: “Classification”
  - Y is metric: “Regression”
- In real-life data, there are often many predictor variables (genetic data: up to 10’000 predictors)
- Can even be n << p (much more variables (p) than data points (n))
- This case can be difficult to handle with conventional methods (for example linear regression)

Challenges of high-dimensional data

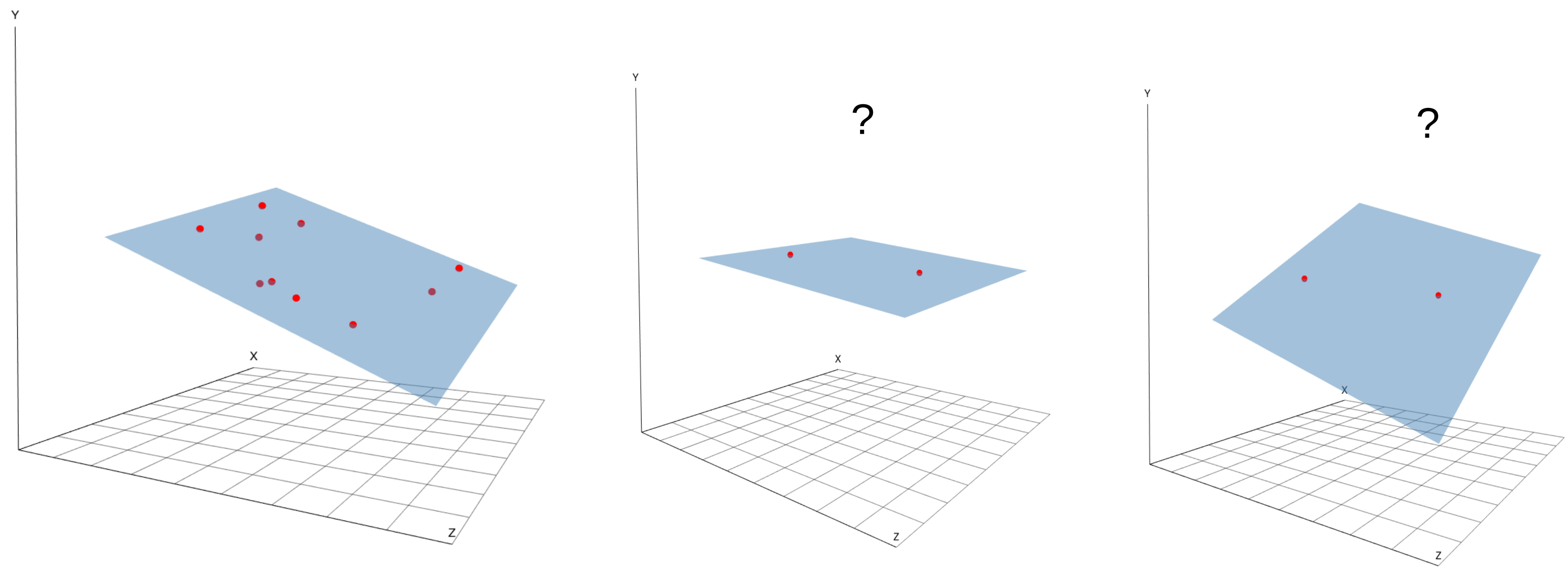
- For example linear regression only works for n > p :



- We need methods for situations with n < p
- Machine Learning methods are usually able to handle n < p situations

Challenges of high-dimensional data

- For example linear regression only works for n > p :

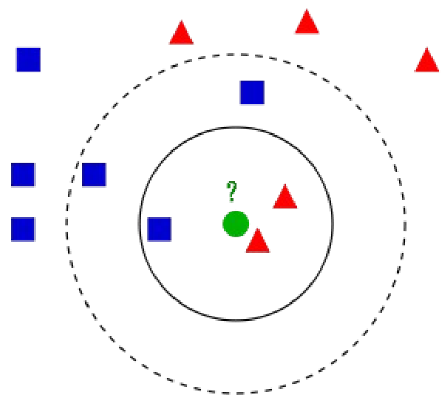


n > p   n = 2

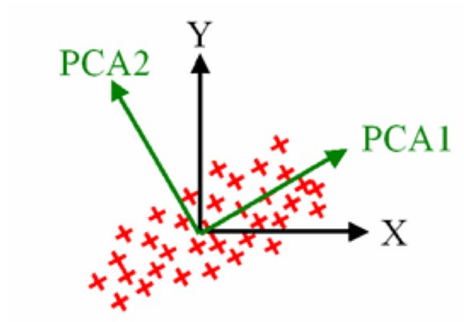
- We need methods for situations with n < p
- Machine Learning methods are usually able to handle n < p situations

University of Zurich

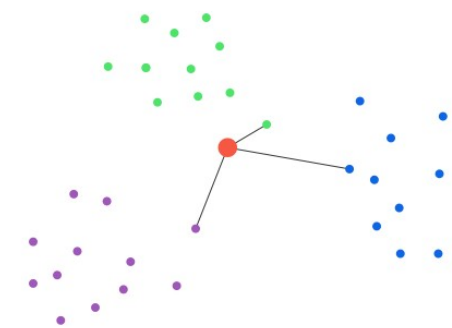
Outlook: Machine Learning methods



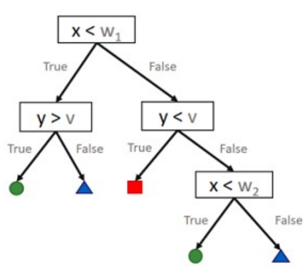
K-nearest neighbor



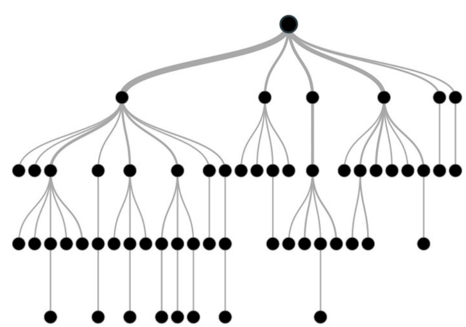
Principal Component Analysis



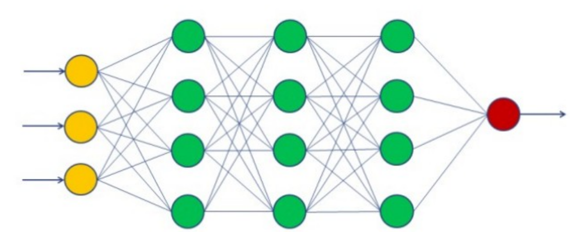
K-means clustering



Decision trees



Random Forest



Neural Networks

Quick R questionnaire

To get an impression of how used you are to R, think about the following statements:

- “I have never used R or R-Studio.”
- “I learned the basics of R once, but most of it is not really present anymore.”
- “I know the R basics.”
- “I am used to handling data sets in R and writing R scripts.”
- “I am used to writing if() statements and for() loops in R”
- “I use R every day.”

R – statistical computing

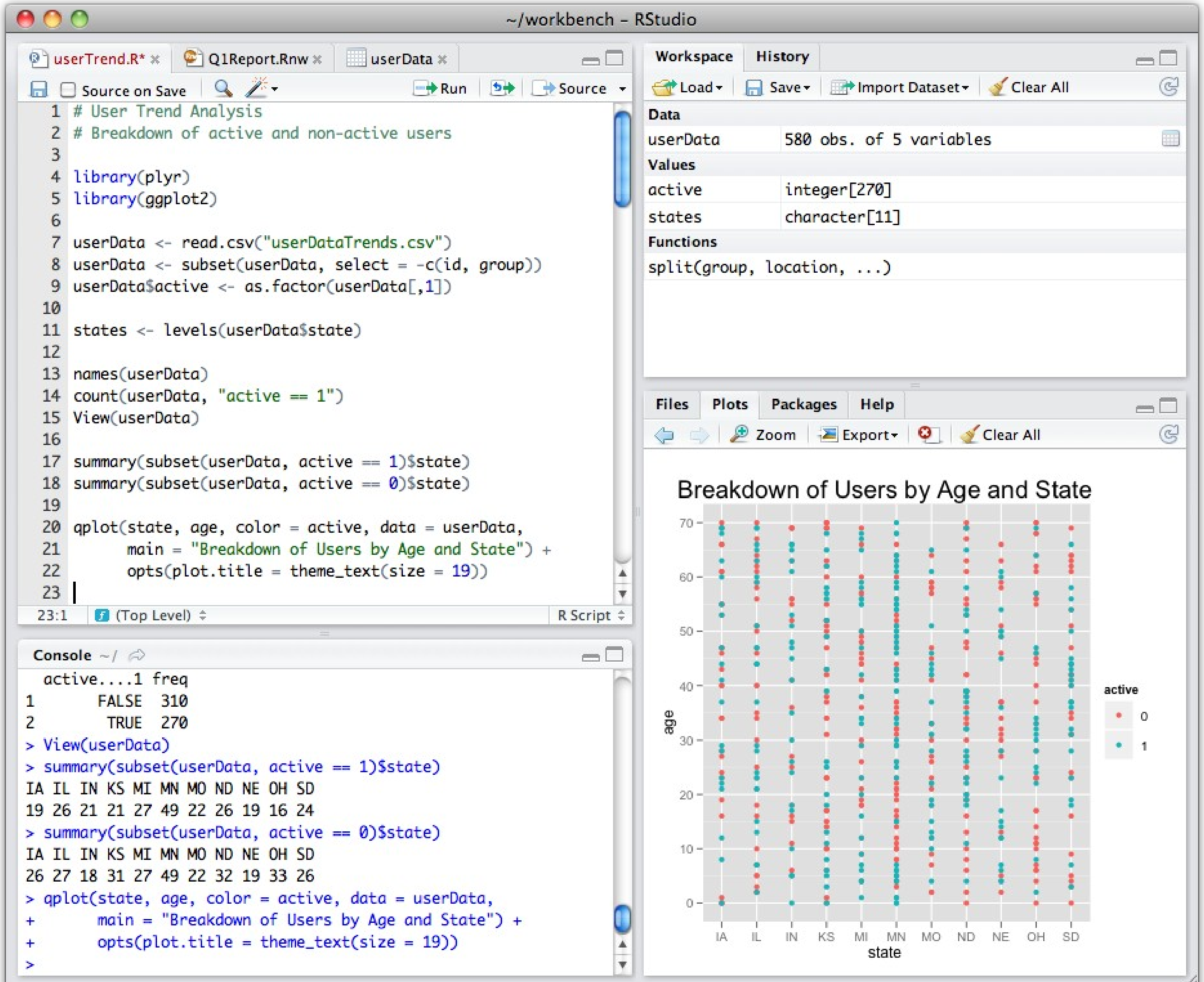
R is:

- Free to use, open-source
- Very flexible (> 10’000 add-on packages for R)
- Widely used among statisticians

In this lecture:

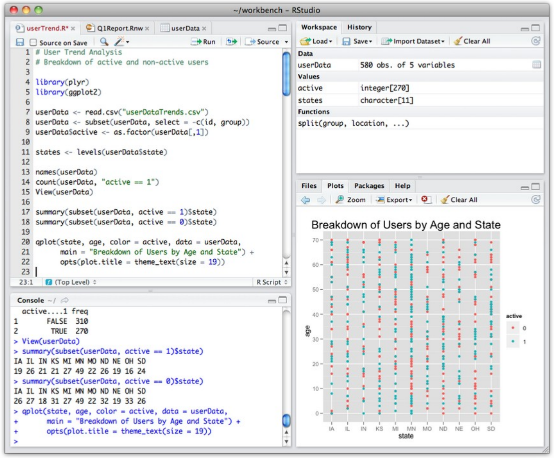
- Look at **R basics** (basic data types, simple functions, plotting, importing data,...)

R-studio: Integrated Development Environment (IDE) for R



R Introduction

- Classically we work in Rstudio in an R-script
- Execute the code from the script in the R-console
  - “source” to execute the whole script
  - Ctrl+enter to execute the current line or selection in script



- Using R as a calculator and to create objects:

```
> 1 + 1
[1] 2
> 10 * 30
[1] 300
> a <- 3 # create the variable a, which holds the value of 3
> b <- 25
> a + b
[1] 28
> a * b
[1] 75
```

Vectors (numerical and character)

- A vector is a combination of multiple elements and is created with **c()**:

```
> vec1 <- c(20, 122, 39)
> vec1
[1] 20 122 39
```

- A vector containing the integers from x to y can be created with **x:y**:

```
> 1:10
[1] 1 2 3 4 5 6 7 8 9 10
```

- Not only numerical values possible, example of a character-vector (text):

```
> vec2 <- c("Category1", "blue", "mixed")
> vec2
[1] "Category1" "blue" "mixed"
```

Selection of elements in vector

- When processing data one is often interested in selecting specific elements of the data. The selection of elements in R is performed with the square brackets **[ ]**:

```
# Selection of elements in vector:
> vecA <- c(2, 6, 7, 9)
> vecA
[1] 2 6 7 9
> vecA[2]
[1] 6
```

- Multiple elements can be selected by passing the wanted positions as a vector:

```
> vecA[c(2, 4)]
[1] 6 9
```

- Nested call:

```
> s <- c(2, 4)
> vecA[s]
[1] 6 9
```

- We can use negative indices to remove cases:

```
> vecA[-c(2, 4)]
[1] 2 7

> vecA[-s]
[1] 2 7
```

Data Frames (and reading in data)

- Data frames are commonly used to represent tabular data in R. The columns of a data frame are vectors. Exemplary data frame:

```
> dat
  vecA   vecB vecC
1    2   good 0.40
2    6    bad 0.20
3    7 medium 0.42
4    9   good 0.90
```

- Usually, a data frame is obtained by reading in a data file. In this workshop we will work with **.rda** files, .rda files can be read in with the **load()** command:

```
> load("toyDataFrame.rda")
> ls()   # List objects in (global) environment to see what was loaded
```

- When reading in files as shown above, the **working directory** has to be set to the location of the file
  - The working directory is the folder where R looks for files to read in or where R saves created files (if not told otherwise)
  - The working directory can be set with **setwd**(PATH TO DIRECTORY) and viewed with **getwd**(). The working directory can also be set with the RStudio GUI (**Session** > **Set Working Directory**)

Selection of elements in a data frame

- In a data frame, columns can be selected using the **$** sign:

```
> dat
  vecA   vecB vecC
1    2   good 0.40
2    6    bad 0.20
3    7 medium 0.42
4    9   good 0.90

> dat$vecC
[1] 0.40 0.20 0.42 0.90

> dat$vecB
[1] "good" "bad" "medium" "good"

> dat$vecB[2]
[1] "bad"
```

- We can also use the $ sign to add a new column or remove a column:

```
# Add new column:
> dat$newCol <- 1:2
> dat
  vecA   vecB vecC newCol
1    2   good 0.40      1
2    6    bad 0.20      2
3    7 medium 0.42      1
4    9   good 0.90      2

# Remove column:
> dat$vecB <- NULL
> dat
  vecA vecC newCol
1    2 0.40      1
2    6 0.20      2
3    7 0.42      1
4    9 0.90      2
```

Selection of elements in a data frame

- The square brackets can also be used to select elements in a data frame. In that case, the wanted row and column positions are passed to the brackets, separated by a comma:

```
> dat
  vecA   vecB vecC
1    2   good 0.40
2    6    bad 0.20
3    7 medium 0.42
4    9   good 0.90
# Element of dat in the second row
in the third column:
> dat[2, 3]
[1] 0.20
# All elements in the first column:
> dat[, 1]
[1] 2 6 7 9
> dat[, "vecA"]   # Selection with name
[1] 2 6 7 9
```

```
# First and fourth row with
second and third column:
> dat[c(1, 4), c(2, 3)]
  vecB vecC
1 good  0.4
4 good  0.9
```

University of Zurich[UZH]

Logicals

- **Logicals** are an important class in R
- Logicals can only take the values **TRUE** or **FALSE**:

```
> LVec <- c(TRUE, TRUE, FALSE, TRUE)
> LVec
[1]  TRUE  TRUE FALSE  TRUE
```

- **Logical operators** compare values:

```
> 5 > 3
[1] TRUE
> a <- 33
> b <- 5
> a == b   # ‘is equal to’
[1] FALSE
> c(6, 7, 2, 4, 1) < 3
[1] FALSE FALSE TRUE FALSE TRUE
```

Logicals can be used for selection

```
# Selection with logicals:
> v <- 1:4
> v
[1] 1 2 3 4
> v[ c(TRUE, TRUE, FALSE, FALSE) ]
[1] 1 2

> dat
  vecA   vecB vecC
1    2   good 0.40
2    6    bad 0.20
3    7 medium 0.42
4    9   good 0.90

> dat[ c(1,2), ]
  vecA vecB vecC
1    2 good  0.4
2    6  bad  0.2
```

same thing

```
> dat[ c(TRUE, TRUE, FALSE, FALSE), ]
  vecA vecB vecC
1    2 good  0.4
2    6  bad  0.2

> dat$vecA
[1] 2 6 7 9

> dat$vecA < 7
[1]  TRUE  TRUE FALSE FALSE

> dat[ dat$vecA < 7, ]
  vecA vecB vecC
1    2 good  0.4
2    6  bad  0.2
```

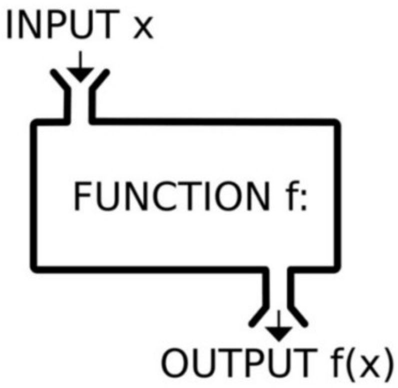
Categorical variables in R

- Categorical variables should ideally not be coded as numerical vectors in R
- There is the datatype **factor** specifically for categorical variables
- A factor not only contains the values of the individual elements, but also the list of possible categories ("levels")
- A factor can be created with **factor()**:

```
> d <- c(1, 2, 2, 3, 1)
> dfc <- factor(d, levels = c(1, 2, 3), labels = c("catA", "catB", "catC"))
> dfc
[1] catA catB catB catC catA
Levels: catA catB catC
```

Functions in R

- A function in R takes an **input**, processes it and returns an **output**
- Functions are applied by writing their name followed by normal brackets. The input for the function is defined in the brackets
- There are many ready-to-use functions in R:




```
# Calculate the mean value of a vector:
> d <- c(1, 2, 2, 3, 1)
> mean(d)
[1] 1.8
# Return the absolute value of a number:
> abs(-2)
[1] 2
# Calculate the correlation coefficient of two vectors:
> cor(x = c(2,3,4,5,6), y = c(6,4,3,6,19), method ="spearman")
[1] 0.4616903
# Take a random sample from a vector:
> sample(1:100, size = 5)
[1]  5 34 49 96 80
```

Functions in R

- How a function is applied (e.g. which input arguments exist, how they are named, …) can be viewed by calling its help page (or searching the internet):

```
> ?mean
> ?cor
> ?sample
```

R Packages

- Publicly available R packages contain functions and objects for specific purposes
- R packages can be installed with **install.packages**("NAME OF PACKAGE") or via the RStudio GUI
- To make the content of a package readily available in an R session, the installed package has to be loaded with **library**(NAME OF PACKAGE)

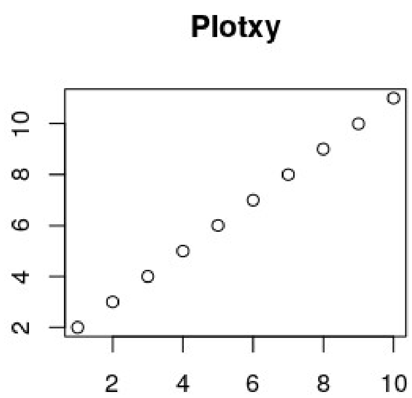
```
# Installing a package only has to be done once:
> install.packages("lme4")   # Package for mixed models

# Loading a package has to be done in each session:
> library(lme4)
> lmer(y ~ ., data=d)   # lmer() is a function from lme4
```

Creating plots in R

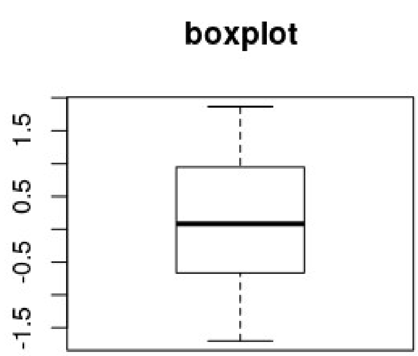
- There are many different ways and packages to create plots in R
  - Basic scatterplots can be created with **plot**():

```
> plot(x = 1:10, y = 2:11, main = "Plotxy")
```



  - Boxplots can be created with **boxplot():**

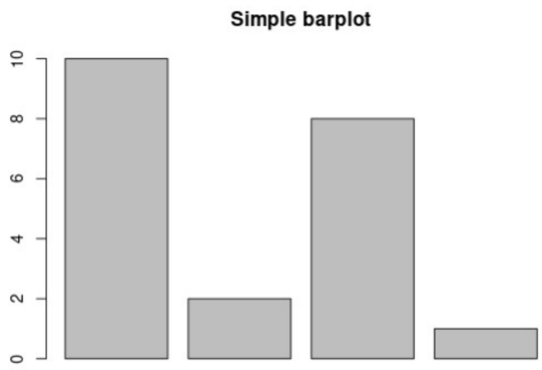
```
> boxplot(rnorm(30), main='boxplot')  # rnorm(30) draws 30 datapoints from
                                      # the (standard) normal distribution
```



Creating plots in R

- There are many different ways and packages to create plots in R
  - Barplot created with **barplot()**:

```
> barplot( c(10, 2, 8, 1) , main = "Simple barplot")
```



Two basic programming structures

- If()-statements
- for()-loops

If() statements

- If-statements are a very important tool in programming
- An if-statement tells R to execute a block of commands, if a condition is true
- Structure is always the same:
  - **if** (condition) {*execute if TRUE*} **else** {*execute if FALSE*}
  - Simple example:

logical condition
(TRUE in this case)

```
a <- 2
b <- 5
if (a < b){
  a <- a + 1  # execute if condition is true
}else{
  a <- a - 1  # execute if condition is false
}

> a
[1] 3  # 2 + 1
```

FALSE

```
# reversed case:
a <- 20
b <- 5
if (a < b){
  a <- a + 1
}else{
  a <- a - 1  # gets executed
}

> a
[1] 19  # 20 - 1
```

If() statements

- One can also use more complicated conditions:
  - If(a > b **&** a > c){…} "execute code if a is bigger than b **and** a is bigger than c"
  - If(a > b **|** a > c){…} "execute code if a is bigger than b **or** a is bigger than c"
- **Example**: Use if-statement to see if there are **invalid values** in a vector
  - For a collected variable it is known that values larger than 10 are not possible (e.g. questionnaire where 10 is maximum answer)

```
> questionnaire_data
 [1]  6 11  3  1  6  5  3  6  1  5  3  3  9  5  7  5  3  3  3  4
> if(max(questionnaire_data) > 10){
   cat('there are invalid entries in this data')  # execute if true
  } else {
  cat('no invalid data')  # execute if false
  }
there are invalid entries in this data
```

If() statements

- One can also combine multiple if statements using *else if*:

```
if (condition1) {
    execute this
} else if (condition2) {
    execute this
} else {
    execute this
}
```

For() loops

- **For**-loops are another important tool in programming
- For-loops execute a selection of code multiple times
- The structure in R looks as follows:
  - for(variable in vector) { *execute code for variable*}
  - Simple example:

```
> for(i in c(2,5,6,7)){  # i is variable, c(2,5,6,7) is the vector
    a <- i + 5
    print(a)  # force to show a
  }
[1] 7
[1] 10
[1] 11
[1] 12
```

For() loops

- **Good trick**: use the variable of the for-loop as an **index**:

```
> a <- NA  # declare “a” because we use a[] later
> for(i in 1:5){
    a[i] <- i + 5
  }
> a
[1]  6  7  8  9 10
```

- **If-statements** and **for-loops** are often combined with each other
  - E.g. Double only the values smaller than 20 in a vector:

```
> a <- c(2, 44, 6, 4.5, 94)
> a
[1]  2.0 44.0  6.0  4.5 94.0
> b <- NA
> for (i in 1:length(a)){  # same as 1:5
    if(a[i] < 20){b[i] <- a[i]*2} else {
      b[i] <- a[i]
    }
  }
```

```
> b
[1]  4 44 12  9 94
```