GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Applied Machine Learning with R

Felix Heinrich

Breeding Informatics Group
Department of Animal Science
Georg-August University Göttingen

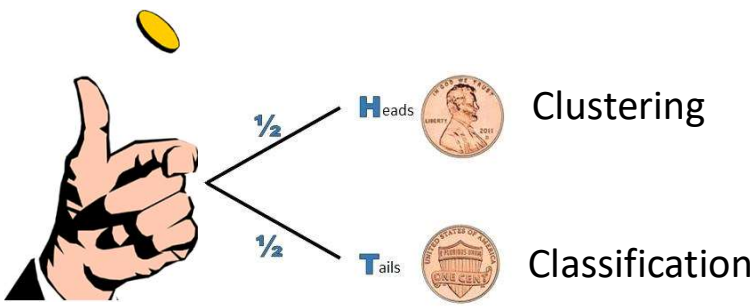
Introduction to Machine Learning

GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Course Program:

- **Important dates:**
  - From 27th February to 10th March
  - From 9:15 to 15 o'clock (not necessarily)
- **Room- L09**
  - Practical Exercises: During the course
    - Johanna-Sophie Schlüter

  - Exam Registration is mandatory (**M.IPA.0015.Mp: Applied Machine Learning in Agriculture with R**)
    - Oral examination (approx. 20 minutes)
    - Examination date: 10.03.22
    - More information later
    - **No term paper**



Course Program:

From 27th February to 10th March

- **What is Machine Learning**
- **Exercises with R**
  - Read external files
  - Manipulate the datasets
  - Advanced programming with R
  - Graphical visualisation with **ggplot**
- **Clustering & Dimensionality Reduction**
  - k-means, hierarchical clustering and principle component analysis (PCA)
- **Classification**
  - Support vector machine (SVM), decision trees, random forest, neuronal networks

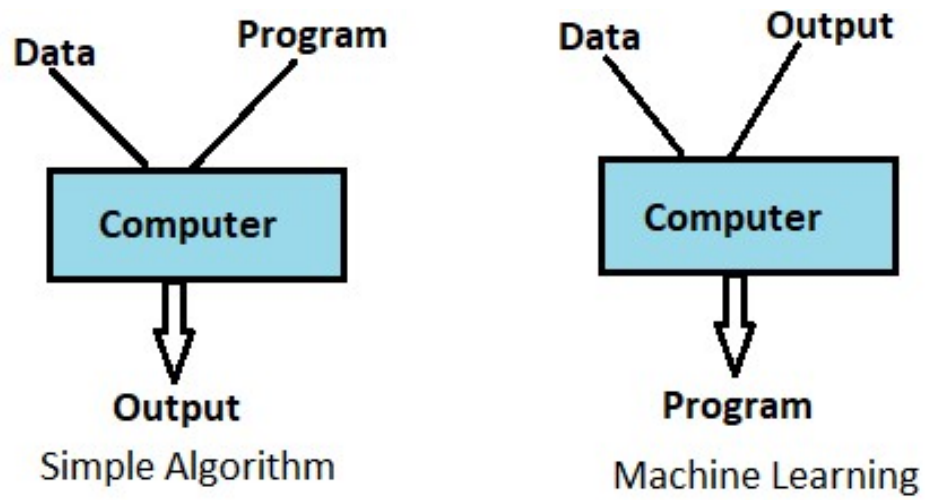
- The deeper descriptions of mathematical methods are not the main focus of the course.

Introduction to Machine Learning

GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Machine learning:

- **Definition:**
  - Computational methods using experience to improve performance, e.g., to make accurate predictions.
- **Experience**:
  - Data-driven task, thus statistics, probability.
- **Example:**
  - Use height and weight to predict gender
- **Informatics:**
  - Need to design efficient and accurate algorithms, analysis of complexity, theoretical guarantees.



Introduction to Machine Learning

- It is very hard to write programs that solve problems like recognizing a face.
  - We don't know what program to write because we don't know how our brain does it.
  - Even if we had a good idea about how to do it, the program would be very complicated.

- Instead of writing a program by hand, we collect lots of examples that specify the correct output for a given input.

- A machine learning algorithm takes these examples and produces a program that does the job.
  - The program produced by the learning algorithm may look very different from a typical hand-written program. It may contain millions of numbers.
  - **If we do it right, the program works for new cases as well as the ones we trained it on.**

Introduction to Machine Learning

- "**Learning**" is used when:
  - Human expertise does not exist (navigating on Mars),
  - Humans are unable to explain their expertise (speech recognition)
  - Solution changes in time (routing on a computer network)

- Examples of tasks that can be solved by using a learning algorithm
  - **Recognizing patterns**:
    - Facial identities or facial expressions
    - Medical images
  - **Generating patterns**:
    - Generating images or motion sequences
  - **Recognizing anomalies**:
    - Unusual sequences of credit card transactions
  - **Prediction**:
    - Future stock prices or currency exchange rates

Introduction to Machine Learning

- **Machine learning** is programming computers to optimize a performance criterion using example data or past experience.

- **Role of Statistics:**
  - Inference from a sample

- **Role of Computer science:**
  - Efficient algorithms to solve the optimization problem
  - Representing and evaluating the model for inference

Build a model that **is a good and useful approximation** of the data.

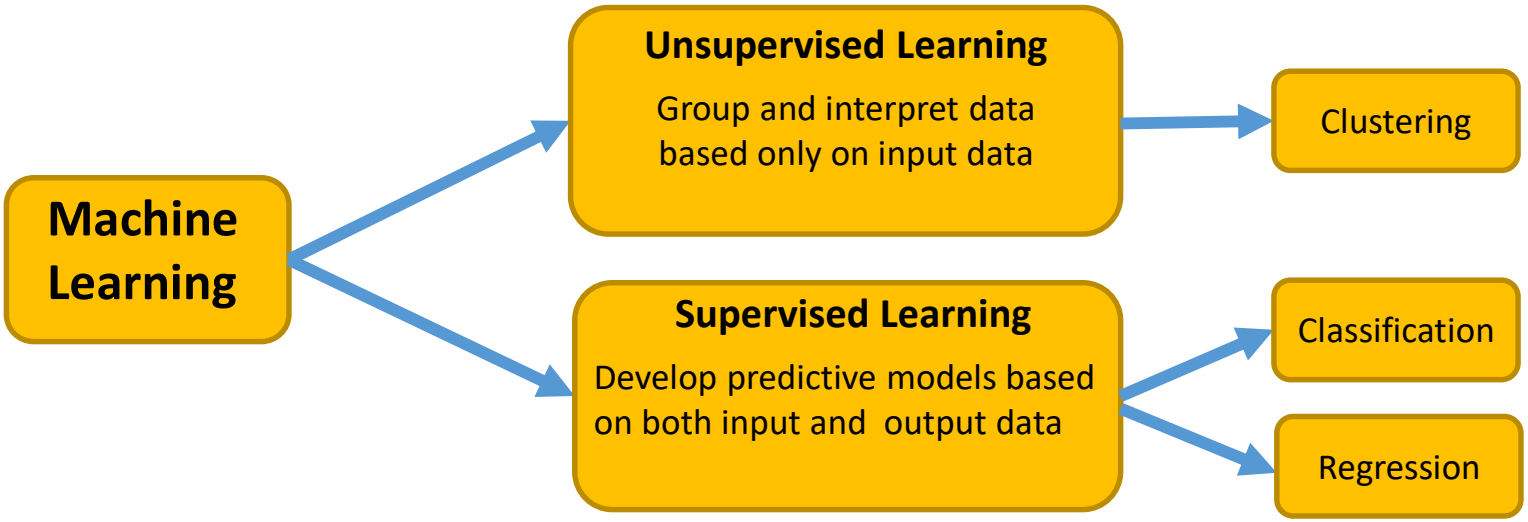
Performance

- **<u>Several factors can affect the performance:</u>**
  - **Types of training** provided
  - The form and extent of any initial **background knowledge**
  - The **type of feedback** provided
  - The **learning algorithms** used

- **Two important factors:**
  - Modeling
  - Optimization

Algorithms

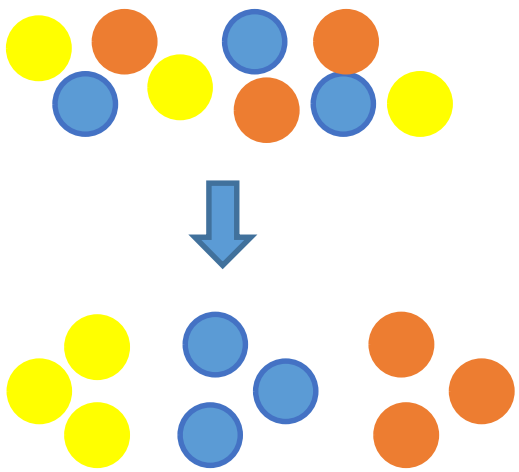
- The success of machine learning systems also depends on the algorithms.
- The algorithms control the search to find and build the knowledge structures.
- The learning algorithms should extract useful information from training examples.



Clustering

- A cluster is a group of related objects

- Clustering:
  - Identification of natural groups/subsets of objects in a data set



Data clustering

A good clustering method detects high quality clusters in which:

- The **homogenity** is high: high **intra-cluster** similarity
- The **separation** is high: low **inter-cluster** similarity

- The quality of a clustering result → **similarity measure** used

Aim of clustering

Clustering is a function that maximizes similarity between objects within a cluster and minimizes similarity between objects in different clusters.

Data clustering

- Identify relationships and patterns in a set of data
- Identify similar groups between objects
- Get more insights in underlying data structure

- Deal with numerical measures of the data
  - Distance metrics, d
    - $d(a,b) \geq 0$
    - $d(a,a)= 0$
    - $d(a,b) = 0$ if and only if $a = b$
    - $d(a,b) = d(b,a)$ → symmetry
    - $d(a,c) \leq d(a,b) + d(b,c)$ → triangle inequality

Similarity and Distance

Similarity and distance are measures of how closely- or distantly-related objects are based upon a collection of characters that describe those objects.

Example: relationships among some common animals:

	Characters			
Animals	*Hair*	*Lungs*	*Egg-laying*	*Milk*
Dog	1	1	0	1
Turtle	0	1	1	0
Canary	0	1	1	0
Goldfish	0	0	1	0

1=present, 0=absent

Step-1: Measure of similarity

- Numbers of matches & mismatches

		Animal-j	
		1	0
Animal-i	1	a	b
	0	c	d

Step-2: Simple matching coefficient

$$s_{ij} = \frac{a+d}{a+b+c+d}$$

Jaccard coefficient:

$$s_{ij} = \frac{a}{a+b+c}$$

		Canary	
		1	0
Goldfish	1	1	0
	0	1	2

$\Rightarrow$ s_{ij}= (1+2)/4=0.75

Step-3: Similarity matrix between all animals

	Dog	Turtle	Canary	Goldfish
Dog	1	0.25	0.25	0
Turtle	0.25	1	1	0.75
Canary	0.25	1	1	0.75
Goldfish	0	0.75	0.75	1

Similarity and Distance

	Characters			
Animals	*Hair*	*Lungs*	*Egg-laying*	*Milk*
Dog	1	1	0	1
Turtle	0	1	1	0
Canary	0	1	1	0
Goldfish	0	0	1	0

1=present, 0=absent

Measure of dissimilarity

Distance metrics:

- Euclidean distance: $d_{ij} = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \ldots}$
- Manhattan distance: $d_{ij} = \sum_{k=1} |x_{ik} - x_{jk}|$

Dissimilarity matrix between all animals

	Dog	Turtle	Canary	Goldfish
Dog	**0**			
Turtle	**1.73**	**0**		
Canary	**1.73**	**0**	**0**	
Goldfish	**2.0**	**1.0**	**1.0**	**0**

Clustering ≠ Classification

Common definition:

- Classification is the grouping of information or objects based on similarities

Remember the definition of clustering:

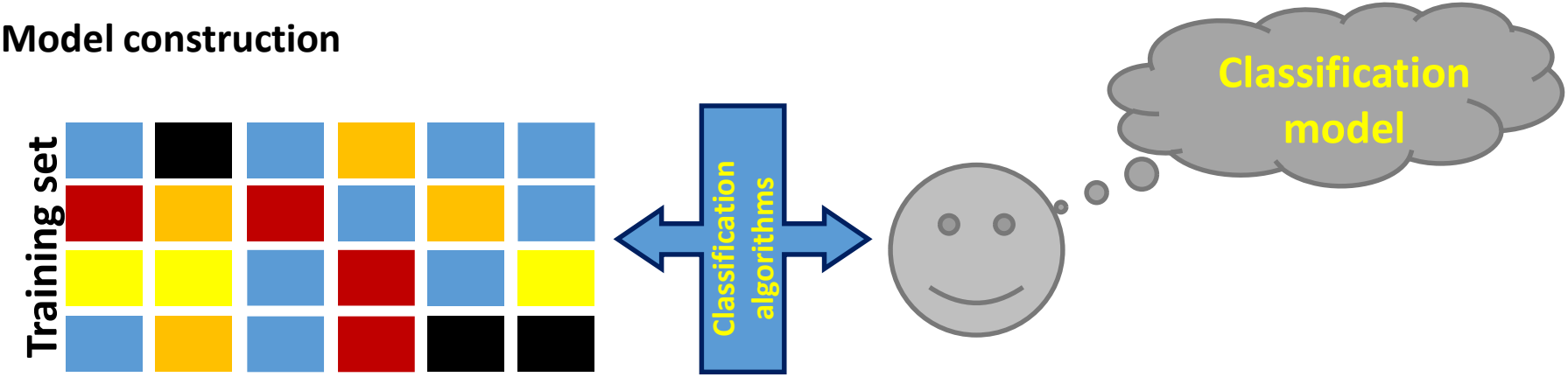
- A cluster is a group of related objects and is identified by maximizing the similarity between objects within a cluster

The aim of classification

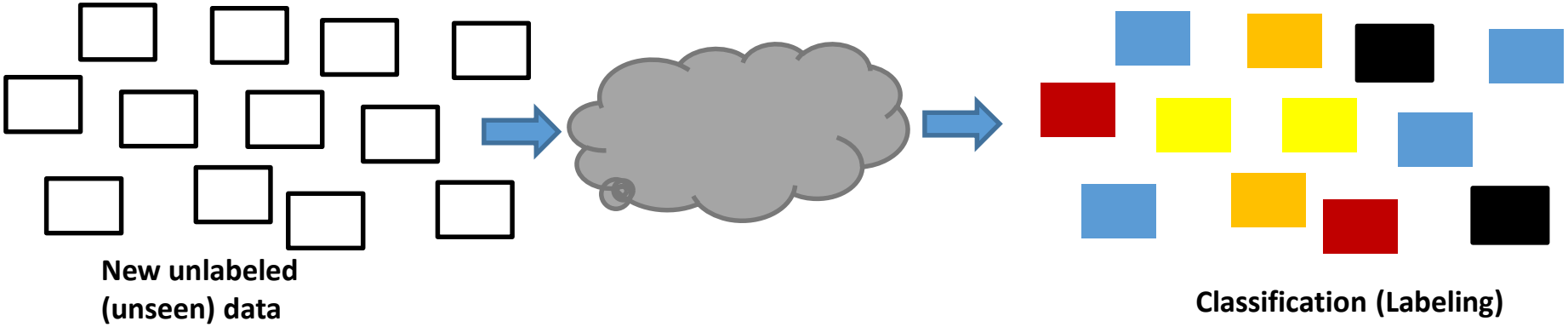
- The goal of data classification is to organize and categorize data into distinct classes
  - A model is first created based on training data (learning)
  - The model is then validated on the testing data
  - Finally, the model is used to classify the new data

Classification process: Learning the model

1. **Model construction**

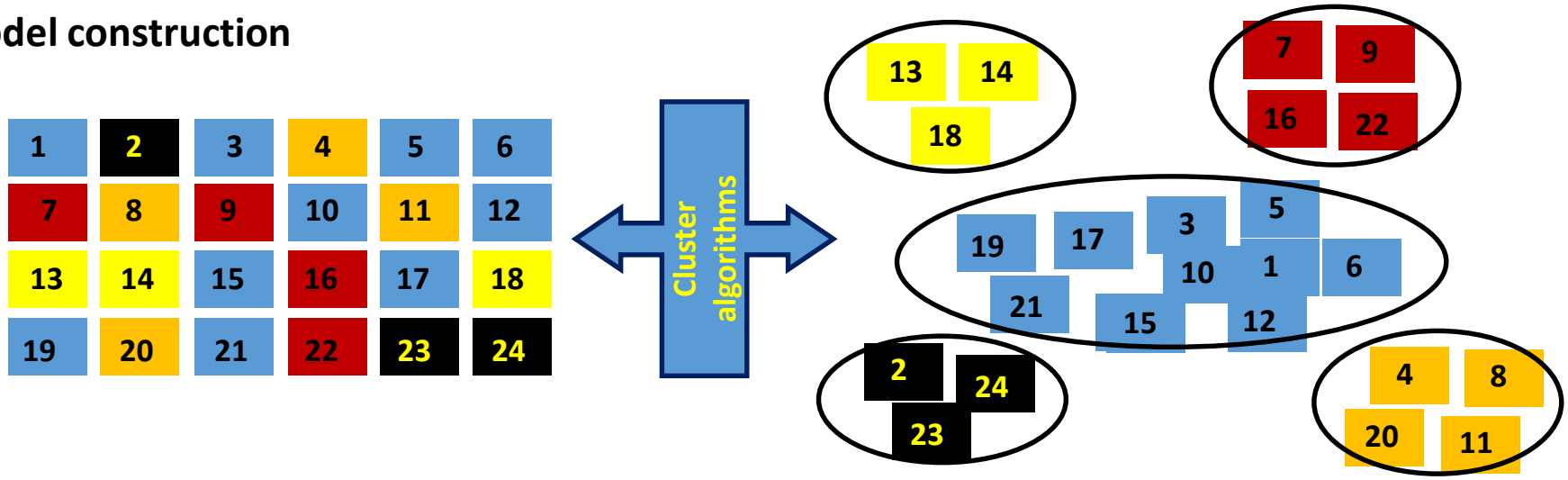


2. **Use the model in the prediction**



Clustering process:

1. **Model construction**



The process of putting similar data together

- **No training data**
- **A notion is needed (e.g. similarity or distance)**

Clustering vs. Classification

Clustering	Classification
Unknown number of groups	Known number of classes
No prior knowledge	Training data is needed
Used to explore data	Used to make new predictions
A form of unsupervised learning	A form of supervised learning

⇩

You can compute it without the need of knowing the correct solution

Exercises with R:

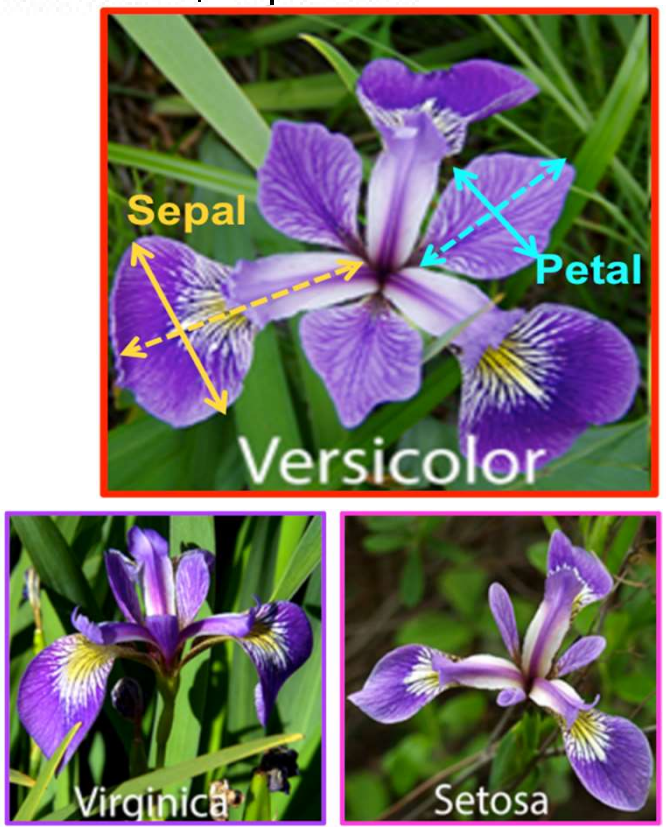
- Read external files
- Data manipulation
- Advanced programming with R
- Graphical visualisation with **ggplot2**

Exercises with R: Read external files

- Data contained in external text files can be imported in R using the following function.
- **read.table()**: This function reads a file in table format and creates a data.frame from it
  read.table(file, header = FALSE, sep = ";", dec = ".", stringsAsFactors = TRUE)
- Arguments of read.table:
  - **file**: to specify the file name and location
  - **header**: a logical value indicating whether the file contains the names of the variables as its first line
  - **sep**: the field separator character. If sep = " " (the default for read.table) the separator is '**white space**'
  - **dec**: the character used in the file for decimal points (the default is **.**)
  - **stringsAsFactors:** a logical value indicating whether the characters should be interpreted as factors (categories) (the default is FALSE)

Exercises with R: Read external files

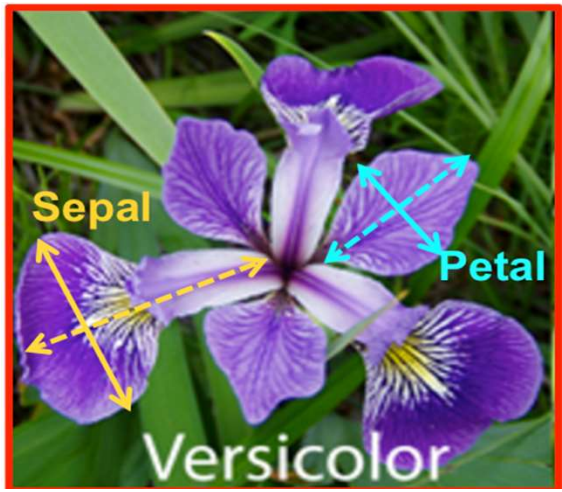
```
"Sepal.Length","Sepal.width","Petal.Length","Petal.width","Species"
5.1,3.5,1.4,0.2,"setosa"
4.9,3,1.4,0.2,"setosa"
4.7,3.2,1.3,0.2,"setosa"
4.6,3.1,1.5,0.2,"setosa"
5,3.6,1.4,0.2,"setosa"
5.4,3.9,1.7,0.4,"setosa"
4.6,3.4,1.4,0.3,"setosa"
5,3.4,1.5,0.2,"setosa"
4.4,2.9,1.4,0.2,"setosa"
4.9,3.1,1.5,0.1,"setosa"
5.4,3.7,1.5,0.2,"setosa"
4.8,3.4,1.6,0.2,"setosa"
```



Exercises with R: Read external files

```
"Sepal.Length","Sepal.width","Petal.Length","Petal.width","Species"
5.1,3.5,1.4,0.2,"setosa"
4.9,3,1.4,0.2,"setosa"
4.7,3.2,1.3,0.2,"setosa"
4.6,3.1,1.5,0.2,"setosa"
5,3.6,1.4,0.2,"setosa"
5.4,3.9,1.7,0.4,"setosa"
4.6,3.4,1.4,0.3,"setosa"
5,3.4,1.5,0.2,"setosa"
4.4,2.9,1.4,0.2,"setosa"
4.9,3.1,1.5,0.1,"setosa"
5.4,3.7,1.5,0.2,"setosa"
4.8,3.4,1.6,0.2,"setosa"
```

Header

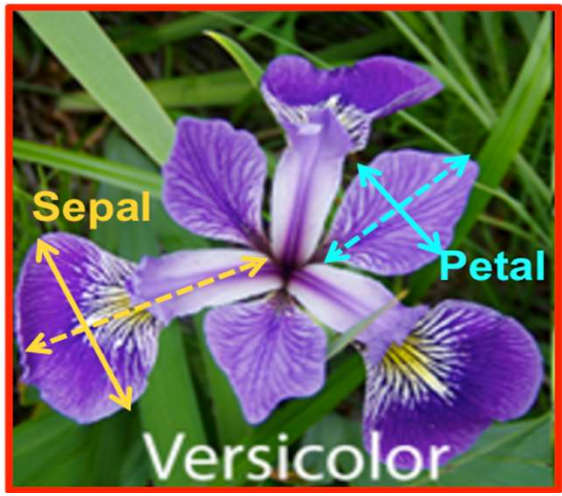


Exercises with R: Read external files

```
"Sepal.Length","Sepal.width","Petal.Length","Petal.width","Species"
5.1,3.5,1.4,0.2,"setosa"
4.9,3,1.4,0.2,"setosa"
4.7,3.2,1.3,0.2,"setosa"
4.6,3.1,1.5,0.2,"setosa"
5,3.6,1.4,0.2,"setosa"
5.4,3.9,1.7,0.4,"setosa"
4.6,3.4,1.4,0.3,"setosa"
5,3.4,1.5,0.2,"setosa"
4.4,2.9,1.4,0.2,"setosa"
4.9,3.1,1.5,0.1,"setosa"
5.4,3.7,1.5,0.2,"setosa"
4.8,3.4,1.6,0.2,"setosa"
```

sep

Header



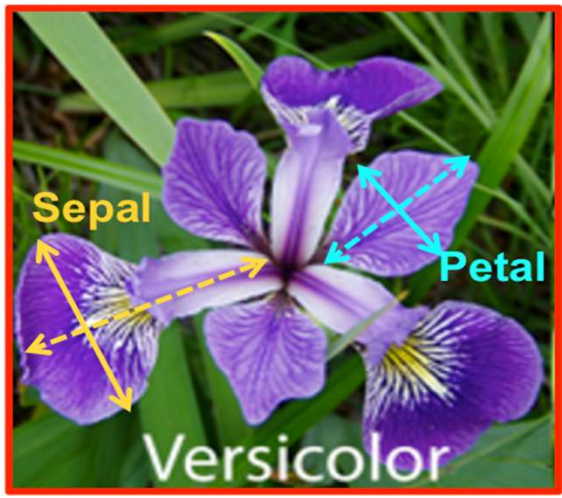
Exercises with R: Read external files

```
"Sepal.Length","Sepal.width","Petal.Length","Petal.width","Species"
5.1,3.5,1.4,0.2,"setosa"
4.9,3,1.4,0.2,"setosa"
4.7,3.2,1.3,0.2,"setosa"
4.6,3.1,1.5,0.2,"setosa"
5,3.6,1.4,0.2,"setosa"
5.4,3.9,1.7,0.4,"setosa"
4.6,3.4,1.4,0.3,"setosa"
5,3.4,1.5,0.2,"setosa"
4.4,2.9,1.4,0.2,"setosa"
4.9,3.1,1.5,0.1,"setosa"
5.4,3.7,1.5,0.2,"setosa"
4.8.3.4.1.6.0.2."setosa"
```

sep

Header

dec



Exercises with R: Read external files

```
#plantData.csv can be downloaded from Stud.IP
>plantData = read.table(file="plantData.csv", header=TRUE, sep=",", dec=".", stringsAsFactors = TRUE)
>plantData
 Sepal.Length   Sepal.Width   Petal.Length   Petal.Width   Species
1          5.1          3.5            1.4           0.2   setosa
2          4.9          3.0            1.4           0.2   setosa
3          4.7          3.2            1.3           0.2   setosa
     …            …            …
     …            …            …
     …            …            …

148        6.5          3.0            5.2           2.0   virginica
149        6.2          3.4            5.4           2.3   virginica
150        5.9          3.0            5.1           1.8   virginica
```

Exercises with R: Manipulate the data sets

- The **summary** function provides a statistical overview of the data
  - *counts* for factors
  - measures such as: *mean, median, min, max …* for numeric columns
  - number of missing values

```
>summary(plantData)

Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
Mean   :5.844   Mean   :3.057   Mean   :3.758   Mean   :1.199
3rd Qu.:6.400   3rd Qu.:3.325   3rd Qu.:5.100   3rd Qu.:1.800
Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
NA's   :2       NA's   :2
```

Exercises with R: Manipulate the data sets

- How to deal with missing values?
  - Ignore them
  - Remove the complete row where a value is missing
  - Replace them with default values i.e. 0 for numerical values
  - Replace them with „realistic“ values i.e. the median or mean values of the column

- In this case the last option is the most suitable for the following analysis.

Exercises with R: Manipulate the data sets

- Replace missing values with mean values of the column

```
# step 1: control if there are missing values in the dataset
# step 2: calculate the mean values of Sepal.Length & Sepal.Width
# step 3: replace missing values with mean values
```

Exercises with R: Manipulate the data sets

- Replace missing values with mean values of the column

```
# step 1: control if there are missing values in the dataset
> summary(plantData)  # na ➔ Sepal.Length & Sepal.Width

# step 2: calculate the mean values of Sepal.Length & Sepal.Width
> meanSL= mean(plantData$Sepal.Length, na.rm = TRUE)
> meanSW= mean(plantData$Sepal.Width, na.rm = TRUE)

# step 3: replace missing values with mean values
> plantData$Sepal.Length[is.na(plantData$Sepal.Length)]=  meanSL
> plantData$Sepal.Width[is.na(plantData$Sepal.Width)] =  meanSW
```

Exercises with R: Programming with R

- R is a full-fledged programming language
- Here we want to introduce three important concepts of programming that will be very useful in the rest of the course
  - **Conditional statements**
  - **Loops**
  - **Functions**

Exercises with R: Programming with R

Conditional statements

- They are used for case differentiation with two cases
- Classical **if…else** statement:

  if(condition) {statement 1} **else**{statement 2}
- **condition** is a logical expression
  - If it is evaluated as true then **statement 1** will be executed
  - If it is evaluated as false then **statement 2** will be executed

- The **else** part is optional

  if(condition) {statement 1}
  - If **condition** is **true** then only **statement 1** will be executed
  - If **condition** is **false** nothing happens

Exercises with R: Programming with R

Conditional statements

- **if** statements can be nested multiple times

```
if(condition 1)          {statement 1}
else if(condition 2)     {statement 2}
else if(condition 3)     {statement 3}
…                        …
else                     {statement n}
```

Exercises with R: Programming with R

Conditional statements

- Example: Calculation of median

```
> x = c(4,5,6,10,12,15,20)
> n = length(x)
> if(n %% 2 == 1){
+       print(x[(n+1)/2]
+  } else{
+       print(1/2 * (x[n/2] + x[(n/2) + 1]))
+ }
```

Exercise: Calculation of median of *Sepal.Length and Sepal.Width* in *plantData using* **if** statements
The values have to be sorted first (use *sort()*)

Exercises with R: Programming with R

Loops

- Loops are used in programming to repeat a specific block of code
- There are three different types of loops in R
  - **for, while** and **repeat**
- The types differ in how the number of repeats is determined

Loops in R	Short description
repeat { statement }	iterate over a block of code multiple number of times
while (test_expression) { statement }	iterate until a specific condition is met.
for (i in M) {statement }	iterate for every **i** $\in$ $\boldsymbol{M}$
next	skip the current iteration of a loop
break	stop the iterations

Exercises with R: Programming with R

```
>for(i in sequence)
+ {
+ statement
+ }
```

- A **for-loop** is used to evaluate a statement for a certain set of indices given in form of a sequence
- There will be as many repetitions as there are elements in the sequence
- In each repetition **i** takes on the value of the sequence at the current position

Exercises with R: Programming with R

Loops

- Example for a simple for-loop

```
>x=c(1:6)
>for( i in x) { print(i) }
 [1] 1
 [1] 2
 [1] 3
 [1] 4
[1] 5
[1] 6
```

- The loop iterates over the six elements in the sequence from 1 to 6
- i takes on the respective value of the sequence

Exercises with R: Programming with R

Loops

- Example for a simple for-loop

```
>x=c(1:6)
>for( i in x) { print(i^2) }
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
```

Exercises with R: Programming with R

Loops

- The value of i can for example be used to access the values of a vector

```
>x = c(4,5,6,10,12)
>for( i in 1:length(x)) { cat("i = ", i, " and x[", i,"] = ", x[i], "\n") }
i =  1  and x[ 1 ] =  4
i =  2  and x[ 2 ] =  5
i =  3  and x[ 3 ] =  6
i =  4  and x[ 4 ] =  10
i =  5  and x[ 5 ] =  12
```

Exercises with R: Programming with R

Loops

- Example: Calculate the mean value of the vector **x** using the **for-loop**

```
>x = c(4,5,6,10,12)
>sum = 0
>mean = 0
>size = length(x)
>for( i in 1:size) { sum = sum + x[i]}
>mean = sum / size
```

Exercises with R: Programming with R

Loops

- **<u>Exercise:</u>** Given two DNA sequences calculate the number of differences between the two sequences

```
>seq1 = c("A", "A", "C", "T", "T", "C ", "G", "G", "A", "C")
>seq2 = c("A", "C", "C", "A", "T", "C ", "T", "G", "T", "C")
>diff = 0
>for( i in 1:length(seq1)) {
+ if(seq1[i] != seq2[i]) {diff = diff +1}
+}
>print(diff)
# calculate the similarity between both sequences
> similarity= (length(seq1)-diff)/length(seq1)
```

Exercises with R: Programming with R

Functions

- Functions are used to logically break your code into simpler parts
- The parts can be saved with a specific name
- This reduces the amount of code you need to write by reusing existing code
- Makes the code easier to maintain and understand
- Examples for already existing functions in R
  - mean()
  - min()
  - max()
  - read.table()
  - etc.

Exercises with R: Programming with R

Functions

- In order to create your own function, you use the reserved word **function**

```
>func_name = function(argument){
+       statement
+}
```

- The statement between the curly braces forms the body of the function
- This body is executed when you call the function
- You call the function using **func_name()**
- **argument** is a list of parameters that the function takes

Exercises with R: Programming with R

Functions

- Toy example : Power function

```
>pow = function(x,y){
+       result = x^y
+       cat("The result is ", result) #The result is 8
+}
>pow(2,3)
```

- In order to return a value from the function you use the **return()** function

```
>pow = function(x,y){
+       result = x^y
+       return(result)
+}
>pow(2,3)
[1] 8
```

Exercises with R: Programming with R

Functions

- **Exercise:** Write a function to calculate the similarity score between two DNA sequences

```
>similarity = function(seq1, seq2){
+       sim = 0
+       for(i in 1 : length(seq1)){
+                       if(seq1[i] == seq2[i]) {
+                               sim = sim +1
+                       }
+       }
+        ratio= sim/length(seq1)
+       return(ratio)
+}
>similarity (c("A","C","T"),c("A","T","T"))
[1] 0.666
>similarity(c("C","C","T"),c("A","T","T"))
[1] ?
> similarity(c("A","A","C","T","T","C","G","G","A","C"), c("A","C","C","A","T","C", "T","G","T","C"))
[1] ?
```

Graphical visualisation with ggplot2

Graphical visualisation with ggplot2

- R has a special function for each kind of plot (i.e. histogram, boxplot, pie chart,…)
- **ggplot2** is an R package that offers an alternative way to create plots
- It uses a "grammar"-based approach to build plots component-by-component instead of using premade graphs
- A grammar defines the rules for structuring elements
  - Languages: words and phrases are combined into meaningful sentences
  - **ggplot2**: mathematics and aesthetic elements are combined into a meaningful plot
- First step is to install the package(one time only) and then load it into R (every session)

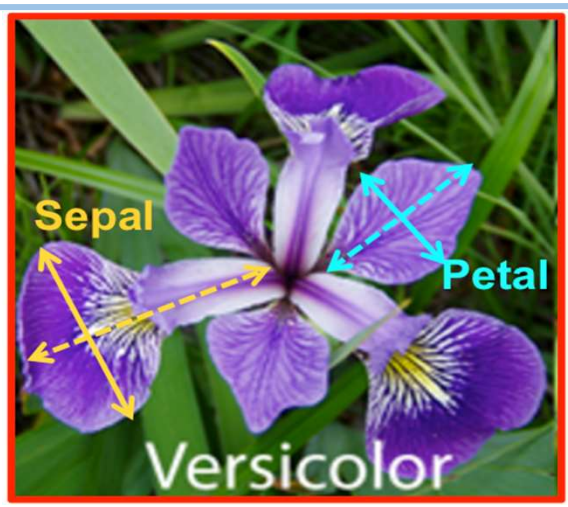
```
>install.packages("ggplot2")
>library(ggplot2)
```

Graphical visualisation with ggplot2

Elements of grammar of graphics

- **Data**: variables mapped to aesthetic features of the graph
- **Geoms**: objects/shapes on the graph
- **Stats**: statistical transformations (e.g. mean, confidence intervals)
- **Scales**: mappings of aesthetic values to data values for legends and axes
- **Coordinate systems**: plane on which data are mapped
- **Faceting**: splitting the data into subsets to create variations of the same graph

Graphical visualisation with ggplot2



- Data must be stored in a **data.frame** in order to plot it with **ggplot2**
- For example the previously used **plantData** dataset

> head(plantData)

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Graphical visualisation with ggplot2

1. Using **plot and lines functions**,

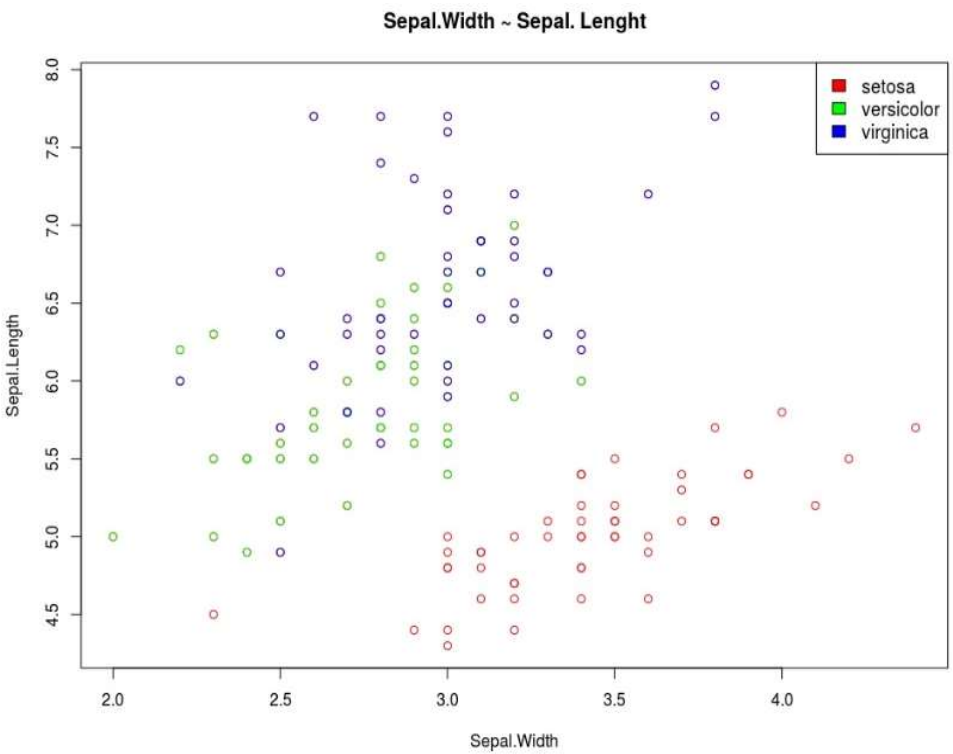
- create a **scatterplot** between **sepal width and length** of all samples
- color the points of each species individually
- Use legend function and write color-codes of species on the topright side

2. Create the same graphic using **ggplot2**

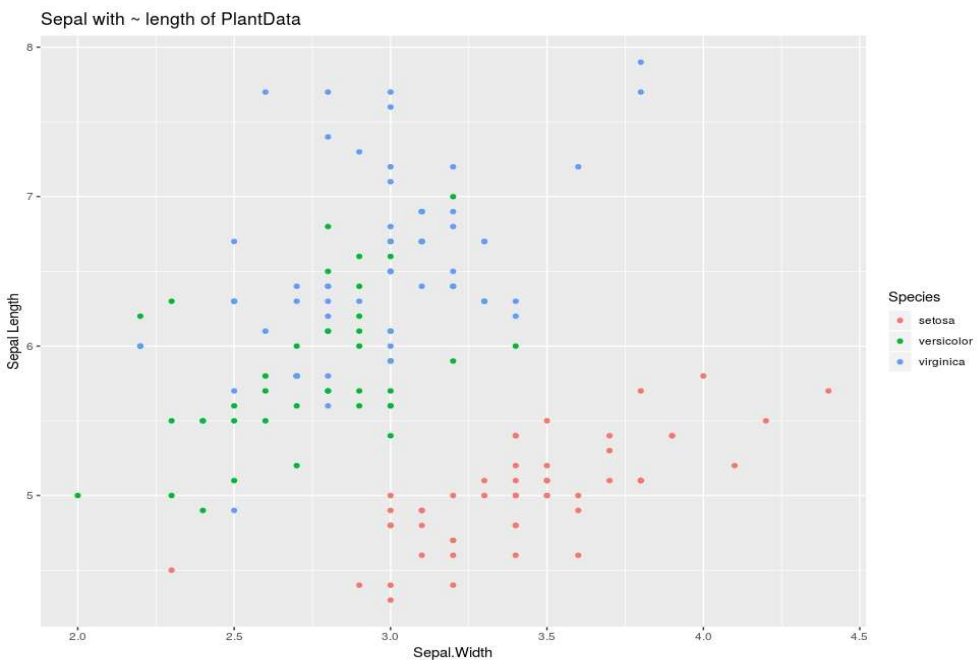
GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN

Graphical visualisation with ggplot2

With **plot and lines functions**,

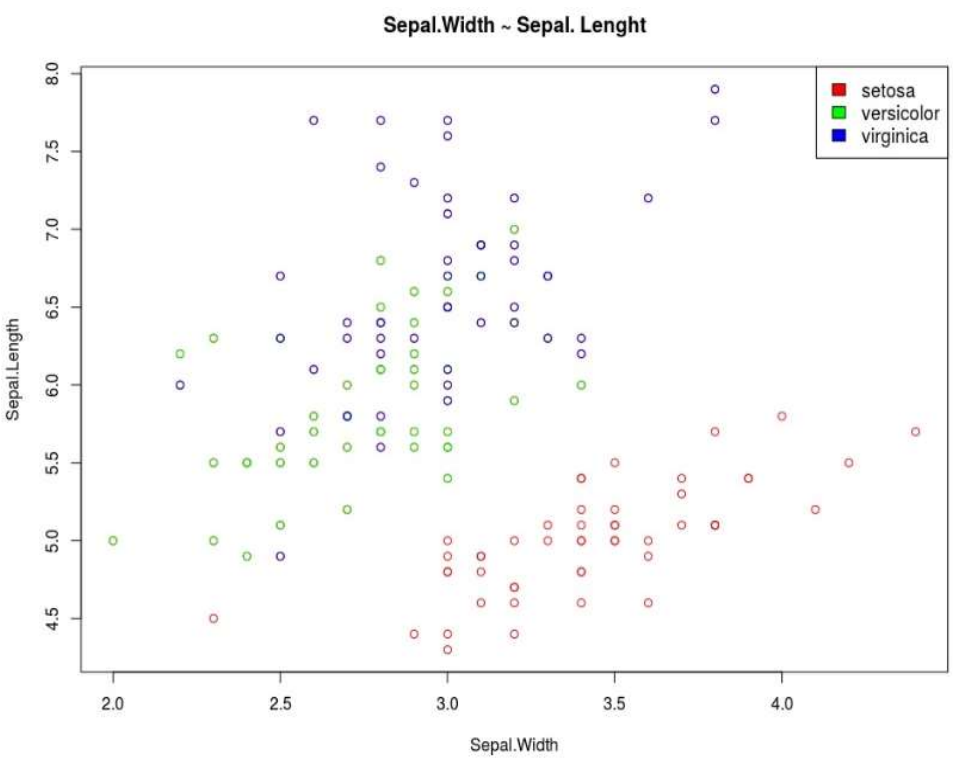


With **ggplot2**



Graphical visualisation with ggplot2

With **plot and lines functions**,

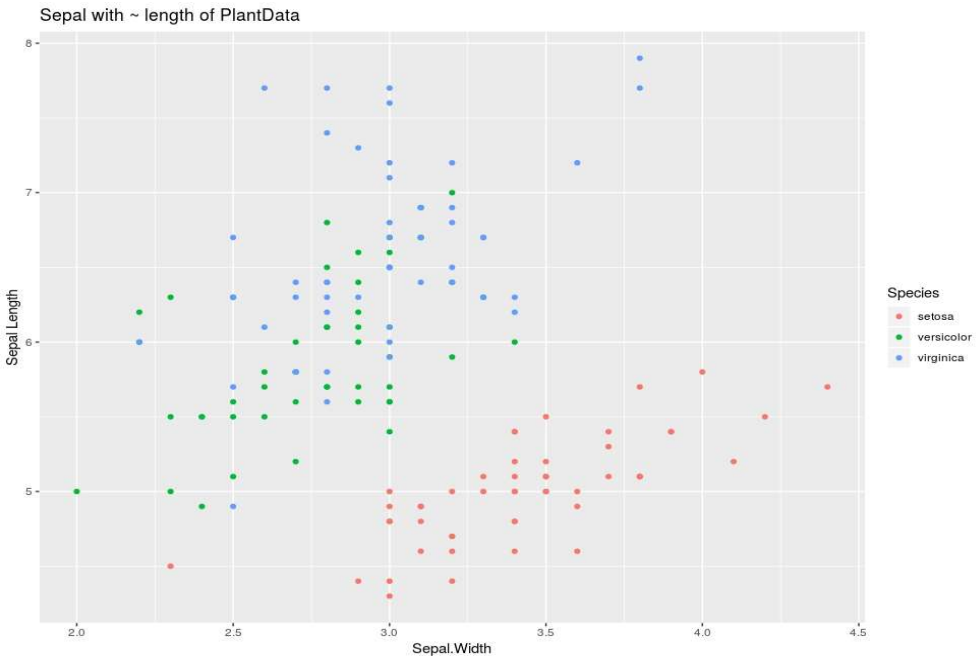



```
plot(x=plantData$Sepal.Width, y=plantData$Sepal.Length,
     main="Sepal.Width ~ Sepal. Lenght",
     xlab="Sepal.Width",
     ylab="Sepal.Length", col="red")

lines(x=plantData$Sepal.Width[plantData$Species=="versicolor"],
      y=plantData$Sepal.Length[plantData$Species=="versicolor"],
      type="p", col="green")

lines(x=plantData$Sepal.Width[plantData$Species=="virginica"],
     y=plantData$Sepal.Length[plantData$Species=="virginica"],
     type="p", col="blue")

legend("topright", legend=c("setosa", "versicolor" , "virginica"),
         fill=c("red", "green", "blue"), cex = 1.1)
```

Graphical visualisation with ggplot2

With **ggplot2**

```
ggplot(data = plantData,
        aes(x = Sepal.Width,y = Sepal.Length)) +
       geom_point(aes(color = Species)) +
       labs(x = "Sepal.Width",  y = "Sepal Length",
            title = "Sepal with ~ length of PlantData")
```



The command consists of multiple components which we will go through step-by-step

Graphical visualisation with ggplot2

First example with ggplot2: Plot the sepal width and length of all samples

> ggplot(data = plantData, aes(x=Sepal.Width,y = Sepal.Length))

- Each plot begins with the **ggplot()** function (**!** not **ggplot2**, the name of the package)
  - ***data*** specifies the dataset the plot is going to be based upon
  - ***aes()*** is a function that is used to specify aesthetics i.e. the visually perceivable components
    - In this case we specify which variables appear on the x-axis and the y-axis

- Results in the "**background**" of the plot

Graphical visualisation with ggplot2

```
>ggplot(data = plantData, aes(x=Sepal.Width,y = Sepal.Length)) +   geom_point()
```

- Additional components (called layers) are added to the **ggplot()** function with the character **+**
- A layer consists of graphics created by either a **geom** or a **stat** function
- **geom_point()** simply creates a **scatterplot** based on the data and aesthetics defined in **ggplot()**

Graphical visualisation with ggplot2

```
>ggplot(data = plantData, aes(x=Sepal.Width,y = Sepal.Length)) +
        geom_point(aes(color = Species))
```

- Additional aesthetics can be specified in each layer
- *aes(color = Species)* ➔ **color the points** based on the species to which they belong
- Legend for the **colors** is created automatically

Graphical visualisation with ggplot2

```
>ggplot(data = plantData, aes(x=Sepal.Width,y = Sepal.Length)) +
        geom_point(aes(color = Species)) +
        labs(x = "Index", y = "Sepal Length“, title = “Sepal lengths of Iris”)
```

- **labs()** is used to modify the labels of axis and legends as well as title and caption of the plot
- Here we change the labels for the x-axis and the y-axis
- Additionally we add a title for the plot