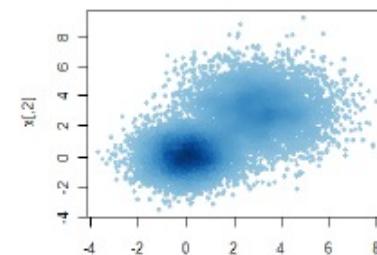
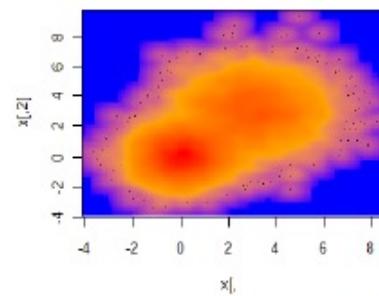
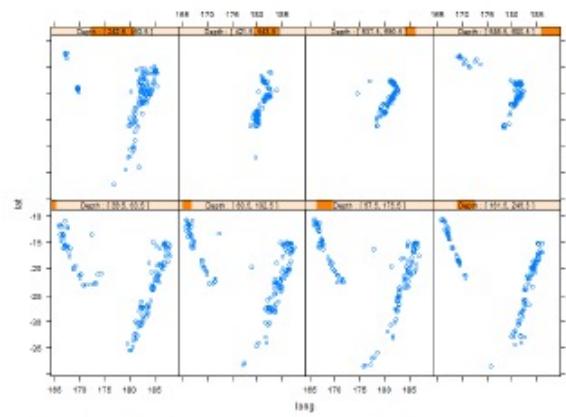
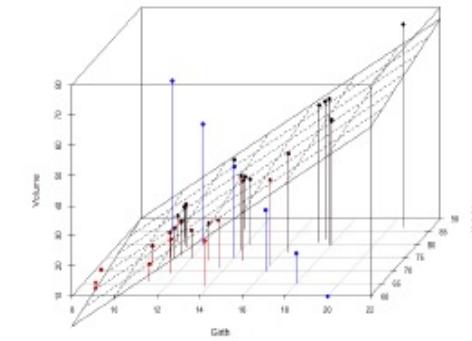
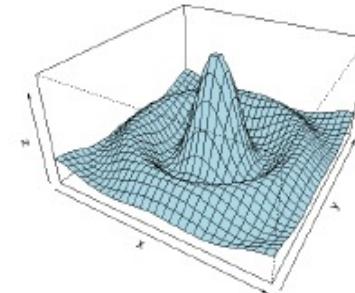
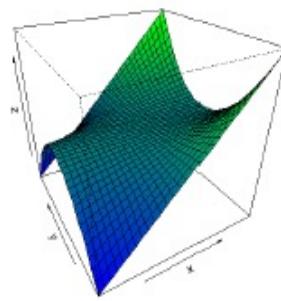
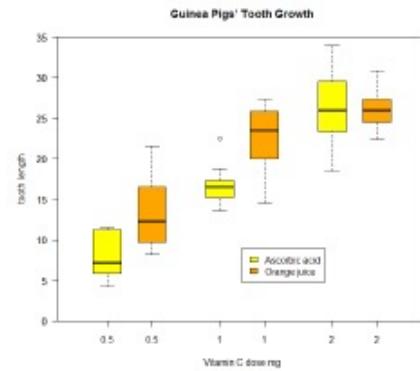


$u^b$

b  
UNIVERSITÄT  
BERN

OESCHGER CENTRE  
CLIMATE CHANGE RESEARCH

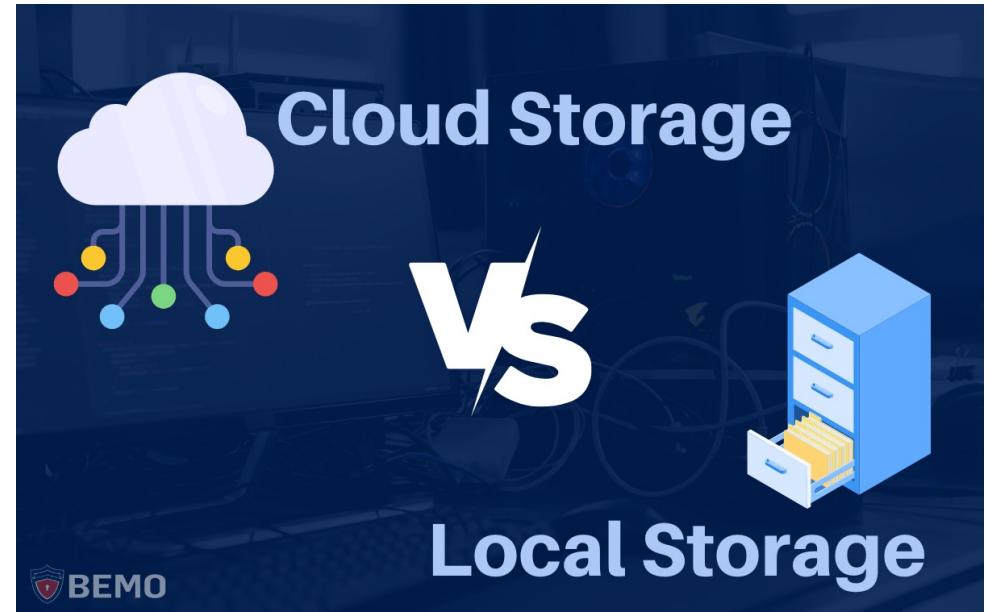
# Computer Basics and R



# Computer Basics

## Dateisystem

- > Dateisystem und Mountpunkte in Finder und Konsole zeigen



# Computer Basics

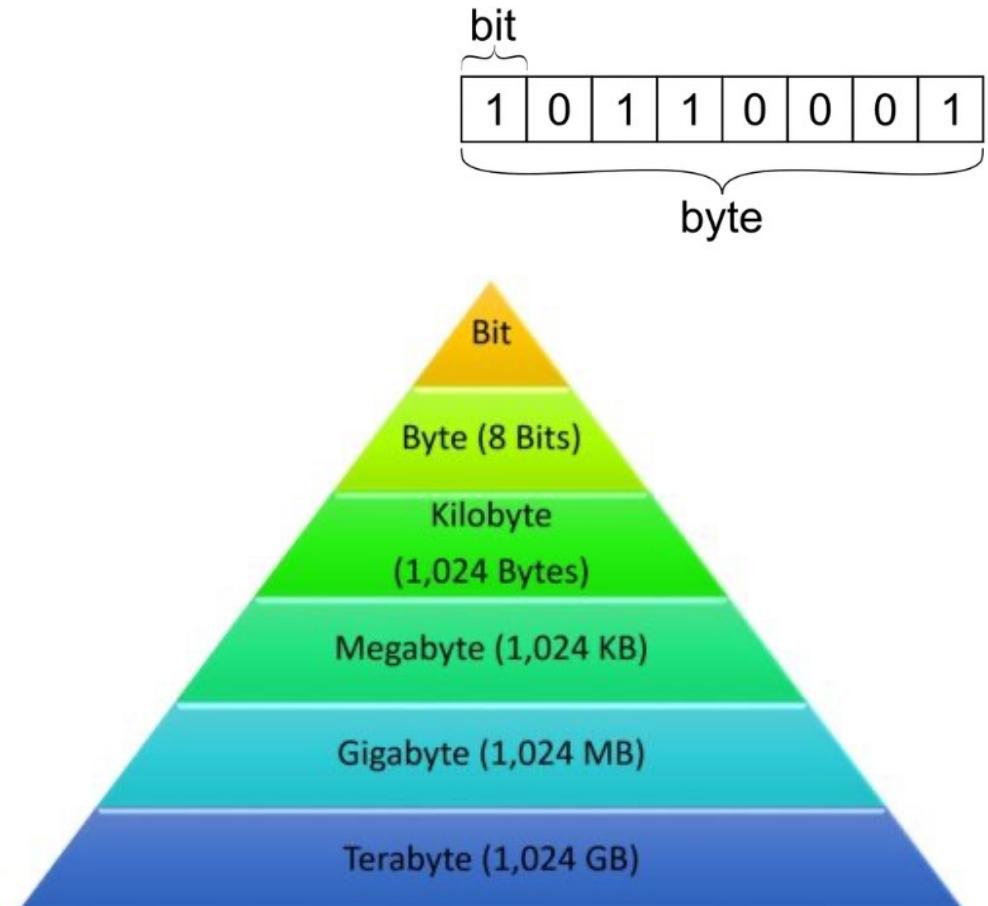
1 Bit: 1/0

1 Byte: kodiert einen Buchstaben

1 Megabyte ?

1 Gigabyte ?

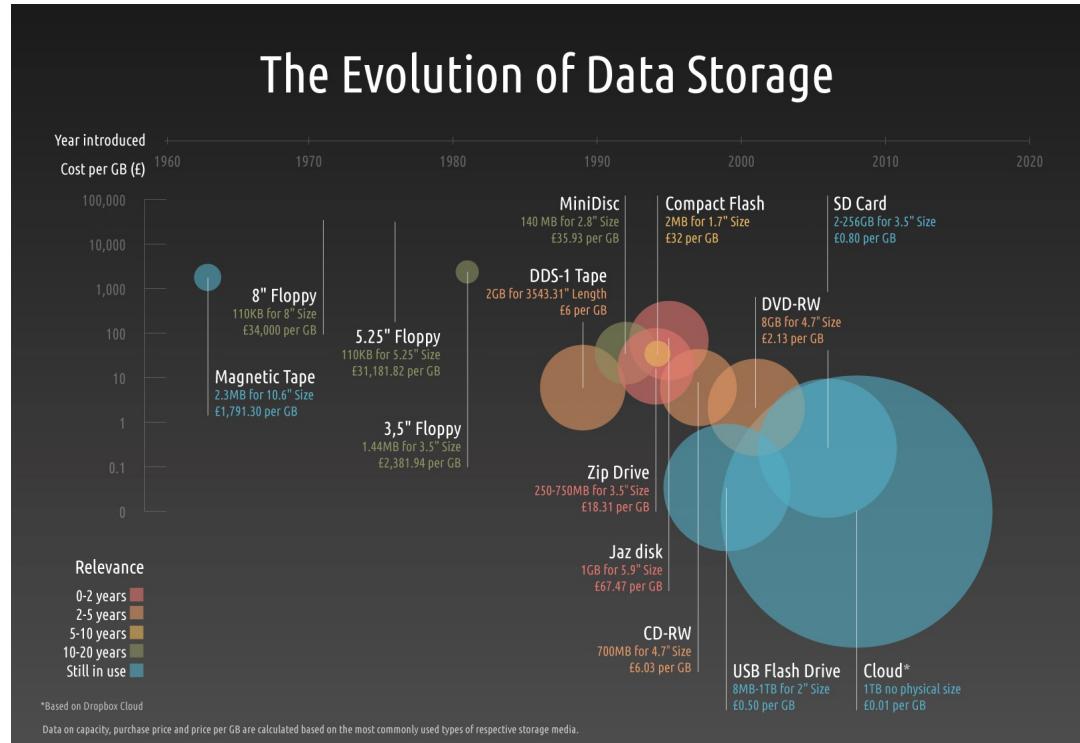
1 Terabyte ?



# Computer Basics

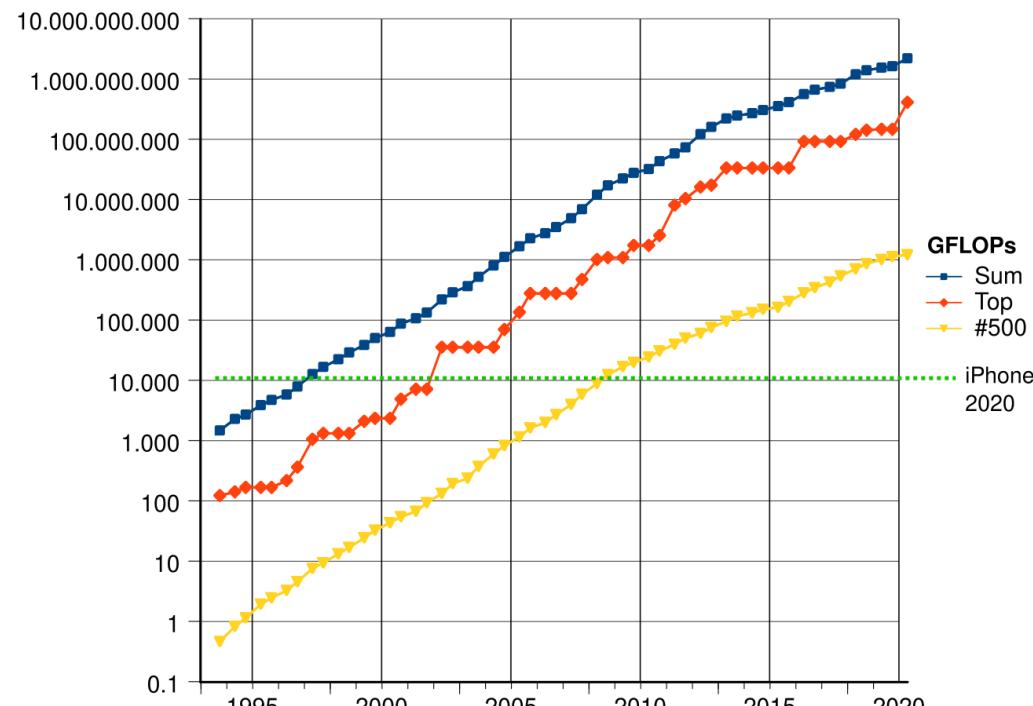
## Datengrößen:

- > 100 Jahresmittel der Temperatur an einer Station wenige KB
- > 100 Jahre Tagesmittel einer Station ca. 1 MB
- > Zeitschritt eines Wettervorhersagemodells multivariat bei 1 km räumlicher Auflösung ca. 2 TB

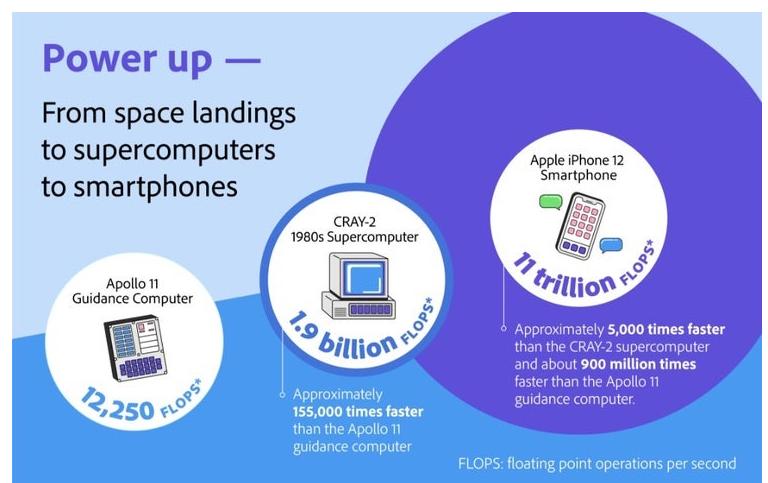


# Computer Basics

Computerleistungen einschätzen (iPhone vs Supercomputer)



FLOPS = Floating point operations per second



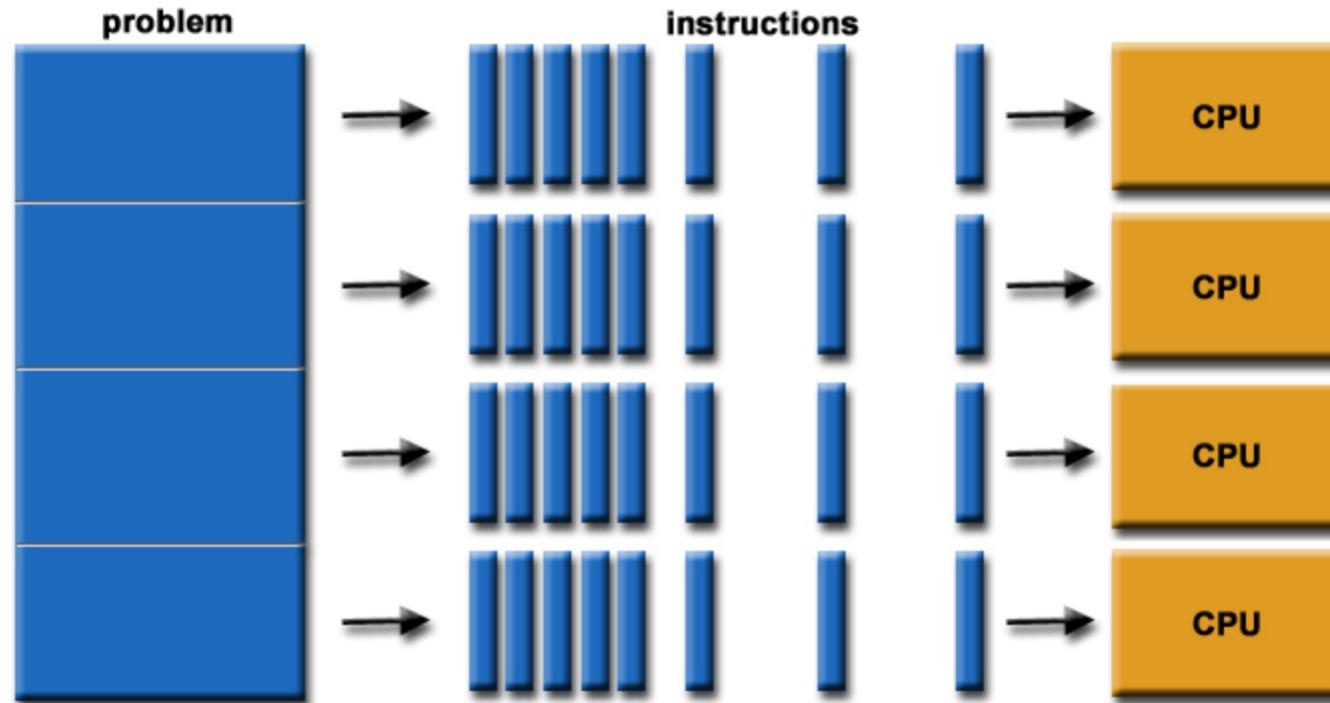
# Cloud Computing

- › benutzt ihr die Online Office Version im Browser oder eine auf eurem Computer installierte und wieso?



# Computer Basics

## CPU Kerne / Parallelisierung



Flaschenhals: Rechenzeit oder Schreibzeit

# Computer Basics

## Speicherplatzoptimierung?

- > Zwischenschritte speichern?
- > Am Ende wieder löschen?
- > Dateiformate/Komprimierung

## Codeoptimierung ?

- > Zeit zum Code schreiben / optimieren
- > Laufzeit des Codes
- > Anzahl der Codeläufe

# Computer Basics

## Organisation, Dateiformate und Dateinamen

- > KEINE Leerzeichen
- > KEINE Sonderzeichen wie Umlaute
- > Betriebssystemunabhängigkeit
- > ACHTUNG: Länderspezifische Besonderheiten wie Dezimalkomma vs Dezimalpunkt

### 1 FIGURE OUT LOCATION

Files can be stored in desktop, documents, downloads, or shared drive.



### 2 ORGANIZE BY FOLDERS

Organize by dates, departments, or events.



### 3 ORGANIZE BY SUBFOLDERS

Use date or project type.



### 4 FORMAT AND GROUP FILES

Format based off file type like excel, doc, pdf, jpeg, and png.



### 5 NAME FILES STRATEGICALLY

Think about how you would search for it in the future.

### 6 DOCUMENT THE PROCESS

Create a Standard Operating Procedure (SOP's) for consistency.



### 7 MAINTENANCE IS KING

Have a routine to update, move, and delete files regularly.

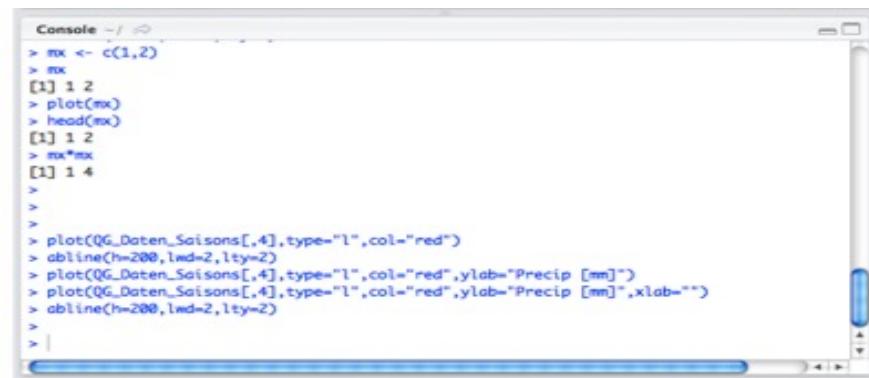


### 8 OTHER BEST PRACTICES

File immediately and do not store files on your desktop.

# Was ist R?

- > Einfache Programmiersprache zur Datenanalyse, statistischen Auswertung und Visualisierung, d.h.
- > geeignet für jegliche Datenbearbeitung
- > inklusive Abbildungen jeder Art, auch Karten
- > "Einfach" da kein Compilieren notwendig ist
- > Freie Software
- > Kein offizielles Menü-System -> Kommando basiert + Text editor
- > 'packages' mit ständig von Benutzer ergänzten Zusatzfunktionen frei im Internet verfügbar
- > Gute Hilfe im Netz



The screenshot shows an R console window with the following text:

```
Console - / 
> mx <- c(1,2)
> mx
[1] 1 2
> plot(mx)
> head(mx)
[1] 1 2
> mx*mx
[1] 1 4
>
>
> plot(QG_Daten_Saisons[,4],type="l",col="red")
> abline(h=200,lwd=2,lty=2)
> plot(QG_Daten_Saisons[,4],type="l",col="red",ylab="Precip [mm]")
> plot(QG_Daten_Saisons[,4],type="l",col="red",ylab="Precip [mm]",xlab="")
>
>
```

# Literatur: <https://r4ds.had.co.nz>

Welcome

☰    ⚅    A    ⌂    i    R for Data Science

1 Introduction

I Explore

2 Introduction

3 Data visualisation

4 Workflow: basics

5 Data transformation

6 Workflow: scripts

7 Exploratory Data Analysis

8 Workflow: projects

II Wrangle

9 Introduction

10 Tibbles

11 Data import

12 Tidy data

13 Relational data

14 Strings

15 Factors

16 Dates and times

III Program

17 Introduction

## R for Data Science

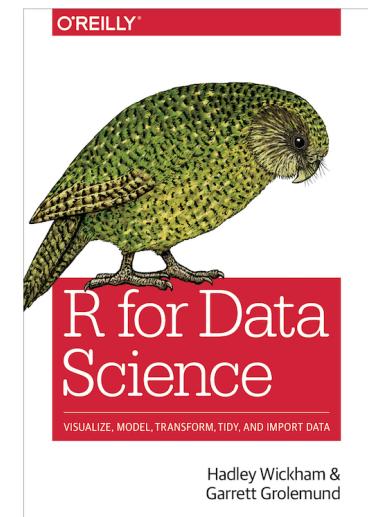
Garrett Grolemund

Hadley Wickham

## Welcome

This is the website for “R for Data Science”. This book will teach you how to do data science with R: You’ll learn how to get your data into R, get it into the most useful structure, transform it, visualise it and model it. In this book, you will find a practicum of skills for data science. Just as a chemist learns how to clean test tubes and stock a lab, you’ll learn how to clean data and draw plots—and many other things besides. These are the skills that allow data science to happen, and here you will find the best practices for doing each of these things with R. You’ll learn how to use the grammar of graphics, literate programming, and reproducible research to save time. You’ll also learn how to manage cognitive resources to facilitate discoveries when wrangling, visualising, and exploring data.

This website is (and will always be) **free to use**, and is licensed under the [Creative Commons Attribution-NonCommercial-NoDerivs 3.0](#) License. If you’d like a **physical**



Editor, hier wird programmiert und der Code dann in der Konsole ausgeführt

## R-Studio (graphische Benutzeroberfläche)

Konsole / Kommandozeile

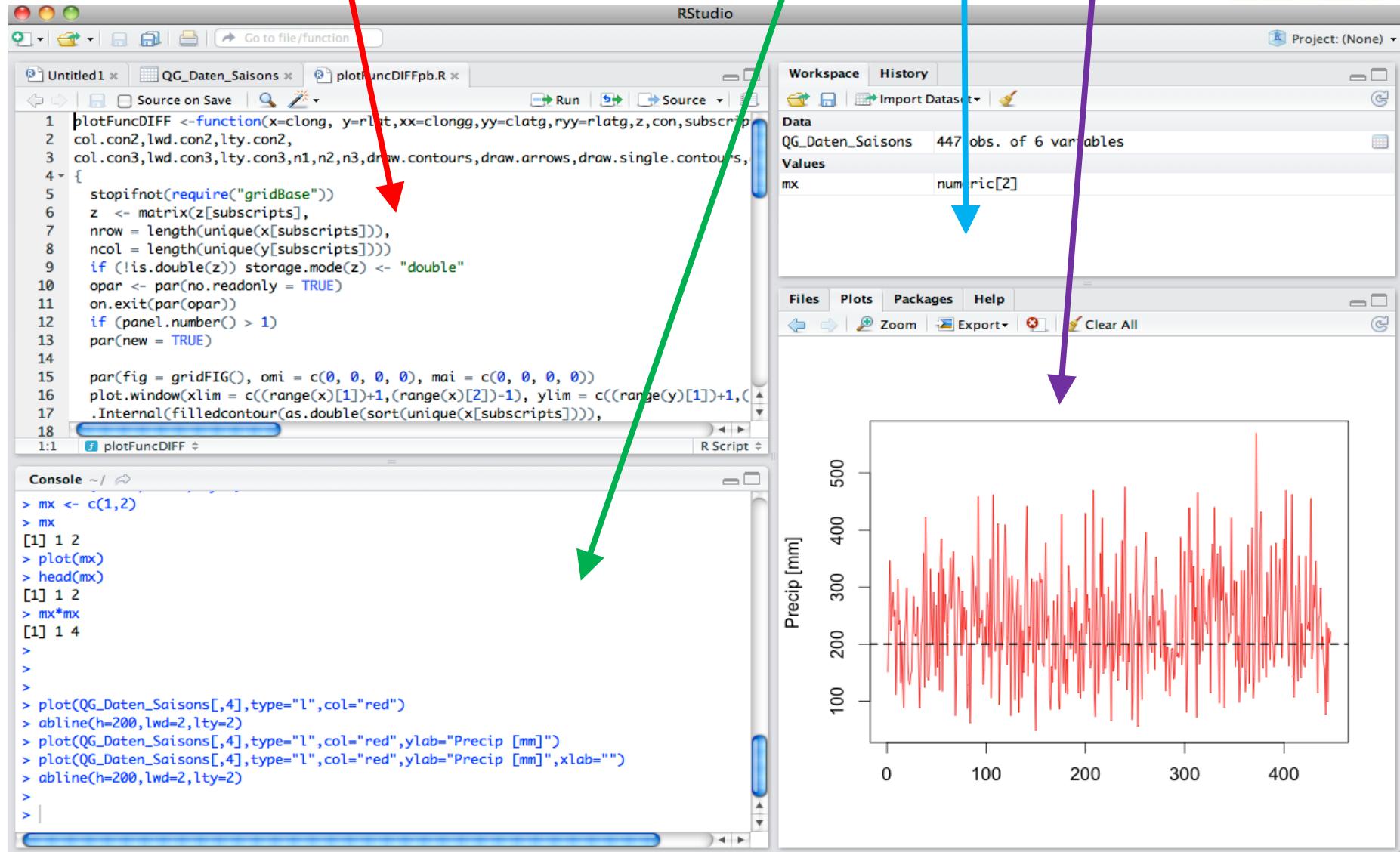
Plots/Hilfe

*u*<sup>b</sup>

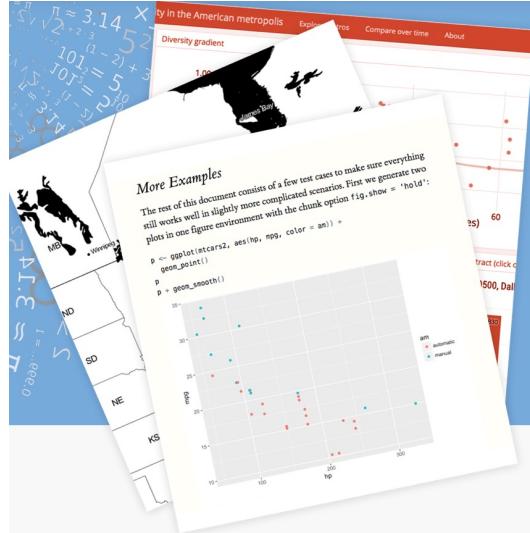
UNIVERSITÄT  
BERN

OESCHGER CENTRE

RCH



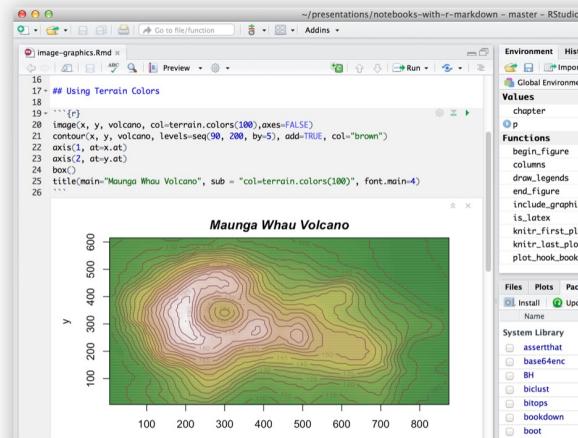
# R Markdown or Notebooks



R Markdown documents are fully reproducible. Use a productive [notebook interface](#) to weave together narrative text and code to produce elegantly formatted output. Use [multiple languages](#) including R, Python, and SQL.

Analyze. Share. Reproduce.

Your data tells a story. Tell it with R Markdown.  
Turn your analyses into high quality documents, reports, presentations and dashboards.



R Markdown supports dozens of static and dynamic output formats including [HTML](#), [PDF](#), [MS Word](#), [Beamer](#), [HTML5 slides](#),

# R Markdown

Cheat Sheet  
learn more at [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com)

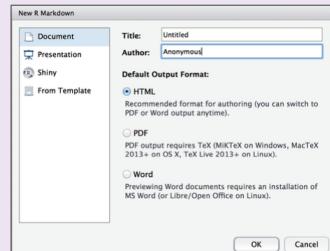
rmarkdown 0.2.50 Updated: 8/14



## 2. Open File

Start by saving a text file with the extension .Rmd, or open an RStudio Rmd template

- In the menu bar, click **File ▶ New File ▶ R Markdown...**
- A window will open. Select the class of output you would like to make with your .Rmd file
- Select the specific type of output to make with the radio buttons (you can change this later)
- Click OK



## 4. Choose Output

Write a YAML header that explains what type of document to build from your R Markdown file.

### YAML

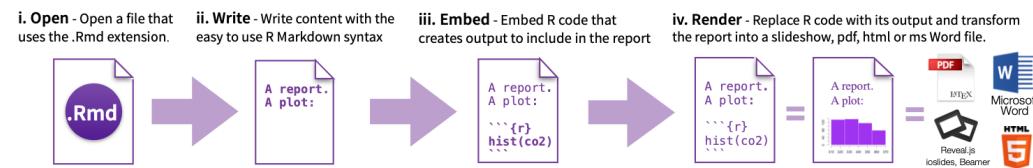
A YAML header is a set of key: value pairs at the start of your file. Begin and end the header with a line of three dashes (---)

```
title: "Untitled"
author: "Anonymous"
output: html_document
---
This is the start of my report. The above is metadata saved in a YAML header.
```

The RStudio template writes the YAML header for you

## 1. Workflow

R Markdown is a format for writing reproducible, dynamic reports with R. Use it to embed R code and results into slideshows, pdfs, html documents, Word files and more. To make a report:



## 3. Markdown

Next, write your report in plain text. Use markdown syntax to describe how to format text in the final report.

### syntax

Plain text  
End a line with two spaces to start a new paragraph.  
\*italics\* and \_italics\_  
\*\*bold\*\* and \_\_bold\_\_  
superscript^2^  
~~strikethrough~~  
[link](www.rstudio.com)

```
# Header 1
## Header 2
### Header 3
#### Header 4
##### Header 5
###### Header 6

endash: --
emdash: ---
ellipsis: ...
inline equation: $A = \pi r^2$
image: 

horizontal rule (or slide break):
***
```

### becomes

Plain text  
End a line with two spaces to start a new paragraph.  
italics and italics  
bold and bold  
superscript<sup>2</sup>  
strikethrough  
[link](#)

## Header 1

## Header 2

## Header 3

## Header 4

## Header 5

## Header 6

endash: –  
emdash: —  
ellipsis: ...  
inline equation:  $A = \pi r^2$

image:

horizontal rule (or slide break):

block quote

# R-Struktur, Objektarten

Datentypen: numerisch, alphanumerisch (Text), logisch (TRUE/FALSE)

## Objektarten

**Vektoren**: kein Mischen von Datentypen

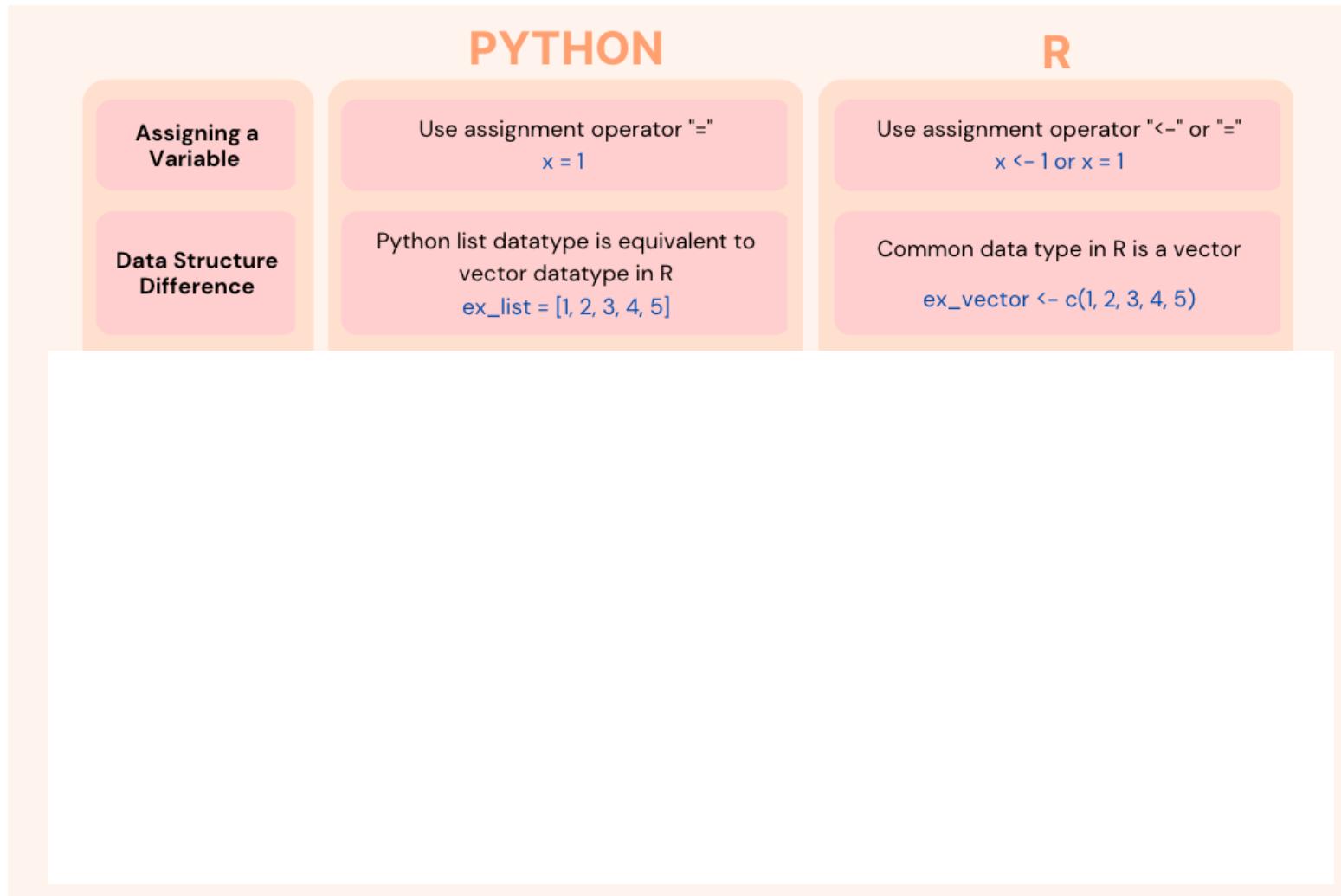
**Data Frames ('Tabellen')**: Mischen möglich

**Matrizen**: kein Mischen von Datentypen

Listen: mehrere Vektoren, Data Frames, Matrizen können in Listen zusammengefasst werden

**Objekte erzeugt** man in dem man einen Objekt-Namen vergibt und Werte mit <- oder = zuweist!

# Python vs R



ACHTUNG: copy/paste funktioniert NICHT!!!



---

b  
UNIVERSITÄT  
BERN

OESCHGER CENTRE  
CLIMATE CHANGE RESEARCH

## Objektart: Vektor

Numerische Vektoren

```
> a <- c(4, 2, 7) # c Funktion steht für combine or concatenate  
# Parameter von Funktionen immer mit RUNDEN Klammern
```

```
> a
```

```
[1] 4 2 7
```

```
> b <- c(3.1, 5, -0.7)
```

```
> c <- c(a, b)
```

```
> c
```

```
[1] 4.0 2.0 7.0 3.1 5.0 -0.7
```

**Einfache oder doppelte Anführungszeichen funktionieren!**

Alphanumerische Vektoren

```
> station <- c('Bern', 'Biel', 'Olten')
```

Logische Vektoren

```
> e <- c(TRUE, FALSE, TRUE)
```

## Objektart: Vektor

Logische Vektoren

```
> f <- 1:5  
[1] 1 2 3 4 5  
> f >= 3  
[1] FALSE FALSE TRUE TRUE TRUE
```

Vergleichsoperationen <, <=, >, >=, ==, !=

Operationen & (und), | (oder), ! (nicht)

```
> g <- (f>2) & (f<5)  
> g
```

```
[1] FALSE FALSE TRUE TRUE FALSE
```

## Objektart: Vektor

> `seq(0, 3, by=0.5)` ... erzeugt Sequenz von 0 bis 3 im 0.5 Abstand  
[1] 0.0 0.5 1.0 1.5 2.0 2.5 3.0

> `rep(5, 3)` ... wiederholt 3x den Wert 5  
[1] 5 5 5

> `rep(c(0, 3), length=9)` ) ... wiederholt 0 u. 3 bis Vektor 9 Elemente hat  
[1] 0 3 0 3 0 3 0 3 0

> `rep(c(0:3), length=9)` ... wiederholt 0,1,2,3 bis Vektor 9 Elem. hat  
[1] 0 1 2 3 0 1 2 3 0 1

# Objektart: Vektor

- > Wichtige grundlegende Funktionen

Funktion, Beispiel	Bedeutung
<i>length(a)</i>	Länge, Anzahl Elemente
<i>sum(a)</i>	Summe aller Elemente
<i>mean(a)</i>	arithmetisches Mittel der Elemente
<i>var(a)</i>	empirische Varianz
<i>sd(a)</i>	emp. Standardabweichung
<i>range(a)</i>	Wertebereich
<i>min(a), max(a)</i>	Minimum, Maximum Wert des Vektors

# Python vs R

PYTHON	R
<p>Chaining operations</p> <p>Method chains with the "." operator <code>df.head(n)</code> or <code>df.describe()</code></p>	<p>Can chain operations using following symbol: "%&gt;%" <code>df %&gt;% head(n)</code> or <code>df %&gt;% summary()</code></p>

<https://towardsdatascience.com/the-starter-guide-for-transitioning-your-python-projects-to-r-8de4122b04ad>

## Objektart: Data Frame

Vektoren unterschiedlicher Datentypen als Tabelle (=Data Frame) zusammenfassen

> *Dat <- data.frame(station, a, b)* ... Data Frame mit Namen 'Dat' erzeugen

	<i>station</i>	<i>a</i>	<i>b</i>	... Spaltennamen (z.B. <i>a</i> ist Bewölkung, <i>b</i> ist Temperatur zu einem Zeitpunkt)
1	<i>Bern</i>	4	3.1	
2	<i>Biel</i>	2	5.0	
3	<i>Olten</i>	7	-0.7	

# Elemente aus Objekten auswählen

## Vektor [ ELEMENTNUMMER ]

> *a[3]; a[1:2]* ... 3. Element; ... 1. bis 2. Element (also 1. und 2.)

## Data Frame/Matrix [ ZEILE , SPALTE ]

> *Dat[1,1]* ... 1. Zeile, 1. Spalte

> *Dat[c(1,2),1]* ... 1. und 2. Zeile der 1. Spalte

> *Dat[1:2,3]* ... 1. bis 2. Zeile der 3. Spalte

> *Dat[,3]* ... alle Zeilen der 3. Spalte (*leer = alle*)

> *Dat[-3,]* ... alle Zeilen bis auf die 3. Zeile, alle Spalten

oder mit Spalten/Zeilennamen

> *Dat[1:2,'b']*

[1] 3.1 5.0

# Python vs R

## PYTHON

## R

### Indexing and slicing

Indexing starts from 0, inclusive of the start index and excludes the end index.  
`ex_list[0]` and `ex_list[0:2]`  
`output: 1`      `output: [1, 2]`

Indexing starts from 1, Indexing is inclusive of both start and end index  
`ex_vector[1]` and `ex_vector[1:2]`  
`output: 1`      `output: 1, 2`

## Hilfe in R

mit **?Funktionsname** oder **help(Funktionsname)**

```
> ?seq  
> ??histogram  
> help(sum)  
> example(histogram)  
> ...
```

Hilfe im Netz: **GOOGLE, CHATBOTS!**

## R als Taschenrechner

> *2 + 5*

[1] 7

> *(2:5)^2* ... auf Vektoren Operationen elementweise angewandt;

[1] 4 9 16 25 ^ oder \*\* =Potenzfunktion

> *(a+1) \* b* ... Klammern wie üblich

[1] 15.5 15.0 -5.6 11.7

> *abs(b)* ... Absolutwert (Betrag)

[1] 3.1 5.0 0.7 1.3

> *exp(a)* ... Exponentialfunktion ( $e^a$ )

[1] 54.598150 7.389056 1096.633158 2980.957987

> *log(a)* ... Natürlicher Logarithmus ( $\exp(1)^{\log(a)}=a$ )

[1] 1.3862944 0.6931472 1.9459101 2.0794415

> *sqrt(a)* ... Wurzel (Square root)

# Neues R-Skript oder R Notebook erstellen und umbedingt speichern!!!

*u*<sup>b</sup>

<sup>b</sup>  
UNIVERSITÄT  
BERN

OESCHGER CENTRE

RCH

The screenshot shows the RStudio interface. The top menu bar includes 'File', 'Edit', 'View', 'Code', 'Tools', 'Help', and 'Project: (None)'. The main area consists of several panes:

- Script Editor:** Shows the code for 'plotFuncDIFFpb.R'. A green arrow points to the 'Save' icon (disk icon with a plus sign), and a red arrow points to the 'Run' icon (play button).
- Console:** Shows the R session history, including commands like `mx <- c(1,2)` and `plot(mx)`.
- Workspace:** Displays the dataset 'QG\_Daten\_Saisons' with 447 observations and 6 variables, and a variable 'mx' of type numeric[2].
- Plots:** A line plot titled 'Precip [mm]' showing precipitation over time, with a horizontal dashed line at 200 mm.

# Verzeichnis finden

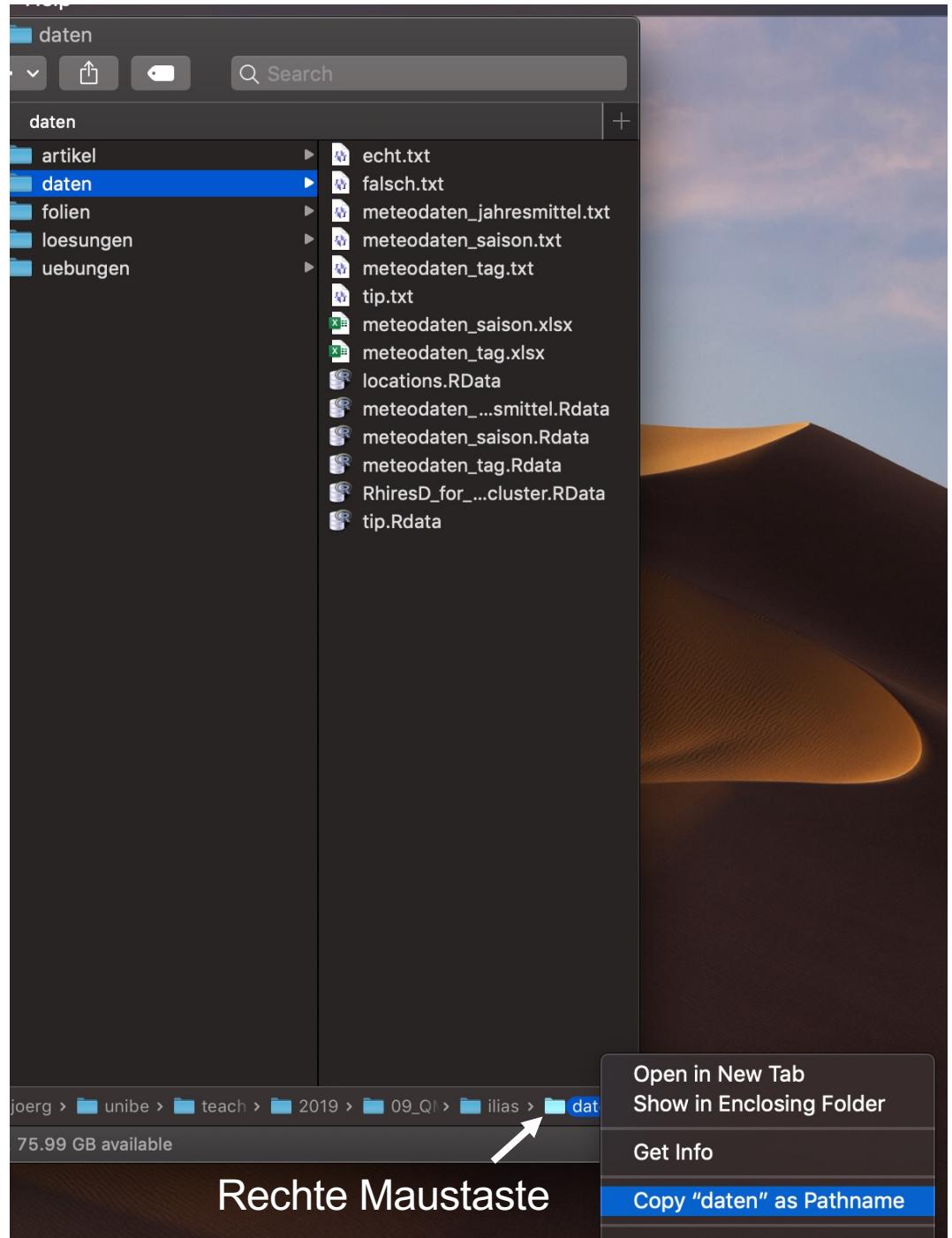
1. Im Datei "Explorer" (Windows) oder "Finder" (Mac) das Verzeichnis finden, wo ihr eure Dateien zu dieser Veranstaltung speichert
2. Beispieldaten von Ilias in dieses R Verzeichnis speichern/runterladen
3. Vollständigen Pfadnamen kopieren (siehe Abb. rechts)

Mac: '/Users/name/quant\_meth...' )

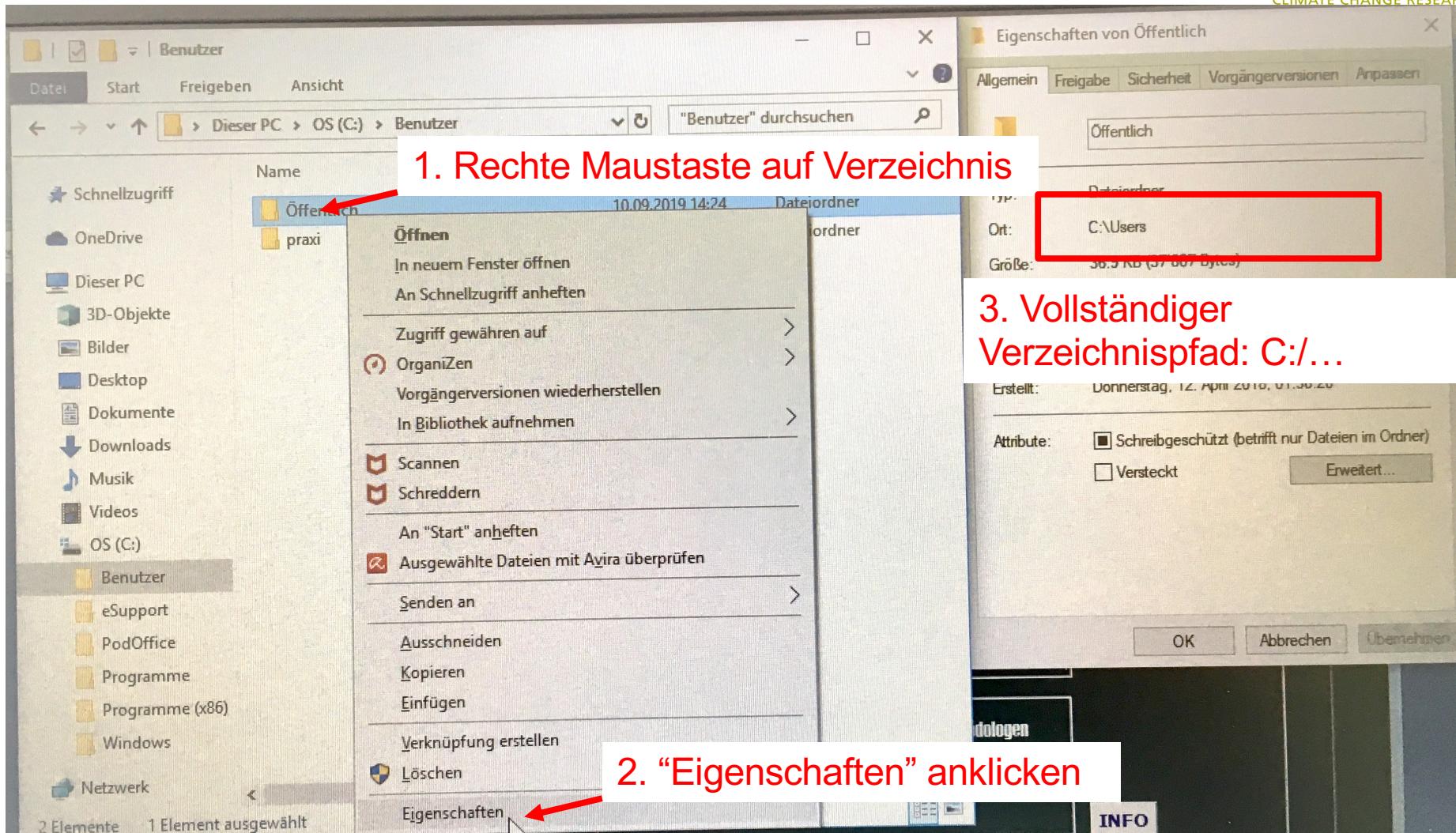
bzw.

Windows: 'C:/Users/...'

**ACHTUNG bei Windows wird \ benutzt und muss für R in / verwandelt werden!**



# Arbeitsverzeichnis im Windows Explorer



# Daten einlesen

## Dateien im Verzeichnis anzeigen

```
> list.files('/Users/name/quant_meth...')
```

# sollte alle Dateien und Unterverzeichnisse im angegeben Verzeichnis anzeigen

## ASCII Daten einlesen:

### Textdatei 'file.txt' bzw. 'file.csv' aus Excel exportiert

```
> saison <- read.table('/.../.../.../meteodaten_saison.csv',  
    # Pfad zur Datei angeben in ''  
    header=...,          # Erste Zeile Spaltennamen? TRUE oder FALSE  
    na.strings='..',      # wie sind Fehlwerte kodiert? z.B. '-99', NA  
    sep='... '           # optional: wie sind Werte separiert? wenn durch  
    # tabs getrennt nicht notwendig,  
    # sonst z.B. ',' für .csv (Comma separated values)
```

## Wie sehen eingelesene Daten aus?

- > *str(saison)* ... zeigt die Struktur des Objekts
- > *head(saison)* ... zeigt die ersten paar Zeilen
- > *tail(saison)* ... zeigt die letzten paar Zeilen
- > *summary(saison)* ... einige statistische Grunddaten
- > *saison[1:10,]* ... zeigt die ersten 10 Zeilen an

## Auswahl von Elementen/Spalten/Zeilen

- > *saison[1,1]*
- > *saison[23:24, 'SPALTENNAME']*
- > ...

## Euer Skript startet mit:

```
> saison <- read.table('/euer_r_verzeichnis/meteodaten_saison.csv',  
sep=",", header=T) # 'C:/...' unter Windows  
  
> head(saison) # zeigt erste Zeilen der geladenen Daten an  
  
> str(saison) # zeigt an, ob Daten korrekt als numerisch gelesen wurden
```

**Skript speichern nicht vergessen!**

**Auf korrekte GROSS- und klein-Schreibung achten!**

# Programmierempfehlungen

Kommentare, falls ihr nicht mit “Notebooks/Markdown” arbeitet

```
> # Zeitreihe der Sommertemperatur in Genf, geglättet mit 31-jährigem laufenden  
Mittelwert  
> plot(running.mean(saison[saison[,2]=='Sommer JJA',3],31),ty='l,col='red)
```

Selbsterklärende Objektnamen

```
JJA_temp_genf_31yr-smooth <- running.mean(saison[saison[,2]=='Sommer  
JJA',3],31)  
> plot(JJA_temp_genf_31yr-smooth ,ty='l,col='red)
```

Struktur mit Zeileneinschüben, Leerzeichen, etc.:

```
> for ( i in 3:5) {  
    print ( i + 1 )  
}
```

*u*<sup>b</sup>

---

*b*  
**UNIVERSITÄT  
BERN**

**OESCHGER CENTRE**  
CLIMATE CHANGE RESEARCH

# ÜBUNGEN 1

# R-Übungen 1

ACHTUNG: wenn + statt > am Zeilenanfang steht, ist eine Funktion nicht beendet, d.h. ',),]' oder } fehlen



b  
UNIVERSITÄT  
BERN

OESCHGER CENTRE  
CLIMATE CHANGE RESEARCH

**1.1) Vektoren:** Überlegt euch die erwarteten Lösungen vor dem Eintippen!

```
x <- c(5,2,1,4)  
xx <- c(1,10,15,18)  
y <- rep(1,5)  
z <- c(TRUE, FALSE, TRUE, TRUE)
```

- a) sum (x)  
range(x)  
length(x)  
sum(x)
- b) c(x,y,13)
- c) x[4] \* y[2]  
xx[2:4] + x[1:3]
- d) xx <= 12  
xx [xx <=12]

- e) plot(x,xx)  
plot(x[z],xx[z])

**1.2) Zahlenfolgen:** Erzeugt mit den *rep* und *seq* Funktionen folgende Zahlenfolgen:

- a) 1 2 3 4 5 6 7 8 9
- b) 'm' 'w' 'm' 'w' 'm' 'w'
- c) 1 2 3 4 1 2 3 4 1 2 3 4
- d) 1 2 2 3 3 3 4 4 4 4

**1.3) Lest die Datei**  
"meteodaten\_saison.csv" in R ein:

```
> saison <- read.table('/pfad/.../  
meteodaten_saison.csv', sep=',',  
header=TRUE)
```

Überprüft, ob der Import korrekt verlief?

Welche Spalten sind chr, int, num, ...?

```
> head(), str(), summary(), tail(), class()...
```

*u*<sup>b</sup>

---

<sup>b</sup>  
**UNIVERSITÄT  
BERN**

**OESCHGER CENTRE**  
CLIMATE CHANGE RESEARCH

## Daten auswählen aus data.frame bzw. matrix

**Auswahl in kleinerem data.frame speichern**

```
> sommer <- saison[saison[,2]=='Sommer(JJA)',]  
# Objekt saison, aber nur Zeilen, wo in 2. Spalte 'Sommer (JJA)' steht
```

**Einzelne Variable/Spalte auswählen und in Vektor speichern**

```
> sommer_temp_genf <- sommer[,3] oder  
> sommer_temp_genf <- sommer[, 'Genf_Mitteltemperatur']  
  
> jahre <- sommer[,1]
```

# Rechnungen über ganze Spalten / Zeilen mit APPLY

> *apply(saison[,3:6], 2, mean)* ...2 = spaltenweise (1 wäre zeilenweise)

<i>Genf_Mitteltemperatur</i>	<i>Genf_Niederschlagssumme</i>	<i>GrStBernhard_Mitteltemperatur</i>	<i>GrStBernhard_Niederschlagssumme</i>
9.732662	234.364206	-1.182476	549.379418

Nur für Auswahl (z.B. Sommersaison; siehe vorherige Folie)

> *apply(saison[saison[,2]=='Sommer(JJA)',3:6], 2, mean)*

<i>Genf_Mitteltemperatur</i>	<i>Genf_Niederschlagssumme</i>	<i>GrStBernhard_Mitteltemperatur</i>	<i>GrStBernhard_Niederschlagssumme</i>
18.168750	254.375893	5.987202	432.576786

oder direkt mit dem neu erzeugen data.frame 'sommer':

> *apply(sommer[,3:6], 2, mean)*

oder einfach das Mittel des vorher erzeugten Vektors 'sommer\_temp\_genf':

> *mean(sommer\_temp\_genf)*

# Rechnungen nach Klassen

Funktion **aggregate()**

Syntax: **aggregate(numerischer Vektor(en), list(Klassen), FUN= Funktion)**

> **s.agg <- aggregate(saison[,3:6], list(saison[,2]), FUN = mean)**

Group.1	Genf_Mitteltemperatur	Genf_Niederschlagssumme	GrStBernhard_Mitteltemperatur
1 Fruehling(MAM)	9.369940	209.3482	-3.325595238
2 Herbst(SON)	9.905706	264.9090	-0.006306306
3 Sommer(JJA)	18.168750	254.3759	5.987202381
4 Winter(DJF)	1.487798	209.0964	-7.374702381

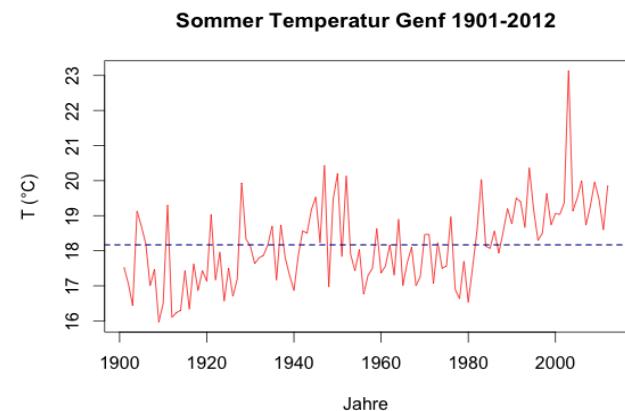
	GrStBernhard_Niederschlagssumme
1	596.0366
2	558.9613
3	432.5768
4	610.0286

## Grafiken in R

„High level‘ Plot Funktionen – „zaubern aus allem eine Grafik‘, jedoch nicht immer nach Wunsch, je nach Daten/Objekt

*plot(), hist(), boxplot(), pairs(), ...*

```
> plot(saison)  
> plot(saison[, 'Genf_Mitteltemperatur'])  
> plot(sommer[,3])
```



„Low level‘ Plot Funktionen – kann man einer bestehenden Grafik nur hinzufügen

*lines(), abline(), points(), ...*  
> abline(h=18)

## Funktion ***plot(x,y)***

```
> ?plot
```

```
> plot(sommer[,1],sommer[,3])
```

```
> plot(sommer[,1],sommer[,4],
```

```
      type ='l',
```

....Darstellung: Punkte „p', Linie „l',

```
      xlab='Jahr',
```

....Beschriftung x-Achse,

```
      ylab='Niederschlag (mm)', ....Beschriftung y-Achse,
```

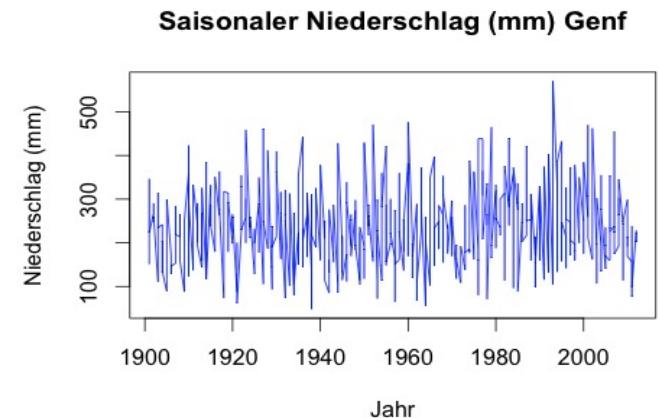
```
      main='Saisonaler Niederschlag (mm) Genf', ...Überschrift
```

```
      col='blue',
```

...Farbe z.B. „green', 'red', ...

```
      ylim=c(80,600), xlim=c(...,...),
```

```
      ...)
```



weitere Grafik-Parameter findet ihr mit: **?par**

## Funktion ***boxplot(y~x)***

> *?boxplot*

> *boxplot(saison[,3] ~ saison[, 'Saison'])*

... *boxplot(daten~gruppeninfo)*

> *boxplot(saison[,3] ~ saison[,2],*

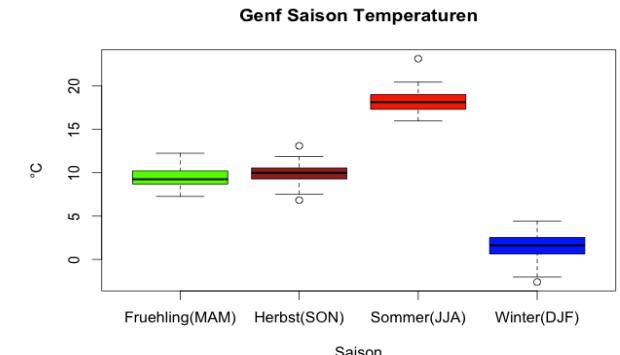
*main='Genf Saison Temperaturen',* ... Überschrift

*ylab='° C',*       , ... Beschriftung y-Achse,

*xlab='Saison',*   , ... Beschriftung x-Achse,

*col=c('green', 'brown', 'red','blue')* ... Farbe(n)

*...)*



## Funktion *hist(x)*

```
> ?hist  
> hist(sommer[,3])  
  
> hist(sommer[,3],  
       main='Histogramm',  
       ylab='Häufigkeiten',  
       xlab='° C',      ,  
       col='green',  
       breaks=20,  
       ...)
```

... Überschrift  
... Beschriftung y-Achse,  
... Beschriftung x-Achse,  
... Farbe(n)  
... ungefähre Anzahl Klassen

## Funktion **barplot(x)**

```
> ?barplot  
> s.agg <- aggregate(saison[,3:6], list(saison[,2]), FUN = mean)  
                                ... von Folie mit aggregate() Funktion  
> barplot(s.agg[,2])  
  
> barplot(s.agg[,3],  
          names.arg=s.agg[,1],  
          main='Barplot',  
          ylab='° C',  
          xlab='Saison',    ,  
          col=c('green', 'brown', 'red','blue'), ... Farbe(n)  
          ...)
```

... Beschriftung der bars

... Überschrift

... Beschriftung y-Achse,

... Beschriftung x-Achse,

... Farbe(n)

# Hinzufügen von „low-level“ Plot Funktionen

zuerst High-Level Funktion z.B. **plot(x,y)**, Grafik öffnet sich

> **plot(sommer[,1], sommer[,3], type='l', ...)**

Hinzufügen von „low-level“ Funktionen bei offener Grafik

> **abline(h=20)** ... horizontale Linie bei 20 (auch **vertikal**)

> **abline(h=mean(sommer[,3]), col='blue')** ... horiz. Linie bei Mittelwert

> **points(sommer[,1], sommer[,3], pch=2)** ... Punkte hinzufügen  
... Punkt-typ

> **lines(x,y,...)**

> **legend(...)** ... Legende hinzufügen

## Grafikfunktion **par(...)** vor plots

```
> ?par          ... Grafikparameter für alle Grafiken festlegen  
> par(mfrow=c(2,1))    ... plot mit 2 Grafiken untereinander (2 Zeilen)  
> barplot(s.agg[,2] names.arg=s.agg[,1])  
> plot(sommer[,1], sommer[,3], type='l', xlab='Jahr', col=„red“)
```

## Legende hinzufügen, nach plots (low-level Funktion), **?legend**

```
> legend('topleft',  
         legend='Sommer',  
         col='red',  
         lty=1,      ... wenn Linie->Linientyp (1=durchgezogen)  
                   wenn Symbol: pch (pointcharakter)=(z.B.) 1  
         cex=0.8, ...)        ... Grösse der Legende (relative zu 1)
```

*u*<sup>b</sup>

---

*b*  
**UNIVERSITÄT  
BERN**

**OESCHGER CENTRE**  
CLIMATE CHANGE RESEARCH

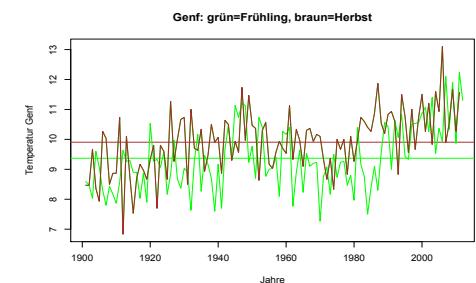
# ÜBUNGEN 2

# R-Übungen 2

ACHTUNG: immer Teil des Codes einzeln ausführen, wenn etwas nicht funktioniert! Hier z.B.:  
`saison[,2]=='Fruehling(MAM)'`

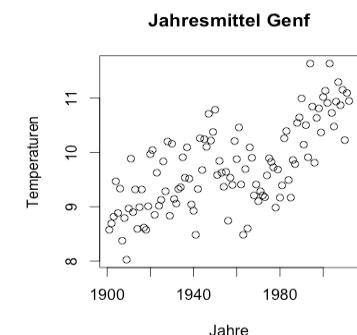
## 2.1) Grafik erstellen

- > Extrahiert aus den saisonalen Daten erstens nur die Frühlingsdaten, zweitens nur die Sommerdaten und drittens nur die Herbstdaten z.B.  
`fruehling <- saison[saison[,2]=='Fruehling(MAM)',]`
- > Erstellt einen Plot, mit den Jahren auf der x-Achse und der Temperatur in Genf auf der y-Achse. Stellt dabei die **Frühlings-, Sommer- und Herbsttemperaturen** als Linien mit unterschiedlichen Farben im gleichen Plot dar:
  - > zuerst `plot(x,y,col=' ', xlab=' ',...)`
  - > dann mit `lines(x,y,col=..)` weitere Saisons
  - > Vergebt eine Überschrift und beschriftet auch beide Achsen
  - > Fügt Linien der beiden Mittelwerte hinzu mit `abline(h=...)`

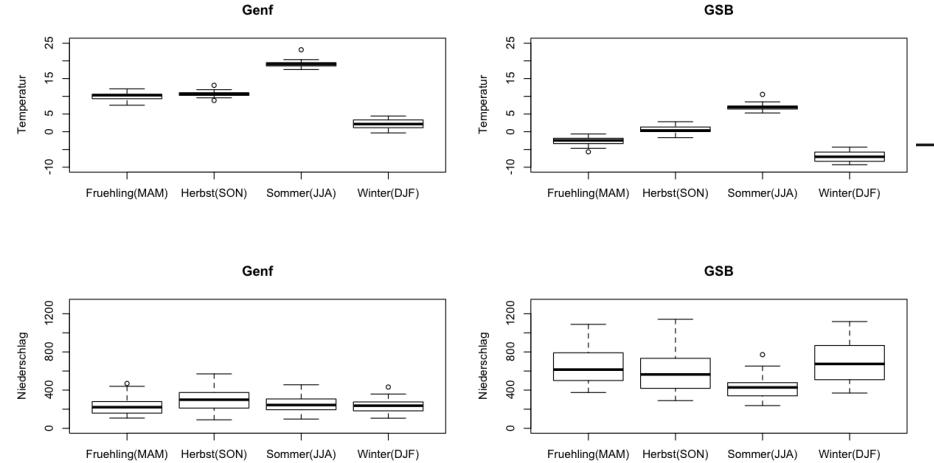


## 2.2)

- > Erstellt mittels `aggregate()` die **Jahresmittelwerte** der Temperatur für Genf und stellt diese in einem Scatterplot mit Punkten `plot(x,y)` dar.
- > Beschriftet die Achsen und vergeb einen Titel.



# R-Übungen 2



## 2.3)

- > Wählt den **Zeitraum 1981-2010**, z.B. `zeit <- saison[,1]>=1981 & ...`
- > Stellt die Temperatur- und Niederschlagsverteilungen der Saisons in Genf und Gr. S. Bernhard für diesen Zeitraum in vier `boxplot()` dar.
- > Das Grafikausgabefenster kann mit `par(mfrow=c(2,2))` in 2 Zeilen und 2 Spalten geteilt werden.
- > Beschriftet wieder die Achsen und vergeb Titel. Achtet darauf, für beide Stationen gleiche y-Achsen zu wählen, so dass die Plots gut visuell vergleichbar sind, z.B. bei Niederschlag je `ylim=c(0,1300)`
- > Das Grafikausgabefenster mit `par(mfrow=c(1,1))` wieder auf 1 Zeilen und 1 Spalten zurücksetzen.

*u*<sup>b</sup>

---

<sup>b</sup>  
**UNIVERSITÄT  
BERN**

**OESCHGER CENTRE**  
CLIMATE CHANGE RESEARCH

## Graphik speichern

als eps, jpg, pdf,...    '**plot.pdf**'

in R-Studio:

'Export' ... 'Save Plot as PDF' oder 'Image' ... (bei 'Image' gewünschtes Format wählen)... 'Directory' angeben (=Pfad) oder manuell

> **pdf('name.pdf', width=9, height=4.5, ...)** ... oder

> **png(...)** ... zum Grafikdatei anlegen/öffnen, je nach Formatwunsch

> **plot(...)** ... *Grafik erstellen*

> **dev.off()** ... *Grafik abschliessen*

## Daten speichern

als ASCII Textdatei 'file.txt'

```
> write.table(saison, ... welches Objekt soll gespeichert werden  
  file='.../.../.../file.txt') ... Pfad und gewünschten Dateinamen angeben
```

**Es gibt spezielle Funktionen zum Lesen und Schreiben vieler weiterer Datenformate. Diese findet ihr bei Bedarf leicht im Internet.**

# Umgang mit Fehlwerten in R

R kodiert Fehlwerte mit 'NA'

**bei Funktionen:**

> *a* <- *c(1, 3, 4, NA, 5)*

> *sum(a)*

[1] NA ...man muss angeben wie mit NAs umgehen will mit **na.rm**

> *sum(a, na.rm=TRUE)* ...z.B. na.rm (NA remove) TRUE

[1] 13 ... NAs werden ignoriert

auch für *mean()*, *sd()*, *apply()*, *aggregate()*,...

**beim Daten einlesen:**

> *x* <- *read.table('data.txt', na.strings=...)* ..angeben wie NAs im  
einzulesenden Datensatzkodiert sind z.B. „-999“, „NA“...

oder nach einlesen mit *replace(x, x==...,NA)*

# Umgang mit Fehlwerten in R

## ABER: Abfrage von Fehlwerten mit „is.na“

```
> a <- c(1, 3, 4, NA, 5)
```

```
> is.na(a)
```

```
[1] FALSE FALSE FALSE TRUE FALSE
```

...nicht a==NA

...logischer Vector

```
> which(is.na(a))
```

```
[1] 4
```

...Vektor mit Elementnummer

umgekehrt

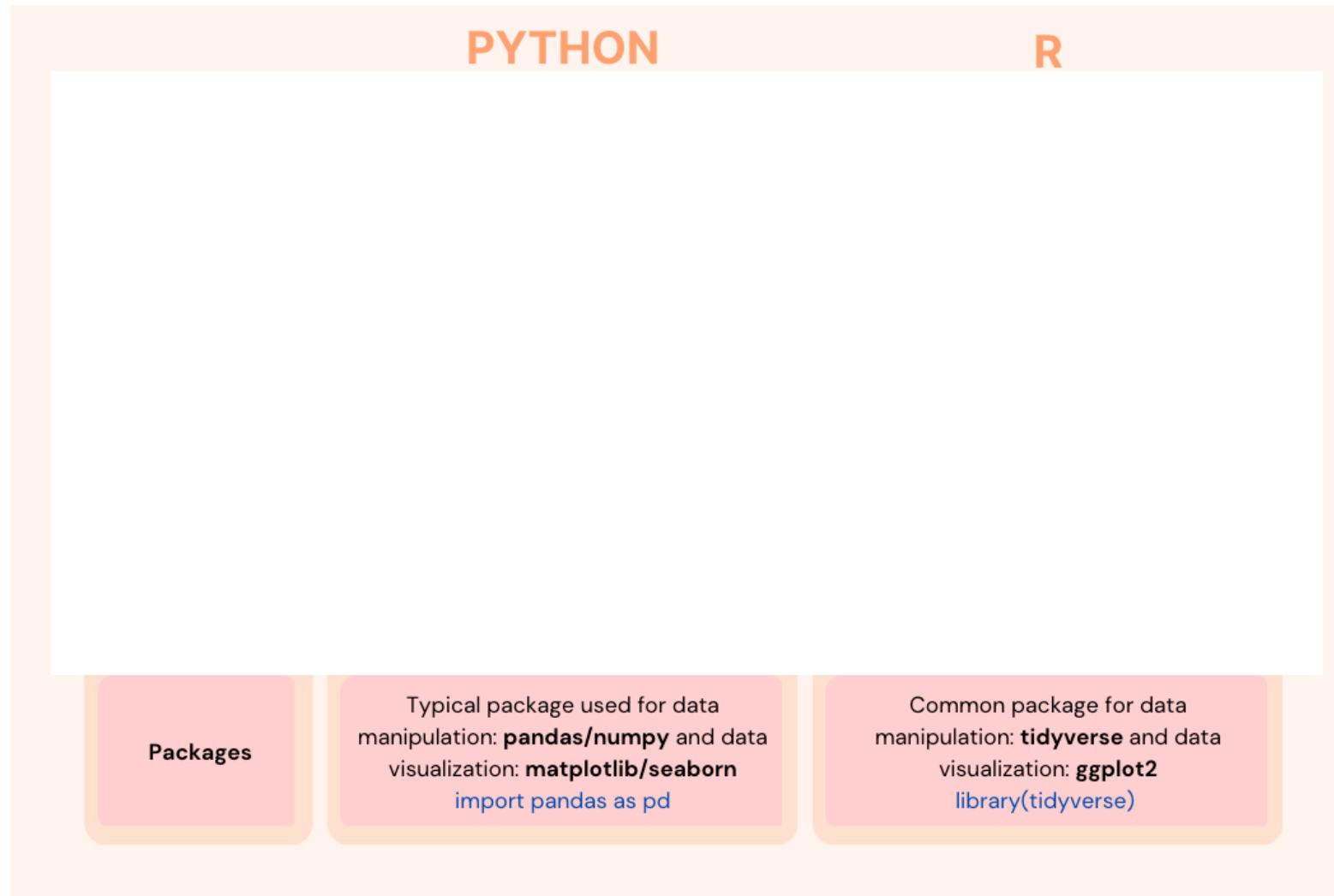
```
> !is.na(a)
```

```
[1] TRUE TRUE TRUE FALSE TRUE
```

...! bedeutet NICHT

```
> which(! is.na(a))
```

# Python vs R



<https://towardsdatascience.com/the-starter-guide-for-transitioning-your-python-projects-to-r-8de4122b04ad>

# Packages/Libraries (Pakete/Bibliotheken) installieren und laden

*Am Beginn ins Script schreiben welche Pakete benötigt und geladen werden sollen (z.B. `library('lattice')`)*

Dazu ist einmalige Installation auf Computer notwendig:

> `install.packages('lattice', dependencies=TRUE)` ... nur 1 x, dann auf  
Computer installiert

Dann am Anfang von jedem Skript laden, welches die Funktionen der Bibliothek nutzen soll:

> `library(lattice)` ... Paket laden (in jeder Session!)

Hilfe zur Bibliothek:

> `library(help=lattice)` ... Hilfe/Information zu package  
> `?lattice` ... Hilfe/Information zu package  
> `demo(lattice)` ... einige packages haben demo files

## Tidyverse Library

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

Objekt type “tibble” einlesen, ähnlich data.frame()

```
> saison=read_csv("meteodaten_saison.csv")
```

Datentransformationen

```
> winter <- filter(saison, Saison == "Winter(DJF)") # auswählen
> arrange(saison, desc(Jahr)) # sortieren
> select(saison, Jahr, Saison, Genf_Mitteltemperatur)
```

Pipes

```
> saison = saison %>% mutate(temp_diff_genf_san_bern = Genf_Mitteltemperatur -
  GrStBernhard_Mitteltemperatur)
```

## ggplot()

```
> ggplot(data = winter) +  
  geom_point(mapping = aes(x = Jahr, y = Genf_Mitteltemperatur))  
  # Fügt Punkte hinzu
```

Weitere “aes” Parameter:

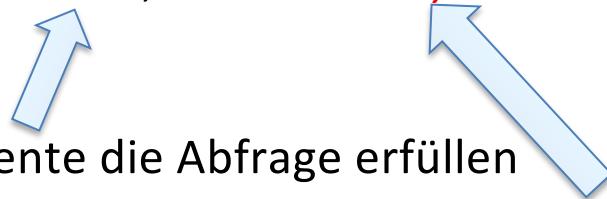
*color = Saison ; size = ... ; alpha = ... ; shape = ...*

Mehrere Abbildungen kombinieren:

```
> ggplot(data = saison) +  
>   geom_point(mapping = aes(x = Jahr, y = Genf_Mitteltemperatur, color=Saison)) +  
>   facet_wrap(~ Saison, nrow = 2, ncol = 2)
```

# ifelse – Bedingte Anweisung für Vektoren

> **ifelse** ( Abfrage , wenn TRUE, wenn FALSE )



Operation für alle Elemente die Abfrage erfüllen

Operation für alle Elemente die Abfrage NICHT erfüllen

z.B.:

> **a <- c( 5, 7, 10)**

> **ifelse ( a==7, a+1, a-1 )**

> [1] 4 8 9

## if – Bedingte Anweisung für einzelne Elemente, keine Vektoren

> *if* ( Bedingung ) { was ausführen wenn Bedingung TRUE }

> *if* ( Bedingung ) {  
  was ausführen wenn Bedingung TRUE  
} *else* {  
  was ausführen wenn Bedingung FALSE  
}

z.B.:

> *b* <- 5  
> *if* ( *b*==10 ) { *b* <- *b*+1 } *else* { *b* <- *b*-1 }  
> *b*  
[1] 4

## Schleifen (loops): z.B. *for()* – Schleife

wenn gleiche Operationen mehrmal vorgenommen werden  
müssen z. B. in Algorithmen

> *for* (beliebiger Name *in* Vektor angeben) { *Operationen* }

für alle „beliebiger Name“ (z.B. „i“) aus dem angegebenen „Vektor“ wird  
nacheinander eine Operation durchgeführt

z.B.:

> *for* ( *i* *in* 3:5 ) { *print* ( *i*+1 ) }

[1] 4

[1] 5

[1] 6

## Schleifen (loops): z.B. *for()* – Schleife mit Laufindex

z.B.:

```
> x <- c(8, 10, 12)
```

```
> xx <- vector() # leeren Vektor für Ergebnisse definieren
```

```
> j=1 # laufender Index (ausserhalb Schleife!)
```

```
> for ( i in x) {  
  xx [j] <- ( i+1 )  
  j=j+1  
}
```

```
> xx  
[1] 9 11 13  
> j  
[1] 4
```

# eigene Funktionen I

> *Funktionsname <- function (Argument 1, Argument 2, ...)* Ausdruck

> *Funktionsname <- function (Argument 1, Argument 2, ...){*

*mehrere Ausdrücke*

*...*

*}*

# eigene Funktionen II

*z.B.: Funktion die 2 Werte multipliziert*

```
> myFunc <- function (x, y) {  
  w <- x * y  
  return(w)      # return() gibt Ergebnis aus  
}
```

```
> myFunc(1,2)      # Funktion aufrufen mit Argumenten  
[1] 2
```

## eigene Funktionen III

z.B.: Funktion die nur alle positiven Werte eines Vektors ausgibt

```
> myFunc2 <- function (x) {  
  y <- x[x>0]  
  return(y)      # return() gibt Ergebnis aus  
}
```

```
> myFunc2(c(1,2,-1,4))    # Funktion aufrufen mit Argumenten  
[1] 1 2 4
```

!! Funktionen kann man in eigenem R-script speichern ('auslagern') und mit **source()** in Arbeitsskript laden

# Python vs R

	PYTHON	R
Assigning a Variable	Use assignment operator "=" <code>x = 1</code>	Use assignment operator "<- " or "=" <code>x &lt;- 1 or x = 1</code>
Data Structure Difference	Python list datatype is equivalent to vector datatype in R <code>ex_list = [1, 2, 3, 4, 5]</code>	Common data type in R is a vector <code>ex_vector &lt;- c(1, 2, 3, 4, 5)</code>
Indexing and slicing	Indexing starts from 0, inclusive of the start index and excludes the end index. <code>ex_list[0] and ex_list[0:2]</code> <code>output: 1      output: [1, 2]</code>	Indexing starts from 1, Indexing is inclusive of both start and end index <code>ex_vector[1] and ex_vector[1:2]</code> <code>output: 1      output: 1, 2</code>
Chaining operations	Method chains with the "." operator <code>df.head(n) or df.describe()</code>	Can chain operations using following symbol: "%>%" <code>df %&gt;% head(n) or df %&gt;% summary()</code>
Packages	Typical package used for data manipulation: <b>pandas/numpy</b> and data visualization: <b>matplotlib/seaborn</b> <code>import pandas as pd</code>	Common package for data manipulation: <b>tidyverse</b> and data visualization: <b>ggplot2</b> <code>library(tidyverse)</code>

# Chatbots

Chatbot Einsatz an Uni allgemein: Was habt ihr schon lernt und selber gemacht?

# Chatbots

- > Besser Fragen stellen lassen nach den ein virtueller Gesprächspartner definiert wurde als Antworten geben lassen
- > Brainstorming
- > Text überarbeiten (vor allen in Nicht-Muttersprache)
- > aber auch Sprachniveau, Stil, etc.
- > immer kontrollieren, ihr seid verantwortlich!

# Chatbots - Prompting

- > • **Kontext:** Konkretisieren Sie, in welchem Zusammenhang Ihre Anfrage steht. Durch zusätzliche Informationen können Genauigkeit und Qualität der vom Chatbot generierten Ausgaben verbessert werden.
- > • **Zielgruppe:** Identifizieren Sie die Zielgruppe und berücksichtigen Sie deren Kenntnisse und Erfahrungen. Geben Sie diese Informationen in klarer und verständlicher Sprache an.
- > • **Zielsetzung:** Beschreiben Sie präzise, welches Ziel mit dem Prompt erreicht werden soll und überlegen Sie sich, welche Art von Informationen der Chatbot hierfür benötigt.
- > • **Datenquelle:** Sollten Sie zusätzliche Daten eingeben, stellen Sie sicher, dass die Daten von hoher Qualität und Präzision sind.

# Chatbots - Programmieren

## 1. Interaktion mit einem Chatbot

- > Fragen zum Programmieren, Funktionen oder Syntax stellen
- > Beispiel: "Wie erstelle ich eine Weltkarte mit Kontinentgrenzen in R?" oder "Was entspricht einem Python dictionary in R?"

## 2. Chatbots um Fehlermeldungen zu verstehen

- > Fehlermeldungen oder Code der Problem verursacht einfügen und nach Hilfe fragen.
- > Dies für meist zu leichter verständlichen Lösungen.

## 3. Lernen

- > Chatbot nach Empfehlungen für gutes Codieren fragen oder nach effizienterer Variante.

*u*<sup>b</sup>

---

*b*  
**UNIVERSITÄT  
BERN**

**OESCHGER CENTRE**  
CLIMATE CHANGE RESEARCH

# ÜBUNGEN 3

## Übungen 3.1

- > Bereche die Sommer (JJA) Temperaturanomalien zur Referenzperiode 1961 bis 1990 in Bern mit Excel
- > Schreibe R Code, um die gleiche Berechnung durchzuführen.
- > Neuerdings können Chatbots direkt Datenanalysen ohne Programmierkenntnisse durchführen:  
<https://help.openai.com/en/articles/8437071-data-analysis-with-chatgpt>  
Überprüfe den generierten Code.
- > Diskutiere die Vorteile, Nachteile und Risiken der drei Methoden.

# Übungen 3

mit Standard R (Beispiele unten) oder  
tidyverse und ggplot mit Chatbot Hilfe



b  
UNIVERSITÄT  
BERN

OESCHGER CENTRE  
CLIMATE CHANGE RESEARCH

## 3.2) Klimadiagramm

- > Ladet den Datensatz meteodataen\_tag.csv nach dem Excel Export in R  
**(ACHTUNG:** NA-Werte sind sowohl mit '-' als auch mit 'NA') kodiert, deshalb:  
`na.strings = c('-', 'NA')`
- > Mit `str()` ansehen, ob Daten korrekt (z.B. als **numerisch**) gelesen wurden.
- > Erstellt ein Histogramm (`hist()`) mit den Tagestemperaturen mit feinen Abständen  
(`breaks=40`).
- > Wie sieht die Verteilung nach Augenmaß aus?
- > Berechnet die Monatsmittelwerte der Temperatur und der Bewölkung über alle Jahre (also Mittel über alle Jan, alle Feb,... wie in Klimadiagrammen). Achtung: Fehlwerte vorhanden!
- > Erstellt in eine Abbildung mit zwei Barplots der Ergebnisse übereinander  
(`par(mfrow=c(2,1))`). Was erwartet ihr?

## Übungen 3

### 3.3) Boxplots

- > Wählt den Zeitraum **2000-2001** in den täglichen Daten.
- > z.B. `zeit <- meteodaten_tag[,1] >= 2000 & meteodaten_tag[,1] <= 2001`
- > Stellt die Temperaturen dieses Zeitraumes als Funktion der Bewölkung in einem `boxplot()` dar (je einen Boxplot pro Bewölkungsklasse).
- > Beschriftet die Achsen und vergeb einen Titel.
- > Unter welchen Bewölkungs- bedingungen ist die Spannweite und Varianz der Temperatur am grössten?
- > Findet heraus welcher Monat im Mittel der bewölkungsärmste und der -reichste Monat ist (im Mittel der 2 Jahre). Wieviel Bewölkung gibt es im Mittel in diesen Monaten (in Octas)?

## Übungen 3

### 3.4) R als GIS Ersatz

- > Installiert das Paket 'maps' und ladet es in R (z.B. library(maps)). Findet die x,y-Koordinaten von Genf und dem Gr. S. Bernhard heraus (ungefähr).
- > Versucht eine Europakarte herzustellen und Genf und G.S.Bernhard als Punkte auf der Karte zu plotten und die Punkte mit Stationsnamen zu versehen:

```
# zuerst einen leeren plot erstellen mit
# Koordinaten von Europa.
> plot(x=c(-5,30), y=c(35,60),
       type='n', xlab='lon', ylab='lat')
# type=' n" heisst es wird nichts geplottet
> map("world",add=TRUE)
# jetzt die Stationen als Punkte dazu (evtl.
# in verschiedenen Farben, mit
# unterschiedlichen Symbolen und mit
# Text!? Verwendet Google!!)
> points(...)
```

*u*<sup>b</sup>

---

<sup>b</sup>  
**UNIVERSITÄT  
BERN**

**OESCHGER CENTRE**  
CLIMATE CHANGE RESEARCH

## Take-home messages

- > Insbesondere für grosse Datensätze viel besser als Tabellenkalkulation
- > Erzeugt mit einer Zeile Code grundlegende statistische Auswertungen und Abbildungen wie Histogramme und Test, die mit Excel kompliziert oder gar nicht möglich sind
- > Bei Fragen Internetsuchmaschine oder Chatbot gebrauchen
- > Programmieren muss gar nicht so kompliziert sein
- > ACHTUNG: man bekommt (fast) immer etwas heraus, stellt sicher, dass es auch das Richtige ist!
- > In der Prüfung müsst ihr nicht alle Funktionen und deren Syntax in R kennen, aber einfachen Code verstehen können und beschreiben können wie ein Problem lösen würdet, z.B. Scheife über alle Stationen, um statistischen Test an allen Orten durchzuführen.