

Statistik für GeisteswissenschaftlR

Vorbereitung

Vortrag der Forschung 20.7.2022
Andreas Blombach, Philipp Heinrich



Friedrich-Alexander-Universität
Philosophische Fakultät und
Fachbereich Theologie

Worum geht's?

- Wie öffne und speichere ich Datensätze, um sie statistisch auswerten zu können?
- Wie lassen sich Daten mit R zusammenfassen und graphisch darstellen?
- Wie funktionieren inferenzstatistische Analysen in R?
- Wie bekomme ich R dazu, das zu tun, was ich möchte?
- Wie kann ich meine Kenntnisse gezielt erweitern?



Zum Lesen und Üben



Friedrich-Alexander-Universität
Philosophische Fakultät und
Fachbereich Theologie

Literatur (1)

- Wahrscheinlich das beste (und garantiert lustigste!) Lehrbuch zum Thema:
Field, Andy / Miles, Jeremy / Field, Zoë (2012): Discovering Statistics Using R. London [etc.]: Sage Publications.
Überarbeitete Neuauflage in Planung; Material dazu:
<https://www.discovr.rocks>
- Neuer und in Form einer Geschichte:
Field, Andy (2016): An Adventure in Statistics. The Reality Enigma. London [etc.]: Sage Publications.
Mit interaktiven Übungen: <http://milton-the-cat.rocks/home/adventr.html>
- Skript zur Einführung in Statistik und R mit Beispielen aus der Sprachforschung: <http://janhove.github.io/statintro.html>

Literatur (2)

- Verständliche, moderne und umfangreiche Einführung:
Ismay, Chester / Kim, Albert Y. (2020): Statistical Inference via Data Science. A ModernDive into R and the Tidyverse. Boca Raton [etc.]: Chapman & Hall/CRC
Auch online verfügbar: <https://moderndive.com/index.html>
- Verständliche, moderne Einführung für Linguisten mit klarem Schwerpunkt auf statistischer Modellierung:
Winter, Bodo (2019): Statistics for Linguists. An Introduction Using R. New York [etc.]: Routledge.
- Besonders nützlich für Datenvisualisierung:
Wickham, Hadley / Grolemund, Garrett (2017): R for Data Science. Sebastopol, CA: O'Reilly.
Auch online verfügbar: <http://r4ds.had.co.nz>

Literatur (3)

- Stolpersteine in der Forschungspraxis:
Reinhart, Alex (2015): Statistics Done Wrong. The Woefully Complete Guide. San Francisco: No Starch Press.
- Statistische Modellierung für Fortgeschrittene (Methoden, Modellauswahl, Nichtlinearität, Entscheidungsbäume und Random Forests, SVMs, Clustering usw.):
James, Gareth / Witten, Daniela / Hastie, Trevor / Tibshirani, Robert (2021): An Introduction to Statistical Learning: with Applications in R. 2. Aufl. New York: Springer.
Auch online verfügbar: <https://www.statlearning.com>
- Programmieren mit R:
Wickham, Hadley (2019): Advanced R. 2. Aufl. Boca Raton: Chapman & Hall/CRC.
Auch online verfügbar: <https://adv-r.hadley.nz>

Anlaufstellen im Netz

- Bei statistischen Fragen und Problemen:
<https://stats.stackexchange.com>
<https://www.reddit.com/r/AskStatistics>
- Bei Problemen mit R:
<https://stackoverflow.com/questions/tagged/r>
<https://www.reddit.com/r/rstats>
- Interaktive R-Kurse bei DataCamp:
<https://www.datacamp.com/courses/tech:r> (siehe auch
<https://www.datacamp.com/community/open-courses>)
- Interaktive Web-Anwendungen mit Shiny:
<https://shiny.rstudio.com>



R und RStudio

alternativ hierzu: Video-Einführung von Andy Field: http://milton-the-cat.rocks/learnr/r/r_getting_started



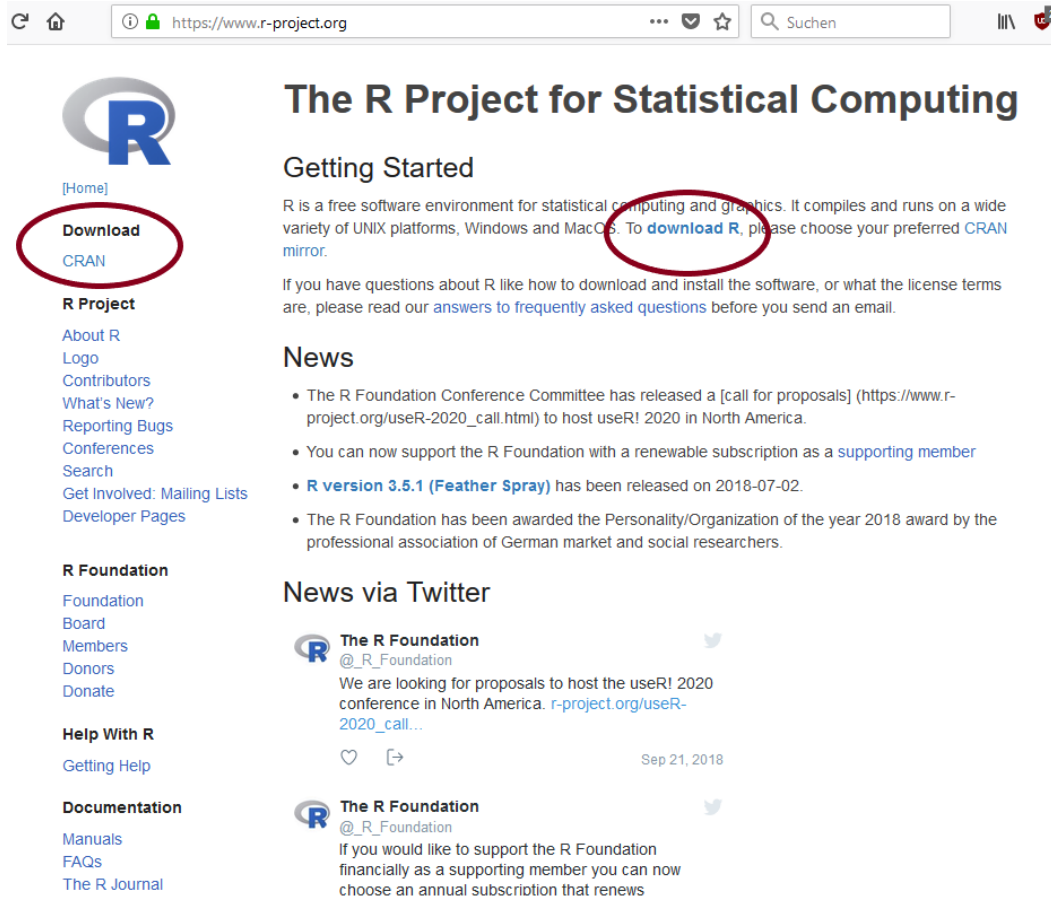
Friedrich-Alexander-Universität
Philosophische Fakultät und
Fachbereich Theologie

Warum ausgerechnet R?

- Es gibt zahlreiche Statistikprogramme, darunter die bekannten kommerziellen Programme SPSS, SAS und STATA – aber nur R ist ähnlich verbreitet *und* kostenlos.
- R hat einen riesigen Funktionsumfang – es gibt für fast alles ein Modul (was man allerdings ggf. erst suchen muss).
- R ist plattformunabhängig.
- Hilfe zu R ist im Netz leicht zu finden (z.B.: *Stack Overflow*, *Cross Validated*).
- R wird von vielen laufend weiterentwickelt und erscheint sehr zukunftsicher.
- R kann hervorragende Grafiken erzeugen.
- **Nachteil:** R ist nicht ganz einfach zu erlernen – man kann nicht einfach nur herunklicken. Das heißt aber auch, dass man oft besser versteht, was man eigentlich tut.

Installation von R (1)

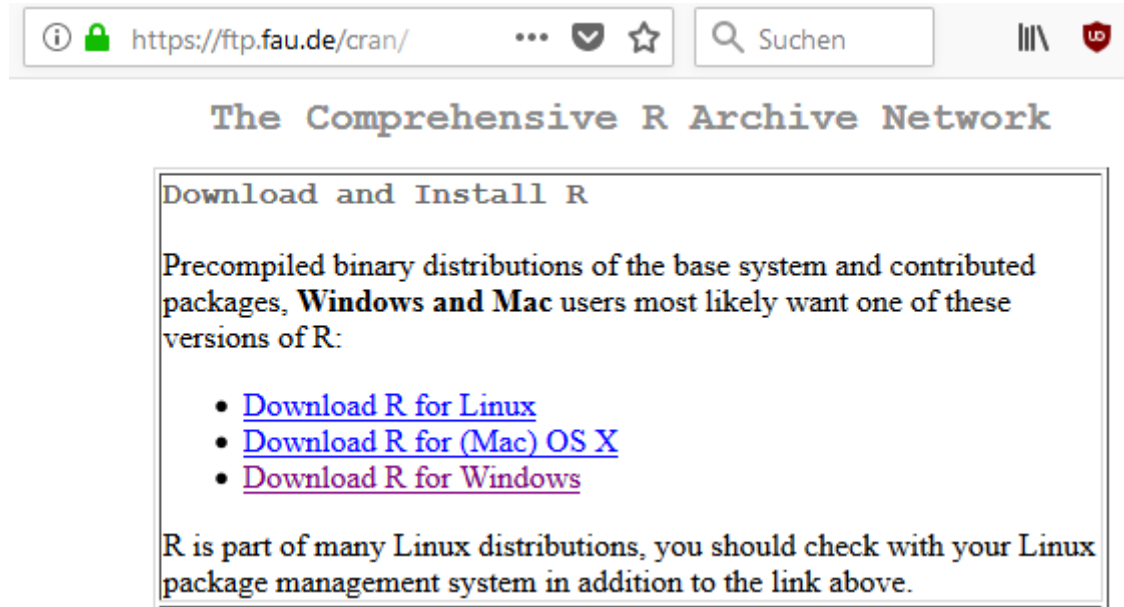
- <https://www.r-project.org> aufrufen und auf einen der im Bild rot markierten Links klicken:



The screenshot shows the homepage of The R Project for Statistical Computing. The browser address bar displays <https://www.r-project.org>. The page features a navigation sidebar on the left with links such as [Home], Download (circled in red), CRAN, R Project, About R, Logo, Contributors, What's New?, Reporting Bugs, Conferences, Search, Get Involved: Mailing Lists, Developer Pages, R Foundation, Foundation, Board, Members, Donors, Donate, Help With R, Getting Help, Documentation, Manuals, FAQs, and The R Journal. The main content area is titled 'The R Project for Statistical Computing' and includes a 'Getting Started' section. In this section, the text states: 'R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred [CRAN mirror](#).' The 'download R' link is circled in red. Below this, there is a 'News' section with several bullet points, and a 'News via Twitter' section displaying tweets from @_R_Foundation. The date 'Sep 21, 2018' is visible at the bottom of the Twitter feed.

Installation von R (2)

- Einen passenden Mirror zum Download wählen, z.B. den der FAU Erlangen-Nürnberg (unter „Germany“ gucken“).
- Dies führt auf folgende Seite: <https://ftp.fau.de/cran>
- Link für das passende Betriebssystem anklicken:



Installation von R (3)

- Einen der im Bild markierten Links anklicken (führt zu <https://ftp.fau.de/cran/bin/windows/base>), anschließend den obersten Link anklicken, um die Installationsdatei herunterzuladen.
- Datei ausführen – R wird installiert.



R for Windows

Subdirectories:

[base](#)

[contrib](#)

[old contrib](#)

[Rtools](#)

Binaries for base distribution. This is what you want to **install R for the first time**.

Binaries of contributed CRAN packages (for R \geq 2.13.x; managed by Uwe Ligges). There is also information on [third party software](#) available for CRAN Windows services and corresponding environment and make variables.

Binaries of contributed CRAN packages for outdated versions of R (for R $<$ 2.13.x; managed by Uwe Ligges).

Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the [R FAQ](#) and [R for Windows FAQ](#).

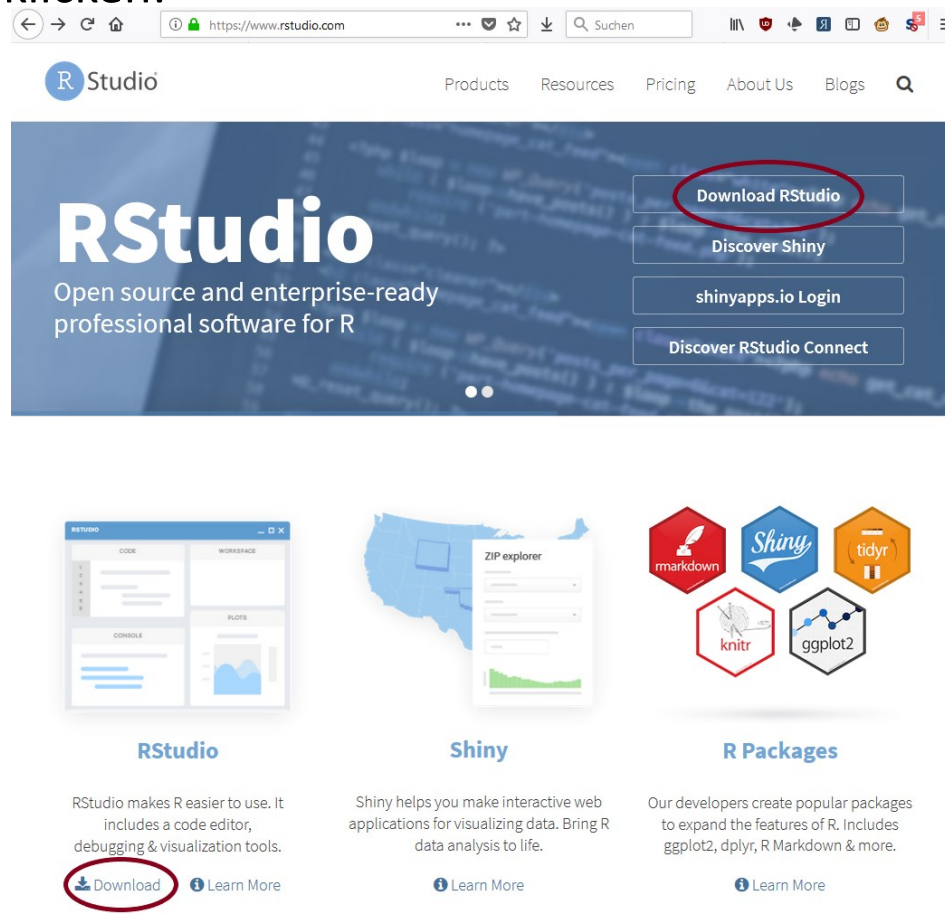
Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

RStudio

- Man kann direkt mit R arbeiten, das ist allerdings unkomfortabel.
- Es gibt einige graphische Benutzeroberflächen (GUIs) für R, die die Möglichkeit bieten, wichtige Befehle auch per Mausklick auszuführen (u.U. lernt man so auch neue Befehle kennen):
 - Interessant v.a. für Anfänger ist vielleicht **jamovi**: <https://www.jamovi.org> (im „syntax mode“ lässt sich der R-Code einsehen und kopieren, der für die Analysen verwendet wird)
 - **R Commander** (Paket: Rcmdr) und **Deducer** (Paket: Deducer) lassen sich von R aus aufrufen und bieten Möglichkeiten, wie man sie vielleicht von SPSS o.ä. kennt.
- **RStudio** ist v.a. eine Entwicklungsumgebung (IDE), die die Arbeit mit R und die Entwicklung von Programmen erleichtern soll (u.a. durch Autovervollständigung, Syntaxhervorhebung, automatische Einrückungen, Paketverwaltung usw.).

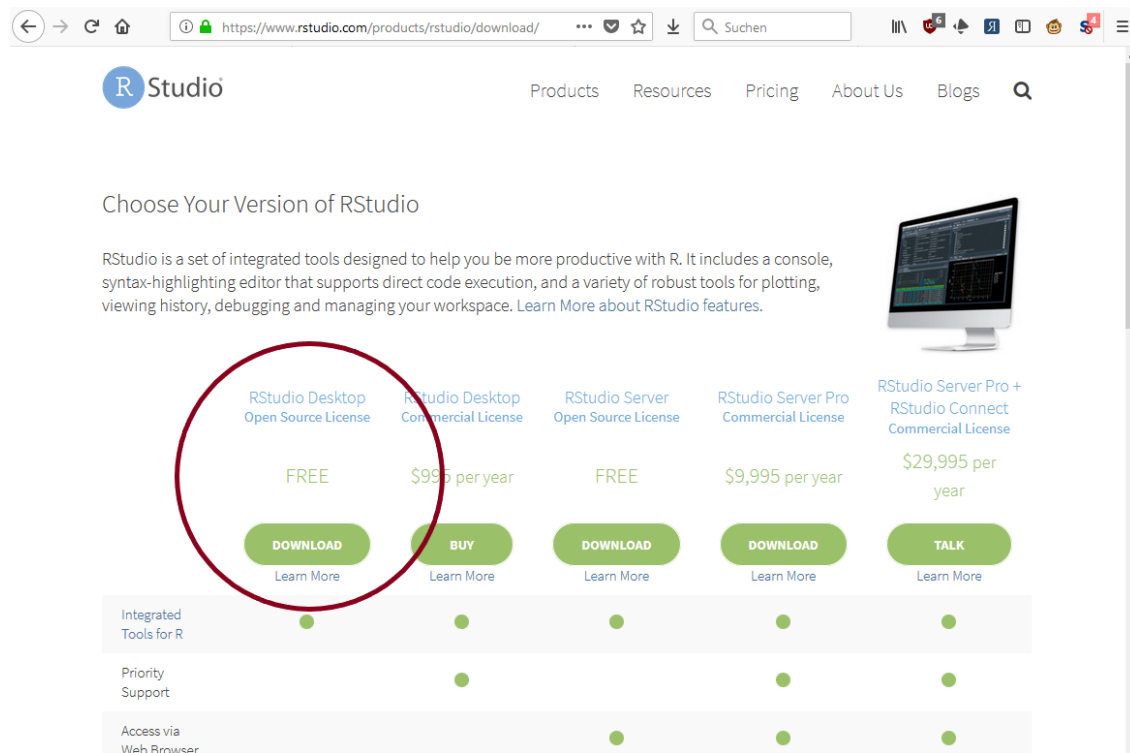
Installation von RStudio (1)

- <https://www.rstudio.com> aufrufen und einen der im Bild rot markierten Links anklicken:



Installation von RStudio (2)

- Dies führt zu <https://www.rstudio.com/products/rstudio/download/>. Dort den markierten Download-Button anklicken und unter „Installers for Supported Platforms“ die Datei herunterladen und installieren, die zum eigenen System passt.





Erste Schritte mit R und RStudio



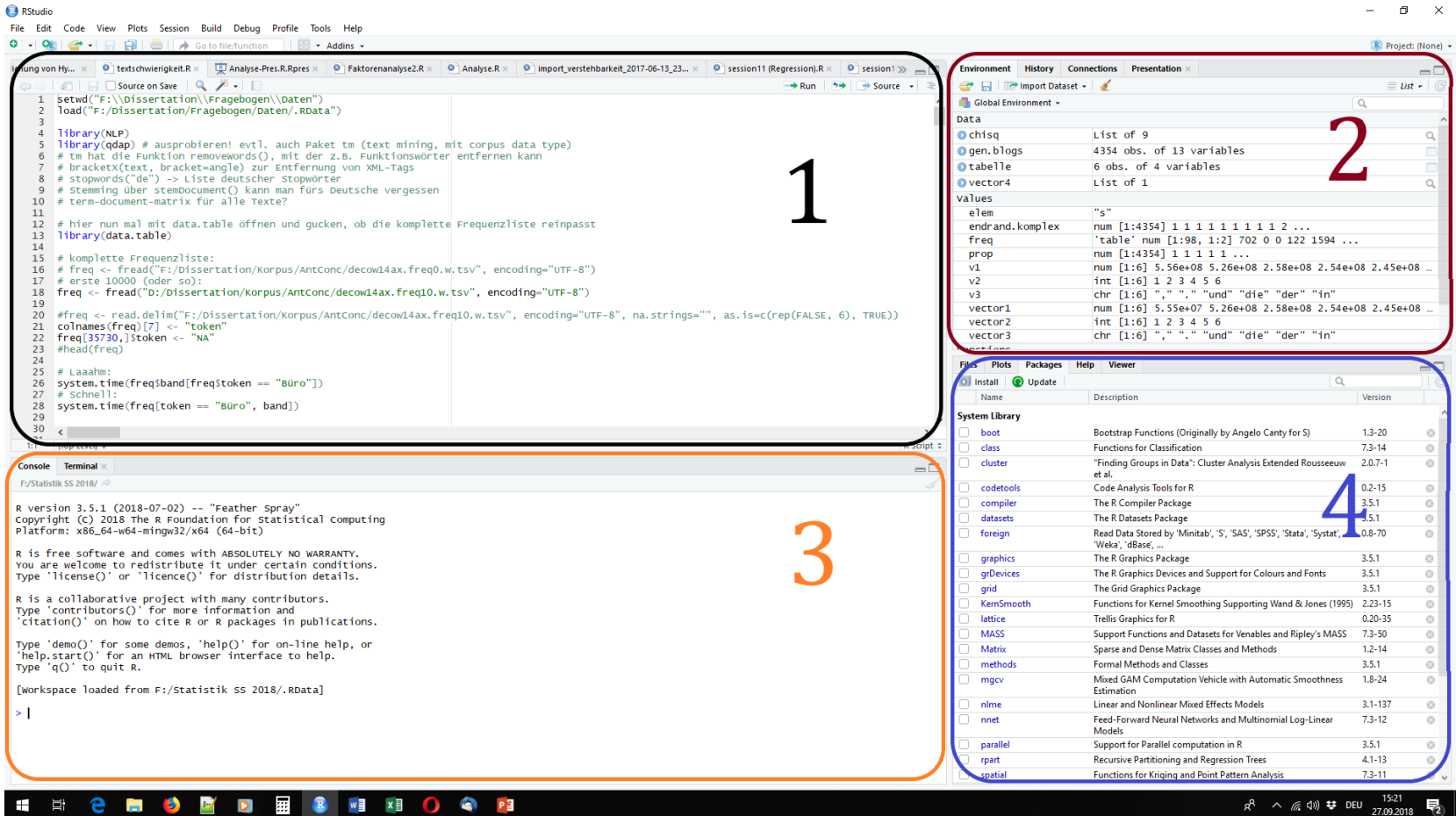
Friedrich-Alexander-Universität
Philosophische Fakultät und
Fachbereich Theologie

RStudio: Übersicht (1)

Bevor es richtig losgeht, sehen wir uns erst einmal an, wie RStudio aufgebaut ist (siehe Bild auf der nächsten Seite):

- (1) Der Bereich links oben wird nur angezeigt, wenn man eine Datei geöffnet oder neu angelegt hat (über *File* → *New File*). Wie in einem Browser können dort mehrere Skriptdateien, Präsentationen o.a. nebeneinander geöffnet werden. Arbeitet man an vielen unterschiedlichen Projekten, empfiehlt es sich, dafür verschiedene Projektordner zu verwenden (*File* → *New Project...*). Jedes Projekt hat dann seine eigenen Tabs und seine eigene Arbeitsumgebung (*Environment*).
- (2) Diese Arbeitsumgebung sieht man rechts oben. Dort sind alle Objekte – Datensätze, Variablenwerte und Funktionen – zu sehen, die R gerade im Speicher hat und auf die man somit über ihre Namen zugreifen kann. Außerdem kann man hier u.a. unter *History* noch einmal alle Befehle sehen, die in der laufenden *Session* (dazu später mehr) eingegeben wurden.

RStudio: Übersicht (2)



1 Source Editor (Script Editor): Displays the R script being executed. The code includes loading data from a file, installing and loading the 'tm' and 'NLP' packages, and performing text mining operations.

```

1 setwd("F:/Dissertation/Fragebogen/Daten")
2 load("F:/Dissertation/Fragebogen/Daten/.RData")
3
4 library(NLP)
5 library(tm) # ausprobieren! evtl. auch Paket tm (text mining, mit corpus data type)
6 # tm hat die Funktion removeWords(), mit der z.B. Funktionswörter entfernen kann
7 # bracketx(text, bracket=angle) zur Entfernung von XML-Tags
8 # stopwords("de") -> Liste deutscher Stopwörter
9 # Stemming über stemDocument() kann man fürs Deutsche vergessen
10 # term-document-matrix für alle Texte?
11
12 # hier nun mal mit data.table öffnen und gucken, ob die komplette Frequenzliste reinpasst
13 library(data.table)
14
15 # komplette Frequenzliste:
16 freq <- fread("F:/Dissertation/korpus/AntConc/decow14ax.freq0.w.tsv", encoding="UTF-8")
17 # erste 10000 (oder so):
18 freq <- fread("D:/Dissertation/korpus/AntConc/decow14ax.freq10.w.tsv", encoding="UTF-8")
19
20 #freq <- read.delim("F:/Dissertation/korpus/AntConc/decow14ax.freq10.w.tsv", encoding="UTF-8", na.strings="", as.is=c(rep(FALSE, 6), TRUE))
21 colnames(freq)[7] <- "token"
22 freq[35730,]$token <- "NA"
23 #head(freq)
24
25 # Laaahm;
26 system.time(freq$band[freq$token == "Büro"])
27 # schnell!
28 system.time(freq[token == "Büro", band])
29
30
  
```

2 Environment Pane: Shows the objects loaded in the global environment. It lists objects like 'chisq', 'gen.blogs', 'tabelle', and 'vector4'.

3 Console: Displays the output of the R session, including the R version (3.5.1), copyright information, and the workspace loaded from the .RData file.

4 Packages Pane: Shows the list of installed and available packages. It includes a table with columns for Name, Description, and Version.

Name	Description	Version
boot	Bootstrap Functions (Originally by Angelo Canty for S)	1.3-20
class	Functions for Classification	7.3-14
cluster	"Finding Groups in Data": Cluster Analysis Extended Rousseeuw et al.	2.0.7-1
codetools	Code Analysis Tools for R	0.2-15
compiler	The R Compiler Package	3.5.1
datasets	The R Datasets Package	3.5.1
foreign	Read Data Stored by 'Minitab', 'S', 'SAS', 'SPSS', 'Stata', 'Systat', 'Weka', 'dBase', ...	0.8-70
graphics	The R Graphics Package	3.5.1
grDevices	The R Graphics Devices and Support for Colours and Fonts	3.5.1
grid	The Grid Graphics Package	3.5.1
KernSmooth	Functions for Kernel Smoothing Supporting Wand & Jones (1995)	2.23-15
lattice	Trellis Graphics for R	0.20-35
MASS	Support Functions and Datasets for Venables and Ripley's MASS	7.3-50
Matrix	Sparse and Dense Matrix Classes and Methods	1.2-14
methods	Formal Methods and Classes	3.5.1
mgcv	Mixed GAM Computation Vehicle with Automatic Smoothness Estimation	1.8-24
nlme	Linear and Nonlinear Mixed Effects Models	3.1-137
nnet	Feed-Forward Neural Networks and Multinomial Log-Linear Models	7.3-12
parallel	Support for Parallel computation in R	3.5.1
rpart	Recursive Partitioning and Regression Trees	4.1-13
spatial	Functions for Kriging and Point Pattern Analysis	7.3-11

RStudio: Übersicht (3)

- (3) Links unten ist die Kommandozeile/Konsole (*console*) zu sehen (oder im ganzen linken Bereich, wenn noch keine Datei geöffnet wurde; diese Konsole sieht man übrigens auch, wenn man R ohne RStudio startet). Dort kann man Befehle eingeben, die dann direkt von R ausgeführt werden. Ergebnisse (sofern es welche gibt), Warn- und Fehlermeldungen werden ebenfalls hier ausgegeben. Befehle in Skriptdateien (also im Bereich links oben) können durch Klick auf *Run* oder durch Eingabe von Strg+Enter ausgeführt werden. Ausgeführt wird dabei stets die Zeile, in der sich gerade die Schreibmarke befindet, oder ein markierter Bereich.
- (4) Unten rechts werden Diagramme (*Plots*) angezeigt, außerdem die Hilfe zu Befehlen und installierte Pakete (*Packages*, dazu später mehr).

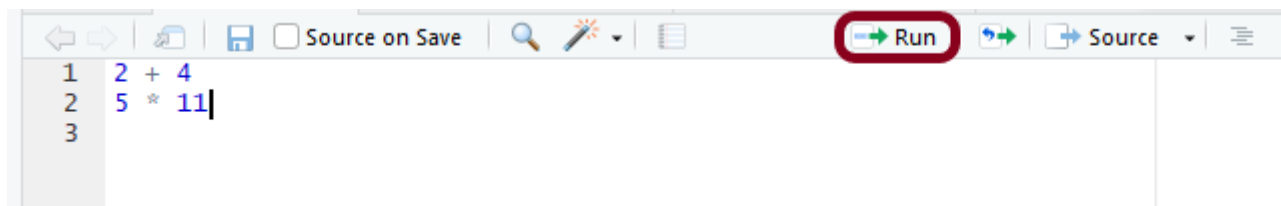
Sessions in R

- Sobald RStudio (oder R) gestartet wird, beginnt eine neue Sitzung oder *Session*.
- Dabei werden üblicherweise allerlei Daten als Objekte (ab S. 36) gespeichert (sichtbar rechts oben unter *Environment*).
- Beim Beenden von RStudio (und damit der Session) muss entschieden werden, ob diese Variablen gelöscht oder lokal gespeichert werden sollen (als *workspace image*) – in letzterem Falle stehen sie beim nächsten Start wieder zur Verfügung. Gespeichert werden sie stets im aktuellen Arbeitsverzeichnis (*working directory*), das sich ändern lässt (siehe S. 41) – so kann man also auch Daten aus mehreren Projekten separat speichern und wieder laden, sodass man nicht alle auf einem Haufen hat und den Durchblick verliert.
- Eine Session kann auch mit dem Befehl *q()* oder *quit()* manuell beendet werden.

Erste Schritte (1)

Genug des Vorgeplänckels – Zeit, selbst aktiv zu werden:

- Lege in RStudio ein neues Skript an: *File* → *New File* → *R Script*
- Im Bereich links oben solltest du jetzt eine leere Datei sehen.
- Hier kannst du nun Befehle eintragen (einen pro Zeile, es sei denn, du trennst sie mit Semikolon).
- Gib eine kleine Rechenaufgabe wie $2 + 4$, $5 * 11$ ein. Führe dann die aktuelle Zeile in der Konsole aus, indem du **Strg+Enter** drückst oder den Button *Run* anklickst:

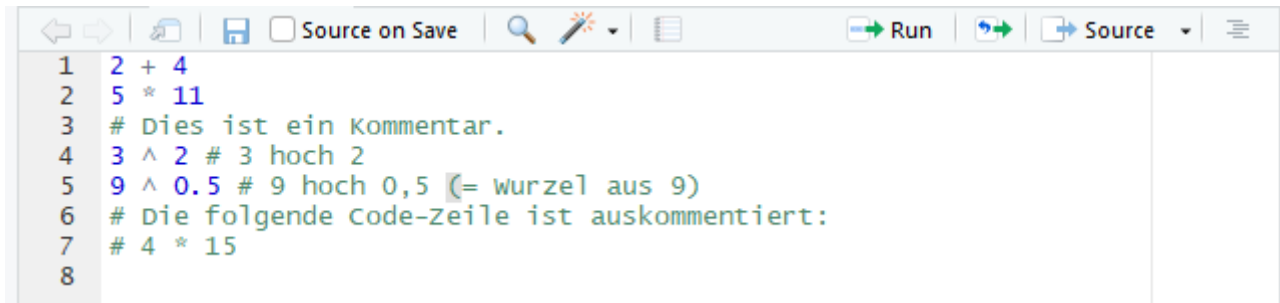


- Unten in der Konsole solltest du nun das Ergebnis sehen:

```
> 5 * 11  
[1] 55  
>
```

Erste Schritte (2)

- Befehle wie diese kann man auch direkt unten links in der Konsole eingeben (und mit Enter ausführen). Meistens ist es aber sinnvoller, im eigenen Skript zu arbeiten, fehlerhafte Befehle dort auszubessern, überflüssige zu entfernen und somit am Ende ein sauberes Protokoll der eigenen Aktivitäten zu haben, das sich Zeile für Zeile erneut aufrufen lässt – oder das man an andere weitergeben kann.
- Um solchen Code verständlich zu machen (nicht nur für andere, sondern auch für sich selbst – sonst kann man z.B. nach längerer Pause böse Überraschungen erleben!), sollte man ihn ausgiebig kommentieren.
- Kommentare werden mit einem Doppelkreuz (#, engl. *hash*) markiert:



```
1 2 + 4
2 5 * 11
3 # Dies ist ein Kommentar.
4 3 ^ 2 # 3 hoch 2
5 9 ^ 0.5 # 9 hoch 0,5 (= wurzel aus 9)
6 # Die folgende Code-Zeile ist auskommentiert:
7 # 4 * 15
8
```

Erste Schritte (3)

- Code-Zeilen, die (gerade) nicht ausgeführt werden sollen, kann man auskommentieren, indem man ihnen das Kommentarzeichen voranstellt.
- R-Studio markiert Kommentare netterweise grün.
- Schreib ein, zwei weitere Berechnungen in dein Skript (du kannst z.B. eine längere Berechnung mit mehr als zwei Zahlen und mit Klammern ausprobieren). Führe eine davon aus, kommentiere sie dann aus und probiere erneut, sie auszuführen – diesmal solltest du in der Konsole kein Ergebnis erhalten.

Funktionen (1)

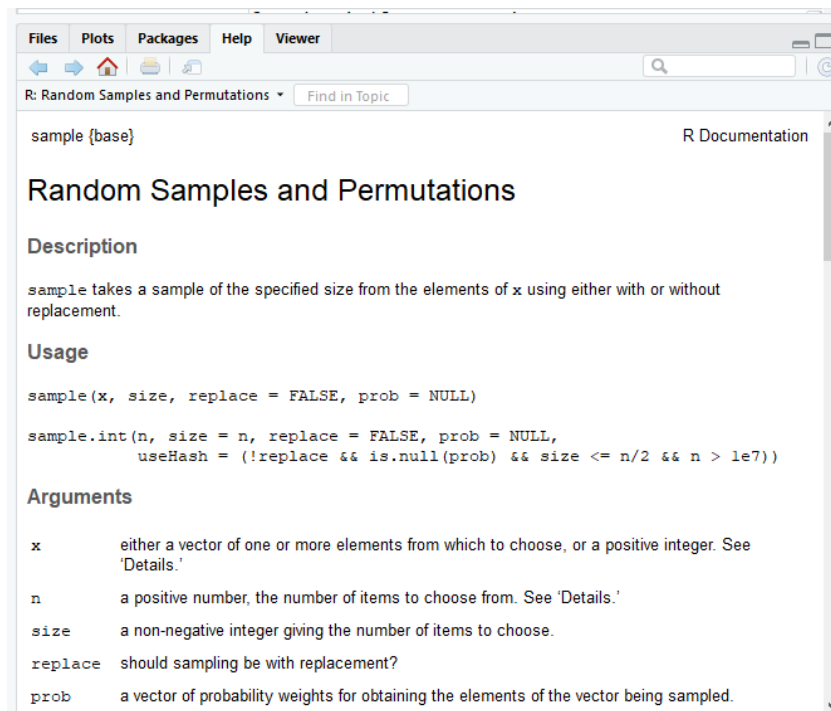
- Schreib folgenden Befehl in dein Skript und führe ihn aus:
sqrt(9)
- Als Ergebnis erhältst du die Wurzel von 9. *sqrt()* ist eine Funktion, zu erkennen an den runden Klammern.
- Der Text vor den Klammern ist der Funktionsname (*sqrt* steht – wenig überraschend – für *square root*). Funktionsnamen können lang oder kurz sein, wobei häufig verwendete Funktion oft kurze Namen haben (Linguisten kennen dieses Prinzip ...).
- In den Klammern lassen sich der Funktion Argumente übergeben. Die mögliche Zahl von Argumenten hängt von der Funktion ab, und es gibt obligatorische und fakultative Argumente. Wird ein fakultatives Argument nicht übergeben, wird dafür ein (in der Funktion definierter) Standardwert verwendet.

Funktionen (2)

- Auch Berechnungen oder Funktionen können Argumente von Funktionen sein. Probier' folgende Befehle in deinem Skript aus:
sqrt(9+7)
sqrt(sqrt(4096))
- Solche Verschachtelungen werden immer von innen nach außen aufgelöst.
- Lassen sich einer Funktion mehrere Argumente übergeben, dann werden diese mit Kommata voneinander abgetrennt. Folgender Befehl gibt z.B. fünf Zufallszahlen von 1 bis 10 aus:
sample(1:10, 5)
- *1:10* gibt den Wertebereich an, aus dem Zahlen gezogen werden sollen (du kannst *1:10* testweise einmal auf der Konsole oder im Skript ausführen), *5* die Anzahl zu ziehender Zahlen.

Funktionen (3)

- Woher weiß man, welche Argumente man einer Funktion überhaupt übergeben kann – und in welcher Reihenfolge? Und woher weiß man, was eine Funktion überhaupt tut?
- Ruf die eingebaute Hilfe zu *sample()* auf:
?sample



The screenshot shows the R Documentation window for the `sample` function. The window has a menu bar with 'Files', 'Plots', 'Packages', 'Help', and 'Viewer'. Below the menu bar is a search bar and a 'Find in Topic' button. The main content area is titled 'R: Random Samples and Permutations' and contains the following sections:

- sample {base}**
- Random Samples and Permutations**
- Description**

`sample` takes a sample of the specified size from the elements of `x` using either with or without replacement.
- Usage**

```
sample(x, size, replace = FALSE, prob = NULL)
sample.int(n, size = n, replace = FALSE, prob = NULL,
           useHash = (!replace && is.null(prob) && size <= n/2 && n > 1e7))
```
- Arguments**

<code>x</code>	either a vector of one or more elements from which to choose, or a positive integer. See 'Details.'
<code>n</code>	a positive number, the number of items to choose from. See 'Details.'
<code>size</code>	a non-negative integer giving the number of items to choose.
<code>replace</code>	should sampling be with replacement?
<code>prob</code>	a vector of probability weights for obtaining the elements of the vector being sampled.

Funktionen (4)

- Der Abschnitt *Usage* zeigt uns die Argumente der Funktion, ihre Namen, ihre Standardreihenfolge und ihre Standardwerte:
sample(x, size, replace = FALSE, prob = NULL)
- Unter *Arguments* werden diese Argumente näher beschrieben.
- *sample()* nimmt also bis zu vier Argumente. *replace* und *prob* haben Standardwerte, die ihnen mit Gleichheitszeichen zugewiesen sind (*FALSE* bzw. *NULL*). Wenn man der Funktion diese Argumente nicht übergibt, nehmen sie diese Standardwerte an – das macht die Argumente fakultativ. *x* und *size* dagegen haben keine Standardwerte und müssen der Funktion immer übergeben werden.
- Die Namen der Argumente braucht man nicht, es sei denn, man weicht von der Standardreihenfolge der Argumente ab. Beispiele auf der nächsten Seite!

Funktionen (5)

- Wir rufen *sample()* diesmal mit drei Argumenten auf: *replace* (Standardwert: *FALSE*) kommt hinzu. Dies regelt, ob mit oder ohne Zurücklegen gezogen wird, ob also Zahlen auch mehrmals gezogen werden können. *FALSE* heißt hier nein, *TRUE* ja. Standardmäßig könnten also z.B. aus der Zahlenmenge *1:10* (1 bis 10) maximal zehn Zahlen gezogen werden, da jede nur einmal gezogen werden kann (zum Ausprobieren: *sample(1:10, 11)* produziert eine Fehlermeldung).
- Da *replace* das dritte Argument in der Standardreihenfolge ist, funktioniert folgendes:
sample(1:10, 11, TRUE)
- Man kann aber auch die Argumentnamen angeben, was gerade bei vielen Argumenten übersichtlicher und verständlicher ist:
sample(x = 1:10, size = 11, replace = TRUE)

Funktionen (6)

- Argumentnamen anzugeben, hat auch den Vorteil, dass man die Standardreihenfolge der Argumente ignorieren kann (weil R so trotzdem weiß, welcher Wert welchem Argument zugeordnet ist):

sample(replace = TRUE, x = 1:10, size = 11)

- Die Namen der Argumente darf man übrigens auch abkürzen, solange sie dabei eindeutig zuordenbar bleiben:

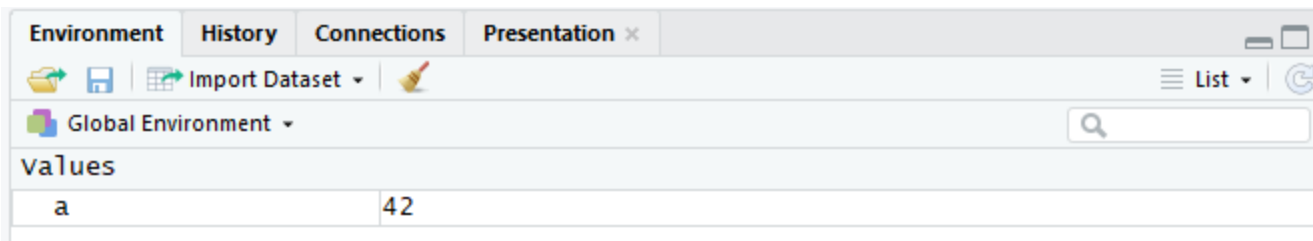
sample(rep = TRUE, x = 1:10, s = 5)

Objekte/Variablen (1)

- Um Werte, Datenstrukturen, Ergebnisse oder auch eigene Funktionen zu speichern, brauchen wir *Objekte*.
- Dabei geht es nicht um die Speicherung auf der Festplatte, sondern in der Umgebung von R (*Environment* der aktuellen Session).
- Jedes Objekt braucht einen Namen, über den man wieder darauf zugreifen kann.
- Speichern wir nun einmal den Wert 42. Dem Objekt geben wir den Namen *a*:
a <- 42
- **Hinweis zur Terminologie:** Mitunter spricht man auch einfach von *Variablen*, wenn man Objekte meint – besonders, wenn es um die Speicherung einfacher Werte geht. Das ist nicht weiter schlimm, kann aber zu Verwirrung führen, weil man in der Statistik auch von statistischen Variablen spricht.

Objekte/Variablen (2)

- Wenn du die Code-Zeile von eben ausführst, erhältst du keine Rückmeldung in der Konsole – aber in RStudio solltest du rechts oben unter *Environment* das neue Objekt sehen:



- Lass R nun den Wert des Objekts in der Konsole ausgeben, indem du einfach seinen Namen als Kommando eingibst:
a
- Erstelle ein zweites Objekt *b*, dem du den Wert von *a* zuweist:
b <- a

Objekte/Variablen (3)

- Wird einem Objekt ein Wert zugewiesen, wird erst dieser Wert ermittelt und dann zugewiesen. Daher ist z.B. folgendes möglich:
 $a \leftarrow a * a$
- Welchen Wert hat a nun? Ändert sich dadurch auch der Wert von b ?
- **Anmerkung zu Objektnamen:** Namen sollten mit einem Buchstaben beginnen, danach sind auch Zahlen und einige Sonderzeichen erlaubt, insbesondere Punkte und Unterstriche (die oft zur Strukturierung anstelle von Leerzeichen o.ä. verwendet werden).*
- **Achtung:** R ist pingelig, was Groß- und Kleinschreibung angeht!

* Es gibt mehr Möglichkeiten, die für unsere Zwecke aber nicht relevant sind und daher eher verwirrend wären.

Objekte/Variablen (4)

- Alle* Objekte der aktuellen Umgebung (*Environment*) sieht man nicht nur rechts oben in RStudio, sondern kann man sich auch mit der Funktion *ls()* anzeigen lassen:

ls()

Dies ist übrigens einmal eine Funktion, der man keine Argumente übergeben muss.

- Um Objekte wieder loszuwerden, kann man die Funktion *rm()* verwenden (*rm* steht für *remove*):

rm(a)

- Wenn du versuchst, das Objekt *a* nach diesem Befehl wieder aufzurufen, erhältst du eine Fehlermeldung:

```
> rm(a)
> a
Error: object 'a' not found
> |
```

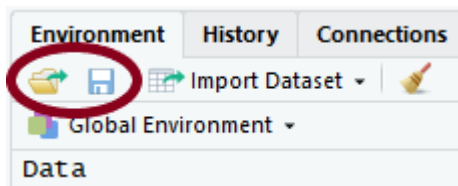
* Außer solche, deren Namen mit einem Punkt beginnen (diese sind „versteckt“). Dafür gibt es dann *ls(all.names = TRUE)* ...

Objekte/Variablen (5)

- Wenn man die Umgebung aufräumen möchte (oder muss ...), kann man auch sämtliche Objekte auf einmal löschen:
`rm(list = ls())`
- Falls man nicht von vornherein mit verschiedenen Projekten in RStudio arbeitet, bietet es sich eventuell an, diesen Befehl an den Anfang eigener Skriptdateien zu setzen, um „sauber“ beginnen zu können.

Arbeitsverzeichnis festlegen (1)

- Objekte der aktuellen Session (der *Workspace*) können beim Beenden von RStudio auf der Festplatte gespeichert werden, sodass sie beim nächsten Start wieder automatisch zur Verfügung stehen (siehe S. 26).
- Man kann einen *Workspace* auch manuell speichern und laden: entweder mit den Funktionen *save()* und *load()* oder in RStudio über die entsprechenden Symbole im Bereich rechts oben (was noch einfacher ist):



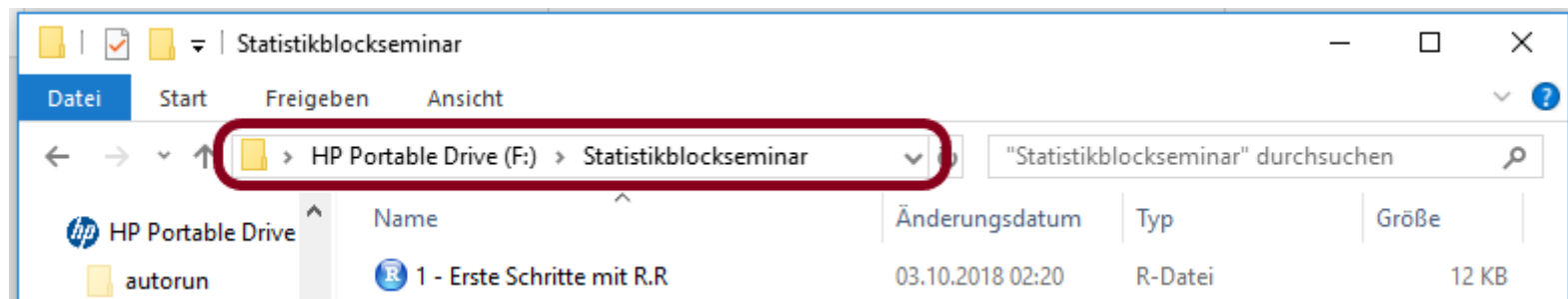
- Standardmäßig (also z.B. beim Beenden) wird der *Workspace* im aktuellen Arbeitsverzeichnis (*working directory*) gespeichert.

Arbeitsverzeichnis festlegen (2)

- Neben dem *Workspace* werden auch die in der Session eingegebenen Befehle (*history*) im Arbeitsverzeichnis gespeichert.
- Praxisrelevant ist aber v.a., dass Dateien, die man einlesen möchte (üblicherweise Tabellen), um damit in R zu arbeiten, zuerst im aktuellen Arbeitsverzeichnis gesucht werden (wenn man nicht den kompletten Dateipfad angeben möchte).
- Deshalb ist es oft sinnvoll, als Arbeitsverzeichnis einen Ordner festzulegen, in dem sich die Daten befinden, mit denen man arbeiten möchte.
- Lass dir zunächst das aktuelle Arbeitsverzeichnis anzeigen:
getwd()

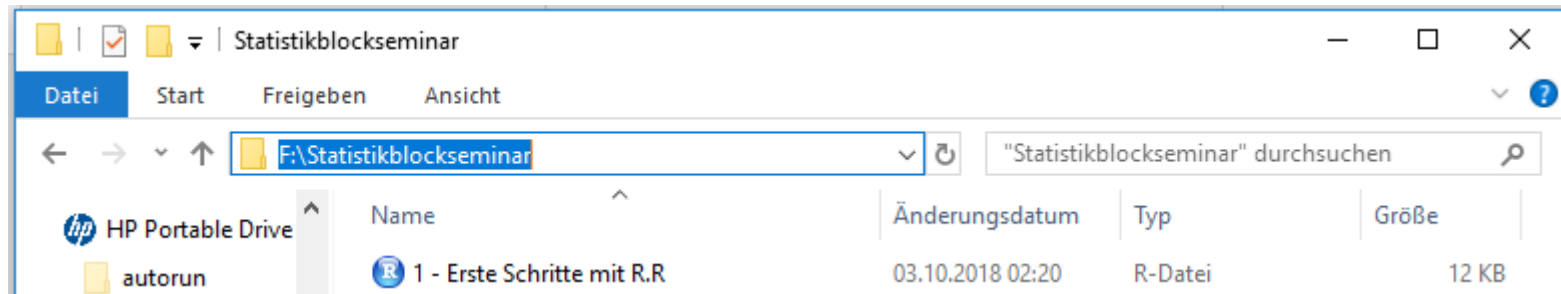
Arbeitsverzeichnis festlegen (3)

- Um das Arbeitsverzeichnis zu ändern, brauchst du den Befehl `setwd()` (*wd* steht für *working directory*).
- Ein Blick in die Hilfe (`?setwd`) zeigt, dass `setwd()` nur ein Argument benötigt, nämlich einen Dateipfad.
- Einen Dateipfad kann man (unter Windows) z.B. herausfinden, indem man den entsprechenden Ordner aufruft und in die Adressleiste klickt:



Arbeitsverzeichnis festlegen (4)

- Der Pfad lässt sich dann leicht kopieren:

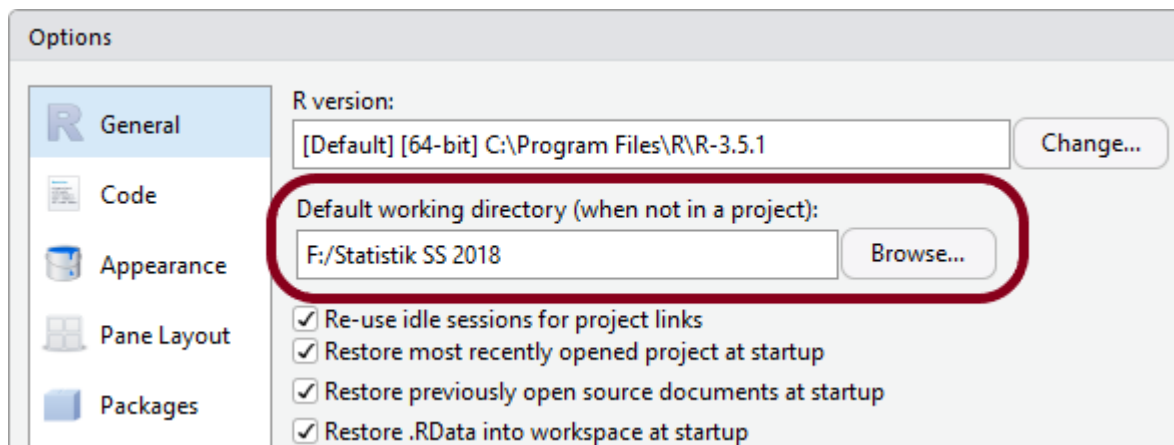


- Ärgerlicherweise mag R das Windows-Format mit Backslashes („\“) nicht. Jeder Backslash muss entweder durch einen normalen Schrägstrich („/“) ersetzt werden oder mit einem weiteren Backslash davor „maskiert“ werden.*
- Der Dateipfad selbst wird von Anführungszeichen umschlossen (keine typographischen!) – Hochkommata funktionieren auch.
- Um also den Ordner oben als Arbeitsverzeichnis festzulegen:
`setwd("F:/Statistikblockseminar")`

* Siehe <https://de.wikipedia.org/wiki/Maskierungszeichen>

Arbeitsverzeichnis festlegen (5)

- Lege in deinem Skript ein eigenes Arbeitsverzeichnis fest. Idealerweise legst du dafür einen Ordner für dieses Blockseminar an, in dem du die Materialien des Seminars sowie eigene Skripte speichern kannst.
- In RStudio lässt sich unter *Tools* → *Global Options...* auch ein Standardarbeitsverzeichnis festlegen:



Pakete installieren und einbinden (1)

- R stellt standardmäßig bereits eine ganze Menge an Funktionen bereit. Diese Menge lässt sich über Pakete (*packages*) noch erheblich erweitern.
- Ein Paket ist eine Sammlung von Funktionen und/oder Datensätzen, meist mit einem gemeinsamen Einsatzgebiet (z.B. Graphikerstellung, spezielle statistische Verfahren oder Korpusverwaltung und -analyse).
- Pakete werden heruntergeladen (oder selbst erstellt) und lokal (z.B. auf der Festplatte) gespeichert. Die Menge dieser installierten Pakete kann man sich als Bibliothek (*library*) vorstellen, aus der man bei Bedarf in der aktuellen Session einzelne ausleihen (➔ aktivieren) kann.
- Erst wenn ein Paket in der Session aktiviert wurde, kann man auf die darin enthaltenen Funktionen und Datensätze zugreifen.*

* Nicht ganz korrekt, man kann auch über `paketname::funktion()` ohne vorherige Aktivierung auf einzelne Funktionen zugreifen. Das tut man normalerweise aber seltener, v.a. dann, wenn mehrere Pakete verschiedene Funktionen mit denselben Namen bereitstellen – also zur eindeutigen Identifizierung.

Pakete installieren und einbinden (2)

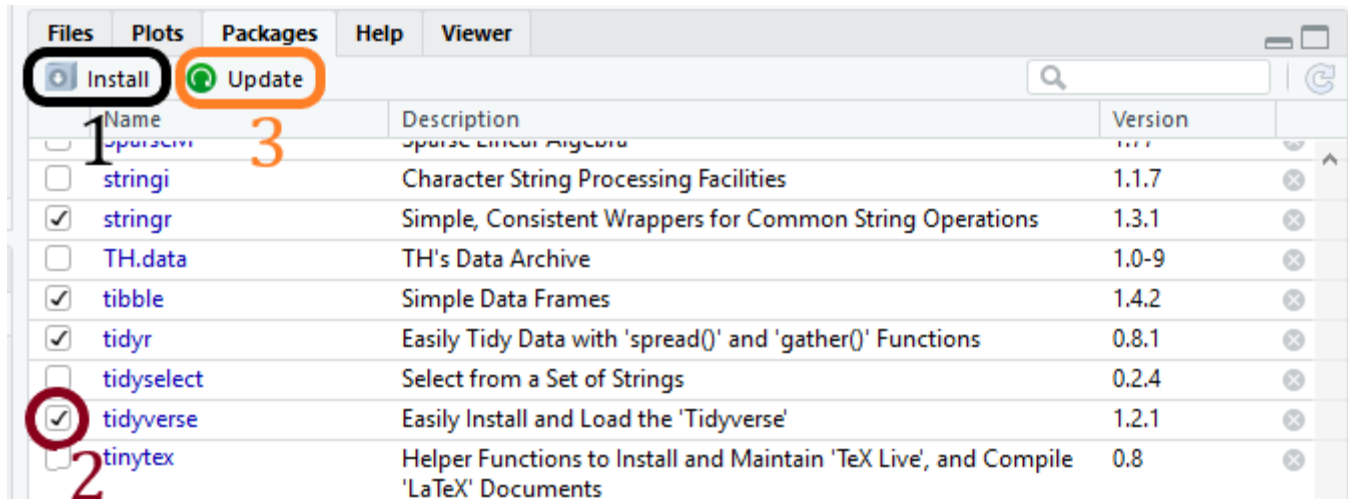
- Um ein Paket (oder eine Menge von Paketen) zu installieren, gibt es standardmäßig die Funktion *install.packages()*. Als Argument erhält die Funktion entweder
 - den Namen des Pakets in (nicht-typographischen) Anführungszeichen:
install.packages("tidyverse")
oder
 - einen Vektor (dazu gleich mehr) mit mehreren Paketnamen:
install.packages(c("car", "corrplot"))
- Bei der Installation eines Pakets werden standardmäßig auch Pakete mitinstalliert, von denen das neue Paket abhängig ist – also nicht erschrecken, wenn plötzlich ganz viele Pakete heruntergeladen werden.

Pakete installieren und einbinden (3)

- Um ein installiertes Paket in der aktuellen Session zu aktivieren/einzubinden, verwendet man die Funktion *library()*.
- Als Argument wird der Name des Pakets angegeben (Anführungszeichen können hierbei weggelassen werden):
library(tidyverse)

Pakete installieren und einbinden (4)

- In RStudio kann man im Bereich rechts unten unter *Packages* auch Pakete installieren (1) und installierte Pakete mit Mausklicks aktivieren und wieder deaktivieren (2):



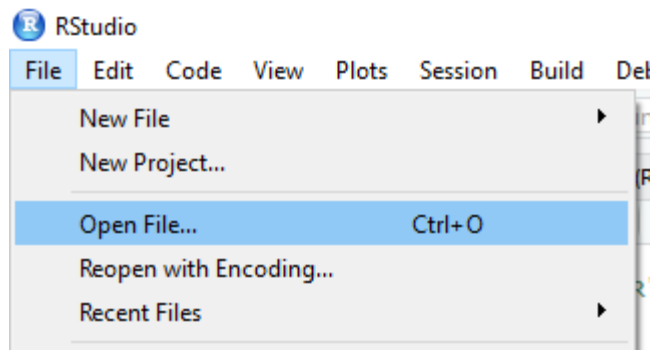
- Installierte Pakete lassen sich hier auch einfach aktualisieren (3).

Pakete installieren und einbinden (5)

- Ausführliche Informationen zu Paketen – auch für Fortgeschrittene – gibt es hier: <https://www.datacamp.com/community/tutorials/r-packages-guide>

Weiter in der Skriptdatei ... (1)

- Weiter geht es direkt in RStudio mit der Datei „Erste Schritte mit R.Rmd“.
- Um sie zu öffnen, einfach die Datei über *File* → *Open File...* auswählen:



Weiter in der Skriptdatei ... (2)

- Sollten Umlaute und Sonderzeichen im Skript nicht richtig dargestellt werden, klicke bitte auf *File* → *Reopen with Encoding...*, wähle dort *UTF-8* aus, setze den unteren Haken (damit du das bei weiteren Skriptdateien nicht immer wieder tun musst) und bestätige mit *OK*. Nun stimmt hoffentlich alles!

