

# Erste Schritte mit R

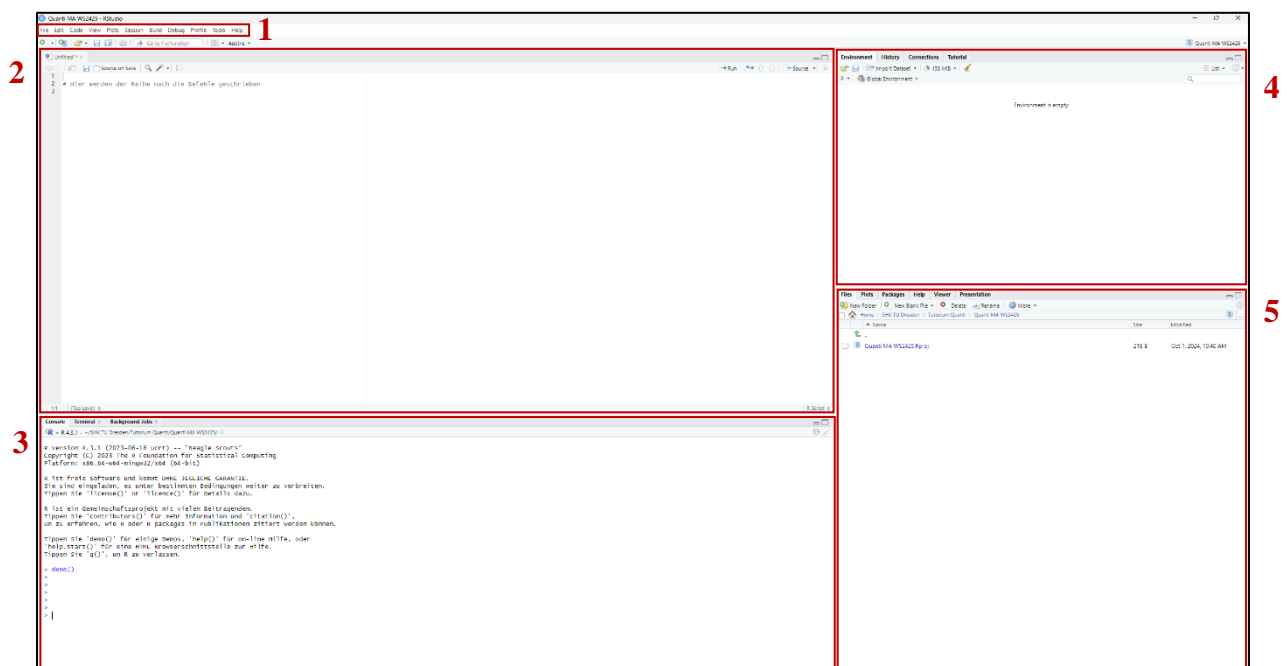
## 1 Grundlegende Information:

R ist eine kostenlose und frei verfügbare skriptbasierter Programmiersprache für statistische Berechnungen und Grafiken. Während in SPSS statistische Berechnungen mit Hilfe einer grafischen Benutzeroberfläche möglich sind, muss in R jede Berechnung durch einen geschriebenen Befehl in einer Konsole durchgeführt werden. R ist OpenSource und bietet dementsprechend viele Vorteile. So werden von einer breiten wissenschaftlichen Community fortlaufend neue Berechnungsmethoden und statistische Verfahren bereitgestellt, welche nur selten in SPSS implementiert sind.

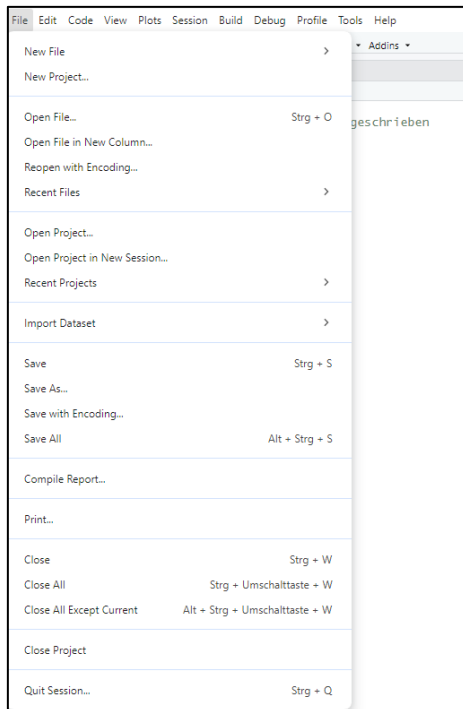
### 1.1 R-Studio

R-Studio ist eine zusätzliche grafische Oberfläche, welche die Nutzung von R als Programmiersprache vereinfacht. Sie wird parallel mit R heruntergeladen und installiert.

Im folgenden die wichtigsten Bestandteile in R-Studio.



1

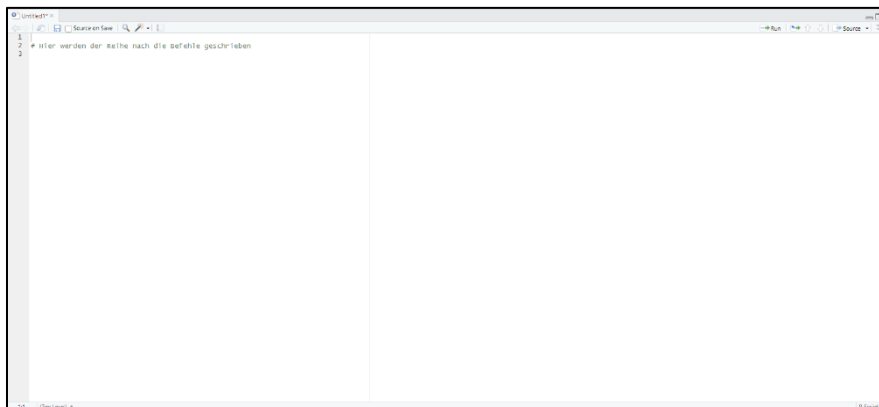


In dem oberen Registerfeld können die wichtigsten Funktionen genutzt werden. Das Laden von kompletten Skripten, die Generierung neuer Projekte und setzen von Arbeitspfaden (Wo Daten geladen und gespeichert werden sollen) ist hier möglich ohne Nutzung der Konsole.

---

In dem oberen linken Fenster finden sich die Skripte, welche vorab geschrieben wurden. Hier

2

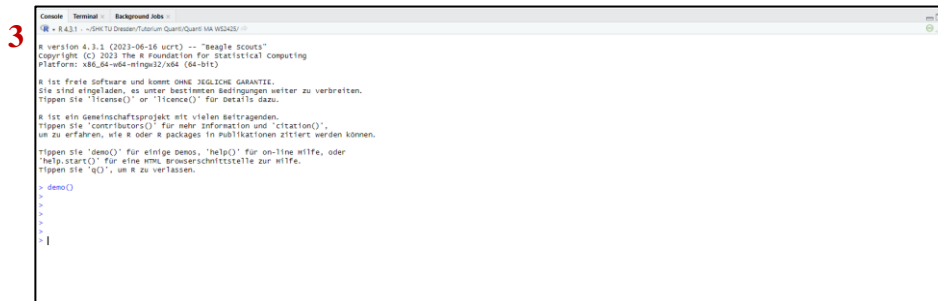


werden die Befehle der Reihe nach eingeschrieben und mit STRG + Enter ausgeführt.

Neben den Befehlen können auch Kommentare gesetzt

werden, welche einen besseren Überblick für Personen schaffen kann, die jenes Skript nicht geschrieben haben. Dies ist entweder mit # oder mit , ' möglich. Kommentare werden Grün hervorgehoben. Insbesondere bei komplexeren Skripten empfiehlt sich die Verwendung von Kommentaren.

In dem unteren linken Fenster befindet sich die Befehlskonsole von R. Ohne die Oberfläche R-Studio, würde nur dieses Feld für statistische Berechnungen bereitstehen. In diesem Feld werden die zuvor geschriebenen Befehle ausgeführt. Findet die Bearbeitung eines Befehles

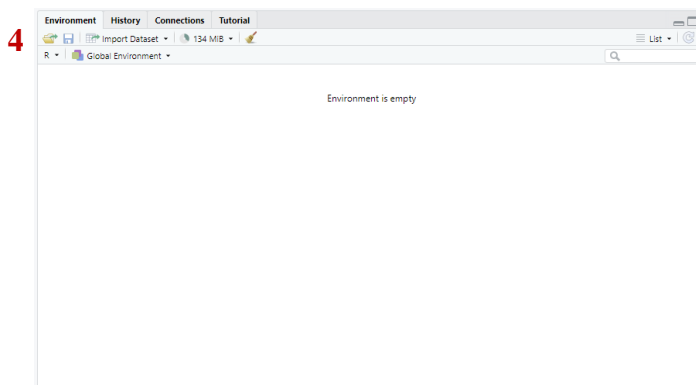


statt, so erscheint im oberen rechten Eck ein rotes STOP-Symbol. Bei ungewollter längerer Ausführung oder bei

fehlerhaften Befehlen, kann es wenigen Fällen nötig sein durch ein Klicken auf dieses Symbol den Bearbeitungsprozess zu beenden. Darauf hin kommt es zu einer Fehlermeldung.

In der Konsole finden sich zudem nach der Ausführung des Befehls die Ergebnisse einer Berechnung. Kommt es zu Fehlern, zum Beispiel eine Division durch 0, so werden in diesem Feld rot gefärbte Fehlermeldungen ausgegeben.

In der oberen rechten Ecke befindet sich das Environment. Hier werden alle geladenen und gesetzten Variablen und Datensätze angezeigt. Eine wichtige Information kann dabei der mitangezeigte Datentyp sein. Sollte zum Beispiel die Zahl 42 als Zeichenkette (als String) gespeichert sein, so ist es nicht möglich sie für Berechnungen zu verwenden. In dem Falle erkennt R (wie auch viele weitere Programmiersprachen) die 42 als eigenständiges Wort und nicht mehr als Zahl<sup>1</sup>. Es ist dann nötig mit einem Befehl den Datentyp zu ändern.



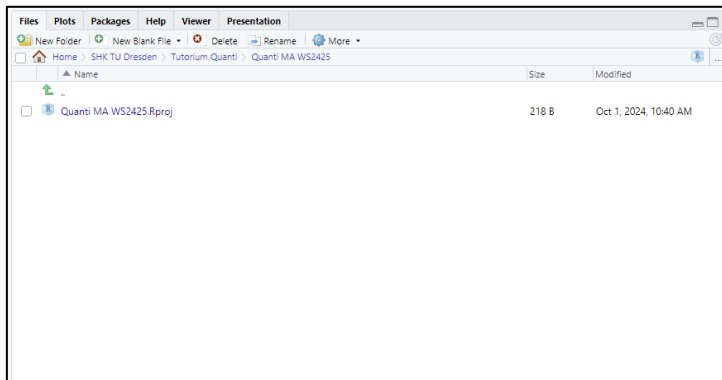
Mit einem Klick auf den angezeigt Datensatz oder Variable können die Inhalte in einem separaten Fenster angeschaut werden. Diese öffnet sich dann in dem linken oberen Fenster, wo auch die Skripte angezeigt werden.

Neben den Variablen werden auch Informationen, wie der verwendete Arbeitsspeicher, angezeigt. Mit einem Klick auf den Besen werden alle Datensätze und Variablen aus dem Environment entfernt. Dies kann in manchen

<sup>1</sup> Für einen besseren Einblick beispielsweise hier: <https://databraineo.com/ki-training-resources/r-programmierung/datentypen-in-r-einfach-erklart/> vorbeischaun

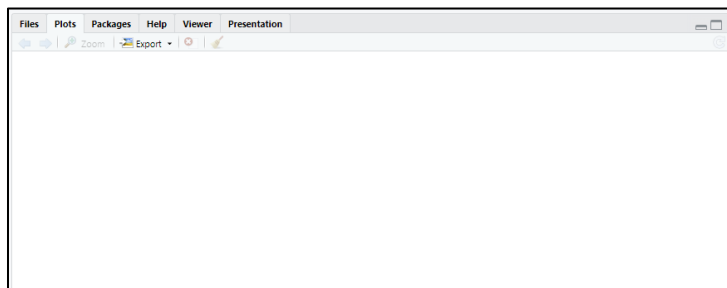
Fällen hilfreich sein, wenn der Speicher voll ist oder ein neues Skript zur Berechnung von neuen Datensätzen verwendet wird.

5



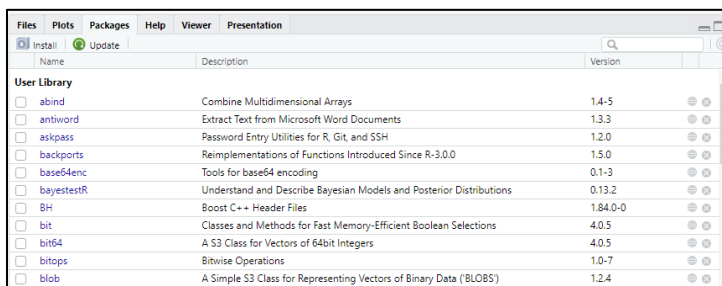
In dem rechten unteren Fenster werden die Dateien in dem aktuellen Arbeitspfad, generierte Grafiken, geladene Pakete und die Hilfeseite angezeigt.

Die Grafiken können direkt mit über die Schaltfläche „Export“ gespeichert werden. Alternativ



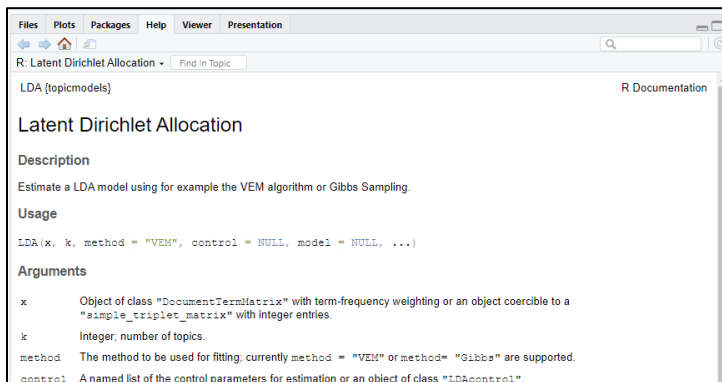
ist dies auch über Befehle wie `png()` möglich. Über die Pfeiltasten können vorher ausgeführt Grafiken erneut angeschaut werden.

Pakete sind ein wichtiger Bestandteil in der Arbeit mit R. Pakete liefern vorgefertigte



Funktionen und Befehle, welche wir für die Analyse von Daten benötigen. In diesem Fenster können Pakete aktiviert oder auch deaktiviert werden. Dies ist nötig, um auf die Befehle in den Paketen

zuzugreifen. Alternativ geschieht das geläufig durch den Befehl `library()`. Neben der Standardausrüstung die R bereit hält (z.B. Paket „base“) sind auch zusätzliche Pakete aus der Community wie „dplyr“ und „ggplot2“ von hoher Relevanz. Ersteres vereinfacht die Programmierung, zweiteres ermöglicht komplexere grafische Ausgaben.

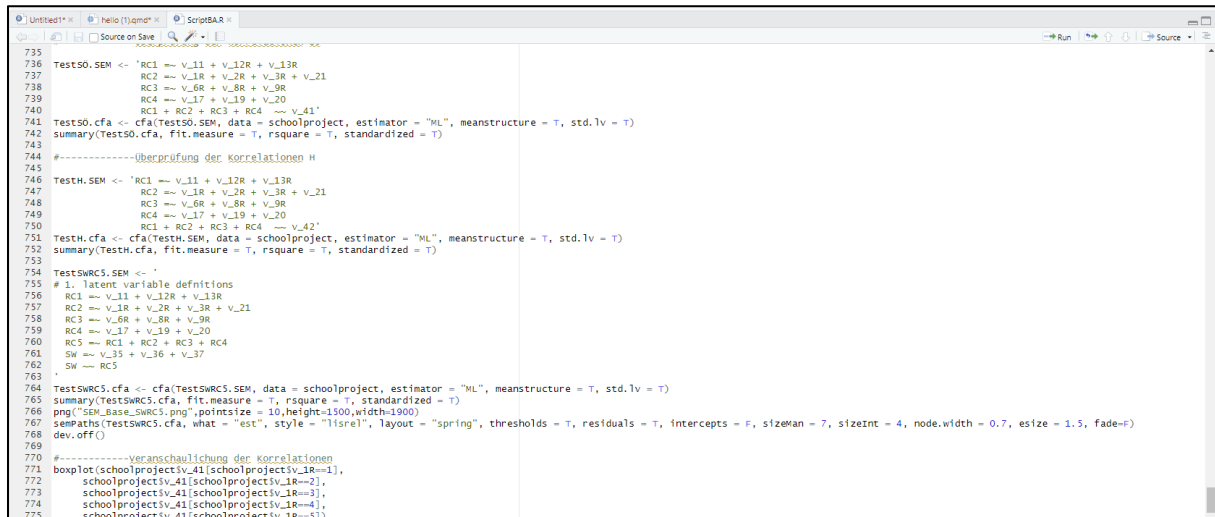


Die Hilfeseite ist sprichwörtlich dein bester Freund. Hier werden Struktur und theoretischer Hintergrund von Befehlen ausführlich aufgezeigt. Alternativ kann die Suche nach einem Befehl mit einem `?Befehl` oder `??Befehl` in

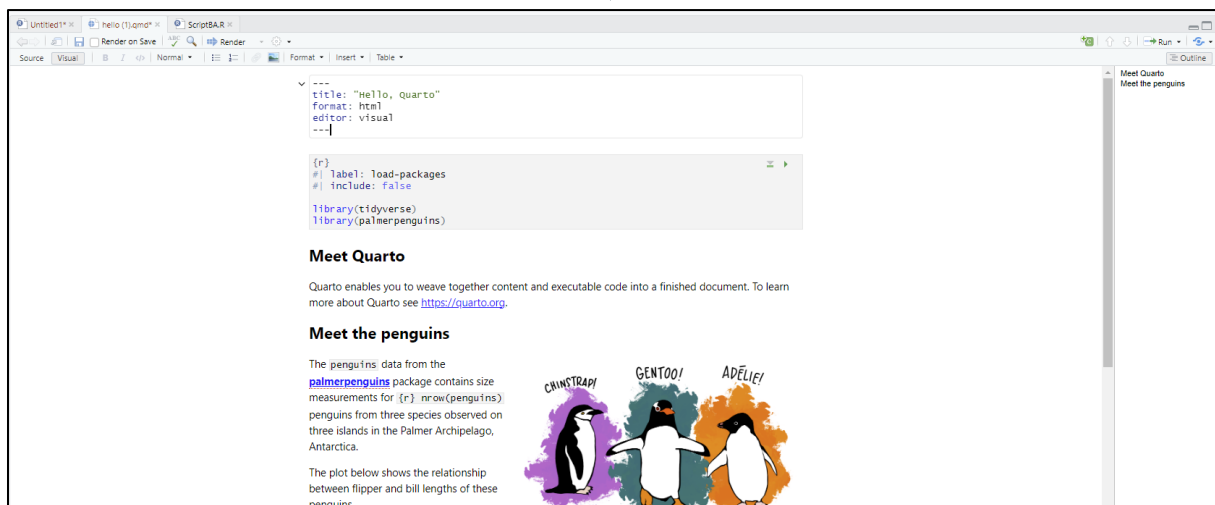
der Konsole erfolgen. Sollte ein Befehl nicht funktionieren, ist dies die erste Anlaufstelle um die Ursache des Problems zu finden.

## 1.2 Quarto

Neben R-Studio ist ebenso die Verwendung von Quarto zu empfehlen. Hier wird das zuvor beschriebene obere linke Fenster in R-Studio angepasst.



```
735
736 TestS0.SEM <- 'RC1 =~ V_11 + V_12R + V_13R
737             RC2 =~ V_1R + V_2R + V_3R + V_21
738             RC3 =~ V_6R + V_8R + V_9R
739             RC4 =~ V_17 + V_19 + V_20
740             RC1 + RC2 + RC3 + RC4 =~ V_41'
741 TestS0.cfa <- cfa(TestS0.SEM, data = schoolproject, estimator = "ML", meanstructure = T, std.lv = T)
742 summary(TestS0.cfa, fit.measures = T, rsquare = T, standardized = T)
743
744 #-----Überprüfung der Korrelationen H
745
746 TestH.SEM <- 'RC1 =~ V_11 + V_12R + V_13R
747             RC2 =~ V_1R + V_2R + V_3R + V_21
748             RC3 =~ V_6R + V_8R + V_9R
749             RC4 =~ V_17 + V_19 + V_20
750             RC1 + RC2 + RC3 + RC4 =~ V_42'
751 TestH.cfa <- cfa(TestH.SEM, data = schoolproject, estimator = "ML", meanstructure = T, std.lv = T)
752 summary(TestH.cfa, fit.measures = T, rsquare = T, standardized = T)
753
754 TestSWRCS.SEM <- '
755 # 1. latent variable definitions
756 RC1 =~ V_11 + V_12R + V_13R
757 RC2 =~ V_1R + V_2R + V_3R + V_21
758 RC3 =~ V_6R + V_8R + V_9R
759 RC4 =~ V_17 + V_19 + V_20
760 RC5 =~ RC1 + RC2 + RC3 + RC4
761 SW =~ V_35 + V_36 + V_37
762 SW =~ RC5
763 '
764 TestSWRCS.cfa <- cfa(TestSWRCS.SEM, data = schoolproject, estimator = "ML", meanstructure = T, std.lv = T)
765 summary(TestSWRCS.cfa, fit.measures = T, rsquare = T, standardized = T)
766 png("SEM_Base_SWRCS.png", pointsize = 10, height=1500, width=1900)
767 semPaths(TestSWRCS.cfa, what = "est", style = "lslrel", layout = "spring", thresholds = T, residuals = T, intercepts = F, sizeMan = 7, sizeInt = 4, node.width = 0.7, esize = 1.5, fade=F)
768 dev.off()
769
770 #-----Veranschaulichung der Korrelationen
771 booplot(schoolproject$V_41[schoolproject$V_1R==1],
772        schoolproject$V_41[schoolproject$V_1R==2],
773        schoolproject$V_41[schoolproject$V_1R==3],
774        schoolproject$V_41[schoolproject$V_1R==4],
775        schoolproject$V_41[schoolproject$V_1R==5])
```



Quarto ermöglicht eine leichtere Nachvollziehbarkeit der Skripte, sowie eine handlichere Ausführung. Ebenso ist es möglich die Skripte direkt in GitHub zu veröffentlichen<sup>2</sup>. Dies kann bei der Veröffentlichung von Forschungsergebnissen eine große Hilfe sein. Ebenso ist es möglich mit Quarto interaktive Oberflächen auf einem separaten Server (teilweise kostenpflichtig) für interessierte Personen bereitzustellen. Quarto-Dateien sind im Gegensatz zu einfachen R-Skripten mit dem Datenzusatz `.qmd` versehen.

<sup>2</sup> Mehr dazu hier: <https://quarto.org/docs/publishing/github-pages.html>

## 2 Download und Einrichtung von R, R-Studio und Quarto

R und R-Studio lassen sich ganz von der folgenden Website herunterladen:

<https://posit.co/download/rstudio-desktop/>

**1: Install R**

RStudio requires R 3.6.0+. Choose a version of R that matches your computer's operating system.

*R is not a Posit product. By clicking on the link below to download and install R, you are leaving the Posit website. Posit disclaims any obligations and all liability with respect to R and the R website.*

**DOWNLOAD AND INSTALL R**

**2: Install RStudio**

Size: 265.55 MB | SHA-256: 513216FE | Version: 2024.09.0+375 | Released: 2024-09-23

**The Comprehensive R Archive Network**

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux (Debian, Fedora/Redhat, Ubuntu)
- Download R for macOS
- Download R for Windows

**Auswahl nach entsprechendem Betriebssystem**

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Subdirectories:

[base](#) Binaries for base distribution. This is what you want to **install R for the first time**.

[contrib](#) Binaries for contributed packages. This is what you want to install R packages (for R 3.0.0+).

**R-4.4.1 for Windows**

**Download R-4.4.1 for Windows** (82 megabytes, 64 bit)

[README on the Windows binary distribution](#)

[New features in this version](#)

This build requires UCRT, which is part of Windows since Windows 10 and Windows Server 2016. On older systems, UCRT has to be installed manually from [here](#).

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the [md5sum](#) of the .exe to the [fingerprint](#) on the master server.

[Frequently asked questions](#)

- Does R run under my version of Windows?
- How do I update packages in my previous version of R?

Quarto muss separat unter folgendem Link <https://quarto.org/docs/get-started/> heruntergeladen werden. Für einen ersten Eindruck empfiehlt es sich die folgende Seite: <https://quarto.org/docs/get-started/hello/rstudio.html>.

Nach der Installation ist es wichtig den Rechner neu zu starten, andernfalls kann es zu Problemen bei der Ausführung von R kommen.

## 3 Schreiben in R

### 3.1 Zu Beginn

*Hinweis: Die farblichen Hervorhebungen weichen aufgrund des Exportes zu Word von der Erscheinung in R-Studio ab. Die originale .qmd-Datei findet sich auf Opal.*

R ist wie bereits erwähnt eine skriptbasierte Sprache. Das bedeutet wir führen unsere Berechnungen mit geschriebenen Befehlen aus. Doch bevor wir dazu kommen, erstmal grundlegende Informationen.

Sofern dieses Skript das erste Mal geöffnet ist sollte das linke obere Fenster "Environment" noch leer sein. Dort werden die Variablen angezeigt, welche wir zum Berechnen verwenden. Der erste Schritt wird demnach erstmal sein, probeweise **Variablen anzulegen**

Dies funktioniert wie folgt:

*Hinweis: Das ausführen eines Befehls in einer Zeile ist durch die Tastenkombination STRG + Enter möglich. Soll ein kompletter Abschnitt ausgeführt werden, so kann dieser markiert und mit jener Tastenkombination ausgeführt werden. Die visuelle Oberfläche von Quarto ermöglicht ebenso die Nutzung von Buttons, welche sich in den "executable cells" (Befehlszellen) oben rechts befinden. Damit kann die komplette grau hinterlegte Box ausgeführt werden.*

*# Mit dem "<-" weisen wir einer selbstgewählten Variable (mit selbstgewählter Bezeichnung) einen bestimmten Wert zu.*

```
variable <- 42
```

*# Nun lassen wir uns diese Variable zur Überprüfung ausgeben.*

```
variable
```

```
[1] 42
```

Nun können wir in der Environment unsere Variable "variable" mit dem dazugehörigen Wert sehen.

Legen wir wie folgt weitere Variablen mit verschiedenen **Datentypen** an.

*# Wir weisen "x" wieder die Zahl 42 zu. x wird als Datentyp "numeric" gespeichert.*

```
x <- 42
```

*# y soll die Frage "Was ist der Sinn des Lebens" enthalten. Um dies zu ermöglichen müssen wir Textzeichen immer in " " anführen. y wird als Datentyp "character" gespeichert.*

```
y <- "Was ist der Sinn des Lebens?"
```

*# z soll einen Boolean-Wert (in R als Logical bezeichnet) beinhalten. Dieser ist wichtig, wenn wir eine Berechnung unter bestimmten Bedingungen durchf*

ühren wollen, die Bedingungen also WAHR oder FALSCH sind. Boolesche Werte sind in R "FALSE" oder "TRUE", sie können auch abgekürzt als "F" oder "T" geschrieben werden. Diese Werte werden von R-Studio hellblau gefärbt. z wird als Datentyp "logical" gespeichert

```
z <- FALSE
```

# v soll ein Vektor sein. Vektoren enthalten eine Reihe von Werten, welche v verschiedenen Datentypen angehören können. Die Werte werden in einer Reihe aufgelistet. Um einen Vektor zu generieren benötigen wir den Befehl c(). Zu den Befehlen gleich mehr.

# In dem Vektor kommt die Bezeichnung NA vor, dies steht stellvertretend für fehlende Werte. Ein fehlender Wert kann entstehen, wenn eine befragte Person in einem Fragebogen keine Auskunft gibt.

```
v <- c(1, 2, 4, 5, 1, "2", NA, 3, 7, 9)
```

Und nun zu den **Befehlen**. Ein Befehl beginnt immer mit der Bezeichnung des Befehles gefolgt von ( ). Besonders wichtig ist bei einem Befehl die Einhaltung der Syntax (der Grammatik der Programmiersprache/des Befehles). Auf der bereits erwähnten Hilfeseite findet sich in der Regel zu jedem Befehl eine detaillierte Beschreibung der Syntax.

Wir wollen nun probeweise den Mittelwert des Vektors "v" berechnen. Dies ist mit dem Befehl mean() möglich.

*Hinweis: Bei dem Schreiben von Befehlen empfiehlt R-Studio automatisch existierende Funktionen. Diese können dann mit TAB alternativ ausgewählt werden. Dies erspart bei längeren Funktionsnamen Zeit.*

```
mean(v)
```

```
Warning in mean.default(v): Argument ist weder numerisch noch boolesch: gebe NA zurück
```

```
[1] NA
```

Wir erhalten eine Warn-/Fehlermeldung und das Ergebnis lautet "NA" ("kein Wert")

Der "spaßigste" Teil beim Programmieren ist das Debuggen und nimmt in der Regel die meiste Zeit in Anspruch (insbesondere wenn eine Programmiersprache neu gelernt wird). Debuggen bedeutet nichts anderes als das Entfernen von Fehlern. Bei auftauchenden Fehlern kann entweder das Internet gefragt werden oder man überprüft zuerst die Syntax.

Zweiteres wollen wir mit ?mean() versuchen.

```
?mean()
```

```
starte den http Server für die Hilfe fertig
```

In dem unteren rechten Fenster erscheint nun die Informationsseite zu dem Befehl. Unter *Usage* sehen wir die Syntax und unter *Arguments* eine genauere Ausführung der Variablen, welche wir für die Berechnung verwenden dürfen, und Optionen, mit denen wir den Befehl anpassen können. Eine wichtige zusätzliche Option beim Befehl mean() ist das Ignorieren von NA-Werten. Optionen in einem Befehl werden durch ein Komma ( , ) abgegrenzt.

Da wir einen NA-Wert in dem Vektor haben, probieren wir dies.



```
mean(v, na.rm = TRUE)
```

```
Warning in mean.default(v, na.rm = TRUE): Argument ist weder numerisch noch  
boolesch: gebe NA zurück
```

```
[1] NA
```

Dies hat nicht funktioniert. Einen Blick auf die Warnmeldung zeigt an, dass das Argument (also v) keine numerischen oder booleschen Werte enthält. Ein Blick in das obere rechte Fenster (Environment) verdeutlicht dies. Hier wird v als charakter (chr) bestehend aus einer Reihe und zehn Spalten [1:10] angezeigt. Ebenso sind alle Zahlen in " " aufgeführt. Die Ursache dafür liegt in dem Wert "2", welchen wir mit in dem Vektor eingegeben haben. Da es sich hier um den Datentyp charakter handelt, wandelt R auch alle anderen Zahlen in den Vektor zu den Datentyp charakter um. R will dahingehend vereinheitlichen.

Um dieses Problem zu lösen können wir mit dem Befehl `as.numeric()` den Datentyp ändern.

```
v
```

```
[1] "1" "2" "4" "5" "1" "2" NA  "3" "7" "9"
```

```
as.numeric(v)
```

```
[1] 1  2  4  5  1  2 NA  3  7  9
```

In Verbindung mit dem Befehl `mean()` sieht dies nun wie folgt aus.

```
mean(as.numeric(v), na.rm = TRUE)
```

```
[1] 3.777778
```

Bisher haben wir v nur temporär geändert, wodurch wir in jeder weiteren Berechnung den Befehl `as.numeric` immer wieder verwenden müssten. Wir ersparen uns das, indem wir v überschreiben.

```
v <- as.numeric(v)
```

```
v
```

```
[1] 1  2  4  5  1  2 NA  3  7  9
```

## 3.2 Dateipfad setzen und suchen

Nachdem ein neues Projekt angelegt wurde, wird auch immer ein Arbeitsort festgelegt, in welchem das Projekt als solches, die generierten und zu ladenden Datensätze, wie auch die gespeicherten Grafiken oder auch Tabellen zu finden sind. Manchmal kann es notwendig sein diesen Ort zu ändern. Dies ist entweder in der Registerleiste oben links unter "Session" –> "Set Working Directory" oder über den Befehl `setwd()` möglich.

Bevor wir einen neuen Speicherort setzen können wir und mit `getwd()` unseren aktuellen einsehen

```
getwd()
```

```
[1] "C:/Users/Arbeitsdesktop/OneDrive/Dokumente/SHK TU Dresden/Tutorium Quanti/Quanti MA WS2425"
```

Und nun wird ein neuer Dateipfad gesetzt. (Beachte die Richtung des “/”)

```
# Beispiel:
setwd("C:/Users/Arbeitsdesktop/OneDrive/Dokumente/SHK TU Dresden/Tutorium Q
uant")
getwd()

[1] "C:/Users/Arbeitsdesktop/OneDrive/Dokumente/SHK TU Dresden/Tutorium Qua
nti/Quanti MA WS2425"
```

### 3.3 Pakete installieren und laden

Die Grundausstattung von R enthält viele Funktionen, wie der Berechnung des Mittelwertes, des Medians, der Standardabweichung, und so weiter. Es ist allerdings oftmals nötig zusätzliche Funktionen aus dem Internet zu importieren, wie zum Beispiel die Berechnung mittels neuer Methoden wie dem LDA (Latent Dirichlet Allocation, ein Methode der Textverarbeitung - Topic Modeling).

Diese zusätzlichen Funktionen finden sich in bestimmten Paketen, die mehrere weitere Funktionen beinhalten. Wichtige Pakete sind zum Beispiel: “dplyr”, “psych”, “car”, “ggplot2” oder auch “foreign”.

Diese Pakete müssen einmalig installiert und bei jedem Neustart von R aktiviert (geladen) werden.

Die Installation ist mit dem Befehl `install.packages("Paketname")` möglich:

```
install.packages("dplyr")
install.packages("tidyr")
install.packages("psych")
install.packages("car")
install.packages("ggplot2")
install.packages("foreign")
```

Mit dem Befehl `library(Paketname)` werden diese anschließend aktiviert. Dieser Befehl steht in der Regel immer am Anfang eines Skriptes.

```
library(dplyr)
library(tidyr)
library(psych)
library(car)
library(ggplot2)
library(foreign)
```

### 3.4 Youtube-Empfehlung

Soweit erstmal das wichtigste Fundament für die weitere Arbeit. Im Rahmen des Kurses werden noch viele weitere Befehle hinzukommen und es wird gezeigt wie ein Datensatz importiert wird und welche Pakete es dazu benötigt.

Da das Lernen einer Programmiersprache viel Zeit benötigt und oftmals sich nach dem Motto “Learning-by-Doing” vertieft, möchte ich noch folgende Kanalempfehlung mitgeben:

[R programming for beginners - R Programming 101](#)