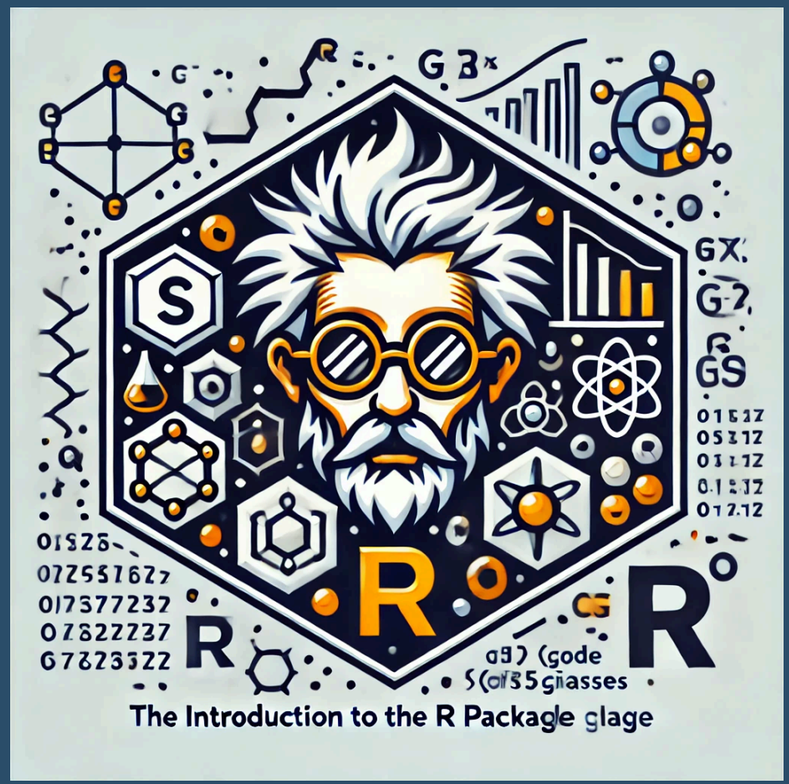
# Teil I - Erste Grundlagen

Grundlagen der Datenanalyse und Statistik mit R | WS 2024/25

Prof. Dr. Daniel Schnitzlein




GitHub-Repo: https://github.com/dschnitzlein/GDSR_1_Part_1/

# Seminarablauf

| Datum | Uhrzeit | Thema |
|---|---|---|
| **18. Oktober 2024** | **08:00 – 12:15 Uhr** | **Termin 1** |
| 22. November 2024 | 08:00 - 12:15 Uhr | Termin 2 |
| 06. Dezember 2024 | 08:00 - 12:15 Uhr | Termin 3 |
| 10. Januar 2025 | 08:00 - 12:15 Uhr | Termin 4 |
| 17. Januar 2025 | 08:00 - 10:00 Uhr (?) | Termin 5 |

Alle Termine finden **online** via **Zoom** statt. Der Link wird rechtzeitig vor dem jeweiligen Termin bereitgestellt. Nach jedem Termin wird es ein Problem Set (Hausaufgabe) geben, das bis zum nächsten Termin bearbeitet werden muss. Die Abgabe der Hausaufgaben ist Voraussetzung für die Bescheinigung der Kursteilnahme.

GitHub-Repo: https://github.com/dschnitzlein/GDSR_1_Part_1/

# (Vorläufige) Themenliste

1. **Einführung in R und R-Studio: Überblick über die Installations- und Einrichtungsprozesse | Grundlegende Funktionen und Bedienung von R und RStudio**
2. **Grundlagen der Statistiksprache R: Syntax und Datenstrukturen in R | Einführung in Funktionen und Pakete**
3. Datenmanagement in R: Methoden der Datenorganisation und -vorbereitung | Importieren, Bereinigen und Transformieren von Datensätzen
4. Einführung in die Pakete des tidyverse: Überblick über die wichtigsten tidyverse-Pakete wie z.B. dplyr und ggplot2 | Anwendung dieser Pakete zur effizienten Datenanalyse und -visualisierung
5. Deskriptive Statistik in R: Berechnung und Interpretation grundlegender statistischer Kennzahlen | Anwendung von deskriptiven Methoden zur Datenexploration | Einführung in die statistische Modellierung am Beispiel linearer Modelle
6. Datenvisualisierung in R: Erstellen von publikationsreifen Grafiken und Diagrammen mit ggplot2 | Gestaltung und Interpretation von Datenvisualisierungen zur Unterstützung der Datenanalyse

GitHub-Repo: https://github.com/dschnitzlein/GDSR_1_Part_1/

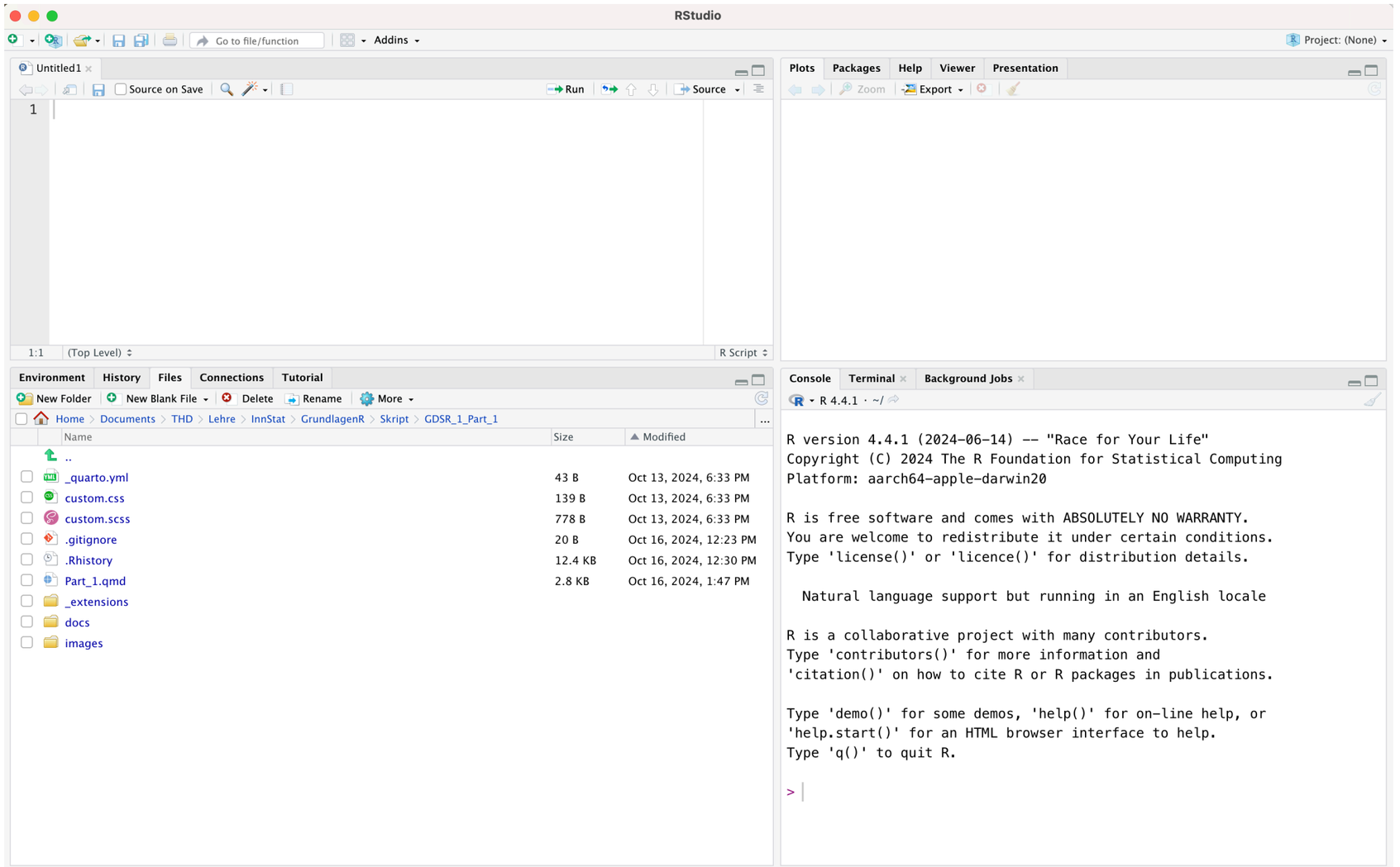
# Tagesablauf

| Uhrzeit | Titel | Thema | Raum |
|---|---|---|---|
| 08:00 - 09:30 | Block 1 | Teil I | Hauptraum |
| 09:45 - 11:15 | Block 2 | Teil I / Übung I | Break-Out Räume |
| 11:30 - 12:15 | Block 3 | Übung I | Hauptraum |

Die heutige Einführung ist eine Adaption (und Übersetzung) des Einstiegskurs zu R aus dem Lehrbuch *An Introduction to Statistical Learning - with Applications*

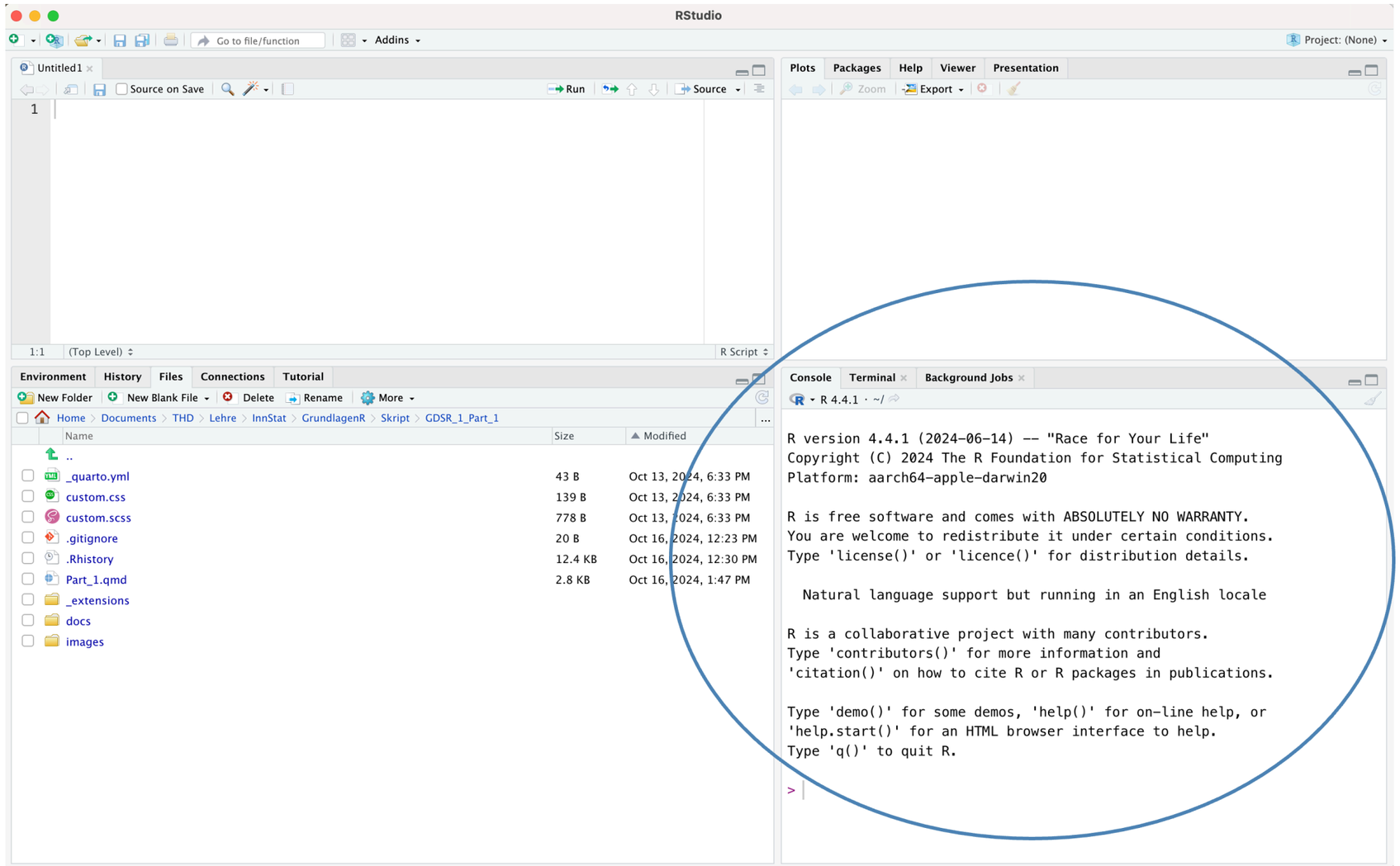
GitHub-Repo: https://github.com/dschmitzlein/GDSR_1_Part_1/

# RStudio ist die bevorzugte IDE für R



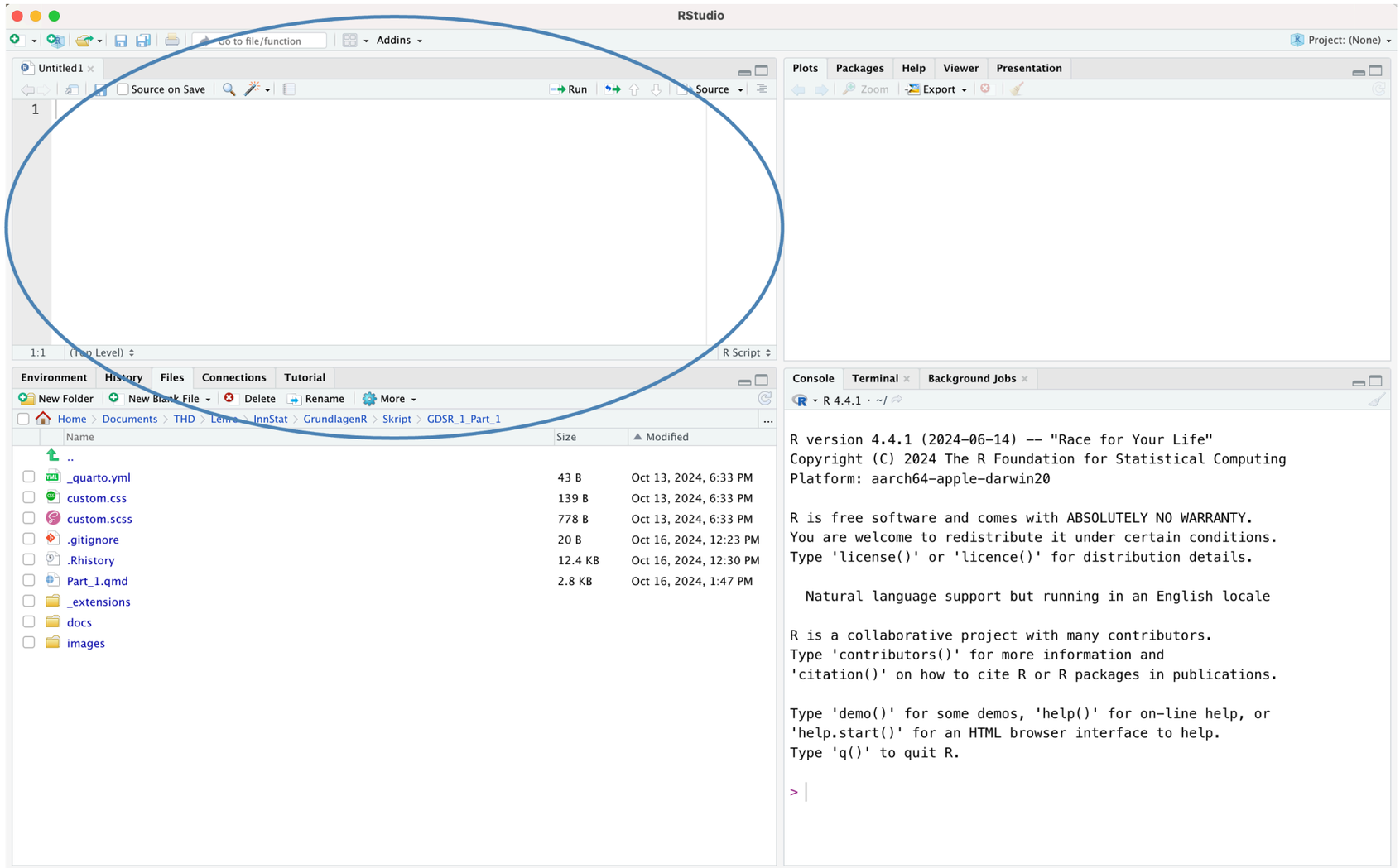

GitHub-Repo: https://github.com/dschnitzlein/GDSR_1_Part_1/

# Sie können R interaktiv in der R-Konsole nutzen




GitHub-Repo: https://github.com/dschnitzlein/GDSR_1_Part_1/

# Empfehlenswert ist es aber immer mit einem R-Skript zu arbeiten




GitHub-Repo: https://github.com/dschnitzlein/GDSR_1_Part_1/

# R als Taschenrechner und Objekte

In der einfachsten Anwendung können Sie R im interaktiven Gebrauch als Taschenrechner nutzen:

R Code | Start Over | Run Code

```
1 + 3
```

Mit dem Operator `<-` weisen Sie R an, das Ergebnis des Ausdrucks der rechts des Operators steht in dem Objekt abzulegen (und ggf. das Objekt zu erzeugen), das links vom Operator steht.

R Code | Start Over | Run Code

```
ergebnis <- 1 + 3
```

In diesem Fall wird das Ergebnis von `1 + 3` in dem Objekt `ergebnis` abgelegt. Um den Inhalt eines Projekts anzuzeigen, müssen Sie (in den allermeisten Fällen) einfach den Namen des Objekts eingeben. Ist das Objekt komplexer (Modellergebnisse etc.) kann es sein, dass Sie explizit angeben müssen, dass das Objekt angezeigt werden soll `print(Objekt)`.

# Funktionen (1)

R verwendet *Funktionen* um Operationen durchzuführen. Um eine Funktion namens `Funktionsname` auszuführen, geben wir `Funktionsname(Eingabe1, Eingabe2, ...)` ein, wobei die Eingaben (oder *Argumente*) `Eingabe1` und `Eingabe2` R mitteilen, wie die Funktion ausgeführt werden soll. Eine Funktion kann eine beliebige Anzahl von Eingaben haben.

Um zum Beispiel einen Zahlenvektor aus einzelnen Zahlen zu erzeugen, verwenden wir die Funktion `c()`, die alle Zahlen innerhalb der Klammern zu einem Vektor zusammenfügt.

Der folgende Befehl weist R an, die Zahlen 1, 3, 2 und 5 zusammenzufügen und als einen Vektor mit dem Namen `x` zu speichern. Wenn wir `x` eintippen, bekommen Sie den Vektor als Ergebnis angezeigt.

R Code | Start Over | Run Code

```r
x <- c(1, 6, 2)
x
y <- c(1, 4, 3)
y
```

GitHub-Repo: https://github.com/dschnitzlein/GDSR_1_Part_1/

# Wo bekomme ich Hilfe?

Wenn Sie mehr Informationen über eine bestimmte Funktion erfahren möchten, tippen Sie auf der Konsole `?Funktionsname` ein.

R Code | Start Over | Run Code

```
1 ?c
```

Wir können R anweisen, zwei Zahlenvektoren zu addieren. R addiert dann die erste Zahl von `x` zur ersten Zahl von `y` und so weiter. Allerdings müssen `x` und `y` die gleiche Länge haben. Wir können ihre Länge mit der Funktion `length()` überprüfen.

R Code | Start Over | Run Code

```
1 length(x)
2 length(y)
3 x + y
```

# Objekte

R arbeitet mit Objekten. Mit der Funktion `ls()` können wir eine Liste aller bisher geladenen Objekte, wie z.B. Daten und Funktionen, anzeigen. Mit der Funktion `rm()` kann man alle Objekte löschen, die nicht mehr benötigt werden. Dies kann u.a. zur Speicheroptimierung sehr hilfreich sein.

R Code ⟳ Start Over ▷ Run Code

```r
ls()
rm(x, y)
ls()
```

Sie können auch alle Objekte auf einmal löschen.

R Code ⟳ Start Over ▷ Run Code

```r
rm(list = ls())
```

# Matrizen

Die `matrix()` Funktion erstellt eine Matrix von Zahlen. Das Hilfe-File (`?matrix`) listet, dass die Funktion eine ganze Reihe von Argumenten akzeptiert. Wir fokussieren allerdings auf die ersten drei: Die Daten (Einträge in der Matrix), die Anzahl der Zeilen und die Anzahl der Spalten. Wir starten mit einer einfachen Matrix.

R Code | Start Over | Run Code

```r
x <- matrix(data = c(1, 2, 3, 4), nrow = 2, ncol = 2)
x
```

**Hinweis:** Die Zusätze `data=`, `nrow=`, und `ncol=` in der `matrix()` Funktion sind nicht unbedingt notwendig aber machen den Code lesbarer. Probieren Sie es einfach aus!

# Reihenfolge der Argumente in einer Funktion

Oft kann es allerdings hilfreich sein, die Namen der übergebenen Argumente explizit anzugeben, da R sonst davon ausgeht, dass die Funktionsargumente in der gleichen Reihenfolge an die Funktion übergeben werden, wie sie in der Hilfedatei der Funktion angegeben sind.

Im obigen Beispiel erstellt R standardmäßig Matrizen, indem es nacheinander die Spalten ausfüllt. Alternativ kann die Option `byrow = TRUE` verwendet werden, um die Matrix in der Reihenfolge der Zeilen aufzufüllen.

Probieren Sie es aus:

R Code | Start Over | Run Code

```r
matrix(data = c(1, 2, 3, 4), . . .)
```

Beachten Sie, dass wir im obigen Befehl die Matrix nicht einem Objekt wie x zugewiesen haben. In diesem Fall wird die Matrix auf dem Bildschirm ausgegeben, aber nicht für zukünftige Berechnungen gespeichert.

# Funktionen (2)

Die Funktion `rnorm()` erzeugt einen Vektor von standardnormalverteilten Zufallszahlen, wobei das erste Argument `n` der Stichprobenumfang ist. Jedes Mal wenn wir diese Funktion aufrufen, erhalten wir eine andere Antwort. Hier bilden wir zwei korrelierte Vektoren, `x` und `y`, und verwenden die Funktion `cor()`, um die Korrelation zwischen ihnen zu berechnen. Wenn Sie den Code mehrfach ausführen, sehen Sie, dass sich die Ergebnisse verändern.

R Code | Start Over | Run Code

```r
x <- rnorm(50)
y <- x + rnorm(50, mean = 50, sd = .1)
cor(x, y)
```

Standardmäßig erzeugt `rnorm()` standardnormalverteilte Zufallszahlen mit einem Mittelwert von $0$ und einer Standardabweichung von $1$. Der Mittelwert und die Standardabweichung können jedoch mit den Argumenten `mean` und `sd` geändert werden, wie oben dargestellt.

# Funktionen (3)

Meistens wollen wir, dass unser Code genau denselben Satz von Zufallszahlen (re)produziert; dazu können wir die Funktion `set.seed()` verwenden. Die Funktion `set.seed()` benötigt ein (beliebiges) ganzzahliges Argument.

R Code | Start Over | Run Code

```
1 set.seed(42)
2 x <- rnorm(50)
3 y <- x + rnorm(50, mean = 50, sd = .1)
4 cor(x, y)
```

Sie sollten `set.seed()` immer dann nutzen, wenn Reproduzierbarkeit der Ergebnisse wichtig ist (also fast immer).

# Funktionen (4)

Die Funktionen `mean()` und `var()` berechnen den Mittelwert und die Varianz eines Vektors von Zahlen. Wendet man `sqrt()` auf die Ausgabe von `var()` an, erhält man die Standardabweichung, alternativ nutzen Sie die Funktion `sd()` .

R Code | Start Over | Run Code

```r
set.seed(42)
y <- rnorm(100)
mean(y)
var(y)
sqrt(var(y))
sd(y)
```

# Funktionen (5)

Die Funktion `seq()` erstellt eine Folge (Sequenz) von Zahlen. Z.B. `seq(a, b)` erstellt einen Vektor von Ganzzahlen (Integer) zwischen `a` und `b`. Auch `seq()` hat viele mögliche Argumente: z.B. `seq(0, 1, length = 10)` erstellt eine Folge von `10` Zahlen mit gleichen Abständen zwischen `0` und `1`. (Anmerkung: `3:11` ist kurz für `seq(3, 11)` für Ganzzahlen.)

R Code | Start Over | Run Code

```r
x <- seq(1, 10)
x
x <- 1:10
x
x <- seq(-pi, pi, length = 50)
x
```

# Die `plot()` Funktion (1)

Die `plot()` Funktion ist der einfachste (und direkteste) Weg Daten in R zu plotten. Z.B. `plot(x, y)` ergibt einen Scatterplot der Daten in `x` und `y`. Bitte beachten Sie, die `plot()` Funktion ist vor allem in der Analysephase wichtig und hilfreich. Die Abbildungen sind in der Regel **nicht** hübsch im Sinne von **publikationsreif**.

Eine große Stärke von R ist jedoch auch publikationsreife Abbildungen erstellen zu können. Im Regelfall verwenden Sie hierfür aber dann das Paket `ggplot2`. Das Erstellen von solchen Abbildungen wird später im Seminar ein eigener Schwerpunktblock sein.

R Code ⟳ Start Over ▷ Run Code

```r
x <- rnorm(100)
y <- rnorm(100)
plot(x, y)
```

# Die `plot()` Funktion (2)

Die `plot()` Funktion bietet viele zusätzliche Optionen, z.B. das Argument `xlab` um die x-Achse zu beschriften. Wie immer finden Sie mehr Details via `?plot`.

R Code | Start Over | Run Code

```r
plot(x, y,
     xlab = "Dies ist die X-Achse",
     ylab = "Dies ist die Y-Achse",
     main = "Plot von X vs Y")
```

Beachten Sie, oben haben wir `Einrückungen` genutzt, um den Code lesbarer zu machen. R erkennt (fast immer) automatisch ob ein Befehl der über mehrere Zeilen geht zusammenhängt.

RStudio bietet Ihnen diese Einrückungen im R-Skript Editor automatisch an. Es gibt auch Plug-Ins um diese explizit sichtbar zu machen.

# Die `plot()` Funktion (3)

Oft möchte man die Ausgabe eines R-Plots speichern. Der Befehl, den wir dafür verwenden, hängt von dem Dateityp ab, den wir erstellen möchten. Um zum Beispiel eine pdf-Datei zu erstellen, verwenden wir die Funktion `pdf()` und um ein jpeg zu erstellen, verwenden wir die Funktion `jpeg()`.

R Code | Start Over | Run Code

```r
pdf("Figure.pdf")
plot(x, y,
     xlab = "Dies ist die X-Achse",
     ylab = "Dies ist die Y-Achse",
     main = "Plot von X vs Y")
dev.off()
```

Die Funktion `dev.off()` zeigt `R` an, dass wir mit der Erstellung des Plot fertig sind (und dieser gespeichert wird). Alternativ können wir die Abbildung einfach kopieren und in ein Word-Dokument, eine Powerpoint-Präsentation etc., einfügen. Bitte beachten Sie: Der obige Code funktioniert nur in RStudio, nicht auf den Folien, da Sie in den Folien nicht speichern können.

# Daten auswählen (1)

Oft sind wir nur an einem Teil eines Datensatzes interessiert. Nehmen wir an, unser Datensatz ist in der Matrix A gespeichert.

R Code | Start Over | Run Code

```r
A <- matrix(1:16, 4, 4)
A
```

Dann kann via

R Code | Start Over | Run Code

```r
A[2, 3]
```

das Element in der zweiten Reihe und dritten Spalte ausgewählt werden. Die erste Ziffer in der eckigen Klammer `[ ]` bezieht sich immer auf die Zeile und die zweite immer auf die Spalte. Wir können auch mehrere Zellen auswählen, indem wir den Sequenz-Operator von oben einsetzten.

# Daten auswählen (2)

R Code | Start Over | Run Code

```r
A[c(1, 3), c(2, 4)]
A[1:3, 2:4]
A[1:2, ]
A[, 1:2]
```

Die letzten beiden Beispiele enthalten entweder keinen Index für die Spalten oder keinen Index für die Zeilen. Das bedeutet, dass R alle Spalten oder alle Zeilen auswählen soll. R behandelt eine einzelne Spalte oder Zeile als Vektor.

R Code | Start Over | Run Code

```r
A[1, ]
```

# Daten auswählen (3)

Ein – Zeichen im Index weist R an, alle Zeilen oder Spalten auszuwählen, außer den angegeben.

R Code | Start Over | Run Code

```r
A[-c(1, 3), ]
A[-c(1, 3), -c(1, 3, 4)]
```

Über die `dim()` Funktion erfahren wir die Anzahl der Zeilen gefolgt von der Anzahl der Spalten einer Matrix.

R Code | Start Over | Run Code

```r
dim(A)
```

# Daten auswählen (4)

**Bitte beachten Sie:** Wenn Sie umfangreichere Datenoperationen durchführen wollen oder komplexere Variablen generieren wollen, ist es sinnvoll auf die Funktionalität der `tidyverse` (`https://www.tidyverse.org`) Pakete zurückzugreifen.

Wir werden hier im Kurs dazu einen Schwerpunkttermin machen.

# Daten von externen Quellen einladen (1)

Für die meisten Analysen ist der erste Schritt der Import eines neuen (externen) Datensatzes in R. Die Funktion `read.table()` ist eine wichtige (wenn auch nicht die einzige) Möglichkeit, dies zu tun. Wie immer finden Sie mehr Informationen unter `?read.table`. Je nach Format des Datensatzes können/müssen Sie auch die Funktionen `read.csv`, `read.excel` oder `heaven` nutzen.

Bevor wir versuchen, einen Datensatz zu laden, müssen wir sicherstellen, dass R nach den Daten im richtigen Verzeichnis suchen kann (siehe Folie Arbeitsverzeichnis). Die Einzelheiten der Vorgehensweise hängen jedoch vom verwendeten Betriebssystem (z. B. Windows, Mac, Linux) ab.

In der Praxis ist es am einfachsten, wenn Sie den Datensatz beim ersten Mal via des Menüs in `RStudio` einlesen und sich den dabei generierten Code anzeigen lassen.

# Daten von externen Quellen einladen (2)

Wir beginnen mit dem Laden des Datensatzes `Auto.data`. Um die Funktion `read.table()` zu veranschaulichen, laden wir diesen nun aus einer Textdatei, `Auto.data`, die Sie auf StudOn/Github finden.

Der folgende Befehl lädt die Datei `Auto.data` in `R` und speichert sie als Objekt namens `Auto` in einem Format, das als `DataFrame` bezeichnet wird. Sobald die Daten geladen sind, kann die Funktion `View()` verwendet werden, um sie in einem Excel-ähnlichen Fenster (Variable Viewer) in RStudio anzuzeigen. Alternativ können wir mit `head()` die ersten Zeilen anzeigen lassen.

R Code | Start Over | Run Code

```r
Auto <- read.table("data/Auto.data")
head(Auto)
```

# Daten von externen Quellen einladen (3)

Beachten Sie, dass `Auto.data` einfach eine Textdatei ist, die Sie alternativ auf Ihrem Computer mit einem Standard-Texteditor öffnen können. Es ist oft eine gute Idee, einen Datensatz mit einem Texteditor oder einer anderen Software wie Excel anzusehen, bevor man ihn in R lädt.

Dieser spezielle Datensatz wurde z.B. nicht korrekt geladen, weil R angenommen hat, dass die Variablennamen Teil der Daten sind und sie daher in die erste Zeile aufgenommen hat. Der Datensatz enthält auch eine Reihe fehlender Beobachtungen, die durch ein Fragezeichen ? gekennzeichnet sind.

# Daten von externen Quellen einladen (4)

Fehlende Werte sind in realen Datensätzen häufig anzutreffen. Die Verwendung der Option `header = T` (oder `header = TRUE`) in der Funktion `read.table()` sagt R, dass die erste Zeile der Datei die Variablennamen enthält, und die Verwendung der Option `na.strings` sagt R, dass jedes Mal, wenn es ein bestimmtes Zeichen (z.B. ein Fragezeichen) sieht, dieses Element als fehlendes Element in der Datenmatrix behandelt werden soll.

R Code | Start Over | ▷ Run Code

```r
Auto <- read.table("data/Auto.data", header = T, na.strings = "?", stringsAsFactors = T)
head(Auto)
```

# Daten von externen Quellen einladen (5)

Das Argument `stringsAsFactors = T` teilt `R` mit, dass jede Variable, die (nicht-numerische) Zeichen enthält, als qualitative Variable (Faktorvariable) interpretiert werden soll und dass jede einzelne Zeichenkombination eine Einheit (Level) dieser qualitative Variable darstellt.

Mit der `dim()` Funktion können wir ermitteln, dass der Datensatz $397$ Beobachtungen (Zeilen) und 9 Variablen (Spalten) hat.

R Code | Start Over | Run Code

```
1 dim(Auto)
```

# Das Arbeitsverzeichnis

- R nutzt das Konzept des Arbeitsverzeichnisses. Das aktuelle Arbeitsverzeichnis kann mit dem Befehl `getwd()` angezeigt werden. Standardmäßig geht R davon aus, dass alle zu ladenden Datensätze, Speicherorte für Abbildungen und Speicherorte für Daten sich, soweit nicht anders angegeben, auf das Arbeitsverzeichnis beziehen.
- Natürlich können die Datensätze auch in anderen Verzeichnissen (oder auf einem Server) liegen. In diesem Fall muss aber der Pfad zur jeweiligen Datei mit angegeben werden.
- Das Arbeitsverzeichnis kann entweder grafisch über RStudio gesetzt werden oder mit dem Befehl `setwd()` im R-Skript.
- Eine effektive Projektorganisation in R organisiert alle Dateien (Skripte, Daten, Abbildungen) in einem gemeinsamen Verzeichnis mit spezifischen Unterverzeichnissen (Daten, Abbildungen etc.) und nutzt (nach der Definition des Arbeitsverzeichnis) in allen Skripten relative Pfade.
- Auf diese Weise kann ein Projektordner sehr einfach transferiert werden, ohne den Code anpassen zu müssen.

# Erste Aufbereitung: Fehlende Werte

Es gibt verschiedene Wege mit fehlenden Werten umzugehen. In unserem Beispiel enthalten 5 Beobachtungen fehlende Werte. Daher nutzen wir die `na.omit()` Funktion und löschen diese Beobachtungen. Wir werden uns später im mit alternativen Wegen beschäftigen.

R Code | Start Over | Run Code

```r
Auto <- na.omit(Auto)
dim(Auto)
```

Wir können nun `names()` nutzen um die Variablennamen (Spaltennamen) anzuzeigen.

R Code | Start Over | Run Code

```r
names(Auto)
```

# Grafische Zusammenfassung (1)

Wie oben können wir die Funktion `plot()` verwenden, um Streudiagramme für die quantitativen Variablen zu erstellen. Die einfache Eingabe der Variablennamen führt jedoch zu einer Fehlermeldung, da R nicht weiß, wo diese Variablen gesucht werden sollen.

R Code | Start Over | Run Code

```
1 plot(cylinders, mpg)
```

GitHub-Repo: https://github.com/dschnitzlein/GDSR_1_Part_1/

# Grafische Zusammenfassung (2)

Um auf eine Variable zu verweisen, müssen wir den Datensatz und den Variablennamen zusammen mit einem `$`-Symbol eingeben. Alternativ könnten wir die Funktion `attach()` verwenden, um `R` anzuweisen, die Variablen in diesem `DataFrame` nur über ihren Namen verfügbar zu machen. Das ist aber, speziell wenn man mit mehreren `DataFrame` gleichzeitig arbeitet, sehr fehleranfällig und sollte vermieden werden.

R Code | Start Over | ▷ Run Code

```
1 plot(Auto$cylinders, Auto$mpg)
```

# Grafische Zusammenfassung (3)

Die Variable `cylinders` ist als numerischer Vektor definiert, daher hat `R` sie auch so verwendet. Allerdings ist es hier inhaltlich sinnvoll `cylinders` als qualitative Variable zu verwenden (Faktor). Die Funktion `as.factor()` konvertiert quantitative Variablen in qualitative Variablen.

Wenn die angegebene Variable für die x-Achse qualitativ ist, dann gibt `plot()` automatisch Boxplots statt Streudiagramme aus. Wie immer gibt es unter `?plot()` viele Optionen um die Abbildungen anzupassen.

R Code | Start Over | Run Code

```r
Auto$cylinders <- as.factor(Auto$cylinders)
plot(Auto$cylinders, Auto$mpg, col = "red", xlab = "cylinders", ylab = "MPG")
```

# Numerische Zusammenfassung der Daten

Mit der `summary()` Funktion können wir deskriptive Statistiken eines gesamten Datensatzes anzeigen lassen ...

R Code | Start Over | Run Code

```
1 summary(Auto)
```

... oder nur einer einzelnen Variablen.

R Code | Start Over | Run Code

```
1 summary(Auto$mpg)
```