Einführung in R und RStudio

Kausalanalyse: Economics meets Data Science

Marc Justin Schmidt

15. Oktober 2026

Willkommen zur Übung der Kausalanalyse!

Die Übung zur Kausalanyse startet mit...

- ... der Installation und Einrichtung von R und RStudio (falls noch nicht geschehen)
- ... der Einordnung von R in die Welt der Statistik-Software
- ... einer Einführung in die IDE (Entwicklungsumgebung)
- ... den wichtigsten R-Grundlagen und ersten Datenanalysen

Ein Gefühl für R bekommen!

Installation

1. R installieren

cran.r-project.org

2. IDE installieren

RStudio (Bewährt) | Positron (Modern) | Visual Studio Code

- rstudio.com
- Von Posit (ehemals RStudio genannt)
- Weit verbreitet, viele Tutorials

Installation

1. R installieren

cran.r-project.org

2. IDE installieren

RStudio (Bewährt) | **Positron (Modern)** | Visual Studio Code

- github.com/posit-dev/positron
- Neueste IDE von Posit
- Bessere Performance

Installation

1. R installieren

cran.r-project.org

2. IDE installieren

RStudio (Bewährt) | Positron (Modern) | **Visual Studio Code**

- code.visualstudio.com
- Von Microsoft
- Viele Erweiterungen für R
- Für viele verschiedene Programmiersprachen

Was ist R?

> R ist eine kostenlose, nicht kommerzielle Open-Source-Software für statistische Berechnungen und Grafiken. Sie hat viele Vorteile, vor allem bei Datenanalyseprojekten.

- `R` basiert auf der Programmiersprache `S`, die wiederum auf der Kernsprache `C` aufbaut
- Erste Version veröffentlicht 1993 von **R**oss Ihaka und **R**obert Gentleman (der Name leitet sich von ihren Anfangsbuchstaben ab)
- Ziel von `R`: eine Umgebung schaffen, in der man nicht bewusst über Programmierung nachdenken muss

Warum R (für Kausalanalyse)?

Vorteile von R

- **Open Source** und kostenlos
- **Riesige Community** und Paket-Ökosystem (über 20.000 Pakete)
- **Reproduzierbare** Analysen und Forschung
- **Visualisierungsmöglichkeiten** sind exzellent
- **Plattformunabhängig** (läuft sogar auf der PlayStation)
- **Breite Anwendbarkeit** (Datenanalysen, Web-Apps, Dashboards, Berichte, Präsentationen, uvm.) [1]

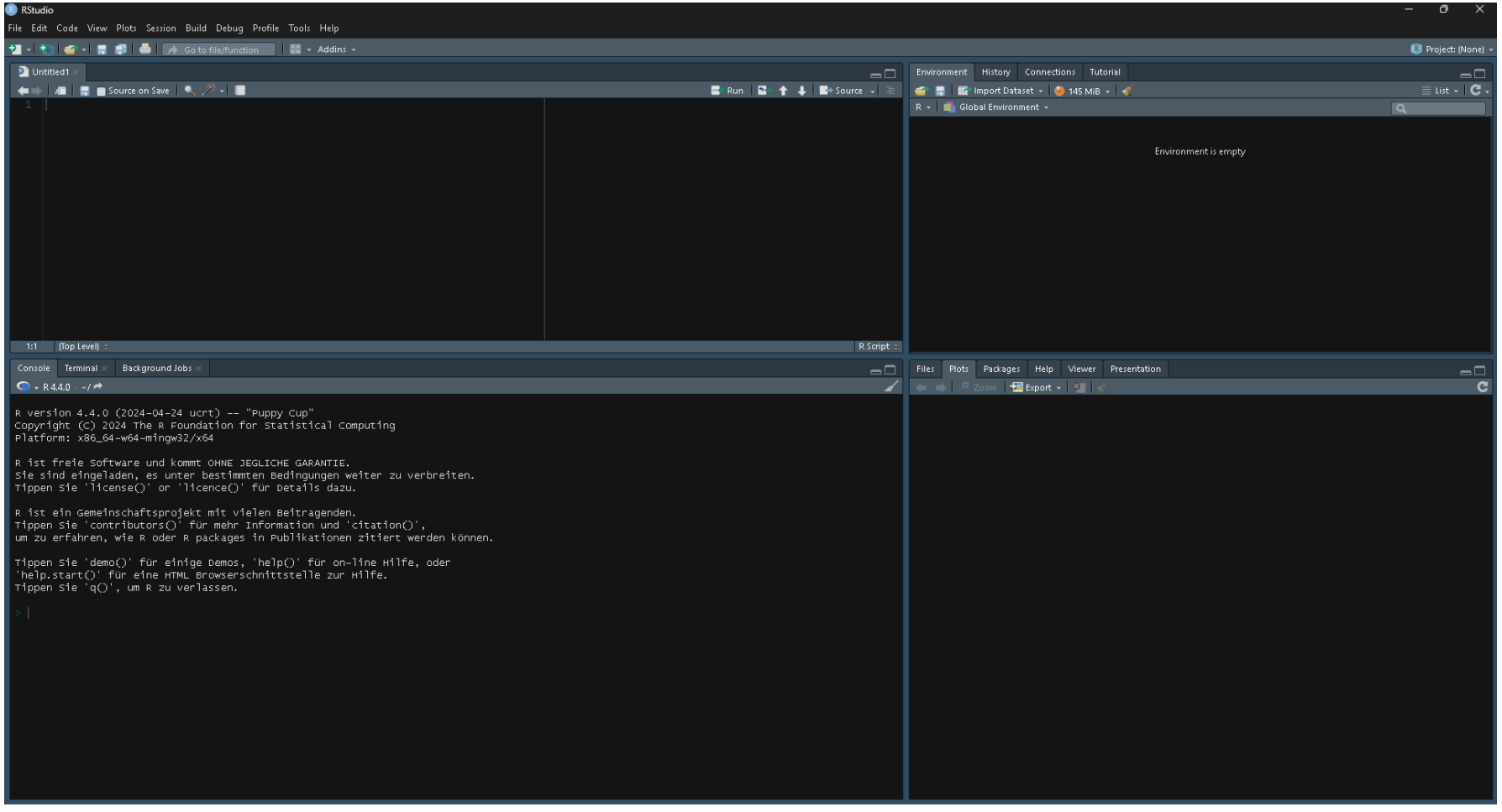
Populäre Alternativen

- `Python` (Allzweck, aber weniger Statistik-orientiert)
- `Stata` (Proprietär, aber benutzerfreundlich)
- `SPSS` (Proprietär, GUI-basiert, weniger flexibel)
- ...

Der Wechsel von einer Programmiersprache zu einer anderen ist leichter als der allgemeine Einstieg in Programmierung!

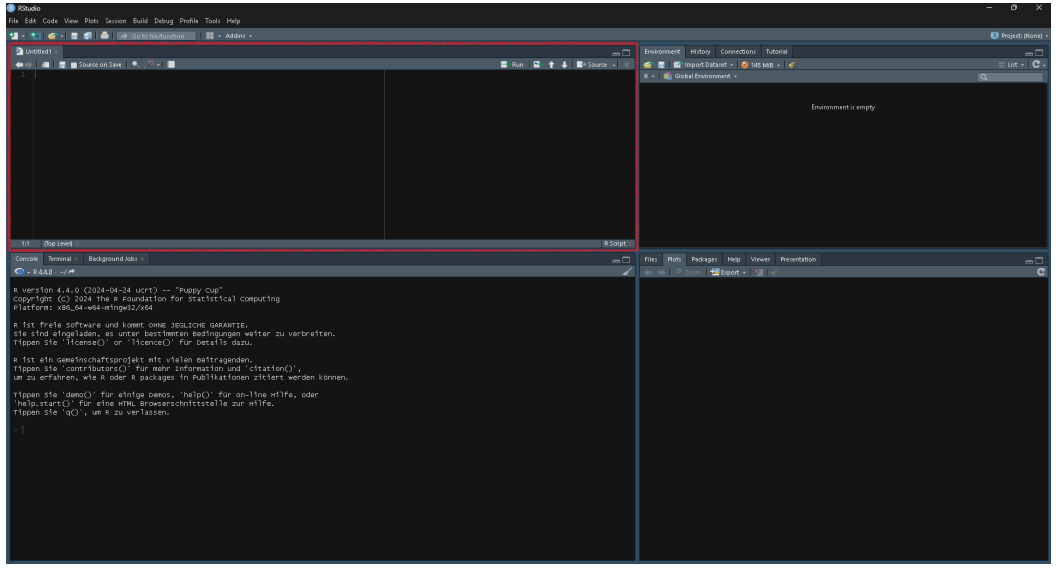
[1] Die Präsentationsfolien, die Aufgabenblätter und die Webseite zur Übung sind auch mit `R` geschrieben!

RStudio Interface



Die vier Panels

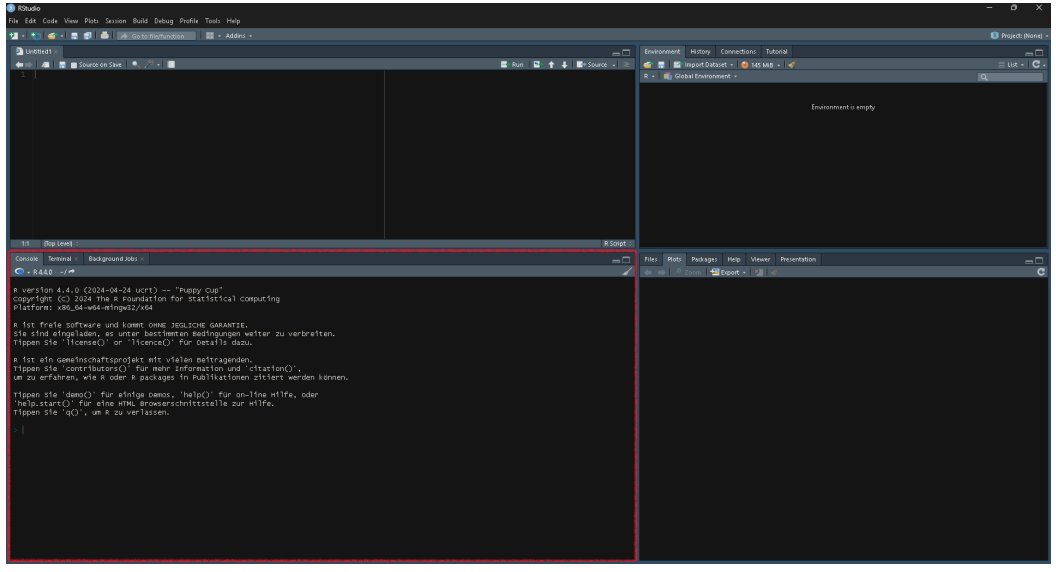
"Code Editor" "Execution Panel" "Knowledge Hub" "Showroom"



- Code-Editor für `.R`, `.Rmd`, `...` Dateien
  - Syntax-Highlighting
  - Skripte erstellen

Die vier Panels

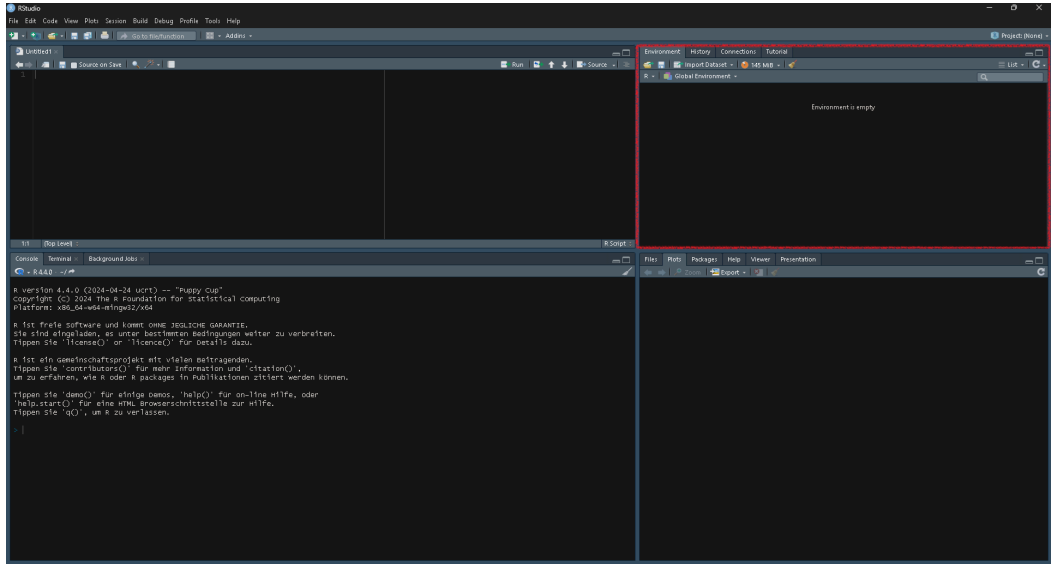
"Code Editor" | "Execution Panel" | "Knowledge Hub" | "Showroom"



- Interaktive R-Kommandozeile
  - Direktes Ausführen von Befehlen
  - Zugriff auf Terminal (=Bash, PowerShell, …)

Die vier Panels

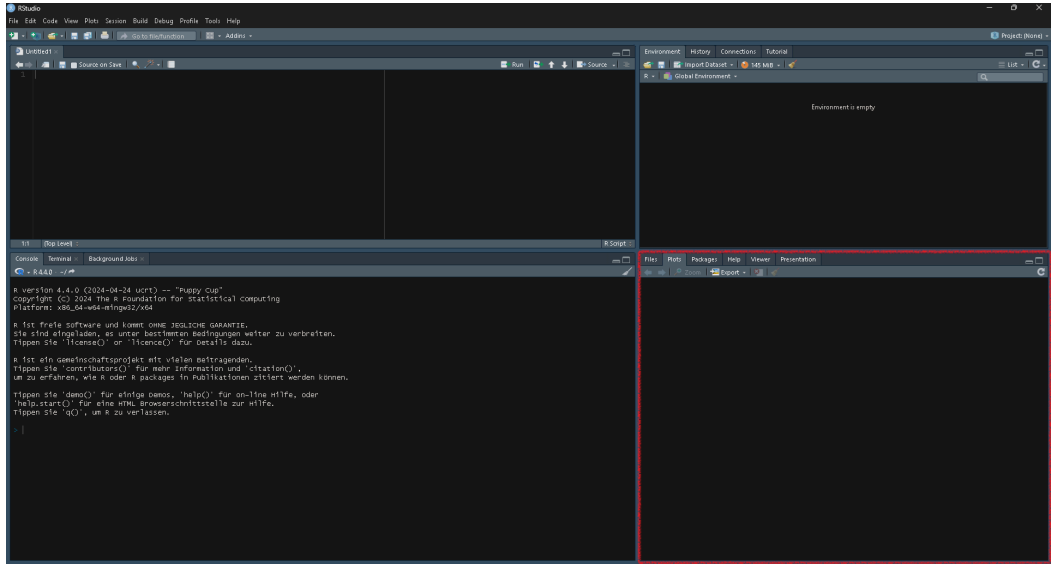
"Code Editor" "Execution Panel" **"Knowledge Hub"** "Showroom"



- Überblick über Variablen und Objekte
  - History der Befehle
  - Ggf. Git-Integration

Die vier Panels

"Code Editor" "Execution Panel" "Knowledge Hub" **"Showroom"**



- Dateien verwalten
  - Plots anzeigen
  - Pakete installieren

Live-Demo

IDE und grundlegende Funktionalitäten



Wichtige Shortcuts

Shortcut	Funktion
`Strg + Enter`	Code ausführen
`Strg + S`	Speichern
`Strg + Shift + M`	Pipe `%>%` (Tidyverse)
`Tab`	Auto-Vervollständigung (Copilot)
`Strg + Shift + C`	Kommentieren
`Strg + A`	Alles auswählen
`Strg + C`	Kopieren
`Strg + V`	Einfügen

Shortcuts lernen = 10x schneller arbeiten!

Best Practices von Anfang an

> Good coding style is like using correct punctuation. You can manage without it, but it sure makes things easier to read. [...] you really should use a consistent style. Good style is important because while your code only has one author, it'll usually have multiple readers. This is especially true when you're writing code with others. (Wickham, 2019)

Code-Stil

- **Aussagekräftige Namen** für Variablen und Datensätze: `alter` nicht `x`
- **Konsistente Einrückung**
- **Kommentare** und entsprechende Dokumentation

Projektorganisation (fortgeschritten)

- **Klare Ordnerstruktur**
- **Funktionen** für wiederholte Aufgaben schreiben
- **Relative Pfade** verwenden
- **README.md** schreiben
- **Versionskontrolle** (Git)

Base R

Kann sowohl im Skript-Editor als auch in der Konsole genutzt werden. Punkt-vor-Strich-Rechnung wird beachtet.

```
2 + 3
```

```
## [1] 5
```

```
6 * 7
```

```
## [1] 42
```

```
2^3
```

```
## [1] 8
```

Base R

Der "<-" Operator ist der R-Standard für Zuweisungen (alternativ "=")

```
# Variablen erstellen
x <- 5
y <- 10
z <- x + y # Addition
# Ausgabe
print(z)
```

```
## [1] 15
```

```
# Ausgabe
z
```

```
## [1] 15
```

Base R


```
# Vergleich zweier Objekte (!=, <=, >=, ==)
3 < 1
```

```
## [1] FALSE
```

```
# Logisches ODER "|"
(1 > 2) | (1 > 0)
```

```
## [1] TRUE
```

```
# LOGISCHES UND "&"
(8 > 4) & (8 > 7.99)
```

```
## [1] TRUE
```

Base R


```
zahlen <- c(2, 4, 6, 8, 10)
zahlen[2:3] # Indexing
```

```
## [1] 4 6
```

```
zahlen[c(1, 2)] # Indexing
```

```
## [1] 2 4
```

```
zahlen * 2 # Elementweise Operation (Multiplikation)
```

```
## [1]  4  8 12 16 20
```

Base R


```
mat <- matrix(1:9, nrow = 3, byrow = TRUE) # Im Grunde ein Datensatz!
mat[1, 2] # Indexing
```

```
## [1] 2
```

```
mat[, 3] # Ganze dritte Spalte
```

```
## [1] 3 6 9
```

```
sqrt(mat) # Elementweise Operation (Wurzel)
```

```
##          [,1]     [,2]     [,3]
## [1,] 1.000000 1.414214 1.732051
## [2,] 2.000000 2.236068 2.449490
## [3,] 2.645751 2.828427 3.000000
```

Base R

Funktionen, die in base R eingebaut sind (ohne weitere Installation ausführbar).

```
zahlen <- c(1, 2, 3, 4, 5)
mean(zahlen) # Mittelwert
```

```
## [1] 3
```

```
round(3.14159) # Runden
```

```
## [1] 3
```

```
round(3.14159, digits = 2) # Auf zweite Nachkommastelle runden
```

```
## [1] 3.14
```

Base R


```
class(2) # Numerisch
```

```
## [1] "numeric"
```

```
class("Hallo Welt") # String
```

```
## [1] "character"
```

```
wahrheit <- TRUE; class(wahrheit) # Logisch
```

```
## [1] "logical"
```

```
kategorie <- factor("A"); class(kategorie) # Faktor (Kategorie, z.B. Geschlecht)
```

```
## [1] "factor"
```

Base R

Datentypen beschreiben die Art der Werte (z.B. Zahlen oder Text). Datenstrukturen beschreiben, wie mehrere Werte organisiert sind.

Datenstruktur	Aufbau	Datentypen
Vektor	Eindimensional	Ein gemeinsamer Datentyp
Matrix	Zweidimensional: Zeilen und Spalten	Ein gemeinsamer Datentyp
Array	Auch mehr als zwei Dimensionen möglich	Ein gemeinsamer Datentyp
Liste	Sammlung beliebiger Objekte	Unterschiedliche Datentypen möglich
Dataframe	Tabelle aus gleich langen Spalten	Jede Spalte kann einen anderen Datentyp haben

Base R


```
df <- data.frame(
  name=c("Anna", "Bob", "Clara"),
  alter=c(25, 30, 28)
)
df
```

```
##    name alter
## 1  Anna    25
## 2   Bob    30
## 3 Clara    28
```

```
head(mtcars, 2)
```

```
##                mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4       21   6  160 110  3.9 2.620 16.46  0  1    4    4
## Mazda RX4 Wag   21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

Base R


```
x <- 5
if (x > 0) "positiv" else "nicht positiv"
```

```
## [1] "positiv"
```

```
for (i in 1:3) print(i^2)
```

```
## [1] 1
## [1] 4
## [1] 9
```

```
quadriere <- function(x) x^2
quadriere(4)
```

```
## [1] 16
```

R Extensions

Pakete sind Sammlungen von Funktionen und Daten, die von der R-Community bereitgestellt werden. Diese erweitern die Basisfunktionalität von R.

```
# Pakete installieren (einmalig)
install.packages("tidyverse") # Das vmtl. wichtigste Paket

# Pakete laden (jede Session erneut!)
library(tidyverse)

# Paket-Funktionen ohne Laden nutzen mit "::"
dplyr::filter(mtcars, cyl == 4)
```

Wir werden die Funktionalität von `tidyverse` öfters nutzen!

R Extensions

Mithilfe von `help()`, `?`, `example()` und `??` kann man sich Informationen zu Funktionen und Paketen anzeigen lassen.

```
# Hilfe zu Funktionen
help(mean)
?mean

# Beispiele anzeigen
example(mean)

# Paket-Dokumentation
help(package = "ggplot2")

# Fuzzy Search (Unscharfe Suche)
??regression
```

R Extensions

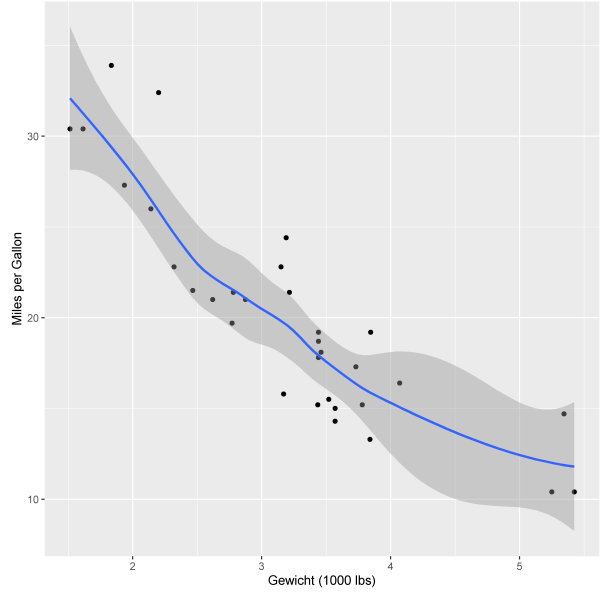

```
library(tidyverse)

# Moderne R-Syntax mit Pipes (Datensatz ist in RStudio vorinstalliert)
mtcars %>%
  filter(cyl == 4) %>%
  select(mpg, hp, wt) %>%
  head(4)
```

```
##                mpg hp   wt
## Datsun 710    22.8 93 2.32
## Merc 240D     24.4 62 3.19
## Merc 230      22.8 95 3.15
## Fiat 128      32.4 66 2.20
```

R Extensions


```
ggplot(mtcars, aes(x = wt, y = mpg)) + # Tidyverse Funktion
  geom_point() + # Punkte zeichnen
  geom_smooth() + # Trendlinie zeichnen
  labs(x = "Gewicht (1000 lbs)", y = "Miles per Gallon") # Achsenbeschriftung
```



Zeit für die Übungen!

Datei: 01_Aufgabenblatt_Einfuehrung.pdf



Vielen Dank!

Fragen?

 marcjustin.schmidt@tu-dortmund.de

Weiter zu Tutorial 2: Statistische Grundlagen

Zurück zur Hauptseite