

14.11.23

Einführung in Data Science und maschinelles Lernen

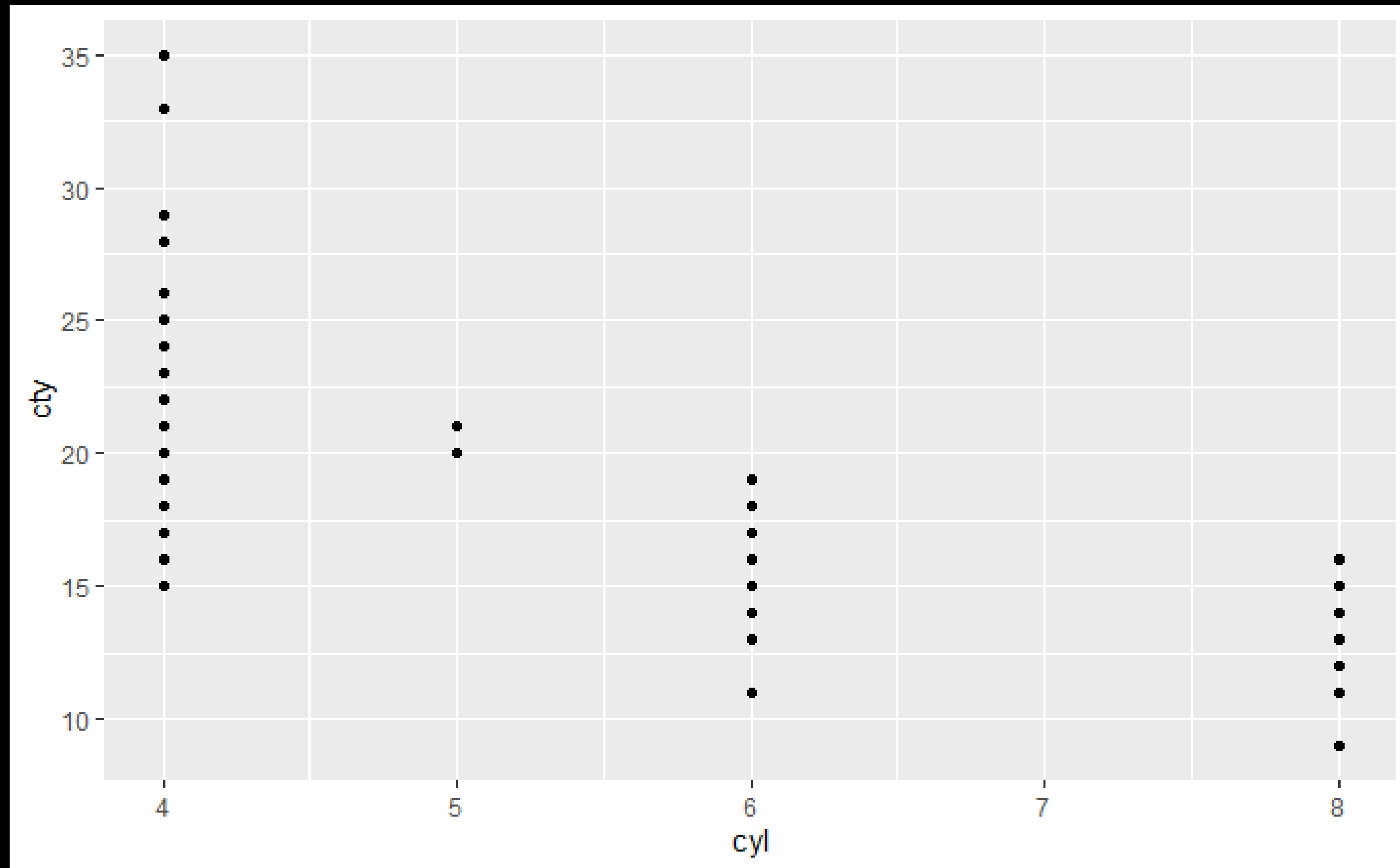
VERSIONIERUNG MIT GIT (TEIL 1) UND DATENAUFBEREITUNG MIT TIDYVERSE

- **Zusammenstellung der Teams**
- **Kennenlernen und Besprechung Übungsaufgaben**
- **Weitere Beispiele zu Diagrammen**
- **Statistische Signifikanz**
- **Einführung in die Versionierung mit git**
- **Einführung in Tidyverse und die Datenaufbereitung**

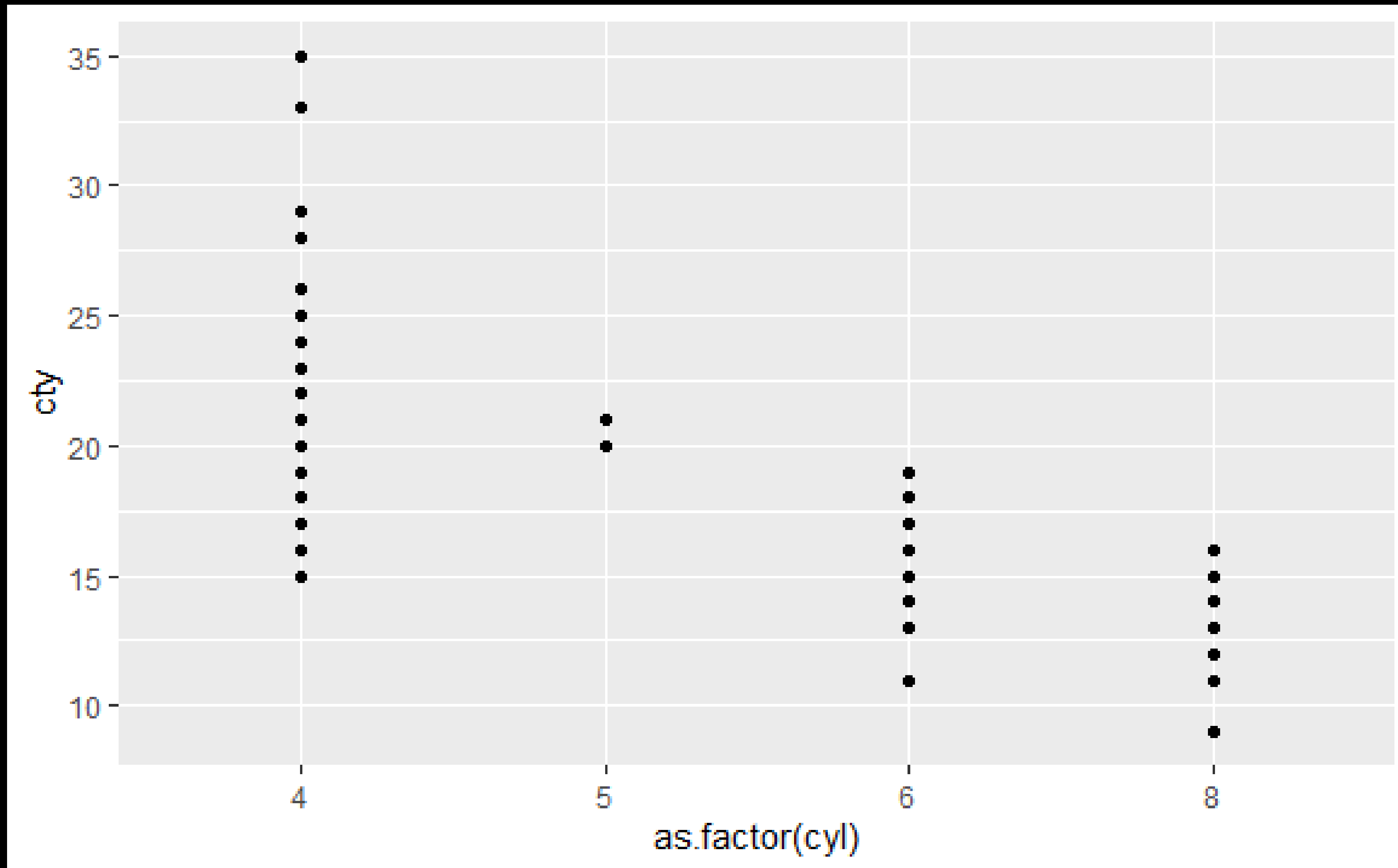
ZUSAMMENSTELLUNG DER TEAMS

BREAKOUT

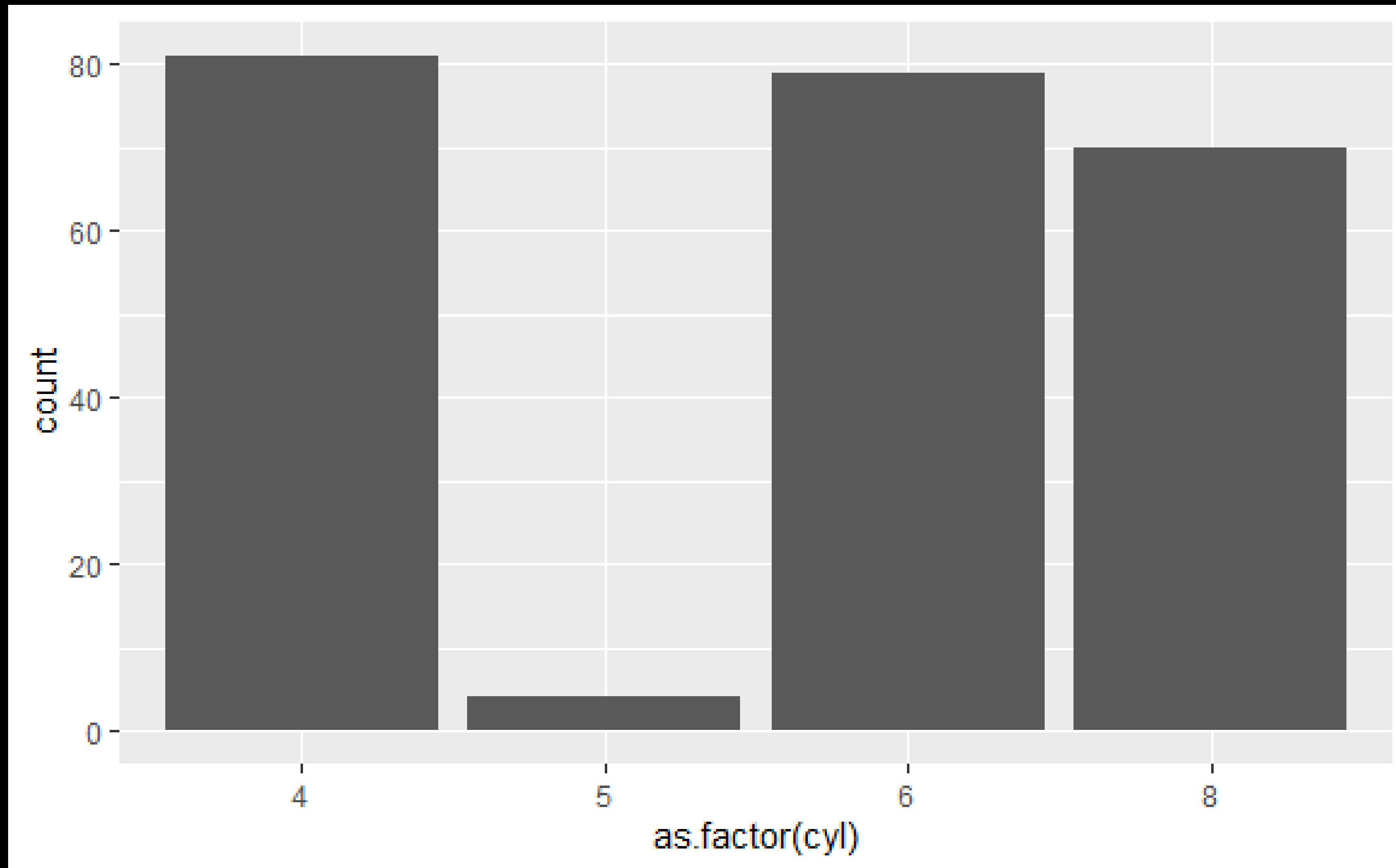
- **Vorstellungsrunde**
- **Anlegen eines Team-Channels im Chat**
- **Besprechung der Übungsaufgaben**



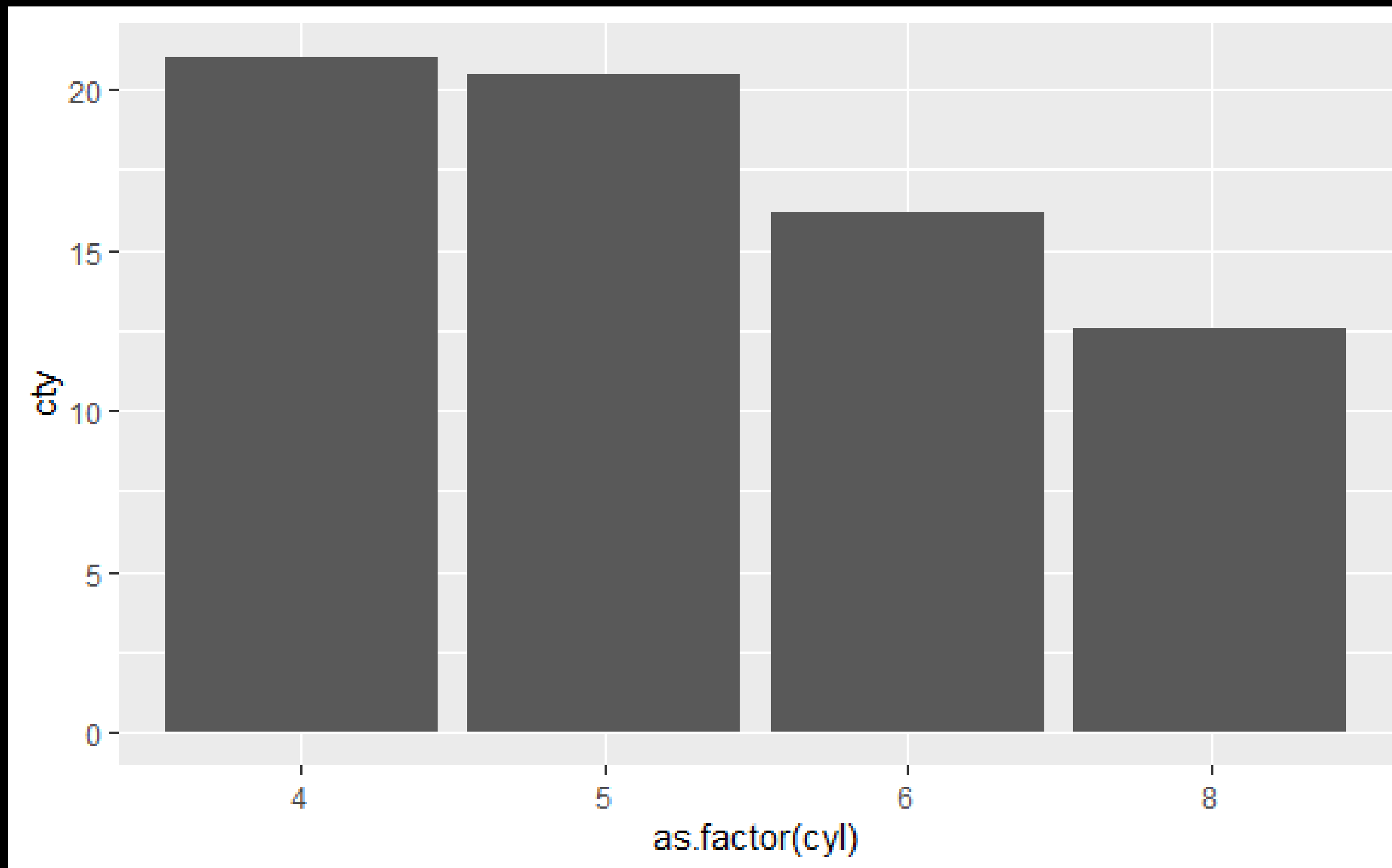
```
ggplot(mpg)+  
  geom_point(aes(x = cyl, y = cty))
```



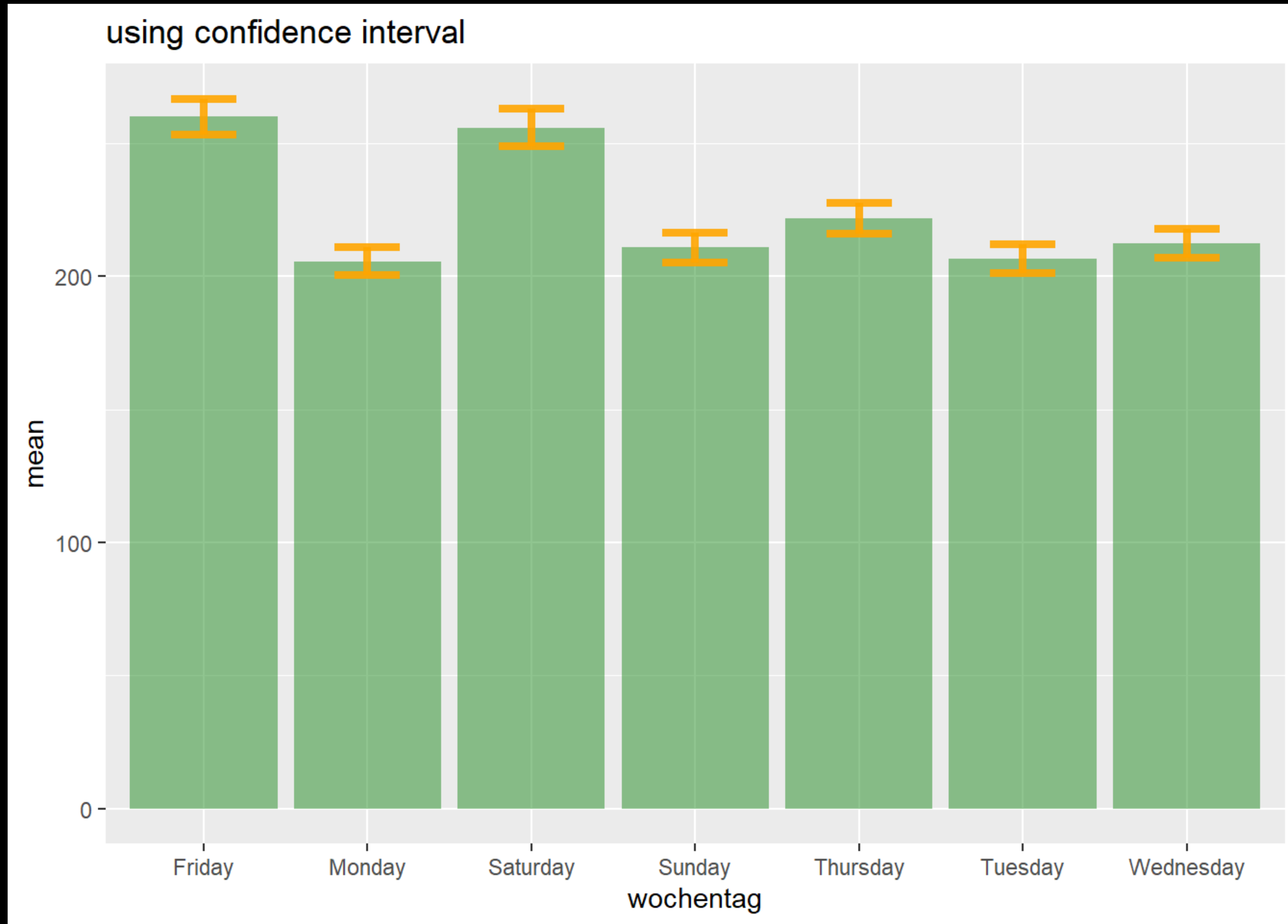
```
ggplot(mpg)+  
  geom_point(aes(x = as.factor(cyl), y = cty))
```



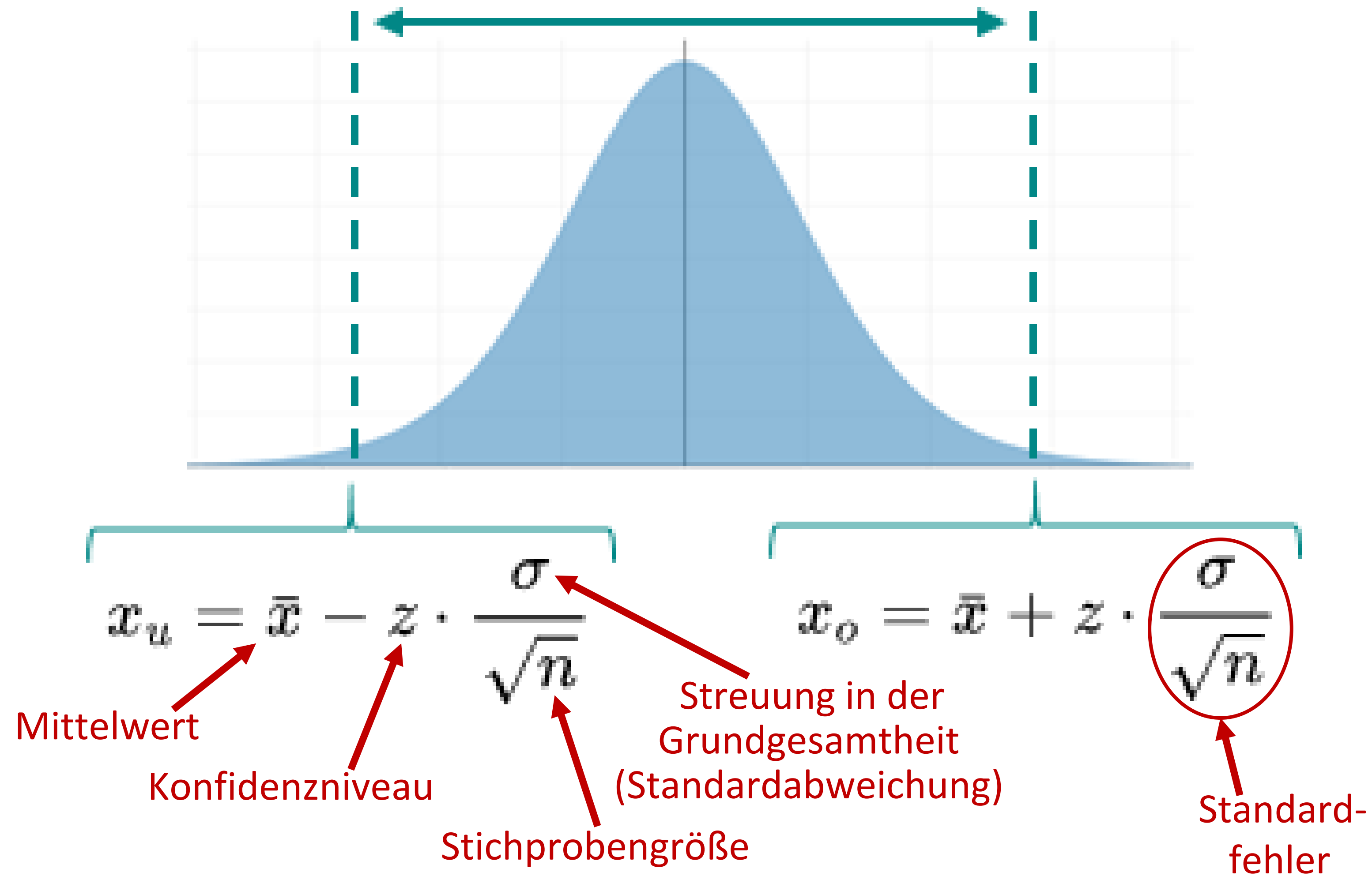
```
ggplot(mpg)+  
  geom_bar(aes(x = as.factor(cyl)))
```



```
ggplot(mpg)+  
  geom_bar(aes(x = as.factor(cyl), y = cty),  
    stat="summary", fun="mean")
```

Konfidenzintervall



STANDARDABWEICHUNG VS. STANDARDFEHLER

Standardabweichung:

- Information über die Streuung der Beobachtungen zu einer Gruppe

Standardfehler:

- Information über den Fehler des Mittelwerts der Beobachtungen zu einer Gruppe, *wenn man diese als repräsentative Stichprobe für eine größere Grundgesamtheit betrachtet.*

SIGNIFIKANZ-TESTS (1)

- **Wie weiß man, ob ein Mittelwert „statistisch signifikant“ oberhalb eines definierten Werts liegt?**
 - **Man weiß mit (in der Regel) 95%er Sicherheit , dass der Mittelwert größer als x ist.**
 - **Das 95%-Konfidenzintervall liegt unterhalb von x .**

SIGNIFIKANZ-TESTS (2)

- **Wie weiß man, dass sich zwei Mittelwerte „statistisch signifikant“ voneinander unterscheiden?**
 - **Man weiß mit (in der Regel) 95%er Sicherheit, dass die Mittelwerte sich unterscheiden.**
 - **Die 95%-Konfidenzintervalle beider Mittelwerte überschneiden sich nicht.**
 - **Die Differenz der Mittelwerte ist mit 95%er Sicherheit von 0 verschieden.**

T-TEST

```
> t.test(july$Temp, may$Temp)
```

```
Welch Two Sample t-test
```

```
data: july$Temp and may$Temp
```

```
t = 12.616, df = 50.552, p-value < 2.2e-16
```

```
alternative hypothesis: true difference in means is not equal to 0
```

```
95 percent confidence interval:
```

```
15.43351 21.27617
```

```
sample estimates:
```

```
mean of x mean of y
```

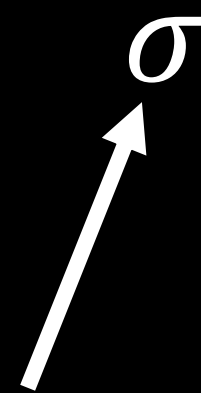
```
83.90323 65.54839
```

SIGNIFIKANZ VERSUS RELEVANZ

- **Eigenschaften können sich statistisch signifikant unterscheiden, aber der Unterschied kann ohne praktische Relevanz (ohne „Effekt“) sein.**

Betrachtung der Effektstärke nach Cohen:
(Größe des Mittelwertsunterschieds im Vergleich zur Standardabweichung)

$$d = \frac{\mu_1 - \mu_2}{\sigma}$$



vereinfachter Fall gleicher
Stichprobengröße und
Standardabweichung

$$|d| > 0,2$$

kleiner Effekt

$$|d| > 0,5$$

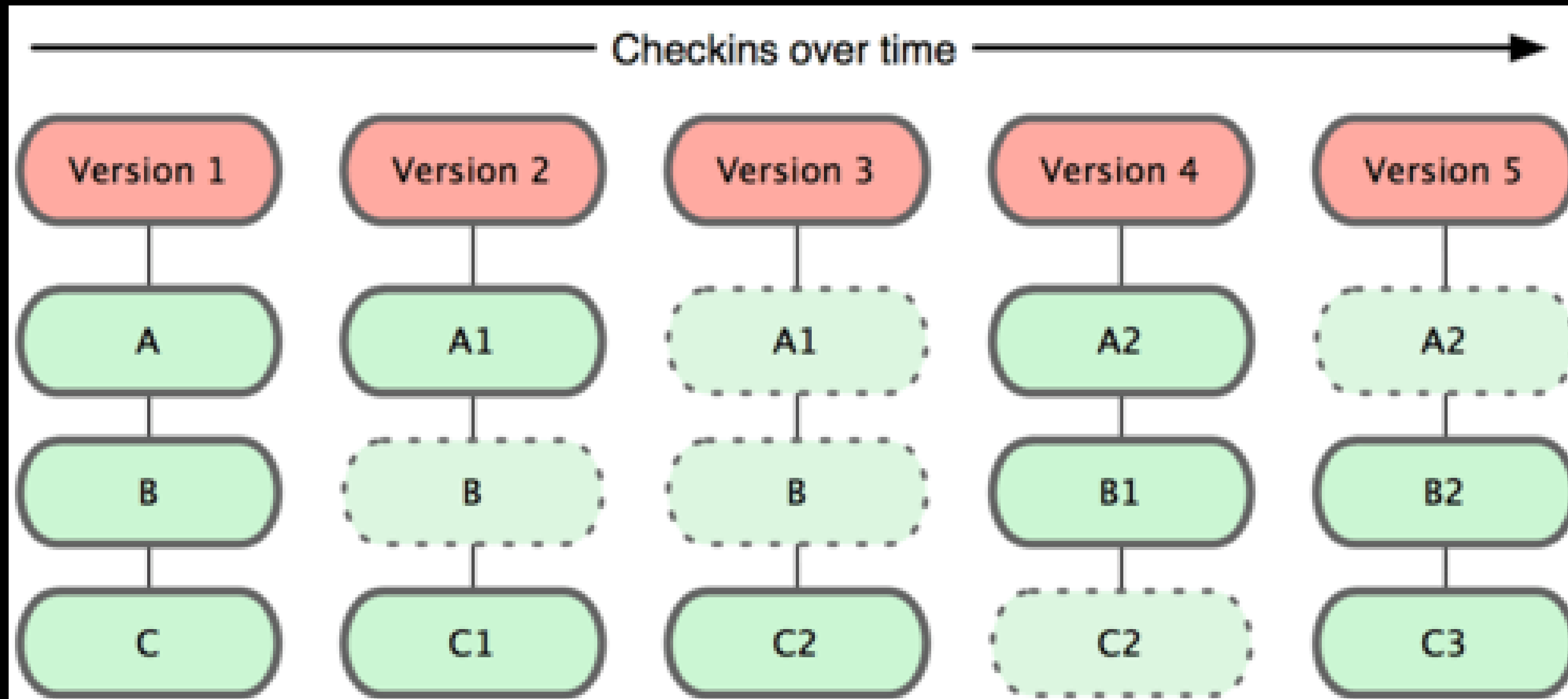
mittlerer Effekt

$$|d| > 0,8$$

großer Effekt

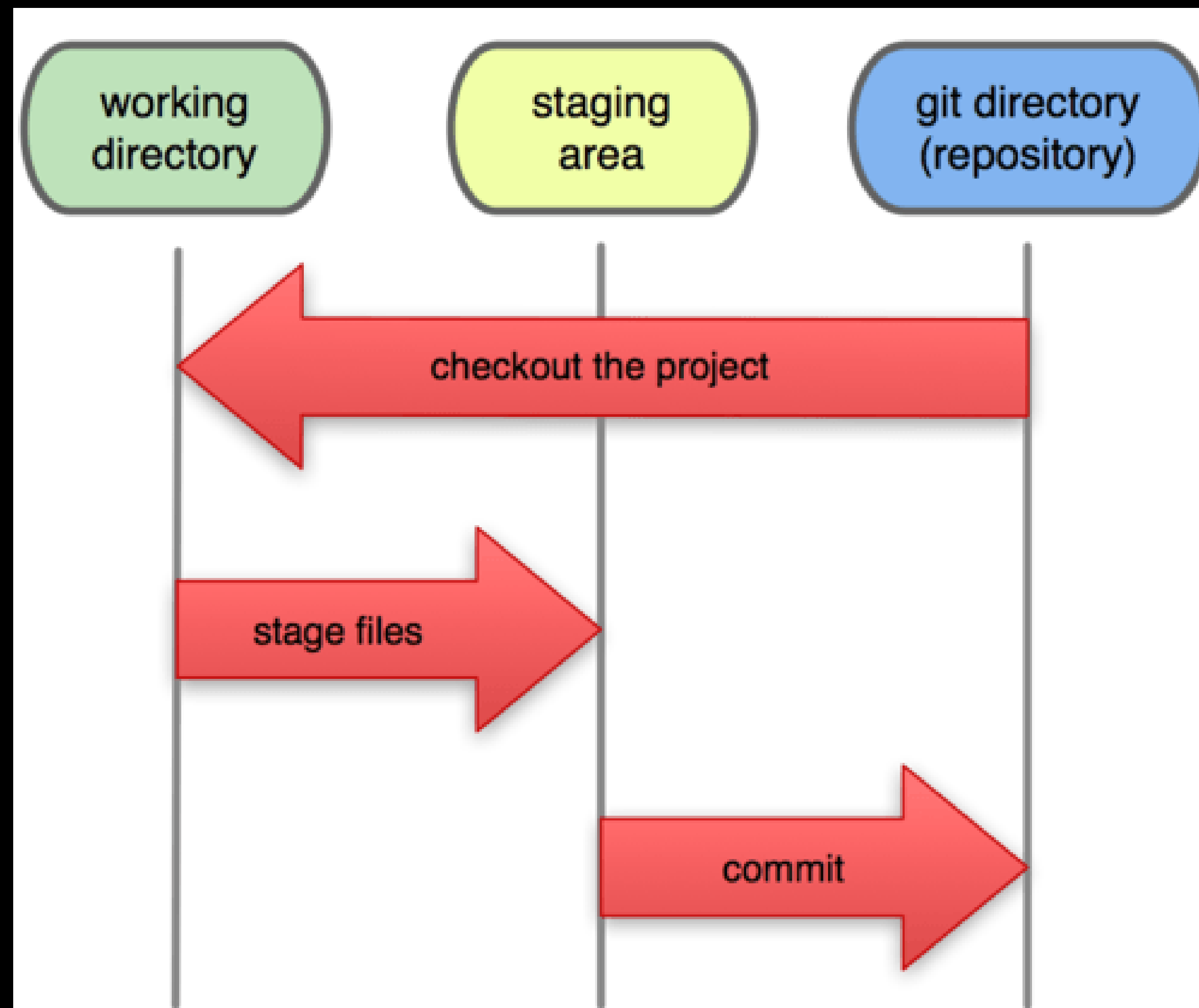
VERSIONIERUNG MIT GIT

LOKALE VERSIONIERUNG MIT GIT



- Alle Versionen werden in einem lokalen „Repository“ abgelegt.
- Jede Version enthält alle Dateien des Projektes, intern werden jedoch nur die eigentlichen Änderungen je Version gespeichert.

LOKALE VERSIONIERUNG MIT GIT



Eine Datei kann drei mögliche Zustände haben:

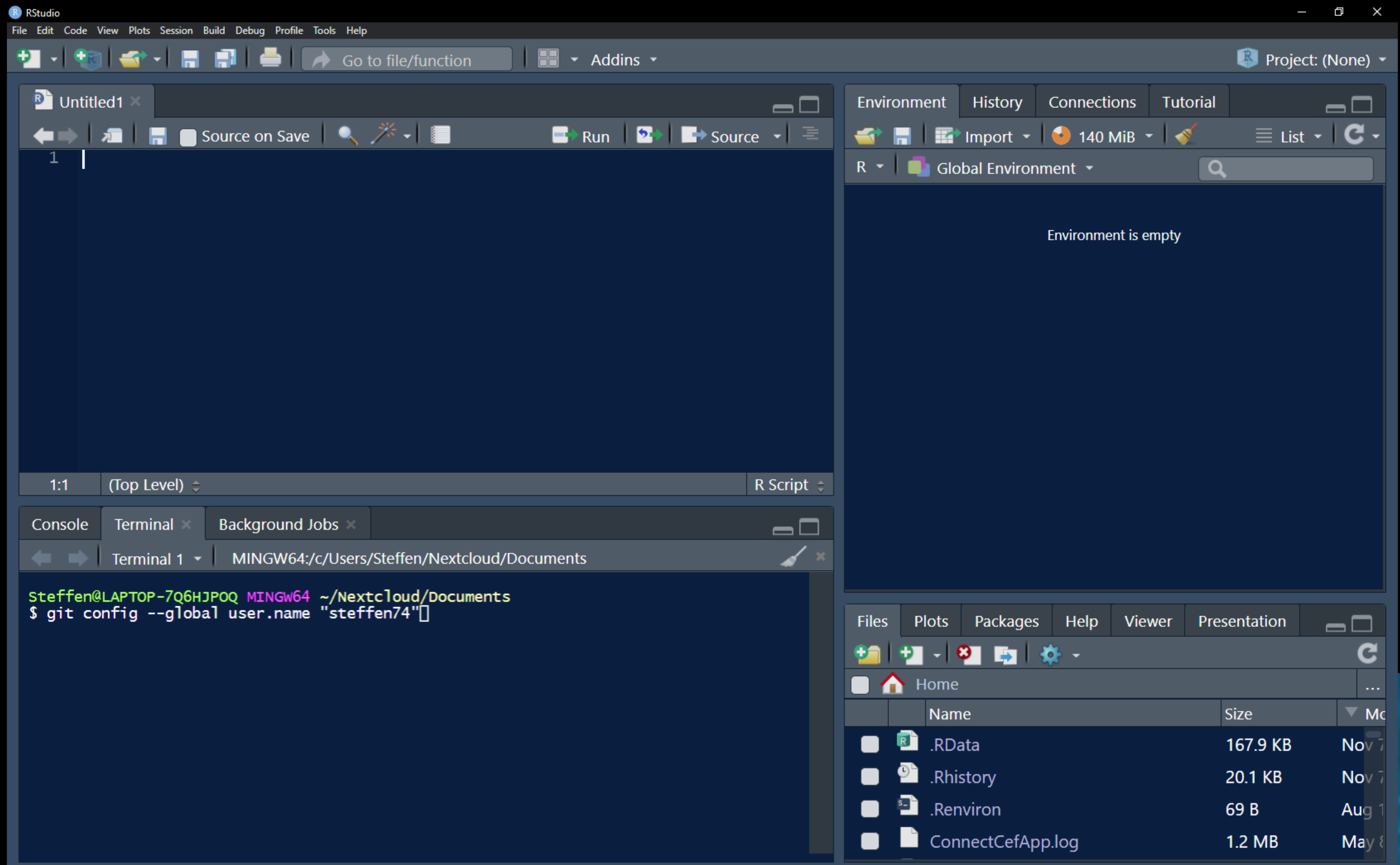
- **modified („geändert“)**
- **staged („vorgemerkt“) und**
- **committed („versioniert“).**

KONFIGURATION VON GIT

Vor der erstmaligen Verwendung von Git, muss einmalig definiert werden, in wessen Namen die Repositories des installierten Git verwaltet werden.

- **Führt in RStudio unten links im Fenster im Tab „Terminal“ (nicht verwechseln mit dem Tab „Console“!) die folgenden zwei Zeilen mit Eurem GitHub-Benutzernamen und Eurer E-Mail-Adresse aus :**

- ```
git config --global user.name "your_username"
git config --global user.email your_email@example.com
```



# BREAKOUT

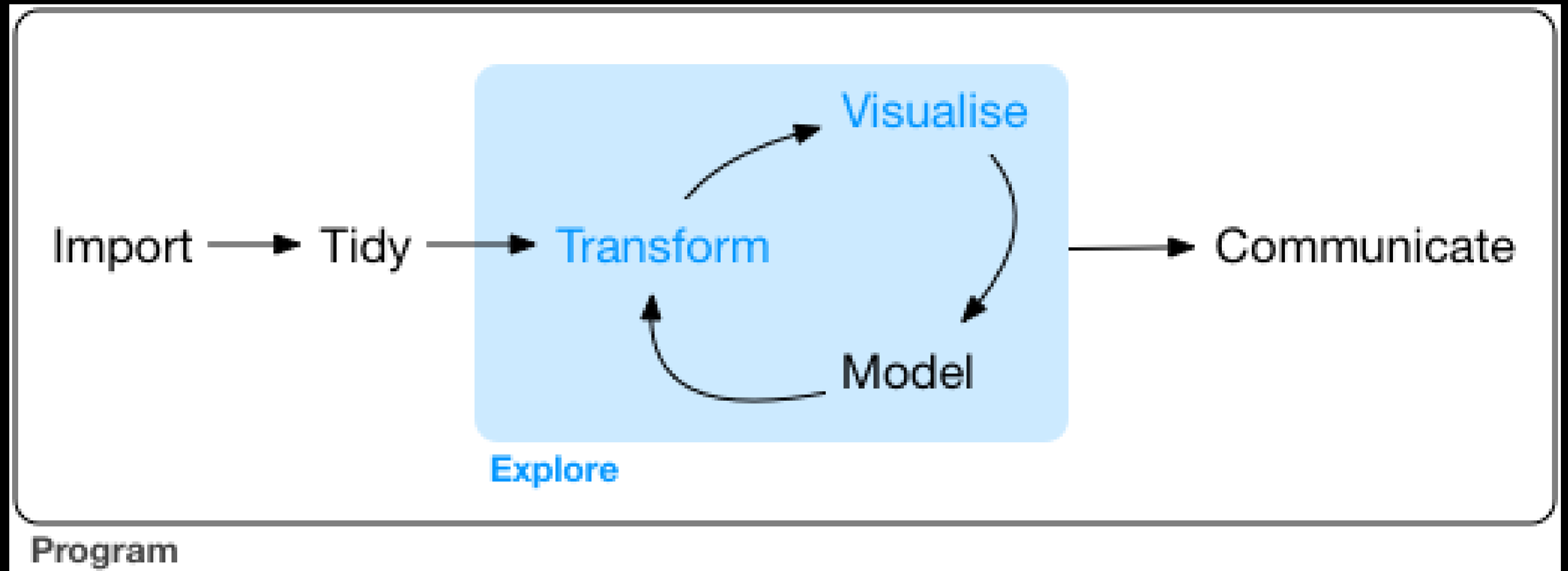
- 1) Wählt git als Versionierungsanwendung für Euer Projektverzeichnis aus.**
- 2) „Staged“ alle Dateien (markiert sie für das nächste Commit), die Ihr versionieren wollt und „committed“ sie dann.**
- 3) Führt ein erstes „Commit“ aus, um eine erste Projektversion mit allen bisherigen Dateien anzulegen.**
- 4) Legt ein neues R-Notebook im Projektverzeichnis an und legt eine neue Version einschließlich des Notebooks an.**
- 5) Schaut Euch die History Eures Repositories an.**

# **VERSIONIERUNG MIT GIT**

**Teil 1: Lokale Versionierung**

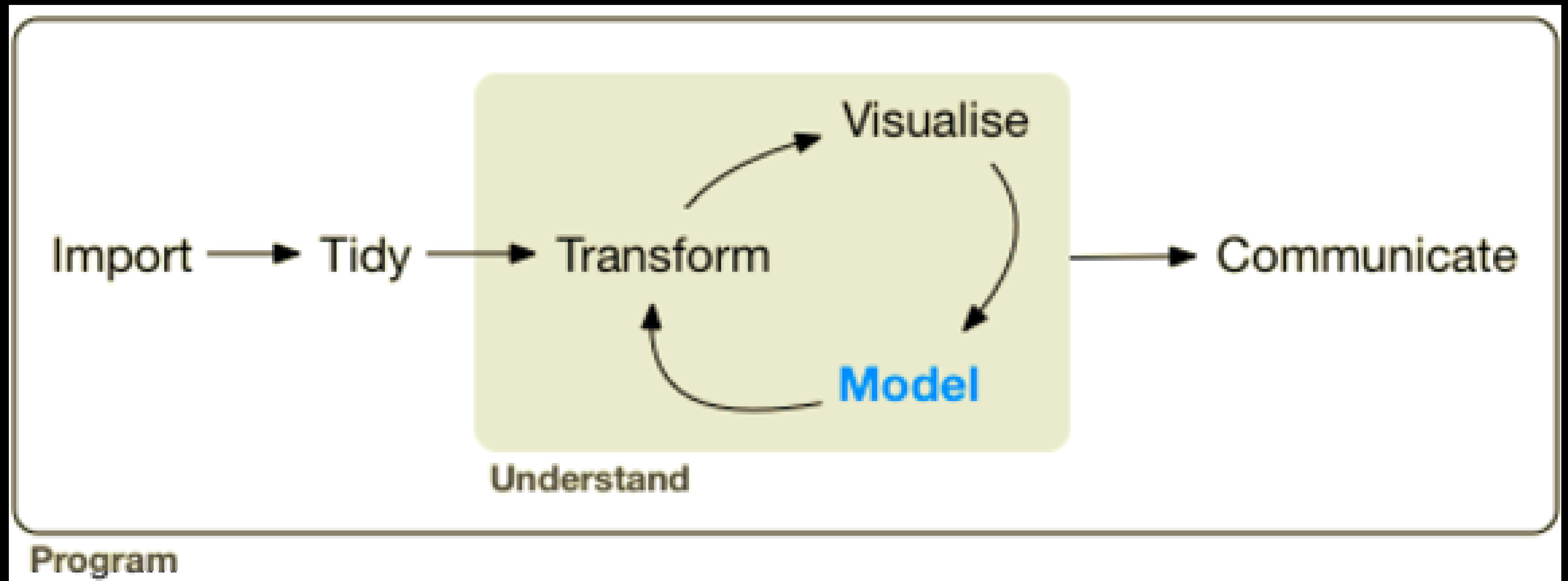
**Teil 2: Synchronisation der lokalen Versionierung mit einer Remote-Versionierung und Arbeiten im Team**

# DATENAUFBEREITUNG



(Wickham & Grolemund, 2016)

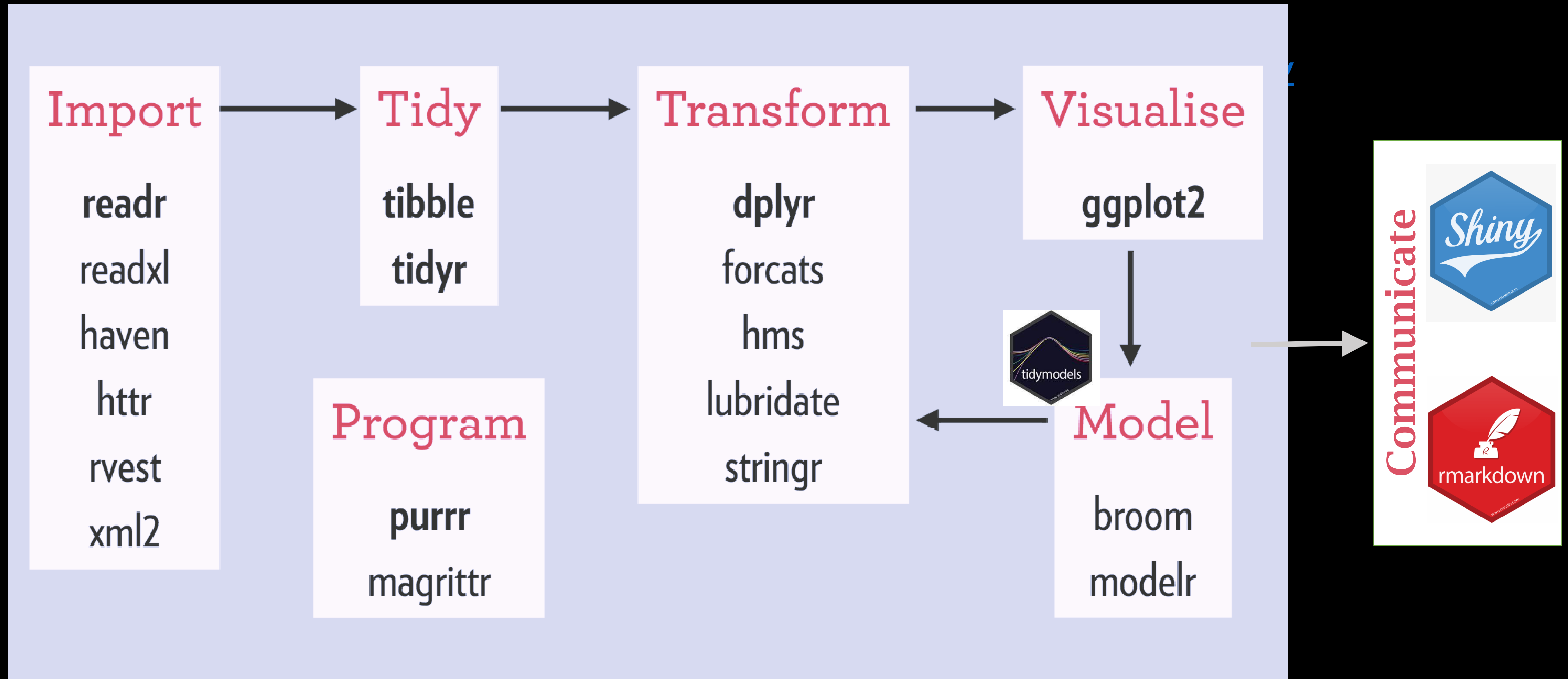
# DATENMODELLIERUNG



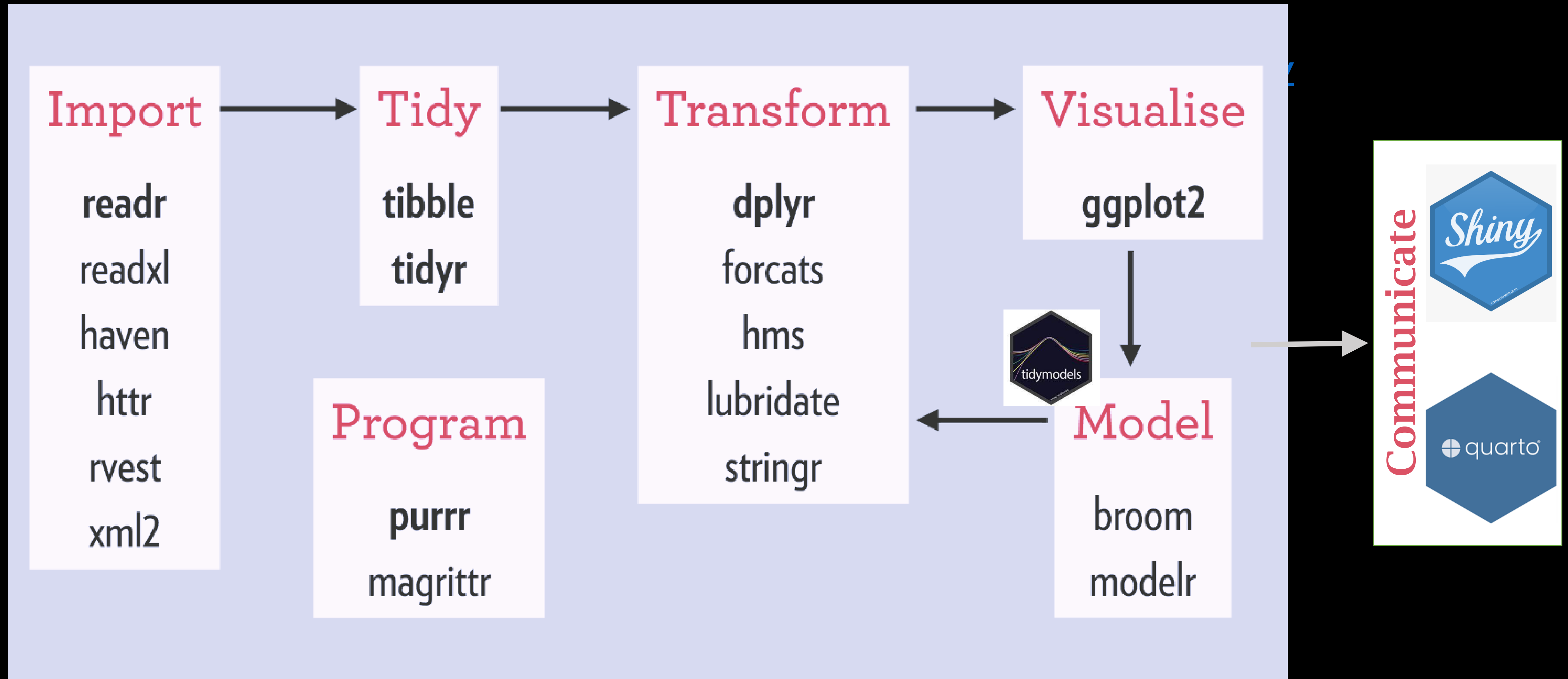
(Wickham & Grolemund, 2016)



# TIDYVERSE



# TIDYVERSE



# DPLYR – DATEN SELEKTIEREN & HINZUFÜGEN

## **select()**

Variablen (Spalten) auswählen

```
mpg %>%
```

## **filter()**

Fälle (Zeilen) auswählen

```
 select (class, hwy, cty) %>%
```

```
 filter (class=="suv") %>%
```

```
 mutate (mix = .5*hwy + .5*cty)
```

## **mutate()**

Variablen hinzufügen

# BEISPIEL 1

```
mpg %>%
```

```
 mutate(mean_milage = (cty+hwy)/2) %>%
```

```
 filter(mean_milage > 20) %>%
```

```
 select(manufacturer, mean_milage)
```

**Pipe Operator: %>%**

- Schrittweise Datenaufbereitung
- Vermeidung von Hilfsvariablen
- Erhöhung der Lesbarkeit des Programmcodes

# BEISPIEL 2

```
mpg %>%
 group_by(cyl) %>%
 summarise(n(), t.test(cty,hwy)$p.value)
```

**Gruppierung von Daten:** `group_by()`

- Vermeiden von Hilfsvariablen
- Deutliche Verkürzung des Programmcodes
- Erhöht die Lesbarkeit des Programmcodes

# DPLYR – DATENTABELLEN ZUSAMMENFÜHREN

## **left\_join(x, y)**

Return all rows from x, and all columns from x and y. Rows in x with no match in y will have NA values in the new columns. If there are multiple matches between x and y, all combinations of the matches are returned.

## **inner\_join(x, y)**

Return all rows from x where there are matching values in y, and all columns from x and y. If there are multiple matches between x and y, all combination of the matches are returned.

## **right\_join(), full\_join()**

```
daten <- left_join(umsatzdaten, kiwo)
```

# LUBRIDATE

## Umwandlung von Strings in ein Datumsformat

- Zum Beispiel: `dmy()` oder `ymd()`
- Erkennt automatisch unterschiedlich Formatierungen  
`mdy("4/1/17")`

## Umwandlung von Datumformaten in kategoriale Variablen

- Erkennt automatisch unterschiedlich Formatierungen
- Zum Beispiel: `mday()` oder `wday()`  
`economics %>%  
 mutate(weekday=wday(date))`

# STRINGR

## Allgemeine Funktion zur Zeichenersetzung

- `str_replace()`
- Erlaubt die Verwendung von „regular expressions“
  - `str_replace("AAA", "A", "B")`
  - `str_replace("AAA", "A$", "B")`

## Funktionen für spezielle Aufgaben

- „Wrapper-Funktionen“ von `str_replace()`
- Z.B. zum entfernen führender und nachstehender Leerzeichen:

`str_trim(" Vorname ")`

→ `str_replace_all(" Vorname ", "^[ \\s]+|[ \\s]+$", "")`



# PACKAGES ZUR DATENEXPLORATION

## **Skimr**

Gibt anhand verschiedener Statistiken einen schnellen Überblick zu den Variablen in einer Datentabelle. Je nach Inhalt der Variablen sind die einzelnen Statistiken aussagekräftig.

## **DataExplorer**

Enthält verschiedene Funktionen für grafische Darstellungen zu allen Variablen in einer Datentabelle, etwa mit Hilfe von Histogrammen.

# AUFGABEN

- Installiert das Package [styler](#) und schaut Euch [diesen](#) Coding Styleguide an
- Für eine Einführung in die Möglichkeiten von Regular Expressions, schaut Euch [dieses](#) Video (11 Minuten) an.
- Importiert die Daten `umsatzdaten_gekuerzt.csv`, `kiwo.csv` und `wetter.csv` und führt sie in einem Tibble zusammen.
- Nutzt die Funktionen aus den Packages [skimR](#) und [DataExplorer](#), um Euch die Daten in dem erstellten Tibble genauer anzusehen (teilt Euch die Aufgabe im Team auf).