Data Science für die Humangeographie: Ein pragmatischer Einstieg mit R

Konzeption quantitativer Forschung

Till Straube
straube@geo.uni-frankfurt.de

Institut für Humangeographie
Goethe-Universität Frankfurt

Wintersemester 2020/21

Inhaltsverzeichnis

Terminüberblick

Alle Sitzungen finden von 13 bis 16h c.t. statt

Datum	Sitzung	Inhalt
2. November 2020	1	Vorbesprechung
9. November 2020	2	Erste Schritte
16. November 2020	3	Text: Anderson 2008
23. November 2020	4	Datenstrukturen
30. November 2020	5	Visualisierungen
7. Dezember 2020	6	Text: Shelton et al. 2014
14. Dezember 2020	7	Geodaten
11. Januar 2021	8	Choroplethen
18. Januar 2021	9	Text: Chandra 2014
25. Januar 2021	10	HTML-Tabellen
1. Februar 2021	11	Web scraping
8. Februar 2021	12	Text: Straube 2021
15. Februar 2021	13	Präsentationen
31. März 2021		*Abgabe Exposé*

Online-Ressourcen

R Tutorials und eBooks

- **R for Data Science**
  https://r4ds.had.co.nz/
  Ausführliches Handbuch, Fokus auf Data Science
- **RStudio Cloud Primers**
  https://rstudio.cloud/learn/primers/1
- **Swirl**
  https://swirlstats.com/students.html
  Interaktives Tutorial als R-Paket, mit verschiedenen Lektionen
- **Quick-R**
  https://www.statmethods.net/r-tutorial/index.html
  Überblickartiges Tutorial, kurz und bündig
- **RStudio Cheat Sheets**

https://www.rstudio.com/resources/cheatsheets/
Einseitige Cheat Sheets zu verschiedenen Themen

- **Google's R Style Guide**
  https://google.github.io/styleguide/Rguide.xml
  Regeln für leserlichen R Code

Inspiration für Visualisierungen

- **R Graph Gallery**
  https://www.r-graph-gallery.com/
  Viele Beispiele für verschiedenste Visualisierungen
- **DDJ Katalog**
  http://katalog.datenjournalismus.net/#/
  Portfolio Datenjournalismus, leider etwas veraltet
- **Subreddits**
  https://www.reddit.com/r/dataisbeautiful
  https://www.reddit.com/r/DataArt/
  https://www.reddit.com/r/MapPorn/
- **Infographics**
  https://www.listendata.com/2019/06/create-infographics-with-r.html

Spezialthemen

- **HTML-Überblick**
  https://www.tutorialspoint.com/de/html/
- **Tutorial Reguläre Ausdrücke**
  https://danielfett.de/en/tutorials/tutorial-regulare-ausdrucke/
  Deutschsprachige Einführung zu regulären Ausdrücken

1 Vorbesprechung

1.1 Überblick

1.1.1 Seminar im Curriculum

- Dieses Seminar ist Bestandteil des Moduls BA3.
- Das Projektseminar besteht aus zwei Teilen über zwei Semester:
  - Konzeption quantitativer Forschung (Wintersemester)
  - Analyse quantitativer Daten (Sommersemester)
- Im Winter gibt es 12 inhaltliche Termine (davon 4x Textarbeit).
- Im Sommer wird das Seminar mit den selben Teilnehmer*innen fortgeführt.

1.1.2 Lernziele für das Wintersemester

Sie können…

- einfache Skripte in R eigenständig erstellen.
- Datensätze in vielfältigen Formaten visualisieren.

- Online-Ressourcen gezielt einsetzen.
- Möglichkeiten der Datenbeschaffung identifizieren.
- epistemologische Verschiebungen durch Data Science wiedergeben.

1.1.3 Technische Anforderungen

- Es sind keine Vorkenntnisse in R erforderlich.
- Sie brauchen einen Laptop, mit dem Sie gut arbeiten können.
- Wir benutzen die RStudio Cloud als Plattform.
- Sie brauchen einen ruhigen Arbeitsplatz.

1.1.4 Unterstützung im Corona-Semester

- Die Uni bietet einen "Semesterlaptop" an.
- Bei Bedarf kann ich gerne versuchen, Arbeitsplätze im Seminarraum (PEG) anzubieten.
- Bitte kontaktieren Sie mich per E-Mail, falls Sie einen Arbeitsplatz regelmäßig in Anspruch nehmen wollen würden.

1.2 Seminarformat

- Das Seminar findet jede Woche Montags, 13–16h c.t. statt.
- Der Zoom-Link, den Sie per E-Mail erhalten haben, bleibt gleich.
- Wir machen um ca. 14:25h eine zehnminütige Pause.
- Für Textbesprechungen wird die Gruppe zweigeteilt.
- Dieses Seminar findet in verschiedenen Modi statt:

1.2.1 Input und Plenum

- Ich rede oder moderiere (mit Folien oder ohne)
- Sie hören mir und Ihren Kommiliton*innen aufmerksam zu
- Sie "melden" sich für Redebeiträge oder Fragen (Zoom-Funktion)
- Die*der Chat-Verantwortliche unterbricht mich bei Klärungsbedarf

1.2.2 Think-pair-share

- Sie bearbeiten eine Fragestellung in zufälligen Zweier-Konstellationen (Breakout-Session)
- Nach einer vorgegebenen Zeitspanne kehren Sie ins Plenum zurück
- Ich fordere Sie ggf. auf, Ergebnisse und offene Fragen mit der Gruppe zu teilen

1.2.3 Follow the recipe

- Ich teile ein unvollständiges Beispielprojekt.
- Wir gehen die Teilschritte nach und nach durch.
- Ich "habe den Plan", stelle aber immer wieder Fragen ans Plenum.
- Sie vollziehen die Schritte an Ihrer eigenen Kopie des Projekts nach.
- Die*der Chat-Verantwortliche unterbricht mich bei Klärungsbedarf

1.2.4 Hands-on session

- Sie bearbeiten praktische Aufgabenstellungen alleine.
- Dabei sind sie in zufälligen Dreier-Konstellationen (Breakout-Session).
- Bei Fragen oder Problemen wenden Sie sich zunächst an Ihre Kleingruppe.
- Falls Sie nicht weiterkommen, fordern Sie Hilfe an (Zoom-Funktion).
- Ich reagiere auf Hilfegesuche oder schaue in zufälligen Gruppen vorbei.

1.2.5 Share your work

- Ich wähle eine Teilnehmer*in zufällig aus.
- Die Person teilt ihren Bildschirm und berichtet von ihrer Bearbeitung eines Problems.
- Alle anderen unterstützen solidarisch durch aktives Nachvollziehen, Nachfragen und Hinweise.

1.3 Leistungsnachweise

1.3.1 Exposé

- Zum Ende des Wintersemesters geben Sie ein Exposé für ein Untersuchungsvorhaben für das Sommersemester ab.
- Sie können sich mit bis zu vier Personen zusammenschließen.
- Die Projektgruppe besteht dann verbindlich für das Sommersemester.
- Damit steigen aber auch die Anforderungen an Umfang, Detail und technischen Anspruch.
- Umfang für das Exposé: max. 15k Zeichen inkl. Leerzeichen, exkl. Literaturverzeichnis
- Als Abgabetermin haben wir den 31. März vereinbart.

1.3.1.1 Inhalte

- Einführung ins Thema
- Forschungsstand / Literaturüberblick
- Herleitung einer klar abgegrenzten (vorläufigen) Forschungsfrage
- Konkrete Datenquellen
- Ideen für Verfahren und Visualisierungen

1.3.1.2 Bewertungskriterien

Alle kriterien werden mit einer (runden) Schulnote bewertet. Der gewichtete Schnitt ergibt die Gesamtnote.

Kriterium	Gewichtung	Erläuterung
Ziterweise und Formatierung	10%	Der Text erfüllt formale Anforderungen an Wissenschaftlichkeit.
Ausdruck und Rechtschreibung	10%	Der Text ist sprachlich gelungen.
Roter Faden	10%	Der Text ist übersichtlich strukturiert und die Einzelteile greifen gut ineinander.
Literatur	10%	Die zitierten Quellen sind für eine Einführung ins Thema geeignet und werden gut zusammengefasst.
Theorie	10%	Relevante wissenschaftliche Perspektiven werden anhand von geeigneter Fachliteratur aufgezeigt.

Kriterium	Gewichtung	Erläuterung
Fragestellung	10%	Die Forschungsfrage ist für das Vorhaben geeignet und wird überzeugend hergeleitet.
Datenquellen	20%	Die Datenquellen sind geeignet und detailliert beschrieben.
Design	20%	Das Untersuchungsvorhaben ist nachvollziehbar beschrieben, und der technische Anspruch ist dem Projektseminar angemessen.

1.3.2 Anwesenheit

- Es besteht Anwesenheitspflicht.
- Für Ihre ersten zwei Fehltermine pro Semester brauche ich keine Entschuldigung (aber Sie sollten das ggf. mit ihrer Projektgruppe absprechen).
- Sie sind dann selbstständig für die Nacharbeit der behandelten Themen zuständig.
- Im Falle eines zusätzlichen Fehltermins brauche ich ein Attest und einen Nachweis über Nacharbeit.
- Zur Anwesenheit gehört…
  - uneingeschränkte Aufmerksamkeit über die komplette Veranstaltungsdauer,
  - aktive Mitarbeit an Beispielen,
  - Bearbeitung von Übungsaufgaben,
  - aktive Beteiligung an Diskussionen.
- Eine eingeschaltete Kamera macht das allen Beteiligten leichter!

1.4 Lehrphilosophie

- Die folgenden vier “Säulen” habe ich mal im Rahmen einer Fortbildung als meine Lehrphilosophie definiert.
- Sie spiegeln meinen eigenen Anspruch an meine Lehre wider und sind als Vorschlag für ein gutes Miteinander zu verstehen.
- Begreifen Sie das gerne auch als Ermunterung, Aspekte hiervon einzufordern, wenn sie in der Veranstaltung zu kurz kommen.

1.4.1 Transparenz

- Erforderliche Leistungen und Bewertungskriterien sind vorab bekannt.
- Termine und Regelungen werden in der Vorbereitungssitzung verbindlich vereinbart.
- Aktuelle Lehrmaterialien stehen online durchgängig zur Verfügung.

1.4.2 Praktische Übungen

- Eigenständige Anwendung steht im Vordergrund.
- Verfahren und Techniken werden mit Beispielen und Übungen erarbeitet.
- Die perfekte Aufgabe ist immer ein bisschen “zu schwer”.
- Toleranz für Frustration ist eine wichtige Fähigkeit und lässt sich trainieren.

1.4.3 Geschützte Räume

- Alle können sich im Plenum respektiert und sicher fühlen. Verletzendes Verhalten wird benannt.
- Es gibt einen vertrauensvollen Rahmen für ehrlichen Austausch.
- Frustrationen und Momente des Scheiterns werden ernst genommen und konstruktiv bearbeitet.

1.4.4 Kritische Reflexion

- Auch Teilnehmende, die kein weiterführendes Interesse an der Anwendung quantitativer Verfahren haben, sind im Seminar gut aufgehoben.
- Verfahren werden kontextualisiert, ihre Limitationen werden aufgezeigt.
- Kritische Forschung zu quantitativen Praktiken wird besprochen.

2 Erste Schritte

2.1 Vorbereitung

- Machen Sie sich einen kostenlosen Account auf https://rstudio.cloud
- Treten Sie dem Seminar-Workspace bei. (Sie erhalten eine Einladung per E-Mail.)
- Optional/alternativ: installieren Sie R und RStudio auf Ihrem Computer.

2.2 Lernziele für diese Sitzung

Sie können…

- Rechenoperatoren einsetzen.
- Variablen zuweisen.
- Funktionen aufrufen.
- Hilfe zu Funktionen anzeigen.
- die wichtigsten Variablentypen bestimmen.
- zwischen Variablentypen konvertieren.

2.3 Operatoren

Zunächst stellen wir fest, dass man die R-Konsole ganz banal als Taschenrechner benutzen kann:

```
1 + 4
## [1] 5
8 / 3
## [1] 2.666667
(2.45 + 3.5) * 7
## [1] 41.65
```

Die Zeichen +, −, ∗ usw. heißen in der Informatik Operatoren oder Infixe (weil sie immer zwischen zwei Werten stehen).

2.4 Variablen

Variablen funktionieren so, dass man einem *Wert* einen Namen gibt. Die Zuweisung folgt dabei dem Schema `NAME <- WERT`:

```
x <- 5
```

Nach einer erfolgreichen Variablenzuweisung gibt die Konsole *keine* Rückmeldung, sondern nur bei Fehlern.

`x` steht jetzt für die Zahl fünf. Mit dieser Variable können wir jetzt genauso rechnen wie mit einer Zahl:

```
x + 3
## [1] 8
```

Auch die Zuweisung von Variablen kann Rechenoperationen und andere Variablen enthalten:

```
y <- (x * 2) - 1
print(y)
## [1] 9
```

Der Befehl `print(y)` ist dabei ganz einfach die Anweisung an die Konsole, den Wert für `y` auszugeben. Das passiert zwar auch, wenn man nur `y` eingibt, aber `print(y)` (oder `print(x)`, `print(1 + 1)`, usw.) ist die formal korrekte Schreibweise.

Der Wert einer Variable kann auch verändert werden. Dafür weisen wir ihr einfach einen neuen Wert zu:

```
x <- 20
print(x)
## [1] 20
```

Eine Besonderheit ist, dass der alte Wert der Variable auch innerhalb der Zuweisung eines neuen Werts benutzt werden darf. Das kann in einem Script sehr praktisch sein. Wenn wir `x` also um 0,5 erhöhen wollen, sieht das so aus:

```
x <- x + 0.5
print(x)
## [1] 20.5
```

Dabei wird als Dezimaltrennzeichen ausschließlich der Punkt verwendet.

2.5 Konstanten

Manche benannten Werte sind schon in R eingebaut:

```
print(pi)
## [1] 3
```

Diese Werte heißen üblicherweise “Konstanten” – allerdings lassen sie sich in R auch überschreiben!

```
pi <- 3
print(pi)
## [1] 3
```

2.6 Funktionen

Mit `print()` haben wir schon unsere erste *Funktion* kennengelernt. R stellt uns eine Vielzahl von verschiedenen Funktionen zur Verfügung, und sie werden immer nach dem gleichen Schema benutzt: `FUNKTIONSNAME(PARAMETER)`.

Parameter (auf Englisch auch "arguments") sind die Werte, die als Input an die Funktion übergeben werden. Je nach Funktion können das auch mehrere Werte sein, die dann durch Kommas getrennt werden. So nimmt die Funktion `max()`, die den Maximalwert bestimmt, beliebig viele Zahlen als Parameter:

```
max(1, 2, 2, 5, 4, 3)
## [1] 5
```

Die Funktion `round()` hat als optionalen Parameter die Anzahl der Nachkommastellen, auf die gerundet werden soll. Wenn er nicht angegeben wird, nimmmt dieser Parameter immer den Wert 0 an:

```
round(4.567)
## [1] 5
```

Aber er lässt sich auch spezifizieren:

```
round(4.567, digits = 2)
## [1] 4.57
```

Dabei sind die folgenden Ausdrücke identisch:

```
round(4.567, digits = 2)
## [1] 4.57
round(4.567, 2)
## [1] 4.57
round(digits = 2, 4.567)
## [1] 4.57
```

Was Funktionen genau machen und welche Parameter sie dabei nehmen, ist in der R-Dokumentation sehr ausführlich (und auf den ersten Blick recht kompliziert) beschrieben. Ganz am Ende der Hilfeseite finden sich oft Beipsiele. Die Hilfe zu einer Funktion kann mit folgendem Befehl aufgerufen werden:

```
?max
```

Notiz am Rande: Auch die Infix-Operatoren +, −, ∗, usw. sind eigentlich nur verkürzte Schreibweisen von Funktionen. Mit "backticks" (‘) lassen sie sich in vollwertige Funktionen zurückverwandeln:

```
`+`(2, 2)
## [1] 4
```

2.7 Strings

R kann nicht nur mit Zahlen umgehen, sondern auch mit Text. Ein *String* ist eine Aneinanderreihung von Buchstaben, und wird mit einfachen oder doppelten Anführungszeichen umschlossen:

```
print("Hello, World!")
## [1] "Hello, World!"
```

Auch Variablen können Strings als Wert haben:

```
name <- "Hase"
```

Es gibt auch Funktionen, die Strings als Parameter nehmen. `paste` fügt Strings aneinander:

```
paste("Mein Name ist", name)
## [1] "Mein Name ist Hase"
```

2.8 Datentypen

Den *Typ* einer Variable oder eines Wertes bestimmen wir durch den Befehl `str()`:

```
str(name)
##  chr "Hase"
str(10)
##  num 10
```

Dabei steht `chr` („character“) für Strings und `num` („numeric“) für Zahlen.

Ein weiterer Variablentyp ist `logi` („logical“), der prinzipiell nur die Werte `TRUE` oder `FALSE` annehmen kann. Dieser Typ heißt auch Boolsche Variabel:

```
str(FALSE)
##  logi FALSE
```

Soweit es ein eindeutiges Ergebnis gibt, kann R mit den entsprechenden Befehlen Werte vom einen in den anderen Typ umwandeln:

```
as.numeric("1000")
## [1] 1000
as.character(x)
## [1] "20.5"
as.logical(0)
## [1] FALSE
```

Kann R einen Wert nicht umwandeln, dann kommt dabei `NA` raus (mit einer Warnung):

```
as.numeric("Hallo!")
## [1] NA
```

`NA` („not available/assigned“) ist dabei ein besonderer Wert, den jeder Variablentyp annehmen kann.

2.9 Aufgaben

2.9.1 Rechnen

Lösen Sie folgende Rechenaufgaben mit Hilfe von R:

- 4 plus 10
- 8 mal 12
- 4 minus 7
- 3 hoch 18
- 4,5 geteilt durch die Summe von 5 und 8
- Quadratwurzel aus 101
- Kubikwurzel aus 12

2.9.2 Variablen

Weisen Sie den Variablen a bis g folgende Werte zu:

a) TRUE
b) 2
c) Ihren Namen

d) Die Quadratwurzel aus b
e) 8 $\frac{1}{4}$
f) Das vierfache von e
g) Die aktuelle Uhrzeit mit Datum und Zeitzone (automatisch generiert)

2.9.3 Datentypen

Bestimmen Sie die Typen der Variablen a bis g.

Finden Sie je zwei Beispiele für die Umwandlung…

- von `numeric` zu `character`
- von `numeric` zu `logical`
- von `character` zu `logical`
- von `character` zu `numeric`
- von `logical` zu `character`
- von `logical` zu `numeric`
- von `character` zu `Date`
- von `Date` zu `numeric`

(`Date` ist kein eigentlicher Datentyp, aber erfüllt an dieser Stelle denselben Zweck.)

2.9.4 Swirl

Folgen Sie den Anleitungen, um Swirl zu installieren: https://swirlstats.com/students.html

Absolvieren Sie Lektion 1 („Basic Building Blocks“).

2.9.5 Recherche

Recherchieren Sie:

- Welche Funktion gibt den absoluten Wert einer Zahl aus? (z.B. -4 ergibt 4, 8 ergibt 8)
- Welche Konstanten sind in R „eingebaut“?
- Wie bestimmt man den „Rest“ einer Division? (z.B. 40 geteilt durch 7 hat den Rest 5)
- In der Statistik wird zwischen stetigen und diskreten Variablen unterschieden. Welche äquivalente Unterscheidung nimmt R vor?

2.9.6 Kniffliges

Lösen Sie die folgenden Probleme:

- Durch welchen Ausdruck lässt sich eine Zahl auf die nächste *gerade* Zahl runden? (z.B. 18,9 auf 18,0 oder 21,2 auf 22,0)
- Durch welchen Ausdruck lässt sich eine Zahl auf die nächste *halbe* Zahl abrunden? (z.B. 18,9 auf 18,5 oder 21,2 auf 21,0)
- Absolvieren Sie in die Lektion 8 („Logic“).
- Machen Sie sich mit der Funktion `xor()` vertraut. Finden Sie einen Ausdruck, der `xor()` simuliert, aber nur aus Infix-Operatoren besteht.
- Was bedeutet „strong“ bzw „weak typing“? Wie ist R hier einzuordnen?
- Was sind funktionale Programmiersprachen? Welche Eigenschaften von R sind funktional, welche nicht?
- Starten Sie den R Track in Excercism
- Richten Sie sich ein IDE *außer* RStudio für einen R Workflow ein.

3 Text: Anderson 2008

3.1 Lesetext

Anderson, Chris. 2008. *The End of Theory: The Data Deluge Makes the Scientific Method Obsolete.* URL: https://www.wired.com/2008/06/pb-theory/ (zugegriffen: 11. Juli 2017).

3.2 Fragen an den Text

1. Um welche Art von Text handelt es sich? Wer ist der Autor, und an wen wendet er sich?
2. Was ist das zentrale Anliegen des Texts? Welche Entwicklungen werden beschrieben?
3. Mit welchen Begriffen würden wir diese Phänomene heute beschreiben?
4. Aus heutiger Perspektive: Hatte der Autor recht? Warum / warum nicht?
5. In welchen Punkten stimmen Sie dem Autor zu? Wie würden Sie den Text problematisieren?

4 Datenstrukturen

4.1 Lernziele dieser Sitzung

Sie können…

- die verschiedenen Strukturen für Datensätze in R benennen.
- Vektoren generieren.
- einfache Befehle mit Vektoren durchführen.
- Beispieldatensätze aufrufen und beschreiben.

4.2 Vektoren

Vektoren (engl. *vectors*) sind eindimensionale Reihen von Werten gleichen Typs. Sie bilden einen wichtigen Baustein von R und von den hier im Seminar besprochenen Inhalten.

Sie können manuell mit der Funktion `c(...)` erstellt werden und wie Variablen benannt werden:

```
alter <- c(39, 49, 63, 44, 40)
alter
## [1] 39 49 63 44 40
```

Es gibt darüber hinaus aber auch Möglichkeiten, Vektoren automatisch zu generieren:

```
1:10
##  [1]  1  2  3  4  5  6  7  8  9 10
seq(100, 10, by=-10)
##  [1] 100  90  80  70  60  50  40  30  20  10
```

Buchstaben sind als Vektor in R eingebaut:

```
letters
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
## [13] "m" "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x"
## [25] "y" "z"
LETTERS
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L"
```

```
## [13] "M" "N" "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X"
## [25] "Y" "Z"
```

Manche Funktionen sind speziell für Vektoren gedacht:

```
rev(1:10)
##  [1] 10  9  8  7  6  5  4  3  2  1
```

Andere Funktionen, die für einzelne Werte gedacht sind, werden für jeden Wert einzeln ausgeführt:

```
toupper("hallo")
## [1] "HALLO"
toupper(c("ein", "paar", "strings"))
## [1] "EIN"     "PAAR"    "STRINGS"
```

Elemente von Vektoren können mit eckigen Klammern einzeln oder selektiv angesprochen bzw entfernt werden:

```
letters[2]
## [1] "b"
letters[2:3]
## [1] "b" "c"
letters[-2]
##  [1] "a" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m"
## [13] "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y"
## [25] "z"
```

Vektoren können wie Variablen benutzt werden:

```
2018 - alter
## [1] 1979 1969 1955 1974 1978
paste(alter, "ist ein gutes Alter")
## [1] "39 ist ein gutes Alter" "49 ist ein gutes Alter"
## [3] "63 ist ein gutes Alter" "44 ist ein gutes Alter"
## [5] "40 ist ein gutes Alter"
```

`length(x)` gibt die Anzahl der Elemente in einem Vektor `x` aus:

```
length(alter)
## [1] 5
```

Von Verteilungen, die als Vektoren vorliegen, lassen sich statistische Parameter einfach errechnen:

```
mean(alter)
## [1] 47
median(alter)
## [1] 44
sd(alter)
## [1] 9.77241
IQR(alter)
## [1] 9
```

(Aber `IQR()` berechnet anders als in der Vorlesung besprochen!)

Wir können den Mittelwert auch mit Hilfe der `sum()` und `length()` Funktionen selbst berechnen:

```
sum(alter) / length(alter)
## [1] 47
```

4.3 Matritzen

Matritzen (engl. *matrix*) sind zweidimensionale Reihen von Werten gleichen Typs.

```
matrix(1:15, nrow=3)
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7   10   13
## [2,]    2    5    8   11   14
## [3,]    3    6    9   12   15
```

Sie spielen in diesem Seminar aber keine große Rolle.

4.4 Listen

Listen sind eindimensionale Reihen von Werten, wobei der Typ egal ist:

```
list("Hallo", 10, F)
## [[1]]
## [1] "Hallo"
##
## [[2]]
## [1] 10
##
## [[3]]
## [1] FALSE
```

Dabei können die Werte benannt sein, und Listen können Unterlisten enthalten:

```
profil <- list(name="Till", plz=60326, x=list(TRUE, TRUE, FALSE))
str(profil)
## List of 3
##  $ name: chr "Till"
##  $ plz : num 60326
##  $ x   :List of 3
##   ..$ : logi TRUE
##   ..$ : logi TRUE
##   ..$ : logi FALSE
```

4.5 Data Frames

Data frames sind tabellarische Daten. Die Werte in jeder Spalte haben dabei denselben Typ.

Viele Beispieldatensätze sind in Form von data frames in R eingebaut.

`head(x)` gibt nur die ersten sechs Zeilen aus:

```
head(mtcars)
##                    mpg cyl disp  hp drat    wt  qsec
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02
## Valiant           18.1   6  225 105 2.76 3.460 20.22
##                   vs am gear carb
## Mazda RX4          0  1    4    4
## Mazda RX4 Wag      0  1    4    4
## Datsun 710         1  1    4    1
## Hornet 4 Drive     1  0    3    1
## Hornet Sportabout  0  0    3    2
## Valiant            1  0    3    1
```

4.6 Tibbles

Tibbles können alles, was data frames können, und haben darüber hinaus noch Funktionen, die wir später kennenlernen werden.

Sie sind teil der Paketsammlung `tidyverse`, die einmalig installiert werden muss und dann geladen werden kann:

```
library(tidyverse)
```

Ein Beispieldatensatz ist `diamonds`:

```
diamonds
## # A tibble: 53,940 x 10
##    carat cut       color clarity depth table price     x
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl>
##  1 0.23  Ideal     E     SI2      61.5    55   326  3.95
##  2 0.21  Premium   E     SI1      59.8    61   326  3.89
##  3 0.23  Good      E     VS1      56.9    65   327  4.05
##  4 0.290 Premium   I     VS2      62.4    58   334  4.2
##  5 0.31  Good      J     SI2      63.3    58   335  4.34
##  6 0.24  Very Go~  J     VVS2     62.8    57   336  3.94
##  7 0.24  Very Go~  I     VVS1     62.3    57   336  3.95
##  8 0.26  Very Go~  H     SI1      61.9    55   337  4.07
##  9 0.22  Fair      E     VS2      65.1    61   337  3.87
## 10 0.23  Very Go~  H     VS1      59.4    61   338  4
## # ... with 53,930 more rows, and 2 more variables:
## #   y <dbl>, z <dbl>
```

Einzelne Spalten lassen sich mit $ ansprechen und verhalten sich dann wie Vektoren:

```
str(diamonds$carat)
##  num [1:53940] 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
mean(diamonds$depth)
```

```
## [1] 61.7494
```

4.7 Aufgaben

4.7.1 Vektoren

- Generieren Sie die folgenden Vektoren (und seien Sie dabei möglichst faul).

```
## [1]  TRUE FALSE FALSE
##  [1]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE
##  [9]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE
## [17]  TRUE FALSE  TRUE FALSE
##  [1]  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
## [18] 18 19 20
##  [1] "Z" "Y" "X" "W" "V" "U" "T" "S" "R" "Q" "P" "O"
## [13] "N" "M" "L" "K" "J" "I" "H" "G" "F" "E" "D" "C"
## [25] "B" "A"
##  [1] "aA" "bB" "cC" "dD" "eE" "fF" "gG" "hH" "iI" "jJ"
## [11] "kK" "lL" "mM" "nN" "oO" "pP" "qQ" "rR" "sS" "tT"
## [21] "uU" "vV" "wW" "xX" "yY" "zZ"
```

- Wandeln Sie die Typen der ersten drei obigen Vektoren um:

```
## [1] 1 0 0
##  [1] 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
##  [1] "2"  "4"  "6"  "8"  "10" "12" "14" "16" "18" "20"
## [11] "22" "24" "26" "28" "30" "32" "34"
```

4.7.2 Tibbles

- Schauen Sie sich den Beispieldatensatz `faithful` an.
- Wandeln Sie den Datensatz `faithful` in einen tibble um.
- Wenden Sie `str()` auf den Datensatz an. und Interpretieren Sie das Ergebnis.
- Erstellen Sie einen eigenen tibble mit Vornamen, Nachnamen und Alter von (ausgedachten?) Menschen.
- Lassen Sie sich nur die zweite Zeile des tibbles `diamonds` anzeigen
- Lassen Sie sich nur jede zweite Zeile des tibbles `diamonds` anzeigen

4.7.3 Statistik

- Berechnen Sie die durchschnittliche Eruptionszeit im Datensatz `faithful` (als tibble).
- Berechnen Sie Varianz und Standardabweichung der Karatzahl im Beispieldatensatz `diamonds`
- Was sagen die einzelnen Kennzahlen des Befehls `summary(x)` aus?

4.7.4 Swirl

Absolvieren Sie die folgenden Swirl-Lektionen (Anleitung zu Swirl s. letzte Lektion):

- 3: Sequences of Numbers
- 4: Vectors
- 5: Missing Values
- 6: Subsetting Vectors

4.7.5 Recherche

- Nach welcher Methode berechnet R den Quartilsabstand einer Verteilung (im Unterschied zur Vorlesung)?
- Finden Sie fünf Befehle, die mit tibbles funktionieren, aber nicht mit data frames.
- Welche Pakete sind Teil des `tidyverse`? Wofür sind sie gedacht?
- Lesen Sie die Hilfe zu `tibble::tibble`. Recherchieren Sie eigenständig unklare Begriffe.

4.7.6 Kniffliges

- Kehren Sie auf möglichst elegante und allgemeingültige Weise die Reihenfolge eines Vektors um, ohne die Funktion `rev()` zu benutzen.

5 Visualisierungen

5.1 Lernziele dieser Sitzung

Sie können…

- einfache Befehle zur Visualisierung in Base R anwenden.
- die Grammatik von `ggplot2` für Visualisierungen in Grundzügen wiedergeben und anwenden.
- eigene Ideen für Visualisierungen entwickeln und umsetzen.

5.2 Voraussetzungen

Für diese Lektion benötigen wir das Paket `tidyverse`:

```
library(tidyverse)
```

Und einen Datensatz, der in Form eines tibble vorliegt. Der Beispieldatensatz `diamonds` wird mitgeliefert:

```
diamonds
## # A tibble: 53,940 x 10
##    carat cut       color clarity depth table price     x
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl>
##  1 0.23  Ideal     E     SI2      61.5    55   326  3.95
##  2 0.21  Premium   E     SI1      59.8    61   326  3.89
##  3 0.23  Good      E     VS1      56.9    65   327  4.05
##  4 0.290 Premium   I     VS2      62.4    58   334  4.2
##  5 0.31  Good      J     SI2      63.3    58   335  4.34
##  6 0.24  Very Go~  J     VVS2     62.8    57   336  3.94
##  7 0.24  Very Go~  I     VVS1     62.3    57   336  3.95
##  8 0.26  Very Go~  H     SI1      61.9    55   337  4.07
##  9 0.22  Fair      E     VS2      65.1    61   337  3.87
## 10 0.23  Very Go~  H     VS1      59.4    61   338  4
## # ... with 53,930 more rows, and 2 more variables:
## #   y <dbl>, z <dbl>
```

Wenn wir mögen, können wir ihn mit der Funktion `data()` explizit in unser Environment laden:

```
data(diamonds)
```

5.3 Überblick

Einen ersten Überblick kriegen wir zum Einen durch den Befehl `str()`, der uns die Typen in den Spalten anzeigt:

```
str(diamonds)
## tibble [53,940 x 10] (S3: tbl_df/tbl/data.frame)
##  $ carat  : num [1:53940] 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
##  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
##  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
##  $ depth  : num [1:53940] 61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table  : num [1:53940] 55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int [1:53940] 326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num [1:53940] 3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num [1:53940] 3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num [1:53940] 2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

Zum Anderen gibt die Hilfefunktion Auskunft über den Datensatz und die einzelnen Variablen (Metadaten):

```
?diamonds
```

Einen Überblick über die wichtigsten statistischen Parameter erhalten wir mit:

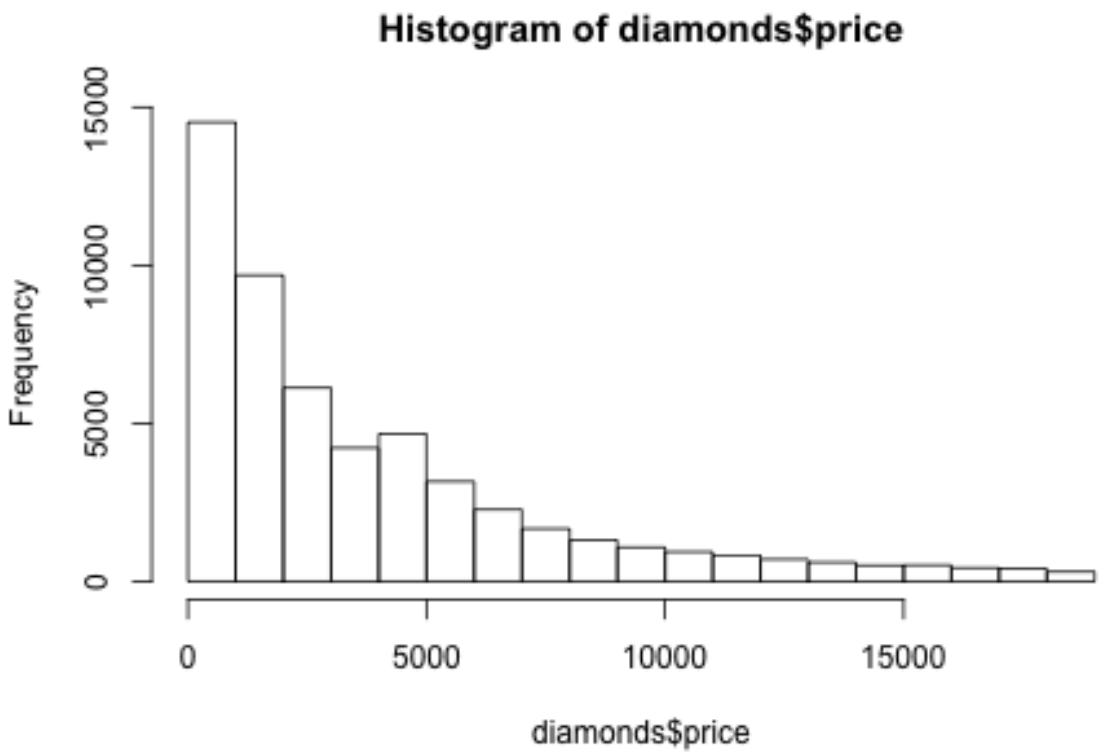
```
summary(diamonds)
##      carat               cut        color
##  Min.   :0.2000   Fair      : 1610   D: 6775
##  1st Qu.:0.4000   Good      : 4906   E: 9797
##  Median :0.7000   Very Good:12082   F: 9542
##  Mean   :0.7979   Premium   :13791   G:11292
##  3rd Qu.:1.0400   Ideal     :21551   H: 8304
##  Max.   :5.0100                      I: 5422
##                                      J: 2808
##     clarity          depth            table
##  SI1    :13065   Min.   :43.00   Min.   :43.00
##  VS2    :12258   1st Qu.:61.00   1st Qu.:56.00
##  SI2    : 9194   Median :61.80   Median :57.00
##  VS1    : 8171   Mean   :61.75   Mean   :57.46
##  VVS2   : 5066   3rd Qu.:62.50   3rd Qu.:59.00
##  VVS1   : 3655   Max.   :79.00   Max.   :95.00
##  (Other): 2531
##      price             x                y
##  Min.   :  326   Min.   : 0.000   Min.   : 0.000
##  1st Qu.:  950   1st Qu.: 4.710   1st Qu.: 4.720
##  Median : 2401   Median : 5.700   Median : 5.710
##  Mean   : 3933   Mean   : 5.731   Mean   : 5.735
##  3rd Qu.: 5324   3rd Qu.: 6.540   3rd Qu.: 6.540
```

```
##  Max.   :18823   Max.   :10.740   Max.   :58.900
##
##        z
##  Min.   : 0.000
##  1st Qu.: 2.910
##  Median : 3.530
##  Mean   : 3.539
##  3rd Qu.: 4.040
##  Max.   :31.800
##
```

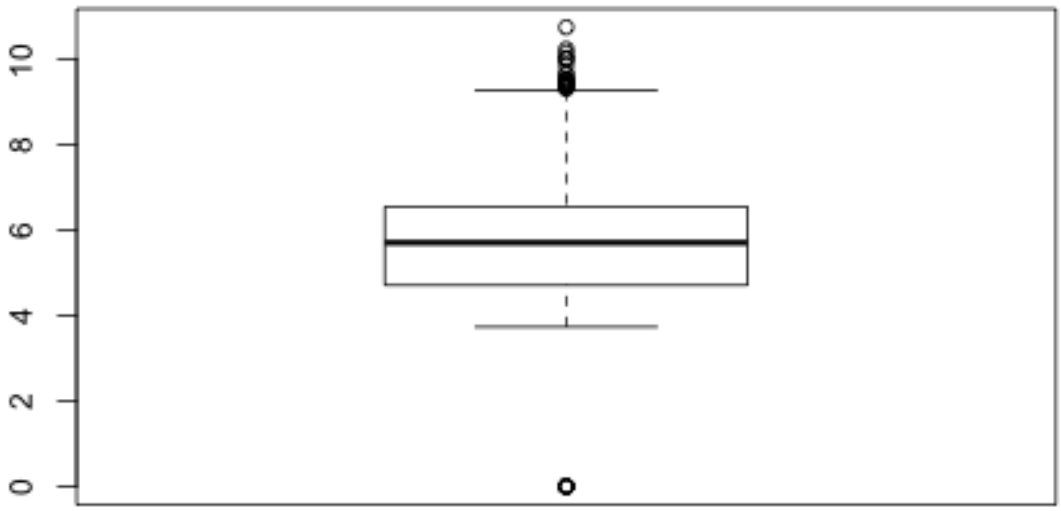
5.4 Visualisierung mit dem Standardpaket

Es gibt in R mehrere grundlegend verschiedene Möblichkeiten, Daten zu visualiseren. Für einen schnellen Überblick sind z.B. `hist()` und `boxplot()` hilfreich:

```
hist(diamonds$price)
```




```
boxplot(diamonds$x)
```



5.5 Visualisierung mit `ggplot()`

Das Paket `ggplot2` ist Teil vom `tidyverse`. Hiermit lassen sich sehr flexible Graphiken gestalten. Wir werden ausschließlich mit diesem System arbeiten.

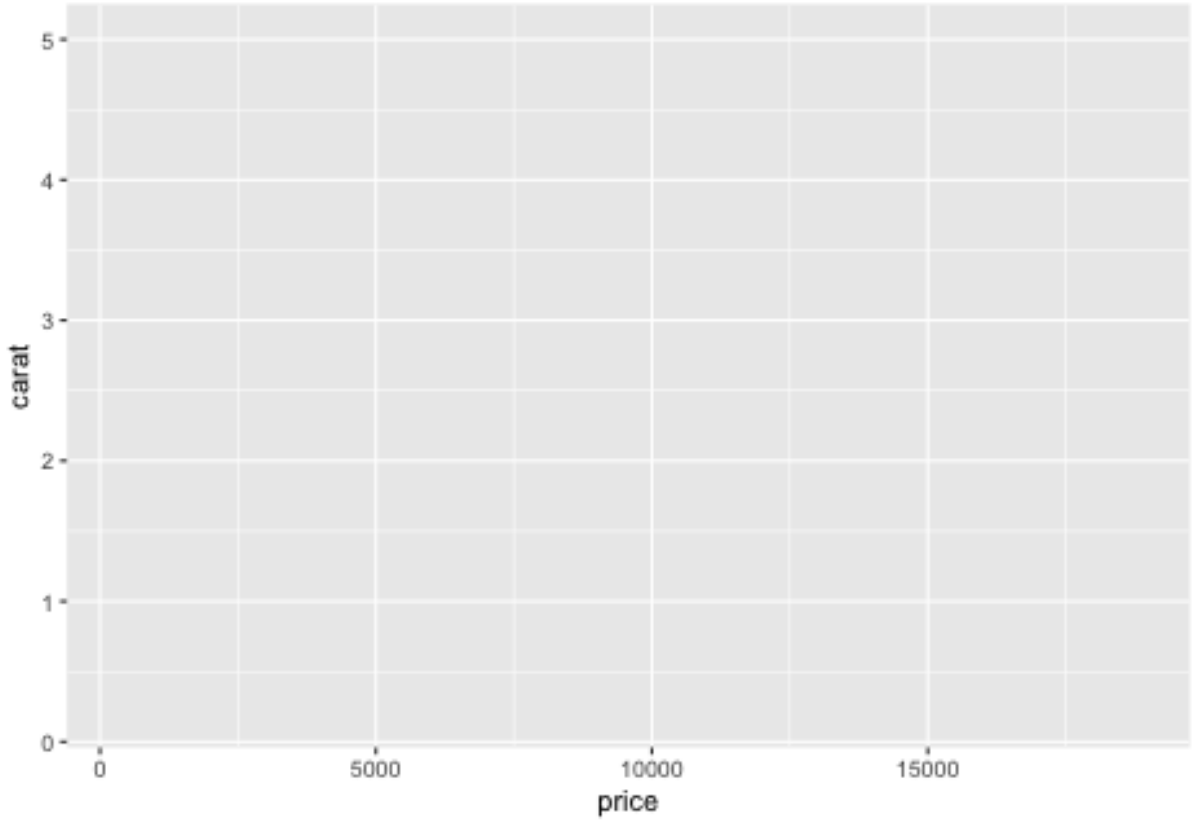
Die Syntax ist dabei auf den ersten Blick etwas komplexer.

Am Anfang steht der Befehl `ggplot(x)` mit dem Datensatz als Parameter

```
ggplot(data=diamonds)
```

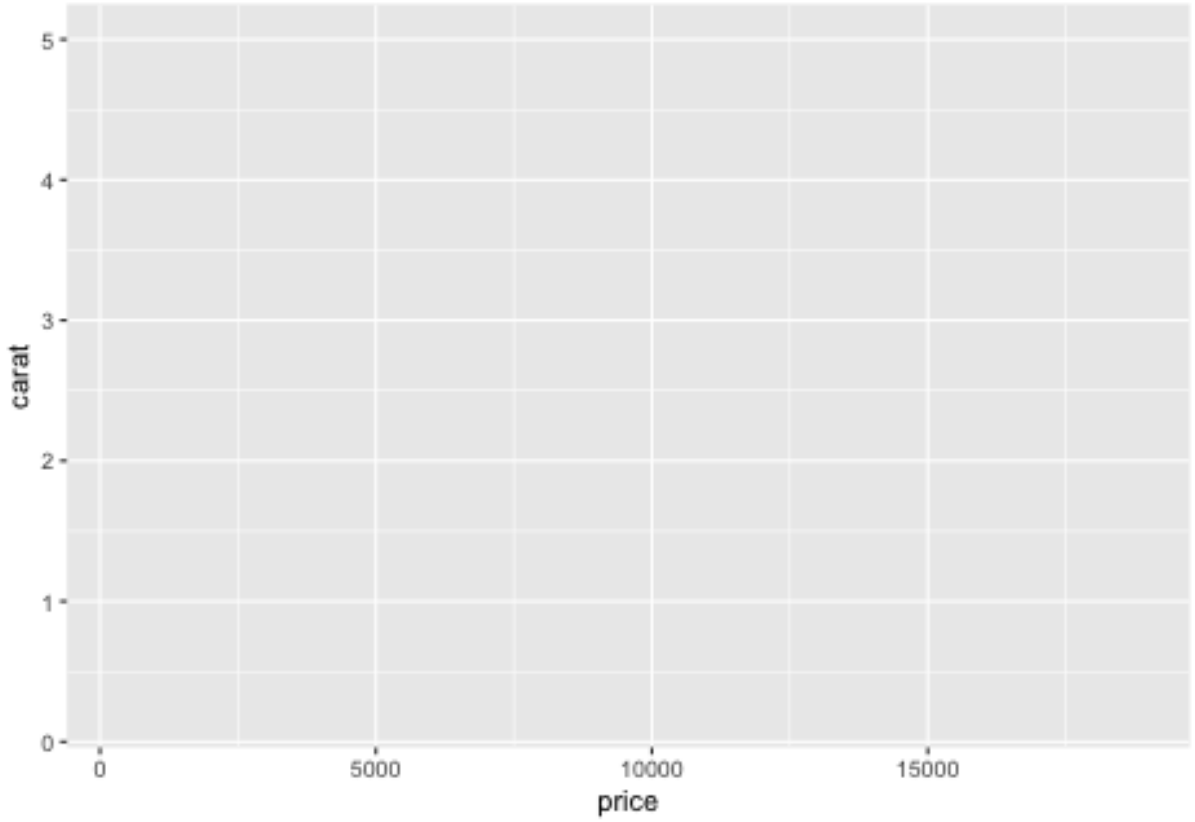
Mit einem Mapping-Parameter legen wir die Dimensionen fest:

```
ggplot(data=diamonds, mapping=aes(x=price, y=carat))
```



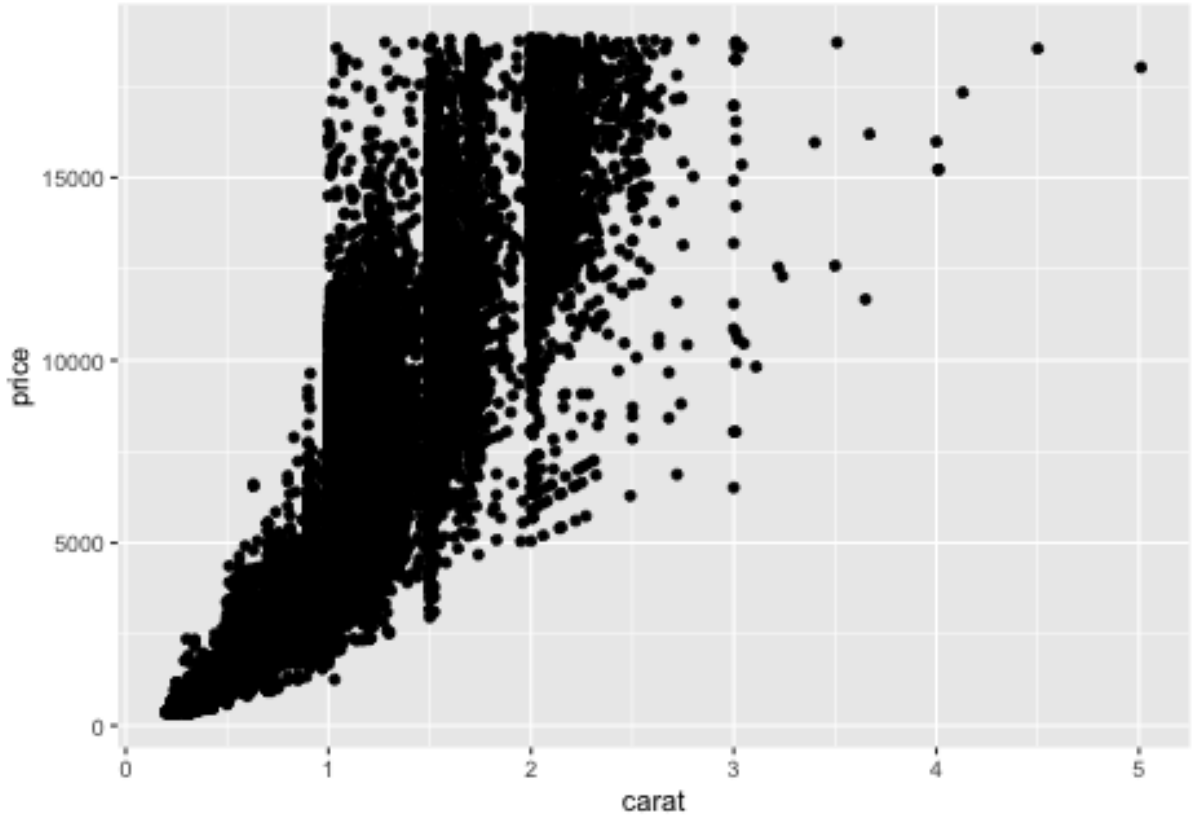
Das gleiche ohne Parameternamen:

```
ggplot(diamonds, aes(price, carat))
```



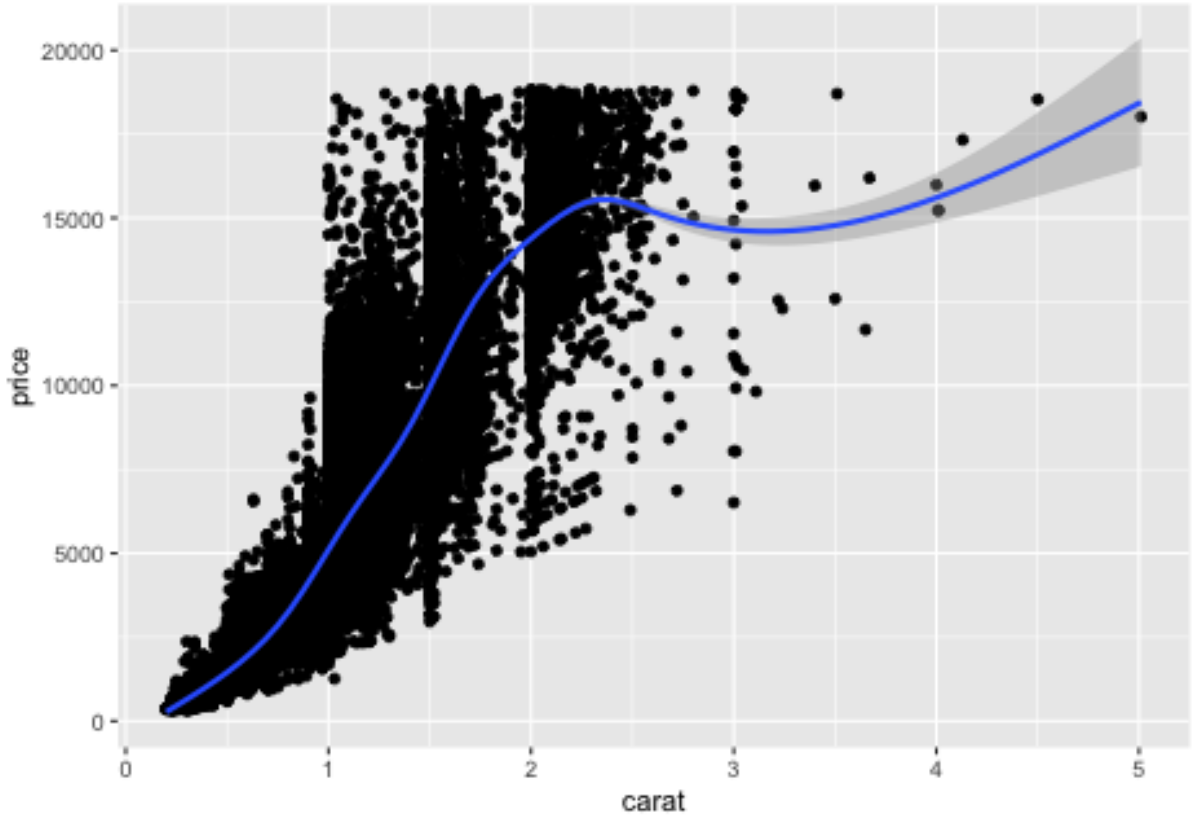
Nun kann mit dem +-Operator ein “geometrischer” Layer hinzugefügt werden:

```
ggplot(diamonds, aes(x=carat, y=price)) +
  geom_point()
```



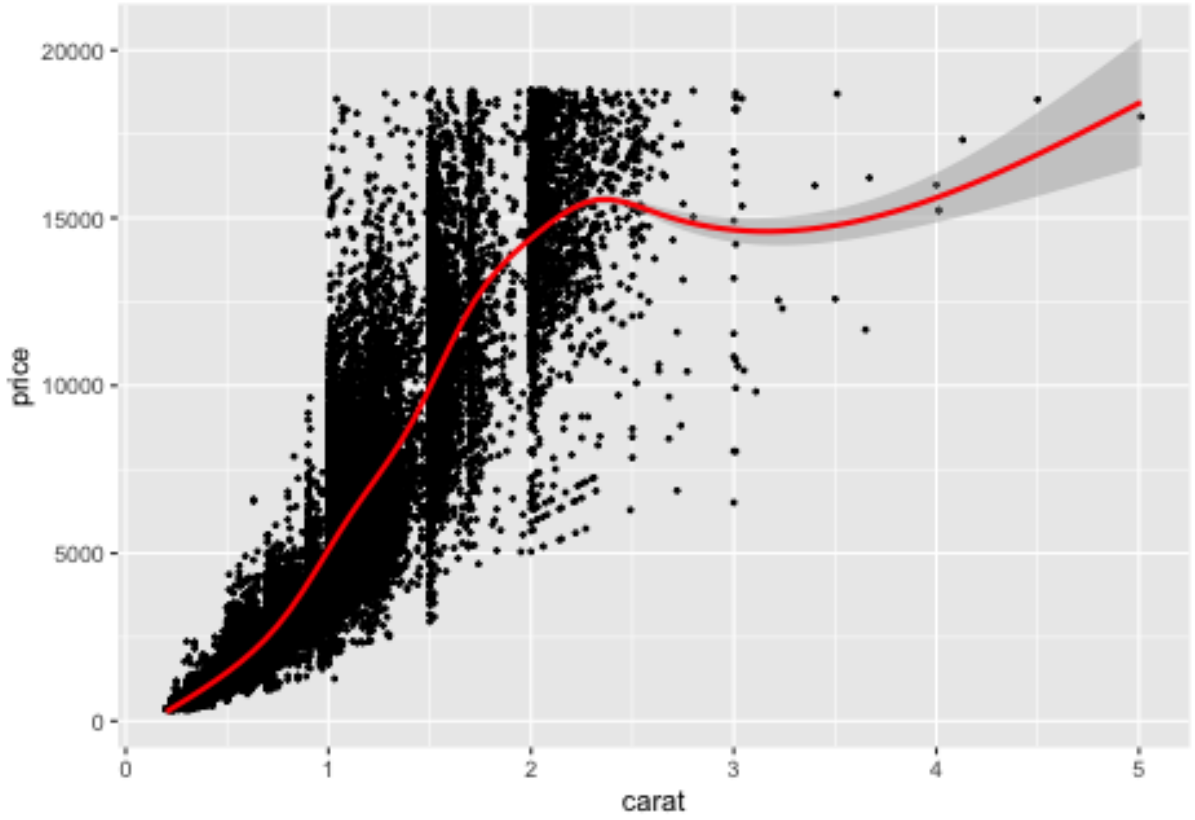
Weitere `geom`-Layer lassen sich mit dem +-Operator hinzufügen:

```
ggplot(diamonds, aes(x=carat, y=price)) +
  geom_point() +
  geom_smooth()
```



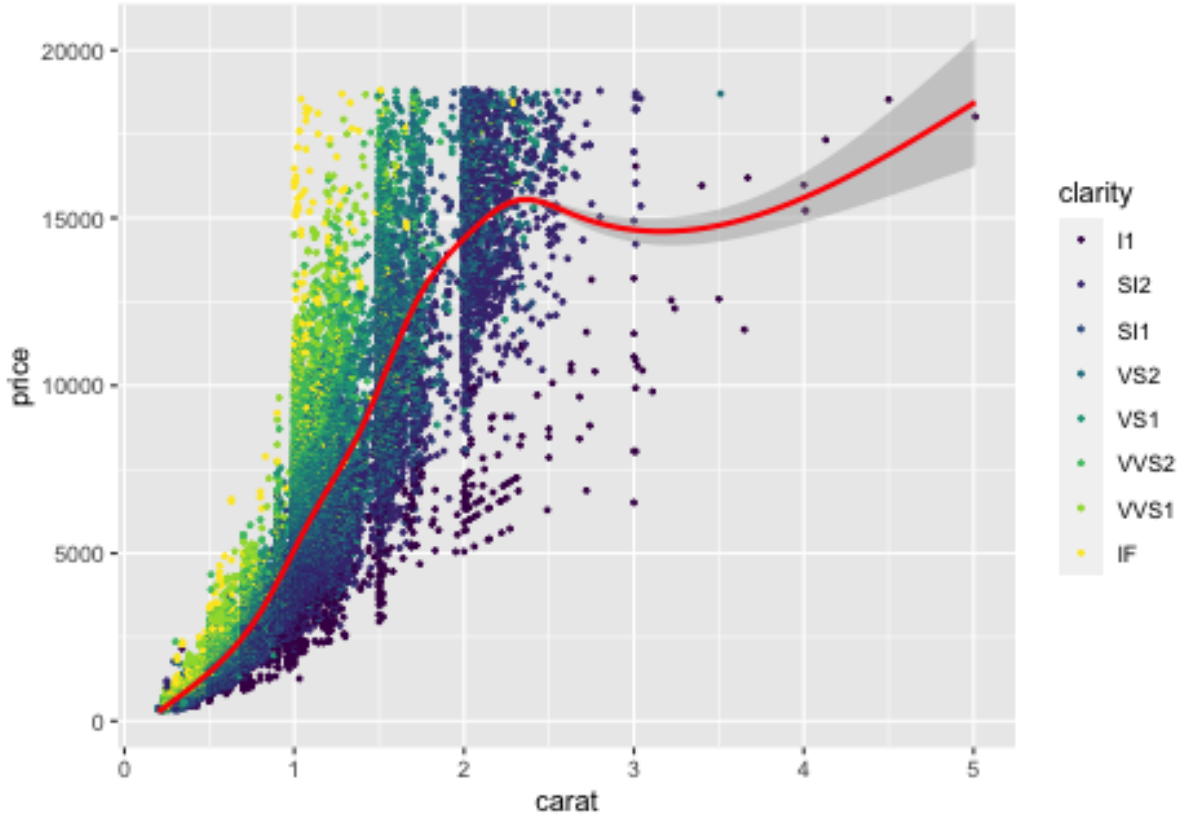
Die Layer-Funktionen können durch Parameter angepasst werden:

```
ggplot(diamonds, aes(x=carat, y=price)) +
  geom_point(size=0.5) +
  geom_smooth(color="red")
```



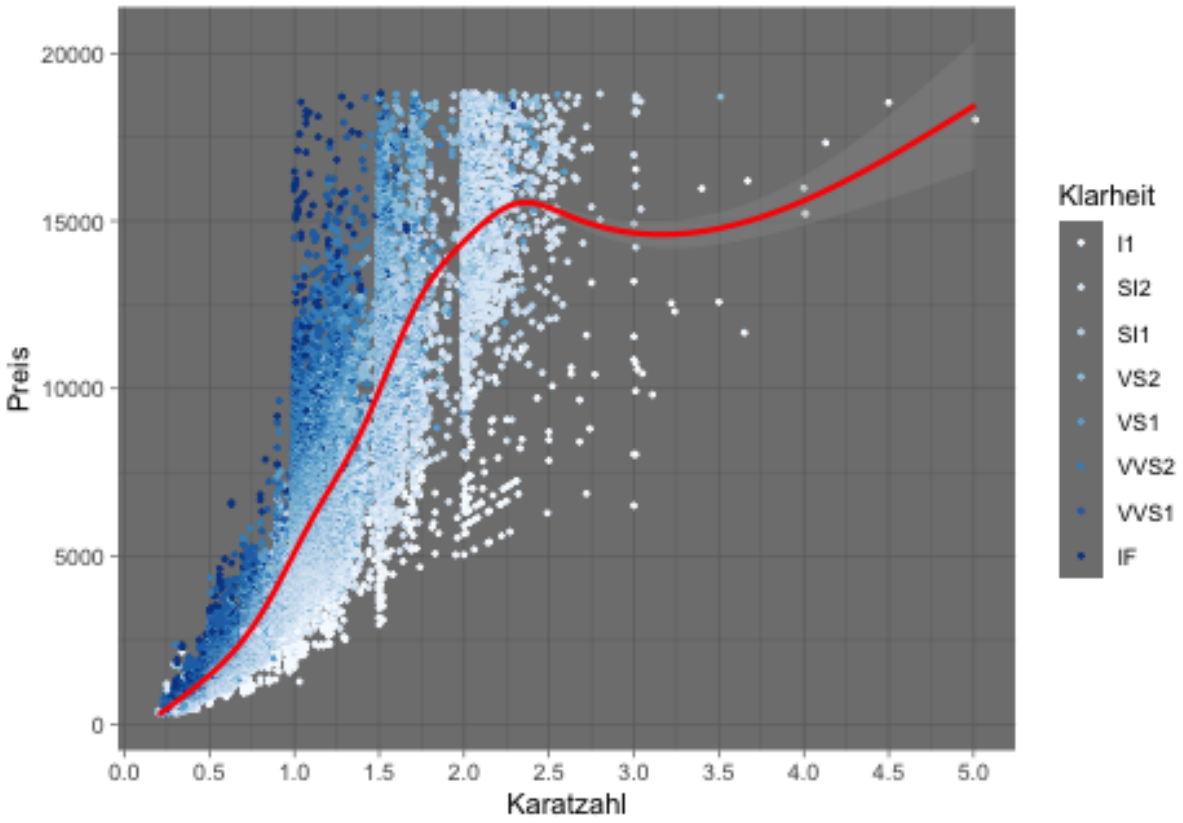
Dabei lassen sich in den einzelnen Layers mappings hinzufügen oder verändern:

```
ggplot(diamonds, aes(x=carat, y=price)) +
  geom_point(aes(color=clarity), size=0.5) +
  geom_smooth(color="red")
```



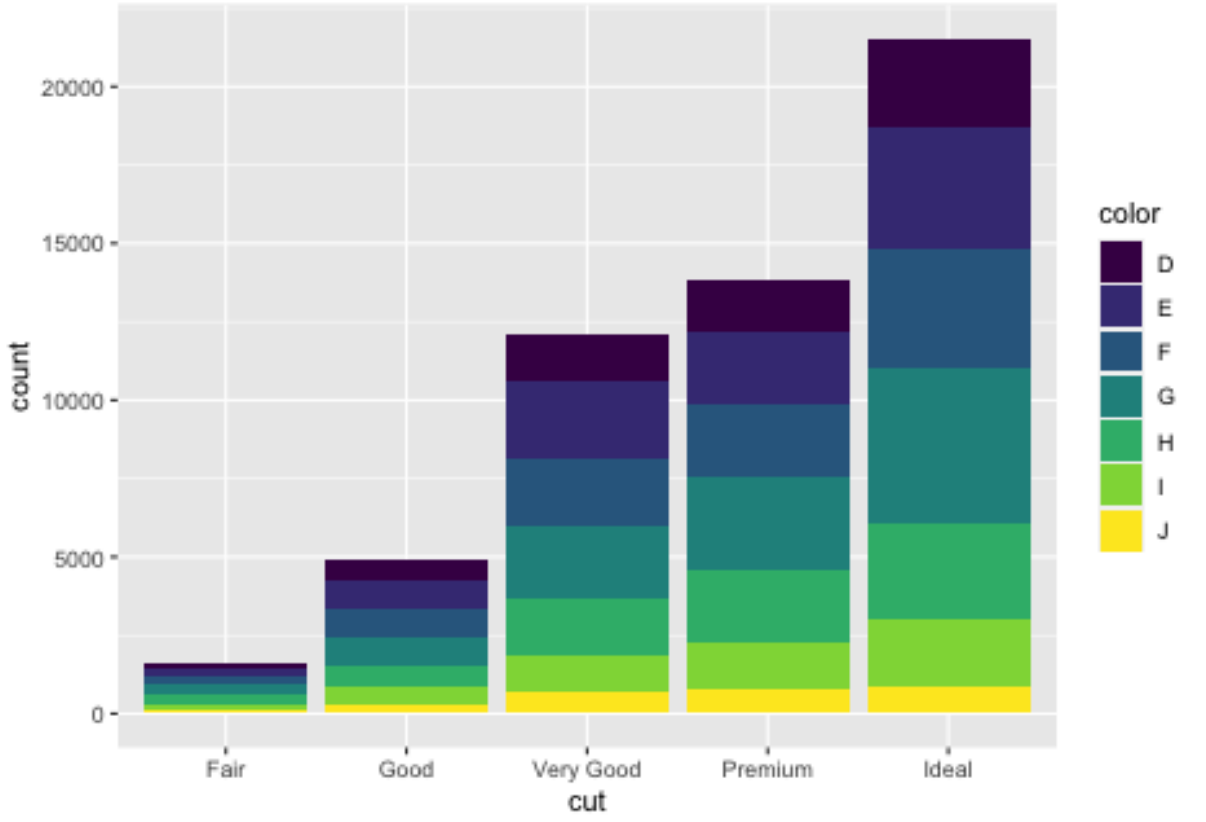
Schließlich lassen sich noch viele weitere optische Aspekte anpassen, z.B. Achsen, Farben, etc.:

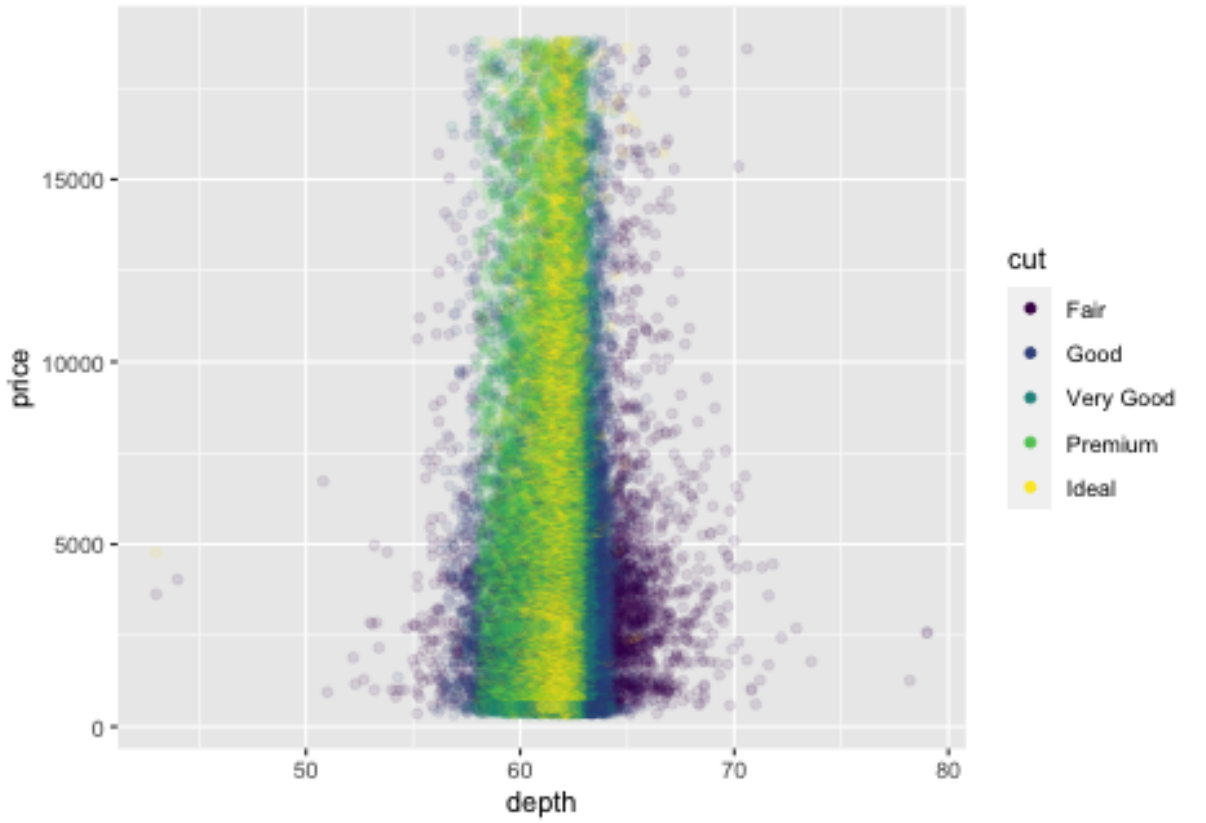
```
ggplot(diamonds, aes(x=carat, y=price)) +
  geom_point(aes(color=clarity), size=0.5) +
  geom_smooth(color="red") +
  scale_x_continuous("Karatzahl", breaks=seq(0,5,0.5)) +
  scale_y_continuous("Preis") +
  scale_color_brewer("Klarheit") +
  theme_dark()
```

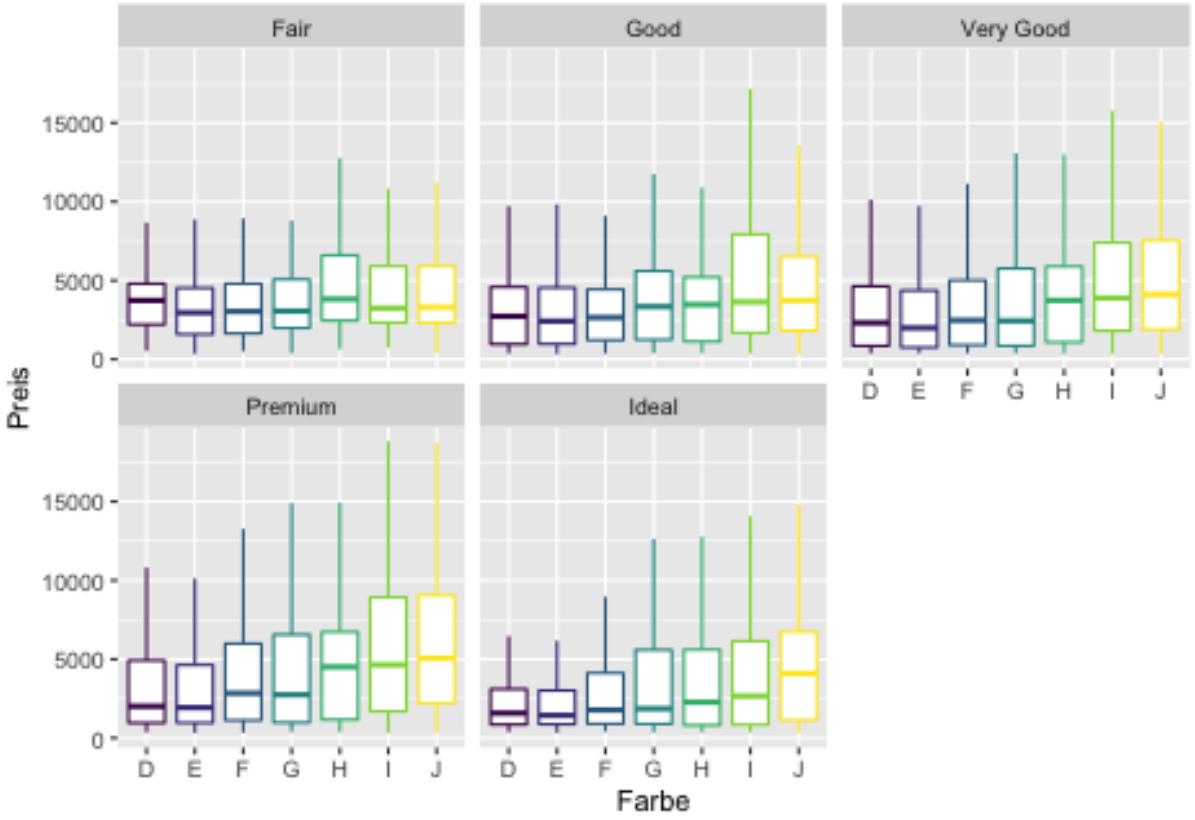


5.6 Aufgaben

Versuchen Sie, folgende Visualisierungen des Datensatzes `diamonds` auszugeben:


20000
15000
10000
5000
0
count
Fair
Good
Very Good
Premium
Ideal
cut
color
D
E
F
G
H
I
J


15000
10000
5000
0
price
50
60
70
80
depth
cut
Fair
Good
Very Good
Premium
Ideal



5.6.1 R for Data Science

Schauen Sie sich die Publikation R for Data Science an.

Was ist das für ein Buch? Wer ist das Zielpublikum?

Lesen Sie das Kapitel “3: Data Visualization” und vollziehen Sie die Visualisierungen nach.

Bearbeiten Sie die Aufgaben.

Bearbeiten Sie die RStudio Primers zu Datenvisualisierung.

6 Text: Shelton et al. 2014

6.1 Lesetext

Shelton, Taylor, Ate Poorthuis, Mark Graham und Matthew Zook. 2014. Mapping the Data Shadows of Hurricane Sandy. Uncovering the Sociospatial Dimensions of ‘Big Data’. *Geoforum 52.* 167–79.

6.2 Fragen an den Text

1. Um welche Art von Text handelt es sich? Wer sind die Autoren, und an wen wenden sie sich?
2. Was war Hurricane Sandy, von dem der Text erzählt? Warum wird ausgerechnet dieser Hurricane herangezogen?
3. Welche Methoden wenden die Autoren an, und zu welchen Ergebnissen kommen sie?

4. Im Abstract versprechen die Autoren: “We also seek to fill a conceptual lacuna…” Was heißt das, und wie geht der Text das an?
5. In welchen Punkten finden Sie den Text überzeugend? Welche Kritik haben Sie am Text?

7 Geodaten

7.1 Lernziele dieser Sitzung

Sie können…

- Pipes benutzen
- einfache `dplyr`-Befehle ausführen
- Koordinaten visualisieren

7.2 Voraussetzungen

Wir laden erstmal `tidyverse`:

```
library(tidyverse)
```

7.3 Exkurs: Pipes

Teil vom `tidyverse` ist auch das Paket `magrittr`, das einen besonderen Operator enthält: %>%

Der Operator %>% heißt “Pipe” und setzt das Ergebnis der vorherigen Funktion als ersten Parameter in die nächste Funktion ein. Zur Veranschaulichung:

```
anzahl_buchstaben <- length(letters)
sqrt(anzahl_buchstaben)
```

…ist das gleiche wie…

```
sqrt(length(letters))
```

…ist das gleiche wie…

```
length(letters) %>%
  sqrt()
```

…ist das gleiche wie…

```
letters %>%
  length %>%
  sqrt()
```

So können beliebig viele Funktionen aneinandergereiht werden. Und mit -> kann eine Variable „in die andere Richtung“ zugewiesen werden

```
letters %>%
  length() %>%
  sqrt() %>%
  round() %>%
  as.character() ->
  my_var
```

Gerade bei komplizierteren Zusammenhängen wird der Code so oft lesbarer, weil die Logik von links nach rechts, bzw. von oben nach unten gelesen werden kann.

7.4 Daten importieren

Beim Open-Data-Portal der Stadt Frankfurt steht ein Baumkataster zur Verfügung.

Die Datei im CSV-Format (comma separated values) kann entweder heruntergeladen und durch klicken importiert werden, oder direkt über den Befehl:

```
baumkataster <- read_csv2("http://offenedaten.frankfurt.de/dataset/73c5a6b3-c033-4dad-bb7d-8
```

7.5 Überblick verschaffen

Mit `summary()` lässt sich eine Zusammenfassung der Werte generieren:

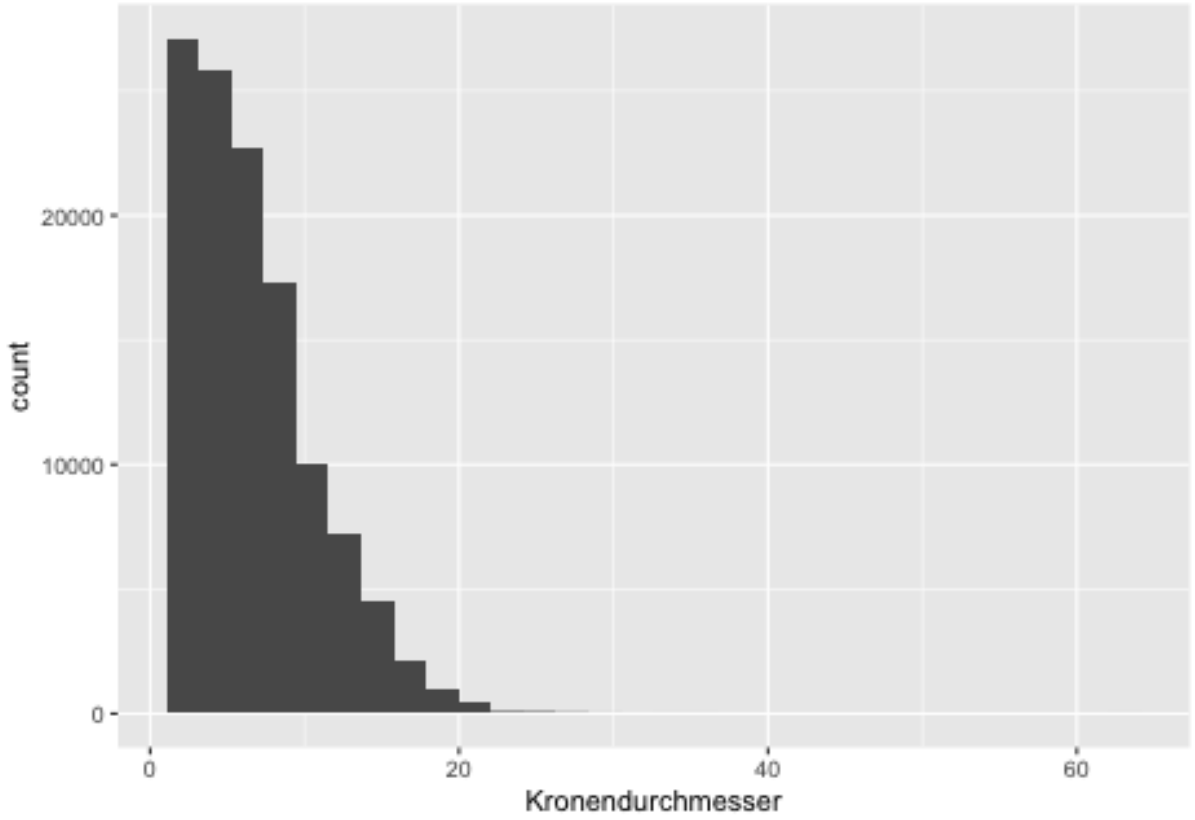
```
summary(baumkataster)
##  Gattung/Art/Deutscher Name   Baumnummer
##  Length:118403              Min.   :    1.0
##  Class :character           1st Qu.:   24.0
##  Mode  :character           Median :   82.0
##                             Mean   :  232.7
##                             3rd Qu.:  270.0
##                             Max.   :20158.0
##                             NA's   :1853
##     Objekt            Pflanzjahr   Kronendurchmesser
##  Length:118403      Min.   :1645   Min.   : 2.000
##  Class :character   1st Qu.:1970   1st Qu.: 4.000
##  Mode  :character   Median :1982   Median : 6.000
##                     Mean   :1979   Mean   : 6.688
##                     3rd Qu.:1995   3rd Qu.: 9.000
##                     Max.   :2017   Max.   :63.000
##
##     HOCHWERT         RECHTSWERT
##  Min.   :5545117   Min.   :463163
##  1st Qu.:5550428   1st Qu.:472715
##  Median :5552601   Median :475219
##  Mean   :5552953   Mean   :475244
##  3rd Qu.:5555165   3rd Qu.:478201
##  Max.   :5563639   Max.   :485361
##
```

Genauere Infos über diese Merkmale gibt es auf dem Datenportal.

7.6 Visualisieren

Wie in der letzten Lektion besprochen, lässt sich der Datensatz mit `ggplot()` visualisieren, z.B.:

```
ggplot(baumkataster, aes(x=Kronendurchmesser)) +
  geom_histogram()
```



Eine neue Messreihe lässt sich z.B. so errechnen:

```
alter <- 2020 - baumkataster$Pflanzjahr
head(alter)
## [1] 100 100 100 100 100 100
```

Der Befehl `mutate()` funktioniert sehr ähnlich, gibt aber den veränderten Datensatz zurück:

```
mutate(baumkataster, alter = 2020 - Pflanzjahr)
## # A tibble: 118,403 x 8
##    `Gattung/Art/De~ Baumnummer Objekt Pflanzjahr
##    <chr>                <dbl> <chr>       <dbl>
##  1 Platanus x hisp~          1 Acker~       1920
##  2 Platanus x hisp~          2 Acker~       1920
##  3 Platanus x hisp~          3 Acker~       1920
##  4 Platanus x hisp~          4 Acker~       1920
##  5 Platanus x hisp~          5 Acker~       1920
##  6 Platanus x hisp~          6 Acker~       1920
##  7 Platanus x hisp~          7 Acker~       1920
##  8 Platanus x hisp~          8 Acker~       1920
##  9 Platanus x hisp~          9 Acker~       1920
## 10 Platanus x hisp~         10 Acker~       1920
## # ... with 118,393 more rows, and 4 more variables:
## #   Kronendurchmesser <dbl>, HOCHWERT <dbl>,
## #   RECHTSWERT <dbl>, alter <dbl>
```

Derselbe Befehl mit dem Pipe-Operator:

```
baumkataster %>%
  mutate(alter = 2020 - Pflanzjahr)
## # A tibble: 118,403 x 8
##    `Gattung/Art/De~ Baumnummer Objekt Pflanzjahr
##    <chr>                <dbl> <chr>       <dbl>
##  1 Platanus x hisp~         1 Acker~       1920
##  2 Platanus x hisp~         2 Acker~       1920
##  3 Platanus x hisp~         3 Acker~       1920
##  4 Platanus x hisp~         4 Acker~       1920
##  5 Platanus x hisp~         5 Acker~       1920
##  6 Platanus x hisp~         6 Acker~       1920
##  7 Platanus x hisp~         7 Acker~       1920
##  8 Platanus x hisp~         8 Acker~       1920
##  9 Platanus x hisp~         9 Acker~       1920
## 10 Platanus x hisp~        10 Acker~       1920
## # ... with 118,393 more rows, and 4 more variables:
## #   Kronendurchmesser <dbl>, HOCHWERT <dbl>,
## #   RECHTSWERT <dbl>, alter <dbl>
```

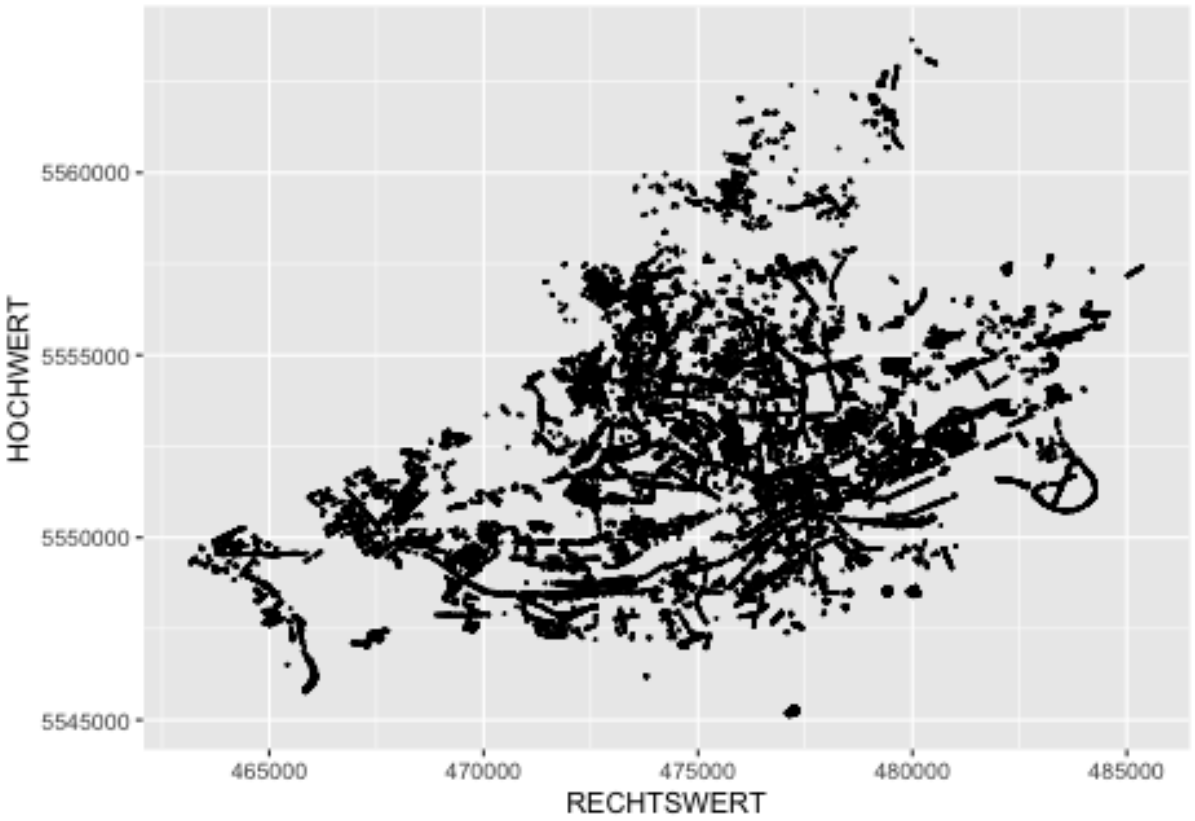
So lassen sich auch hier verschiedene Befehle verknüpfen. `filter()` beschränkt den Datensatz auf Merkmalsträger, die den Kriterien entsprechen:

```
baumkataster %>%
  mutate(alter = 2020 - Pflanzjahr) %>%
  filter(alter > 30) ->
  alte_baeume

summary(alte_baeume)
##  Gattung/Art/Deutscher Name   Baumnummer
##  Length:73859               Min.   :    1.0
##  Class :character           1st Qu.:   29.0
##  Mode  :character           Median :   97.0
##                             Mean   :  263.2
##                             3rd Qu.:  314.0
##                             Max.   :10489.0
##                             NA's   :684
##     Objekt            Pflanzjahr   Kronendurchmesser
##  Length:73859       Min.   :1645   Min.   : 2.000
##  Class :character   1st Qu.:1960   1st Qu.: 6.000
##  Mode  :character   Median :1974   Median : 8.000
##                     Mean   :1966   Mean   : 8.503
##                     3rd Qu.:1980   3rd Qu.:10.000
##                     Max.   :1989   Max.   :35.000
##
##     HOCHWERT         RECHTSWERT         alter
##  Min.   :5545117   Min.   :463163   Min.   : 31.00
##  1st Qu.:5550415   1st Qu.:472667   1st Qu.: 40.00
##  Median :5552480   Median :475708   Median : 46.00
```

```
##  Mean   :5552593   Mean   :475402   Mean   : 53.54
##  3rd Qu.:5554589   3rd Qu.:478539   3rd Qu.: 60.00
##  Max.   :5563639   Max.   :485360   Max.   :375.00
##
```

Schließlich ergibt das Streudiagramm von Koordinaten so eine art Karte:

```
ggplot(alte_baeume) +
    geom_point(size = 0.1, aes(x = RECHTSWERT, y = HOCHWERT))
```



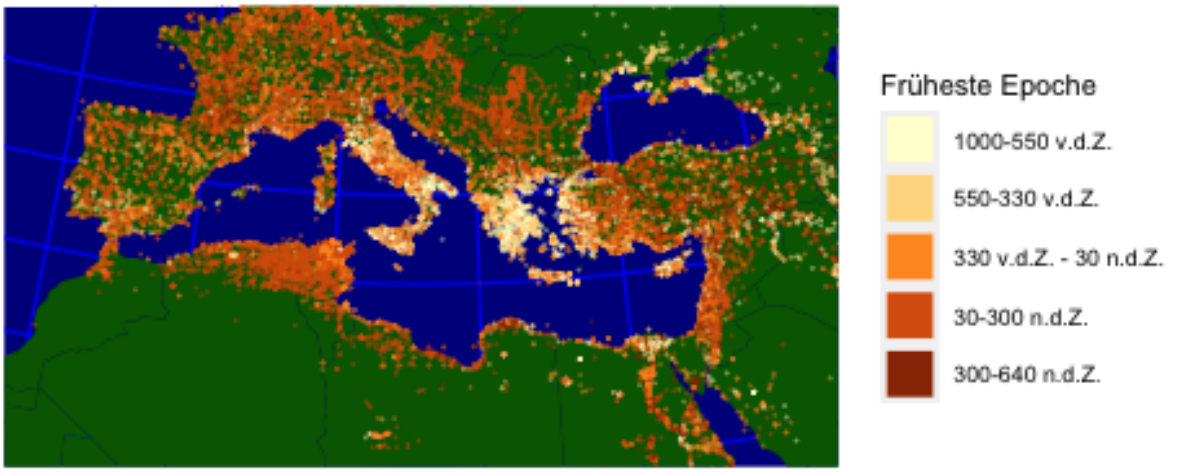
Diesen Ansatz werden wir in der nächsten Lektion vertiefen.

7.7 Aufgaben

1. Besuchen Sie https://pleiades.stoa.org/ - worum geht es hier?
2. Finden Sie den kompletten aktuellen Datensatz für „locations“ als CSV-Datei.
3. Importieren Sie ihn in R und weisen Sie dem Datensatz den Namen `pleiades` zu.
4. Finden Sie geeignete Werte für (einzelne) Längen- und Breitengrade im Datensatz.
5. Plotten Sie die Koordinaten auf x- und y-Achse mit `ggplot()`. Was erkennen Sie?
6. Halbieren Sie die Größe und setzen Sie den Alpha-Wert der Punkte auf 0,2.
7. Bringen Sie die Grafik in die Mercator-Projektion.
8. Schauen Sie sich diesen Befehl an:

```
map_data("world") %>%
  ggplot() +
    geom_polygon(mapping = aes(x = long,
                               y = lat,
                               group = group)) +
    coord_quickmap(xlim = c(-8, 40),
                   ylim = c(26, 48))
```

9. Versuchen Sie, jede einzelne Zeile nachzuvollziehen, indem Sie die entsprechenden Funktionen recherchieren.
10. Führen Sie den Befehl aus.
11. Ändern Sie die Farbe der Flächen in hellgrau.
12. Wählen Sie einen Kartenausschnitt, auf dem Portugal, Ägypten, Irak und Frankreich komplett zu sehen sind.
13. Plotten Sie auf diesem Hintergrund den Datensatz `pleiades`. Passen Sie dabei die Parameter so an, dass es Ihnen optisch zusagt.
14. Wählen Sie für die Karte die Bonnesche Projektion mit Standardparallele bei 40°N.
15. Entfernen Sie alle Achsenbeschriftungen.
16. (Achtung: extrem knifflig!) Bilden Sie diese Grafik nach, die die Orte geordnet nach ältestem Fund darstellt:



8 Choroplethen

8.1 Lernziele

Sie können…

- Geodaten als Simple Features importieren,
- CRS bestimmen und umwandeln,
- einfache Verschneidungen von Simple Features durchführen und
- Simple Features kartographisch darstellen.

8.2 Vorbereitung

Für diese Lektion werden zwei Pakete geladen:

```
library(tidyverse)
library(sf)
```

8.3 Ziel

Ziel ist, eine Choroplethenkarte von Frankfurt zu erstellen, die die Versorgung mit Kiosken darstellt.

8.4 Grundkarte

Eine Shapefile der Frankfurter Stadtteile findet sich hier: http://www.offenedaten.frankfurt.de/dataset/frankfurter-stadtteilgrenzen-fur-gis-systeme

Wir laden die Zip-Datei herunter und speichern den enthaltenen Ordner `stadtteile` in unserem Arbeitsverzeichnis. Es ist eine gute Angewohnheit, einen Unterordner für Ressourcen anzulegen.

Dann importieren wir den Geodatensatz als Simple Features (Paket `sf`):

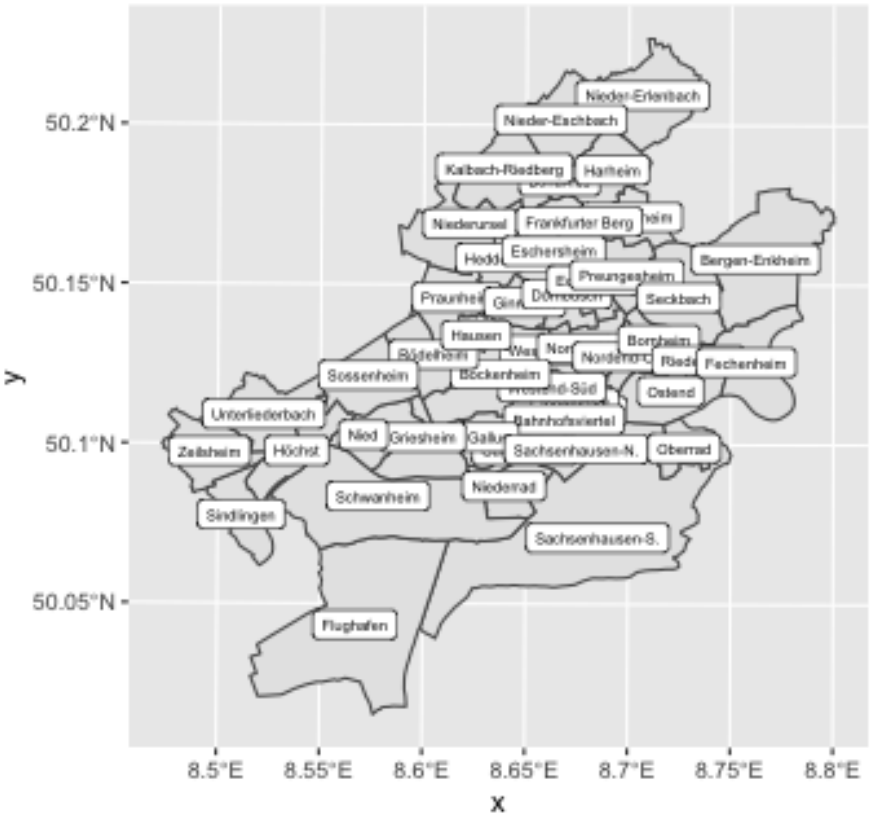
```
stadtteile <- st_read("resources/stadtteile/Stadtteile_Frankfurt_am_Main.shp")
## Reading layer `Stadtteile_Frankfurt_am_Main' from data source `/Users/till/teaching/2020x
## Simple feature collection with 46 features and 2 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: 462292.7 ymin: 5540412 xmax: 485744.8 ymax: 5563925
## projected CRS:  ETRS89 / UTM zone 32N
```

Simple Features sind Datensätze, die eine Spalte `geometry` enthalten, in der Geodaten in einem standardisierten Format hinterlegt sind.

```
str(stadtteile)
## Classes 'sf' and 'data.frame':   46 obs. of  3 variables:
##  $ STTLNR  : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ STTLNAME: Factor w/ 46 levels "Altstadt","Bahnhofsviertel",..: 1 22 2 45 44 30 29 32 7
##  $ geometry:sfc_POLYGON of length 46; first list element: List of 1
##   ..$ : num [1:46, 1:2] 476934 476890 476852 476813 476799 ...
##   ..- attr(*, "class")= chr  "XY" "POLYGON" "sfg"
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA
##   ..- attr(*, "names")= chr  "STTLNR" "STTLNAME"
```

Eine Vorschau:

```
ggplot(stadtteile) +
  geom_sf() +
  geom_sf_label(aes(label = STTLNAME), size = 2)
```



8.5 OSM-Daten

Im OSM Wiki suchen wir den richtigen *tag* heraus. In diesem Fall `shop=kiosk`

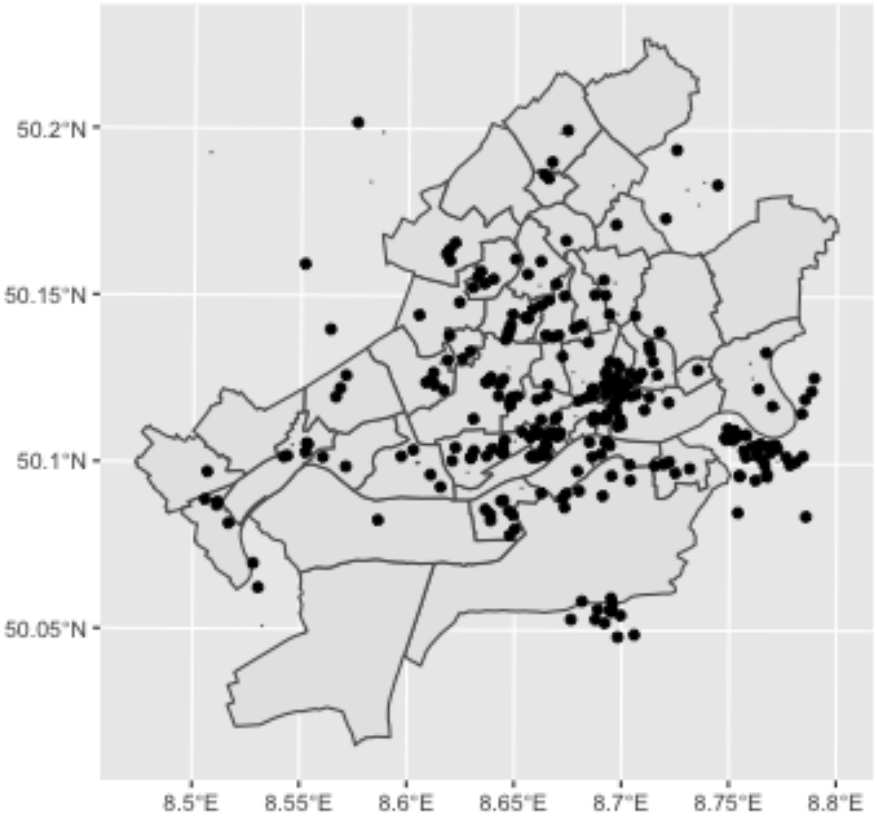
Dann bauen wir auf Overpass Turbo die Abfrage und laden den Datensatz herunter.

Schließlich importieren wir:

```
kioske <- st_read("resources/kioske.geojson")
## Reading layer `kioske' from data source `/Users/till/teaching/2020x21_Data_Science/public
## Simple feature collection with 325 features and 74 fields
## geometry type:  GEOMETRY
## dimension:      XY
## bbox:           xmin: 8.505468 ymin: 50.04801 xmax: 8.789538 ymax: 50.20185
## geographic CRS: WGS 84
```

Eine Vorschau:

```
ggplot() +
  geom_sf(data = stadtteile) +
  geom_sf(data = kioske)
```



8.6 Koordinatenreferenzsysteme

Der OSM-Datensatz ist mit WGS84 (EPSG 4326) referenziert:

```
st_crs(kioske)
## Coordinate Reference System:
##   User input: WGS 84
##   wkt:
## GEOGCRS["WGS 84",
##     DATUM["World Geodetic System 1984",
##         ELLIPSOID["WGS 84",6378137,298.257223563,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["geodetic latitude (Lat)",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["geodetic longitude (Lon)",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     ID["EPSG",4326]]
```

Die Stadtteilen hingegen sind sind in ETSR89 (EPSG 25832):

```
st_crs(stadtteile)
## Coordinate Reference System:
##   User input: ETRS89 / UTM zone 32N
##   wkt:
## PROJCRS["ETRS89 / UTM zone 32N",
##     BASEGEOGCRS["ETRS89",
##         DATUM["European Terrestrial Reference System 1989",
##             ELLIPSOID["GRS 1980",6378137,298.257222101,
##                 LENGTHUNIT["metre",1]]],
##         PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433]],
##         ID["EPSG",4258]],
##     CONVERSION["UTM zone 32N",
##         METHOD["Transverse Mercator",
##             ID["EPSG",9807]],
##         PARAMETER["Latitude of natural origin",0,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8801]],
##         PARAMETER["Longitude of natural origin",9,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8802]],
##         PARAMETER["Scale factor at natural origin",0.9996,
##             SCALEUNIT["unity",1],
##             ID["EPSG",8805]],
##         PARAMETER["False easting",500000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8806]],
##         PARAMETER["False northing",0,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8807]]],
##     CS[Cartesian,2],
##         AXIS["(E)",east,
##             ORDER[1],
##             LENGTHUNIT["metre",1]],
##         AXIS["(N)",north,
##             ORDER[2],
##             LENGTHUNIT["metre",1]],
##     USAGE[
##         SCOPE["Engineering survey, topographic mapping."],
##         AREA["Europe between 6°E and 12°E: Austria; Belgium; Denmark - onshore and offsho
##         BBOX[38.76,6,83.92,12]],
##     ID["EPSG",25832]]
```

Der Datensatz lässt sich allerdings transformieren:

```
stadtteile %>%
  st_transform(4326) %>%
  st_crs()
## Coordinate Reference System:
```

```
##   User input: EPSG:4326
##   wkt:
## GEOGCRS["WGS 84",
##     DATUM["World Geodetic System 1984",
##         ELLIPSOID["WGS 84",6378137,298.257223563,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["geodetic latitude (Lat)",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["geodetic longitude (Lon)",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     USAGE[
##         SCOPE["Horizontal component of 3D system."],
##         AREA["World."],
##         BBOX[-90,-180,90,180]],
##     ID["EPSG",4326]]
```

Jetzt haben beide Datensätze den selben EPSG-Code. Das ist die Voraussetzung für den nächsten Schritt.

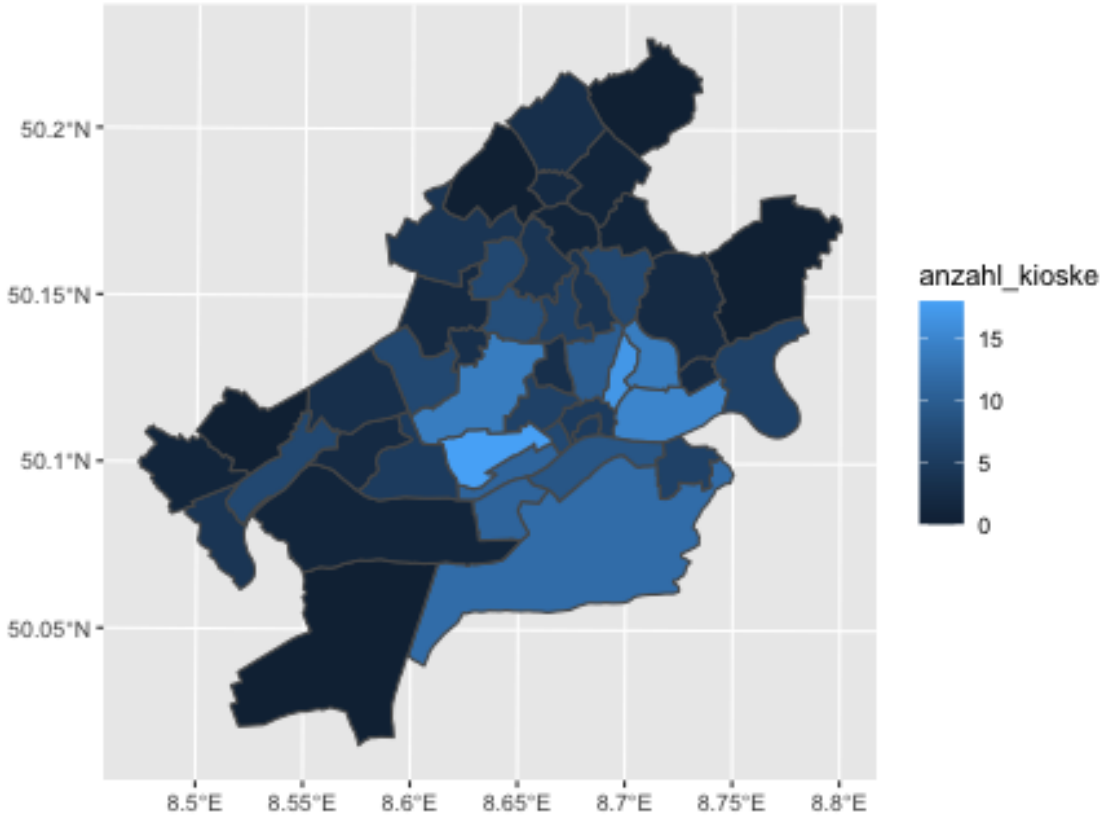
8.7 Verschneiden

Mit `st_covers()` und `lengths()` lassen sich die Anzahl der Kioske in jedem Stadtteil zählen und einer neuen Spalte im Originaldatensatz zuordnen:

```
stadtteile %>%
  st_transform(4326) %>%
  st_covers(kioske) %>%
  lengths() -> stadtteile$anzahl_kioske
```

Auf einer Karte veranschaulicht:

```
ggplot(stadtteile) +
  geom_sf(aes(fill = anzahl_kioske))
```



Allerdings wäre es schöner, die Kioskdichte (nach Fläche) darzustellen. Dazu berechnen wir zunächst die Flächen der Stadtteile:

```
st_area(stadtteile) %>%
  as.numeric() / 1000 / 1000 ->
  stadtteile$qkm
```

Oder mit Pipes:

```
stadtteile %>%
  mutate(qkm = st_area(.) %>% as.numeric() / 1000 / 1000)
## Simple feature collection with 46 features and 4 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: 462292.7 ymin: 5540412 xmax: 485744.8 ymax: 5563925
## projected CRS:  ETRS89 / UTM zone 32N
## First 10 features:
##    STTLNR        STTLNAME
## 1       1        Altstadt
## 2       2      Innenstadt
## 3       3 Bahnhofsviertel
## 4       4     Westend-Süd
## 5       5    Westend-Nord
## 6       6    Nordend-West
## 7       7     Nordend-Ost
## 8       8          Ostend
```

```
## 9       9        Bornheim
## 10     10  Gutleutviertel
##                          geometry anzahl_kioske
## 1  POLYGON ((476934.3 5550541,...             5
## 2  POLYGON ((477611.9 5552034,...             5
## 3  POLYGON ((475831 5550785, 4...             7
## 4  POLYGON ((475745.4 5552373,...             6
## 5  POLYGON ((476497.9 5553910,...             3
## 6  POLYGON ((478362.5 5553898,...            10
## 7  POLYGON ((478397.9 5551924,...            17
## 8  POLYGON ((481955.2 5552141,...            15
## 9  POLYGON ((478959.8 5552336,...            14
## 10 POLYGON ((472942 5548802, 4...            11
##          qkm
## 1  0.5065673
## 2  1.4902009
## 3  0.5425421
## 4  2.4948957
## 5  1.6307925
## 6  3.0977694
## 7  1.5305338
## 8  5.5573382
## 9  2.7840413
## 10 2.1982354
```

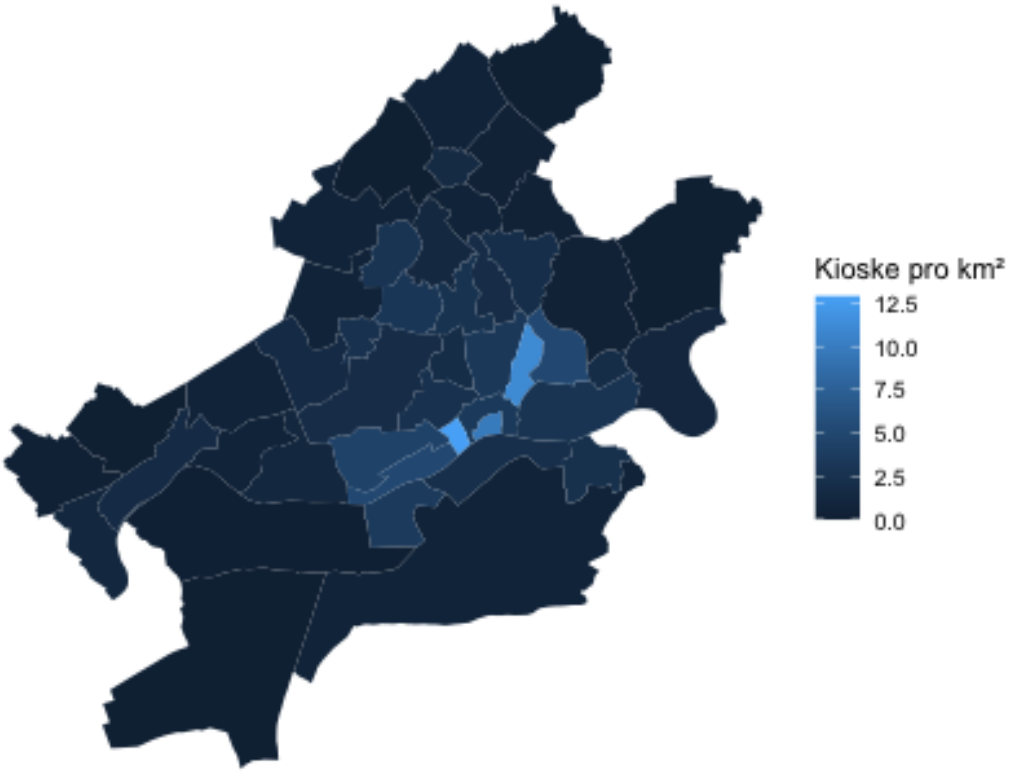
Und dann die Kioskdichte:

```
stadtteile %>%
  mutate(qkm = st_area(.) %>% as.numeric() / 1000 / 1000,
         kioskdichte = anzahl_kioske / qkm)
## Simple feature collection with 46 features and 5 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: 462292.7 ymin: 5540412 xmax: 485744.8 ymax: 5563925
## projected CRS:  ETRS89 / UTM zone 32N
## First 10 features:
##    STTLNR        STTLNAME
## 1       1        Altstadt
## 2       2      Innenstadt
## 3       3 Bahnhofsviertel
## 4       4     Westend-Süd
## 5       5    Westend-Nord
## 6       6    Nordend-West
## 7       7     Nordend-Ost
## 8       8          Ostend
## 9       9        Bornheim
## 10     10  Gutleutviertel
##                          geometry anzahl_kioske
## 1  POLYGON ((476934.3 5550541,...             5
```

```
## 2  POLYGON ((477611.9 5552034,...              5
## 3  POLYGON ((475831 5550785, 4...              7
## 4  POLYGON ((475745.4 5552373,...              6
## 5  POLYGON ((476497.9 5553910,...              3
## 6  POLYGON ((478362.5 5553898,...             10
## 7  POLYGON ((478397.9 5551924,...             17
## 8  POLYGON ((481955.2 5552141,...             15
## 9  POLYGON ((478959.8 5552336,...             14
## 10 POLYGON ((472942 5548802, 4...             11
##          qkm kioskdichte
## 1  0.5065673    9.870357
## 2  1.4902009    3.355252
## 3  0.5425421   12.902225
## 4  2.4948957    2.404910
## 5  1.6307925    1.839596
## 6  3.0977694    3.228129
## 7  1.5305338   11.107236
## 8  5.5573382    2.699134
## 9  2.7840413    5.028661
## 10 2.1982354    5.004014
```

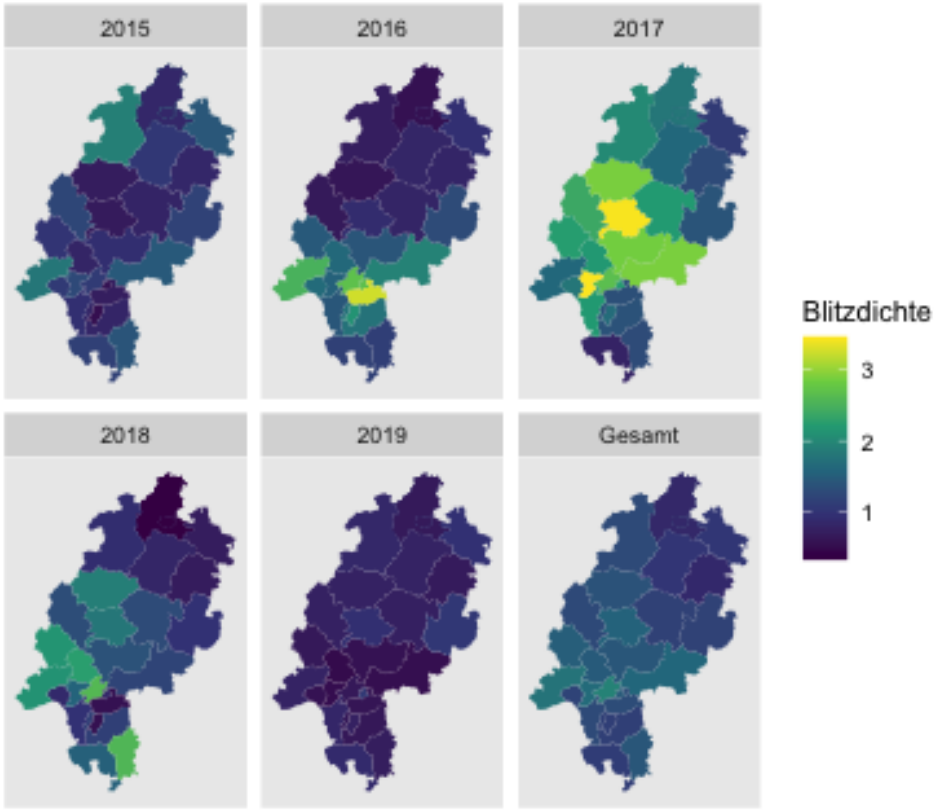
Schließlich die Karte:

```
stadtteile %>%
  mutate(qkm = st_area(.) %>% as.numeric() / 1000 / 1000,
         kioskdichte = anzahl_kioske / qkm) %>%
  ggplot() +
    geom_sf(aes(fill = kioskdichte), color=NA) +
    scale_fill_continuous("Kioske pro km²") +
    theme_void()
```



8.8 Aufgaben

1. Erstellen Sie eine Choroplethenkarte der Frankfurter Stadtteile, in der Sie die Anzahl bzw. die Dichte von Apotheken darstellen. (Schritte analog zu oben.)
2. Welche Stadtteile haben mehr Kioske? Welche mehr Apotheken? Wie ausgeprägt ist das Verhältnis? Erstellen Sie eine Karte, die das zum Ausdruck bringt.
3. (Achtung, knifflig!) Siemens veröffentlicht einen Blitzatlas. Laden Sie den Datensatz herunter und bauen Sie die folgende Ansicht nach:



9 Text: Chandra 2014

9.1 Lesetext

Chandra, Vikram. 2014. *Geek Sublime: The Beauty of Code, the Code of Beauty.* Graywolf Press, Minneapolis.

Daraus:

- Kapitel 1: Hello, World! (S. 1–8)
- Kapitel 3: The Language of Logic (S. 19–40)

9.2 Fragen an den Text

1. Um welche Art von Text handelt es sich? Wer ist der Autor, und an wen wendet er sich?
2. Was ist “literate programming”, und was könnte das in R konkret bedeuten?
3. Auf S. 37 ist die Rede von “our journey down the stack of languages”. Was ist damit gemeint?
4. Was findet der Autor an Code und Computern so faszinierend? Wo können Sie das nachvollziehen, und wo nicht?

9.3 Themenfindung

- Bereiten Sie *ein* Thema vor, zu dem Sie sich vorstellen könnten im Sommer zu arbeiten.
- Achten Sie darauf, dass das Thema nicht zu allgemein ist (“Finanzmarkt”) aber auch nicht zu speziell (“Zusammenhang zwischen Quadratmeterzahl und Mietpreis für Ladenflächen in Ginnheim”).

- Es wird in einem nächsten Schritt darum gehen, interessante Datenquellen zu finden (vielleicht haben Sie schon eine Idee?) und einer Fragestellung näher zu kommen.
- Sie werden die Themen kurz vorstellen um Überschneidungen zu identifizieren und ggf. Gruppen zu bilden (wenn gewünscht).

10 HTML-Tabellen

10.1 Lernziele dieser Sitzung

Sie können…

- sich den Quellcode einer Webseite anzeigen lassen und interpretieren.
- HTML-Tabellen als Datensatz einlesen.
- Fortgeschrittene Methoden der Datenbereinigung nachvollziehen.

10.2 Vorbereitung

Am Beispiel der Küstenlängen verschiedener Länder besprechen wir Techniken der Datenerhebung/-erfassung und -visualisierung. Unser Ziel ist es, die Daten zu den Küstenlängen in einer Grafik darzustellen.

Für die folgenden Aufgaben benötigen wir die Pakete `rvest` und `tidyverse`. Zunächst müssen diese installiert und in unsere Umgebung geladen werden.

```
library(tidyverse)
library(rvest)
```

10.3 Datenbeschaffung

Auf dem Internetauftritt der CIA gab es es eine Tabelle, welche die Küstenlänge (inklusive der Inseln) der einzelnen Länder enthält.

Über die Archivierungsplattform WayBackMachine ist die Seite immer noch abrufbar: https://web.archive.org/web/20190802010710/https://www.cia.gov/library/publications/the-world-factbook/fields/282.html

In einem ersten Schritt wird die URL der Tabelle der Variable `url` zugewiesen, sodass der Quellcode mit dem Befehl `read_html()` eingelesen werden kann.

```
url <- "https://web.archive.org/web/20190802010710/https://www.cia.gov/library/publications/
reply <- read_html(url)
```

Der Befehl `html_table()` ermöglicht das Auslesen *aller* Tabellen auf der Seite. Mithilfe des Befehls `str()` sehen wir, dass die Seite genau eine Tabelle enthält, welche die Informationen zu den Küstenlängen enthält.

```
tables <- html_table(reply, fill = TRUE)
str(tables)
## List of 1
##  $ :'data.frame':    266 obs. of  2 variables:
##   ..$ Country  : chr [1:266] "Afghanistan" "Akrotiri" "Albania" "Algeria" ...
##   ..$ Coastline: chr [1:266] "0 km\n          (landlocked)" "56.3 km" "362 km" "998 km" .
```

Durch die Umformung zu einem tibble erhalten wir eine Tabelle mit den gewünschten Informationen:

```
as_tibble(tables[[1]])
## # A tibble: 266 x 2
##    Country             Coastline
##    <chr>               <chr>
##  1 Afghanistan         "0 km\n          (landlocked)"
##  2 Akrotiri            "56.3 km"
##  3 Albania             "362 km"
##  4 Algeria             "998 km"
##  5 American Samoa      "116 km"
##  6 Andorra             "0 km\n          (landlocked)"
##  7 Angola              "1,600 km"
##  8 Anguilla            "61 km"
##  9 Antarctica          "17,968 km"
## 10 Antigua and Barbuda "153 km"
## # ... with 256 more rows
```

Mit pipes können wir die obigen Befehle zusammenfassen und somit das Ganze auf einmal ausführen.

```
"https://web.archive.org/web/20190802010710/https://www.cia.gov/library/publications/the-wor
  read_html() %>%
  html_table(fill = T) %>%
  .[[1]] %>%
  as_tibble() -> coast
```

10.4 Datenformatierung

Zur Datenformatierung nutzen wir Funktionen aus dem Paket `stringr`. Die Spalte mit der Küstenlänge soll keinen Text, keine Einheit direkt hinter den Zahlenwerten und keine Kommata zur Trennung der Zahlenwerte enthalten.

Der Befehl "str_extract()" sucht nach vorgegebenen Mustern (engl. *patterns*) und wählt diese aus. Diese patterns werden auch reguläre Ausdrücke (*regular expressions / regex*) genannt und sind eigentlich ein Thema für sich. Das Pattern `[0-9,.]+ km` extrahiert die Kilometerangaben.

```
km <- str_extract(coast$Coastline, "[0-9,.]+ km")
```

Die ausgewählten Muster (in unserem Fall Kommata und Text) können durch den Befehl `str_replace_all()` gelöscht oder ersetzt werden. Wir ersetzen alle Zeichen *außer* Zahlen und Dezimalpunkt mit einem leeren String, so dass sie verschwinden.

```
str_replace_all(km, "[^0-9.]", "")
##   [1] "0"       "56.3"    "362"     "998"     "116"
##   [6] "0"       "1600"    "61"      "17968"   "153"
##  [11] "45389"   "4989"    "0"       "68.5"    "74.1"
##  [16] "111866"  "25760"   "0"       "0"       "3542"
##  [21] "161"     "580"     "97"      "0"       "66.5"
##  [26] "386"     "121"     "103"     "0"       "0"
##  [31] "20"      "0"       "29.6"    "7491"    "698"
##  [36] "80"      "161"     "354"     "0"       "1930"
```

```
##  [41] "0"      "965"    "443"    "402"    "202080"
##  [46] "160"    "0"      "0"      "6435"   "14500"
##  [51] "138.9"  "11.1"   "26"     "3208"   "340"
##  [56] "37"     "169"    "120"    "3095"   "1290"
##  [61] "515"    "5835"   "3735"   "364"    "648"
##  [66] "0"      "7314"   "27.5"   "314"    "148"
##  [71] "1288"   "2237"   "2450"   "307"    "296"
##  [76] "2234"   "3794"   "0"      "0"      "65992.9"
##  [81] "1288"   "1117"   "1129"   "1250"   "4853"
##  [86] "2525"   "28"     "885"    "80"     "40"
##  [91] "310"    "2389"   "539"    "12"     "13676"
##  [96] "44087"  "121"    "125.5"  "400"    "50"
## [101] "320"    "350"    "459"    "1771"   "101.9"
## [106] "0"      "823"    "733"    "6.4"    "0"
## [111] "4970"   "7000"   "66526"  "54716"  "2440"
## [116] "58"     "1448"   "160"    "273"    "7600"
## [121] "1022"   "124.1"  "29751"  "8"      "70"
## [126] "34"     "26"     "0"      "536"    "3"
## [131] "1143"   "2495"   "2413"   "0"      "499"
## [136] "0"      "0"      "498"    "225"    "0"
## [141] "579"    "1770"   "0"      "90"     "0"
## [146] "41"     "4828"   "0"      "4675"   "644"
## [151] "0"      "196.8"  "370.4"  "754"    "177"
## [156] "9330"   "6112"   "15"     "0"      "4.1"
## [161] "0"      "293.5"  "40"     "1835"   "2470"
## [166] "1572"   "30"     "8"      "0"      "451"
## [171] "2254"   "15134"  "910"    "0"      "853"
## [176] "64"     "32"     "0"      "1482"   "25148"
## [181] "2092"   "135663" "1046"   "1519"   "14.5"
## [186] "2490"   "5152"   "518"    "0"      "2414"
## [191] "36289"  "51"     "440"    "1793"   "501"
## [196] "563"    "225"    "37653"  "0"      "60"
## [201] "135"    "158"    "58.9"   "120"    "84"
## [206] "403"    "0"      "209"    "2640"   "531"
## [211] "0"      "491"    "402"    "193"    "58.9"
## [216] "0"      "46.6"   "5313"   "3025"   "2798"
## [221] NA       "0"      "17968"  "4964"   "926"
## [226] "1340"   "853"    "386"    "3587"   "3218"
## [231] "0"      "193"    "1566.3" "0"      "1424"
## [236] "3219"   "706"    "56"     "101"    "419"
## [241] "362"    "1148"   "7200"   "0"      "389"
## [246] "24"     "0"      "2782"   "1318"   "12429"
## [251] "19924"  "4.8"    "660"    "0"      "2528"
## [256] "2800"   "3444"   "188"    "19.3"   "129"
## [261] "0"      "1110"   "356000" "1906"   "0"
## [266] "0"
```

Auch hier kann alles in einen Befehl gepackt werden:

```
coast$Coastline %>%
  str_extract("[0-9,.]+ km") %>%
  str_replace_all("[^0-9.]", "") %>%
  as.numeric() -> coast$coast_num
```

10.5 Datenaufbereitung

Mit dem Befehl `arrange()` kann die Tabelle sortiert werden. Zunächst aufsteigend,

```
coast %>%
  arrange(coast_num)
## # A tibble: 266 x 3
##    Country    Coastline                    coast_num
##    <chr>      <chr>                            <dbl>
##  1 Afghanist~ "0 km\n          (landlocked)"         0
##  2 Andorra    "0 km\n          (landlocked)"         0
##  3 Armenia    "0 km\n          (landlocked)"         0
##  4 Austria    "0 km\n          (landlocked)"         0
##  5 Azerbaijan "0 km\n          (landlocked);~        0
##  6 Belarus    "0 km\n          (landlocked)"         0
##  7 Bhutan     "0 km\n          (landlocked)"         0
##  8 Bolivia    "0 km\n          (landlocked)"         0
##  9 Botswana   "0 km\n          (landlocked)"         0
## 10 Burkina F~ "0 km\n          (landlocked)"         0
## # ... with 256 more rows
```

und schließlich absteigend, sodass die größten Werte an erster Stelle stehen.

```
coast %>%
  arrange(desc(coast_num))
## # A tibble: 266 x 3
##    Country     Coastline                       coast_num
##    <chr>       <chr>                               <dbl>
##  1 World       "356,000 km\n          \n    ~   356000
##  2 Canada      "202,080 km\n          \n    ~   202080
##  3 Pacific Oc~ "135,663 km"                       135663
##  4 Atlantic O~ "111,866 km"                       111866
##  5 Indian Oce~ "66,526 km"                         66526
##  6 European U~ "65,992.9 km"                       65993.
##  7 Indonesia   "54,716 km"                         54716
##  8 Arctic Oce~ "45,389 km"                         45389
##  9 Greenland   "44,087 km"                         44087
## 10 Russia      "37,653 km"                         37653
## # ... with 256 more rows
```

Bevor wir jedoch eine vollständig sortierte Liste haben, muss der Datensatz noch von falschen Einträgen gesäubert werden. Dafür benutzen wir den Befehl `filter()`. Wir suchen wieder nach einem bestimmten Muster (hier zum Beispiel dem Wort “Ocean”) und filtern es aus dem Datensatz.

Die Grafik soll nur aus den ersten 30 Einträgen der Tabelle bestehen, welche uns der Befehl `head()`

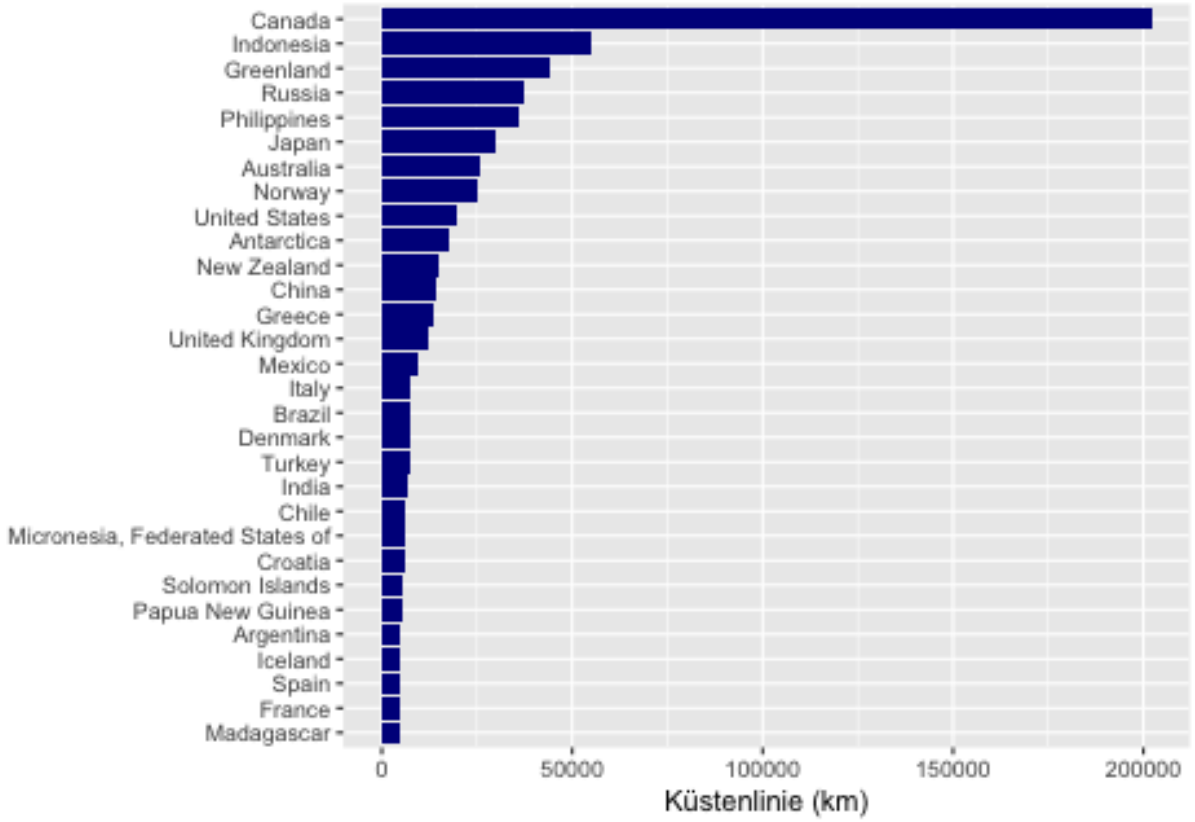
ausgibt.

```
coast %>%
  arrange(desc(coast_num)) %>%
  filter(!str_detect(Country, "Ocean")) %>%
  filter(!Country %in% c("World", "European Union")) %>%
  head(30) -> top_30
```

10.6 Datenvisualisierung

Das Balkendiagramm erhalten wir durch den “ggplot” Befehl. Hierbei gibt es verschiedenste Einstellmöglichkeiten. Wichitg sind vor allem die Angabe des verwendeten Datensatzes und die Art der Grafik (ob Kartendarstellung oder Balkendiagramm). Desweiteren kann man noch Farben der Eigenschaften, eine Achsenbeschriftung u. v. m. bestimmen.

```
ggplot(top_30, aes(x = reorder(Country, coast_num), y=coast_num)) +
  geom_bar(stat='identity', fill="darkblue") +
  coord_flip() +
  scale_x_discrete(NULL) +
  scale_y_continuous("Küstenlinie (km)")
```



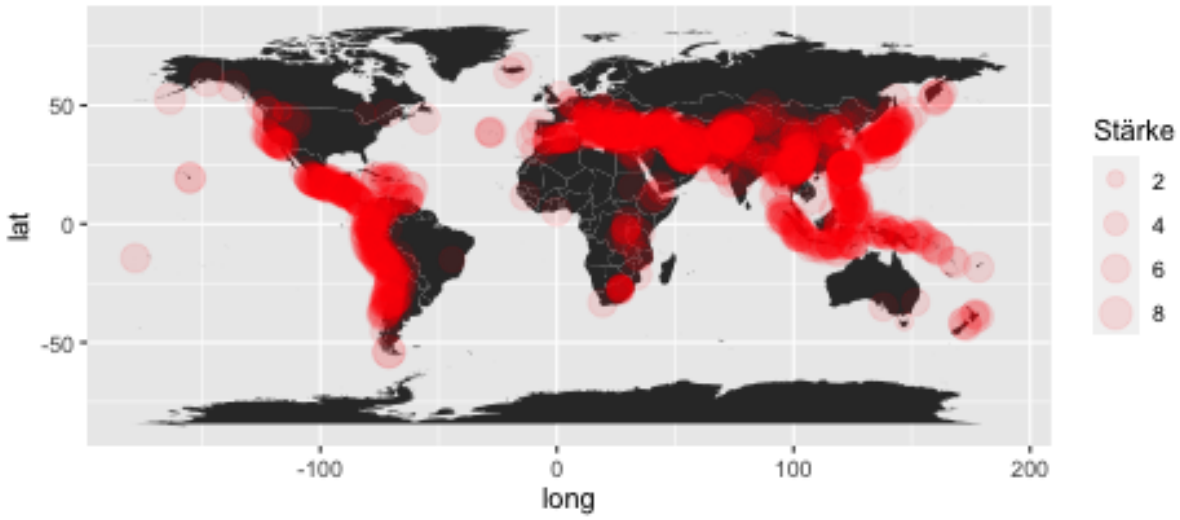
10.7 Aufgaben

1. Importieren Sie die Daten zu den tödlichen Erdbeben auf Wikipedia und formen sie diese zu einem tibble um.

```
"https://en.wikipedia.org/wiki/List_of_deadly_earthquakes_since_1900" %>%
  read_html %>%
  html_table(fill = T) %>%
  .[[5]] %>%
  as.tibble() -> earthquakes_raw
```

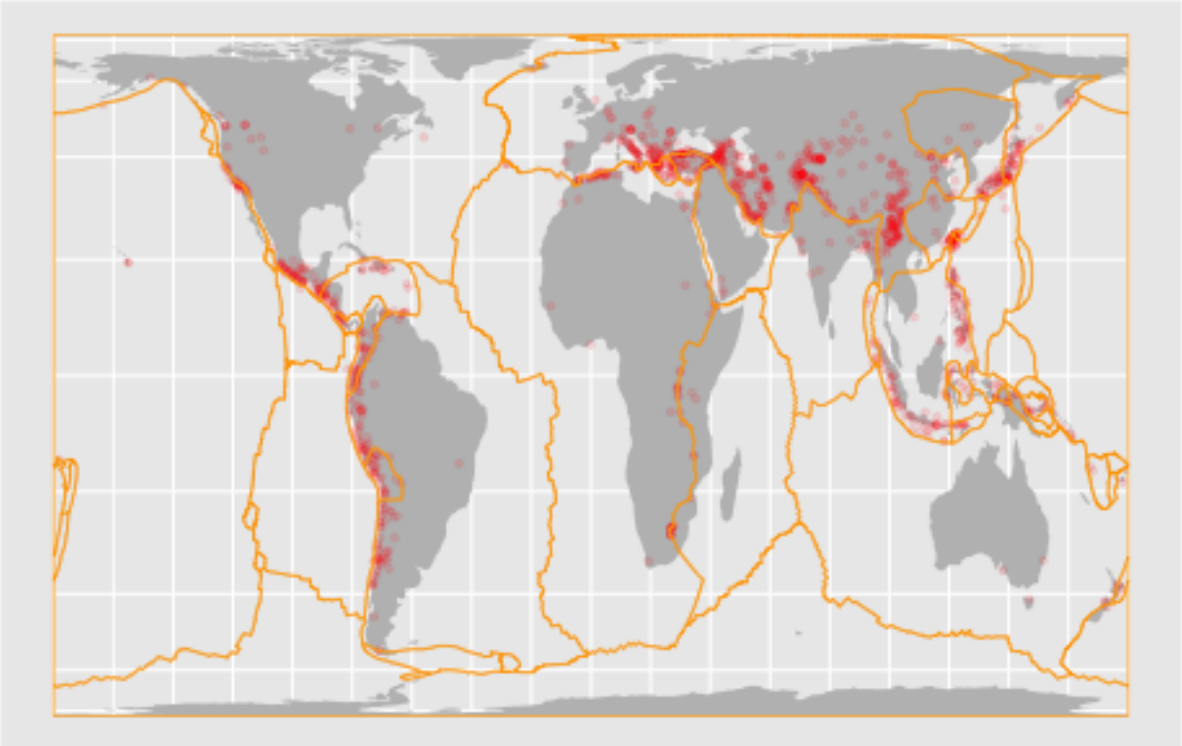
2. Erstellen Sie mit den erhaltenen Daten eine Karte, welche die Lage und die Stärke der Erdbeben angibt:

```
earthquakes_raw %>%
  mutate(Lat = as.numeric(Lat), Long = as.numeric(Long)) %>%
  mutate(magnitude_num = as.numeric(str_extract(Magnitude, "[0-9.]+"))) -> earthquakes

ggplot() +
  geom_polygon(data = map_data("world"), aes(x = long, y = lat, group = group)) +
  geom_point(data = earthquakes,
             aes(x = Long, y = Lat, size = magnitude_num),
             color = "red", alpha = 0.1) +
  coord_quickmap() +
  scale_size_area("Stärke")
```



3. Wandeln Sie den Erdbeben-Datensatz in das Simple Features Format um. Laden Sie zusätzlich eine Weltkarte mit dem Paket `rnaturalearth` und wandeln Sie auch diese in Simple Features um. Finden Sie außerdem einen Geodatensatz zu tektonischen Platten. Visualiseren Sie alles auf einer Welktarte (Projektion: Gall-Peters).

```
library(sf)

earthquakes %>%
  filter(! is.na(Long)) %>%
  st_as_sf(coords=c("Long", "Lat")) %>%
  st_set_crs(4326) -> quakesf

library(rnaturalearth)

ne_download(type="land", category = "physical") %>%
  st_as_sf() %>%
  st_transform('+proj=cea +lon_0=0 +x_0=0 +y_0=0 +lat_ts=45 +ellps=WGS84 +datum=WGS84 +units

st_read("https://raw.githubusercontent.com/fraxen/tectonicplates/master/GeoJSON/PB2002_plate

ggplot() +
  geom_sf(data = earthsf, fill = "gray", color = NA) +
  geom_sf(size = 1, data = quakesf, color = "red", alpha = 0.1) +
  geom_sf(data = plates, color = "orange", fill = NA, lwd = 0.3)
```



11 Web scraping

11.1 Lernziele dieser Sitzung

Sie können…

- HTML in seiner Grundstruktur interpretieren.
- gezielt einzelne Elemente einer Seite mit R auslesen.

11.2 Vorbereitung

Für diese Lektion werden die Pakete benötigt:

```
library(tidyverse)
library(rvest)
```

11.3 Exkurs: HTML

Wenn man eine Webseite ganz normal in einem Browser aufruft, erscheint sie als eine Mischung aus formatiertem Text, Bildern, Designelementen, ggf. Videos, usw. Was aber im Hintergrund eigentlich vom Server an den Browser übertragen wird, ist eine Textdatei in einem bestimmten Format – HTML (Hyptertext Markup Language). Darin wird Text auf eine genau festgelegte Art und Weise annotiert, damit der Browser weiß, wie er ihn anzeigen soll. Im HTML-Dokument kann auch stehen: Lade ein Bild von einer bestimmten Stelle und zeig es an dieser Stelle an.

Einen brauchbaren Überblick über die HTML-Elemente und die Struktur einer HTML-Datei gibt es hier: https://www.tutorialspoint.com/de/html/

An dieser Stelle ist wichtig ist zu wissen: HTML-Elemente (“Tags” oder “Nodes”) sind streng hierarchisch angeordnet. Sie bestehen oft aus einem Anfangs- und einem End-Tag in spitzen Klammern:

```
<html>
  <head>
    <title>Titel meiner Webseite</title>
  </head>
  <body>
    <h1>Überschrift</h1>
    <p>Erster Absatz mit <b>fettem Text</b></p>
    <p>Zweiter Absatz mit <i>kursivem Text</i></p>
    <img src="path/to/image.jpg alt="Ein Bild" />
  </body>
</html>
```

In diesem Beispiel ist das Bild mit `<img />` das einzige Element, das nicht geöffnet und wieder geschlossen wird. Außerdem hat dieses Element Attribute (`src` und `alt`) mit bestimmten Werten. Eine echte Webseite ist weitaus komplexer und unübersichtlicher.

Dem Browser kann man sagen: Zeig mir nicht wie üblich die „gerenderte“ Seite an, sondern die zu Grunde liegende HTML-Datei. Das geht mit „Quelltext anzeigen“ / „View Source“ o.ä.

Viele Browser (hier seien Chrome und Firefox empfohlen) haben auch einen Modus namens „Entwicklertools“ / „Developer tools“, in dem die HTML-Elemente hierarchisch geordnet sind.

11.4 Web Scraping

Beim so genannten Web Scraping ist die Grundidee, dass wir eine Webseite nicht im Browser öffnen, sondern den HTML-Quelltext direkt in R laden. R kann dann aus dem Quelltext bestimmte Elemente extrahieren.

In der letzten Sitzung haben wir schon gesehen, wie Tabellen nach genau diesem Prinzip von einer Webseite direkt in R geladen werden können. Jetzt soll es darum gehen, noch präziser zu sagen, welche Elemente wir von einer bestimmten Webseite ziehen wollen.

Als Beispiel soll die Infoseite eines Wohnheims des Studentenwerks Frankfurt dienen. Die Adresse ist: https://www.studentenwerkfrankfurt.de/wohnen/wohnheime/frankfurt-am-main/kleine-seestrasse-11

Zunächst laden wir den Quelltext in R und nennen ihn `quelltext`:

```
quelltext <- read_html("https://www.studentenwerkfrankfurt.de/wohnen/wohnheime/frankfurt-am-

quelltext
## {html_document}
## <html lang="de" dir="ltr" class="no-js">
## [1] <head>\n<meta http-equiv="Content-Type" content= ...
## [2] <body id="p187" class="page-187 pagelevel-4 lang ...
```

In diesem Schritt hat R den HTML-Quelltext schon "geparsed", d.h. ihn nicht nur als Text gespeichert, sondern als hierarchische Konstruktion mit den beiden Grundelementen `head` und `html`.

Uns soll jetzt das Baujahr interessieren. Auf der Seite stehen die Informationen rechts neben dem Bild. Mit den Entwicklertools können wir schauen, wie die Elemente in HTML genau heißen. Ein geeigneter Ausgangspunkt wäre das `<div>`-Element mit dem Attribut `id="c599"`.

In R können wir dieses einzelne Element ansprechen mit:

```
quelltext %>%
  html_node("div#c599")
## {html_node}
## <div id="c599" class="frame frame-default frame-type-text frame-layout-0 frame-backgroun
## [1] <div class="frame-container"><div class="frame-i ...
```

Dann gehen wir in der Hierarchie drei `<div>`- Elemente „tiefer". (`<div>`- Elemente sind abstrakte Container und werden im Webdesign oft angewendet.)

```
quelltext %>%
  html_node("div#c599") %>%
  html_node("div") %>%
  html_node("div")
## {html_node}
## <div class="frame-inner">
## [1] <p>Kleine Seestraße 11<br>60486 Frankfurt am Mai ...
## [2] <p>25 Wohnheimplätze</p>\n
## [3] <ul class="list-normal">\n<li>5 Wohnküchen</li>\ ...
## [4] <p>Baujahr<strong> </strong>1995\r</p>\n
## [5] <p><strong></strong></p>
```

Alternativ könnten wir auch sagen: Darin das `div` mit `class="frame-inner"`:

```
quelltext %>%
  html_node("div#c599") %>%
  html_node("div.frame-inner")
## {html_node}
## <div class="frame-inner">
## [1] <p>Kleine Seestraße 11<br>60486 Frankfurt am Mai ...
## [2] <p>25 Wohnheimplätze</p>\n
## [3] <ul class="list-normal">\n<li>5 Wohnküchen</li>\ ...
## [4] <p>Baujahr<strong> </strong>1995\r</p>\n
## [5] <p><strong></strong></p>
```

mit `html_nodes()` (Mehrzahl) werden alle Unterelemente eines Typs (hier <p> = Paragraph) angesprochen. Davon dann den Textinhalt (`html_text()`) gibt uns die relevanten Informationen:

```
quelltext %>%
  html_node("div#c599") %>%
  html_node("div.frame-inner") %>%
  html_nodes("p") %>%
  html_text()
## [1] "Kleine Seestraße 1160486 Frankfurt am Main\r"
## [2] "25 Wohnheimplätze"
## [3] "Baujahr 1995\r"
## [4] ""
```

Jetzt ließe sich der dritte Eintrag säubern und als Ergebnis “speichern”:

```
quelltext %>%
  html_node("div#c599") %>%
  html_node("div.frame-inner") %>%
  html_nodes("p") %>%
  html_text() %>%
  .[3] %>%
  str_extract("[0-9]{4}") %>%
  as.numeric() -> baujahr

baujahr
## [1] 1995
```

Wie diese Technik automatisiert auf eine Reihe von Seiten angewendet werden kann, wird zu einem späteren Zeitpunkt besprochen.

11.5 Aufgaben

1. Lesen Sie die Anzahl der Wohnheimplätze aus.
2. Lesen Sie Baujahr und Anzahl der Wohnheimplätze von einem anderen Wohnheim aus. Was muss angepasst werden?
3. Ändern Sie das Script so, dass es auf beiden (allen) Wohnheimseiten funktioniert.

4. Lesen Sie die Adresse eines Wohnheims aus. Speichern Sie dabei Straße, Hausnummer, Postleitzahl und Ort getrennt.
5. Sammeln Sie eine Liste aller Wohnheime mit Link.
6. Sammeln Sie einen Datensatz (tibble) aller "Nutzungsentgelte" mit Wohnheim, Baujahr, Anzahl Wohneinheiten und Adresse. Stellen Sie sich vor es handle sich um Tausende Wohnheime – vermeiden Sie also Copy-Paste-Strategien.