Spatial Analysis mit R (I)

Methodenwoche

Till Straube
straube@geo.uni-frankfurt.de

Institut für Humangeographie
Goethe-Universität Frankfurt

20.–21. September 2021

Inhaltsverzeichnis

Zeitplan

Alle Sitzungen finden über Zoom statt.

Zeit	Montag	Dienstag
10:00–11:30	**(1) Getting started (LAS)**	**(5) Geodaten verschneiden: FTR**
11:30–11:45	*Kaffeepause*	*Kaffeepause*
11:45–13:15	**(2) Daten visualisieren (FTR, HOS)**	**(6) Geodaten verschneiden: HOS, SYW**
13:15–14:15	*Mittagspause*	*Mittagspause*
14:15–15:45	**(3) Karten erstellen (FTR)**	**(7) Nach Hilfe fragen und und publizieren (FTR)**
15:45–16:15	*Kaffeepause*	*Kaffeepause*
16:15–18:00	**(4) Karten erstellen (HOS, SYW)**	**(8) Looking back, looking ahead (LAS)**

1 Getting started

1.1 Formales

1.1.1 Keine reguläre Anrechnung des Workshops

- Die Methodenwoche ist eine außercurriculare Veranstaltung
- Keine Anrechnung als Prüfungsleistung für das reguläre Studium
- Alle Teilnehmer*innen erhalten Methodenzentrum eine Bescheinigung über die erbrachte Leistung (Ende 2021/Anfang 2022)

1.1.2 Methodenzertifikat

- Nur für Bachelor-Studierende der Fachbereiche 02–05
- Kann mit 5 CP beantragt werden
- z. B. Teilnahme Workshop (2 CP) + Leistungsnachweis (3 CP)
- Maximal 20% Fehlzeit zulässig für Teilnahmenachweis
- Schriftliche Leistungsnachweise mit maximal vierwöchiger Abgabefrist
- Alle Fragen dazu bitte an hiwis-methodenzentrum@uni-frankfurt.de

1.2 Inhaltliches

1.2.1 Lernziele der Veranstaltung

Sie können…

- Datenvisualisierungen nachvollziehen und selbst gestalten.
- Geodaten einlesen, transformieren und verschneiden.
- Geodaten kartographisch darstellen.
- Reproducible examples erstellen um nach Hilfe zu fragen.
- Berichte in Rmarkdown verfassen und rendern.

1.2.2 Seminarkonzept

- Kompetenter Umgang mit Geodaten als Kernziel
- Aber nicht im luftleeren Raum
- Das Drumherum ist mindestens genauso wichtig
- Unterlagen sind Auszüge aus einem zweisemestrigen Seminar

1.2.3 “Opinionated…”

- package choices:
  - `tidyverse`
  - `sf`
- coding style:
  - Functional
  - Pipes %>%
- workflow:
  - Incremental commands
  - Rmarkdown (reproducible research)

1.3 Didaktisches

1.3.1 Herausforderungen in der IT-Didaktik

- Unterschiedliche Erfahrungen, Kompetenzen und Herangehensweisen
- Die eine Hälfte versteht gar nichts, die andere langweilt sich
- Kleinster gemeinsamer Nenner: Schritt-für-Schritt-Anleitungen
- In der Praxis wertlos

1.3.2 Everyone fails

- In der Praxis stoßen alle ständig an die Grenzen ihrer technischen Kompetenz.
- Es geht darum, sich am Limit einigermaßen wohl zu fühlen und die Grenzen zu verschieben.
- Gute Angewohnheiten (Strukturen, Formate, Stil) helfen dabei!

1.3.3 Mein Ansatz in der Lehre

- Strategische Überforderung durch schwierige Aufgaben?
- Lösungsorientierte Didaktik!
- Die affektive Seite (Spaß, Frust) ernst nehmen und thematisieren
- Frustrationsschwelle trainieren

1.3.4 “Schattenkompetenzen”

- Über Code reden
- Fehlermeldungen lesen
- Gezielt googlen (und Antworten auswählen)
- Copy, paste, customize
- Gute Fragen (online) stellen

1.3.5 Dieser Workshop findet in verschiedenen Modi statt:

1.3.5.1 Listen and share (LAS)

- Ich rede (mit Folien oder ohne) oder moderiere eine Diskussion.
- Sie hören mir und Ihren Kommiliton*innen aufmerksam zu.
- Sie “melden” sich für Redebeiträge oder Fragen (Zoom-Funktion).

1.3.5.2 Follow the recipe (FTR)

- 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.
- Sie unterbrechen mich mit Nachfragen oder Problemen.

1.3.5.3 Hands-on session (HOS)

- 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 “mache die Runde”.

1.3.5.4 Share your work (SYW)

- 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.

1.4 Technisches

1.4.1 Arbeitsplatz

- Challenge: Zoom (meinen Bildschirm) und R gleichzeitig sehen
- Am allerbesten: Zweiter Bildschirm
- Auch gut: Zweites Gerät (Tablet)

1.4.2 Workshopunterlagen

- Bookdown (statt OLAT)
- ~~Können~~ Werden sich verändern
- Am besten neu laden mit Strg+Umschalt+R
- https://tiny.gu/mwsa

1.4.3 RStudio Cloud

- Grundsätzliche Empfehlung: R und RStudio lokal installieren
- Wir nutzen im Rahmen des Workshops die RStudio Cloud
  - für einfaches Teilen von Code
  - für ein homogenes Setup

2 Daten visualisieren

2.1 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.

2.2 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     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
```

```
##  2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
## 10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
## # ... with 53,930 more rows
```

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

```
data(diamonds)
```

2.3 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:

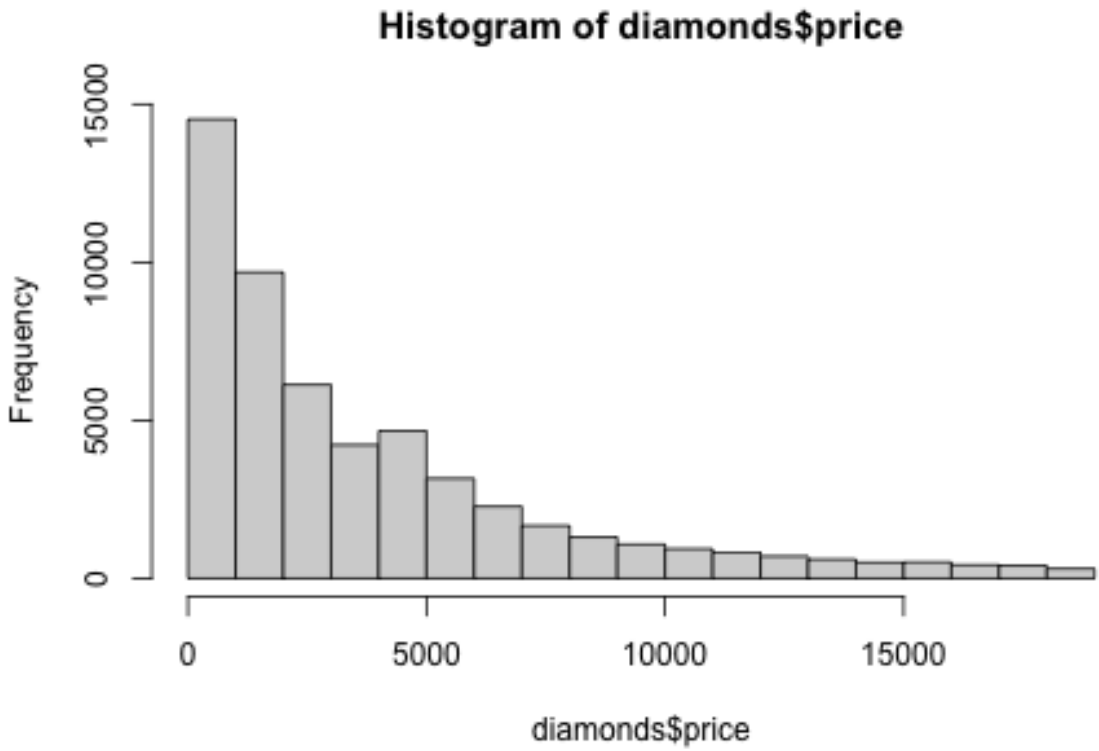
```
summary(diamonds)
##      carat               cut        color        clarity          depth      
##  Min.   :0.2000   Fair      : 1610   D: 6775   SI1    :13065   Min.   :43.00  
##  1st Qu.:0.4000   Good      : 4906   E: 9797   VS2    :12258   1st Qu.:61.00  
##  Median :0.7000   Very Good:12082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

```
##  1st Qu.:56.00   1st Qu.:  950   1st Qu.: 4.710   1st Qu.: 4.720
##  Median :57.00   Median : 2401   Median : 5.700   Median : 5.710
##  Mean   :57.46   Mean   : 3933   Mean   : 5.731   Mean   : 5.735
##  3rd Qu.:59.00   3rd Qu.: 5324   3rd Qu.: 6.540   3rd Qu.: 6.540
##  Max.   :95.00   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
##
```

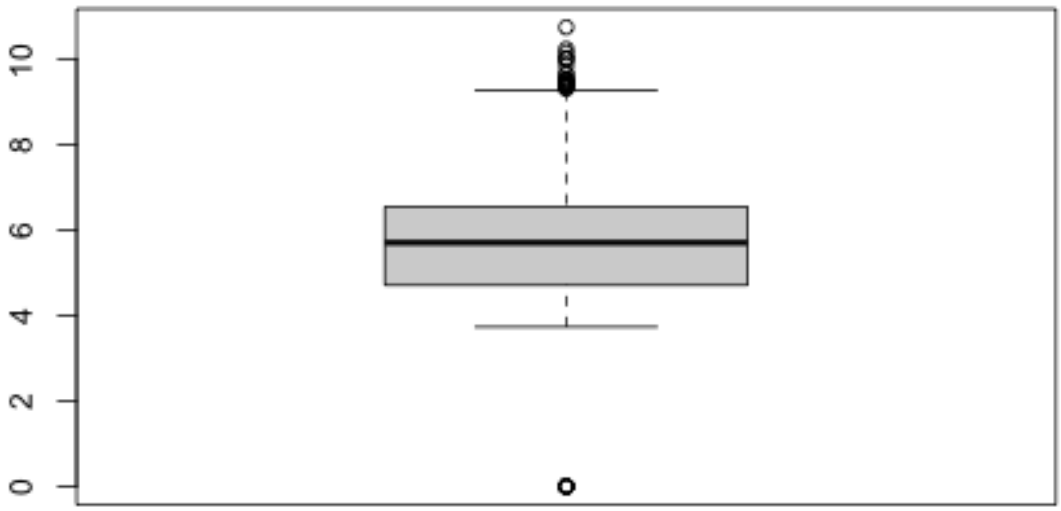
2.4 Visualisierung mit dem Standardpaket

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

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




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



2.5 Visualisierung mit `ggplot()`

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

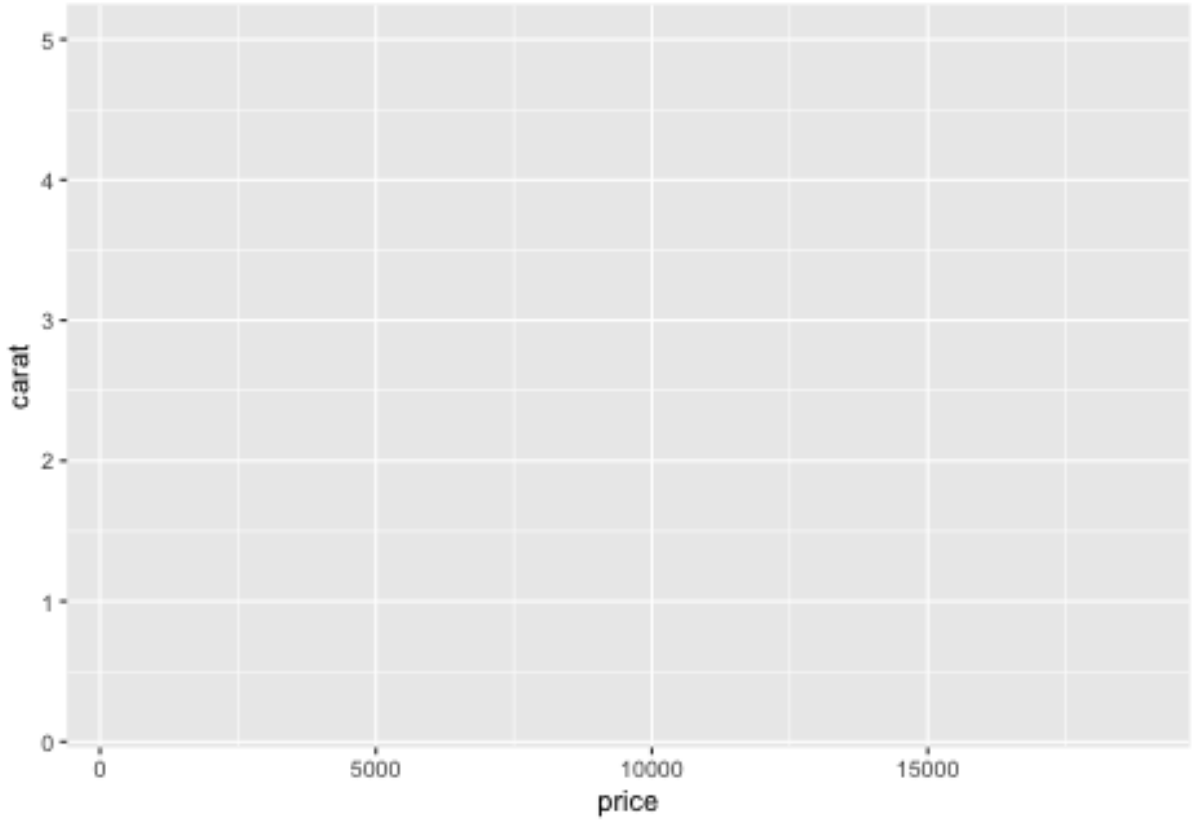
Die Syntax ist dabei auf den ersten Blick etwas komplexer.

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

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

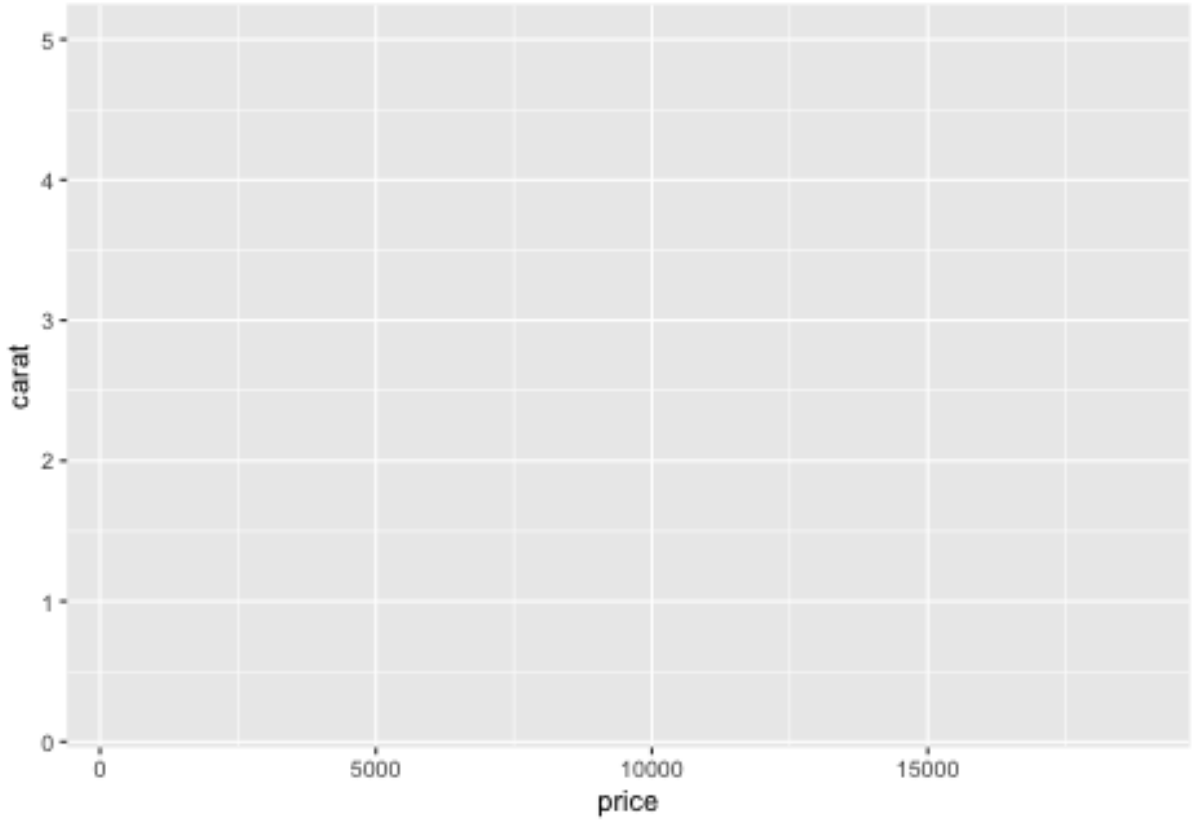
Mit einem Mapping-Parameter legen wir die Dimensionen fest:

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



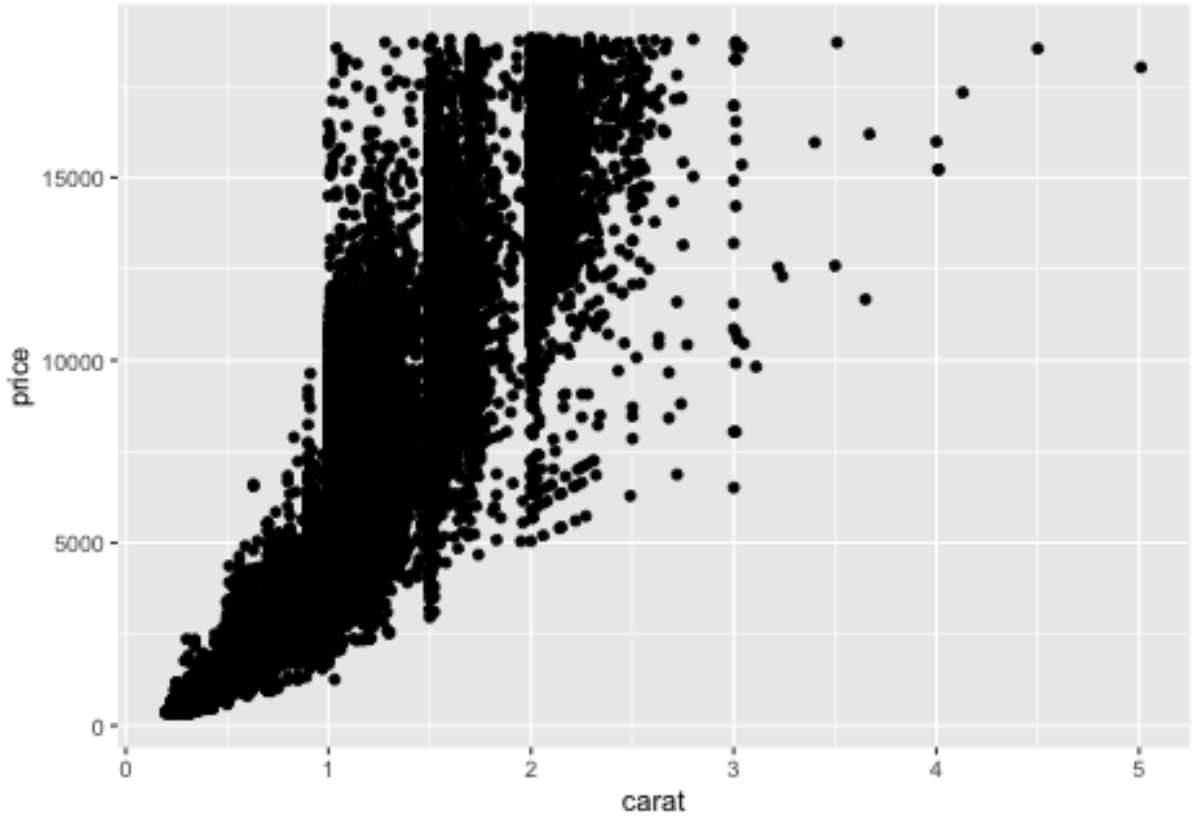
Das gleiche ohne Parameternamen:

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



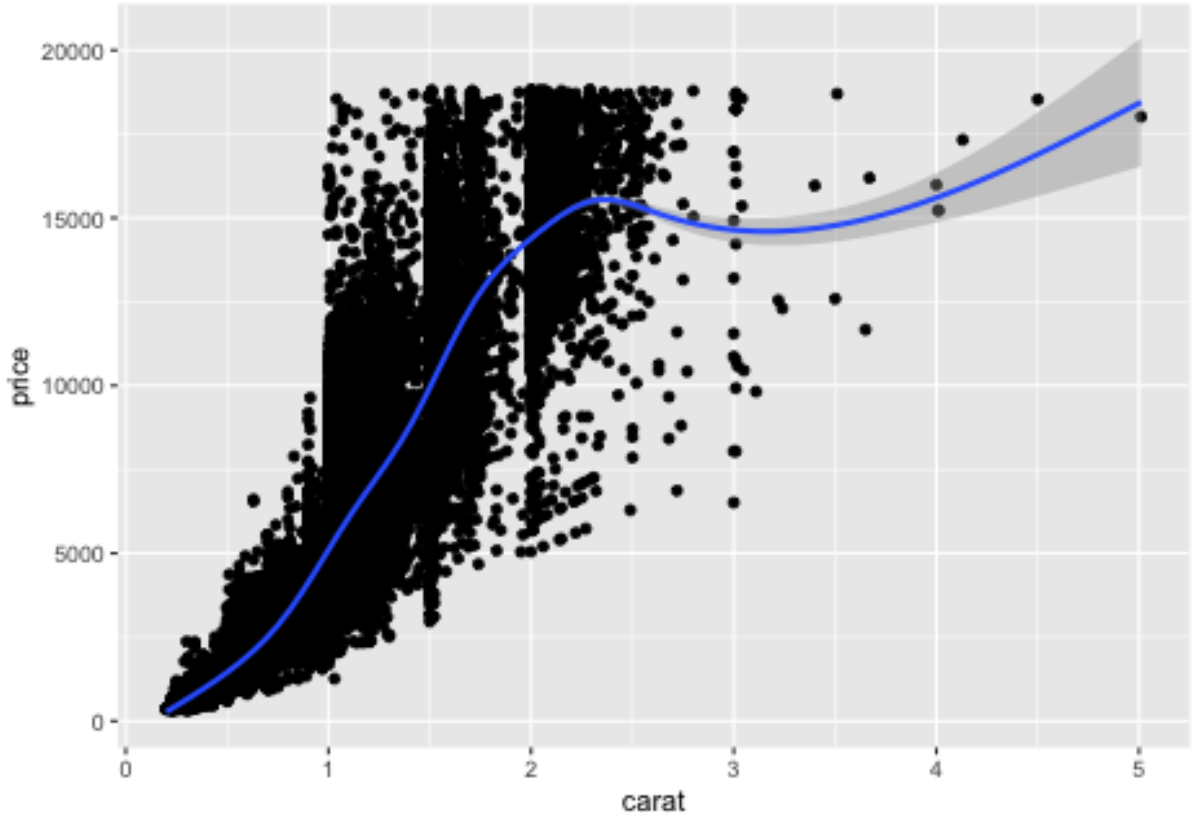
Nun kann mit dem +-Operator ein “geometrischer” Layer hinzugefügt werden:

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



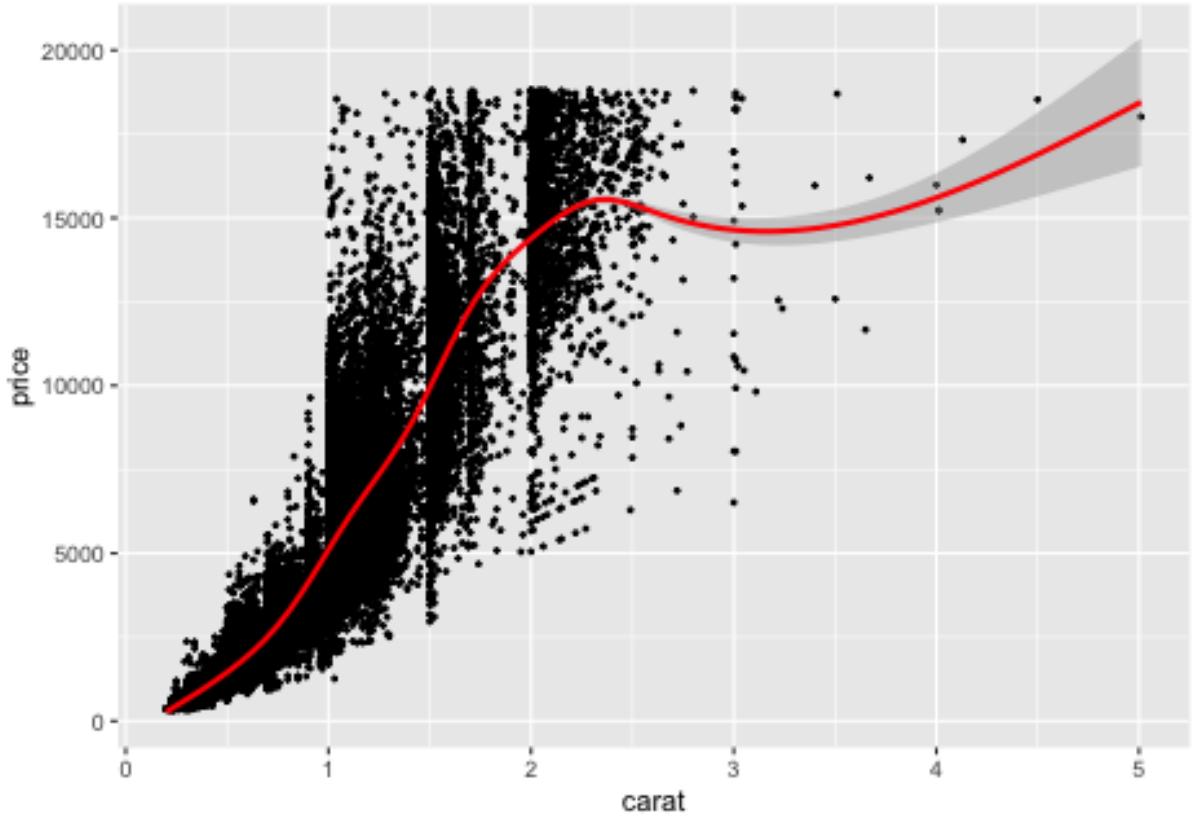
Weitere `geom`-Layer lassen sich mit dem +-Operator hinzufügen:

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



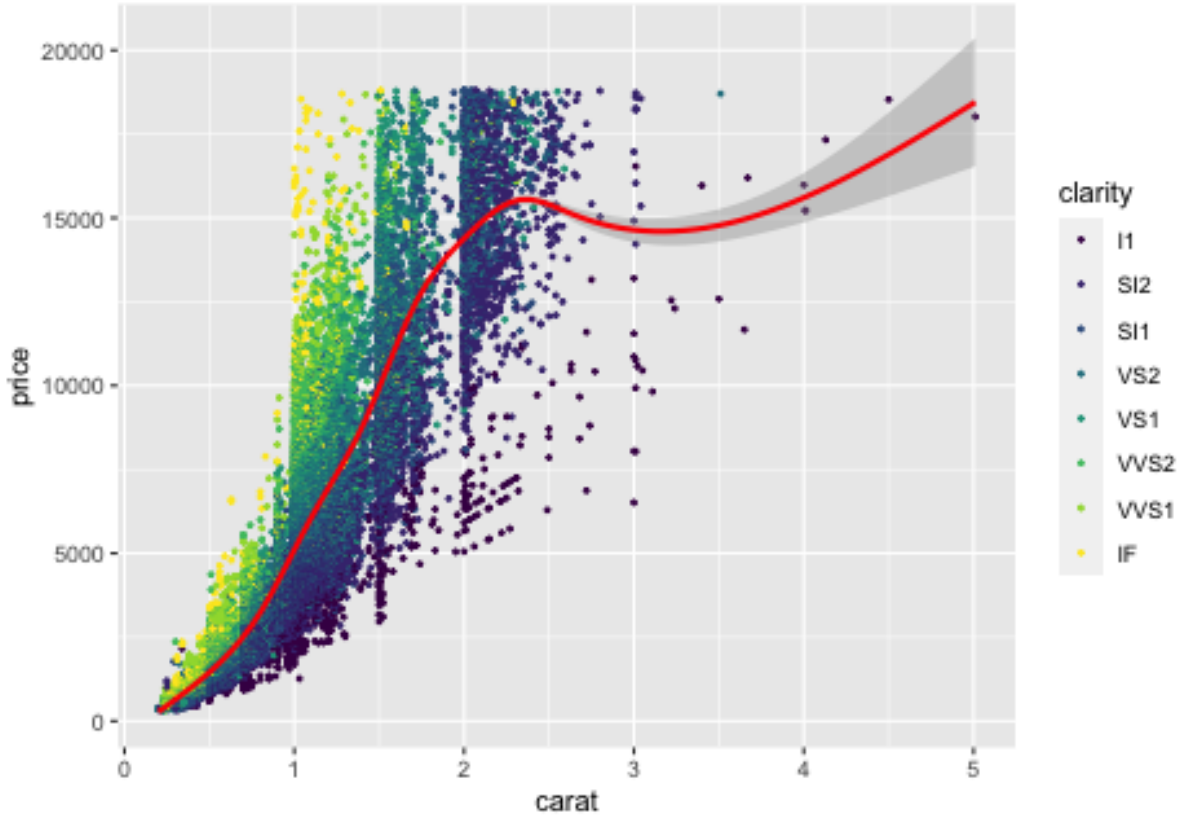
Die Layer-Funktionen können durch Parameter angepasst werden:

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



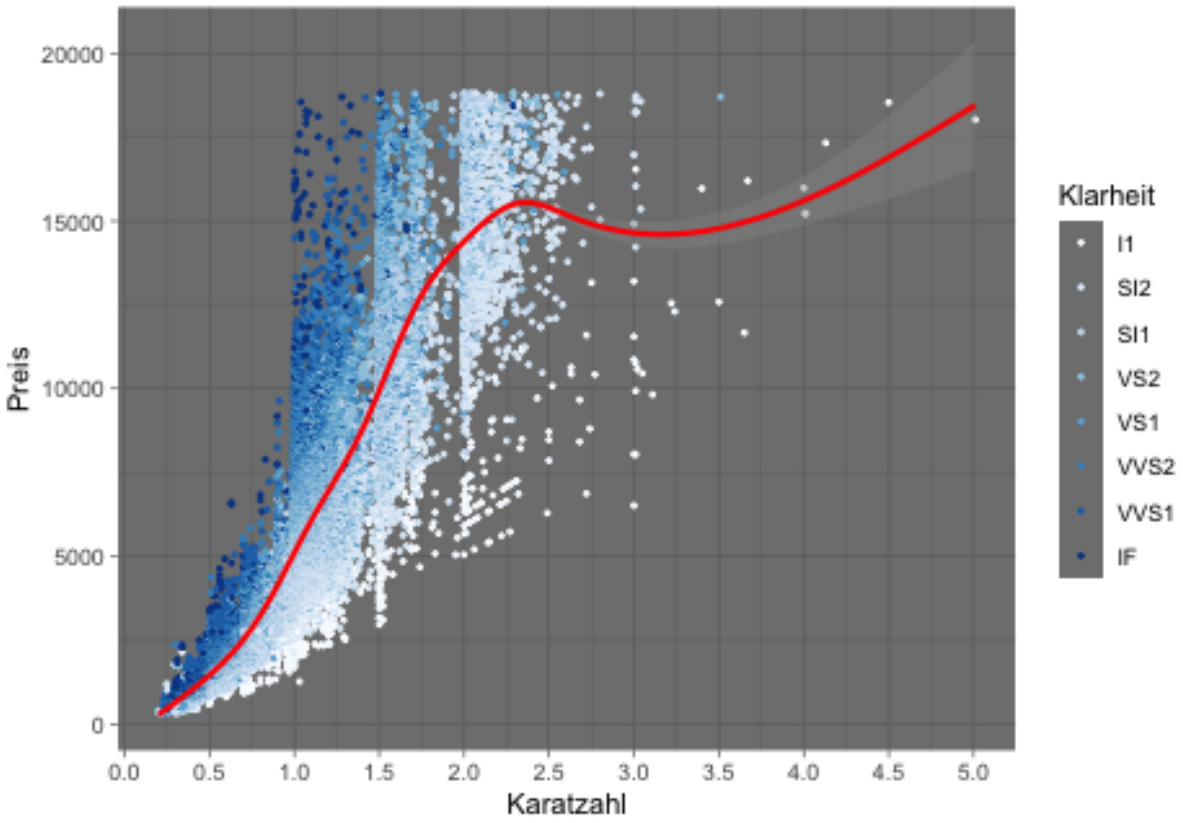
Dabei lassen sich in den einzelnen Layers mappings hinzufügen oder verändern:

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



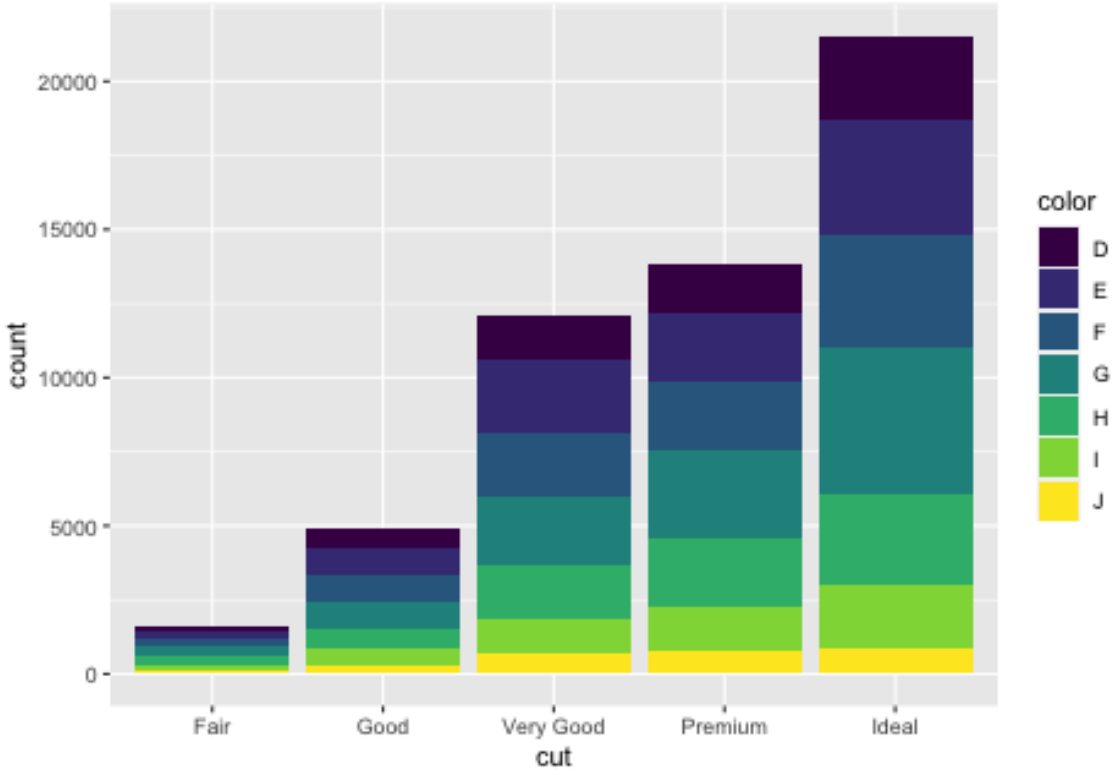
Schließlich lassen sich noch viele weitere optische Aspekte anpassen, z.B. Achsen, Farben, etc.:

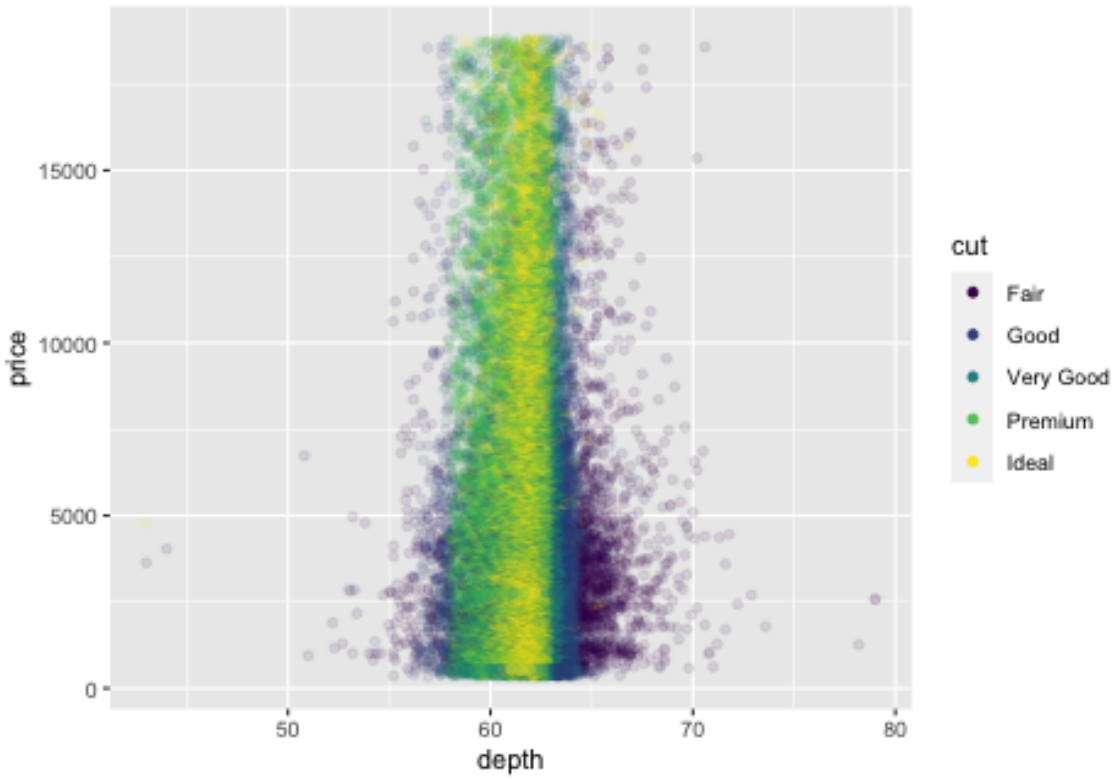
```
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_point(aes(color = clarity), size = 0.5) +
  geom_smooth(color = "red") +
  scale_x_continuous("Karatzahl", breaks = seq(0, 5, 0.5)) +
  scale_y_continuous("Preis") +
  scale_color_brewer("Klarheit") +
  theme_dark()
```

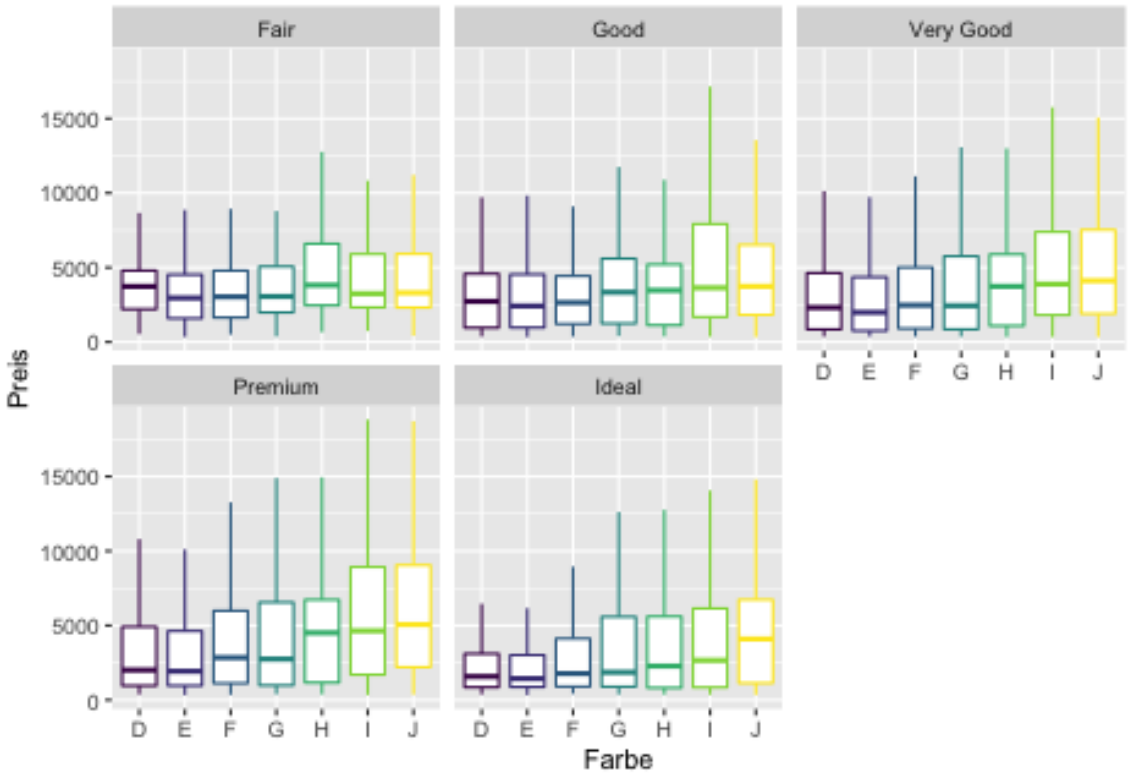


2.6 Aufgaben

1. Versuchen Sie, die folgenden 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



2. Schauen Sie sich die Publikation R for Data Science an.
3. Was ist das für ein Buch? Wer ist das Zielpublikum?
4. Lesen Sie das Kapitel “3: Data Visualization” und vollziehen Sie die Visualisierungen nach.
5. Bearbeiten Sie die Aufgaben.
6. Bearbeiten Sie die RStudio Primers zu Datenvisualisierung.

3 Karten erstellen (FTR)

3.1 Lernziele dieser Sitzung

Sie können…

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

3.2 Voraussetzungen

Wir laden erstmal `tidyverse`:

```
library(tidyverse)
```

3.3 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.

3.4 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-87
```

3.5 Überblick verschaffen

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

```
summary(baumkataster)
##  Gattung/Art/Deutscher Name   Baumnummer         Objekt             Pflanzjahr
##  Length:118403              Min.   :    1.0   Length:118403      Min.   :1645
##  Class :character           1st Qu.:   24.0   Class :character   1st Qu.:1970
##  Mode  :character           Median :   82.0   Mode  :character   Median :1982
##                             Mean   :  232.7                      Mean   :1979
##                             3rd Qu.:  270.0                      3rd Qu.:1995
```

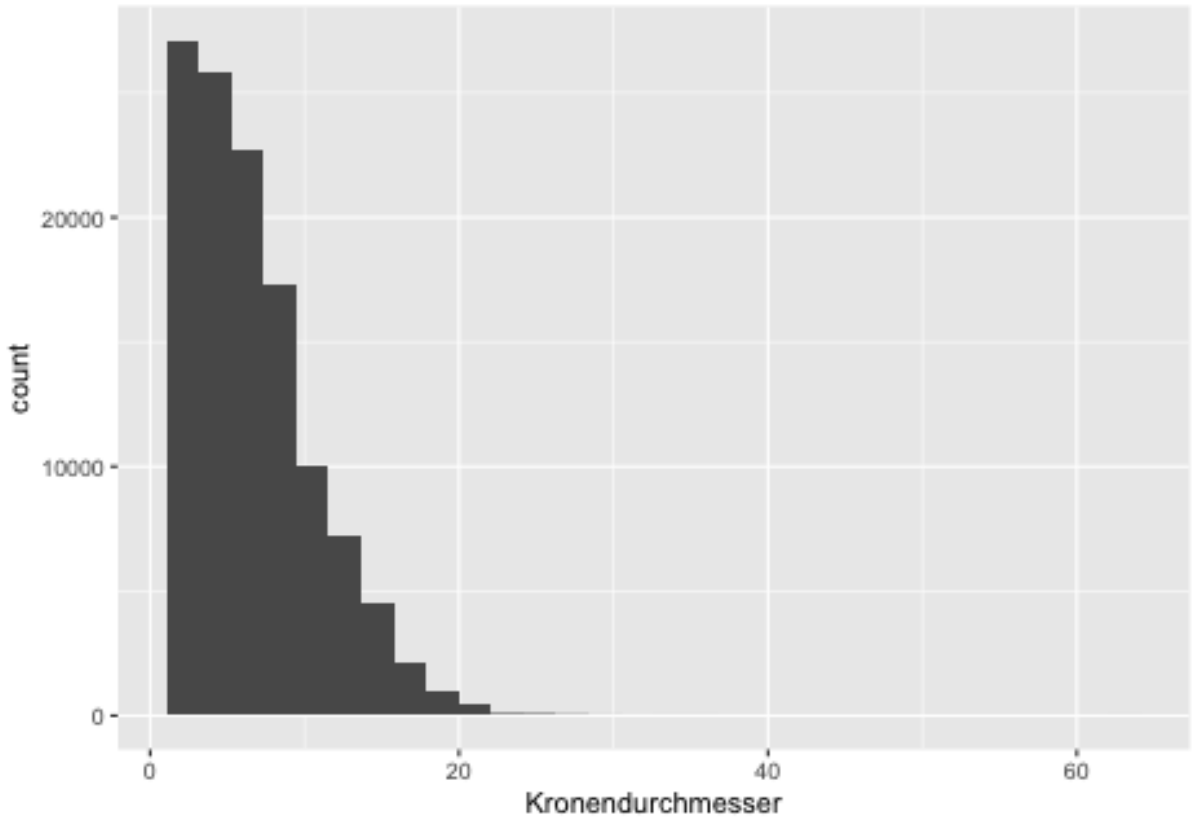
```
##                                Max.   :20158.0                        Max.   :2017
##                                NA's   :1853
##  Kronendurchmesser    HOCHWERT          RECHTSWERT
##  Min.   : 2.000    Min.   :5545117   Min.   :463163
##  1st Qu.: 4.000    1st Qu.:5550428   1st Qu.:472715
##  Median : 6.000    Median :5552601   Median :475219
##  Mean   : 6.688    Mean   :5552953   Mean   :475244
##  3rd Qu.: 9.000    3rd Qu.:5555165   3rd Qu.:478201
##  Max.   :63.000    Max.   :5563639   Max.   :485361
##
```

Genauere Infos über diese Merkmale gibt es auf dem Datenportal.

3.6 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/Deutsch~ Baumnummer Objekt  Pflanzjahr Kronendurchmess~ HOCHWERT
##    <chr>                       <dbl> <chr>        <dbl>            <dbl>    <dbl>
##  1 Platanus x hispanica~           1 Ackerm~       1920                8 5549511.
##  2 Platanus x hispanica~           2 Ackerm~       1920                8 5549517.
##  3 Platanus x hispanica~           3 Ackerm~       1920                8 5549524.
##  4 Platanus x hispanica~           4 Ackerm~       1920                8 5549531
##  5 Platanus x hispanica~           5 Ackerm~       1920                8 5549538.
##  6 Platanus x hispanica~           6 Ackerm~       1920                8 5549544.
##  7 Platanus x hispanica~           7 Ackerm~       1920                8 5549551.
##  8 Platanus x hispanica~           8 Ackerm~       1920                8 5549557.
##  9 Platanus x hispanica~           9 Ackerm~       1920                8 5549564.
## 10 Platanus x hispanica~          10 Ackerm~       1920                8 5549571.
## # ... with 118,393 more rows, and 2 more variables: RECHTSWERT <dbl>,
## #   alter <dbl>
```

Derselbe Befehl mit dem Pipe-Operator:

```
baumkataster %>%
  mutate(alter = 2020 - Pflanzjahr)
## # A tibble: 118,403 x 8
##    `Gattung/Art/Deutsch~ Baumnummer Objekt  Pflanzjahr Kronendurchmess~ HOCHWERT
##    <chr>                       <dbl> <chr>        <dbl>            <dbl>    <dbl>
##  1 Platanus x hispanica~           1 Ackerm~       1920                8 5549511.
##  2 Platanus x hispanica~           2 Ackerm~       1920                8 5549517.
##  3 Platanus x hispanica~           3 Ackerm~       1920                8 5549524.
##  4 Platanus x hispanica~           4 Ackerm~       1920                8 5549531
##  5 Platanus x hispanica~           5 Ackerm~       1920                8 5549538.
##  6 Platanus x hispanica~           6 Ackerm~       1920                8 5549544.
##  7 Platanus x hispanica~           7 Ackerm~       1920                8 5549551.
##  8 Platanus x hispanica~           8 Ackerm~       1920                8 5549557.
##  9 Platanus x hispanica~           9 Ackerm~       1920                8 5549564.
## 10 Platanus x hispanica~          10 Ackerm~       1920                8 5549571.
## # ... with 118,393 more rows, and 2 more variables: RECHTSWERT <dbl>,
## #   alter <dbl>
```

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

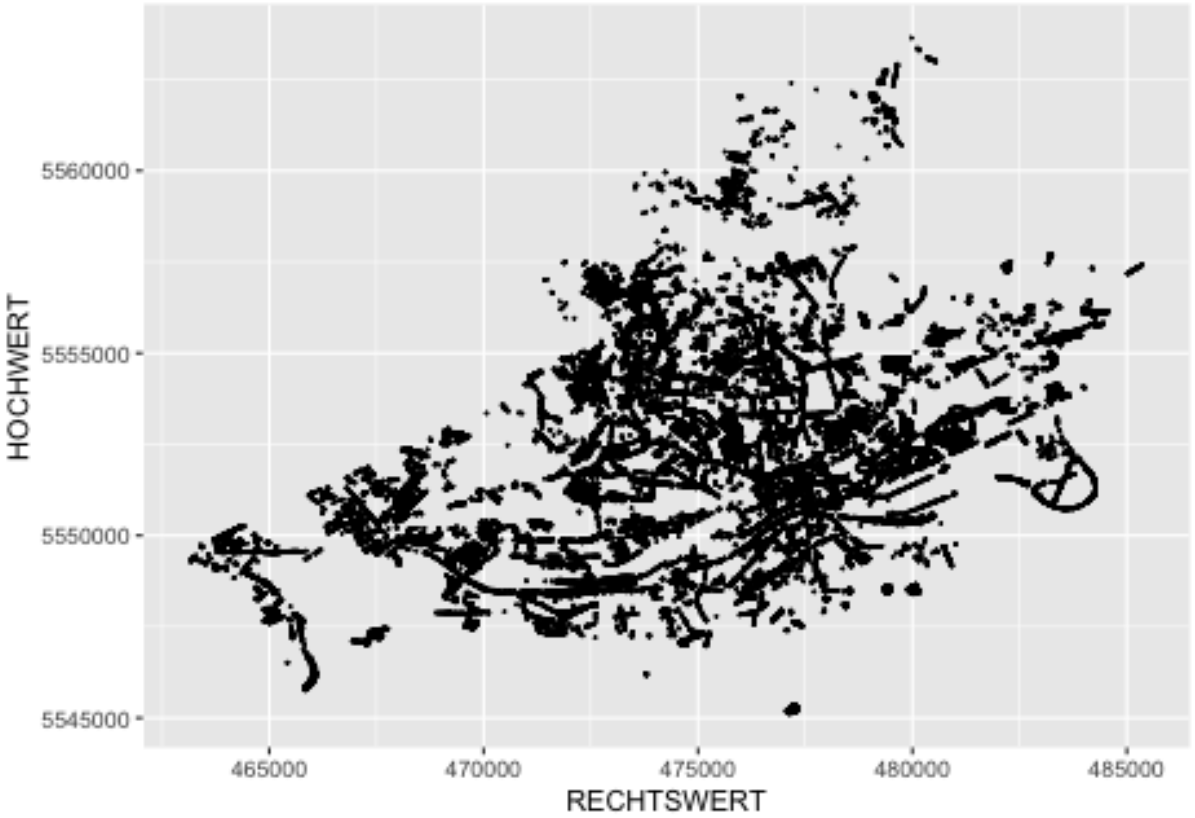
```
baumkataster %>%
  mutate(alter = 2020 - Pflanzjahr) %>%
  filter(alter > 30) ->
  alte_baeume

summary(alte_baeume)
##  Gattung/Art/Deutscher Name   Baumnummer         Objekt            Pflanzjahr
##  Length:73859               Min.   :    1.0   Length:73859       Min.   :1645
##  Class :character           1st Qu.:   29.0   Class :character   1st Qu.:1960
```

```
##  Mode  :character           Median :   97.0   Mode  :character   Median :1974
##                             Mean   :  263.2                     Mean   :1966
##                             3rd Qu.:  314.0                     3rd Qu.:1980
##                             Max.   :10489.0                     Max.   :1989
##                             NA's   :684
##  Kronendurchmesser    HOCHWERT         RECHTSWERT         alter
##  Min.   : 2.000    Min.   :5545117   Min.   :463163   Min.   : 31.00
##  1st Qu.: 6.000    1st Qu.:5550415   1st Qu.:472667   1st Qu.: 40.00
##  Median : 8.000    Median :5552480   Median :475708   Median : 46.00
##  Mean   : 8.503    Mean   :5552593   Mean   :475402   Mean   : 53.54
##  3rd Qu.:10.000    3rd Qu.:5554589   3rd Qu.:478539   3rd Qu.: 60.00
##  Max.   :35.000    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.

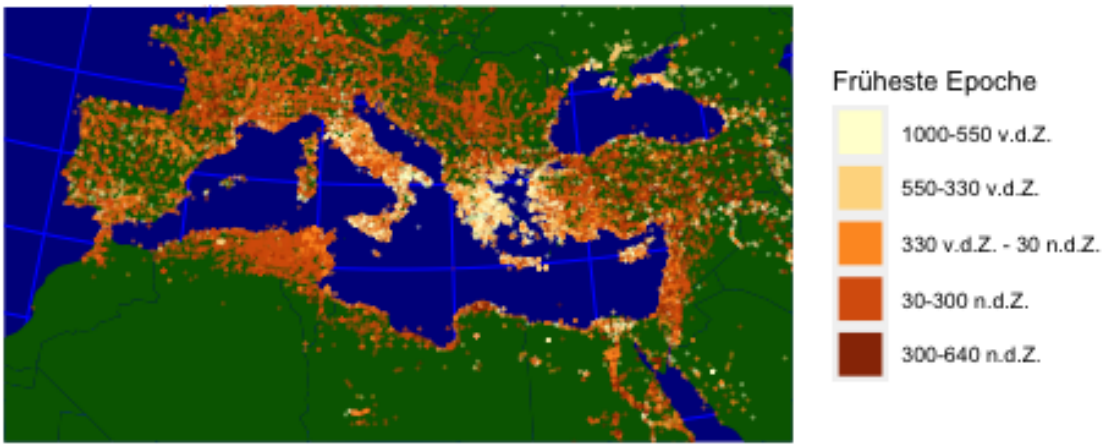
4 Karten erstellen (HOS)

4.1 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: knifflig!) Bilden Sie diese Grafik nach, die die Orte geordnet nach ältestem Fund darstellt:



5 Geodaten beschaffen

5.1 Lernziele dieser Sitzung

Sie können…

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

5.2 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)
```

5.3 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/t
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
##  $ : tibble [266 x 2] (S3: tbl_df/tbl/data.frame)
##   ..$ 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-world
  read_html() %>%
  html_table(fill = T) %>%
  .[[1]] %>%
  as_tibble() -> coast
```

5.4 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"     "0"       "1600"
##   [8] "61"      "17968"   "153"     "45389"   "4989"    "0"       "68.5"
##  [15] "74.1"    "111866"  "25760"   "0"       "0"       "3542"    "161"
##  [22] "580"     "97"      "0"       "66.5"    "386"     "121"     "103"
##  [29] "0"       "0"       "20"      "0"       "29.6"    "7491"    "698"
##  [36] "80"      "161"     "354"     "0"       "1930"    "0"       "965"
##  [43] "443"     "402"     "202080"  "160"     "0"       "0"       "6435"
##  [50] "14500"   "138.9"   "11.1"    "26"      "3208"    "340"     "37"
##  [57] "169"     "120"     "3095"    "1290"    "515"     "5835"    "3735"
##  [64] "364"     "648"     "0"       "7314"    "27.5"    "314"     "148"
##  [71] "1288"    "2237"    "2450"    "307"     "296"     "2234"    "3794"
##  [78] "0"       "0"       "65992.9" "1288"    "1117"    "1129"    "1250"
##  [85] "4853"    "2525"    "28"      "885"     "80"      "40"      "310"
##  [92] "2389"    "539"     "12"      "13676"   "44087"   "121"     "125.5"
##  [99] "400"     "50"      "320"     "350"     "459"     "1771"    "101.9"
## [106] "0"       "823"     "733"     "6.4"     "0"       "4970"    "7000"
## [113] "66526"   "54716"   "2440"    "58"      "1448"    "160"     "273"
## [120] "7600"    "1022"    "124.1"   "29751"   "8"       "70"      "34"
## [127] "26"      "0"       "536"     "3"       "1143"    "2495"    "2413"
## [134] "0"       "499"     "0"       "0"       "498"     "225"     "0"
## [141] "579"     "1770"    "0"       "90"      "0"       "41"      "4828"
## [148] "0"       "4675"    "644"     "0"       "196.8"   "370.4"   "754"
## [155] "177"     "9330"    "6112"    "15"      "0"       "4.1"     "0"
## [162] "293.5"   "40"      "1835"    "2470"    "1572"    "30"      "8"
## [169] "0"       "451"     "2254"    "15134"   "910"     "0"       "853"
## [176] "64"      "32"      "0"       "1482"    "25148"   "2092"    "135663"
## [183] "1046"    "1519"    "14.5"    "2490"    "5152"    "518"     "0"
## [190] "2414"    "36289"   "51"      "440"     "1793"    "501"     "563"
## [197] "225"     "37653"   "0"       "60"      "135"     "158"     "58.9"
## [204] "120"     "84"      "403"     "0"       "209"     "2640"    "531"
## [211] "0"       "491"     "402"     "193"     "58.9"    "0"       "46.6"
## [218] "5313"    "3025"    "2798"    NA        "0"       "17968"   "4964"
## [225] "926"     "1340"    "853"     "386"     "3587"    "3218"    "0"
## [232] "193"     "1566.3"  "0"       "1424"    "3219"    "706"     "56"
## [239] "101"     "419"     "362"     "1148"    "7200"    "0"       "389"
## [246] "24"      "0"       "2782"    "1318"    "12429"   "19924"   "4.8"
## [253] "660"     "0"       "2528"    "2800"    "3444"    "188"     "19.3"
## [260] "129"     "0"       "1110"    "356000"  "1906"    "0"       "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
```

5.5 Datenaufbereitung

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

```
coast %>%
  arrange(coast_num)
## # A tibble: 266 x 3
##    Country     Coastline                                                coast_num
##    <chr>       <chr>                                                        <dbl>
##  1 Afghanistan "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); note - Azerbaijan borde~          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 Fa~ "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          \n          \n\n  \n~   356000
##  2 Canada       "202,080 km\n          \n          \n          \n\n  \n~   202080
##  3 Pacific Oce~ "135,663 km"                                             135663
##  4 Atlantic Oc~ "111,866 km"                                             111866
##  5 Indian Ocean "66,526 km"                                               66526
##  6 European Un~ "65,992.9 km"                                             65993.
##  7 Indonesia    "54,716 km"                                               54716
##  8 Arctic Ocean "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.

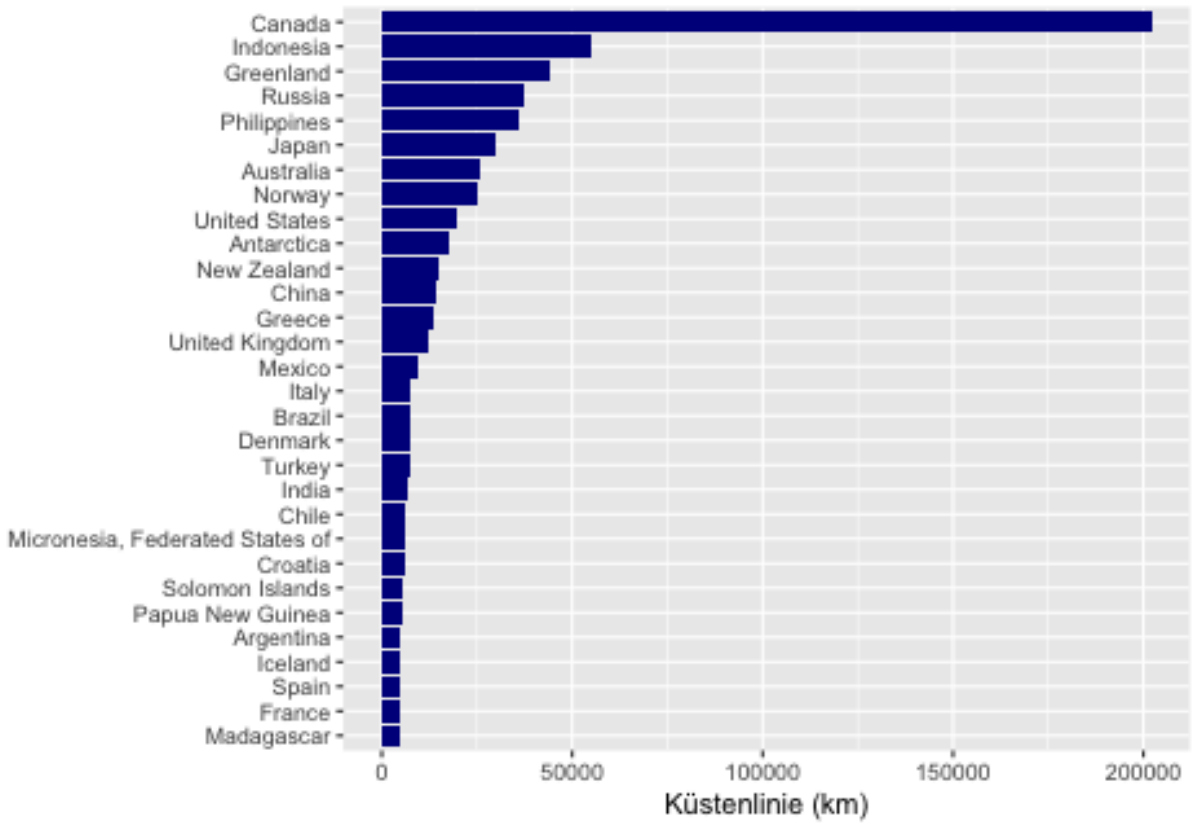
Die Grafik soll nur aus den ersten 30 Einträgen der Tabelle bestehen, welche uns der Befehl `head()` ausgibt.

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

5.6 Datenvisualisierung

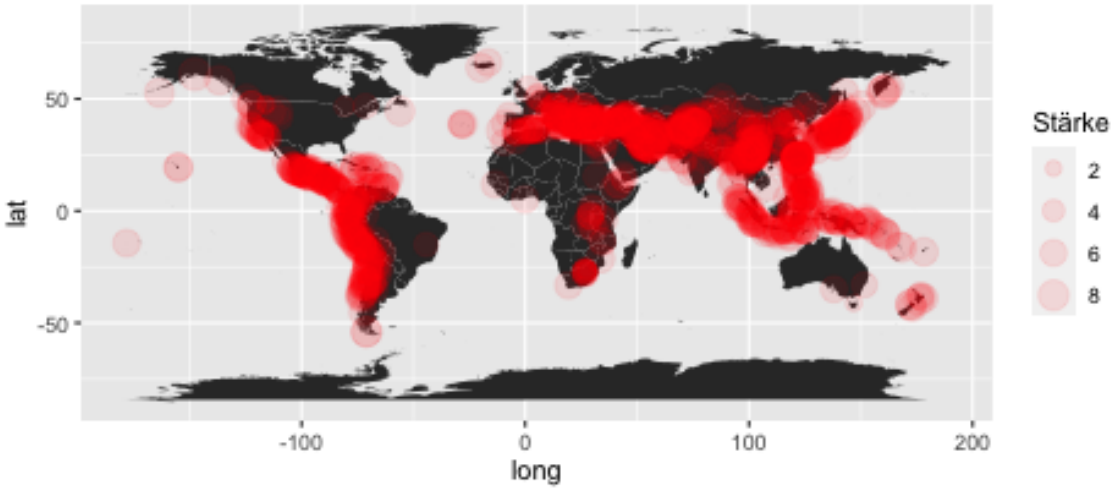
Das Balkendiagramm erhalten wir durch den "ggplot" Befehl. Hierbei gibt es verschiedenste Einstellmöglichkeiten. Wichitg sind vor allem die Angabe des verwendeten Datensatzes und die Art der Grafik (ob Kartendarstellung oder Balkendiagramm). Desweiteren kann man noch Farben der Eigenschaften, eine Achsenbeschriftung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



5.7 Aufgaben

1. Importieren Sie die Daten zu den tödlichen Erdbeben auf Wikipedia und formen sie diese zu einem tibble um.
2. Erstellen Sie mit den erhaltenen Daten eine Karte, welche die Lage und die Stärke der Erdbeben angibt:



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).

6 Geodaten verschneiden

6.1 Lernziele

Sie können…

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

6.2 Vorbereitung

Für diese Lektion werden zwei Pakete geladen:

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

6.3 Ziel

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

6.4 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`):

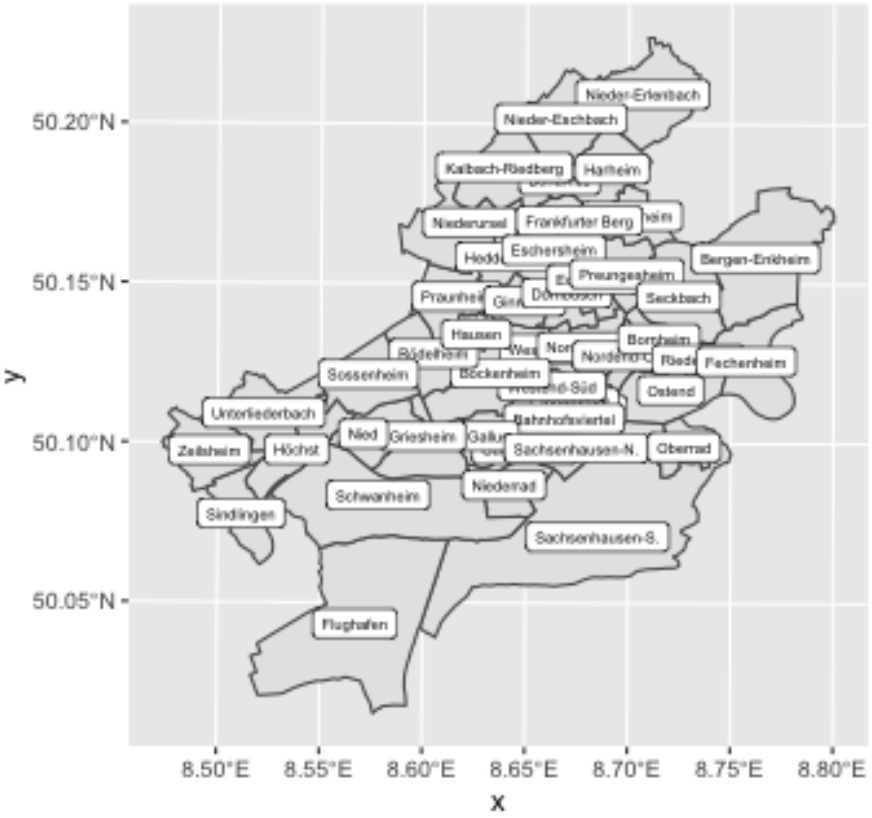
```
stadtteile <- st_read("resources/stadtteile/Stadtteile_Frankfurt_am_Main.shp")
## Reading layer `Stadtteile_Frankfurt_am_Main' from data source
##   `/Users/till/mzs/2021_methodenwoche/skript/resources/stadtteile/Stadtteile_Frankfurt_am_
##   using driver `ESRI Shapefile'
## Simple feature collection with 46 features and 2 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  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: chr  "Altstadt" "Innenstadt" "Bahnhofsviertel" "Westend-Süd" ...
##  $ 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 [1:3] "XY" "POLYGON" "sfg"
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA
##   ..- attr(*, "names")= chr [1:2] "STTLNR" "STTLNAME"
```

Eine Vorschau:

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



6.5 OSM-Daten

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

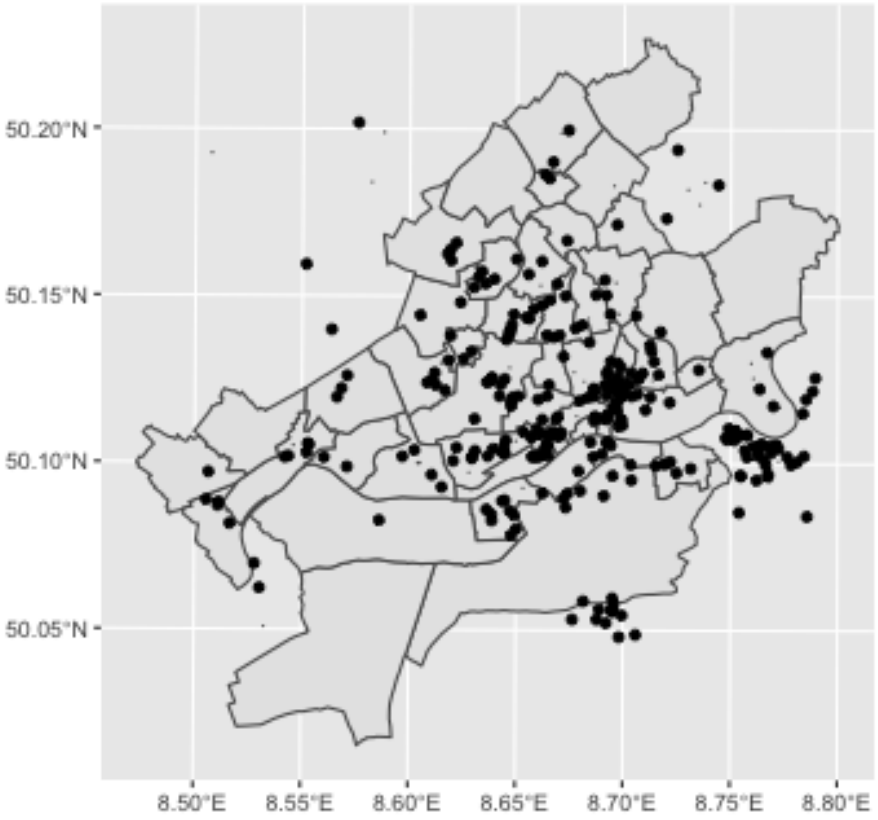
Dann bauen wir auf Overpass Turbo die Abfrage und laden den Datensatz herunter.

Schließlich importieren wir:

```
kioske <- st_read("resources/kioske.geojson")
## Reading layer `kioske' from data source
##   `/Users/till/mzs/2021_methodenwoche/skript/resources/kioske.geojson'
##   using driver `GeoJSON'
## Simple feature collection with 325 features and 74 fields
## Geometry type: GEOMETRY
## Dimension:     XY
## Bounding box:  xmin: 8.505468 ymin: 50.04801 xmax: 8.789538 ymax: 50.20185
## Geodetic CRS:  WGS 84
```

Eine Vorschau:

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



6.6 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 offshor
##         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.

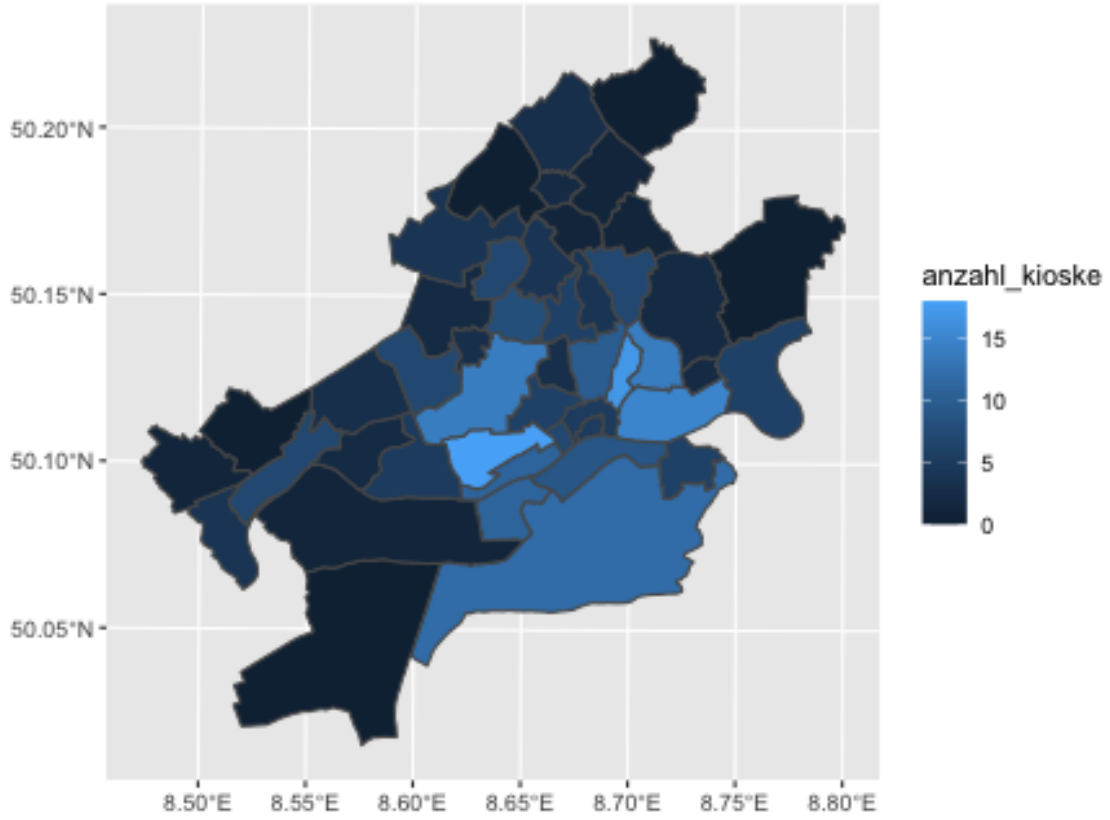
6.7 Verschneiden

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
## Bounding box:  xmin: 462292.7 ymin: 5540412 xmax: 485744.8 ymax: 5563925
## Projected CRS: ETRS89 / UTM zone 32N
## First 10 features:
##    STTLNR        STTLNAME                       geometry anzahl_kioske
## 1       1        Altstadt POLYGON ((476934.3 5550541,...             5
## 2       2      Innenstadt POLYGON ((477611.9 5552034,...             5
## 3       3 Bahnhofsviertel POLYGON ((475831 5550785, 4...             7
## 4       4     Westend-Süd POLYGON ((475745.4 5552373,...             6
## 5       5    Westend-Nord POLYGON ((476497.9 5553910,...             3
## 6       6    Nordend-West POLYGON ((478362.5 5553898,...            10
## 7       7     Nordend-Ost POLYGON ((478397.9 5551924,...            17
## 8       8          Ostend POLYGON ((481955.2 5552141,...            15
```

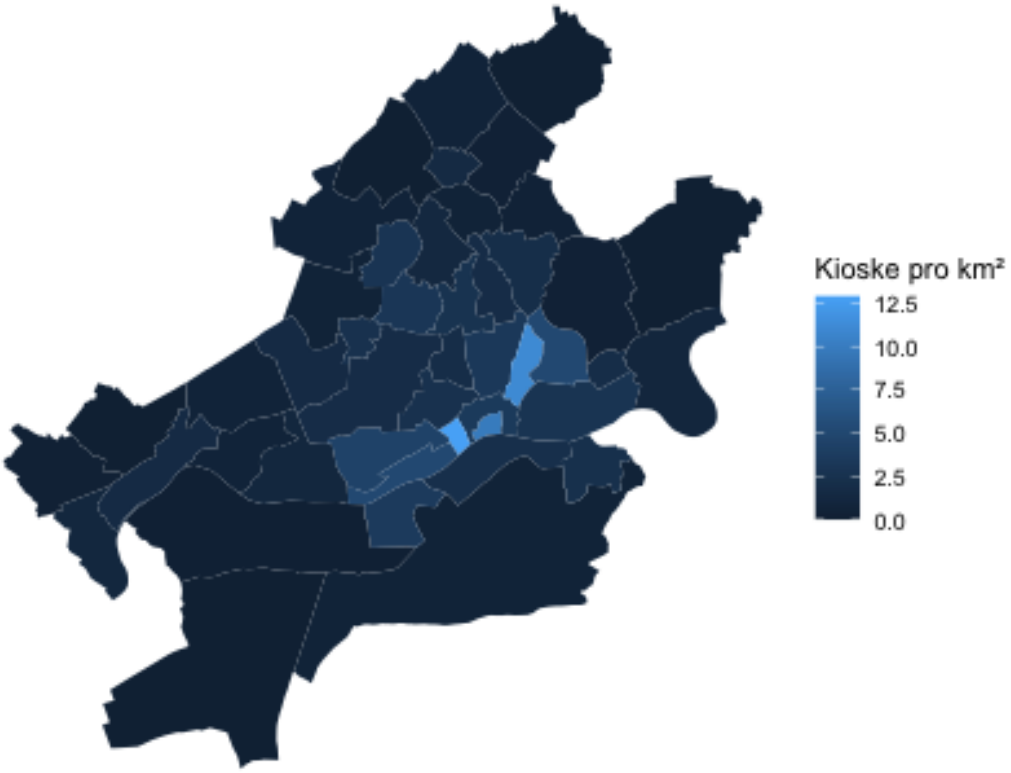
```
## 9        9        Bornheim POLYGON ((478959.8 5552336,...            14
## 10      10  Gutleutviertel POLYGON ((472942 5548802, 4...            11
##           qkm
## 1   0.5065673
## 2   1.4902009
## 3   0.5425421
## 4   2.4948957
## 5   1.6307925
## 6   3.0977694
## 7   1.5305338
## 8   5.5573382
## 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
## Bounding box:  xmin: 462292.7 ymin: 5540412 xmax: 485744.8 ymax: 5563925
## Projected CRS: ETRS89 / UTM zone 32N
## First 10 features:
##    STTLNR        STTLNAME                       geometry anzahl_kioske
## 1       1        Altstadt POLYGON ((476934.3 5550541,...             5
## 2       2      Innenstadt POLYGON ((477611.9 5552034,...             5
## 3       3 Bahnhofsviertel POLYGON ((475831 5550785, 4...             7
## 4       4     Westend-Süd POLYGON ((475745.4 5552373,...             6
## 5       5    Westend-Nord POLYGON ((476497.9 5553910,...             3
## 6       6    Nordend-West POLYGON ((478362.5 5553898,...            10
## 7       7     Nordend-Ost POLYGON ((478397.9 5551924,...            17
## 8       8          Ostend POLYGON ((481955.2 5552141,...            15
## 9       9        Bornheim POLYGON ((478959.8 5552336,...            14
## 10     10  Gutleutviertel 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
```

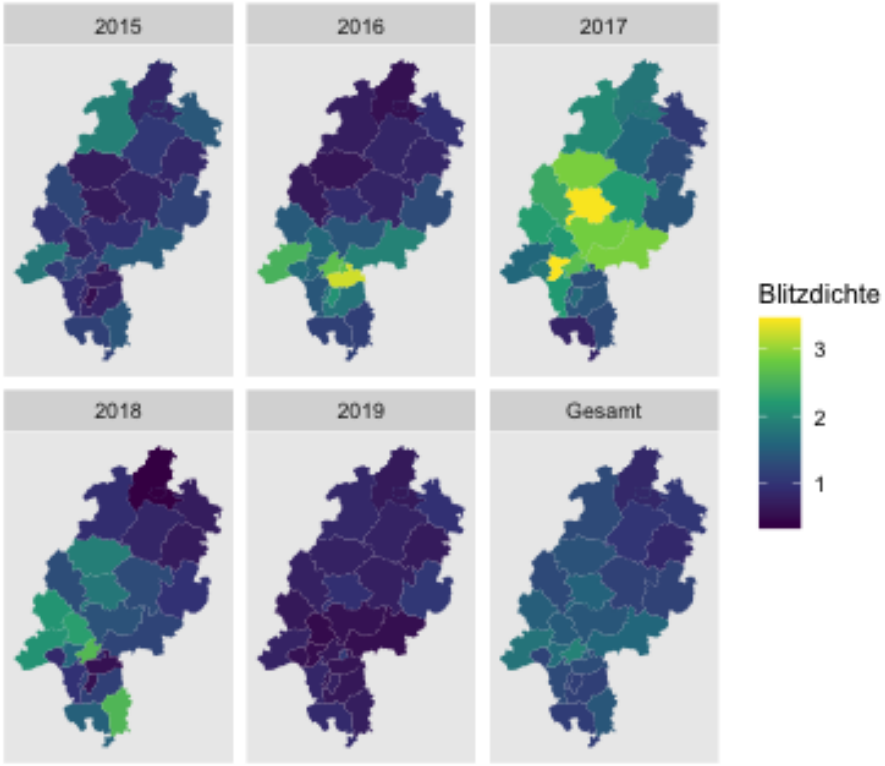
Schließlich die Karte:

```
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()
```



6.8 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:



7 Weitere Methoden

7.1 Vorbereitung

Für diese Lektion brauchen wir folgende Pakete:

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

7.2 Aufgabe

Ziel soll sein, eine Deutschlandkarte mit Tankstellenpreisen für Diesel zu erstellen.

7.3 Daten einlesen

Das “Tankerkönig”-Projekt veröffentlicht aktuelle Tankstellenpreise über eine API, und stellt historische Preise hier bereit: https://dev.azure.com/tankerkoenig/_git/tankerkoenig-data

Wir laden die Dateien für `prices` und `stations` von einem Tag (hier: 26.5.2019) herunter und speichern sie im Unterordner `resources` des Projektordners.

Dann können wir sie einlesen:

```
preise <- read_csv("resources/2019-05-26-prices.csv")
preise
```

```
## # A tibble: 231,174 x 8
##    date                station_uuid     diesel    e5   e10 dieselchange e5change
##    <dttm>              <chr>             <dbl> <dbl> <dbl>        <dbl>    <dbl>
##  1 2019-05-25 22:01:06 51e171d0-1a9c-4~   1.37  1.58  1.56            1        1
##  2 2019-05-25 22:02:06 8a796af1-8d78-4~   1.32  1.56  1.54            1        1
##  3 2019-05-25 22:02:06 2d658127-11b5-4~   1.28  1.52  0               0        1
##  4 2019-05-25 22:03:06 904d3a45-df30-4~   1.32  1.54  1.52            1        1
##  5 2019-05-25 22:03:06 a98ed5d0-261b-4~   1.34  1.57  1.55            1        1
##  6 2019-05-25 22:04:06 7671d5ad-4c7d-4~   1.30  1.54  0               1        1
##  7 2019-05-25 22:04:06 44fa4d12-5571-4~   1.34  1.58  1.56            1        1
##  8 2019-05-25 22:04:06 da9abcda-3218-4~   1.38  1.52  1.50            0        1
##  9 2019-05-25 22:04:06 bcba0c2b-fbe7-4~   1.35  1.55  1.53            0        1
## 10 2019-05-25 22:04:06 00061000-0001-4~   1.20  1.48  1.46            1        1
## # ... with 231,164 more rows, and 1 more variable: e10change <dbl>
```

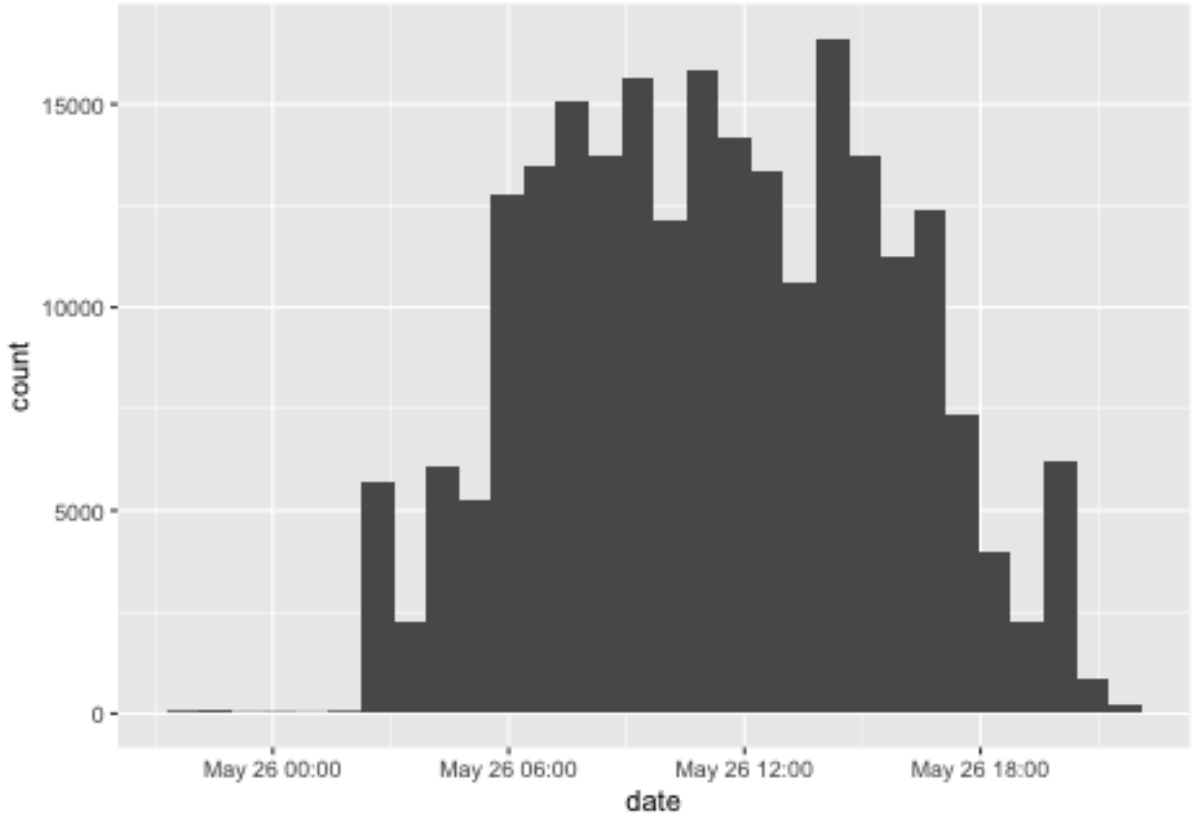
7.4 Überblick verschaffen

Beim nähreren Betrachten fällt auf, dass im Datensatz `preise` 231.174 Zeilen enthalten sind, in `stations` nur 15.668. Das liegt daran, das für jede Station *mehrere* Preisupdates im Datensatz `preise` stehen, jedoch nur *einmal* die gleichbleibenden Informationen (Name, Marke, Adresse, Koordinaten) in `stations`.

Beide Datensätze sind über einen eindeutigen „Key“ verbunden: In `preise` heißt er `station_uuid`, in `stations` einfach nur `uuid`.

Um ein besseres Gefühl für den Datensatz zu bekommen, könnten wir uns z. B. anschauen, zu welcher Uhrzeit wie viele Preise aktualisiert wurden:

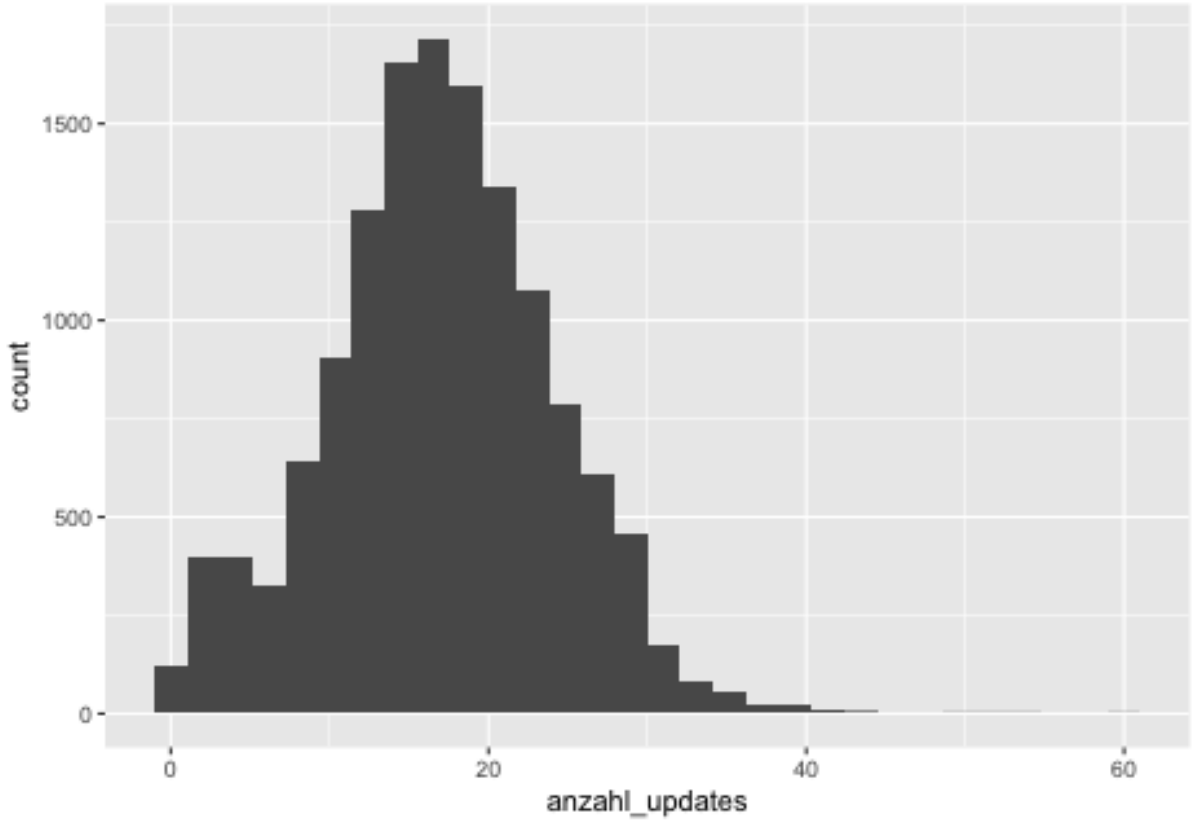
```
ggplot(preise) +
  geom_histogram(aes(x = date))
```



7.5 Zusammenfassen

Eine weitere Frage könnte sein: Wie sieht die Verteilung der Anzahl der Preisupdates je Tankstelle aus? Hierfür müssen wir den Datensatz `preise` anhand der Spalte `station_uuid` zusammenfassen und die Einträge zählen. Das geht mit `group_by()` und `summarize()`:

```
preise %>%
  group_by(station_uuid) %>%
  summarize(anzahl_updates = n()) %>%
  ggplot() +
    geom_histogram(aes(x = anzahl_updates))
```



Um unserem Ziel der Dieselkarte etwas näher zu kommen, sollten wir aber nicht die Anzahl der Updates zusammenfassen, sondern den Dieselpreis. Aber nach welchem Schema? Einfach nur den Durchschnitt (mit `mean()`) zu nehmen, könnte das Bild verfälschen: Man stelle sich z. B. vor, ein besonders teurer (oder günstiger) Preis sei nur wenige Sekunden gültig gewesen.

Wir orientieren uns einfach an der Börse und nehmen einfach den letzten gültigen Preis (wie der Aktienwert bei Börsenschluss). Dafür müssen wir den Datensatz erst mit `arrange()` chronologisch sortieren, dann entsprechend gruppieren und mit `last()` zusammenfassen:

```
preise %>%
  arrange(date) %>%
  group_by(station_uuid) %>%
  summarize(dieselpreis = last(diesel),
            e5preis     = last(e5),
            e10preis    = last(e10)) ->
  preise_nach_tankstelle

preise_nach_tankstelle
## # A tibble: 13,701 x 4
##    station_uuid                         dieselpreis e5preis e10preis
##    <chr>                                      <dbl>   <dbl>    <dbl>
##  1 00006210-0037-4444-8888-acdc00006210        1.33    1.55     1.53
##  2 00016899-3247-4444-8888-acdc00000007        1.31    1.53     1.51
##  3 00060001-d387-4444-8888-acdc00000001        1.37    1.62     1.60
##  4 00060009-3adf-4444-8888-acdc00000001        1.35    1.64     1.62
```

```
##  5 00060014-b0d9-4444-8888-acdc00000002        1.32    1.61     1.59
##  6 00060015-0090-4444-8888-acdc00000090        1.27    1.52     1.50
##  7 00060016-ed96-4444-8888-acdc00000001        1.35    1.57     1.55
##  8 00060034-0011-4444-8888-acdc00000011        1.37    1.61     1.59
##  9 00060051-533e-75a1-87f9-8a9f00060051        1.25    1.50     1.48
## 10 00060055-0001-4444-8888-acdc00000001        1.26    1.53     1.51
## # ... with 13,691 more rows
```

7.6 Verschneiden

Jetzt haben wir für jede Station nur noch eine Zeile mit den Preisen. Um das zu kartieren, fehlen noch die Informationen zu den Tankstellen. Dafür laden wir auch den `stations`-Datensatz für den richtigen Tag herunter und importieren ihn in R:

```
tankstellen <- read_csv("resources/2019-05-26-stations.csv")
tankstellen
```

```
## # A tibble: 15,668 x 11
##    uuid    name    brand  street house_number post_code city  latitude longitude
##    <chr>   <chr>   <chr>  <chr>  <chr>        <chr>     <chr>    <dbl>     <dbl>
##  1 0e18d0~ OIL! T~ OIL!   Evers~ <NA>         80999     Münc~     48.2     11.5
##  2 ad8122~ bft Bo~ bft    Godes~ 55           53175     Bonn      50.7      7.14
##  3 44e2bd~ bft Ta~ <NA>   Schel~ 53           36304     Alsf~     50.8      9.28
##  4 1a8e4d~ Hessol  Hessol Frank~ 65           61279     Gräv~     50.4      8.46
##  5 005056~ star T~ STAR   Leipz~ 11           06217     Mers~     51.4     12.0
##  6 d435f7~ ROSDOR~ Shell  A7 GÖ~ <NA>         37124     Rosd~     51.5      9.88
##  7 88a23d~ AVIA T~ AVIA   Burgs~ 8            63637     Joss~     50.2      9.48
##  8 f0e93f~ Aral T~ ARAL   Eicke~ 357          41063     Mönc~     51.2      6.45
##  9 005056~ star T~ STAR   Celle~ 55           29303     Berg~     52.8      9.97
## 10 8e47dd~ Aral T~ ARAL   Crail~ 32           74532     Ilsh~     49.2      9.93
## # ... with 15,658 more rows, and 2 more variables: first_active <dttm>,
## #   openingtimes_json <chr>
```

Wir verschneiden mit `inner_join()` unter Angabe der relevanten Spaltennamen und wählen die Spalten aus, mit denen wir weiterarbeiten wollen:

```
inner_join(preise_nach_tankstelle, tankstellen,
           by = c("station_uuid" = "uuid")) %>%
  select(dieselpreis, e5preis, e10preis, name, brand, latitude, longitude) ->
  preise_geo

preise_geo
```

```
## # A tibble: 13,700 x 7
##    dieselpreis e5preis e10preis name              brand       latitude longitude
##          <dbl>   <dbl>    <dbl> <chr>             <chr>          <dbl>     <dbl>
##  1        1.33    1.55     1.53 Beducker - Quali~ Beducker        48.6     10.9
##  2        1.31    1.53     1.51 Röttenbach        BFT Pickel~     49.7     10.9
##  3        1.37    1.62     1.60 Haisch Mineralöl~ TankCenter~     48.0      7.59
##  4        1.35    1.64     1.62 Tank-Kontor Wilh~ <NA>            47.9      9.42
##  5        1.32    1.61     1.59 Tank-Kontor Baie~ <NA>            47.8      9.65
```

```
##  6        1.27    1.52     1.50 Schindele, Lochb~ <NA>            47.7      9.53
##  7        1.35    1.57     1.55 bft-Tankstelle H~ BFT             48.1      7.78
##  8        1.37    1.61     1.59 EXTROL Tank- & W~ EXTROL          48.0      7.79
##  9        1.25    1.50     1.48 Wingenfeld Energ~ Wingenfeld~     50.8     10.2
## 10        1.26    1.53     1.51 Wilhlem Heim GmbH Oel - Heim      48.6      9.03
## # ... with 13,690 more rows
```

`inner_join` hat die Besonderheit, dass nur Zeilen im kombinierten Datensatz übrigbleiben, deren Key in *beiden* Datensätzen gefunden wurde. Mit `left_join` würden hier alle Preise behalten werden (und die fehlenden Koordinaten mit `NA` ergänzt), mit `right_join` würden alle Stationen behalten werden (und fehlende Preise mit `NA` ergänzt). `full_join` löscht gar keine Informationen.

7.7 Kartieren

Den georeferenzierten Datensatz der Preise wandeln wir in eine Simple Feature Collection um:

```
preise_geo %>%
  st_as_sf(coords = c("longitude", "latitude")) -> preise_sf
```

`ggplot()` kartiert so einen großen Datensatz nur langsam. Wir nehmen stattdessen das Paket `tmap()` zur Hand, das mit einer ähnlichen Grammatik funktioniert.

Interaktive Karten lassen sich mit `tmap` produzieren, wenn die Option

```
tmap_mode("view")
```

gesetzt ist. Aus technischen Gründen wird an dieser Stelle im Skript darauf verzichtet und wir bleiben beim `plot`-Modus:

```
tm_shape(preise_sf) +
  tm_dots()
```

Zwei Koordinaten sind quatsch! Wir finden ihre ungefähren Werte mit `summary`:

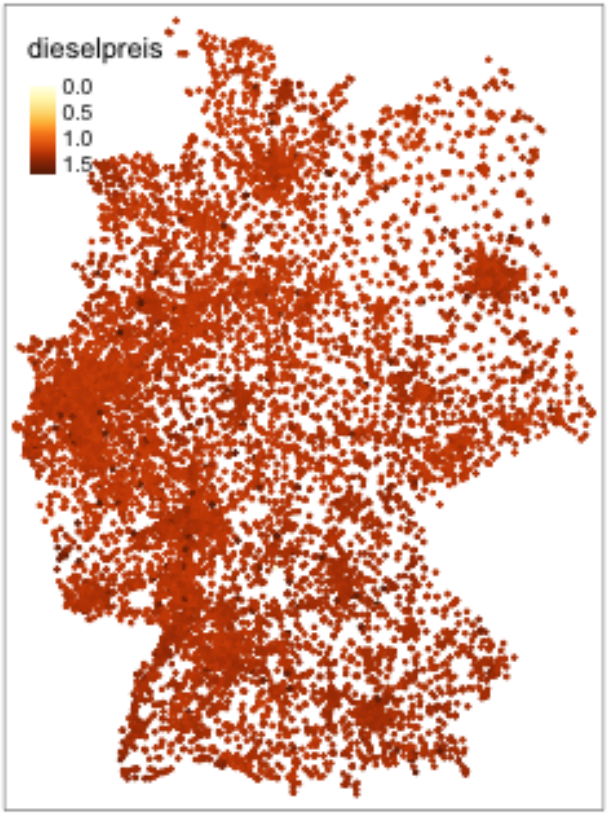
```
summary(preise_geo$longitude)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   5.901   8.021   9.275   9.607  11.058  97.364
```

Und filtern sie raus, und wiederholen die Umwandlung (diesmal auch mit CRS)

```
preise_geo %>%
  filter(longitude < 80) %>%
  st_as_sf(coords = c("longitude", "latitude")) %>%
  st_set_crs(4326) -> preise_sf
```

Dann mappen wir nochmal:

```
tm_shape(preise_sf) +
  tm_dots("dieselpreis", style = "cont")
```



Schon ganz hübsch, aber die Skala wird nun verzerrt durch sehr teure Autobahntankstellen einerseits, und falsche Null-werte andererseits:

```
summary(preise_sf$dieselpreis)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   1.249   1.289   1.290   1.329   1.679
```

7.8 Choroplethen

Eine Lösung wäre, die Daten auf Kreisebene zusammenzufassen, und zwar anhand ihres Medians. Damit würden diese Ausreißer keine Role mehr spielen.

Das `eurostat`-Paket macht es einfach, diese Geodaten einzulesen. NUTS3 ist die Ebene der Stadt- und Landkreise bzw. ihrer europäischen Equivalente.

```
kreise <- eurostat::get_eurostat_geospatial(nuts_level = 3) %>%
  filter(CNTR_CODE == "DE")
```

Mal schauen wie es aussieht:

```
tm_shape(kreise) +
  tm_polygons()
```

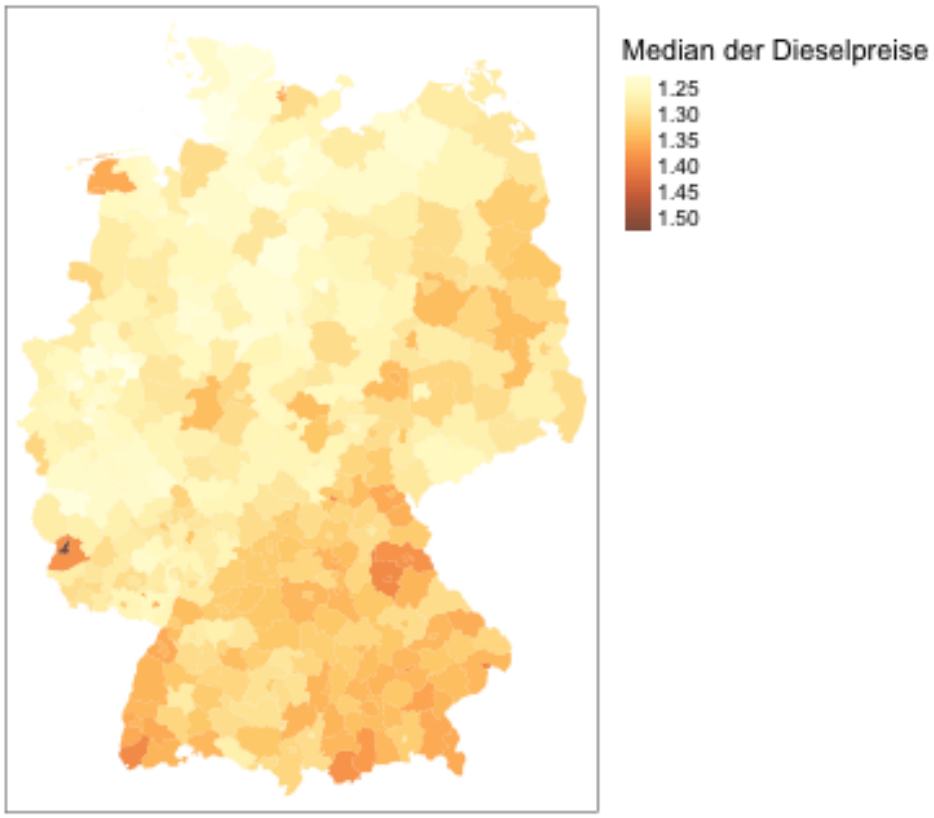
7.9 Räumliches Verschneiden

mit `st_join` werden Datensätze nicht mit einem Key verschnitten, sondern anhand ihrer Geolokation. Dann können wir wieder ganz normal `group_by` und `summarise` verwenden:

```
st_join(kreise, preise_sf) %>%
  group_by(NUTS_ID) %>%
  summarise(dieselpreis = median(dieselpreis),
            e5preis     = median(e5preis),
            e10preis    = median(e10preis)) -> preise_kreise
```

Und so könnte vielleicht ein vorläufiges Ergebnis aussehen:

```
tm_shape(preise_kreise) +
  tm_fill("dieselpreis",
          title = "Median der Dieselpreise",
          style = "cont",
          alpha = 0.8) +
  tm_layout(legend.outside = TRUE)
```



8 Publizieren und nach Hilfe fragen

8.1 Publizieren mit Rmarkdown

8.1.1 Text formatieren

Wir arbeiten schon von Anfang an mit im Rmarkdown-Format. Wie Überschriften, Links, Bilder usw. in Rmarkdown genau funktionieren, ist in dieser Übersicht und auf diesem Cheat Sheet gut festgehalten.

8.1.2 Der Knit-Button

Wenn wir im YAML-Header die Zeile

```
output: html_document
```

setzen, erscheint (nach Abspeichern) ein „Knit“-Button in der GUI. Durch Drücken auf diesen Knopf passiert folgendes:

- R erstellt („strickt“) ein HTML-Dokument aus dem vorliegenden Markdown und den Code Chunks
- Dabei spielen „gespeicherte“ Objekte keine Rolle, die Chunks werden einfach der Reihe nach (in einem neuen Environment) ausgeführt
- Externe Datensätze o. ä. müssen also am Anfang des Dokuments explizit geladen werden (etwa mit `load()`). Für den Abschlussbericht ist es ratsam, einen vorbereiteten Datensatz am Anfang des Rmarkdown-Dokuments so zu laden.

Im YAML-Header können noch viele weitere Angaben gemacht werden, die das Resultat verändern. Hier eine gute Dokumentation: https://bookdown.org/yihui/rmarkdown/html-document.html

8.1.3 Kable

Im `knitr`-Paket sorgt der Befehl `kable()` für eine schöne Darstellung von Tabellen:

```
ggplot2::diamonds %>%
  head() %>%
  knitr::kable()
```

carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
0.24	Very Good	J	VVS2	62.8	57	336	3.94	3.96	2.48

Auch hier gibt es wieder vielfältige Möglichkeiten zur visuellen Gestaltung. Diesen Post finde ich immer besonders hilfreich: https://haozhu233.github.io/kableExtra/awesome_table_in_html.html

8.1.4 Chunk Options

Am Anfang eines Code Chunks kann genau festgelegt werden, ob der Code ausgeführt werden soll, ob er im finalen Dokument erscheinen soll, ob Warnungen oder Fehler ausgegeben werden sollen, etc. Wenn zum Beispiel Libraries „versteckt“ geladen werden sollen, geht das mit diesem Code Chunk:

````
```{r, include = FALSE}
library(tidyverse)
library(rvest)
```
````

Ein überblick über die Chunk-Optionen findet sich hier: https://bookdown.org/yihui/rmarkdown/r-code.html

8.1.5 Reproducible research

Wenn z. B. ein (längeres) Script einen (größeren) Datensatz generiert und ihn dem Objektnamen `mein_datensatz` zuweist, dann erscheint der Datensatz im lokalen Environment. Um den Datensatz mit anderen zu teilen, muss er irgendwie exportiert werden. Hierfür ist es ratsam, die `save()`-Funktion zu nutzen:

```
save(mein_datensatz, file = "zwischenstand_mein_datensatz.Rdata")
```

So wird eine Datei erstellt, die den Datensatz enthält und verschoben oder geteilt werden kann. Eine solche Datei kann auch andere Objekte (Funktionen, Listen, …) und mehrere Objekte auf einmal enthalten. Die Dateiendung ist dabei eigentlich egal, `.Rdata` scheint aber Usus zu sein.

Aus der Datei können Objekte dann jederzeit wieder ins Environment geladen werden mit dem Befehl:

```
load("zwischenstand_mein_datensatz.Rdata")
```

Auch wenn ein Skript auf einmal nicht mehr funktioniert (etwa weil die API sich ändert), ist es hilfreich, auf solche Zwischenstände zurückgreifen zu können.

8.1.6 Wissenschaftliches Zitieren

Wer die Vorzüge von Literaturverwaltungssoftware (wie Citavi, Zotero, …) schon schätzen gelernt hat, kann in Rmarkdown folgendermaßen vorgehen:

8.1.6.1 Schritt 1: Exportieren

Die relevante Literatur in eine BibTex-Datei im R-Arbeitsverzeichnis exportieren, z.B. `literatur.bib`. BibTex (bzw. BibLatex) ist ein bewährtes und gut dokumentiertes Format. Ein Eintrag sieht dann z.B. so aus, wobei `bortz` der „Name“ des Eintrags ist, den wir frei wählen können:

```
@book{bortz,
  author = {Bortz, J{\"u}rgen and Schuster, Christof},
  title = {{Statistik f{\"u}r Human- und Sozialwissenschaftler}},
  publisher = {Springer},
  year = {2010},
  address = {Berlin},
  edition = {7},
}
```

8.1.6.2 Schritt 2: Verlinken

Im YAML-Header des Rmarkdown-dokuments die Angabe ergänzen:

```
bibliography: literatur.bib
```

Damit weiß der „Knit“-Befehl, wo er nach Literatur suchen soll.

8.1.6.3 Schritt 3: Zitieren

Im Text kann dann z. B. so zitiert werden:

```
Das zentrale Grenzwerttheorem besagt, dass die Stichprobenverteilung von
$\bar{x}$ mit steigender Stichprobengröße $n$ in eine Normalverteilung übergeht
[@bortz: 86].
```

8.1.6.4 Schritt 4: Stricken

Beim „Knit“-Befehl wird ein Literaturverzeichnis automatisch erstellt und ans Ende des Dokuments gehängt. Deshalb beendet man das Dokument am besten mit der Zeile:

```
## Literaturverzeichnis
```

8.2 Nach Hilfe fragen

Online-Foren, insb. Stackoverflow sind tolle Ressourcen um Probleme beim Programmieren zu lösen.

8.2.1 Gute Beispiele