

Data Science für die Humangeographie: Ein pragmatischer Einstieg mit R

Konzeption quantitativer Forschung

Till Straube
[straube@geo.uni-frankfurt.de](mailto:strabe@geo.uni-frankfurt.de)

Wintersemester 2020/21

Institut für Humangeographie
Goethe-Universität Frankfurt

Inhaltsverzeichnis

Terminüberblick	2
Online-Ressourcen	2
R Tutorials und eBooks	2
Inspiration für Visualisierungen	3
Spezialthemen	3
1 Vorbesprechung	3
1.1 Überblick	3
1.2 Seminarformat	4
1.3 Leistungsnachweise	5
1.4 Lehrphilosophie	6
2 Erste Schritte	7
2.1 Vorbereitung	7
2.2 Lernziele für diese Sitzung	7
2.3 Operatoren	7
2.4 Variablen	7
2.5 Konstanten	8
2.6 Funktionen	8
2.7 Strings	9
2.8 Datentypen	10
2.9 Aufgaben	10
3 Text: Anderson 2008	12
3.1 Lesetext	12
3.2 Fragen an den Text	12
4 Datenstrukturen	12
4.1 Lernziele dieser Sitzung	12
4.2 Vektoren	12

4.3	Matritzen	14
4.4	Listen	15
4.5	Data Frames	15
4.6	Tibbles	16
4.7	Aufgaben	16
5	Visualisierungen	18
5.1	Voraussetzungen	18
5.2	Überblick	18
5.3	Visualisierung mit dem Standardpaket	19
5.4	Visualisierung mit <code>ggplot()</code>	20
5.5	Aufgaben	28

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	9	Choroplethen
18. Januar 2021	10	Text: Chandra 2014
25. Januar 2021	11	HTML-Tabellen
1. Februar 2021	12	Web Scraping
8. Februar 2021	13	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/>

Spezialthemen

- **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 unterricht 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 unterricht 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.
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: FUNKTIONSSNAME(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, nimmt 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 Beispiele. 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!")
```

```
## Warning: NAs introduced by coercion
```

```
## [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 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 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 Good J      VVS2    62.8    57    336  3.94
## 7 0.24 Very Good I      VVS1    62.3    57    336  3.95
## 8 0.26 Very Good H      SI1     61.9    55    337  4.07
## 9 0.22 Fair       E      VS2     65.1    61    337  3.87
## 10 0.23 Very Good 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)
```

5.2 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 0.24 0.26 0.22 ...
## # $ 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:

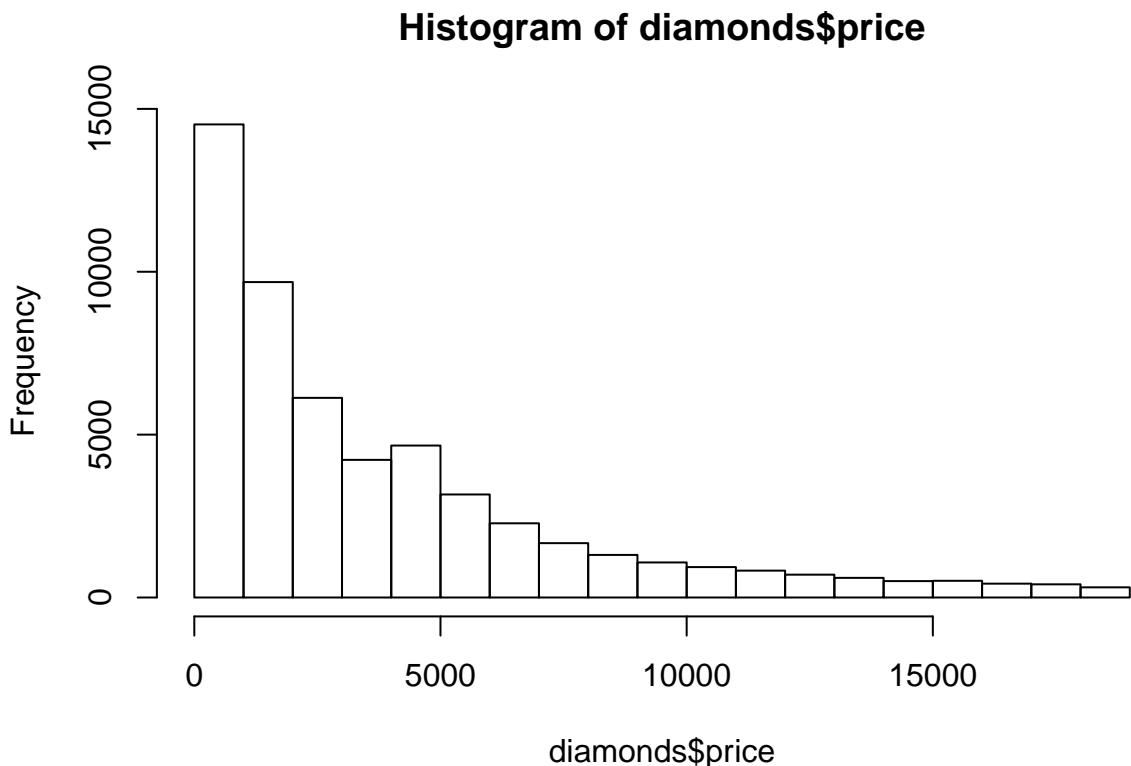
```
summary(diamonds)
```

```
##      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
##
```

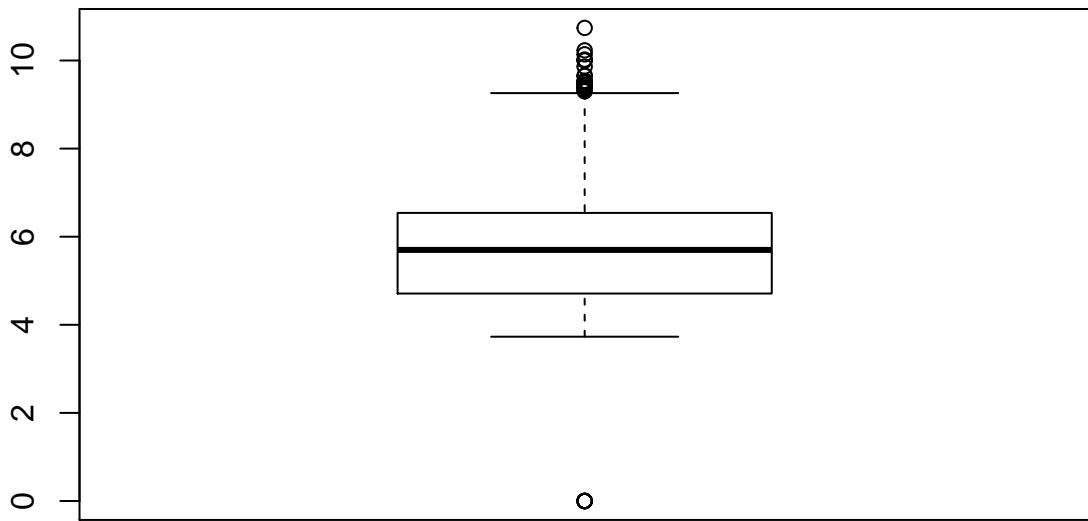
5.3 Visualisierung mit dem Standardpaket

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)
```



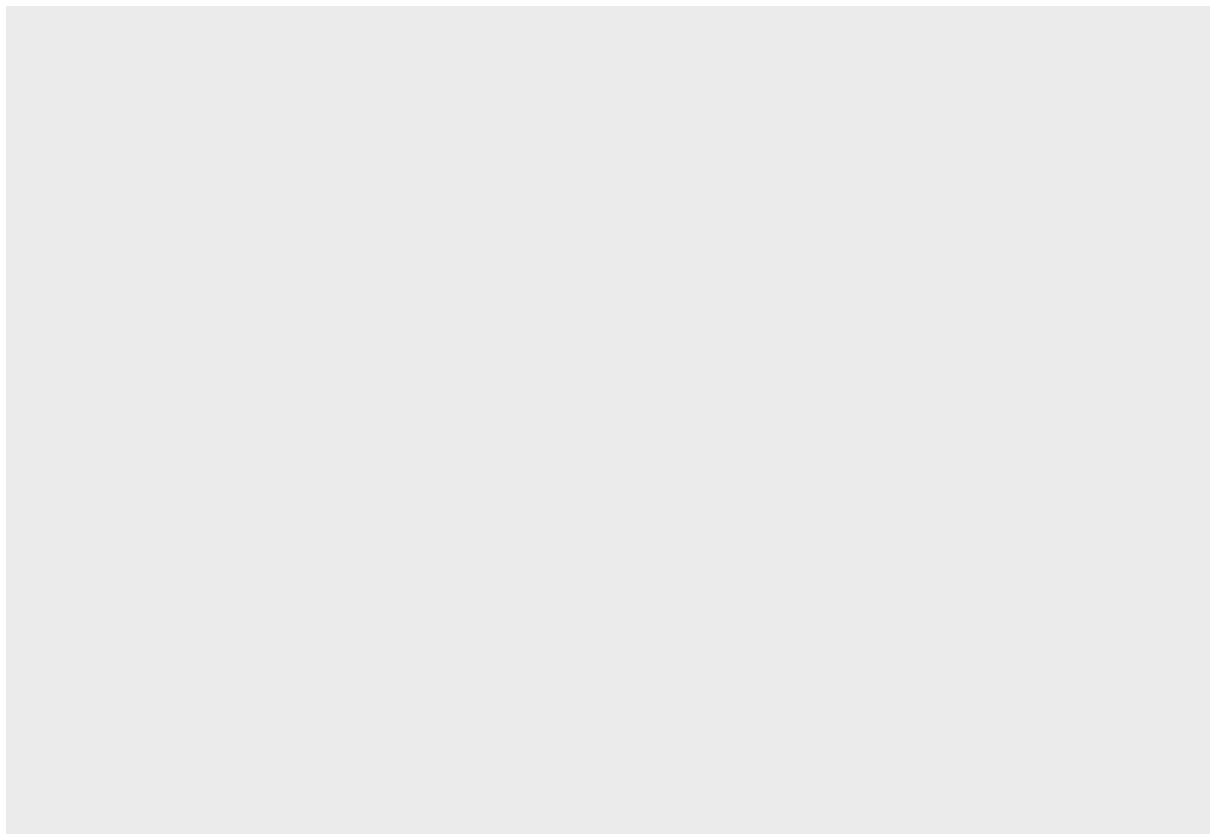
5.4 Visualisierung mit ggplot()

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.

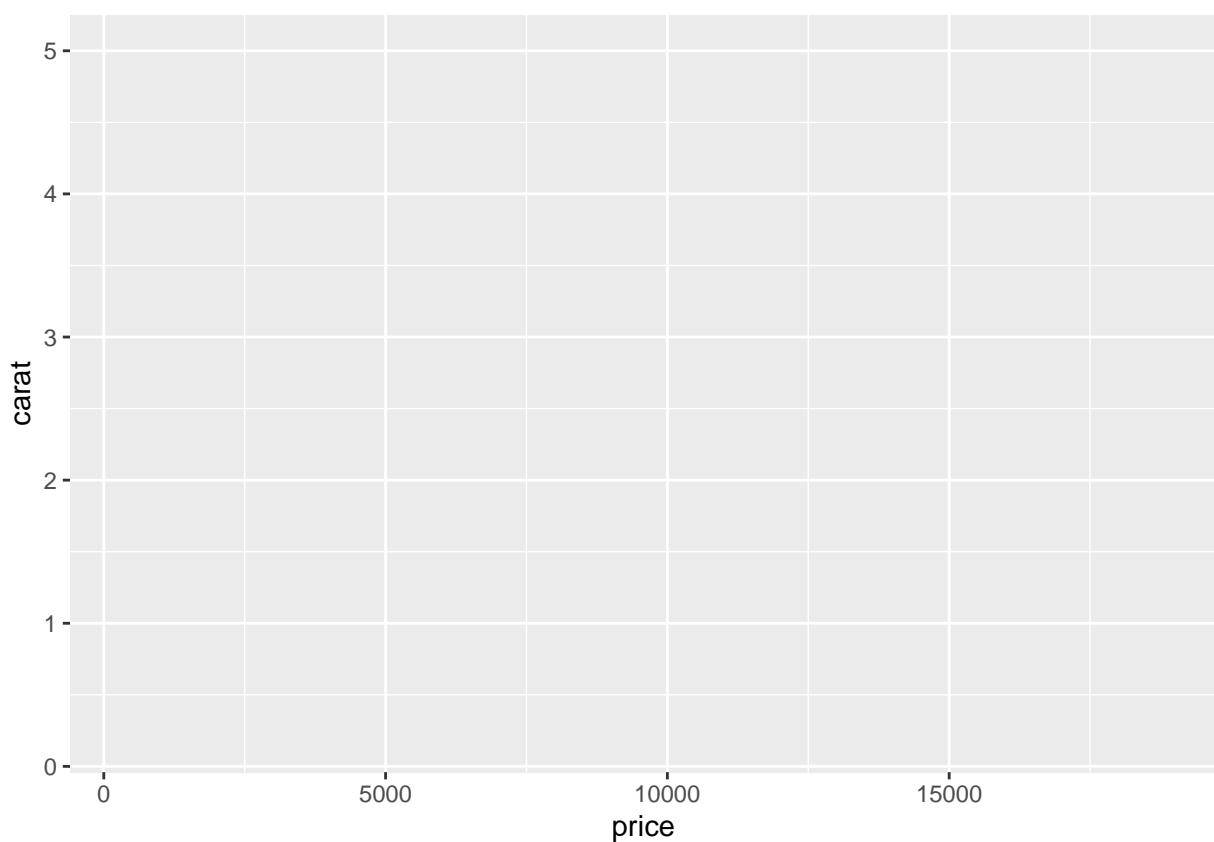
Am Anfang steht der Befehl `ggplot(x)` mit dem Datensatz als Parameter

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



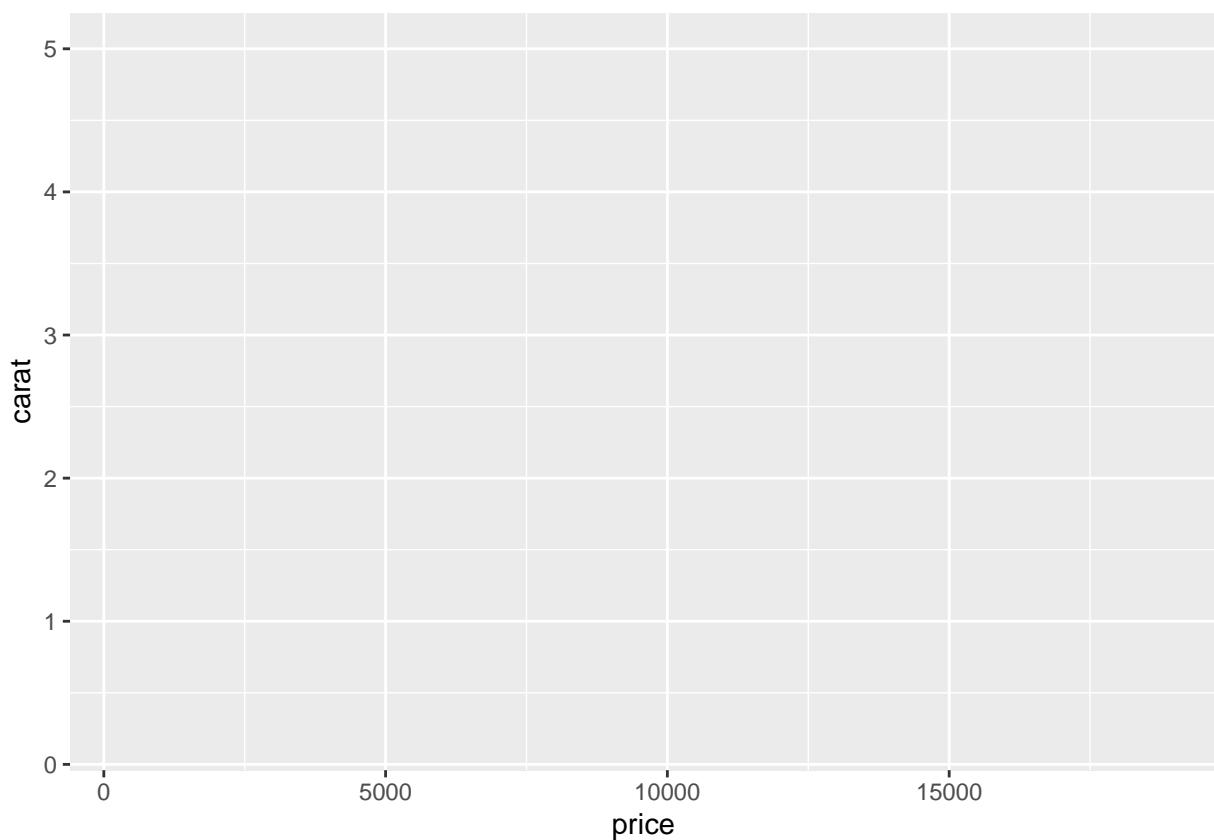
Mit einem Mapping-Parameter legen wir die Dimensionen fest:

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



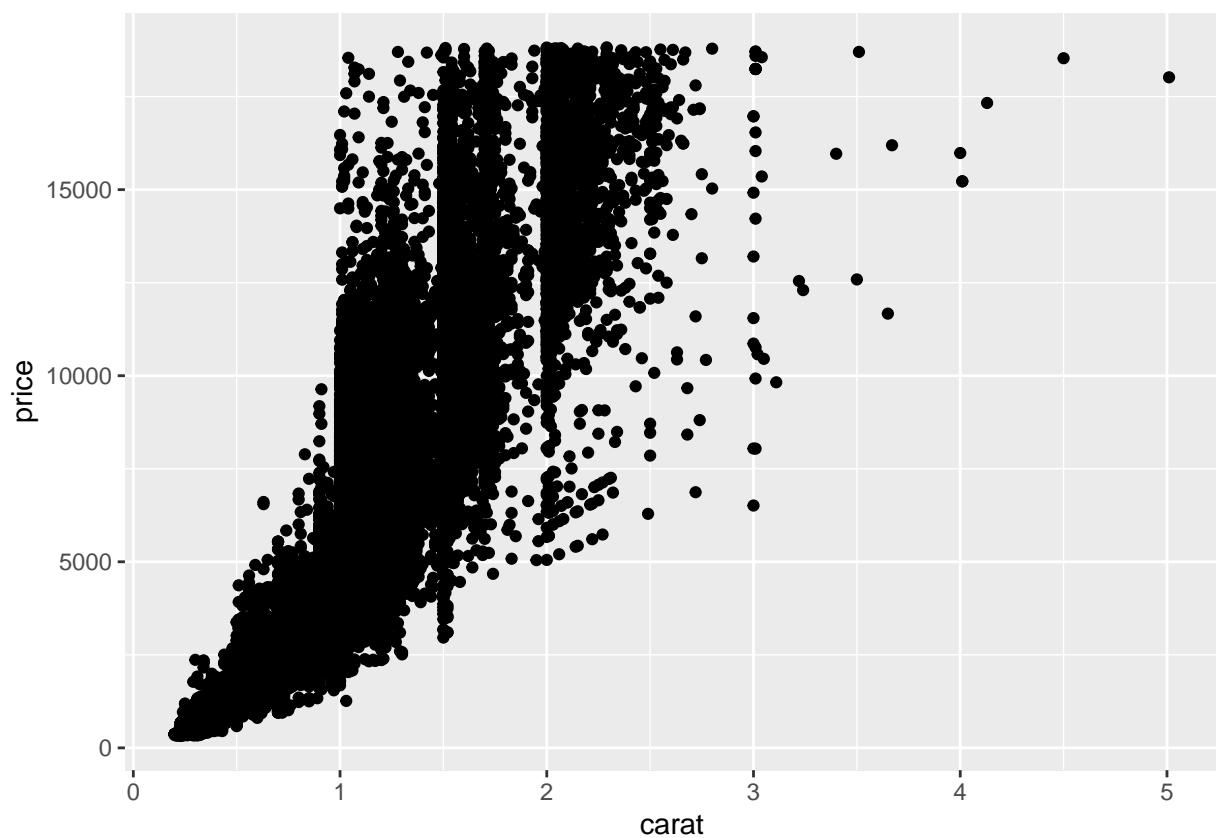
Das gleiche ohne Parameternamen:

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



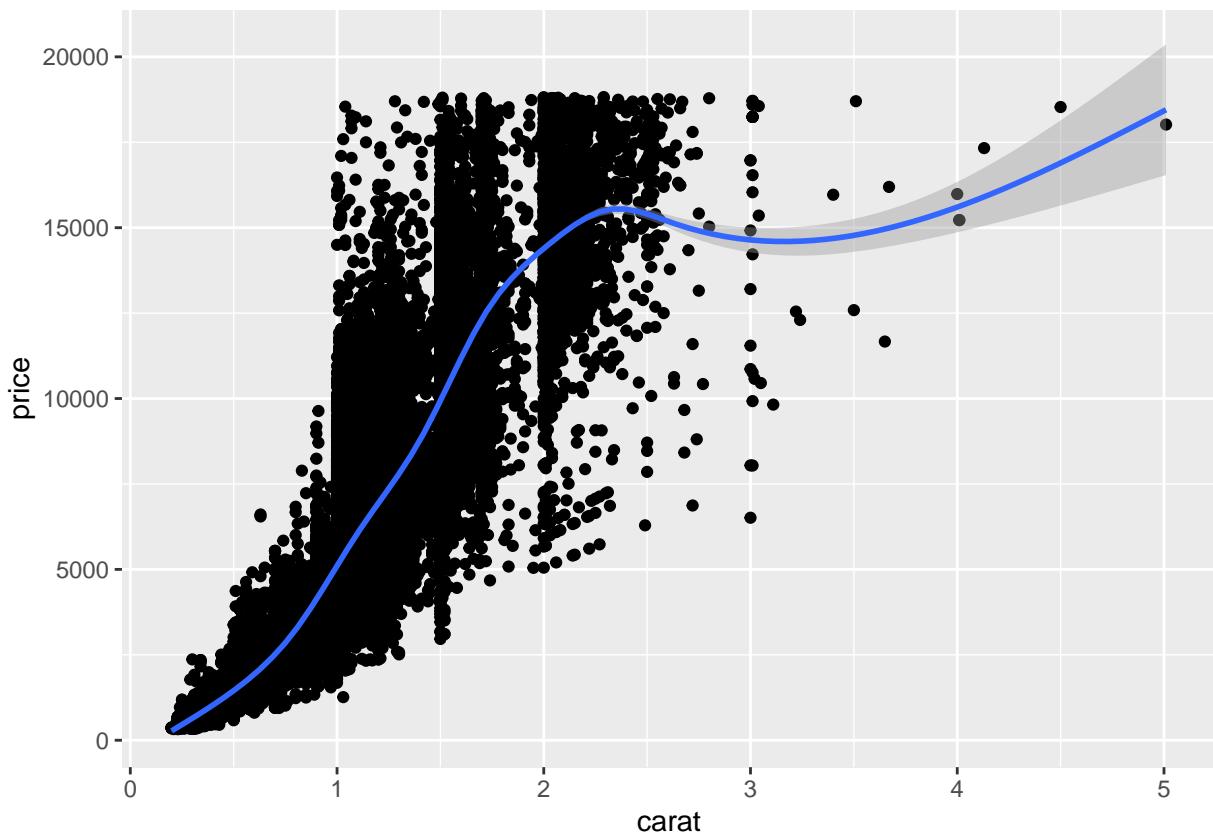
Nun kann mit dem `+`-Operator ein “geometrischer” Layer hinzugefügt werden:

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



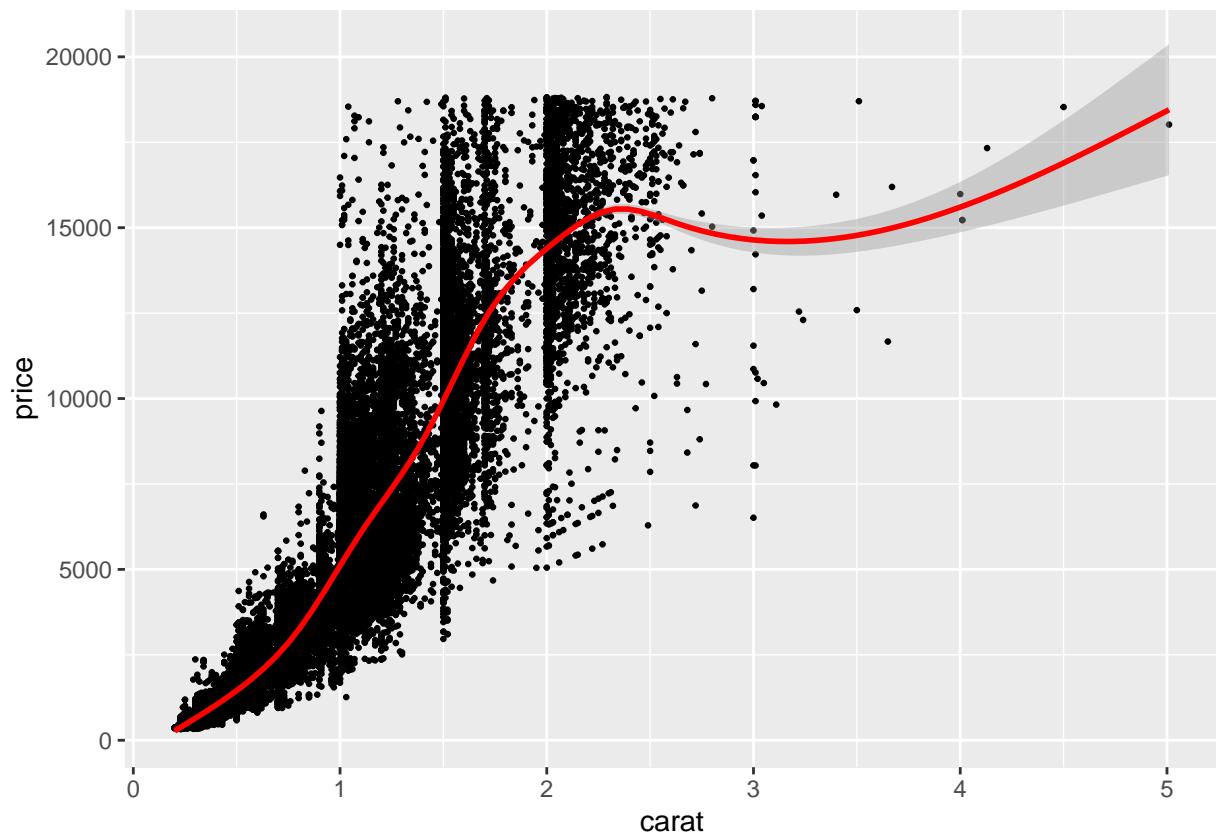
Weitere geom-Layer lassen sich mit dem +-Operator hinzufügen:

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



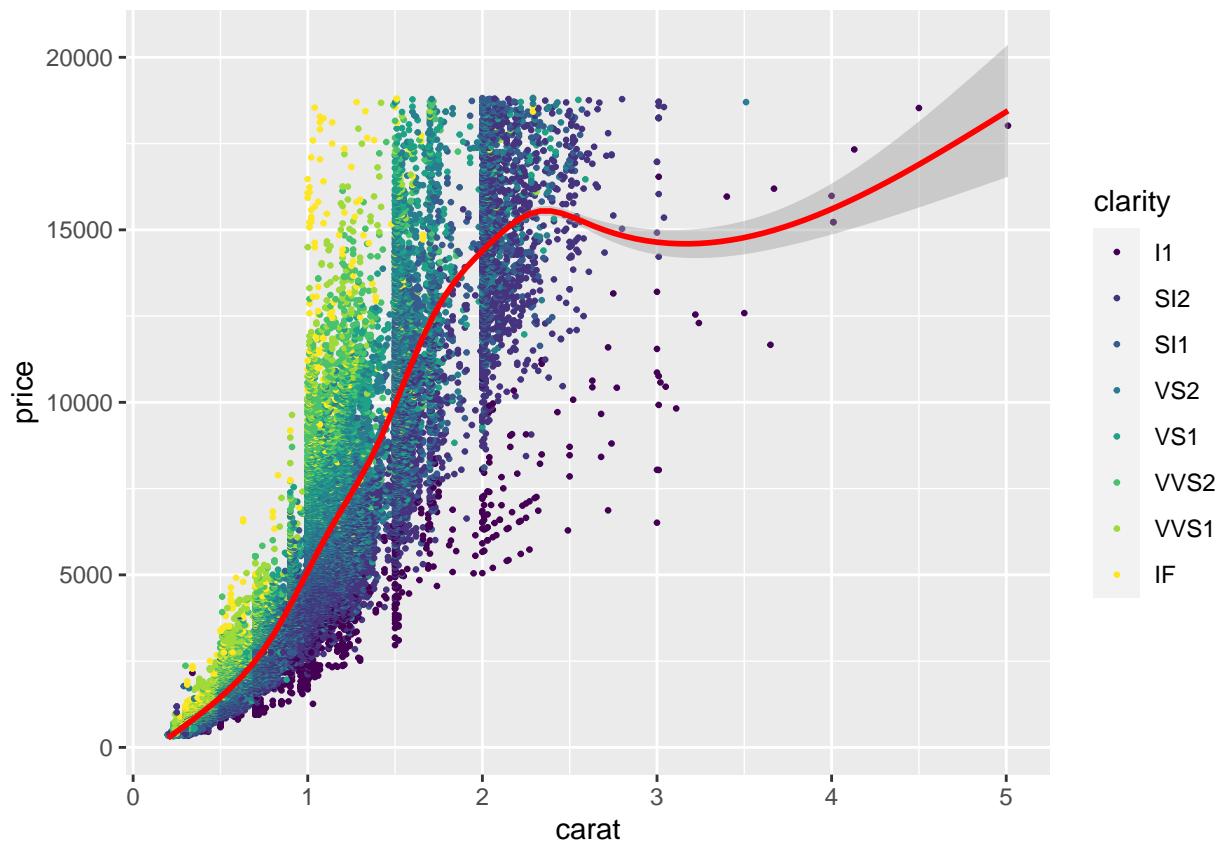
Die Layer-Funktionen können durch Parameter angepasst werden:

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



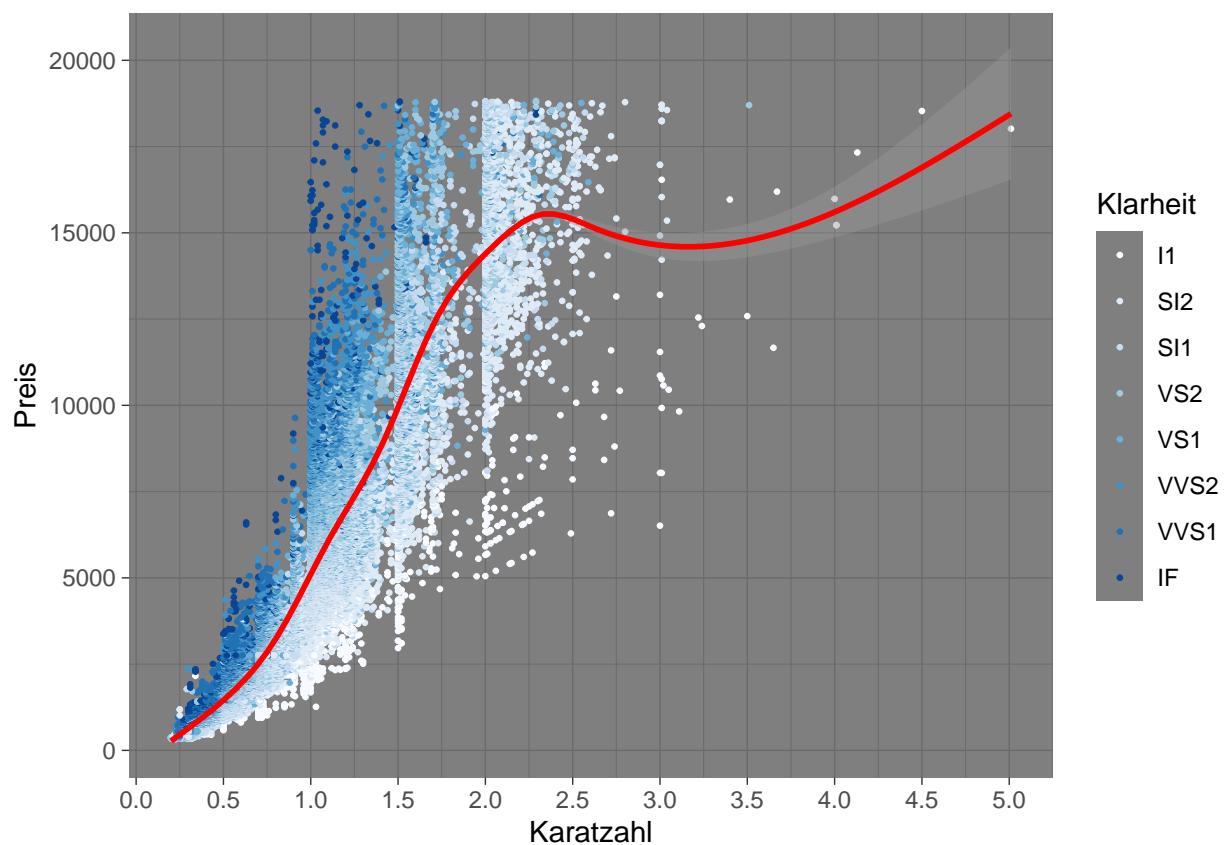
Dabei lassen sich in den einzelnen Layers mappings hinzufügen oder verändern:

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



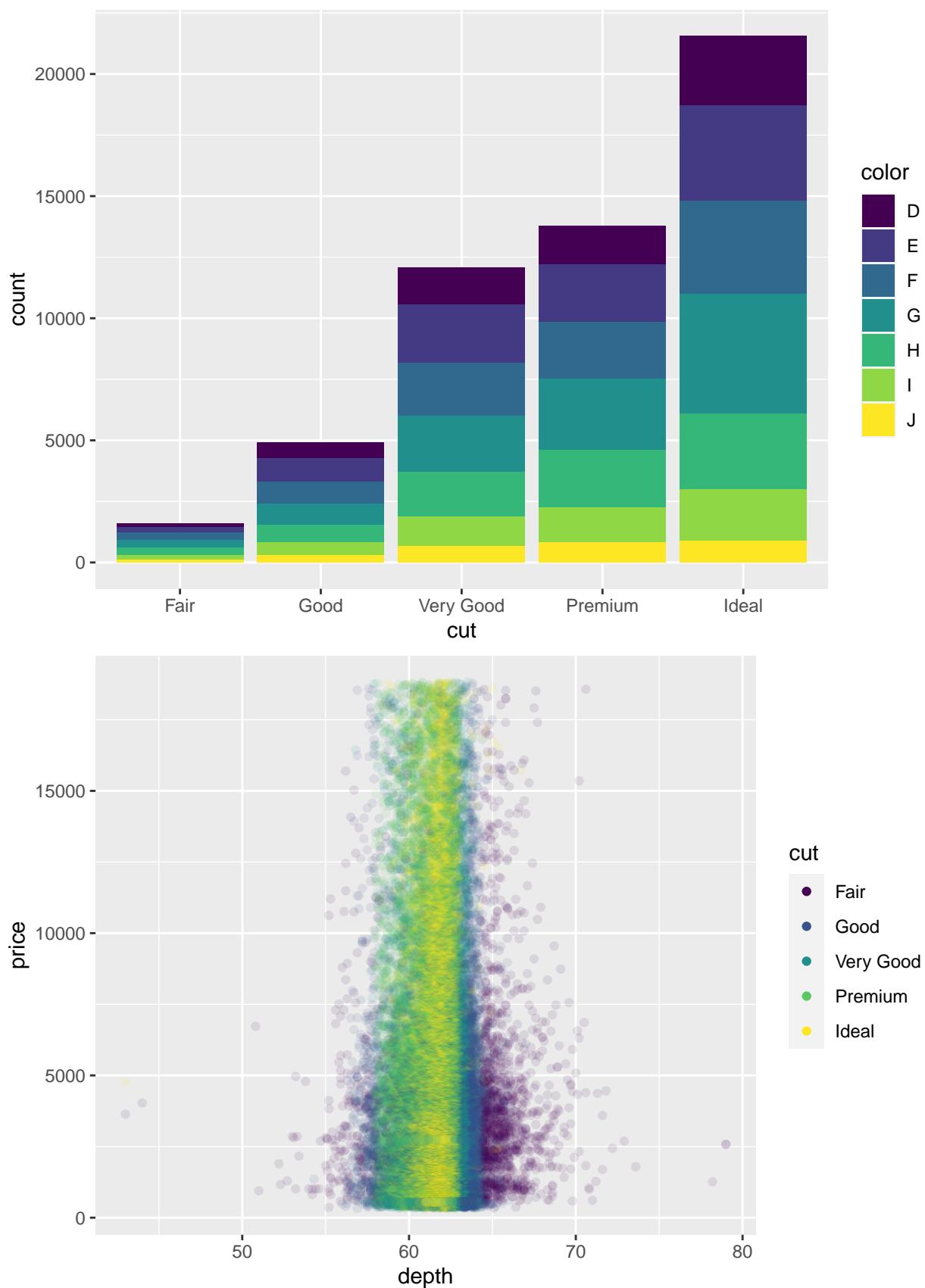
Schließlich lassen sich noch viele weitere optische Aspekte anpassen, z.B. Achsen, Farben, etc.:

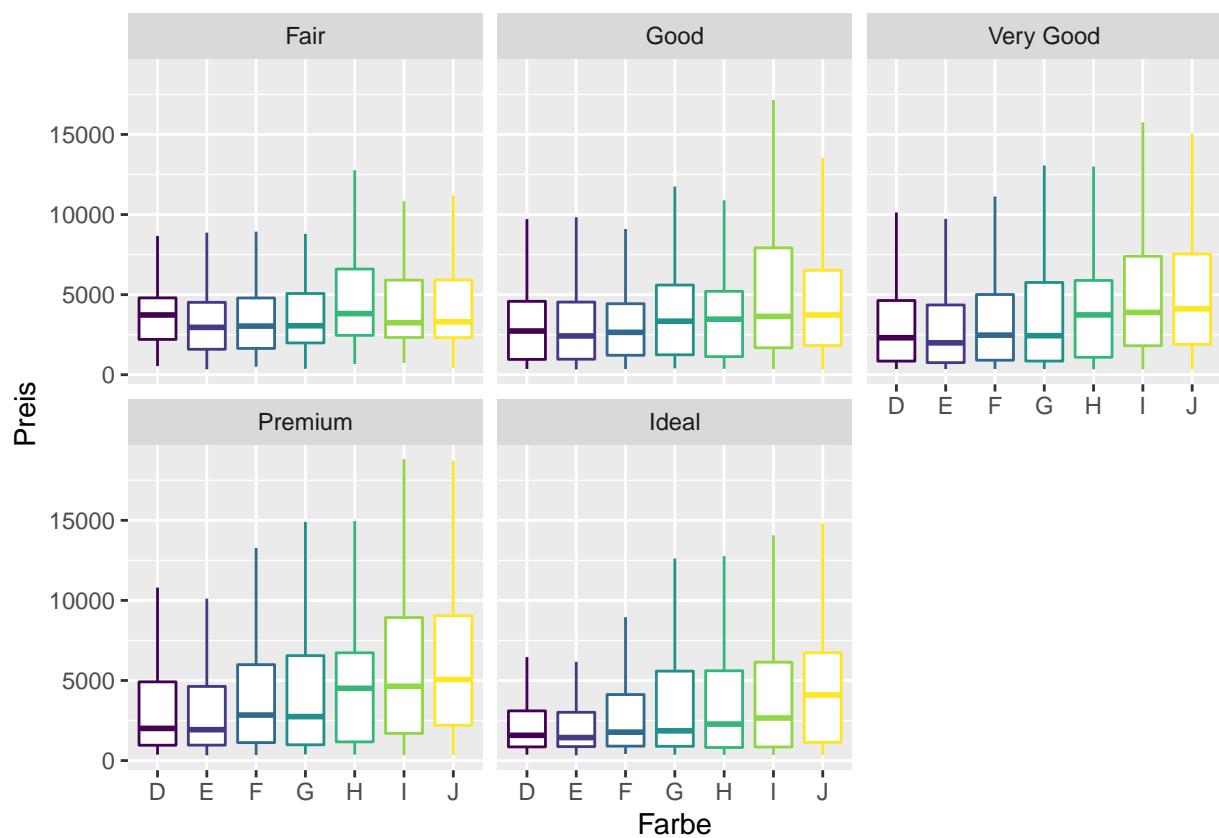
```
ggplot(diamonds, aes(x=carat, y=price)) +
  geom_point(aes(color=clarity), size=0.5) +
  geom_smooth(color="red") +
  scale_x_continuous("Karatzahl", breaks=seq(0,5,0.5)) +
  scale_y_continuous("Preis") +
  scale_color_brewer("Klarheit") +
  theme_dark()
```



5.5 Aufgaben

Versuchen Sie, folgende Visualisierungen des Datensatzes diamonds auszugeben:





5.5.1 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](#).