Grundlagen der Datenanalyse mit R

Max Brede

16.3.20

Contents

Chapter 1

Vorwort

Dieses mit `bookdown` erstellte Dokument ist die Sammlung der in der Einführung am 11.12.20 für's KfN Folien und Aufgaben.

1.0.1 Ablaufplan

Uhrzeit	Thema	Lernziele
		Die Teilnehmenden…
09:00	Rste Schritte	...haben einen Eindruck von den Vor- und Nachteilen von R \
10:30	Kaffeepause	
10:45	Daten manipulieren	...Kennen die Grundlagen des tidyverse-Workflows \ ...Könner
12:15	Mittagspause	
13:00	Daten einlesen und Zusammenfügen	...können Text-, Excel und SPSS-Daten einlesen \ ...können ru
14:30	Kaffeepause	
14:45	Daten darstellen	...Kennen die grundlegende Syntax von ggplot2 \ ...können gä
16:15	Interessante Pakete und Hilfestellungen	…Kennen erste Anlaufstellen bei Problemen mit R \ …Kennen

1.0.2 Voraussetzungen

Da der Kurs auf viele praktische Anwendungen setzen wird, sollte jeder Teilnehmer einen Rechner mitbringen, auf dem R und RStudio als grafische Oberfläche installiert sind. Die Installationsdateien für R für Windows findet man hier und für Mac hier, die Installationsdateien für RStudio für Windows hier und für Mac hier. Einer der zentralen Vorteile von R gegenüber anderen statistischen Software-Paketen ist die Möglichkeit, von der Community entwickelte Funktionen zu nutzen. Im Kurs sollen viele Aufgaben mit Hilfe einer Sammlung dieser in sogenannten `packages` in einem zentralen Archiv bereitgestellten Funktionserweiterungen gelöst werden. Diese Sammlung an Funktionen, das sogenannte `tidyverse`, bietet viele praktische Möglichkeiten, leichter lesbaren und verständlicheren R-Code zu schreiben. Installieren Sie bitte das `tidyverse`

vor dem Kurs durch das Ausführen der folgenden Code-Zeile in R:

```
install.packages('tidyverse',dependencies = TRUE)
```

Öffnen Sie dafür nach der Installation RStudio und kopieren Sie die Zeile einfach in die `Console` links unten. Führen Sie die Zeile anschließend durch das Drücken der Enter-Taste aus.

Chapter 2

Rste Schritte

2.0.1 Warum R?

2.0.1.1 R ist beliebt

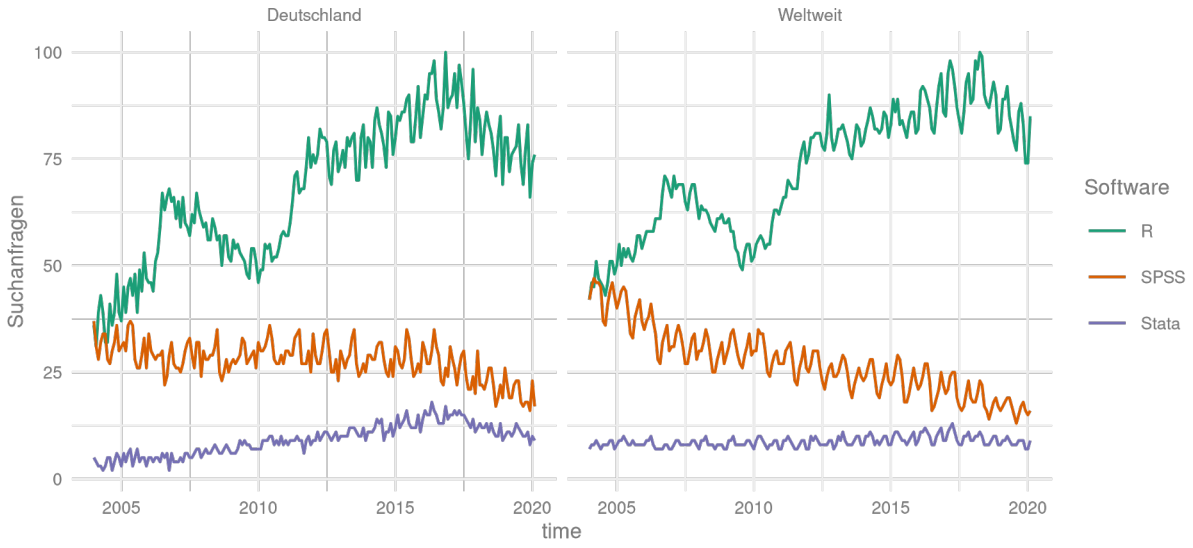


Figure 2.1: Google Suchanfragen, links aus [Deutschland](https://tinyurl.com/vosjrbz) und rechts [weltweit](https://tinyurl.com/vlwwp7a)

2.0.2 ...das sind nicht alles Wissenschaftler!

...stimmt, der Trend zeigt sich aber auch hier:

[BUCH] Discovering **statistics** using **IBM SPSS statistics**
A Field - 2013 - books.google.com
Lecturers/instructors-request a free digital inspection copy here With a little help from his weird band of characters the Fourth Edition of the award-winning book continues, with its unique blend of humour and collection of bizarre examples, to bring **statistics**-from first …
☆ 99 Zitiert von: 60112 Ähnliche Artikel Alle 10 Versionen »

[ZITATION] **IBM SPSS Statistics** for **Windows**, Version 20.0
IBMS **Statistics** - **IBM** Corp., Armonk, New York, 2011
☆ 99 Zitiert von: 78 Ähnliche Artikel

[ZITATION] **IBM SPSS statistics** for **Windows**, version 20.0
I **Spss** - New York: **IBM** Corp, 2011
☆ 99 Zitiert von: 382 Ähnliche Artikel

[ZITATION] **IBM SPSS statistics** for **windows**, version 22.0
IBM Corp - Armonk, NY: **IBM** Corp, 2013
☆ 99 Zitiert von: 353 Ähnliche Artikel

[ZITATION] **Stata Statistical Software**: **Release** 7.0
C Stata - CollegeStation, TX: Stata Press. StataStata Statistical …, 2001
☆ Zitiert von: 65 Ähnliche Artikel

[ZITATION] **Stata statistical software**: **Release** 13
LP StataCorp - 2013 - Statacorp lp College Station, TX
☆ Zitiert von: 6419 Ähnliche Artikel

[ZITATION] **Stata Statistical Software Release** 9
Stata Corp LP - 2005 - Stata Press Publication
☆ Zitiert von: 36 Ähnliche Artikel »

[ZITATION] **Stata Statistical Software Release** 7.0: User's Guide
Stata Corporation - 2001 - Stata Corporation
☆ Zitiert von: 446 Ähnliche Artikel »

[ZITATION] **Stata statistical software**: **release** 6.0
S Corportation - Texas: College station, 1999
☆ Zitiert von: 16 Ähnliche Artikel

[ZITATION] **Stata statistical software**: **Release** 10
SS Press - College Station, TX: StataCorp LP, 2007
☆ Zitiert von: 15 Ähnliche Artikel

[ZITATION] **Stata statistical software**: **release** 12. 2011
LP StataCorp - College Station, TX: StataCorp LP, 2011
☆ Zitiert von: 23 Ähnliche Artikel

[ZITATION] **Stata statistical software** (**Release** 11)[Computer software]
LP StataCorp - College Station, TX: StataCorp LP, 2009
☆ Zitiert von: 173 Ähnliche Artikel

[ZITATION] **Stata statistical software**: **Release** 12
C Stata - Special Edition, 2011
☆ Zitiert von: 75 Ähnliche Artikel

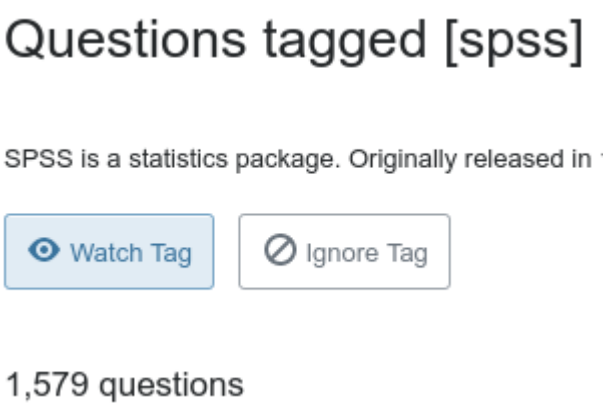
[ZITATION] **Stata Statistical Software**: **Release** 10 College Station
LP StataCorp - 2013 - Texas
☆ Zitiert von: 138 Ähnliche Artikel

[PDF] R: A language and environment for statistical computing
RC Team - 2013 - repo.bppt.go.id
This document describes basic features of dplR by following the initial steps that an analyst might follow when working with a new tree-ring data set. The vignette starts with reading in ring widths and plotting them. We describe a few of the available methods for detrending and then show how to extract basic descriptive statistics. We show how to build and plot a simple mean-value chronology. We also show how to build a chronology using the expressed population signal from the detrended ring widths as an example of how more …
☆ 99 Zitiert von: 158400 Ähnliche Artikel Alle 123 Versionen »

[ZITATION] R: A language and environment for statistical computing
RR Development Core Team - 2011 - R foundation for statistical computing …
☆ 99 Zitiert von: 79875 Ähnliche Artikel

2.0.3 … das sind doch alles nur Hilfeschreie!

…Möglich, aber dafür bekommt man für R auch leicht Hilfe. Hier ein Beispiel von stack overflow:



2.0.4 Warum R?

2.0.4.1 Wir fassen zusammen:

R ist sehr beliebt und hat eine sehr aktive Community

2.0.4.2 Woran liegt das?

Das zentrale Argument:

- R ist Open Source und damit
  - kostenlos

– von der Community erweiterbar

2.0.5 Warum benutzen wir dann nicht alle R?

- Mausnavigierte IDEs wirken erstmal intuitiver
- Man braucht vor allem am Anfang (ein bisschen) Frustrationstoleranz
- Die von der Community geschriebenen Erweiterungen (und über Strecken auch `base R`) haben keine einheitliche Syntax

2.0.5.1 Aber:

- Man findet sehr schnell Hilfe
- Es gibt Paketsammlungen, die einen Großteil der Datenaufbereitung und -analyse vereinheitlichen (z.B. das `tidyverse`)

2.0.6 R-Syntax Basics

Die Absoluten Grundlagen der R Syntax sind:

1. Zuweisungen und das `environment`
2. Funktionen und Argumente
3. Indizierung

2.0.7 1. Zuweisungen und das Environment

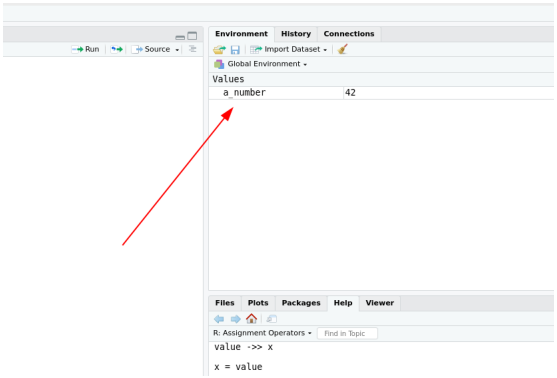
Unter Zuweisung ist erstmal nichts anderes zu verstehen, als das Ablegen eines Zwischenergebnisses unter einem Namen, um es später weiterzuverwenden.

Auch wenn es andere Möglichkeiten gibt, ist die Folgende die lesbarste:

```
a_number <- 42
```

2.0.8 1. Zuweisungen und das `Environment`

Die Zahl 42 ist jetzt für weitere Verwendung im `Environment` abgelegt:



2.0.9 1. Zuweisungen und das `Environment`

Und wie die Zahl alleine weiterzuverwenden:

```
42^2
```

```
## [1] 1764
```

```
a_number^2 ## äquivalent
```

```
## [1] 1764
```

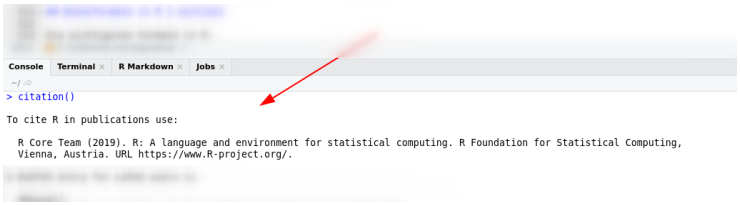
Jede dieser in grau unterlegten Zeilen nennt man auch eine *Anweisung*. R wird in der letzten Zeile angewiesen, den 'Inhalt' von `a_number` zu quadrieren. Dabei wird der dahinter durch das `#`-Symbol eingeleitete Kommentar ignoriert.

2.0.10 2. Funktionen und Argumente

Der Großteil des in R erstellten Codes besteht aus *Funktionen*.
Jede Funktion ist eine Sammlung an *Anweisungen*, die nacheinander augeführt werden sollen.
`citation()` ist ein sehr einfaches Beispiel für eine solche Funktion.

Was macht `citation()`?

`citation()` gibt in der *Konsole* aus, wie man R am Besten zitiert.



2.0.11 2. obligatorische und optionale Argumente

Die meisten Funktionen kommen aber nicht ohne *Argumente* aus.
Argumente können in *obligatorische* und *optionale* unterteilt werden.

Wie die Namen schon sagen, sind *obligatorische* Argumente solche, ohne die die Funktion nicht ausgeführt werden kann.
Obligatorische Argumente sind meistens die Werte, auf deren Basis gerade die Operationen ausgeführt werden sollen.

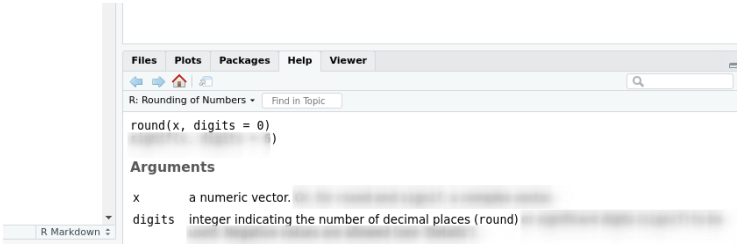
Wenn man keins oder ein falsches obligatorisches Argument übergibt, zeigt R einen Fehler an!

optionale Argumente nennt man die, für die die Autoren der Funktion einen Standard vorgesehen haben. Das sind dann meist Stellschrauben, an denen das gewünschte Ergebnis genauer festgelegt werden kann. Werden diese Argumente nicht explizit gesetzt, wird einfach der Standard verwendet.

2.0.12 2. obligatorische und optionale Argumente

Ein Beispiel für eine Funktion, die obligatorische und optionale Argumente annimmt ist `round()`.

Auf der Hilfeseite von `round()` finden wir folgendes[1] :



Was ist hier das optionale Argument und wie erkennt man es?

2.0.13 2. obligatorische und optionale Argumente

`x` ist hier das obligatorische Argument (kein Standard durch ein `=`) angegeben

Wenn man `round` ohne ausprobiert, gibt es einen Fehler:

```
round()
```

```
## Error in eval(expr, envir, enclos): 0 arguments passed to 'round' which requires 1 a
```

Wen man eine Zahl übergibt, wird auf ganze Zahlen gerundet:

```
round(3.1415)
```

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

[1] Dabei nutzt die RStudio-IDE aber nur die `str()`(für structure)-Funktion.

2.0.14 2. obligatorische und optionale Argumente

Das optionale Argument `digits`, ermöglicht dann, die gewünschte Anzahl der Nachkommastellen anzugeben:

```
round(3.1415, digits = 2)
```

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

Sowohl `3.1415` als auch `digits = 2` setzen Werte für Argumente!

Da die Funktion aber die zu rundende Zahl `x` an erster Stelle erwartet, ergibt der Aufruf das gewünschte Ergebnis.

2.0.15 Position von Argumente

R braucht also nicht unbedingt die Argumentnamen, wenn keine da sind wird die Reihenfolge interpretiert.

```
round(3.1415, 2) ## funktioniert, digits wird an zweiter Stelle erwartet
```

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

Was versucht R, wenn ich die folgende Anweisung ausführe?

```
round(2, 3.1415)
```

2.0.16 positionelle Argumente

R rundet die Zahl 2 auf 3.1415 (also 3) Nachkommastellen.

```
round(2, 3.1415) ## funktioniert, aber vielleicht nicht wie erwartet
```

Wenn man Argumente ohne Namen in falscher Reihenfolge übergibt, gibt es keine Fehlermeldung aber Blödsinn!

2.0.17 Operatoren

Einzelne Zahlen benutzt man aber ja quasi nie. Deswegen hier eine sehr praktische Funktion:

```
1:3
```

```
## [1] 1 2 3
```

Huch! Das sieht ja gar nicht nach einer Funktion aus!

Neben den klassischen Funktionen, die durch ein Codewort und Klammern erkenntlich sind, gibt es in R noch eine Reihe *Operatoren*, die auf den ersten Blick keine Funktionen sind.
Hier wird aber eigentlich ‘:‘(1,3) ausgeführt, das Funktionsschema gilt also auch hier. ‘:‘(1,3) ist nur schrecklich schlecht lesbar und viel zu viel zu tippen.

2.0.18 3. Indizierung

Da wir jetzt aber erste `Vektoren` mit mehr als einem Element erstellen können, gehen wir zu nächsten Part, der *Indizierung* über.

In R lassen sich Elemente eines Objektes auf viele verschiedene Arten aufrufen, am Ende laufen diese aber auf den `[]`, den `[[]]` und den `$`-Operator hinaus.

Für Vektoren reicht erstmal der `[]`-Operator.

2.0.19 3. Indizierung

Das einfachste Beispiel ist der Versuch, den 3. Wert aus einer Zahlenreihe ausgeben zu lassen.

Dafür erstellen wir zuerst die Zahlenreihe von 10 bis 15 und speichern diese im `Environment`

Wie mache ich das?

```
eine_reihe_von_zahlen <- 10:15
```

2.0.20 3. Indizierung

Jetzt kann ich den `[]`-Operator benutzen, um den 3. Wert anzeigen zu lassen:

```
eine_reihe_von_zahlen[3]
```

```
## [1] 12
```

Und fertig. So einfach.

2.0.21 3. Indizierung

Der `[]`-Operator kann aber noch viel mehr. Ich kann zum Beispiel eine Sequenz übergeben, um eine Reihe von Zahlen ausgeben zu lassen:

```
eine_reihe_von_zahlen[1:3]
```

```
## [1] 10 11 12
```

Der erste Wert ist die 10! der Index für die erste Stelle ist also die 1 (im Gegensatz zu Python z.B.)!

Eine weitere Möglichkeit ist die ausschließende Indizierung. Mit einem `-` gibt man an, dass einen alle außer der angegebenen Stelle interessieren.

```
eine_reihe_von_zahlen[-3]
```

```
## [1] 10 11 13 14 15
```

2.0.22 logische Indizierung

Der `[]`-Operator kann außerdem benutzt werden, um über *logische Operatoren* Werte zu indizieren.

Die einfachsten sind hier:

```
1 == 2 ## ist 1 gleich 2
1 != 3 ## ist 1 ungleich 3
1 < 4  ## ist 1 kleiner als 4
2 >= 1 ## ist 2 größer gleich 1
```

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

2.0.23 3. logische Indizierung

Diese Operatoren kann ich auch auf Vektoren anwenden:

```
eine_reihe_von_zahlen>11
```

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

Und kann das Ergebnis auch mit dem `[]`-Operator kombinieren:

```
eine_reihe_von_zahlen[eine_reihe_von_zahlen>11]
```

```
## [1] 12 13 14 15
```

2.0.24 Datenformate in R

Bei der letzten Operation haben wir zwei Datenformate kennengelernt:

- `logical`, eine binär-logische Angabe und
- `numeric`, alle ganze und (darstellbare) rationale Zahlen

Jetzt kennen wir schon 2 der 3 wichtigsten einfachen oder `atomic` Datenformate in R

2.0.25 Datenformate in R

Neben Zahlen muss R aber natürlich auch Text verarbeiten können. Dies geschieht über das `character`-Datenformat.

Wie könnte ich versuchen, ein `character`-Objekt mit dem Inhalt “Ich bin ein String” anzulegen?

```
ein_toller_character <- "Ich bin ein String"
```

2.0.26 Datenformate in R

Diese einfachen Datenformate haben eine Hierarchie, die man so darzustellen versuchen könnte:

logical < numeric < character

Um uns das zu verdeutlichen, lernen wir noch eine neue Funktion:

- `c()` - die Vektor-Funktion. Mit ihr können wir Vektoren erstellen und Werte zu bestehenden Vektoren hinzufügen.

2.0.27 Datenformate in R

```
logical_vector <- c(TRUE, TRUE, FALSE)
logical_vector
```

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

```
c(logical_vector,1)
```

```
## [1] 1 1 0 1
```

Die logischen Werte wurden in Zahlen umgewandelt.

2.0.28 Datenformate in R

Was passiert wohl, wenn wir eine 1 *und* einen `character` hinzufügen?

```
c(logical_vector,1,'ein character')
```

```
## [1] "TRUE"          "TRUE"
## [3] "FALSE"         "1"
## [5] "ein character"
```

Die logischen Werte und die Zahl wurden in `character` umgewandelt

2.0.29 Datenformate in R

Die `atomics` haben eine klare Hierarchie!

Rückgängig machen lässt sich das durch `as.logical`, `as.numeric` und `as.character`. Aber Vorsicht, so können auch leicht fehlende Werte, durch `NA` gekennzeichnet erzeugt werden:

```
ein_umzuwandelnder_vektor <- c('a',1,15,TRUE)
as.numeric(ein_umzuwandelnder_vektor)
```

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

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

2.0.30 Datenformate in R

```
as.numeric(ein_umzuwandelnder_vektor)
```

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

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

Warum fehlt auch der letzte Wert?

Weil das `TRUE` inzwischen ein `character` ist.

```
ein_umzuwandelnder_vektor
```

```
## [1] "a"    "1"    "15"   "TRUE"
```

2.0.31 Datenformate in R

Natürlich gibt es auch komplexere, mehrsimensionale Datenformate in R, um die kümmern wir uns dann nach der Pause.

Chapter 3

Daten manipulieren

3.0.1 Datensätze in R

Wie alle anderen Programme zur statistischen Auswertung hat R natürlich neben den Vektoren auch rechteckige Datenformate.

Das typische rechteckige Datenformat in `base R` ist der `data.frame`. Im Prinzip nichts anderes, als spaltenweise zusammengeklebte Vektoren. Der Konstruktor für ein solches Objekt heißt ist die gleichnamige Funktion, die die Spalten als benannte Argumente nimmt:

```
df <- data.frame(a = 1:3,
                 b = c(TRUE, FALSE, TRUE),
                 c = c('a','b','c'))
df
```

```
##   a     b c
## 1 1  TRUE a
## 2 2 FALSE b
## 3 3  TRUE c
```

3.0.2 Datensätze in R

Das Indizieren im Datensatz geht dann am *Lesbarsten*, durch das Angeben der gewünschten Spalte mit dem `$`-Operator und der Auswahl der Zeile durch den schon bekannten `[]`-Operator.

```
df$c[2] ## 2. Wert in der 'c'-Spalte.
```

```
## [1] "b"
```

Wie könnte ich den 3. Wert in der `b`-Spalte indizieren?

```
df$b[3]
```

3.0.3 Datensätze in R

Der `iris`-Datensatz ist ein im Grundumfang von R mitgelieferter Datensatz, der historische botanische Daten nach Anderson (1935) enthält.

```
iris
```

```
##     Sepal.Length Sepal.Width Petal.Length
## 1            5.1         3.5          1.4
## 2            4.9         3.0          1.4
## 3            4.7         3.2          1.3
## 4            4.6         3.1          1.5
## 5            5.0         3.6          1.4
## 6            5.4         3.9          1.7
## 7            4.6         3.4          1.4
## 8            5.0         3.4          1.5
## 9            4.4         2.9          1.4
## 10           4.9         3.1          1.5
## 11           5.4         3.7          1.5
## 12           4.8         3.4          1.6
## 13           4.8         3.0          1.4
## 14           4.3         3.0          1.1
## 15           5.8         4.0          1.2
## 16           5.7         4.4          1.5
## 17           5.4         3.9          1.3
## 18           5.1         3.5          1.4
## 19           5.7         3.8          1.7
## 20           5.1         3.8          1.5
## 21           5.4         3.4          1.7
## 22           5.1         3.7          1.5
## 23           4.6         3.6          1.0
## 24           5.1         3.3          1.7
## 25           4.8         3.4          1.9
## 26           5.0         3.0          1.6
## 27           5.0         3.4          1.6
## 28           5.2         3.5          1.5
## 29           5.2         3.4          1.4
## 30           4.7         3.2          1.6
## 31           4.8         3.1          1.6
## 32           5.4         3.4          1.5
## 33           5.2         4.1          1.5
## 34           5.5         4.2          1.4
## 35           4.9         3.1          1.5
## 36           5.0         3.2          1.2
## 37           5.5         3.5          1.3
## 38           4.9         3.6          1.4
## 39           4.4         3.0          1.3
```

```
## 40          5.1         3.4         1.5
## 41          5.0         3.5         1.3
## 42          4.5         2.3         1.3
## 43          4.4         3.2         1.3
## 44          5.0         3.5         1.6
## 45          5.1         3.8         1.9
## 46          4.8         3.0         1.4
## 47          5.1         3.8         1.6
## 48          4.6         3.2         1.4
## 49          5.3         3.7         1.5
## 50          5.0         3.3         1.4
## 51          7.0         3.2         4.7
## 52          6.4         3.2         4.5
## 53          6.9         3.1         4.9
## 54          5.5         2.3         4.0
## 55          6.5         2.8         4.6
## 56          5.7         2.8         4.5
## 57          6.3         3.3         4.7
## 58          4.9         2.4         3.3
## 59          6.6         2.9         4.6
## 60          5.2         2.7         3.9
## 61          5.0         2.0         3.5
## 62          5.9         3.0         4.2
## 63          6.0         2.2         4.0
## 64          6.1         2.9         4.7
## 65          5.6         2.9         3.6
## 66          6.7         3.1         4.4
## 67          5.6         3.0         4.5
## 68          5.8         2.7         4.1
## 69          6.2         2.2         4.5
## 70          5.6         2.5         3.9
## 71          5.9         3.2         4.8
## 72          6.1         2.8         4.0
## 73          6.3         2.5         4.9
## 74          6.1         2.8         4.7
## 75          6.4         2.9         4.3
## 76          6.6         3.0         4.4
## 77          6.8         2.8         4.8
## 78          6.7         3.0         5.0
## 79          6.0         2.9         4.5
## 80          5.7         2.6         3.5
## 81          5.5         2.4         3.8
## 82          5.5         2.4         3.7
## 83          5.8         2.7         3.9
## 84          6.0         2.7         5.1
## 85          5.4         3.0         4.5
```

```
## 86          6.0         3.4         4.5
## 87          6.7         3.1         4.7
## 88          6.3         2.3         4.4
## 89          5.6         3.0         4.1
## 90          5.5         2.5         4.0
## 91          5.5         2.6         4.4
## 92          6.1         3.0         4.6
## 93          5.8         2.6         4.0
## 94          5.0         2.3         3.3
## 95          5.6         2.7         4.2
## 96          5.7         3.0         4.2
## 97          5.7         2.9         4.2
## 98          6.2         2.9         4.3
## 99          5.1         2.5         3.0
## 100         5.7         2.8         4.1
## 101         6.3         3.3         6.0
## 102         5.8         2.7         5.1
## 103         7.1         3.0         5.9
## 104         6.3         2.9         5.6
## 105         6.5         3.0         5.8
## 106         7.6         3.0         6.6
## 107         4.9         2.5         4.5
## 108         7.3         2.9         6.3
## 109         6.7         2.5         5.8
## 110         7.2         3.6         6.1
## 111         6.5         3.2         5.1
## 112         6.4         2.7         5.3
## 113         6.8         3.0         5.5
## 114         5.7         2.5         5.0
## 115         5.8         2.8         5.1
## 116         6.4         3.2         5.3
## 117         6.5         3.0         5.5
## 118         7.7         3.8         6.7
## 119         7.7         2.6         6.9
## 120         6.0         2.2         5.0
## 121         6.9         3.2         5.7
## 122         5.6         2.8         4.9
## 123         7.7         2.8         6.7
## 124         6.3         2.7         4.9
## 125         6.7         3.3         5.7
## 126         7.2         3.2         6.0
## 127         6.2         2.8         4.8
## 128         6.1         3.0         4.9
## 129         6.4         2.8         5.6
## 130         7.2         3.0         5.8
## 131         7.4         2.8         6.1
```

```
## 132          7.9         3.8          6.4
## 133          6.4         2.8          5.6
## 134          6.3         2.8          5.1
## 135          6.1         2.6          5.6
## 136          7.7         3.0          6.1
## 137          6.3         3.4          5.6
## 138          6.4         3.1          5.5
## 139          6.0         3.0          4.8
## 140          6.9         3.1          5.4
## 141          6.7         3.1          5.6
## 142          6.9         3.1          5.1
## 143          5.8         2.7          5.1
## 144          6.8         3.2          5.9
## 145          6.7         3.3          5.7
## 146          6.7         3.0          5.2
## 147          6.3         2.5          5.0
## 148          6.5         3.0          5.2
## 149          6.2         3.4          5.4
## 150          5.9         3.0          5.1
##     Petal.Width    Species
## 1           0.2     setosa
## 2           0.2     setosa
## 3           0.2     setosa
## 4           0.2     setosa
## 5           0.2     setosa
## 6           0.4     setosa
## 7           0.3     setosa
## 8           0.2     setosa
## 9           0.2     setosa
## 10          0.1     setosa
## 11          0.2     setosa
## 12          0.2     setosa
## 13          0.1     setosa
## 14          0.1     setosa
## 15          0.2     setosa
## 16          0.4     setosa
## 17          0.4     setosa
## 18          0.3     setosa
## 19          0.3     setosa
## 20          0.3     setosa
## 21          0.2     setosa
## 22          0.4     setosa
## 23          0.2     setosa
## 24          0.5     setosa
## 25          0.2     setosa
## 26          0.2     setosa
```

```
## 27          0.4     setosa
## 28          0.2     setosa
## 29          0.2     setosa
## 30          0.2     setosa
## 31          0.2     setosa
## 32          0.4     setosa
## 33          0.1     setosa
## 34          0.2     setosa
## 35          0.2     setosa
## 36          0.2     setosa
## 37          0.2     setosa
## 38          0.1     setosa
## 39          0.2     setosa
## 40          0.2     setosa
## 41          0.3     setosa
## 42          0.3     setosa
## 43          0.2     setosa
## 44          0.6     setosa
## 45          0.4     setosa
## 46          0.3     setosa
## 47          0.2     setosa
## 48          0.2     setosa
## 49          0.2     setosa
## 50          0.2     setosa
## 51          1.4 versicolor
## 52          1.5 versicolor
## 53          1.5 versicolor
## 54          1.3 versicolor
## 55          1.5 versicolor
## 56          1.3 versicolor
## 57          1.6 versicolor
## 58          1.0 versicolor
## 59          1.3 versicolor
## 60          1.4 versicolor
## 61          1.0 versicolor
## 62          1.5 versicolor
## 63          1.0 versicolor
## 64          1.4 versicolor
## 65          1.3 versicolor
## 66          1.4 versicolor
## 67          1.5 versicolor
## 68          1.0 versicolor
## 69          1.5 versicolor
## 70          1.1 versicolor
## 71          1.8 versicolor
## 72          1.3 versicolor
```

```
## 73          1.5 versicolor
## 74          1.2 versicolor
## 75          1.3 versicolor
## 76          1.4 versicolor
## 77          1.4 versicolor
## 78          1.7 versicolor
## 79          1.5 versicolor
## 80          1.0 versicolor
## 81          1.1 versicolor
## 82          1.0 versicolor
## 83          1.2 versicolor
## 84          1.6 versicolor
## 85          1.5 versicolor
## 86          1.6 versicolor
## 87          1.5 versicolor
## 88          1.3 versicolor
## 89          1.3 versicolor
## 90          1.3 versicolor
## 91          1.2 versicolor
## 92          1.4 versicolor
## 93          1.2 versicolor
## 94          1.0 versicolor
## 95          1.3 versicolor
## 96          1.2 versicolor
## 97          1.3 versicolor
## 98          1.3 versicolor
## 99          1.1 versicolor
## 100         1.3 versicolor
## 101         2.5  virginica
## 102         1.9  virginica
## 103         2.1  virginica
## 104         1.8  virginica
## 105         2.2  virginica
## 106         2.1  virginica
## 107         1.7  virginica
## 108         1.8  virginica
## 109         1.8  virginica
## 110         2.5  virginica
## 111         2.0  virginica
## 112         1.9  virginica
## 113         2.1  virginica
## 114         2.0  virginica
## 115         2.4  virginica
## 116         2.3  virginica
## 117         1.8  virginica
## 118         2.2  virginica
```

```
## 119          2.3  virginica
## 120          1.5  virginica
## 121          2.3  virginica
## 122          2.0  virginica
## 123          2.0  virginica
## 124          1.8  virginica
## 125          2.1  virginica
## 126          1.8  virginica
## 127          1.8  virginica
## 128          1.8  virginica
## 129          2.1  virginica
## 130          1.6  virginica
## 131          1.9  virginica
## 132          2.0  virginica
## 133          2.2  virginica
## 134          1.5  virginica
## 135          1.4  virginica
## 136          2.3  virginica
## 137          2.4  virginica
## 138          1.8  virginica
## 139          1.8  virginica
## 140          2.1  virginica
## 141          2.4  virginica
## 142          2.3  virginica
## 143          1.9  virginica
## 144          2.3  virginica
## 145          2.5  virginica
## 146          2.3  virginica
## 147          1.9  virginica
## 148          2.0  virginica
## 149          2.3  virginica
## 150          1.8  virginica
```

3.0.4 Übersicht über Datensatz verschaffen

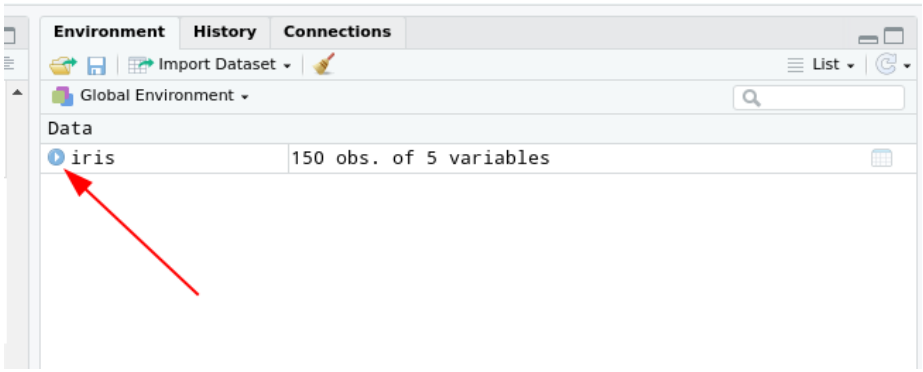
Das ist natürlich ein bisschen unübersichtlich, wie kann man damit umgehen?

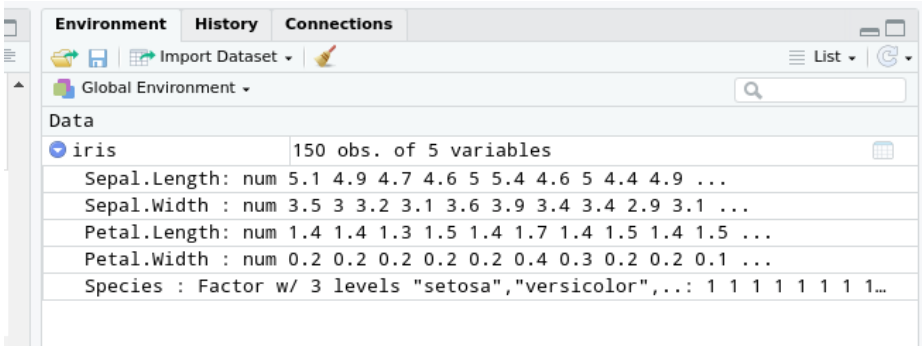
3.0.4.1 1. Möglichkeit:

Wenn man iris explizit in das Environment nimmt, kann man die Oberfläche von RStudio nutze, um sich einen Überblick zu verschaffen[1]:

```
iris <- iris
```

[1] Dabei nutzt die RStudio-IDE aber nur die `str()`(für structure)-Funktion.





3.0.5 Übersicht über Datensatz verschaffen

3.0.5.1 2. Möglichkeit:

Die `summary`-Funktion, die genau das macht, was ihr Name suggeriert:

```
summary(iris)
```

```
##    Sepal.Length    Sepal.Width     Petal.Length
##  Min.   :4.300   Min.   :2.000   Min.   :1.000
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600
##  Median :5.800   Median :3.000   Median :4.350
##  Mean   :5.843   Mean   :3.057   Mean   :3.758
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100
##  Max.   :7.900   Max.   :4.400   Max.   :6.900
##    Petal.Width          Species
##  Min.   :0.100   setosa    :50
##  1st Qu.:0.300   versicolor:50
##  Median :1.300   virginica :50
##  Mean   :1.199
##  3rd Qu.:1.800
##  Max.   :2.500
```

3.0.6 Aufgabe: Deskriptive Kennwerte berechnen

Wir wollen für diesen Datensatz jetzt die folgenden Schritte der Auswertung vollziehen:

1. Ausschluss der Blumen, die breitere Blütenblätter als das 1.5-fache der mittleren Blütenblätter haben und Kelche, die kürzer als das Mittel der Kelchlänge sind
2. Darstellung der Mittelwerte und Streuungen der Blütenblattlänge und -breite pro verbleibende Spezies als Tabelle

3.0.7 Aufgabe: Base-R Lösung

```
df <- iris[iris$Petal.Width <= 1.5 * mean(iris$Petal.Width) &
             iris$Sepal.Length >= mean(iris$Sepal.Length),]
means <- aggregate(cbind(df$Petal.Length,df$Petal.Width),
          by = list(Species = df$Species),
          FUN = mean)
sds <- aggregate(cbind(df$Petal.Length,df$Petal.Width),
          by = list(Species = df$Species),
          FUN = sd)
tab <- data.frame(means, sds[,2:3])
names(tab)[2:5] = c('m_Length', 'm_Width', 'sd_Length', 'sd_Width')
tab
```

```
##      Species m_Length m_Width sd_Length
## 1 versicolor    4.560   1.424 0.2783882
## 2  virginica    5.375   1.500 0.3862210
##     sd_Width
## 1 0.14798649
## 2 0.08164966
```

3.0.8 Auftritt tidyverse

Die selbe Aufgabe wie gerade, jetzt mit dem `tidyverse`:

```
library(tidyverse)
iris %>%
  filter(Petal.Width <= 1.5 * mean(Petal.Width) &
           Sepal.Length >= mean(Sepal.Length)) %>%
  group_by(Species) %>%
  summarise(m_Length = mean(Petal.Length),
            sd_Length = sd(Petal.Length),
            m_Width = mean(Petal.Width),
            sd_Width = sd(Petal.Width))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 5
##   Species    m_Length sd_Length m_Width sd_Width
##   <fct>         <dbl>     <dbl>   <dbl>    <dbl>
## 1 versicolor     4.56     0.278    1.42   0.148
## 2 virginica      5.38     0.386    1.5    0.0816
```

3.0.9 tidy aggregation

Das `tidyverse` ist eine Sammlung von Paketen, deren Hauptziel es ist, Datenaufbereitung in R intuitiver und leichter lesbar zu machen.

Ein zentrales Element dabei ist der `%>%`-Operator, die sogenannte Pipeline. Beim Skript-Lesen und -Schreiben kann man sich diese am Besten als *'dann'* vorstellen

Mit ihrer Hilfe werden Aufbereitungsschritte in einer stringenten Reihe an Operationen formuliert, die sich am Besten als Satz verstehen lassen.

Da die Funktionen im tidyverse alle mit einfachen Verben benannt sind, lässt sich die Operation von eben auch so lesen.

3.0.10 tidy aggregation

```
iris %>%
```

Nimm `iris`, dann …

```
  filter(Petal.Width <= 1.5 * mean(Petal.Width) &
           Sepal.Length >= mean(Sepal.Length)) %>%
```

… filter Zeilenweise nach den gesetzten Regeln, dann…

```
  group_by(Species) %>%
```

…gruppiere nach der Spezies, dann…

```
  summarise(m_Length = mean(Petal.Length),
            sd_Length = sd(Petal.Length),
            m_Width = mean(Petal.Width),
            sd_Width = sd(Petal.Width))
```

… berechne die angegebenen Kenngrößen über die Gruppen.

3.0.11 tidy aggregation

Zweite Beispielaufgabe:

Wir möchten für den `iris`-Datensatz:

1. Eine Spalte hinzufügen, die die z-transformierte Blattlänge enthält

2. Eine Spalte hinzufügen, die als character das Ergebnis eines Mediansplits der gerade erstellten Variable enthält
3. Einen Datensatz erstellen, der nur die Spezies, die z-Transformierte und die Mediansplit-Variable enthält
4. Die Häufigkeiten der Kombinationen von Mediansplit-Gruppe und Spezies auszählen

3.0.12 tidy aggregation

```
df <- iris %>%
  mutate(z_length = (Petal.Length-mean(Petal.Length))/sd(Petal.Length),
         med_split = ifelse(z_length >= median(z_length),
                            'upper',
                            'lower')) %>%
  select(Species, z_length, med_split)
```

3.0.13 tidy aggregation

Hat das geklappt?

Wie könnte ich das überprüfen?

```
summary(df)
```

```
##        Species      z_length
##  setosa    :50   Min.   :-1.5623
##  versicolor:50   1st Qu.:-1.2225
##  virginica :50   Median : 0.3354
##                  Mean   : 0.0000
##                  3rd Qu.: 0.7602
##                  Max.   : 1.7799
##   med_split
##  Length:150
##  Class :character
##  Mode  :character
##
##
##
```

3.0.14 tidy aggregation

Jetzt noch Häufigkeiten auszählen:

```
df %>%
  group_by(Species, med_split) %>%
  summarise(n = n())
```

```
## `summarise()` regrouping output by 'Species' (override with `.groups` argument)
## # A tibble: 4 x 3
## # Groups:   Species [3]
##   Species    med_split     n
##   <fct>      <chr>     <int>
## 1 setosa     lower        50
## 2 versicolor lower        25
## 3 versicolor upper        25
## 4 virginica  upper        50
```

3.0.15 tidy aggregation

```
df %>%
```

Nimm `df`, dann …

```
  group_by(Species, med_split) %>%
```

… gruppiere nach `Species` und `med_split`, dann…

```
  summarise(n = n())
```

…Zähle die absoluten Häufigkeiten aus.

3.0.16 Aufgabe

Machen Sie sich mit dem `swiss`-Datensatz vertraut. Lesen Sie dazu auch die Hilfeseite zu dem Datensatz. Erstellen Sie mit Hilfe einer pipeline einen Datensatz, der…

1. den Anteil der männlichen Population in der Landwirtschaft, die Kindersterblichkeit, das Bildungsniveau und den Anteil der katholischen Familien in Prozent enthält
2. nur Provinzen enthält, deren Einwohner zu mehr als 10% Bestnoten bei der Armee-Untersuchung erhalten haben
3. eine numerische Variable enthält, die für die so ausgewählten Fälle einen Mediansplit der Kindersterblichkeit codiert.

Erstellen Sie anschließend eine kurze pipeline, die den Datensatz mit dem Absteigenden Bildungsniveau als ersten Sortierschlüssel und dem aufsteigenden Anteil katholischer Familien als zweitem Schlüssel sortiert. Nutzen Sie dafür die Hilfeseite der `arrange`-Funktion.

3.0.17 Regressionen in R

Viele Modelle in R benutzen eine `formula`, um die Modellparameter festzulegen. Dabei besteht jede `formula` erstmal aus einem Vorhersage- und einem

vorhergesagten Term, die durch eine Tilde getrennt werden:

`Vorhergesagt ~ Vorhersage`

Die Syntax dafür folgt dem folgenden Schema:

Operator	Bedeutung in Modellformel
+	den folgenden Vorhersageterm hinzufügen
-	den folgenden Vorhersageterm ausschließen
<A> : <B>	Interaktion AxB als Vorhersageterm
<A> * <B>	alle additiven und Interaktionseffekte
1	absoluter Term (Gesamterwartungswert)

3.0.18 Beispiel-Terme

Die Vohersage von `A` durch `B`:

`A ~ B`

Die Vohersage von `A` durch `B` und `C`:

`A ~ B + C`

Die Vohersage von `A` durch die Interaktion von `B` und `C`:

`A ~ B : C`

Die Vohersage von `A` durch `B`, `C` und die Interaktion von `B` und `C`:

`A ~ B * C`

3.0.19 Regression

Alle linearen Regressionen lassen sich mit dem `base`-Umfang von R umsetzen.

Dazu übergibt man einfach die gewünschte Modellformel der `lm`(für linear model)-Funktion. Mit dem `data`-Argument lässt sich außerdem der Datensatz festlegen, auf dessen Basis das Modell gerechnet werden soll.

Als Beispiel rechnen wir hier mal die Vorhersage der Blatt-Breite duch die Kelch-Länge der Blumen im `iris`-Datansatz.

```
model <- lm(Petal.Width ~ Sepal.Length,data = iris)
```

3.0.20 Regression

Der einfache Ausdruck des Modells sind erstmal nur die Schätzer:

```
model
```

```
##
## Call:
## lm(formula = Petal.Width ~ Sepal.Length, data = iris)
##
## Coefficients:
##  (Intercept)  Sepal.Length
##      -3.2002        0.7529
```

Mit der `summary`-Funktion lässt sich dann auch eine Regresssionsanalyse durchführen.

3.0.21 Regressionsanalyse

```
summary(model)
```

```
##
## Call:
## lm(formula = Petal.Width ~ Sepal.Length, data = iris)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.96671 -0.35936 -0.01787  0.28388  1.23329
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -3.20022    0.25689  -12.46   <2e-16
## Sepal.Length  0.75292    0.04353   17.30   <2e-16
##
## (Intercept)  ***
## Sepal.Length ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.44 on 148 degrees of freedom
## Multiple R-squared:  0.669,  Adjusted R-squared:  0.6668
## F-statistic: 299.2 on 1 and 148 DF,  p-value: < 2.2e-16
```

3.0.22 Aufgabe

Bearbeiten Sie auf Basis des eben erstellten Teil-Datensatzes von `swiss` die folgenden Aufgaben:

1. Ein lineares Regressionsmodell mit dem Bildungsniveau als Kriterium und dem Anteil katholischer Familien als Prädiktor
2. Rufen Sie die Hilfeseiten der `tibble`-Funktion auf und erstellen Sie mit

Ihrer Hilfe einen Datensatz, der Prädiktor und Kriterium enthält. Fügen Sie dann diesem Datensatz in einer `pipeline` mit Hilfe der `mutate`-Funktion eine Variable mit den vorhergesagten Werten und eine mit den Residuen hinzu. Rufen Sie dafür die Hilfeseiten der `fitted` und der `residuals`- Funktionen auf.

3.0.23 Aufgabe

3. Ein lineares Regressionsmodell mit der Kindersterblichkeit als Kriterium und den folgenden Prädiktoren:
   - dem in der Landwirtschaft arbeitenden Anteil der Bevölkerung
   - dem Bildungsniveau
   - der Interaktion des Anteils katholischer Familien und des Bildungsniveaus
4. Berechnen Sie auf Basis dieses zweiten Modells das korrigierte R^2

3.0.24 Literatur

3.0.25 Aufgabe 1

Machen Sie sich mit dem `swiss`-Datensatz vertraut. Lesen Sie dazu auch die Hilfeseite zu dem Datensatz. Erstellen Sie mit Hilfe einer pipeline einen Datensatz, der…

1. nur Provinzen enthält, deren Einwohner zu mehr als 10% und weniger als 35% Bestnoten bei der Armee-Untersuchung erhalten haben
2. den Anteil der männlichen Population in der Landwirtschaft, die Kindersterblichkeit, das Bildungsniveau und den Anteil der katholischen Familien enthält
3. eine numerische Variable enthält, die für die so ausgewählten Fälle einen Mediansplit der Kindersterblichkeit codiert.

Erstellen Sie anschließend eine kurze pipeline, die den Datensatz mit dem Absteigenden Bildungsniveau als ersten Sortierschlüssel und dem aufsteigenden Anteil katholischer Familien als zweitem Schlüssel sortiert. Nutzen Sie dafür die Hilfeseite der `arrange`-Funktion.

```
library(tidyverse)
summary(swiss)
```

```
##    Fertility      Agriculture     Examination
##  Min.   :35.00   Min.   : 1.20   Min.   : 3.00
##  1st Qu.:64.70   1st Qu.:35.90   1st Qu.:12.00
##  Median :70.40   Median :54.10   Median :16.00
##  Mean   :70.14   Mean   :50.66   Mean   :16.49
##  3rd Qu.:78.45   3rd Qu.:67.65   3rd Qu.:22.00
##  Max.   :92.50   Max.   :89.70   Max.   :37.00
```

```
##    Education        Catholic
##  Min.   : 1.00   Min.   :  2.150
##  1st Qu.: 6.00   1st Qu.:  5.195
##  Median : 8.00   Median : 15.140
##  Mean   :10.98   Mean   : 41.144
##  3rd Qu.:12.00   3rd Qu.: 93.125
##  Max.   :53.00   Max.   :100.000
##  Infant.Mortality
##  Min.   :10.80
##  1st Qu.:18.15
##  Median :20.00
##  Mean   :19.94
##  3rd Qu.:21.70
##  Max.   :26.60
```

```
sub_swiss <- swiss %>%
  filter(Examination > 10 & Examination < 35) %>%  # 1
  select(Agriculture, Infant.Mortality, Education, Catholic) %>% #2
  mutate(med_split = if_else(Infant.Mortality > median(Infant.Mortality),
                             true = 1,
                             false =  0))

help("arrange")
sub_swiss %>%
  arrange(desc(Education),
          Catholic)
```

```
##    Agriculture Infant.Mortality Education
## 1         46.6             18.2        29
## 2         27.7             19.3        29
## 3         19.4             20.2        28
## 4         15.2             10.8        20
## 5         26.8             20.9        19
## 6         43.5             20.6        15
## 7         16.7             18.9        13
## 8         45.2             24.4        13
## 9         63.1             18.1        13
## 10        60.7             22.7        12
## 11        38.4             20.3        12
## 12        62.0             16.5        12
## 13        17.0             22.2        12
## 14        50.9             16.7        12
## 15         7.7             20.5        11
## 16        59.8             18.0        10
## 17        60.8             16.3        10
## 18        73.0             20.0         9
```

```
## 19        34.0            20.0         8
## 20        58.1            23.8         8
## 21        49.5            22.5         8
## 22        67.8            24.9         8
## 23        67.5            19.1         7
## 24        37.6            20.0         7
## 25        18.7            19.5         7
## 26        36.5            20.3         7
## 27        70.2            23.6         7
## 28        53.3            21.0         7
## 29        54.1            15.3         6
## 30        64.5            24.5         6
## 31        78.2            19.4         6
## 32        69.3            18.7         5
## 33        55.1            22.4         3
## 34        72.6            21.2         2
## 35        71.2            21.0         1
##    Catholic med_split
## 1     50.43         0
## 2     58.33         0
## 3     12.11         0
## 4      2.15         0
## 5     18.46         1
## 6      5.16         1
## 7     11.22         0
## 8     91.38         1
## 9     96.83         0
## 10     4.43         1
## 11     5.62         1
## 12     8.52         0
## 13     9.96         1
## 14    15.14         0
## 15    13.79         1
## 16     5.23         0
## 17     7.72         0
## 18     2.84         0
## 19     3.30         0
## 20     5.23         1
## 21     6.10         1
## 22    97.16         1
## 23     2.27         0
## 24     4.97         0
## 25     8.65         0
## 26    33.77         1
## 27    92.85         1
## 28    97.67         1
```

```
## 29     4.20         0
## 30    98.61         1
## 31    98.96         0
## 32     2.82         0
## 33     4.52         1
## 34    24.20         1
## 35     2.40         1
```

3.0.26 Aufgabe 2

Bearbeiten Sie auf Basis des eben erstellten Teil-Datensatzes von `swiss` die folgenden Aufgaben:

1. Ein lineares Regressionsmodell mit dem Bildungsniveau als Kriterium und dem Anteil katholischer Familien als Prädiktor
2. Erstellen Sie einen Datensatz mit dem Namen `reg_swiss`, der Prädiktor und Kriterium enthält. Erstellen Sie dafür auch wieder eine pipeline, in die Sie Ihren Unterdatensatz geben. Fügen Sie dann diesem Datensatz in einer `pipeline` mit Hilfe der `mutate`-Funktion eine Variable mit den vorhergesagten Werten und eine mit den Residuen hinzu. Rufen Sie dafür die Hilfeseiten der `fitted` und der `residuals`- Funktionen auf.
3. Ein lineares Regressionsmodell mit der Kindersterblichkeit als Kriterium und den folgenden Prädiktoren:
   - dem in der Landwirtschaft arbeitenden Anteil der Bevölkerung
   - dem Bildungsniveau
   - der Interaktion des Anteils katholischer Familien und des Bildungsniveaus
4. Lassen Sie sich für dieses zweiten Modells das korrigierte R^2 ausgeben

```
model_1 <- lm(Education ~ Catholic,data=sub_swiss)

help('fitted')
help("residuals")

reg_swiss <- sub_swiss %>%
  select(Education, Catholic) %>%
  mutate(fitted = fitted(model_1),
         resid = residuals(model_1))

model_2 <- lm(Infant.Mortality ~ Agriculture + Education + Education:Catholic,data = sub_swiss)

summary(model_2)
```

```
##
## Call:
```

```
## lm(formula = Infant.Mortality ~ Agriculture + Education + Education:Catholic,
##     data = sub_swiss)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.7721 -1.2469 -0.1436  2.2176  4.0171
##
## Coefficients:
##                     Estimate Std. Error t value
## (Intercept)        22.476619   2.115070  10.627
## Agriculture        -0.015453   0.029344  -0.527
## Education          -0.239759   0.100405  -2.388
## Education:Catholic  0.002921   0.001301   2.246
##                    Pr(>|t|)
## (Intercept)        7.36e-12 ***
## Agriculture           0.6022
## Education             0.0232 *
## Education:Catholic    0.0320 *
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.671 on 31 degrees of freedom
## Multiple R-squared:  0.2081, Adjusted R-squared:  0.1314
## F-statistic: 2.715 on 3 and 31 DF,  p-value: 0.06169
```

Chapter 4

Daten einlesen

4.0.1 Einlesen von Daten

Das Rechnen mit den mit R mitgelieferten Datensätzen ist natürlich nur bedingt realitätsnah.

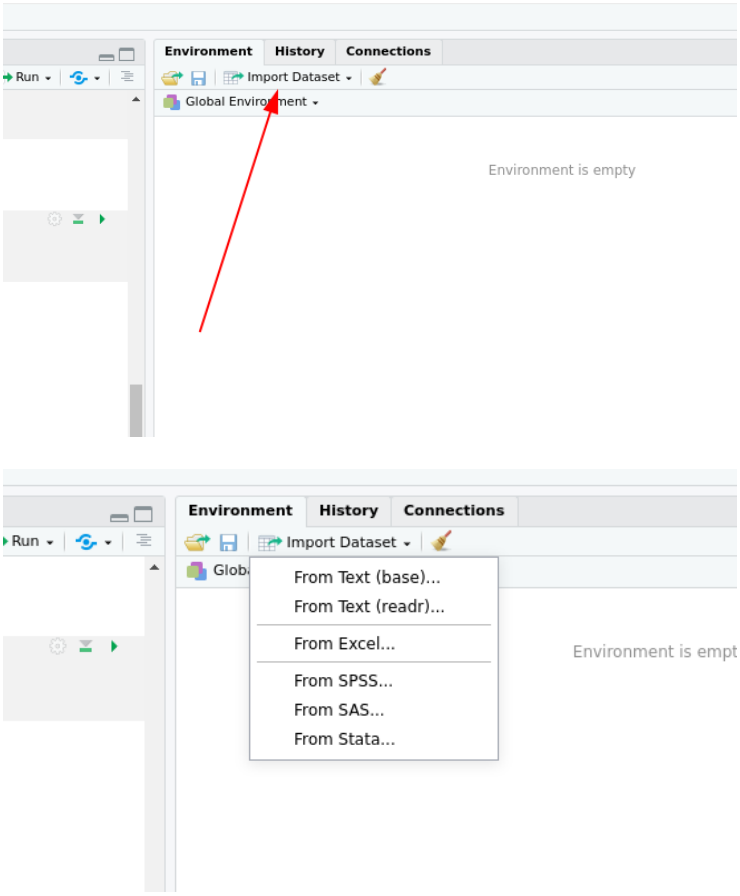
Im durchschnittlichen Anwendungsfall müssen externe Datensätze eingelesen werden.

Dabei sind im `tidyverse` dafür je nach Quelle folgende Pakete vorgesehen:

- Textbasierte Daten(`.txt, .csv, .tsv,...`) → `readr`
- Excel-Mappen(`.xlsx, .xls`) → `readxl`
- Daten aus anderen Statistikpaketen(`.sav, .dta,...`) → `haven`

4.0.2 Einlesen von Textdaten

Alle diese drei Pakete sind auch in der RStudio-GUI implementiert:



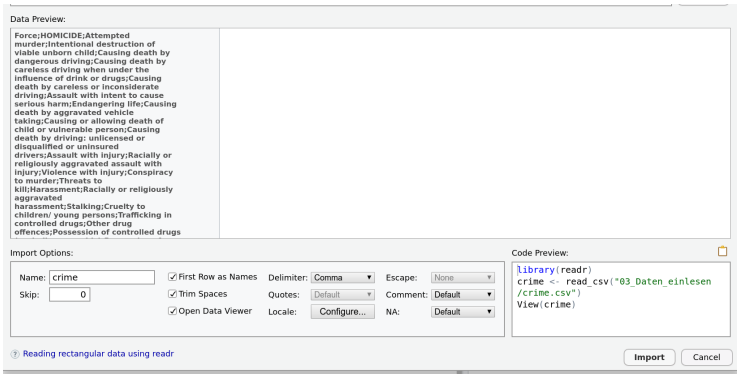
4.0.3 Problem

Das Einlesen und Aufbereiten wird am folgenden Beispiel exerziert: Uns interessiert der Zusammenhang von Drogenmissbrauch, Lebenszufriedenheit und Straftaten in Großbritannien. Dafür haben wir die folgenden drei Datensätz zur Verfügung:

- `'crime.csv'` - Eine Textdatei mit nach Polizeibehörde aufgeschlüsselten Straftaten
- `'drugs.xlsx'` - Eine Excel-Arbeitsmappe mit nach Region aufgeschlüsselten Zahlen zu Krankenhauseinweisungen mit drogenbedingten Diagnosen
- `'satisfaction.sav'` - Ein in SPSS erstellter Datensatz mit nach Region aufgeschlüsselten Ergebnissen einer Bevölkerungsbefragung zur Lebenszufriedenheit

4.0.4 textbasierte Daten

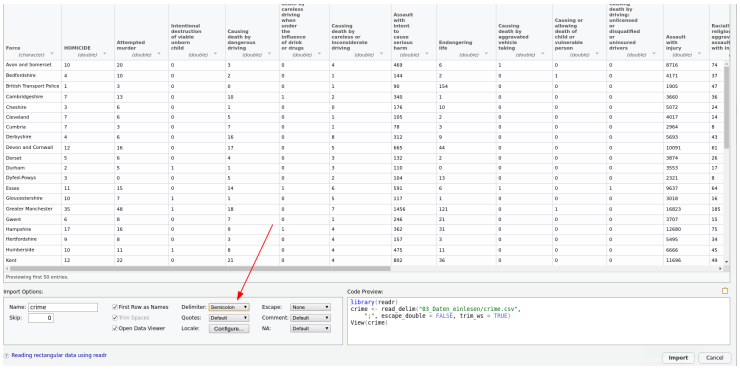
Die GUI ist hier ein guter Start. Wir wollen die Datei `'crime.csv'` einlesen. Diese enthält echte Daten über von britischen Polizeibehörden aufgezeichnete Straftaten von der Website der britischen Regierung. Wenn ich dem Pfad im GUI folge, ergibt sich das folgende Bild:



4.0.5 Einlesen von Textdaten

Was ist das Problem?

Das Trennzeichen(Delimiter) ist falsch gesetzt. In den Daten sind die Zellen offensichtlich durch Semikolons getrennt.



4.0.6 Einlesen von Textdateien

Der für das Einlesen nötige Code wird dann von RStudio in die Konsole kopiert und ausgeführt. Um nicht jedes Mal beim Ausführen desselben Skriptes wieder per Hand den Datensatz einlesen zu müssen, kopiert man den Code dann an den Anfang des Skriptes.

```
> library(readr)
> crime <- read_delim("03_Daten_einlesen/crime.csv",
+     ";", escape_double = FALSE, trim_ws = TRUE)
Parsed with column specification:
cols(
  .default = col_double(),
  Force = col_character()
)
See spec(...) for full column specifications.
> View(crime)
>
```

4.0.7 Einlesen von Textdateien

Was passiert hier?

```
crime <- read_delim("data/crime.csv",
```

Lege in `crime` das Textfile mit Trennzeichen unter dem angegebenen Pfad ab. Dabei…

```
";", escape_double = FALSE, trim_ws = TRUE)
```

…erwarte Semikolons als Trennzeichen, erwarte keine doppelten Anführungszeichen und schneide Leerzeichen von den Einträgen ab.

```
##
## -- Column specification --------------------------
## cols(
##   .default = col_double(),
##   Force = col_character()
## )
## i Use `spec()` for the full column specifications.
```

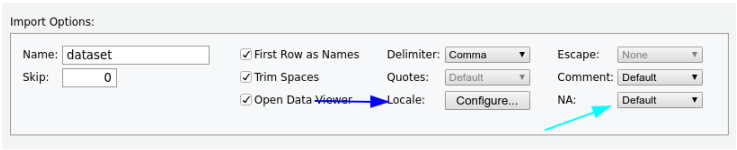
R teilt mit, dass es Kommazahlen als Standard-Zelleninhalt versucht und bei nicht-Funktionieren auf `character` zurückfällt. Das ist trotz der Farbe **keine** Fehlermeldung

```
View(crime)
```

Dann öffne den Datensatz zum Angucken.

4.0.8 Einlesen von Textdateien

Noch zwei wichtige Tricks in dem Einlesetool sind die locale-Schaltfläche und das NA-Menü



Was kann man hier anpassen?

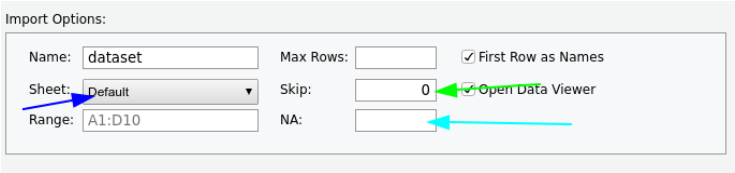
Genau kontrollieren, ob Umlaute, Zellentrennung, fehlende Werte, Dezimaltrennzeichen,… richtig eingestellt waren!

4.0.9 Excel-Arbeitsmappen

Für die Excel-Arbeitsmappen ist die GUI auch der einfachste Weg.

Wie würde man vorgehen um die Datei drugs.xlsx einzulesen?

- Import Dataset → From Excel
- Pfad zum file raussuchen



- Richtiges Sheet aussuchen
- unnötige Zeilen überspringen
- etwaige von leeren Zellen abweichende NA-Codierung anpassen

4.0.10 Excel-Arbeitsmappen

Auch bei Excel-Mappen an das Kopieren des Codes denken!

```
library(readxl)
drugs <- read_excel("data/drugs.xlsx",
                    sheet = "Table 2", na = "*", skip = 10)
```

```
## New names:
## * `` -> ...1
## * `` -> ...2
## * `` -> ...3
## * `` -> ...4
```

Diese Daten sind übrigens auch Originaldaten von der Website des britischen National Health Services

4.0.11 Dateien aus anderer Software

Beispielhaft für SPSS, für Stata etc analog. Die GUI ist wieder ein guter Anfang und hier ziemlich selbsterklärend.

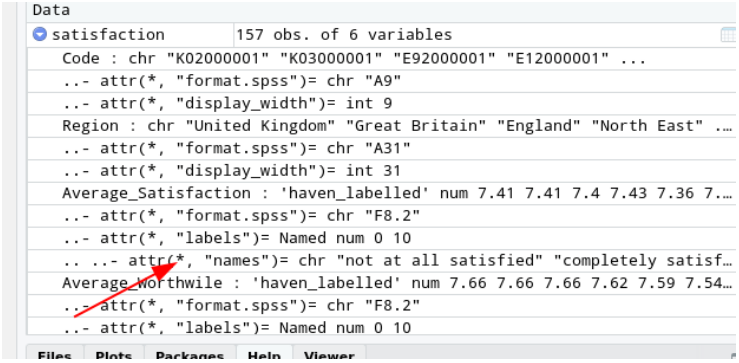
Wie würde man vorgehen um die Datei satisfaction.sav einzulesen?

```
library(haven)
satisfaction <- read_sav("data/satisfaction.sav")
```

Die Daten kommen diesmal vom britischen Office for National Statistics, wurden aber stark abgewandelt.

4.0.12 Dateien aus SPSS einlesen

Wenn man sich die Daten in der RStudio-Oberfläche anguckt, sieht man, dass die für SPSS typischen Variablendefinitionen konserviert wurden:



4.0.13 Dateien aus SPSS einlesen

`haven` bietet mit der `as_factor`-Funktion eine Möglichkeit an, eine dieser Codierung enthaltenden Variablen in einen Faktor umzuwandeln.

Faktoren sind eine Variante um in R kategoriale Variablen anzulegen.

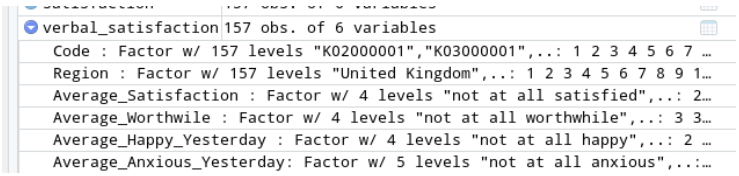
So könnten wir uns zum Beispiel entscheiden, einen neuen, zweiten Datensatz zu erstellen, der die Variablen mit den Verballabels aus SPSS enthält. Da wir auf alle Spalten dafür dieselbe Funktion anwenden wollen, können wir dafür die `mutate_all`-Funktion verwenden.

4.0.14 Dateien aus SPSS einlesen

Dabei benutzen wir die im `tidyverse` beim Batchen von Funktionen verbreitete Funktionsschreibweise:

```
verbal_satisfaction <- satisfaction %>%
  mutate_all(~as_factor(.)) ## Nach der Tilde Funktion für alle Spalten
## Mit dem Punkt wird angegeben, wo die Variablen eingesetzt werden sollen
```

Das Ergebnis sieht in der Oberfläche dann so aus:



Für Tipps zur weitergehenden Bearbeitung von SPSS und Stata-Daten noch hier die sehr gute `haven`-Vignette zu den nötigen Schritten.

4.0.15 Datenaufbereitung

Datenaufbrereitung kann natürlich denkbar komplex sein, deswegen beschränken wir uns hier auf die folgenden Schritte:

1. Ausreißer-Behandlung
2. Umgang mit fehlenden Werten
3. Recodieren von Werten

4.0.16 Ausreißer-Behandlung

Als ersten Schritt zur Bereinigung der drei Datensätze sollen Ausreißer erkannt und durch fehlende Werte ausgeschlossen werden.

Dafür muss man sich natürlich zuerst überlegen, was das Kriterium dafür sein soll. Wir benutzen hier das Kriterium nach Tukey (1977), also wollen wir gerade die Werte ausschließen, die mehr als 1.5 Interquartilabstände über oder unter dem 25% bzw dem 75%-Quantil liegen.

Um uns Tipparbeit zu sparen, schreiben wir dafür unsere erste Funktion:

```
remove_outlier <- function(x){
  ifelse(between(x,
                 quantile(x,.25) - 1.5 * IQR(x),
                 quantile(x,.75) + 1.5 * IQR(x)),
         x,
         NA)}
```

4.0.17 Ausreißer-Behandlung

Kombiniert mit einem `mutate_if` können wir damit alle Ausreißer gegen fehlende Werte austauschen. `mutate_if` ist mit `mutate_all` vergleichbar, nur dass man einen logischen Wert übergeben kann um die umzuwandelnden Spalten auszuwählen.

```
crime <- crime %>%
  mutate_if(.predicate = is.numeric,
            ~remove_outlier(.))
```

4.0.18 Umgang mit fehlenden Werten

Fehlende Werte werden in R generell mit `NA` codiert. Um damit umzugehen bietet das `tidyverse` ein paar Funktionen, wir beschränken uns hier auf zwei.

`drop_na` zum rigorosen Entfernen von Zeilen mit fehlenden Werten:

```
drugs %>%
  drop_na()
```

```
## # A tibble: 0 x 7
## # ... with 7 variables: ...1 <chr>, ...2 <chr>,
## #   ...3 <chr>, ...4 <chr>, `All persons9` <dbl>,
## #   Male <dbl>, Female <dbl>
```

...in unserem Fall vielleicht ein bisschen zu rigoros

4.0.19 Umgang mit fehlenden Werten

Die zweite Möglichkeit ist `replace_na`, eine Funktion die, wie der Name schon sagt, `NA`s durch festgelegte Werte ersetzen kann. Mit einem `mutate_if` kombiniert, können wir so alle fehlenden Zahlen im Datensatz durch 0 ersetzen:

```
drugs %>%
  mutate_if(is.numeric,~replace_na(.,0))
```

```
## # A tibble: 195 x 7
##   ...1  ...2  ...3  ...4  `All persons9`  Male
##   <chr> <chr> <chr> <chr>          <dbl> <dbl>
## 1 <NA>  <NA>  <NA>  <NA>               0     0
## 2 E920~ <NA>  <NA>  Engl~           7139  5294
## 3 <NA>  <NA>  <NA>  <NA>               0     0
## 4 U     <NA>  U     Unkn~            244   202
## 5 <NA>  <NA>  <NA>  <NA>               0     0
## # ... with 190 more rows, and 1 more variable:
## #   Female <dbl>
```

4.0.20 Umgang mit fehlenden Werten

Jetzt können wir noch die fehlenden `character` umgewandeln:

```
drugs <- drugs %>%
  mutate_if(is.numeric,~replace_na(.,0)) %>%
  mutate_if(is.character,~replace_na(.,''))
drugs
```

```
## # A tibble: 195 x 7
##   ...1  ...2  ...3  ...4  `All persons9`  Male
##   <chr> <chr> <chr> <chr>          <dbl> <dbl>
## 1 ""    ""    ""    ""                 0     0
## 2 "E92~ ""    ""    "Eng~           7139  5294
## 3 ""    ""    ""    ""                 0     0
## 4 "U"   ""    "U"   "Unk~            244   202
## 5 ""    ""    ""    ""                 0     0
## # ... with 190 more rows, and 1 more variable:
```

```
## #   Female <dbl>
```

4.0.21 Recodieren von Werten

Das Recodieren von Werten funktioniert einfach über eine `mutate`-pipeline.

Für Kategoriale Daten bietet das `tidyverse` die `recode`-Funktion, hier wollen wir uns aber auf numerische Daten beschränken.

Wie könnte ich die *Anxiety*-Skala im `satisfaction`-Datensatz umpolen?

```
satisfaction <- satisfaction %>%
  mutate(Average_Anxious_Yesterday = -1* (Average_Anxious_Yesterday-10))
```

4.0.22 Zusammenführen von Datensätzen

Zu guter Letzt wollen wir die drei Datensätze zu einem Datensatz zusammenfügen.
Dieser soll pro Region 1. die Anzahl aller drogenbezogenen Krankenhausaufenthalte 2. die Anzahl der (versuchten) Mordfälle 3. die mittlere Zufriedenheit über alle Skalen beinhalten

4.0.23 Vorbereitung

Wir fangen damit an, die Datensätze wie gewünscht vorzubereiten. Aus dem `drugs`-Datensatz brauchen wir die Regionsbezeichnung, die ONS-Codes und natürlich die Zahl der Einweisungen:

```
drugs <- drugs %>%
  select(...1,...4, `All persons9`)
```

4.0.24 Vorbereitung

Aus dem `crime`-Datensatz brauchen wir die Bezeichnung der Niederlassung, Anzahl der Morde und die Anzahl der versuchten Morde:

```
crime <- crime %>%
  select(Force,HOMICIDE,`Attempted murder`)
```

4.0.25 Vorbereitung

Aus dem `satisfaction`-Datensatz brauchen wir den ONS-Code und einen mittleren Zufriedenheitswert

Wie könnte ich das angehen?

```
satisfaction <- satisfaction %>%
  transmute(Code = Code,
```

```
            Satisfaction = (Average_Satisfaction + Average_Worthwile +
              Average_Happy_Yesterday +Average_Anxious_Yesterday)/4)
```

`sum` ist keine vektorisierte Funktion! Um eine neue Summenwert pro Zeile zu bilden, sind `+` und `/` nötig

4.0.26 Zusammenführen von Datensätzen

Jetzt müssen wir das ganze nur noch zusammenfügen. Dafür benutzen wir die Familien der `join`-Funktionen

Zuerst fügen wir die Anzahl der Straftaten zu der Anzahl der Krankenhauseinweisungen hinzu. Dabei matchen wir die Regionen über das Regions-Schlüsselwort und behalten nur die Fälle, in denen in beiden Datensätzen ein Schlüsselwort auftaucht:

```
overall <- drugs %>%
  inner_join(crime, by = c('...4' = 'Force'))
overall
```

```
## # A tibble: 21 x 5
##   ...1  ...4  `All persons9` HOMICIDE
##   <chr> <chr>          <dbl>    <dbl>
## 1 E100~ Cumb~             72        7
## 2 E100~ Lanc~            214       16
## 3 E100~ Nort~             44        9
## 4 E100~ Derb~             81        4
## 5 E100~ Leic~             56        6
## # ... with 16 more rows, and 1 more variable:
## #   `Attempted murder` <dbl>
```

4.0.27 Zusammenführen von Datensätzen

Dem `overall`-Datensatz fügen wir jetzt noch die `satisfaction` hinzu. Hierzu nutzen wir die ONS-Codes. Dabei wollen wir alle Fälle in `overall` behalten:

```
overall <- overall %>%
  left_join(satisfaction, by = c('...1' = 'Code'))
overall
```

```
## # A tibble: 21 x 6
##   ...1  ...4  `All persons9` HOMICIDE
##   <chr> <chr>          <dbl>    <dbl>
## 1 E100~ Cumb~             72        7
## 2 E100~ Lanc~            214       16
## 3 E100~ Nort~             44        9
## 4 E100~ Derb~             81        4
## 5 E100~ Leic~             56        6
```

```
## # ... with 16 more rows, and 2 more variables:
## #   `Attempted murder` <dbl>, Satisfaction <dbl>
```

4.0.28 Zusammenführen von Datensätzen

Alles was jetzt noch fehlt, sind schönere Spaltennamen. Dafür benutzen wir entweder `rename`. `rename` erwartet die Angabe jedes Namens, der geändert werden soll als Wert und die neuen Namen als Namen der Argumente:

```
overall <- overall %>%
  rename('ONS_Code' = '...1',
         'region' = '...4',
         'admissions' = 'All persons9',
         'homicide' = 'HOMICIDE')
```

4.0.29 Zusammenführen von Datensätzen

Damit ist unser Datensatz fertig:

```
overall
```

```
## # A tibble: 21 x 6
##   ONS_Code region admissions homicide
##   <chr>    <chr>       <dbl>    <dbl>
## 1 E100000~ Cumbr~         72        7
## 2 E100000~ Lanca~        214       16
## 3 E100000~ North~         44        9
## 4 E100000~ Derby~         81        4
## 5 E100000~ Leice~         56        6
## # ... with 16 more rows, and 2 more variables:
## #   `Attempted murder` <dbl>, Satisfaction <dbl>
```

Den speichern wir noch eben als csv-Datei ab.

```
overall %>% write_csv('data/drugs_crime_UK.csv')
```

4.0.30 Literatur

Chapter 5

Daten darstellen

5.0.1 pivotieren von Datensätzen

Als Vorbereitung auf die Darstellung von Daten brauchen wir noch eine Funktion.

Für das Grafikpaket, das wir benutzen wollen, müssen die Daten im `long format` vorliegen. Das heißt, dass jede Variable eine Spalte und jede Zeile eine Beobachtung darstellt.

Insbesondere müssen wir darauf achten, dass alle Werte, die wir zum Beispiel an einer Achse darstellen wollen in einer Variable vorliegen.

5.0.2 Vorbereitung zum Pivotieren

Als Beispiel wollen wir den `'crime'`-Datensatz pivotieren. Um das ganze übersichtlicher zu halten, bereiten wir den Datensatz aber noch ein bisschen vor.

Wir wollen dafür

1. Den Datensatz einlesen
2. Die Spalten `Force`, `HOMICIDE`, `Attempted murder`, `Violence with injury` und `Assault with injury` einlesen.
3. Die Mord-Spalte so umbenennen, dass der Name in das restliche Schema passt.
4. Die `Total`-Zeile ausschließen.

5.0.3 Vorbereitung zum Pivotieren

Wie machen wir das?

```
library(tidyverse)
crime <- read_delim("data/crime_plot.csv",
```

```
                    ";", escape_double = FALSE, trim_ws = TRUE) %>%
  select(Force,HOMICIDE,Country,`Attempted murder`,
         `Violence with injury`, `Assault with injury`) %>%
  rename('Homicide' = 'HOMICIDE') %>%
  filter(Force != 'Total')
glimpse(crime)
```

```
## Rows: 44
## Columns: 6
## $ Force                  <chr> "Avon and Some...
## $ Homicide               <dbl> 10, 4, 1, 7, 3...
## $ Country                <chr> "England", "En...
## $ `Attempted murder`     <dbl> 20, 10, 3, 13,...
## $ `Violence with injury` <dbl> 9293, 4368, 22...
## $ `Assault with injury`  <dbl> 8716, 4171, 19...
```

5.0.4 pivotieren von Datensätzen

Um den Datensatz in ein längeres Format zu pivotieren, benutzen wir die `pivot_longer`-Funktion. Wir erstellen dafür hier einen zweiten Datensatz

```
crime_long <- crime %>%
  pivot_longer(
    cols = c('Homicide','Attempted murder',
             'Violence with injury', 'Assault with injury'),
    names_to = 'offence',
    values_to = 'count')
glimpse(crime_long)
```

```
## Rows: 176
## Columns: 4
## $ Force   <chr> "Avon and Somerset", "Avon an...
## $ Country <chr> "England", "England", "Englan...
## $ offence <chr> "Homicide", "Attempted murder...
## $ count   <dbl> 10, 20, 9293, 8716, 4, 10, 43...
```

5.0.5 Vorbereitung für die grafische Darstellung

Abschließend fügen wir noch für später Auswertungen eine Variable hinzu, die codiert, ob die Straftat in einer Verletzung ausgegangen ist oder versuchter/erfolgreicher Mord ist. Dafür benutzen wir die `str_detect`-Funktion.

```
crime_long <- crime_long %>%
  mutate(type_of_offence = ifelse(str_detect(offence, 'injury'),
                                  'injury',
                                  '(attempted) homicide'))
glimpse(crime)
```

```
## Rows: 44
## Columns: 6
## $ Force                  <chr> "Avon and Some...
## $ Homicide               <dbl> 10, 4, 1, 7, 3...
## $ Country                <chr> "England", "En...
## $ `Attempted murder`     <dbl> 20, 10, 3, 13,...
## $ `Violence with injury` <dbl> 9293, 4368, 22...
## $ `Assault with injury`  <dbl> 8716, 4171, 19...
```

5.0.6 ggplot2

Eins der stärksten Argumente für die Benutzung von R und dem `tidyverse` ist das Grafik-Paket `ggplot2`.

Mit ein bisschen Gewöhnung macht `ggplot2` es sehr einfach, hübsche Grafiken zu erstellen.

Die Syntax für `ggplot2` ist dabei aber ein bisschen anders als die, die wir bisher von R gewohnt sind.

Dafür müssen wir zuerst eine Grundebene erstellen, auf die wir die Grafik anschließend layern können.

Diese Grundebene kann man sich ein bisschen wie eine leere Leinwand vorstellen. Dabei wird beim Erstellen der 'Leinwand' direkt festgelegt, auf welchen Daten die Abbildung basieren soll und welche Variablen wie dargestellt werden sollen.

5.0.7 ggplot2

Diese Leinwand erstellt man mit der `ggplot`-Funktion, in die man, wie in die meisten `tidyverse`-Funktionen, pipen kann.

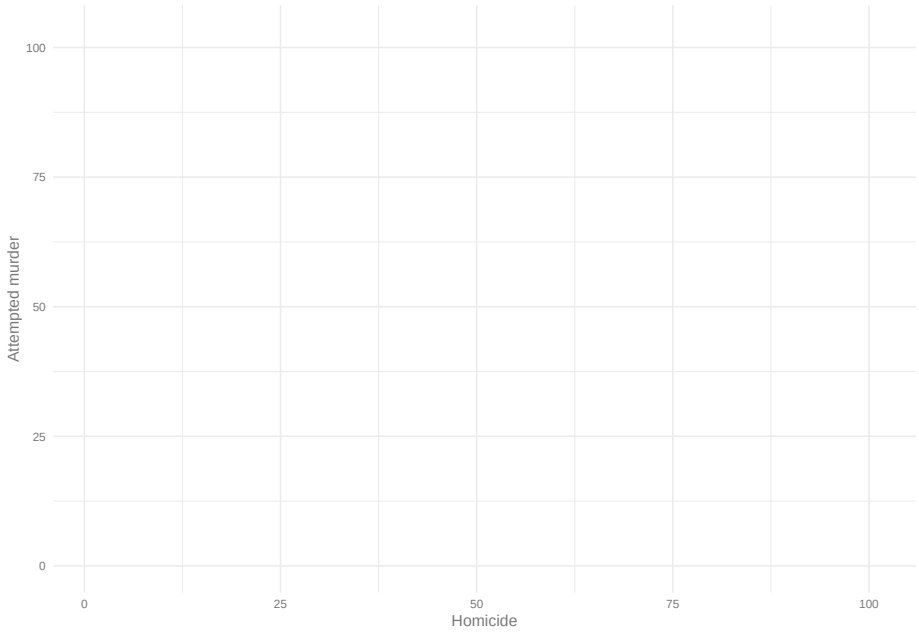
Als zweites Argument nach dem Datensatz erwartet `ggplot` eine Angabe, wie welche Variablen dargestellt werden sollen. Diese Angaben *müssen* mit `aes` für aesthetics erstellt werden:

```
crime_plot <- crime %>%
  ggplot(aes(x = Homicide, y = `Attempted murder`))
```

5.0.8 ggplot2

Diese 'leere Leinwand' sieht so aus:

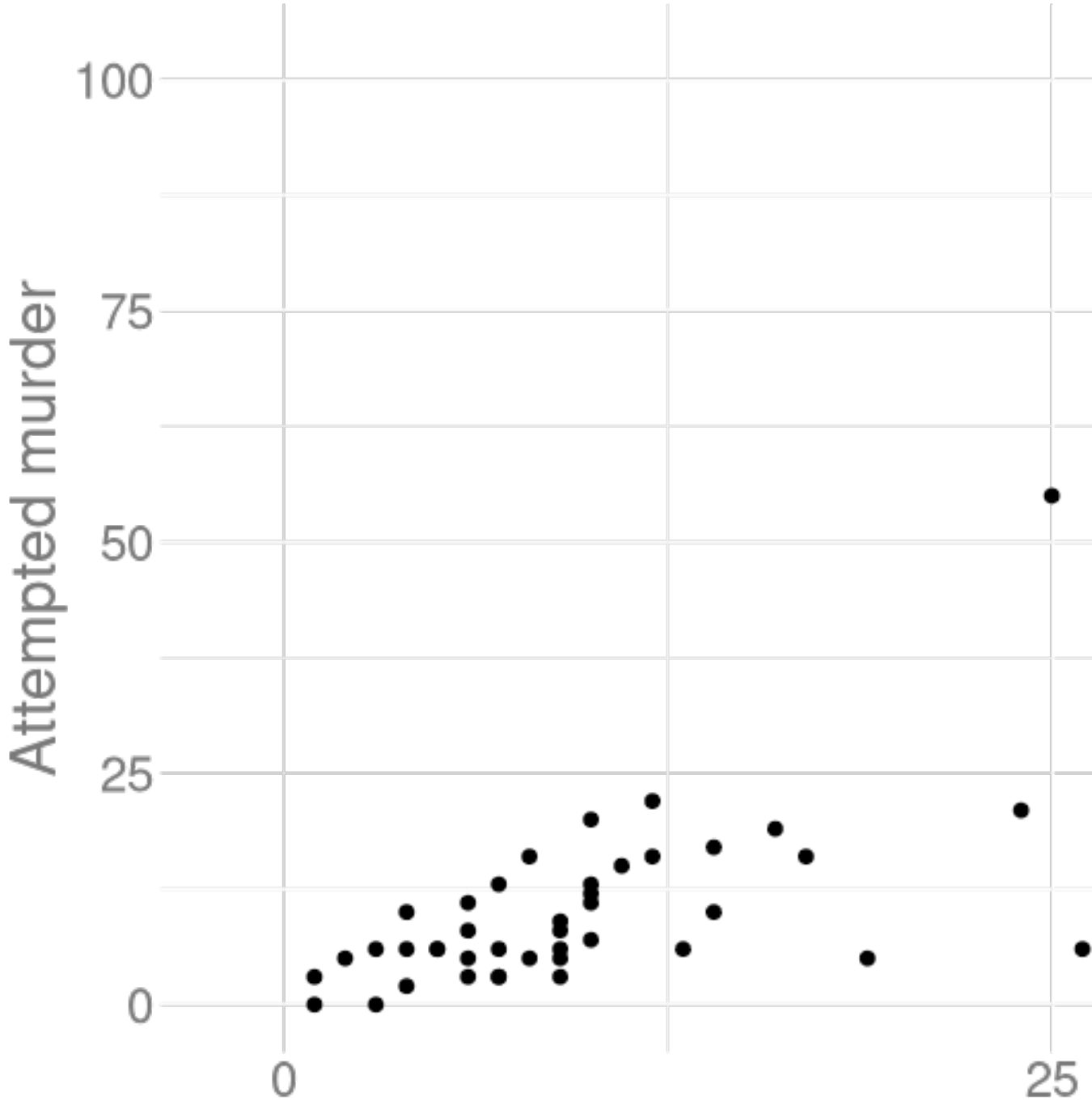
```
crime_plot
```



5.0.9 aesthetics

Auf diese Leinwand können wir dann eine Reihe von verschiedenen grafischen Elementen legen, den so genannten `geom`s. Das einfachste Beispiel auf der eben erstellten Leinwand ist ein Scatterplot. Um der Leinwand Punkte hinzuzufügen, *addieren* wir einfach einen `geom_point`-Layer auf die Grafik:

```
crime_plot + geom_point()
```


Attempted murder
100
75
50
25
0
0
25

5.0.10 aesthetics

Dieser sehr einfache Graph ist aber natürlich nicht alles. Die `aes`- und die `ggplot`-Funktionen können noch eine ganze Reihe an weiteren grafischen Parametern annehmen.

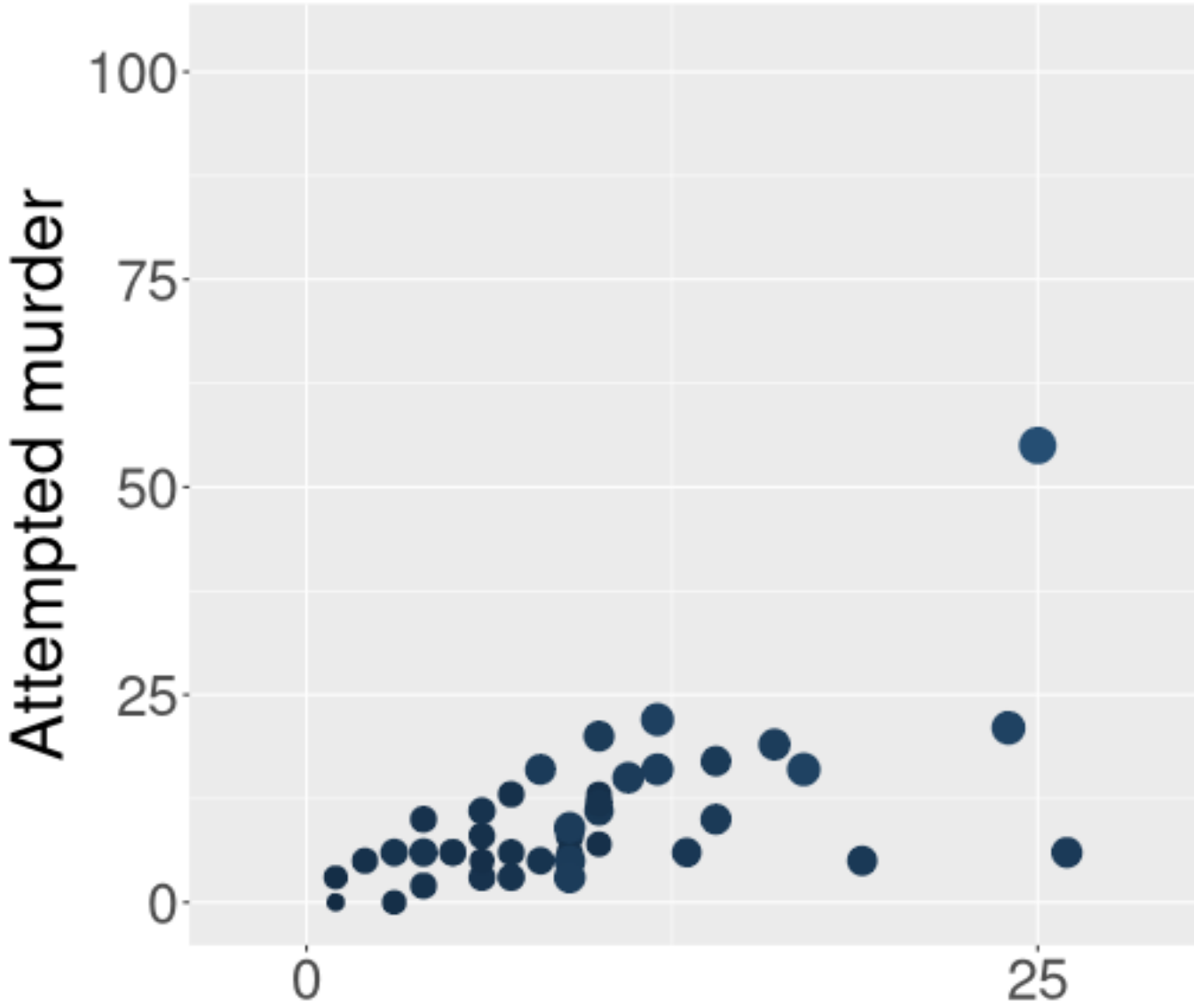
Die für unseren Graphen attraktivsten sind:

- `size` - für die Punktgröße
- `color` - für die Farbe der Punkte
- `shape` - für die Wahl der Symbole

5.0.11 aesthetics

Jedes `geom` hat auch die Möglichkeit, Daten und `aesthetics` zu nehmen. Wenn keine gesetzt werden, werden einfach die des ursprünglichen `ggplot`-Aufrufs übernommen.

```
crime_plot + geom_point(aes(size = `Violence with injury`,
                            color = `Assault with injury`))
```

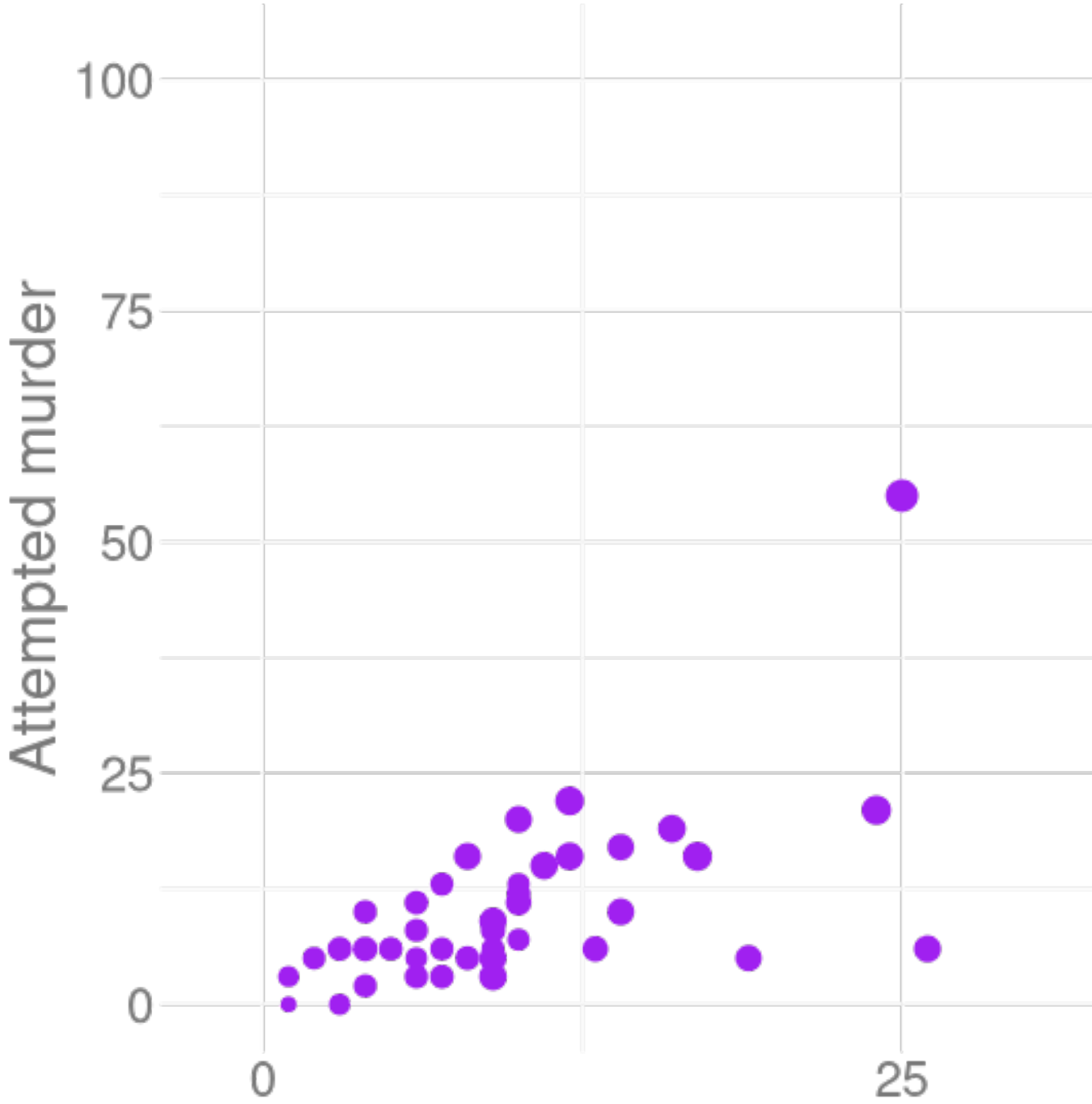


Die aus mehreren Worten bestehenden Variablennamen sind mit Gravis eingeschlossen, nicht mit Anführungszeichen!

5.0.12 aesthetics

Wenn `aesthetic`-Argumente außerhalb der `aes`-Funktion gesetzt werden, geben sie einen konstanten Wert für das `geom` an:

```
crime_plot + geom_point(aes(size = `Violence with injury`),
                            color = 'purple')
```


Attempted murder
100
75
50
25
0
0
25

5.0.13 geoms

Um andere Grafiken zu erstellen, ersetzt man einfach das `geom_point`-geom durch ein anderes.

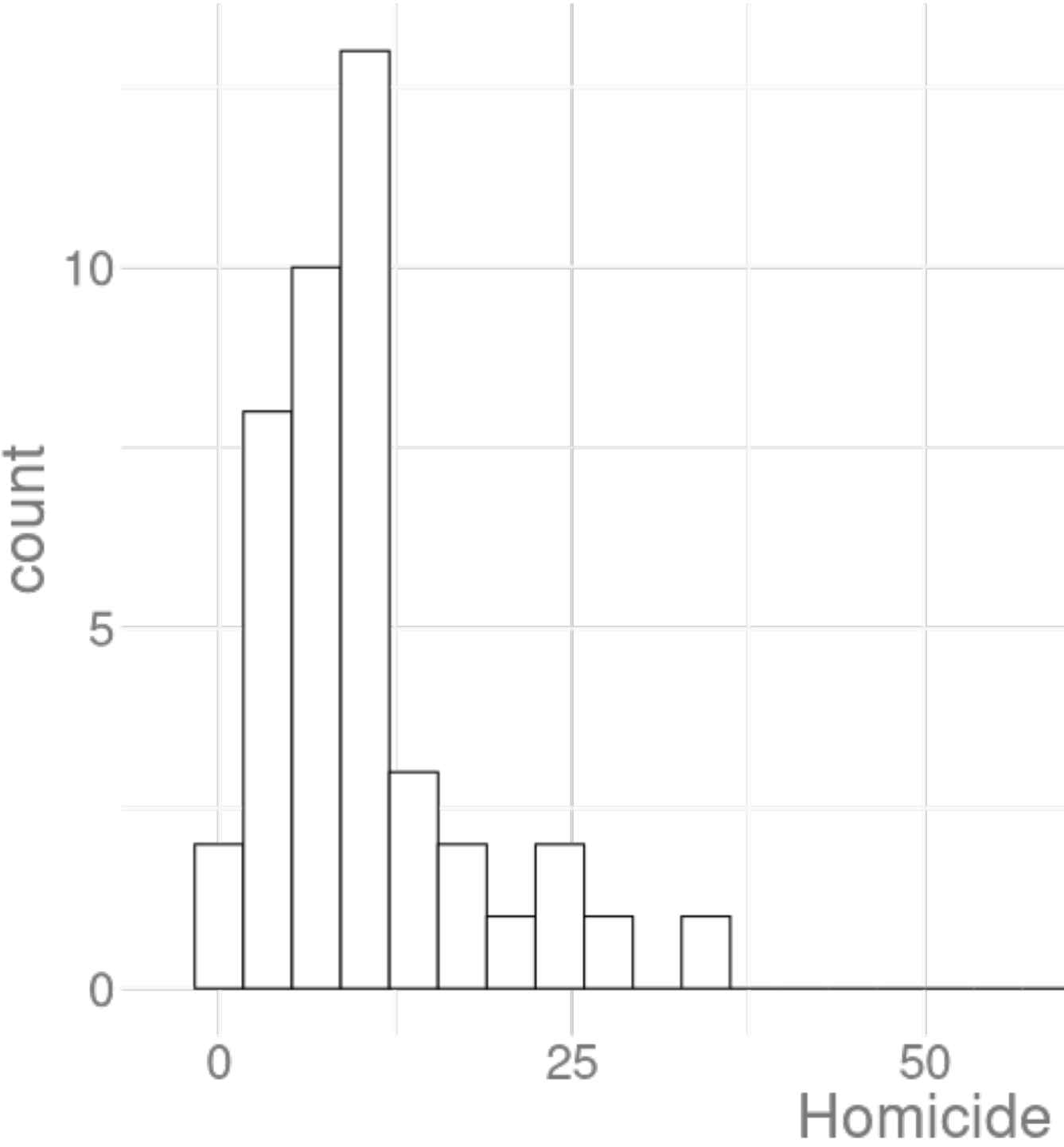
Dabei kann es natürlich nötig sein, eine andere Leinwand zu definieren.

5.0.14 geoms

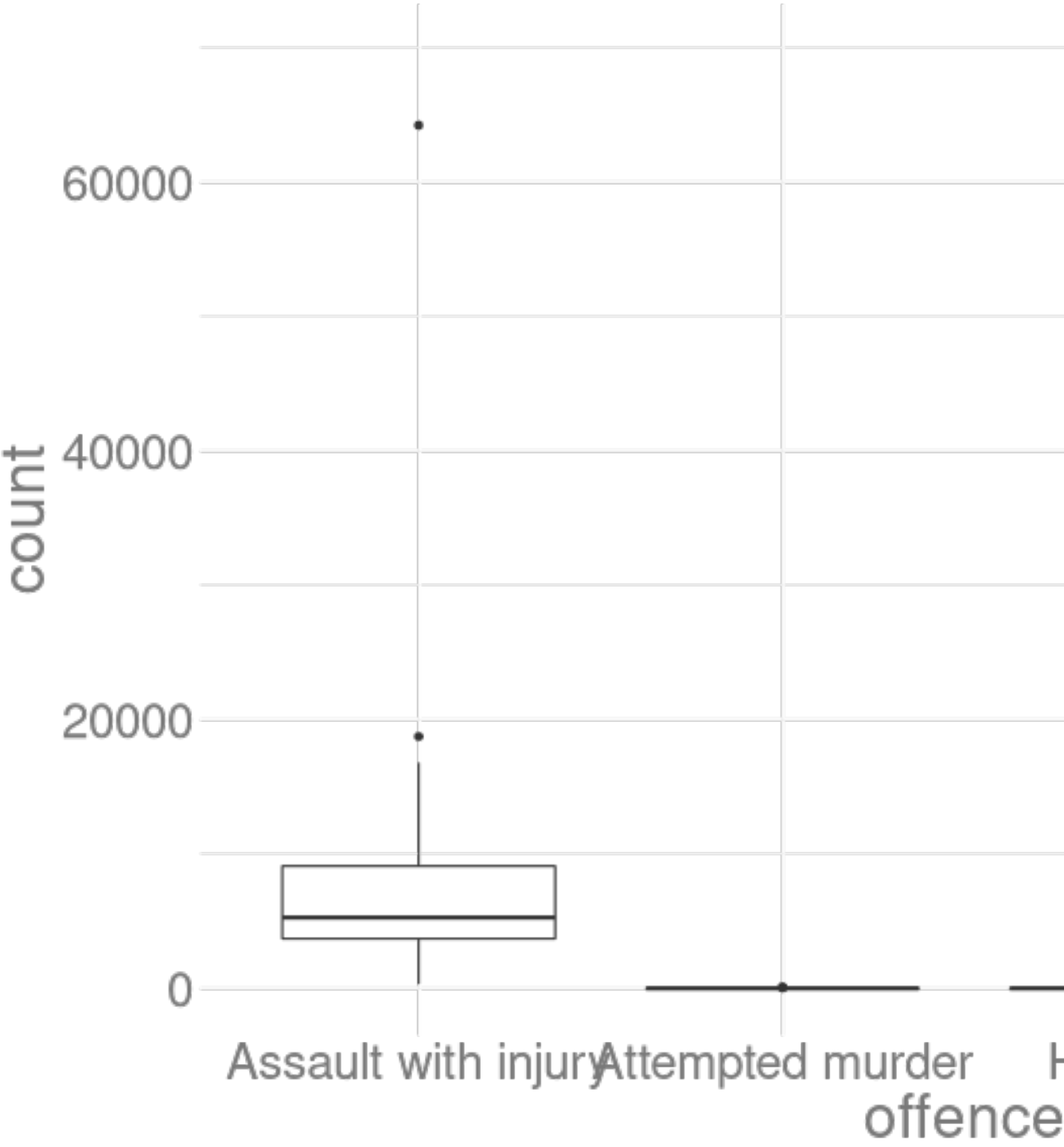
Histogramme, Boxplots und Barcharts: [1]

```
crime %>%
  ggplot(aes(x = Homicide)) +
  geom_histogram(fill = 'white',
                 color = 'black',
                 binwidth = 3)
```

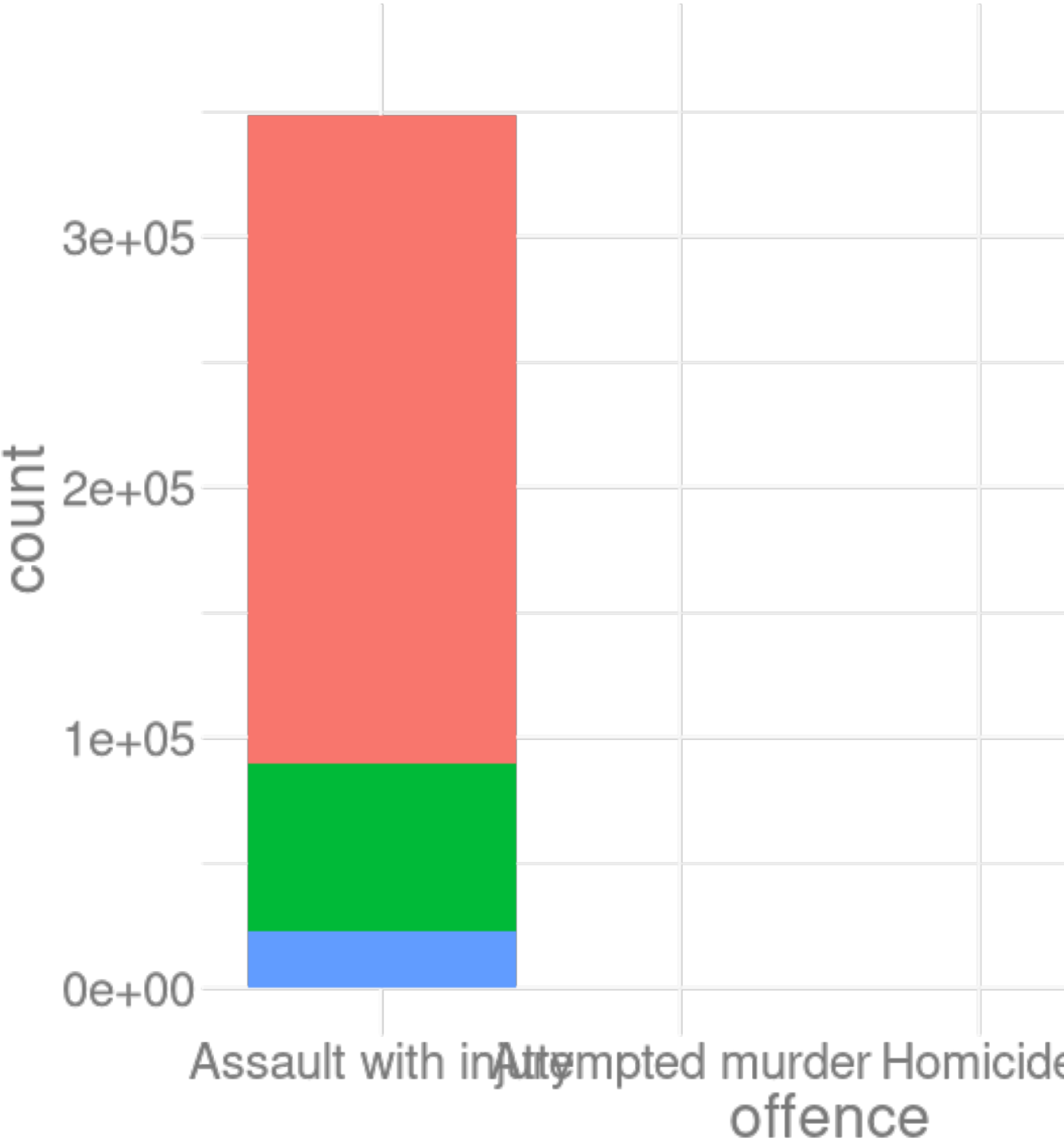
[1] Für über eine Übersicht über mehr mögliche geoms lässt sich das ggplot2-cheatsheet empfehlen.


count
10
5
0
0
25
50
Homicide

```
crime_long %>%
  ggplot(aes(y = count,
             x = offence)) +
  geom_boxplot()
```


60000
40000
20000
0
count
Assault with injury
Attempted murder
offence

```
crime_long %>%
  ggplot(aes(y = count,
             x = offence
             )) +
  geom_col()
```


3e+05
2e+05
1e+05
0e+00
count
Assault with inj
Attempted murder
Homicide
offence

5.0.15 aesthetics

Der Barchart ist ein guter Anlass, den Plot zu optimieren.

Uns stehen die folgenden Informationen zur Verfügung:

```
glimpse(crime_long)
```

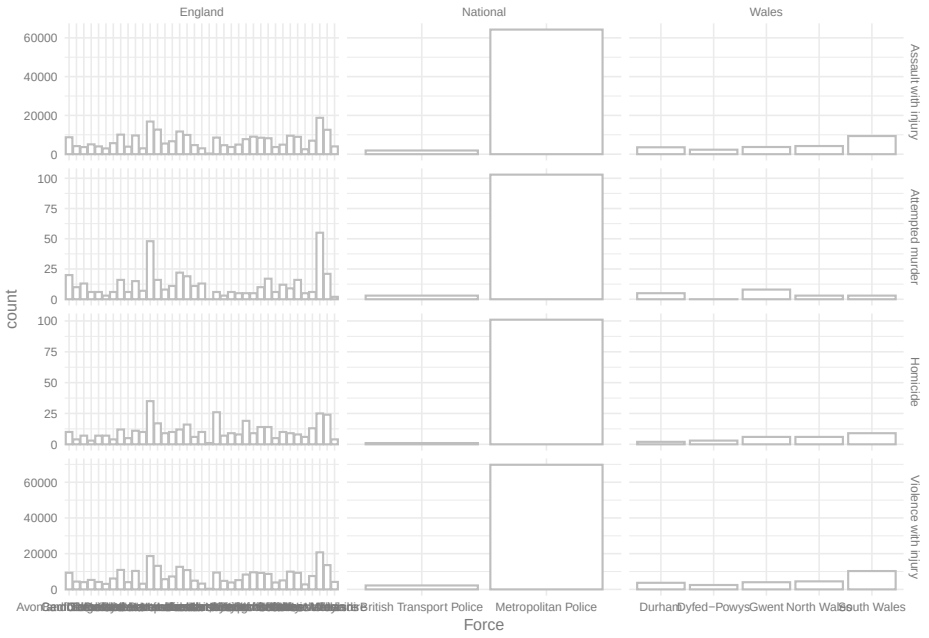
```
## Rows: 176
## Columns: 5
## $ Force           <chr> "Avon and Somerset", ...
## $ Country         <chr> "England", "England",...
## $ offence         <chr> "Homicide", "Attempte...
## $ count           <dbl> 10, 20, 9293, 8716, 4...
## $ type_of_offence <chr> "(attempted) homicide...
```

Um ein bisschen mehr Spielraum zu haben, gibt es in `ggplot` `facets`.

5.0.16 facets!

Facets lassen uns einfach mehrere Subplots definieren, um zusätzlich zu den in unseren `geoms` und `aes`-Aufrufen definierten Aspekten Subgruppen darzustellen.

```
crime_long %>%
  ggplot(aes(y = count,
             x = Force)) +
  geom_col(fill = 'white',
           color = 'grey') +
  facet_grid(offence~Country, ## als facets mit Straftat in Zeilen und Landesteil in Sp
             scales = 'free') ## mit individuellen Skalen
```



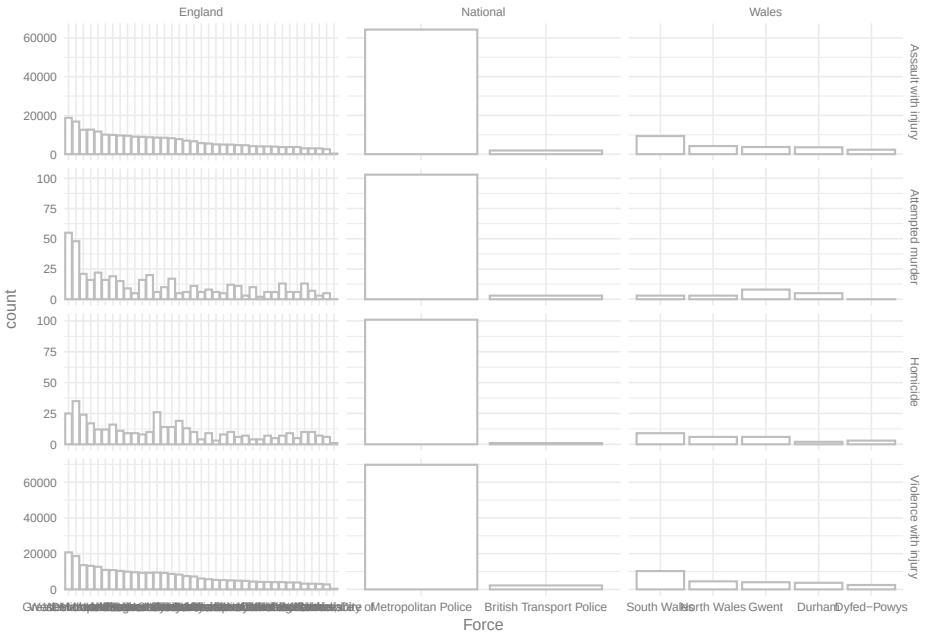
Wie können wir das noch verbessern?

5.0.17 additional tweeks

Zuerst sortieren wir die Polizeistationen in absteigender Reihenfolge der jeweiligen mittleren Fallzahlen.

Dafür benutzen wir das `forcats`-Paket aus dem `tidyverse`. Ein Paket, dass Funktionen zum Verändern und Sortieren von Faktoren bietet.

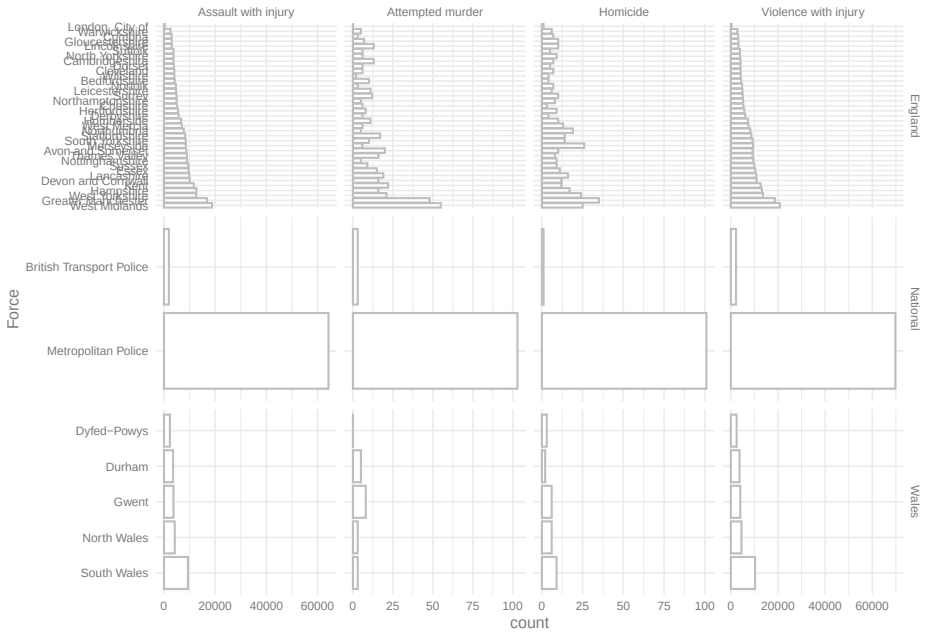
```
crime_long %>%
  mutate(
      Force = as_factor(Force), ## zuerst in Faktor umwandeln
      Force = fct_reorder(Force, count,.fun = mean, .desc = T)) %>% ## dann absteigend sortieren
ggplot(aes(y = count,
             x = Force)) +
  geom_col(fill = 'white',
           color = 'grey') +
  facet_grid(offence~Country,
             scales = 'free')
```



5.0.18 additional tweeks

Um die einzelnen Stationen besser lesbar zu machen, können wir noch die Achsen austauschen.

```
crime_long %>%
  mutate(
      Force = as_factor(Force), ## zuerst in Faktor umwandeln
      Force = fct_reorder(Force, count,.fun = mean, .desc = T)) %>% ## dann absteigend
ggplot(aes(y = count,
             x = Force)) +
  geom_col(fill = 'white',
           color = 'grey') +
  coord_flip() +
  facet_grid(Country ~ offence,
             scales = 'free')
```



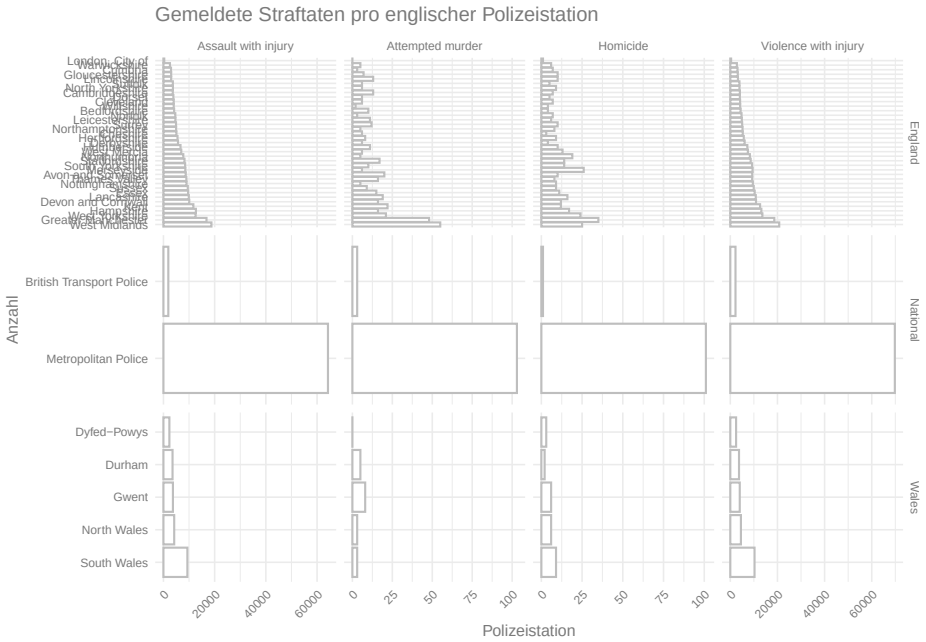
5.0.19 themes

Und zuletzt die Achsenbeschriftungen anpassen und die x-Achsen-Beschriftung rotieren.

- Die Achsenbeschriftungen und Überschriften lassen sich mit der `labs`-Funktion festlegen
- Die Ausrichtung der Beschriftung (wie die meisten grafischen Aspekte) lassen sich mit der `theme`-Funktion fine-tunen

```
crime_long %>%
  mutate(
      Force = as_factor(Force), ## zuerst in Faktor umwandeln
      Force = fct_reorder(Force, count,.fun = mean, .desc = T)) %>% ## dann absteigend sortieren
ggplot(aes(y = count,
             x = Force)) +
  geom_col(fill = 'white',
           color = 'grey') +
  coord_flip() +
  facet_grid(Country ~ offence,
             scales = 'free') +
  labs(x = 'Anzahl',
       y = 'Polizeistation',
       title = 'Gemeldete Straftaten pro englischer Polizeistation') +
  theme(axis.text.x = element_text(angle = 45,
```

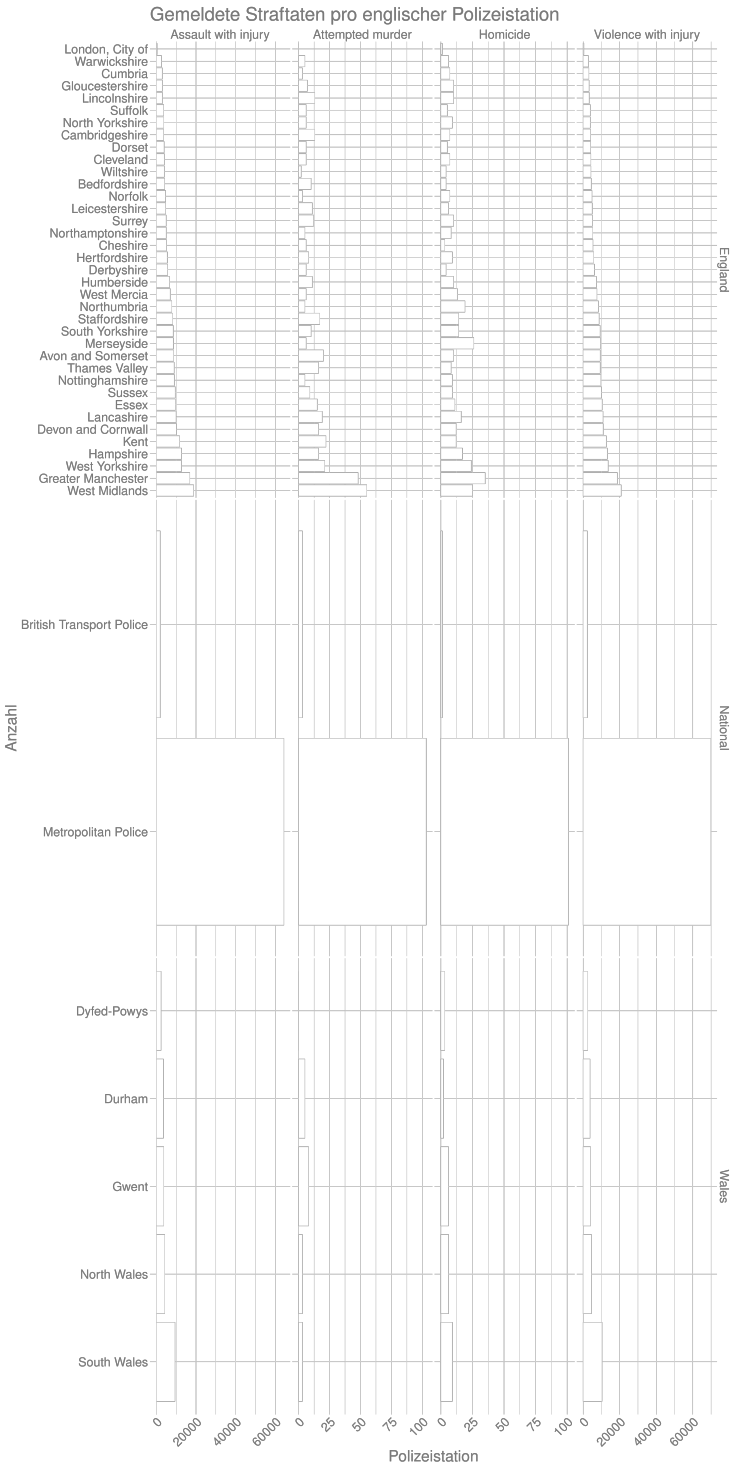
```
                                    hjust = 1))
```



5.0.20 Grafiken exportieren

Um die Auflösung zu verbessern können wir jetzt noch die Grafik mit anderen Seitenverhältnissen exportieren. Die `ggsave`-Funktion lässt uns einfach Grafiken in beliebigen Formaten exportieren.

```
ggsave(filename = 'imgs/police_stations.png',
       width = 50,
       height = 100,
       units = 'cm')
```


Gemeldete Straftaten pro englischer Polizeistation
Assault with injury
Attempted murder
Homicide
Violence with injury
London, City of
Warwickshire
Cumbria
Gloucestershire
Lincolnshire
Suffolk
North Yorkshire
Cambridgeshire
Dorset
Cleveland
Wiltshire
Bedfordshire
Norfolk
Leicestershire
Surrey
Northamptonshire
Cheshire
Hertfordshire
Derbyshire
Humberside
West Mercia
Northumbria
Staffordshire
South Yorkshire
Merseyside
Avon and Somerset
Thames Valley
Nottinghamshire
Sussex
Essex
Lancashire
Devon and Cornwall
Kent
Hampshire
West Yorkshire
Greater Manchester
West Midlands
British Transport Police
Metropolitan Police
Dyfed-Powys
Durham
Gwent
North Wales
South Wales
England
National
Wales
Anzahl
0
20000
40000
60000
0
25
50
75
100
0
25
50
75
100
0
20000
40000
60000
Polizeistation

Chapter 6

Appendix

Bibliography

Anderson, E. (1935). The irises of the Gaspe Peninsula. *Bull. Am. Iris Soc.*, 59:2–5.

Tukey, J. W. (1977). *Exploratory Data Analysis*, volume 2. Reading, Mass.