EDV2

Max Brede

2021-05-26

Contents

Chapter 1

Vorwort

Dieses mit `bookdown` erstellte Dokument ist das über das Sommersemester 2021 hinweg wachsende Skript zur Übung "PSY_B_12-2: Computerunterstützte Datenanalyse II" der CAU zu Kiel.

Chapter 2

Lehrplan

2.1 Semesterplan

2.2 Übungsformat

Die Übung soll zur Hälfte in 45-minütigen Sitzungen im Vorlesungsformat zur Vorstellung der Funktionen und zur anderen Hälfte als 45-minütige praktische Übung stattfinden. Es wird pro Übungs-Sitzung ein Übungszettel ausgegeben, der mit Hilfe der in der Zugehörigen Vorlesung besprochenen Funktionen bearbeitet werden können soll. Diese Zettel sollen nach der jeweiligen Vorlesung für die Übungen vorbereitet werden, in denen der Zettel dann besprochen und mögliche Fragen geklärt werden. Nach den Übungssitzungen haben die Studierenden dann eine Woche Zeit, zusätzliche Hausaufgaben zu bearbeiten.

Eine Ausnahme von diesem Ablauf ist die erste Sitzung, in der organisatorisches und Grundlagen in 90 minütigem Vorlesungsstil besprochen werden sollen. Auch nach dieser Sitzung werden aber Übungszettel und Hausaufgaben ausgegeben.

2.3 Lehrziele für jede Sitzung

Die Studierenden können nach dem Absolvieren der Übung…

Einheit 1

- Gründe für deskriptive Statistik nennen.
- für verschiedene Ausgangssituationen entscheiden, welche Darstellungsform angemessen ist.
- Verfahren zum Umgang mit fehlenden Werten nennen und anwenden.
- Verfahren zum Umgang mit Ausreißern nennen und anwenden.

Einheit 2

- R `formulas` lesen und verwenden.
- Korrelationsanalysen in R durchführen und die Ergebnisse interpretieren.
- einfache lineare Regressionen in R durchführen und die Ergebnisse interpretieren.

Einheit 3

- multiple lineare Regressionen in R durchführen und die Ergebnisse interpretieren.

Einheit 4

- t-Tests in R durchführen und die Ergebnisse interpretieren.
- einfaktorielle Varianzanalysen in R durchführen und die Ergebnisse interpretieren.

Einheit 5

- zweifaktorielle Varianzanalysen in R durchführen und die Ergebnisse interpretieren.

Einheit 6

- beliebige Linearkontraste in R durchführen und die Ergebnisse interpretieren.
- paarweise post-hoc t-Tests in R durchführen und die Ergebnisse interpretieren.

2.4 Prüfungsleistung

Die Studierenden **müssen** während des Semesters die nach den Übungssitzungen ausgegebenen Hausaufgaben innerhalb einer Woche sinnvoll bearbeitet abgeben.

Mit maximal einer nicht sinnvoll bearbeiteten Serie werden die Studierenden zur Gruppenarbeit am Ende des Semesters zugelassen.

Chapter 3

Deskriptive Statistik und Data Cleaning

3.1 Organisatorisches

3.1.1 Semesterplan

Einheit	Vorlesung	Übungswoche	Thema
1	23.04.21	keine Übung	Deskriptive Statistik Data Cleaning
2	07.05.21	KW 19	Hilfsmittel für die Inferenzstatistik Lineare Regression I
3	21.05.21	KW 21	Lineare Regression II
4	04.06.21	KW 23	t- Tests einfaktorielle Varianzanalyse
5	18.06.21	KW 25	zweifaktorielle Varianzanalyse
6	02.07.21	KW 27	Kontrasttests

3.1.2 Übungsablauf

Es wird alle zwei Wochen eine 45-minütige Vorlesung geben und dazu alternierend online-Übungsstunden in mit je einem Kurs. (Eine Ausnahme ist die erste Woche, in der wir eine Vorlesung haben.)

3.1.3 Übungsablauf

	051244	Mo	15:00 - 16:00	Online-Veranstaltung	Maike Splittgerber
		Kurs Gruppe 6			
	051232	Mo	16:15 - 17:00	Online-Veranstaltung	Maike Splittgerber
		Kurs Gruppe 7			
	051247	Mo	17:00 - 17:45	Online-Veranstaltung	Maike Splittgerber
		Kurs Gruppe 8			
	051253	Mi	8:15 - 9:00	Online-Veranstaltung	Johannes Andres
		Kurs Gruppe 4			
	051259	Mi	9:00 - 9:45	Online-Veranstaltung	Johannes Andres
		Kurs Gruppe 3			
	051241	Do	17:00 - 17:45	Online-Veranstaltung	Johannes Andres
		Kurs Gruppe 5			
	051255	Fr	10:15 - 11:00	Online-Veranstaltung	Johannes Andres
		Kurs Gruppe 2			
	051246	Fr	11:00 - 11:45	Online-Veranstaltung	Max Brede
		Kurs Gruppe 1			

3.1.4 Prüfungsleistung

- Gruppenarbeit(4-5 Personen) über 3 Wochen
- Termin für die Klausur entweder vor oder mit Anfang in der Klausurphase.

Genauere Informationen gibt es über das Olat.

Als Zulassung für die Gruppenarbeit ist wieder die sinnvolle Bearbeitung von allen bis auf eine Hausaufgabenserien nötig.

3.1.5 Tutorien

Ronja und Katharina geben (online-)Tutorien dieses Semester.

Katharinas Tutorium wird immer dienstags, 14-16 Uhr stattfinden.

Ronjas Tutorium wird immer dienstags, 16-18 Uhr stattfinden.

3.2 Deskriptive Statistik

3.2.1 Wozu brauche ich das?

Im Gegensatz zur Inferenzstatistik ist das erklärte Ziel der deskriptiven Statistik, (wie der Name schon sagt) beschreibende Aussagen über die vorliegende

Stichprobe zu treffen.

Wir wollen uns also möglichst genau angucken, wie unsere Stichprobe aussieht.

Wozu könnte das gut sein?

3.2.2 Gründe für deskriptive Statistik

- Indikatoren zur externen Validität(Verteilung von Organismusvariablen, Demografie,...)
- Aussagen über Verteilungseigenschaften
  - Schnell zu erfassende Präsentationen von zentraler Tendenz und Streuungen
  - Hinweise auf ungewöhnliche Werte (Ausreißer, fehlende Werte,...)
- Übersicht über Effekte und Zusammenhänge, inklusive solcher, die möglicherweise nicht a-priori erwartet wurden

3.2.3 deskriptive Statistik

Für einen ganz einfachen, schnellen und umfassenden Überblick über die Daten funktioniert die `skim`-Funktion aus dem `skimr`-Paket gut:

```
skimr::skim(df)
```

3.2.4 Demografie

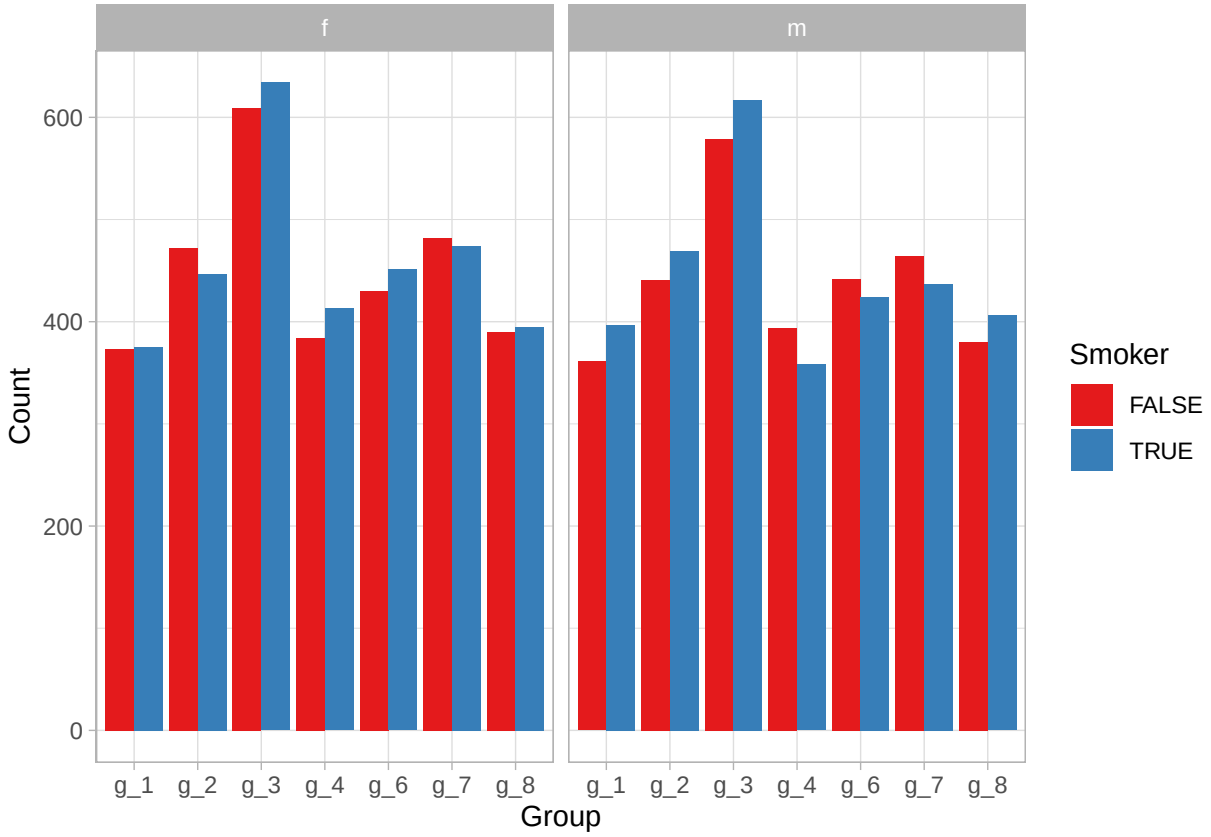
Einfache, schnell zu erfassende Beschreibung der Stichprobe zum Beispiel über eine Tabelle:

```
df %>%
  count(sex, smoker, group) %>%
  pivot_wider(names_from = group,
              values_from = n)
```

```
## # A tibble: 4 x 9
##   sex   smoker   g_1   g_2   g_3   g_4   g_6   g_7
##   <chr> <lgl>  <int> <int> <int> <int> <int> <int>
## 1 f     FALSE    373   472   609   384   430   482
## 2 f     TRUE     375   447   634   413   451   474
## 3 m     FALSE    361   441   579   394   442   464
## 4 m     TRUE     397   469   617   358   424   437
##     g_8
##   <int>
## 1   390
## 2   395
## 3   380
## 4   406
```

Aber vielleicht ein bisschen übersichtlicher in einer Grafik:

```
df %>%
  count(sex, smoker, group) %>%
  ggplot(aes(x = group, fill = smoker, y = n)) +
  geom_col(position = 'dodge') +
  facet_wrap(~sex) +
  labs(x = 'Group',
       y = 'Count',
       fill = 'Smoker')
```



Man sieht auf einen Blick, dass Geschlecht und Raucher ungefähr gleich aufgeteilt wurden, die Gruppen aber wesentlich unterschiedliche Größen aufweisen!

3.2.5 Aussagen über Verteilungseigenschaften

In der `skim`-Ausgabe haben wir ja schon sehen können, dass keine fehlenden Werte vorliegen (`n_missing` war 0).

Wir könnten uns aber noch die Frage stellen, ob Extremwerte in den Gruppen auftauchen, außerdem wollen wir möglichst übersichtlich unsere Verteilungseigenschaften präsentieren.

Das hilft uns zum Einen, um einen besseren Überblick über die Daten zu erhalten, die wir auswerten wollen, zum Anderen hilft es bei einem späteren Bereicht der Leserin, unsere Statistik einzuschätzen. Dazu können wir uns entweder eine Tabelle mit den Verteilungsparametern der numerischen Variablen ausgeben lassen:

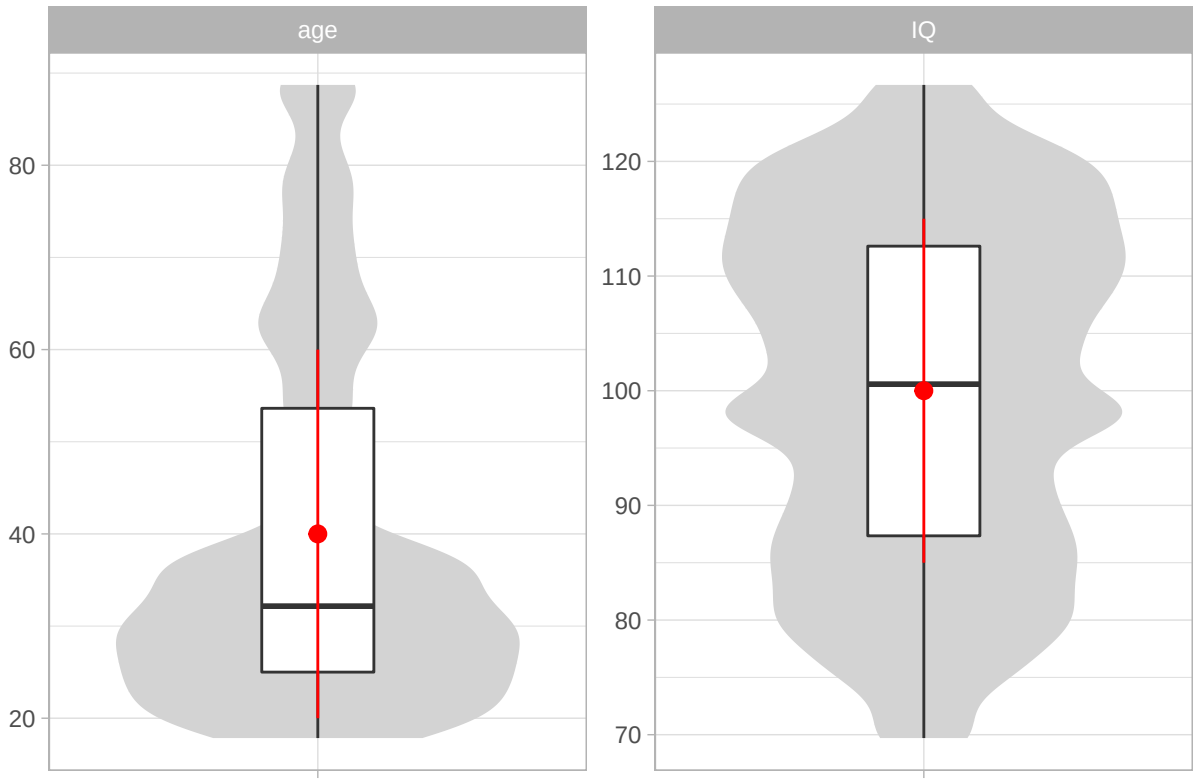
```
df %>%
  pivot_longer(where(is.numeric),
               names_to = 'variable') %>%
  group_by(variable) %>%
  summarise(across(
    value,
    list(
      mean = ~ mean(.),
      sd = ~ sd(.),
      min = ~ min(.),
      q1 = ~ quantile(., .25),
      median = ~ median(.),
      q3 = ~ quantile(., .75),
      max = ~ max(.)
    ),
    .names = '{.fn}'
  )) %>%
  mutate(across(where(is.numeric),
                ~round(.,2)))
```

```
## # A tibble: 2 x 8
##   variable  mean    sd   min    q1 median    q3   max
##   <chr>    <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1 age         40    20  17.8  25     32.2  53.6  88.7
## 2 IQ         100    15  69.7  87.3  101.  113.  127.
```

Oder, wieder ein bisschen übersichtlicher, in einem Diagramm. So könnten wir die ganzen Infos gerade zum Beispiel in einem Boxplot mit eingezeichneten Mittelwerten und Streuungsbalken darstellen:

```
df %>%
  pivot_longer(where(is.numeric),
               names_to = 'variable') %>%
  ggplot(aes(x = '', y = value)) +
  geom_violin(fill = 'lightgrey', color = NA) +
  geom_boxplot(width = .25) +
  stat_summary(fun = mean,
               fun.min = function(x)mean(x) - sd(x),
```

```
              fun.max = function(x)mean(x) + sd(x),
              color = 'red') +
facet_wrap(~variable, scales = 'free') +
labs(x = '',
     y = '')
```



Das Alter ist eindeutig schief verteilt, der IQ dafür mehrgipflig.

Nach der von Tukey aufgestellten Regel (Tukey, 1977) haben wir auch keine Ausreißer (dazu auch später mehr).

3.2.6 Darstellung von Effekten und Zusammenhängen

3.2.6.1 Unterschiede

Da wir eine Reihe von Gruppierungsvariablen haben, könnte der erste Impuls sein, sich die Variablen nach diesen Gruppen aufgeteilt darstellen zu lassen, um mögliche Gruppenunterschiede zu verdeutlichen.

Auch hier können wir uns Tabellen erstellen:

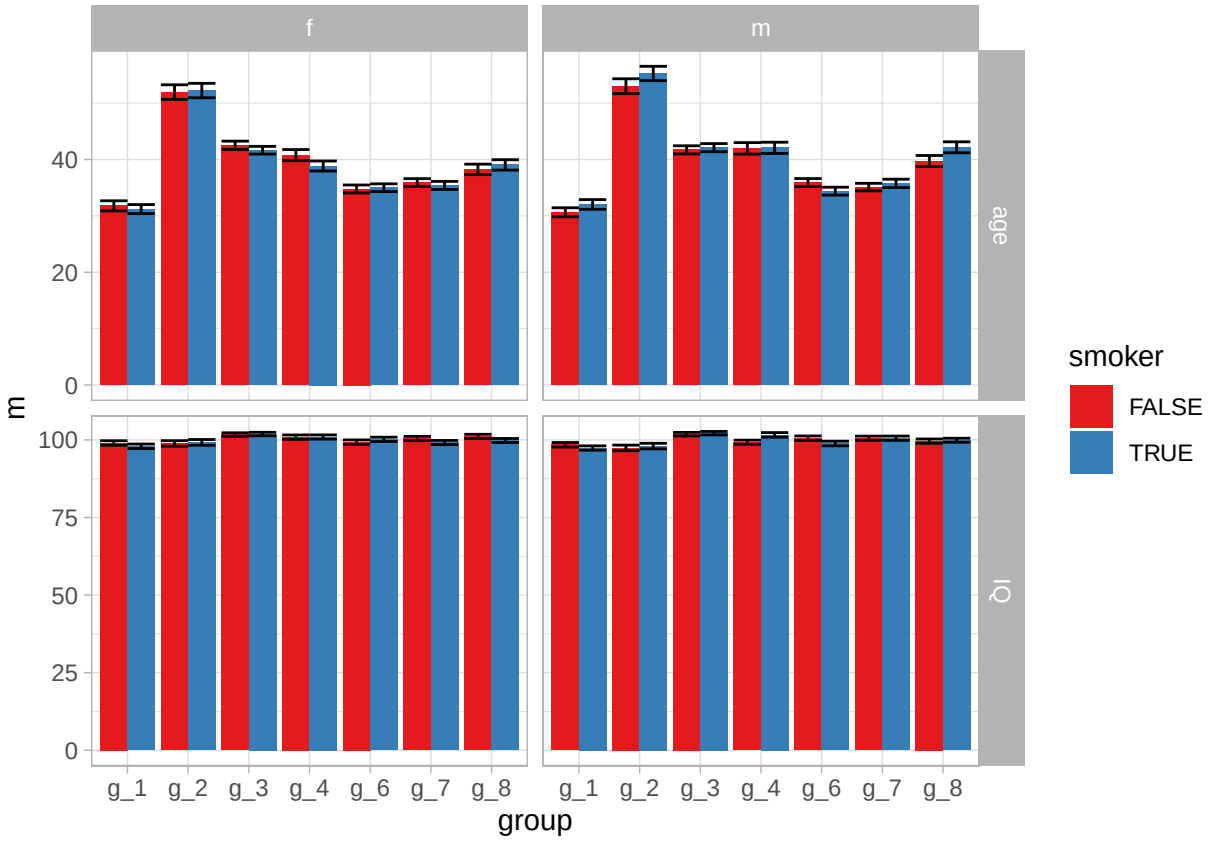
```
df %>%
  group_by(smoker, group, sex) %>%
  summarise(across(where(is.numeric),
                   .fns = list(mean = ~mean(.),
                               sd = ~sd(.),
                               n = ~n()))) %>%
  mutate(across(where(is.numeric),
                ~round(.,2)))
```

```
## # A tibble: 28 x 9
## # Groups:   smoker, group [14]
##    smoker group sex   IQ_mean IQ_sd  IQ_n age_mean
##    <lgl>  <chr> <chr>   <dbl> <dbl> <dbl>    <dbl>
##  1 FALSE  g_1   f        99.0  13.9   373     31.8
##  2 FALSE  g_1   m        98.4  13.9   361     30.6
##  3 FALSE  g_2   f        98.9  19.3   472     52.0
##  4 FALSE  g_2   m        97.4  19.0   441     53.0
##  5 FALSE  g_3   f       102.   14.0   609     42.5
##  6 FALSE  g_3   m       102.   13.7   579     41.7
##  7 FALSE  g_4   f       101.   13.8   384     40.8
##  8 FALSE  g_4   m        99.3  13.1   394     42.0
##  9 FALSE  g_6   f        99.3  14.6   430     34.8
## 10 FALSE  g_6   m       101.   15.4   442     35.9
##    age_sd age_n
##     <dbl> <dbl>
##  1   17.3   373
##  2   15.3   361
##  3   27.9   472
##  4   27.7   441
##  5   18.2   609
##  6   17.4   579
##  7   19.3   384
##  8   20.3   394
##  9   14.6   430
## 10   14.9   442
## # ... with 18 more rows
```

Oder Plots erstellen um die möglichen Unterschiede darzustellen:

```
df %>%
  pivot_longer(cols = where(is.numeric),
               names_to = 'variable') %>%
  group_by(variable, smoker, sex, group) %>%
  summarise(m = mean(value),
            sem = sqrt(var(value)/n()),
            upper = m + sem,
```

```
            lower = m - sem) %>%
ggplot(aes(x = group,
           y = m,
           fill = smoker)) +
geom_col(position = 'dodge') +
geom_errorbar(aes(ymin = lower,
                  ymax = upper),
              position = 'dodge')+
facet_grid( variable ~ sex , scales = 'free')
```



Das wird zwar ein bisschen unübersichtlich (wenn man das wirklich sinnvoll betreiben wollen würde sollte man sich Gedanken dazu machen, welche Variablen tatsächlich von Relevanz sind), man könnte aber zu dem Schluss kommen dass die IQs relativ ähnlich sind, die Altersgruppen aber nicht.

3.2.6.2 Zusammenhänge

Zuletzt wollen wir noch gucken, ob in den Daten irgendwelche (linearen) Zusammenhänge direkt ersichtlich sind. Dazu können wir zuerst Korrelationen berechnen, zum Beispiel einmal für die gesamte Stichprobe und einmal für die Untergruppen:

```
library(magrittr)

df %$%
  cor(age,
      IQ)
```

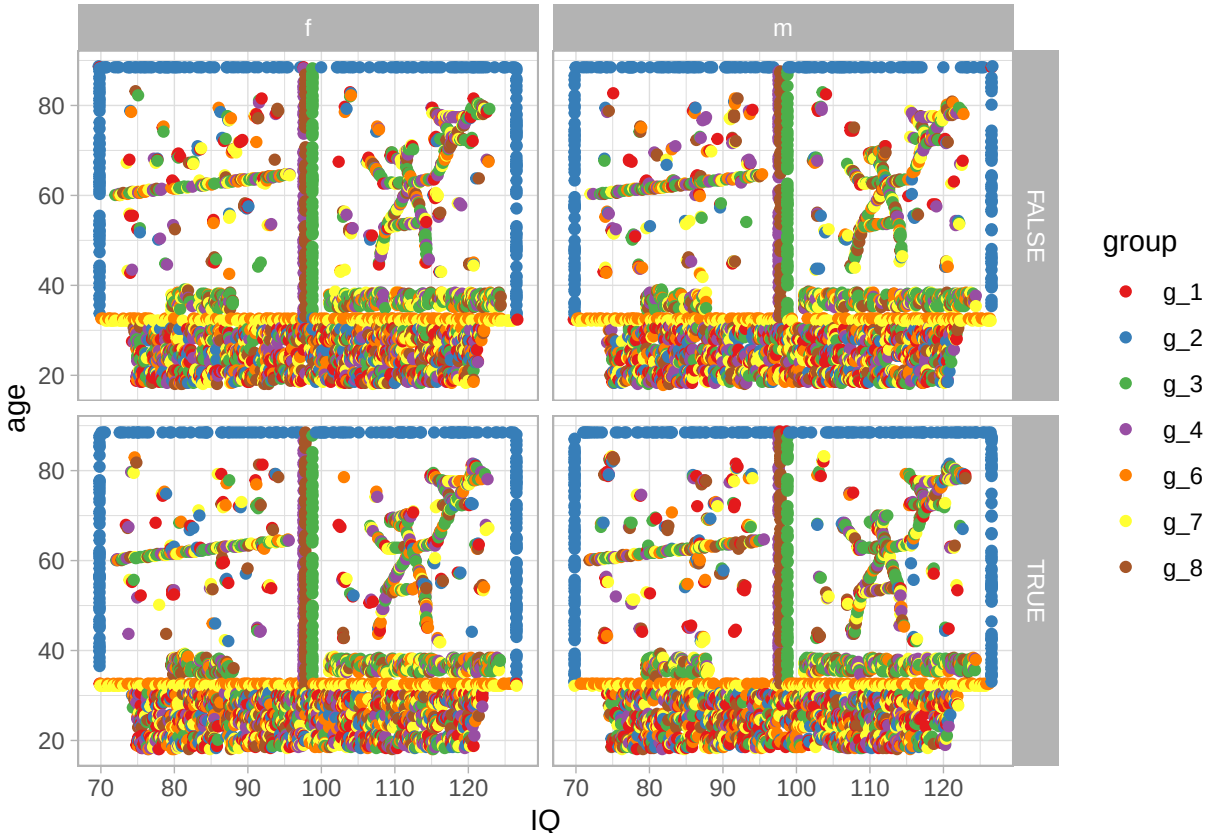
```
## [1] 0.06659135
```

```
df %>%
  group_by(group) %>%
  summarise(r = cor(age, IQ))
```

```
## # A tibble: 7 x 2
##   group       r
##   <chr>   <dbl>
## 1 g_1   0.0540
## 2 g_2   0.00747
## 3 g_3   0.110
## 4 g_4   0.0952
## 5 g_6   0.111
## 6 g_7   0.139
## 7 g_8   0.0636
```

Hier ist so weit nichts auffällig. Ein letzter zu überprüfender Aspekt sind die nicht-linearen Zusammenhänge, zum Beispiel über angemessene Plots. Dies können wir zum Einen für die Untergruppen überprüfen wollen:

```
df %>%
  ggplot(aes(x = IQ,
             y = age,
             color = group)) +
  geom_point() +
  facet_grid(smoker ~ sex)
```



Zum Anderen für die gesamte Stichprobe:

```
df %>%
  ggplot(aes(IQ, age, color = group)) +
  geom_point(size = .001) +
  scale_color_grey()
```

3.3 Data Cleaning

3.3.1 Umgang mit fehlenden Werten

`NA`s sind das in R zur Codierung von fehlenden Werten genutzte Datenformat.

Sie können in Vektoren (und damit auch `tibble`-Spalten) jeden Datenformats auftreten:

```
c(T,NA,F)
```

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

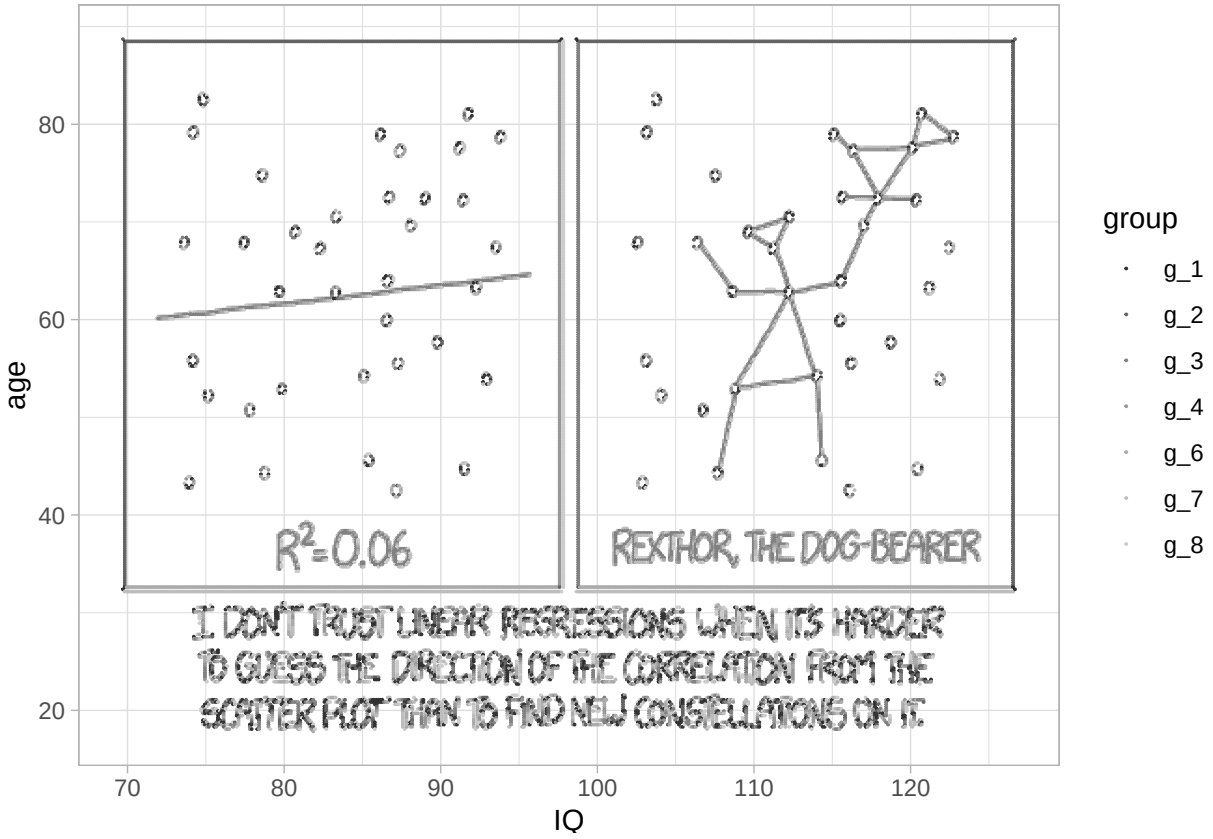


Figure 3.1: Original-Comic von [xkcd](https://xkcd.com/1725/)

```
c(1,NA,2)
```

```
## [1]  1 NA  2
```

```
c('a',NA,'b')
```

```
## [1] "a" NA  "b"
```

Wenn wir versuche mit einem Vektor zu rechnen, der `NA`s beinhaltet, können wir auf Probleme stoßen:

```
aVector <- c(NA,1,2,5,NA,6,8)
length(aVector[aVector > 3])
```

```
## [1] 5
```

```
mean(aVector)
```

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

Mit der `is.na()`-Funktion können wir uns einen logischen Vektor ausgeben lassen, der fehlende Werte codiert. Den können wir dann wie gewohnt benutzen:

```
sum(is.na(aVector))
```

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

```
any(is.na(aVector))
```

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

3.3.2 fehlende Werte und einfache Kennwerte

Die meisten Funktionen im `base R` Umfang haben ein `na.rm`-Argument mit dem wir fehlende Werte von Berechnungen ausschließen können. Das kann an vielen Stellen schon eine sinnvolle Lösung sein, zum Beispiel wenn wir die Infos über die Anzahl fehlender Werte nicht verlieren wollen.

Das könnte dann zum Beispiel so aussehen:

```
mean(aVector)
```

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

```
mean(aVector, na.rm = T)
```

```
## [1] 4.4
```

Dieses Argument können wir auch in die gewohnten Pipelines einsetzen.

Als Beispiel nehmen wir diesen kleinen (unrealistisch unvollständigen) Datensatz `df_2`:

```
(df_2 <- read_csv('data/small_nas.csv'))
```

```
## # A tibble: 12 x 4
##       VP group    t_1    t_2
##    <dbl> <dbl>  <dbl>  <dbl>
##  1     1     1  3.66  -1.53
##  2     2     2 NA      3.32
##  3     3     3  2.23  NA
##  4     4     4  4.82   4.14
##  5     5     1  0.142  0.537
##  6     6     2  1.86   5.04
##  7     7     3 NA      1.46
##  8     8     4  3.60   3.50
##  9     9     1  0.353 NA
## 10    10     2 NA     NA
## 11    11     3  3.79   2.57
## 12    12     4  5.37   4.95
```

Wir könnten jetzt die Pipeline für die Gruppenunterschiede von eben nochmal benutzen, aber um eine Angabe zur Anzahl der fehlenden Werte ergänzen:

```
df_2 %>%
  group_by(group) %>%
  summarise(across(matches('t_'),
                   .fns = list(mean = ~mean(., na.rm = T),
                               sd = ~sd(., na.rm = T),
                               n = ~n(),
                               missing = ~sum(is.na(.))))) %>%
  mutate(across(where(is.numeric),
                ~round(.,2)))
```

```
## # A tibble: 4 x 9
##   group t_1_mean t_1_sd t_1_n t_1_missing t_2_mean
##   <dbl>    <dbl>  <dbl> <dbl>       <dbl>    <dbl>
## 1     1     1.39   1.97     3           0    -0.5
## 2     2     1.86  NA        3           2     4.18
## 3     3     3.01   1.1      3           1     2.01
## 4     4     4.6    0.91     3           0     4.2
##   t_2_sd t_2_n t_2_missing
##    <dbl> <dbl>       <dbl>
## 1   1.46     3           1
## 2   1.22     3           1
## 3   0.79     3           1
## 4   0.73     3           0
```

3.3.3 `NA`-Bereinigung von Datensätzen

Vor statistischen Auswertungen kann es aber einfacher sein, den Datensatz komplett von fehlenden Werten zu bereinigen.

Je nach dem Fall und der Person die man fragt, gibt es verschiedene Vorgehensweisen. Wir gucken uns hier genauer den fallweisen Ausschluss und das ersetzen durch Werte der zentralen Tendenz an.

Die Entscheidung für das Auffüllen oder das Ausschließen muss von Fall zu Fall gefällt werden!

Wenn wir zum Beispiel unseren `df_2` nochmal angucken, fehlt ein Viertel der Werte. Hier die Fälle aufzufüllen und so zu tun als würde man mit 133% der Werte arbeiten, die tatsächlich vorlagen, ist offensichtlich schwierig. Gleichzeitig wird oft das Argument vorgebracht, dass insbesondere diejenigen Versuchspersonen, die in bestimmten Bedingungen keine Antwort produziert haben ein wichtiger Teil der Stichprobe sind und das Auslassen an sich als Form der Antwort betrachtet werden kann. Wenn wir diese Versuchspersonen ausschließen, verzerren wir nach diesem Argument also systematisch unsere Stichprobe.

Wichtig ist also vor jeder Bereinigung, Überlegungen darüber anzustellen, was im gegebenen Fall gerade die angemessenste Lösung darstellt. Bei unserem Datensatz df_2 sind beide Methoden nicht wirklich gut, der Datensatz ist aber auch extrem.

3.3.3.1 Fallweiser Ausschluss

Die radikalste Methode ist der Fallweise Ausschluss, also der Ausschluss aller Eintragungen einer Versuchsperson, die mindestens einen fehlenden Wert vorliegen hat.

Als Erinnerung, hier nochmal unser Datensatz `df_2`:

VP	group	t_1	t_2
1	1	3.6612191	-1.5337696
2	2	NA	3.3223479
3	3	2.2250924	NA
4	4	4.8187796	4.1387727
5	1	0.1419190	0.5373972
6	2	1.8635821	5.0435449
7	3	NA	1.4552116
8	4	3.6001703	3.5013055
9	1	0.3530444	NA
10	2	NA	NA
11	3	3.7877141	2.5676322
12	4	5.3706695	4.9490249

Wir müssten also die Versuchspersonen ausschließen.

Die einfachste Variante dafür ist, den Datensatz ins `wide`-Format zu überführen (wie er es in unserem Fall schon vorliegt) und mit `drop_na` diejenigen Zeilen auszuschließen, die fehlende Werte beinhalten:

```
df_2 %>%
  drop_na()
```

```
## # A tibble: 7 x 4
##      VP group   t_1    t_2
##   <dbl> <dbl> <dbl>  <dbl>
## 1     1     1 3.66  -1.53
## 2     4     4 4.82   4.14
## 3     5     1 0.142  0.537
## 4     6     2 1.86   5.04
## 5     8     4 3.60   3.50
## 6    11     3 3.79   2.57
## 7    12     4 5.37   4.95
```

3.3.3.2 Ersetzen fehlender Werte

Statt radikal alle Fälle auszuschließen, die mindestens einen fehlenden Wert beinhalten, gibt es auch Ansätze, diese aufzufüllen. Gängige Verfahren hier sind die fehlenden Werte hypothesenunabhängig (also nicht gruppenweise) mit dem (getrimmten) Mittelwert, dem Median oder dem Modus der Gesamtstichprobe aufzufüllen. Die Umsetzung in R sieht dann immer gleich aus, die einzige Änderung findet im Kennwert statt, den man zur Ergänzung wählt.

Hier mal ein Beispiel mit dem getrimmten Mittelwert:

```
df_2 %>%
  mutate(across(where(is.numeric),
                ~replace_na(.,
                            mean(.,
                                 na.rm=T,
                                 trim = .05))))
```

```
## # A tibble: 12 x 4
##       VP group   t_1    t_2
##    <dbl> <dbl> <dbl>  <dbl>
##  1     1     1 3.66  -1.53
##  2     2     2 2.87   3.32
##  3     3     3 2.23   2.66
##  4     4     4 4.82   4.14
##  5     5     1 0.142  0.537
##  6     6     2 1.86   5.04
##  7     7     3 2.87   1.46
##  8     8     4 3.60   3.50
##  9     9     1 0.353  2.66
## 10    10     2 2.87   2.66
## 11    11     3 3.79   2.57
## 12    12     4 5.37   4.95
```

3.3.4 Umgang mit Ausreißern

Ausreißerbereinigung sind ein komplexes Thema, über das viel diskutiert werden kann und auch muss.

Da wir uns hier aber im Rahmen einer praktischen Übung befinden sparen wir uns das und nutzen die weit verbreitete Regel, die Tukey (1977) formuliert hat:

Diejenigen Werte sind als Ausreißer zu betrachten, die außerhalb des Intervals

$$Q_1 - 1.5 \cdot \text{IQR} \leq x \leq Q_3 + 1.5 \cdot \text{IQR}$$

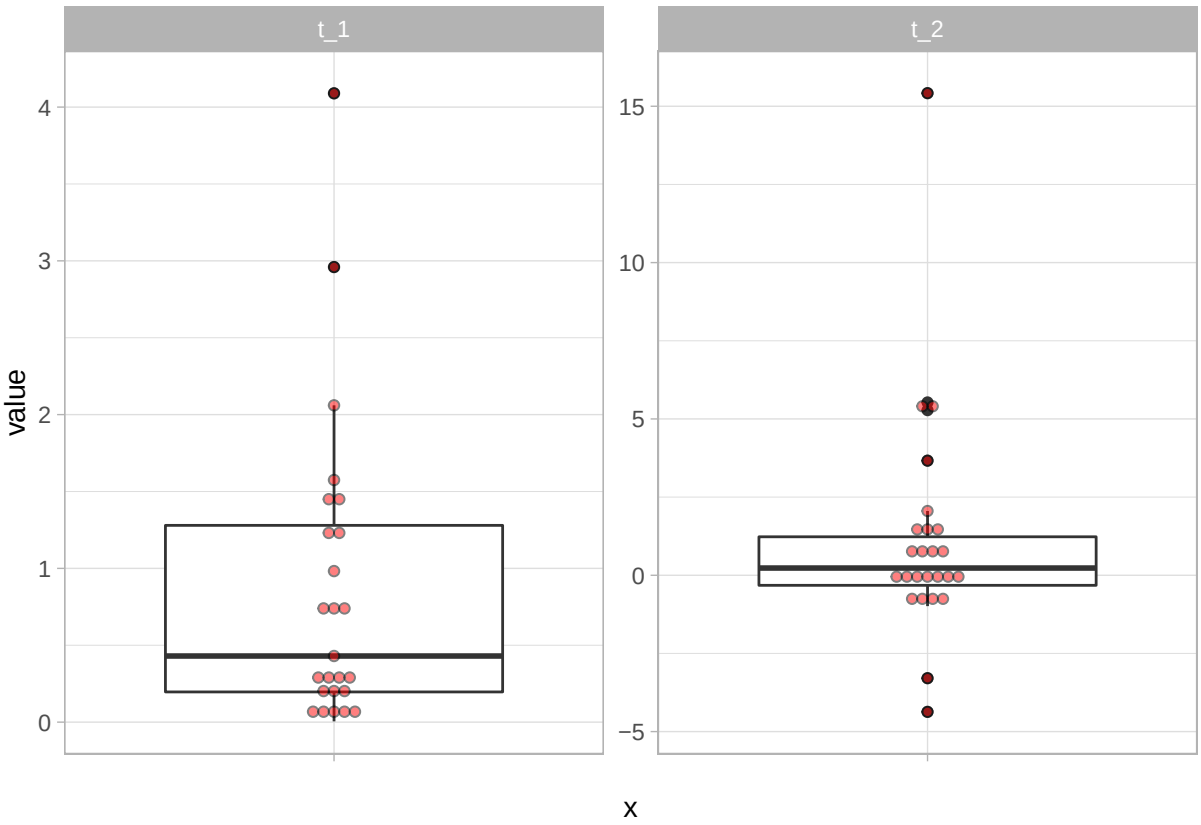
liegen. Diese Regel ist auch der in `geom_boxplot` implementierte Standard, den wir ja auch schon zumindest vom Sehen kennen.

Die Frage ist nun, was wir mit eventuell detektierten Ausreißern machen.

Dafür betrachten wir den folgenden Datensatz `df_3`:

```
df_3 %>%
  pivot_longer(cols = everything()) %>%
  ggplot(aes(y = value, x = '')) +
  geom_boxplot() +
  geom_dotplot(binaxis = 'y',
               stackdir = 'center',
               alpha = .5,
```

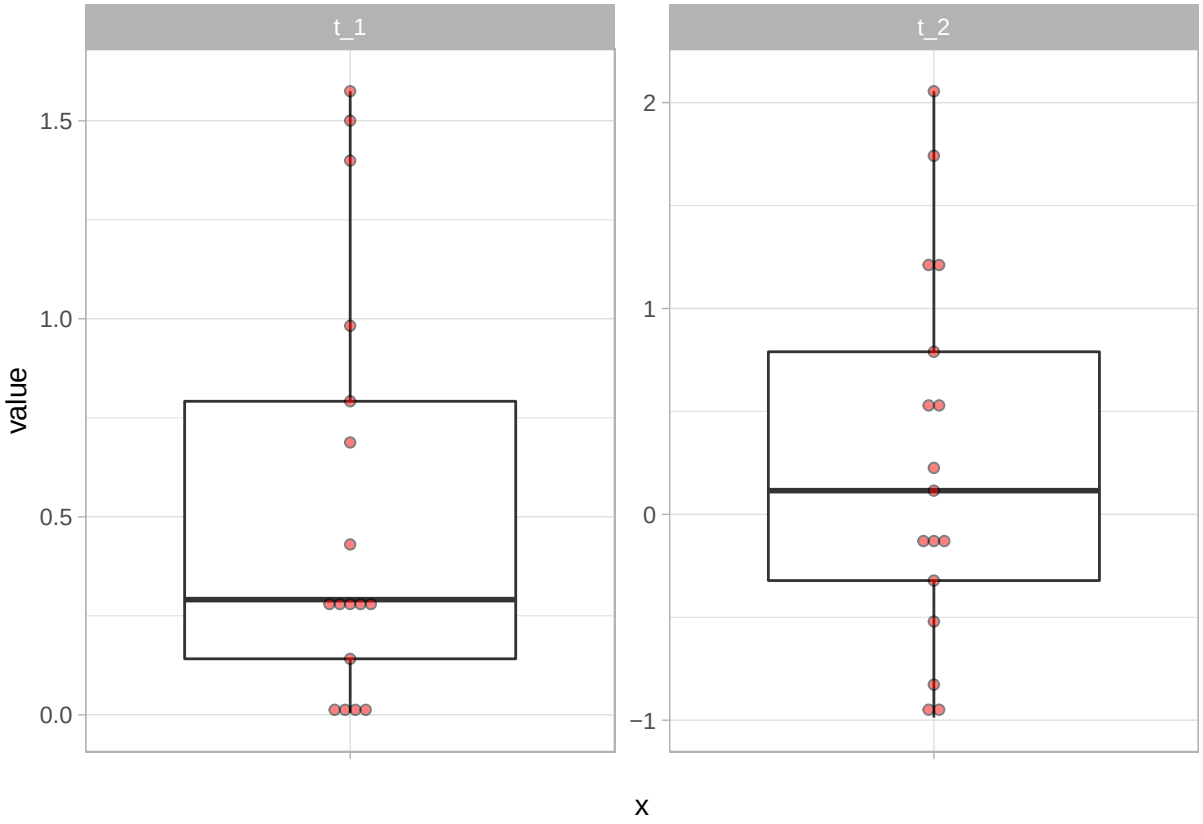
```
               fill = 'red',
               dotsize = .5) +
  facet_wrap(~name,scales = 'free_y')
```



Auch hier können wir wieder zum radikalen Ausschluss greifen und den Datensatz einfach danach filtern, dass unsere Variablen zwischen den "Tukey Fences" liegen:

```
df_3 %>%
  filter(between(t_1,
                 quantile(t_1,.25) - 1.5 * IQR(t_1),
                 quantile(t_1,.75) + 1.5 * IQR(t_1)) &
           between(t_2,
                 quantile(t_2,.25) - 1.5 * IQR(t_2),
                 quantile(t_2,.75) + 1.5 * IQR(t_2))) %>%
  pivot_longer(cols = everything()) %>%
  ggplot(aes(y = value, x = '')) +
  geom_boxplot() +
  geom_dotplot(binaxis = 'y',
```

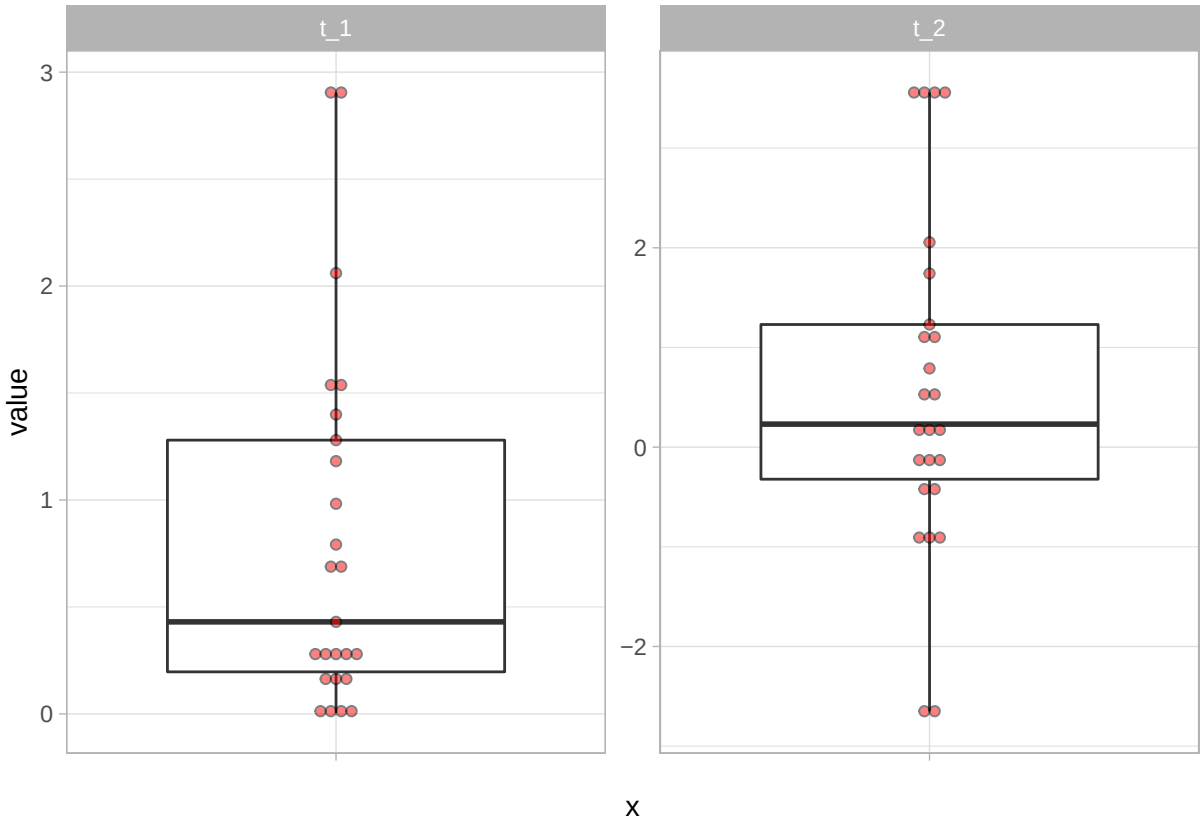
```
                stackdir = 'center',
                alpha = .5,
                fill = 'red',
                dotsize = .5) +
  facet_wrap(~name,scales = 'free_y')
```



Oder wir 'windsorieren' unsere Ausreißer, indem wir sie durch den Wert der jeweiligen Grenze ersetzen. Wir sagen also, dass diejenigen Werte, die außerhalb der Fences liegen auf den Wert der jeweiligen Grenze gesetzt werden sollen:

```
df_3 %>%
  mutate(across(everything(),
                ~ifelse(. >  quantile(.,.75) + 1.5 * IQR(.),
                        quantile(.,.75) + 1.5 * IQR(.),
                        .)),
         across(everything(),
                ~ifelse(. <  quantile(.,.25) - 1.5 * IQR(.),
                        quantile(.,.25) - 1.5 * IQR(.),
                        .))) %>%
```

```
pivot_longer(cols = everything()) %>%
  ggplot(aes(y = value, x = '')) +
  geom_boxplot() +
  geom_dotplot(binaxis = 'y',
               stackdir = 'center',
               alpha = .5,
               fill = 'red',
               dotsize = .5) +
  facet_wrap(~name,scales = 'free_y')
```



3.4 Organisationsformen von Datensätzen

3.4.1 long vs. wide format

3.4.1.0.1 long-format

Name	RT	Bedingung
Snake Müller	2624	1
Snake Müller	3902	2
Snake Müller	6293	3
Vera Baum	1252	1
Vera Baum	2346	2
Vera Baum	4321	3

3.4.1.0.2 wide-format

Name	RT_1	RT_2	RT_3
Snake Müller	2624	3902	6293
Vera Baum	1252	2346	4321

Die `pivot`-Funktionen `pivot_longer` und `pivot_wider` bieten die Möglichkeit, einen Datensatz von einem in das andere Format zu konvertieren.

```
longFormat
```

```
##           Name   RT Bedingung
## 1 Snake Müller 2624         1
## 2 Snake Müller 3902         2
## 3 Snake Müller 6293         3
## 4    Vera Baum 1252         1
## 5    Vera Baum 2346         2
## 6    Vera Baum 4321         3
```

3.4.1.1 long to wide

```
wideFormat <- longFormat %>%
  pivot_wider(names_from = 'Bedingung',
              values_from = 'RT',
              names_prefix = 'RT_')
wideFormat
```

```
## # A tibble: 2 x 4
##   Name          RT_1  RT_2  RT_3
##   <chr>        <dbl> <dbl> <dbl>
## 1 Snake Müller  2624  3902  6293
## 2 Vera Baum     1252  2346  4321
```

3.4.1.2 wide to long

```
longFormat <- wideFormat %>%
  pivot_longer(cols = -1,
               names_prefix = 'RT_',
```

```
               names_to = 'Bedingung',
               values_to = 'RT')
longFormat
```

```
## # A tibble: 6 x 3
##   Name         Bedingung    RT
##   <chr>        <chr>     <dbl>
## 1 Snake Müller 1          2624
## 2 Snake Müller 2          3902
## 3 Snake Müller 3          6293
## 4 Vera Baum    1          1252
## 5 Vera Baum    2          2346
## 6 Vera Baum    3          4321
```

Chapter 4

Hilfsmittel für die Inferenzstatistik

4.1 Organisatorisches

4.1.1 Semesterplan

Einheit	Vorlesung	Übungswoche	Thema
1	23.04.21	keine Übung	Deskriptive Statistik Data Cleaning
2	07.05.21	KW 19	Hilfsmittel für die Inferenzstatistik Lineare Regression I
3	21.05.21	KW 21	Lineare Regression II
4	04.06.21	KW 23	t- Tests einfaktorielle Varianzanalyse
5	18.06.21	KW 25	zweifaktorielle Varianzanalyse
6	02.07.21	KW 27	Kontrasttests

4.2 Hilfsmittel für die Inferenzstatistik

4.2.1 Modellterme

Alle inferenzstatistischen Verfahren im `base-R`-Umfang und viele andere aus Zusatzpaketen nutzen die sogenannte *Formelschreibweise* um Modelle zu definieren. Am Anfang ist die Syntax ein bisschen ungewohnt, am Ende resultiert aus dieser Schreibweise aber eine sehr übersichtliche und schnell erfassbare Modell-Formulierung.

Die Formulierung folgt dabei grundsätzlich dem folgenden System, das sich am Besten analog zu einer mathematischen Funktionsgleichung vorgestellt werden kann. Da das `=` aber schon für Zuweisungen belegt ist, wird es in `formula`-Schreibweise durch eine Tilde (~) ersetzt:

Table 4.1

	modellierte Variable(n)	~	Modellformel
Regression:	Kriterium	~	Prädiktor(en)
Varianzanalyse:	AV	~	UV(s als Faktor(en))

4.2.1.1 Modell-Term

Der Modell-Term auf der rechten Seite der Tilde wird dabei aus einer Reihe von Variablen und Kombinationsoperatoren zusammengesetzt. Zuerst etwas unintuitiv sind diese Operatoren im normalen R-Kontext mit anderen Bedeutungen belegt, in `formula`s funktionieren sie aber so *nicht* Die Operatoren sind die folgenden:

Operator	übliche Bedeutung	Bedeutung in ‘formula‘s
+	Addition	Vorhersageterm hinzufügen
-	Subtraktion	Vorhersageterm ausschließen
<A> : <B>	Sequenz	Interaktion AxB
<A> * <B>	Multiplikation	Effekt von A, B und AxB

Anhand von einer Reihe von Beispielen wird die Formulierung deutlich, dafür führen wir noch kurz eine Hand voll Notationen ein, die meisten davon sind wahrscheinlich nicht überraschend:

Abkürzung	Bedeutung
H_0	Nullhypothese eines statistischen Tests
H_1	Alternativhypothese eines statistischen Tests
UV	unabhängige Variable
AV	abhängige Variable
X_i / Y_i	numerische (Zufalls-) Variable
F_i	kategoriale Varable (Faktor)

4.2.1.2 Regressionsmodelle

`Y ~ X1`: einfache lineare Regression von `Y` auf `X1`

`Y ~ X1 + X2`: multiple lineare Regression von `Y` auf `X1` und `X2`

`Y ~ X1+X2+X1:X2`: multiple lineare Regression von `Y` auf `X1` und `X2` sowie auf den Interaktionsterm von `X1` und `X2`

`Y ~ X1*X2`: multiple lineare Regression von `Y` auf `X1` und `X2` sowie auf den Interaktionsterm von `X1` und `X2`

4.2.1.3 Varianzanalytische Modelle

`Y ~ F1`: einfaktorielle Varianzanalyse

`Y ~ F1 + F2 + F1:F2`: zweifaktorielle Varianzanalyse mit beiden Haupteffekten und der Interaktion

`Y ~ F1 * F2`: auch zweifaktorielle Varianzanalyse mit beiden Haupteffekten und der Interaktion

`Y ~ X1 + F1`: Kovarianzanalyse mit Kovariate X1 und Faktor F1

Innerhalb einer Modellformel können die Terme selbst das Ergebnis der Anwendung von Funktionen auf Variablen sein:

$$\texttt{log}(Y) \sim \texttt{scale}(X)$$

Wenn wir die für die Formulierung genutzten Operatoren für arithmetische Operationen in der Modellformel verwenden wollen, müssen sie mit `I()` eingeschlossen werden um den Kontext klarzumachen:

$$Y \sim \texttt{I}(2 * X)$$

4.2.2 Aufgabe

Welche Hypothese(n) pass(t/en) zu folgender Modellformel:

```
IQ ~ Geschlecht + Raucher
```

- A: Es gibt einen Unterschied zwischen der Intelligenz von Rauchern und Nichtrauchern und zwischen der von Frauen und Männern.
- B: Es gibt einen Unterschied zwischen der Intelligenz von Rauchern und Nichtrauchern und zwischen der von Frauen und Männern sowie einen Unterschied in der Intelligenz zwischen Rauchern und Nichtrauchern, der sich in der Ausprägung zwischen den Geschlechtern unterscheidet.
- C: Es gibt einen Unterschied in der Intelligenz zwischen Rauchern und Nichtrauchern, der sich in der Ausprägung zwischen den Geschlechtern unterscheidet.

- D: Es gibt einen Zusammenhang zwischen Rauchen und Geschlecht auf der einen und Intelligenz auf der anderen Seite.

Lösung

A ist richtig.

Chapter 5

Einfache lineare Zusammenhänge

5.0.1 Datensatz

Für die Tests auf linearen Zusammenhänge werden wir den Datensatz `df_wide` mit den folgenden Variablen benutzen:

Variable	Inhalt
'group'	Treatment-Gruppe
'pre_skill'	motorischer Skill vor dem Treatment
'post_skill'	motorischer Skill nach dem Treatment
'hawie_iq'	Intelligenz-Quotient aus HAWIE
'hawie_wahr_log'	Skalenwert Wahrnehmungsgebundenes logisches Denken aus HAWIE

5.1 Test auf Korrelation

5.1.1 Test-Hintergrund

Die empirische Korrelation zweier gemeinsam normalverteilter Variablen lässt sich daraufhin testen, ob sie mit der H_0 'kein linearer Zusammenhang' (H_0 : $\rho_{X,Y} = 0$) verträglich ist.

Dabei wird genutzt, dass bei Multinormalverteilung und Unkorreliertheit der Variablen X und Y die Teststatistik $t_r = r_{x,y}\sqrt{\frac{n-2}{1-r_{x,y}{}^2}}$ $t-$verteilt ist, mit $n-2$ Freiheitsgraden.

Man testet also die Teststatistik t gegen die t_{n-2} -Verteilung.

Ist der Test signifikant, wird die H_1 angenommen, also dass die 'wahre' Korrelation zwischen X und Y ungleich 0 ist.

Gerichtete Hypothesen lassen sich analog testen.

5.1.2 Test auf Korrelation in R

Man kann den Test in R mit Vektoren als Eingabe...

```
cor.test(df_wide$hawie_iq, df_wide$hawie_wahr_log)
```

```
##
##  Pearson's product-moment correlation
##
## data:  df_wide$hawie_iq and df_wide$hawie_wahr_log
## t = 8.0781, df = 48, p-value = 1.678e-10
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.6094958 0.8564580
## sample estimates:
##       cor
## 0.7590665
```

... und mit Modellformel als Eingabe aufrufen.

```
cor.test(~ hawie_iq + hawie_wahr_log, data = df_wide)
```

```
##
##  Pearson's product-moment correlation
##
## data:  hawie_iq and hawie_wahr_log
## t = 8.0781, df = 48, p-value = 1.678e-10
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.6094958 0.8564580
## sample estimates:
##       cor
## 0.7590665
```

Das `alternative`-Argument bietet die Möglichkeit, die Richtung des Signifikanztests anzugeben.

Dabei steht `'greater'` für einen rechtsseitigen, `'lesser'` für einen linksseitigen und der Standard `'two.sided'` für einen zweiseitigen Test.

```
cor.test(~ hawie_iq + hawie_wahr_log,
         data = df_wide,
         alternative='greater')
```

```
##
```

```
##  Pearson's product-moment correlation
##
## data:  hawie_iq and hawie_wahr_log
## t = 8.0781, df = 48, p-value = 8.392e-11
## alternative hypothesis: true correlation is greater than 0
## 95 percent confidence interval:
##  0.6375781 1.0000000
## sample estimates:
##       cor
## 0.7590665
```

Der Output lässt sich noch ein bisschen schicker mit der `tidy`-Funktion aus dem `broom`-Paket darstellen (ist auch im `tidyverse` enthalten):

```
cor.test(~ hawie_iq + hawie_wahr_log,
         data = df_wide,
         alternative='greater') %>%
  broom::tidy()
```

```
## # A tibble: 1 x 8
##   estimate statistic  p.value parameter conf.low
##      <dbl>     <dbl>    <dbl>     <int>    <dbl>
## 1    0.759      8.08 8.39e-11        48    0.638
##   conf.high method
##       <dbl> <chr>
## 1         1 Pearson's product-moment correlation
##   alternative
##   <chr>
## 1 greater
```

5.1.3 Aufgabe

Wie kann ich das Ergebnis interpretieren?

```
cor.test(~ hawie_iq + hawie_wahr_log,
         data = df_wide,
         alternative='greater') %>%
  broom::tidy()
```

```
## # A tibble: 1 x 8
##   estimate statistic  p.value parameter conf.low
##      <dbl>     <dbl>    <dbl>     <int>    <dbl>
## 1    0.759      8.08 8.39e-11        48    0.638
##   conf.high method
##       <dbl> <chr>
## 1         1 Pearson's product-moment correlation
##   alternative
##   <chr>
```

```
## 1 greater
```

- A: Die Logik-Leitung beeinflusst den IQ signifikant positiv.
- B: Es gibt keine Korrelation zwischen Logik-Leistung und IQ.
- C: Die Logik-Leistung und der IQ sind signifikant von Null unterschiedlich korreliert.
- D: Es gibt einen signifikanten, positiv linearen Zusammenhang zwischen Logik-Leistung und IQ.

Lösung

C und D könnte man so sagen, D hat aber natürlich mehr Informationsgehalt.

5.2 Einfache lineare Regression

5.2.1 Modellanpassung

Bei der einfachen linearen Regression werden anhand der paarweise vorhandenen Daten zweier Variablen X und Y die Parameter a und b der Vorhersagegleichung $\hat{Y} = bX + a$ so bestimmt, dass die Werte von Y (dem Kriterium) bestmöglich mit der Vorhersage $\hat{Y}$ aus den Werten von X (dem Prädiktor) übereinstimmen.

Als Maß für die Güte der Vorhersage wird die Summe der quadrierten Residuen $E = Y - \hat{Y}$, also der Abweichungen von vorhergesagten und Kriteriumswerten herangezogen.

Lineare Modelle wie das der Regression lassen sich mit lm() anpassen und so die Parameter a und b schätzen.

```
lm(formula= <Modellformel> , data=<Datensatz>)
```

Ein von lm() zurückgegebenes Objekt stellt ein deskriptives Modell der Daten dar, das in anderen Funktionen weiter verwendet werden kann.

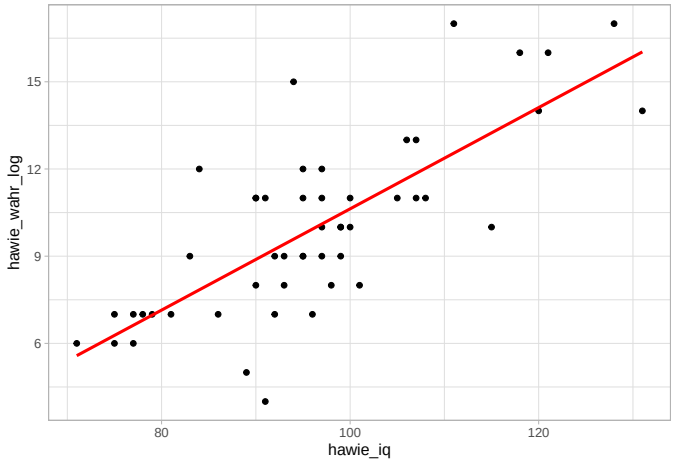
5.2.1.1 Beispiel für deskriptive Modellanpassung

Als Beispiel soll die Leistung auf der Skala zum wahrnehmungsgebundenen logischen Denken als Kriterium mit dem IQ als Prädiktor vorhergesagt werden.

```
(fitI <- lm(hawie_wahr_log ~ hawie_iq, data = df_wide))
```

```
##
## Call:
## lm(formula = hawie_wahr_log ~ hawie_iq, data = df_wide)
##
## Coefficients:
## (Intercept)     hawie_iq
##     -6.7909       0.1742
```

```
ggplot(df_wide, aes(x = hawie_iq, y = hawie_wahr_log)) +
  geom_point() +
  geom_smooth(formula =  y ~ x ,
              method = 'lm',col ='red',se = F)
```



5.2.2 β-Gewicht

Will man statt des b-Gewichtes das standardisierte β-Gewicht angeben, muss in der Modellformel z-transformiert werden.

Dafür könenn wir entweder alle Teile der `formula` `scale`n:

```
(fitZ <- lm(scale(hawie_wahr_log) ~ scale(hawie_iq),
            data = df_wide))
```

```
##
## Call:
## lm(formula = scale(hawie_wahr_log) ~ scale(hawie_iq), data = df_wide)
##
## Coefficients:
##     (Intercept)  scale(hawie_iq)
##      -4.971e-16        7.591e-01
```

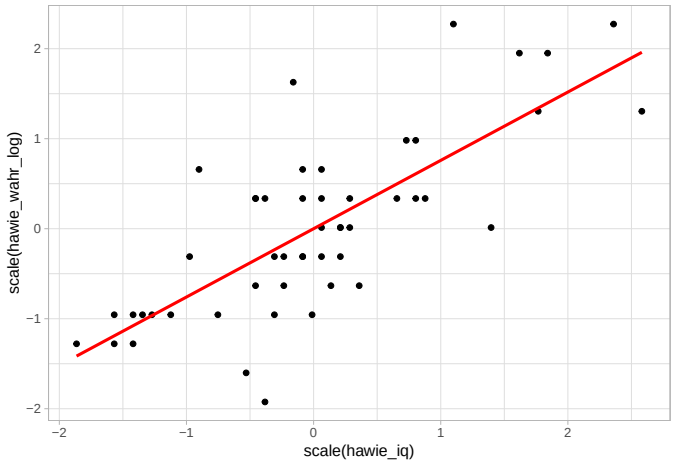
Oder wir benutzen die Index-pipe `%$%` aus dem `magrittr`-Paket und ein zwischengeschaltetes `mutate`, um die Skalierung ein bisschen übersichtlicher zu gestalten:

```
library(magrittr)
fitZ <- df_wide %>%
  mutate(hawie_wahr_log = scale(hawie_wahr_log),
         hawie_iq = scale(hawie_iq)) %$%
  lm(hawie_wahr_log ~ hawie_iq)
```

```
fitZ
```

```
##
## Call:
## lm(formula = hawie_wahr_log ~ hawie_iq)
##
## Coefficients:
## (Intercept)     hawie_iq
##  -4.971e-16    7.591e-01
```

```
df_wide %>%
  mutate(hawie_wahr_log = scale(hawie_wahr_log),
         hawie_iq = scale(hawie_iq)) %>%
  ggplot(aes(x = scale(hawie_iq),
                       y = scale(hawie_wahr_log))) +
    geom_point() +
    geom_smooth(formula =  y ~ x ,
                method = 'lm',col ='red',se = F)
```



5.2.3 weitere Parameter

`lm()` gibt eine Liste zurück, die ein deskriptives Modell der Daten darstellt.

R bietet weitere Funktionen um einzelne Parameter dieses Outputs auszulesen.

Zum Beispiel: `residuals()` zum Anzeigen der Residuen, `coef()` zur Ausgabe der geschätzten Modellparameter und `fitted()` für die vorhergesagten Werte.

```
residuals(fitZ)
```

```
##           1           2           3           4
## -0.132244120  0.553125355 -0.286404915  0.288579211
```

```
##             5             6             7             8
##  1.748985446 -0.244745860 -0.946823542  0.371897320
##             9            10            11            12
##  0.611492617 -0.075993249 -0.036450587 -0.357247601
##            13            14            15            16
## -0.034334195 -0.736411877  1.438547463 -0.469749341
##            17            18            19            20
## -0.146835935  0.682335303 -0.244745860 -0.905164488
##            21            22            23            24
## -1.634309409 -0.330180362 -0.200970413 -0.161427751
##            25            26            27            28
##  0.121942993  0.723994358 -0.721820061  0.682335303
##            29            30            31            32
## -0.146835935  0.136534808  0.721877965 -0.088468673
##            33            34            35            36
## -0.455157526  0.430264583 -1.198894262 -0.655210160
##            37            38            39            40
## -0.203086806 -0.103060488  0.234444733 -0.273929492
##            41            42            43            44
##  0.119826600  0.401080952 -0.384314840  0.065692122
##            45            46            47            48
##  1.342753931 -1.046849860  0.626084433  0.009441252
##            49            50
##  0.482282669  0.428148191
## attr(,"scaled:center")
## [1] 9.96
## attr(,"scaled:scale")
## [1] 3.096805
```

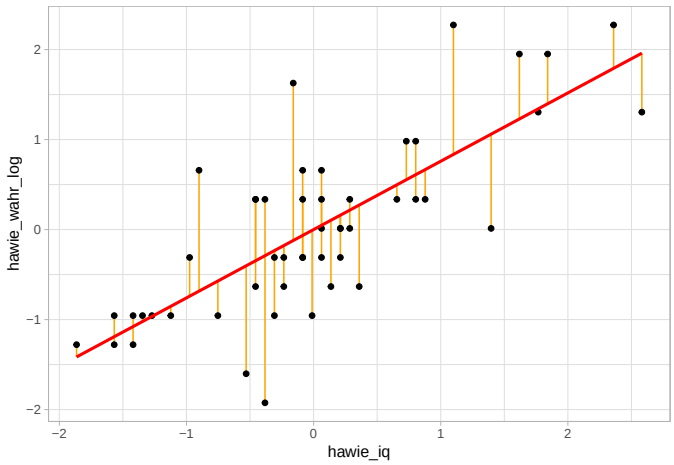
Im `broom`-Paket gibt es außerdem die `augment`-Funktion, die uns den zum Fitten genutzten Datensatz mit einer Reihe von Zusatzinfos ausgibt.

```
broom::augment(fitZ)
```

```
## # A tibble: 50 x 8
##    hawie_wahr_log[,1] hawie_iq[,1]  .fitted  .resid
##                 <dbl>        <dbl>    <dbl>   <dbl>
##  1             -0.310      -0.234  -0.178   -0.132
##  2              1.95        1.84    1.40     0.553
##  3             -0.633      -0.456  -0.347   -0.286
##  4              0.336       0.0622  0.0473   0.289
##  5              1.63       -0.160  -0.122    1.75
##  6             -0.310      -0.0860 -0.0653  -0.245
##  7             -0.956      -0.0119 -0.00900 -0.947
##  8              0.982       0.803   0.610    0.372
##  9              0.659       0.0622  0.0473   0.611
## 10             -0.310      -0.308  -0.234   -0.0760
```

```
##      .hat .sigma  .cooksd .std.resid
##     <dbl>  <dbl>    <dbl>      <dbl>
##  1 0.0211  0.664 0.000445     -0.203
##  2 0.0892  0.659 0.0380        0.881
##  3 0.0243  0.663 0.00241      -0.441
##  4 0.0201  0.663 0.00201       0.443
##  5 0.0205  0.613 0.0756        2.69
##  6 0.0202  0.664 0.00145      -0.376
##  7 0.0200  0.650 0.0216       -1.45
##  8 0.0332  0.662 0.00567       0.575
##  9 0.0201  0.659 0.00904       0.939
## 10 0.0219  0.665 0.000153     -0.117
## # ... with 40 more rows
```

```
fitZ %>%
  broom::augment() %>%
  ggplot(aes(x = hawie_iq,
             y = hawie_wahr_log))+
  geom_linerange(aes(ymin = .fitted, ymax = hawie_wahr_log), col = 'orange') +
    geom_point() +
    geom_smooth(formula =  y ~ x ,
                method = 'lm',col ='red',se = F)
```



5.3 Regressionsanalyse

5.3.1 Test-Hintergrund

Unter Voraussetzungen von Varianzhomogenität und Normalverteilung der Y-Werte für jeden möglichen Wert von X können Regressionskoeffizienten ähnlich wie Korrelationskoeffizienten auf Unterschiedlichkeit von 0 getestet werden.

Dazu wird genutzt, dass der Term $t = \frac{b}{\frac{s_{Y|X}}{s_X\sqrt{N-1}}}$ t_{N-1}-verteilt ist, wenn das tatsächliche b^* nicht unterschiedlich von 0 ist und für jeden Wert von X Y normalverteilt ist mit $\mu = b^*X + a^*$ und einer Varianz σ^2. Die Nullhypothese ist also $H_0 : b^* = 0$.

5.3.2 Test in R

Um zusätzliche Informationen (insbesondere inferenzstatistische Kennwerte) eine mit `lm()` erstellten Regressions-Modells zu erhalten, kann einfach `summary()` verwendet werden.

```
fitZ %>%
  summary()
```

```
##
## Call:
## lm(formula = hawie_wahr_log ~ hawie_iq)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.63431 -0.31924 -0.08223  0.42138  1.74899
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.971e-16  9.302e-02   0.000        1
## hawie_iq     7.591e-01  9.397e-02   8.078 1.68e-10
##
## (Intercept)
## hawie_iq    ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6578 on 48 degrees of freedom
## Multiple R-squared:  0.5762, Adjusted R-squared:  0.5674
## F-statistic: 65.26 on 1 and 48 DF,  p-value: 1.678e-10
```

Außerdem gibt es auch hier einen hübschen `broom`-Output, für den wir mit `conf.int` angeben können, Konfidenzintervalle für die Regressionsgewichte ausgeben lassen zu wollen:

```
fitZ %>%
  broom::tidy(conf.int = T)
```

```
## # A tibble: 2 x 7
##   term         estimate std.error statistic  p.value
##   <chr>           <dbl>     <dbl>     <dbl>    <dbl>
```

```
## 1 (Intercept) -4.97e-16    0.0930 -5.34e-15 1.00e+ 0
## 2 hawie_iq     7.59e- 1    0.0940  8.08e+ 0 1.68e-10
##   conf.low conf.high
##      <dbl>     <dbl>
## 1   -0.187     0.187
## 2    0.570     0.948
```

5.3.3 Aufgabe

```
fitZ %>%
  broom::tidy(conf.int = T)
```

```
## # A tibble: 2 x 7
##   term         estimate std.error statistic  p.value
##   <chr>           <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept) -4.97e-16    0.0930 -5.34e-15 1.00e+ 0
## 2 hawie_iq     7.59e- 1    0.0940  8.08e+ 0 1.68e-10
##   conf.low conf.high
##      <dbl>     <dbl>
## 1   -0.187     0.187
## 2    0.570     0.948
```

Wie lässt sich das Ergebnis interpretieren?

- A: Höhere IQ-Werte hängen mit höheren Logik-Leistungswerten zusammen.
- B: Es gibt keine Korrelation zwischen IQ-Werten und Logik-Leistung.
- C: Mit jedem Anstieg des IQ um eine Streuungs-Einheit, steigt die Vorhersage um 0.7590665 Streuungs-Einheiten.
- D: Man kann wegen der unzureichenden Berücksichtigung nicht-linearer Zusammenhänge keine Aussage treffen.

Lösung

C könnte man so sagen.

Chapter 6

Lineare Zusammenhänge II

6.1 Organisatorisches

6.1.1 Semesterplan

Einheit	Vorlesung	Übungswoche	Thema
1	23.04.21	keine Übung	Deskriptive Statistik Data Cleaning
2	07.05.21	KW 19	Hilfsmittel für die Inferenzstatistik Lineare Regression I
3	21.05.21	KW 21	Lineare Regression II
4	04.06.21	KW 23	t- Tests einfaktorielle Varianzanalyse
5	18.06.21	KW 25	zweifaktorielle Varianzanalyse
6	02.07.21	KW 27	Kontrasttests

6.1.2 Datensatz

Wir benutzen wieder den Datensatz `df_wide` aus der letzten Woche. Hier nochmal eine Übersicht:

Variable	Inhalt
‘group‘	Treatment-Gruppe
‘pre_skill‘	motorischer Skill vor dem Treatment
‘post_skill‘	motorischer Skill nach dem Treatment
‘hawie_iq‘	Intelligenz-Quotient aus HAWIE
‘hawie_wahr_log‘	Skalenwert wahrnehmungsgebundenes logisches Denken aus HAWIE

6.2 Multiple Lineare Regression

6.2.1 Verfahren

Bei der multiplen linearen Regression dienen mehrere quantitative oder dichotome Variablen X_j als Prädiktoren zur Vorhersage des quantitativen Kriteriums Y. Die Vorhersagegleichung hat hier die Form $\hat{Y} = a + b_1X_1 + ... + b_jX_j + ... + b_pX_p$, wobei die Koeffizienten a und b_j auf Basis der empirischen Daten zu ermitteln sind.

Als R-`formula` sieht das wie folgt aus:

$$\text{Kriterium} \sim \text{Prädiktor}_1 + \cdots + \text{Prädiktor}_p$$

Man versucht also, eine stetige Variable durch eine Linearkombination mehrerer Variablen vorherzusagen. Dabei ist aber meistens eher das Ausmaß des Zusammenhangs als die tatsächliche Vorhersage für neue Werte interessant.

6.2.2 Deskriptive Modellanpassung und Regressionsanalyse

Regression vom motorischen Skill nach dem Training auf IQ und wahrnehmungsgebundenes logisches Denken:

```
fit_post_il <- df_wide %>%
   lm(post_skill~hawie_iq+hawie_wahr_log,
      data=.)

fit_post_il
```

```
##
## Call:
## lm(formula = post_skill ~ hawie_iq + hawie_wahr_log, data = .)
##
## Coefficients:
##    (Intercept)        hawie_iq  hawie_wahr_log
##        5.75745        -0.05013         0.42397
```

Für standardisierte Gewichte wie vorher:

```
library(magrittr)
fit_post_il_z <- df_wide %>%
  mutate(across(where(is.numeric), ~scale(.))) %$%
   lm(post_skill~hawie_iq+hawie_wahr_log)
```

6.2.3 Darstellung

Mit der Funktion `scatter3d()` aus dem Paket `car` lassen sich die Daten dann dreidimensional mit Residuen plotten.

```
library(car)

scatter3d(post_skill~hawie_iq+hawie_wahr_log,
          data=df_wide)
```

6.3 Regressionsdiagnostik

6.3.1 Regressionsdiagnostik

Regressionsgleichungen sind sehr anfällig für verschiedene Klassen von Ausreißern. Ein Extremwert kann unter bestimmten Bedingungen die errechneten Koeffizienten extrem beeinflussen und verzerren.

Es gibt drei Klassen von diagnostischen Werten, die unterschiedliche Aspekte möglicher Verfälschung beleuchten.

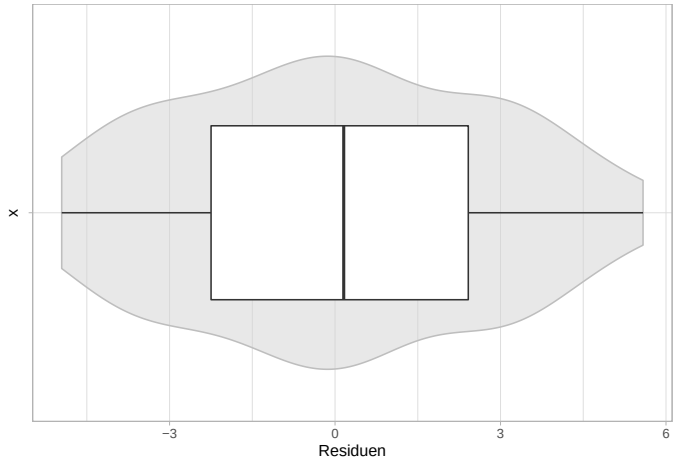
1. Abstand (mögliche Ausreißer im Wertebereich des Kriteriums)
2. Hebelwirkung (mögliche Ausreißer im Wertebereich der Prädiktoren)
3. Einfluss (Kombination von Abstand und Hebelwirkung)

http://omaymas.github.io/InfluenceAnalysis/ gibt es eine Shiny-App, mit der man daran rumspielen kann.

6.3.2 Abstand

Die Plausibilität des Abstandes lässt sich am Besten mit Hilfe der Residuen der Regression überprüfen. Dafür benutzen wir hier eine grafische Darstellung, um einen Überblick über deren Verteilung zu erlangen.

```
df_wide %>%
  mutate(Residuen = residuals(fit_post_il)) %>%
ggplot(aes(y = Residuen, x = '') ) +
  geom_violin(color = 'grey', fill = 'lightgrey',
              alpha = .5) +
  geom_boxplot(width = .5) +
  coord_flip()
```



6.3.3 Hebelwirkung

Hebelwirkung ist das Ausmaß, in dem ein Prädiktor-Wert ungewöhnlich in Bezug auf die restlichen Prädiktorwerte ist. Das für Aussagen darüber genutzte Maß sind die Hebelwerte. Bei der einfachen Regression sind Hebelwerte die Abweichung der einzelnen Prädiktorwerte von deren Mittelwert. Bei der multiplen Regression ist das nicht ganz so einfach.

In beiden Fällen bewegt sich der Hebelwert aber zwischen $\frac{1}{N}$ und 1.

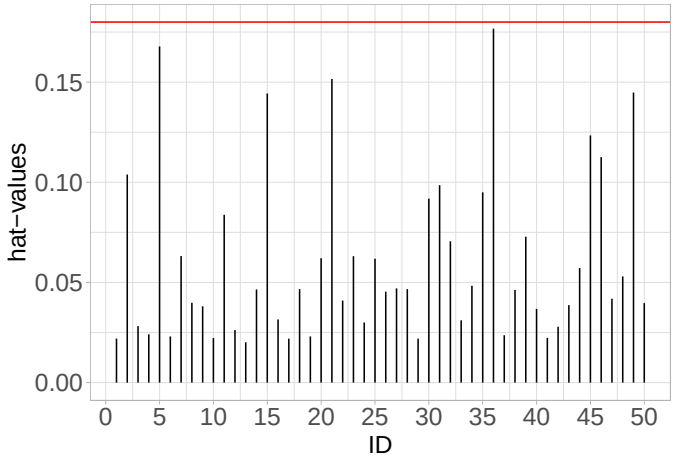
Diese Hebelwerte werden mit `hatvalues()` berechnet.

Faustregel: Bei p Einflussgrößen (Prädiktoren) und N Beobachtungen sind Fälle mit Hebelwerten von größer als $3 \cdot \frac{p+1}{N}$ problematisch.

```
h <- hatvalues(fit_post_il)
```

Mit einem Spikeplot lassen sich diese dann veranschaulichen.

```
tibble(hats = h,
       ID = 1:50) %>%
  ggplot(aes(x= ID, ymax = h, ymin= 0)) +
  geom_linerange() +
  geom_hline(yintercept = (3*(2+1)/50),col='red') +
  scale_x_continuous(breaks = seq(0,50,5)) +
  labs(y = 'hat-values')
```

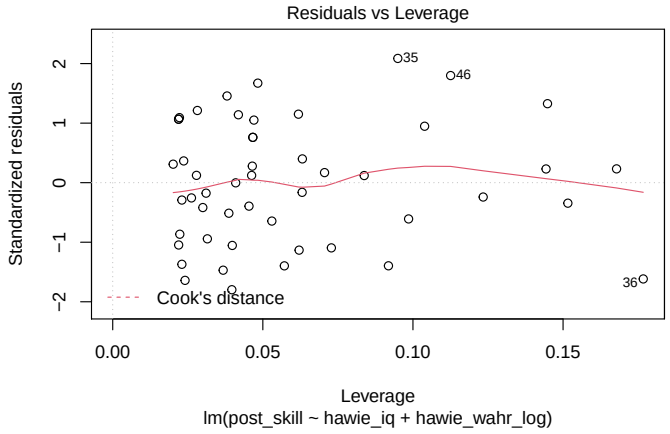


6.3.4 Einfluss

Unter Einfluss oder *influence* versteht man den Einfluss eines einzelnen Datenpunktes auf die gesamte Vorhersage. Er stellt also eine Kombination der vorher genannten Parameter dar.

Am besten lässt sich dieser mit folgender Funktion grafisch überprüfen:

```
plot(fit_post_il, which = 5)
```



Ganz hübsche Alternativen bieten auch die Funktion `influencePlot()` aus dem `car`-Paket:

```
influencePlot(fit_post_il)
```

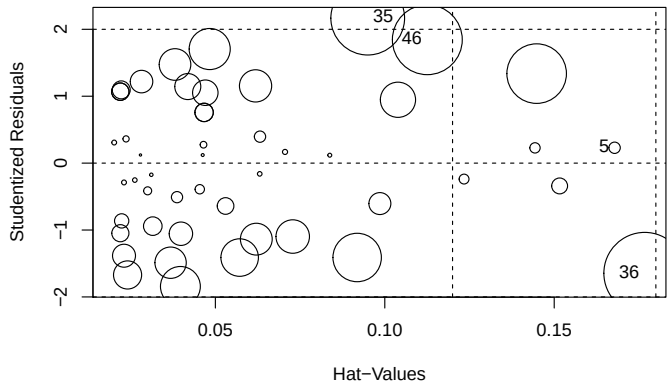


Table 6.1

StudRes	Hat	CookD
0.229	0.168	0.00361
2.17	0.095	0.152
-1.65	0.177	0.187
1.84	0.113	0.137

Außerdem können wir uns auch hier das `broom`-Paket benutzen, diesmal um uns alle diagnostischen Werte in einem praktischen Datensatz ausgeben zu lassen:

```
fit_post_il %>%
  broom::augment()
```

6.3.5 Test der Regressionskoeffizienten

Auch bei der multiplen linearen Regression lassen sich die Regressionskoeffizienten der einzelnen Prädiktoren jeweils auf die Nullhypothese testen, dass die jeweiligen "wahren Koeffizienten" ${b_i}^*$ gleich 0 sind. Die Schätzung der Streuung der Koeffizienten ist ein bisschen komplizierter als im einfachen Fall, deswegen sei hier nur erwähnt, dass es geht.

Die für jeden Prädiktoren gebildete Teststatistik $t = \frac{b_i}{s_{b_i}}$ ist dann bei Gültigkeit der Nullhypothese ($H_0 : {b_i}^* = 0$) t_{N-p-1}-verteilt, wobei p die Gesamtzahl der Prädiktoren und N die Gesamtzahl der Beobachtungen ist.

Der Test der Parameter auf Signifikanz läuft wie im einfachen Fall mit der Funktion `summary()`

```
summary(fit_post_il_z)
```

```
##
## Call:
## lm(formula = post_skill ~ hawie_iq + hawie_wahr_log)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.70753 -0.77446  0.05423  0.83241  1.92405
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)     6.812e-17  1.371e-01   0.000   1.0000
## hawie_iq       -2.331e-01  2.127e-01  -1.096   0.2787
## hawie_wahr_log  4.524e-01  2.127e-01   2.127   0.0387
##
## (Intercept)
## hawie_iq
## hawie_wahr_log *
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9692 on 47 degrees of freedom
## Multiple R-squared:  0.09891,    Adjusted R-squared:  0.06057
## F-statistic:  2.58 on 2 and 47 DF,  p-value: 0.0865
```

Oder auch wieder mit `broom::tidy()`:

```
broom::tidy(fit_post_il_z)
```

6.3.5.1 Aufgabe

Wie lässt sich das Ergebnis interpretieren?

1. Der IQ liefert einen signifikanten Beitrag zur Vorhersage des motorischen Skills nach dem Training.
2. Es gibt keine Korrelation zwischen IQ-Werten und dem motorischen Skill nach dem Training.
3. Größere Werte auf der Skala für wahrnehmungsgebundenes logisches Denken bewirken eine signifikante Steigerung des motorischen Skills nach dem Training.
4. Die Werte der Vorhersage steigen mit denen des wahrnehmungsgebundenen logischen Denkens und die Werte für wahrnehmungsgebundenes logisches Denken leisten einen signifikanten Beitrag zur Vorhersage des motorischen Skills nach der Intervention.

Antwort

Viertens kann man so sagen.

Gegen 3 spricht die Kausalinterpretation, 1 ist verkehrt (keine Signifikanz) und über 2 können wir mit dem Ergebnis des Gesamtmodells direkt keine Aussage treffen.

6.3.6 Test der Signifikanz von R^2

Man kann sich die Frage stellen, ob das Modell mit den gewählten Prädiktoren insgesamt das Kriterium gut vorhersagt. Das lässt sich am einfachsten bewerkstelligen, indem man den Determinationskoeffizienten R^2 auf die H_0 testet, dass der 'wahre' Koeffizient R^* der Population gleich 0 ist. ("$H_0 : R^* = 0$")

Getestet wird diese Hypothese mit der Teststatistik $F = \frac{(N-p-1)R^2}{p(1-R^2)}$, die $F_{p,N-p-1}$-verteilt ist. Dabei ist N wieder die Anzahl der Beobachtungen und p die Anzahl der Prädiktoren.

Um diesen Test durchzuführen können wir entweder auf den unteren Teil des `summary`-Outputs für ein Regressionsmodell gucken:

```
## Residual standard error: 0.9692 on 47 degrees of freedom
## Multiple R squared: 0.0989 , Adjusted R-squared: 0.0606
## F-statistic:  2.58 on 2 and 47 DF, p-value: 0.0865
```

Oder die `broom::glance()`-Funktion nutzen:

```
fit_post_il_z %>%
  broom::glance()
```

6.3.6.1 Aufgabe

Wie lässt sich das Ergebnis interpretieren?

1. Unser Modell ist ein sehr gutes Modell, es klärt einen signifikanten Teil der Varianz auf.
2. Unser Modell sagt den motorischen Skill nicht gut voraus.
3. Unser Modell klärt $\sim 10\%$ der Varianz auf.

Antwort

2 und 3 lassen sich so sagen.

6.3.7 Prüfung der Voraussetzungen

Für die Tests auf Signifikanz in der konventionellen Regressionsanalyse gelten die Voraussetzungen der Varianzhomogenität und der Normalverteiltheit der

Residuen für jede einzelne Prädiktor-Kombination. Wir setzen also wieder voraus, dass Y für jede Kombination der X_p normalverteilt ist mit $\mu = b_1^* X_1 + \cdots + b_p^* X_p + a^*$ und einer Varianz σ^2.

Zusätzlich zu den Voraussetzungen ist für die multiple Regression das Ausmaß der Multikollinearität der Prädiktoren relevant.

6.3.8 Grafische Prüfverfahren

Um die Anforderungen an die Messfehler der Regression heuristisch zu überprüfen, lassen sich verschiedene grafische Darstellungen heranziehen. Dazu benutzt man am besten eine standardisierte Form der Residuen. Zwei davon haben sich durchgesetzt, zum einen die

studentischen

$$E_{stud} = \frac{\frac{E}{s}}{(1 - \frac{1}{N} + h)^{\frac{1}{2}}}$$

und zum anderen die *standardisierten*

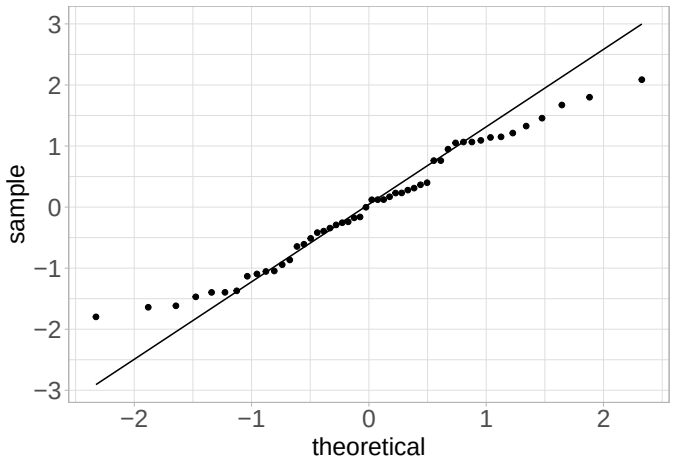
$$E_{stan} = \frac{Y - \hat{Y}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}}$$

Residuen. Die Residuen lassen sich dafür mit `rstudent()` oder `rstandard()` berechnen.

Hier wird die `broom::augment()`-Funktion praktisch, da wir in dem ausgegebenen Datensatz die Residuen und die vorhergesagten Werte praktisch aufbereitet haben

Die Verteilungseigenschaften können wir dann grafisch-heuristisch mit einem qq-Plot überprüfen:

```
fit_post_il %>%
  broom::augment() %>%
  ggplot(aes(sample = .std.resid)) +
  geom_qq() +
  geom_qq_line()
```

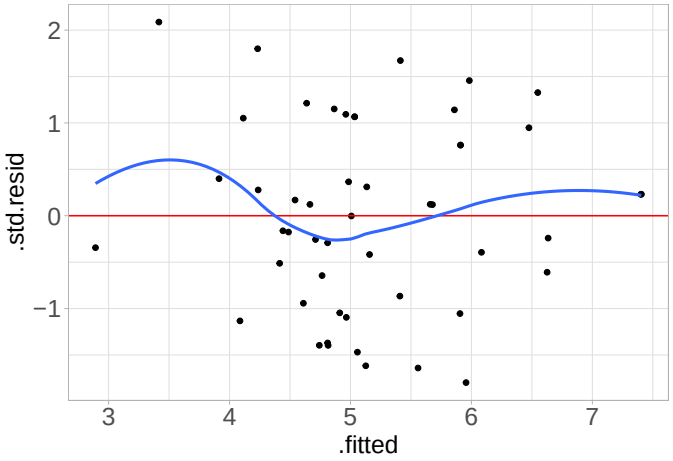


Je weiter die Punkte von der eingezeichneten Gerade abweichen, desto weniger können wir von einer Normalverteiltheit der Residuen ausgehen. Wie groß "noch akzeptable" Abweichung ist, ist ein Stück weit Gefühlssache.

Hier findet man eine kleine shiny-App, mit der an qq-plots rumgespielt werden kann:

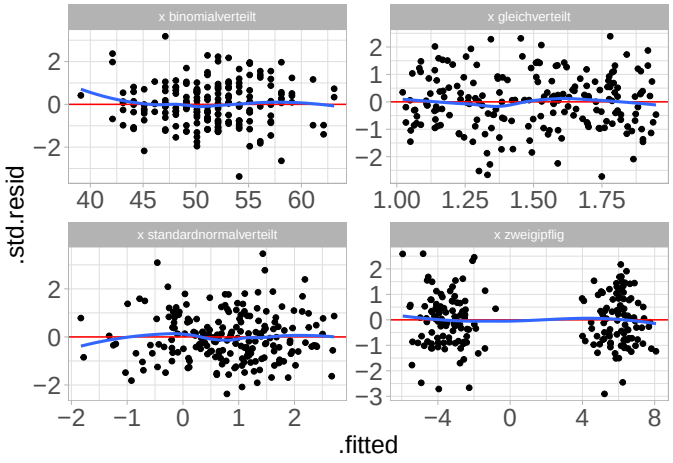
Die Voraussetzung der Varianzhomogenität lässt sich auch so verstehen, dass wir für Y für jede Werte-Kombination unserer Prädiktoren eine gleich große Varianz voraussetzen. Über den Verlauf unserer vorhergesagten Werte sollten wir also um den 0-Punkt ungefähr gleich (breit) streuende Residuen beobachten können. Die genaue Form ist dabei aber natürlich abhängig von der Verteilung der Prädiktoren. Wenn wir diesem Plot jetzt noch einen lokalen Schätzer des Mittelwerts hinzufügen, haben wir einen so genannten Spread-Level-Plot:

```
fit_post_il %>%
  broom::augment() %>%
  ggplot(aes(x = .fitted, y = .std.resid)) +
  geom_point() +
  geom_hline(yintercept = 0, color = 'red') +
  geom_smooth(formula = 'y~x',se = F, method = 'loess')
```

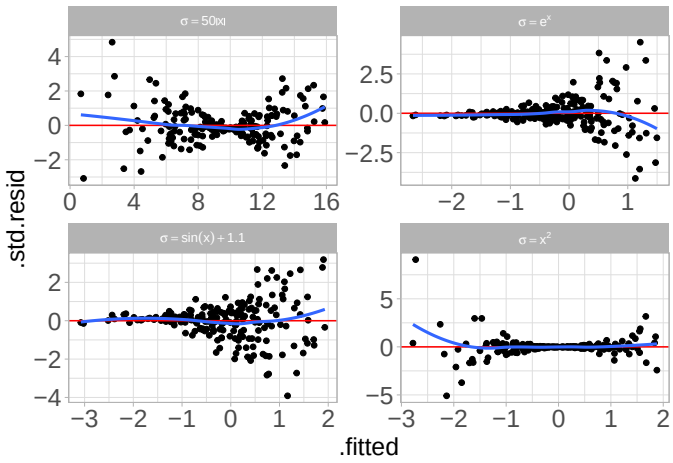


In unserem Fall haben wir scheinbar ein paar Ausreißer (wie vorher auch schon gesehen), sonst ist die Punktwolke aber unproblematisch. Die Abweichung der loess-Regression sieht oft dramatischer aus als es faktisch ist, vor allem bei wenigen Beobachtungen wie bei uns.

Man muss immer im Hinterkopf behalten, dass das genaue Bild stark von der Verteilung der Prädiktoren abhängt. So ist keins der folgenden Muster wirklich eine typische Punktwolke, trotzdem sind die Voraussetzungen überall gegeben:



Wirklich Grund zur Sorge sollten uns Bilder wie die folgenden geben:



6.3.9 Inferenstatistischer Test der Voraussetzungen

Inferenz-statistisch lässt sich die Normalverteilung der Residuen zum Beispiel mit dem Kolmogorov-Smirnov-Test überprüfen:

```
ks.test(x = rstudent(fit_post_il),
        y = 'pnorm')
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  rstudent(fit_post_il)
## D = 0.093678, p-value = 0.7726
## alternative hypothesis: two-sided
```

6.3.9.1 Aufgabe

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  rstudent(fit_post_il)
## D = 0.093678, p-value = 0.7726
## alternative hypothesis: two-sided
```

Wie lässt sich das Ergebnis interpretieren? 1. $1 - p$ ist kleiner als 30%, deswegen können wir keine Normalverteilung annehmen. 2. p ist größer als 20%; da wir eine Normalverteiltheit nicht ausschließen können, nehmen wir diese Voraussetzung als gegeben an. 3. Wir können gar nichts sagen, der Test aller Residuen auf einmal ergibt keinen Sinn.

Antwort

Zweitens ist die übliche Interpretation, auch wenn dem Herrn Andres hier der Dampf aus den Ohren steigt.

6.3.10 Multikollinearität

Multikollinearität liegt dann vor, wenn sich die Werte eines Prädiktors gut aus einer Linearkombination der übrigen Prädiktoren vorhersagen lassen.

Dies ist insbesondere dann der Fall, wenn Prädiktoren paarweise miteinander hoch korrelieren.

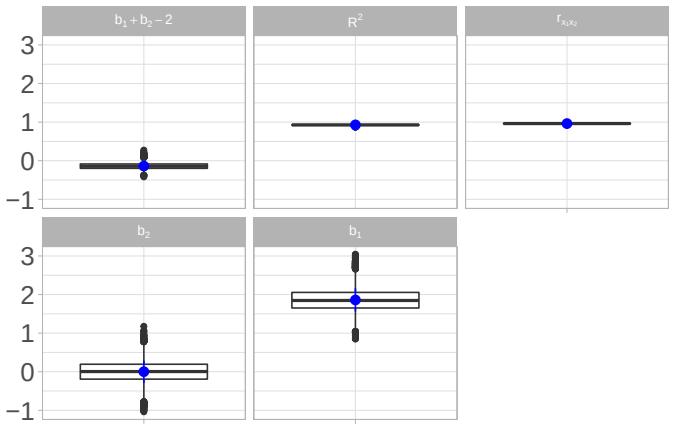
Für die multiple Regression hat dies weniger stabile Schätzungen der Koeffizienten als unerwünschte Konsequenz. Das heißt für die Praxis, dass die Regressionsgewichte schwer interpretierbar werden, sobald die entsprechenden Prädiktoren zu stark korrelieren, da von Stichprobe zu Stichprobe starke Änderungen zu erwarten sind. Modelle mit Multikolliniaren Prädiktoren haben aber meistens relativ stabile Determinationskoeffizienten, wenn uns also so oder so nur das Gesamtmodell interessiert, ist Multikollinearität kein allzu großes Problem.

Als kleines Beispiel sind hier Simulationsergebnisse von 10000 Regressionen mit jeweils korrelierten Prädiktoren:

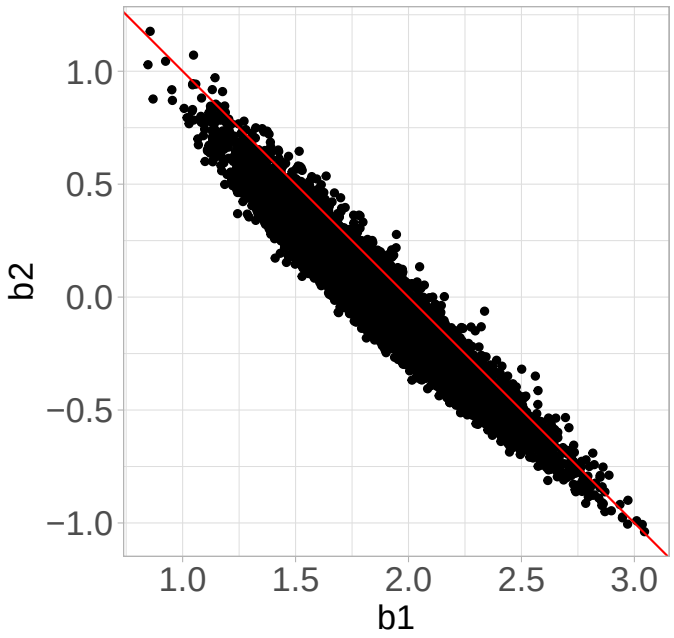
Die Regression wurden jeweils auf einem nach dem folgenden Schema simulierten Datensatz erstellt:

```
tibble(y = rnorm(100),
       x1 = y / 2 * runif(100,.5, 1.5),
       x2 = x1 * runif(100,.5, 1.5))
```

Insgesamt verteilen sich die Ergebnisse wie folgt:



Gewisse Linearkombinationen der Koeffizienten sind deutlich stabiler als die Koeffizienten allein, wie man hier am Beispiel von b_1+b_2+2 gut sehen kann. Welche Linearkombination gerade besonders stabil sein könnte, kann man einfach an einer Punktwolke wie der folgenden ablesen:



Paarweise lineare Zusammenhänge lassen sich anhand der Korrelationsmatrix der Prädiktoren prüfen.

```
df_wide %>%
  select(hawie_iq,
         hawie_wahr_log) %>%
  cor()
```

```
##                 hawie_iq hawie_wahr_log
## hawie_iq       1.0000000      0.7590665
## hawie_wahr_log 0.7590665      1.0000000
```

Faustregel: Korrelationen > 0.8 weisen auf starke Kollinearität hin.

Der `Varianzinflationsfaktor` $\text{VIF}_j = \frac{1}{1-R_j^2}$ jedes Prädiktors j liefert eine weitere Möglichkeit zur Kollinearitätsdiagnostik. Er kann mit der Funktion `vif()` aus dem Paket `car` berechnet werden.

```
library(car)
vif(fit_post_il)
```

```
##       hawie_iq hawie_wahr_log
##       2.359503       2.359503
```

Faustregel: VIF-Faktor $> 4 \rightarrow$ starke Multikollinearität

Table 6.2

post_skill	hawie_iq	hawie_wahr_log	.fitted	.resid	.hat	.sigma	.cooksd	.std.resid
2	93	9	4.91	-2.91	0.022	2.81	0.0082	-1.05
9	121	16	6.48	2.52	0.104	2.82	0.0347	0.948
8	90	8	4.64	3.36	0.0282	2.8	0.0142	1.21
1	97	11	5.56	-4.56	0.0241	2.76	0.0221	-1.64
8	94	15	7.41	0.595	0.168	2.84	0.00361	0.232
4	95	9	4.81	-0.811	0.023	2.84	0.000669	-0.292
5	96	7	3.91	1.09	0.0632	2.84	0.00358	0.399
3	107	13	5.91	-2.91	0.0398	2.81	0.0154	-1.05
10	97	12	5.98	4.02	0.0381	2.78	0.028	1.46
8	92	9	4.96	3.04	0.0222	2.81	0.00904	1.09
6	120	14	5.68	0.322	0.0838	2.84	0.000436	0.12
4	97	9	4.71	-0.711	0.0262	2.84	0.000589	-0.256
6	97	10	5.13	0.865	0.0201	2.84	0.000661	0.311
5	98	8	4.24	0.763	0.0465	2.84	0.00125	0.278
8	111	17	7.4	0.599	0.144	2.84	0.00298	0.23
2	99	9	4.61	-2.61	0.0315	2.82	0.00965	-0.943
8	99	10	5.03	2.97	0.0219	2.81	0.0085	1.07
8	90	11	5.91	2.09	0.0467	2.83	0.00945	0.761
1	95	9	4.81	-3.81	0.023	2.79	0.0148	-1.37
1	101	8	4.09	-3.09	0.0621	2.8	0.0283	-1.13
2	91	4	2.89	-0.892	0.152	2.84	0.00706	-0.344
5	108	11	5.01	-0.00743	0.041	2.84	1.03e-07	-0.0027
4	77	6	4.44	-0.442	0.0631	2.84	0.00059	-0.162
4	105	11	5.16	-1.16	0.03	2.84	0.0018	-0.418
8	77	7	4.87	3.13	0.0619	2.8	0.0291	1.15
5	95	12	6.08	-1.08	0.0454	2.84	0.00246	-0.394
7	92	7	4.11	2.89	0.047	2.81	0.0182	1.05
8	90	11	5.91	2.09	0.0467	2.83	0.00945	0.761
8	99	10	5.03	2.97	0.0219	2.81	0.0085	1.07

Table 6.3

term	estimate	std.error	statistic	p.value
(Intercept)	6.81e-17	0.137	4.97e-16	1
hawie_iq	-0.233	0.213	-1.1	0.279
hawie_wahr_log	0.452	0.213	2.13	0.0387

Table 6.4

term	estimate	std.error	statistic	p.value
(Intercept)	6.81e-17	0.137	4.97e-16	1
hawie_iq	-0.233	0.213	-1.1	0.279
hawie_wahr_log	0.452	0.213	2.13	0.0387

Table 6.5

r.squared	adj.r.squared	sigma	statistic	p.value	df	logLik	AIC	BIC	deviance	df.residual
0.0989	0.0606	0.969	2.58	0.0865	2	-67.8	144	151	44.2	47

Table 6.6

r.squared	adj.r.squared	sigma	statistic	p.value	df	logLik	AIC	BIC	deviance	df.residual
0.0989	0.0606	0.969	2.58	0.0865	2	-67.8	144	151	44.2	47

Table 6.7

y	x1	x2
0.193	0.143	0.0999
1.02	0.735	0.816
0.813	0.21	0.186
0.954	0.361	0.218
2.08	0.931	1.01
1.61	0.71	1.02
0.19	0.142	0.152
-0.999	-0.336	-0.251
-0.615	-0.203	-0.142
3.32	1.41	1.54
0.922	0.545	0.431
1.23	0.467	0.32
-0.216	-0.142	-0.115
-1.34	-0.401	-0.443
-0.304	-0.153	-0.163
-0.0779	-0.0267	-0.0381
-0.193	-0.0836	-0.0737
0.123	0.0864	0.0447
-0.279	-0.16	-0.122
0.758	0.306	0.297
0.0606	0.0302	0.0388
-0.165	-0.0743	-0.0917
0.0715	0.0189	0.0136
0.649	0.22	0.261
1.1	0.359	0.369
-1.37	-0.553	-0.538
-0.127	-0.0864	-0.0849
1.08	0.804	1.09
-1.45	-0.736	-0.967

Bibliography

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