Übungszettel Arbeitsumgebung und Datentransformation

M.Psy.205, Dozent: Dr. Peter Zezula

Johannes Brachem (johannes.brachem@stud.uni-goettingen.de)

Links

Übungszettel als PDF-Datei zum Drucken

Übungszettel mit Lösungen

Lösungszettel als PDF-Datei zum Drucken

Der gesamte Übugszettel als .Rmd-Datei (Zum Downloaden: Rechtsklick > Speichern unter…)

Hinweise zur Bearbeitung

1. Bitte beantworten Sie die Fragen in einer .Rmd Datei. Sie können Sie über `Datei > Neue Datei > R Markdown...` eine neue R Markdown Datei erstellen. Den Text unter dem *Setup Chunk* (ab Zeile 11) können Sie löschen.
2. Informationen, die Sie für die Bearbeitung benötigen, finden Sie auf der Website der Veranstaltung
3. Zögern Sie nicht, im Internet nach Lösungen zu suchen. Das effektive Suchen nach Lösungen für R-Probleme im Internet ist tatsächlich eine sehr nützliche Fähigkeit, auch Profis arbeiten auf diese Weise. Die beste Anlaufstelle dafür ist der R-Bereich der Programmiererplattform Stackoverflow
4. Auf der Website von R Studio finden Sie sehr hilfreiche Übersichtszettel zu vielen verschiedenen R-bezogenen Themen. Ein guter Anfang ist der Base R Cheat Sheet

Tipp der Woche

Mit der Tastenkombination `ctrl` + `alt` + `i` (Windows) oder `cmd` + `alt` + `i` (Mac) können Sie per Knopfdruck ein neues R-Code-Feld (*Chunk*) in R-Markdown Dateien erstellen.

Aufgabe 1: R Markdown

a) Neue Markdown Datei öffnen

1. Laden Sie unter diesem Link unter diesem Link unsere Vorlage für die Erstellung von R-Markdown Dokumenten zur Bearbeitung unserer Übungszettel herunter.(Hinweis: Sie können auch eine neue .Rmd Datei erstellen, indem Sie auf `Datei > Neue Datei > R Markdown...` klicken. Mit unserer Vorlage ist der Start aber vermutlich einfacher.)

2. Speichern Sie die Datei unter einem sinnvollen Namen in einem sinnvollen Ordner ab. In diesem Ordner sollten Sie bestenfalls in der Folge alle .Rmd-Dateien für die Bearbeitung der Übungszettel speichern. Öffnen Sie nun die Datei.
3. Lassen Sie die Datei mit `Strg` + `Shift` + `K` (Windows) oder `Cmd` + `Shift` + `K` (Mac) rendern. Sie sollten nun im "Viewer" unten rechts eine "schön aufpolierte" Version ihrer Datei sehen.

b) Code-Chunks benutzen

1. Erstellen Sie einen R-Code-Chunk mit `ctrl` + `alt` + `i` (Windows) oder `cmd` + `alt` + `i` (Mac)
2. Tippen Sie `5 + 3` in den grau hinterlegten Code-Chunk. Führen Sie die Zeile aus, indem Sie `strg` + `enter` (Windows) oder `cmd` + `enter` drücken. Das Ergebnis, `8` sollte Ihnen in der Konsole nun angezeigt werden.
3. Tippen Sie `test <- 5 + 3` in eine neue Zeile in den Code-Chunk und führen Sie auch diese Zeile aus. Das Ergebnis sollte Ihnen diesmal nicht direkt angezeigt werden, stattdessen haben Sie im *Workspace* oben rechts ein neues Objekt namens "test".
4. Tippen Sie nun `test` in eine neue Zeile in den Code-Chunk und führen Sie sie aus. Nun sollte Ihnen das Ergebnis wieder in der Konsole (unten links) angezeigt werden.
5. Tippen Sie nun `# test anzeigen` in die **gleiche Zeile wie in der vorherigen Aufgabe** und führen Sie die Zeile erneut aus. Sie können sehen, dass die keinen Effekt hatte: `#` kennzeichnet *Kommentare*, alles was hinter einem `#` steht, wird nicht ausgerechnet. Dies ist nützlich für (kurze!) Erklärungen. Längere Erklärungen sollten außerhalb der *Chunks* als normaler Text geschrieben werden.
6. Klicken Sie auf `Tools > Global Options` und wählen Sie auf der linken Seite "R Markdown" aus. Überprüfen Sie, ob das Häkchen bei "*Show Output inline for all Markdown documents*" gesetzt ist. Wenn ja, **entfernen Sie es**. Das Häkchen sollte **nicht** gesetzt sein.

c) Arbeitsverzeichnis setzen

Wenn Sie Dateien in R einlesen möchten, oder aus R abspeichern möchten, ist es wichtig, dass Sie ein Arbeitsverzeichnis verwenden. Das ist der Ordner auf ihrem Computer, in dem R nach Dateien sucht, und in dem R Dateien abspeichert.

1. Setzen Sie das Arbeitsverzeichnis, indem Sie oben auf `Session > Set Working Directory > To Source File Location` klicken.
2. Betrachten Sie die Konsole (unten links). Sie sehen dort einen Befehl, der ähnlich wie dieser aussieht: `setwd("P:/mv")`
3. Kopieren Sie diesen Befehl und fügen ihn in den dafür vorgesehenen Code-Chunk am Anfang ihrer .Rmd-Datei ein. Nun können Sie immer, wenn Sie die Datei erneut öffnen, diesen Befehl ausführen und dadurch automatisch das richtige Arbeitsverzeichnis setzen. **Achtung**: Das funktioniert natürlich nur, wenn Sie immer den gleichen Ordner auf Ihrem Computer verwenden.

Lösung

(Bitte folgen Sie den Anweisungen in der Aufgabenstellung und melden Sie sich beim Problemen.)

Aufgabe 2: Pakete

1. Installieren Sie das Paket-System `tidyverse`, indem Sie `install.packages("tidyverse")` ausführen. Zu diesem System gehören mehrere sehr nützliche Pakete, z.B. `dplyr` für Datenaufbereitung

und `ggplot2` für Plots. Hinweis: Auf den PCs im CIP-Pool müssen Sie diesen Befehl nicht ausführen, die Pakete sind installiert.

2. Laden Sie das Paket-System `tidyverse` mit dem Befehl `library(tidyverse)`. Sie sollten nun diese Anzeige bekommen, die Ihnen sagt, welche Pakete standardmäßig zum `tidyverse` gehören. Diese werden automatisch mitgeladen. Schreiben Sie den Syntaxbefehl zum Laden in den dafür vorgesehenen Code-Chunk am Anfang der .Rmd-Datei.

Lösung

```
# der Befehl unten wird nicht ausgeführt, weil im Chunk eval = FALSE gesetzt ist
# und daher nicht ausgeführt wird beim Rendern
# install.packages() macht in einem RMD, das von seiner Konzeption her wir immer wieder laufen können s
install.packages("tidyverse")
```

Teilaufgabe 1

```
library(tidyverse)
```

Teilaufgabe 2

Aufgabe 3: Mit Objekten umgehen

a) Erstellen von Objekten

Erstellen Sie die folgenden Objekte:

1. Einen Vektor `a1`, der die Zahlen von 1 bis 5 enthält.
2. Einen Vektor `a2` der die Zahlen von 0 bis 4.5 in Schritten von 0.5 enthält (d.h. er beinhaltet 10 Zahlen).
3. Einen Vektor `a3` der Länge 10, der Zahlen von 0 bis 85 in gleichmäßigen abständen enthält.
4. Eine Matrix, die die Zahlen 1 bis 9 in drei Zeilen enthält. Sie sollte so aussehen:

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

5. Einen `tibble`-Datensatz namens `a5`, der die beiden Vektoren `a2` und `a3` enthält.

Hinweis: `tibble`s sind moderne Formen von `data.frame`s. Wir empfehlen, dass Sie immer `tibble`s verwenden. Wenn Sie Daten mit den `readr`-Befehlen `read_csv()` oder `read_delim()` einlesen, werden diese automatisch als `tibble` gespeichert.

6. Einen `tibble`-Datensatz namens `a6`, der die beiden Vektoren `a2` und `a3` enthält. Bei diesem `data frame` sollten Sie bei der Erstellung die beiden Vektoren umbenennen: Nennen Sie `a2` “motivation” und `a3` “result”
7. Eine Liste `a7`, die alle bisher erstellen Elemente beinhaltet.

Lösung

Teilaufgabe 1 Schneller als die Schreibweise per Hand ist hier die Eingabe mit dem Doppelpunkt: `:`.

`0:5` bedeutet zum Beispiel *0 bis 5* und ist das gleiche wie `c(1, 2, 3, 4, 5)`.

```
a1 <- c(1, 2, 3, 4, 5) # lange Schreibweise
a1 <- 1:5 # kurze Schreibweise
```

Teilaufgabe 2 Der Befehl seq() mit dem Argument "by" ist hier hilfreich

```
a2 <- seq(from = 0, to = 4.5, by = 0.5)
```

Teilaufgabe 3 Der Befehl seq() mit dem Argument "length.out" ist hier hilfreich

```
a3 <- seq(from = 0, to = 85, length.out = 10)
```

Teilaufgabe 4 Der Befehl matrix() hilft hier weiter. `byrow = TRUE` sorgt für die Zeilenweise Befüllung und `ncol = 3` sorgt dafür, dass die Matrix 3 Spalten hat.

```
a4 <- matrix(1:9, byrow = TRUE, ncol = 3)
```

Teilaufgabe 5 Mit dem Befehl tibble() können Sie tibble-Datensätze erstellen. Die Objekte, die Sie damit in einem Datensatz verbinden, müssen die gleiche Anzahl von Zeilen haben.

```
a5 <- tibble(a2, a3)
```

Teilaufgabe 6 Sie können die einzelnen Elemente eines tibbles direkt bei der Erstellung benennen.

```
a6 <- tibble(motivation = a2, result = a3)
```

Teilaufgabe 7 Mit dem Befehl list() können Sie Listen erstellen. Diese können Objekte verschiedner Art und Länge enthalten. Listen begegnen uns als PsychologInnen im Alltag meist nur als Output von Analysen. Wir müssen nicht oft selbst welche erstellen, aber es lohnt sich, sie zu kennen.

```
a7 <- list(a1, a2, a3, a4, a5, a6)
```

b) Auf Elemente zugreifen

1. Greifen Sie auf das dritte Element des Vektors `a1` zu.
2. Greifen Sie auf die dritte Spalte der Matrix `a4` zu.
3. Greifen Sie abermals auf die dritte Spalte der Matrix `a4` zu. Sorgen Sie diesmal dafür, dass die Dimensionen der Matrix erhalten bleiben. Das Ergebnis sollte wie folgt aussehen:

```
##      [,1]
## [1,]    3
## [2,]    6
## [3,]    9
```

4. Greifen Sie auf die zweite bis sechste Zeile des data.frame `a6` zu.
5. Greifen Sie auf die Spalte / Variable `result` im data.frame `a6` zu.

Lösung

```
a1[3]
```

Teilaufgabe 1

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

Teilaufgabe 2 In Matrizen (und Datensätzen/tibbles) können Sie ebenfalls mit eckigen Klammern hinter dem Objektnamen auf die einzelnen Elemente des Objektes zugreifen. Beachten Sie hier: Die Zahl vor dem Komma gibt die ausgewählten Zeilen an, die Zahl nach dem Komma die Spalten. Wenn Sie alle Zeilen auswählen wollen, geben Sie nichts vor dem Komma ein.

```
a4[,3]
```

```
## [1] 3 6 9
```

Teilaufgabe 3 Wenn Sie nur eine Zeile einer Matrix auswählen, macht `R` automatisch einen Vektor daraus. Um dieses Verhalten zu verhindern, können Sie die Option `drop = FALSE` an den Auswahl-Befehl anhängen.

```
a4[,3, drop = FALSE]
```

```
##      [,1]
## [1,]    3
## [2,]    6
## [3,]    9
```

Teilaufgabe 4 Der Doppelpunkt-Operator funktioniert auch für den Zugriff auf ELemente.

```
a6[2:6,]
```

```
## # A tibble: 5 x 2
##   motivation result
##        <dbl>  <dbl>
## 1        0.5   9.44
## 2        1    18.9
## 3        1.5  28.3
## 4        2    37.8
## 5        2.5  47.2
```

Teilaufgabe 5 Mit dem `$`-Operator können Sie auf benannte Spalten innerhalb von tibbles/Datensätzen und Listen zugreifen.

```
a6$result
```

```
##  [1]  0.000000  9.444444 18.888889 28.333333 37.777778 47.222222 56.666667 66.111111 75.555556 85.00
```

c) Objekte manipulieren

1. Addieren Sie `3` zu jedem Element des Vektors `a1` und speichern das Ergebnis unter dem Namen `c1`
2. Erstellen Sie einen neuen tibble `c2`, indem Sie nur die Zeilen 3 und 4 aus dem tibble `a6` auswählen.

Lösung

```
c1 <- a1 + 3
```

Teilaufgabe 1

Teilaufgabe 2 Mit dem Zuweisungspfeil `<-` speichern Sie das Ergebnis eines Befehls unter einem Namen ab. So können Sie auch bestehende Objekte überschreiben. *Hinweis*: Sie müssen nicht den Doppelpunkt `:` benutzen, alternativ können Sie auch `a6[c(3,4),]` schreiben. Diese Schreibweise ist äquivalent und kann zusätzlich benutzt werden, um Zeilen auszuwählen, die nicht benachbart sind, z.B. würde `a6[c(1,3,5),]` die Zeilen 1, 3 und 5 auswählen.

```
c2 <- a6[3:4,]
```

Aufgabe 4: Logische Abfragen

1. Greifen Sie auf alle Zeilen des Datensatzes `a6` zu, bei denen `result` größer als 50 ist.
2. Greifen Sie auf alle Zeilen des Datensatzes `a6` zu, bei denen `result` zwischen 30 und 70 liegt.
3. Greifen Sie auf alle Zeilen des Datensatzes `a6` zu, bei denen `result` *kleiner* als 30, oder *größer* als 70 ist.
4. Nutzen Sie sum() und eine logische Abfrage, um die Anzhal von Zeilen in `a6` zu erhalten, bei denen `result` größer als 20 ist.

Lösung

Teilaufgabe 1 Der > Operator heißt *größer als*.

```
a6[a6$result > 50, ]
```

```
## # A tibble: 4 x 2
##   motivation result
##        <dbl>  <dbl>
```

```
## 1        3     56.7
## 2        3.5   66.1
## 3        4     75.6
## 4        4.5   85
```

Teilaufgabe 2 Der `&` Operator heißt *und*.

```
a6[30 < a6$result & a6$result < 70,]
```

```
## # A tibble: 4 x 2
##   motivation result
##        <dbl>  <dbl>
## 1        2     37.8
## 2        2.5   47.2
## 3        3     56.7
## 4        3.5   66.1
```

Teilaufgabe 3 Der `|` Operator heißt *oder*.

```
a6[30 > a6$result | a6$result > 70,]
```

```
## # A tibble: 6 x 2
##   motivation result
##        <dbl>  <dbl>
## 1        0     0
## 2        0.5   9.44
## 3        1    18.9
## 4        1.5  28.3
## 5        4    75.6
## 6        4.5  85
```

Teilaufgabe 4 Wenn man logische Werte addiert, gil "TRUE" = 1 und "FALSE" = 0.

```
sum(a6$result > 20)
```

```
## [1] 7
```

Aufgabe 5: Daten einlesen

1. Laden Sie den Datensatz *seatbelts.csv* hier herunter. Legen Sie diese Datei in einen Ordner namens "data" in Ihrem Ordner für die Statistik-Übungszettel.
2. Lesen Sie den Datensatz *seatbelts.csv* mit dem Befehl `read_csv()` ein und speichern Sie ihn in R unter dem Namen `seatbelt_data`. Tipp: Wenn Sie die Datei wie in 5.1 beschrieben in einem Unterordner "data" in Ihrem Arbeitsverzeichnis abgespeichert haben, ist der Pfad zur Datei für Sie nun `"data/seatbelts.csv"`.

Hinweis: `read_csv` und `read.csv` sind unterschiedliche Befehle. `read_csv` ist die modernere Variante und wird von uns empfohlen. Mehr können Sie bei Interesse hier erfahren.

3. Schauen Sie sich kurz die Bedeutung der Variablen in diesem Datensatz an. Unter diesem Link finden Sie eine kurze Beschreibung.

Lösung

Teilaufgabe 1 (Bitte folgen Sie den Anweisungen im Aufgabentext.)

Teilaufgabe 2 Die Meldung, die Sie ausgegeben bekommen, zeigt Ihnen an, in welchem *Objekttyp* R die einzelnen Variablen in Ihrem Datensatz eingelesen hat. Das Programm versucht automatisch, einen passenden Typ zu wählen. Falls Sie unerwartete Probleme mit Ihren Daten bekommen, können Sie den Objekttyp im Funktionsaufruf von `read_csv()` oder seinen verwandten `read_delim()` manuell auswählen. In diesem Fall hilft Ihnen ein Blick in die Hilfe durch `?read_csv` zum Argument `col_types`.

Beim Einlesen von Daten müssen Sie darauf achten, dass Sie im richtigen Verzeichnis arbeiten. Hier liegt die Datei `seatbelts.csv` im Unterordner "data" unseres Arbeitsverzeichnisses, deshalb geben wir das bei dem Funktionsaufruf mit an.

```
# seatbelt_data <- read_csv("data/seatbelts.csv")
# as the above depends on the real download, we do it via direct access
seatbelt_data <- readr::read_csv("https://md.psych.bio.uni-goettingen.de/mv/data/div/seatbelts.csv")
```

```
## Rows: 192 Columns: 10
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): month
## dbl (9): DriversKilled, drivers, front, rear, kms, PetrolPrice, VanKilled, law, year
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Teilaufgabe 3 (Bitte folgen Sie den Anweisungen im Aufgabentext.)

Aufgabe 6: Mit der Pipe %>% umgehen.

1. Schreiben Sie den folgenden Befehl um, so dass Sie %>% verwenden, um den Code besser lesbar zu machen.

```
filter(seatbelt_data, year == 1969)
```

2. Schreiben Sie den folgenden Befehl ebenfalls mit der Pipe %>% um, um ihn besser lesbar zu machen.

```
select(filter(seatbelt_data, year == 1969, kms > 10000), DriversKilled)
```

3. Nehmen Sie Ihren Code von Aufgabe 6.1 und nutzen Sie ihn, um ein *Subset* von `seatbelt_data` namens `seatbelt_data_69` zu erzeugen, in dem nur die Daten aus dem Jahr 1969 enthalten sind.

Lösung

Teilaufgabe 1 Es lohnt sich zusätzlich, nach jeder *Pipe* einen Zeilenumbruch einzusetzen. So ist ihr Code noch einmal deutlich besser lesbar.

```
seatbelt_data %>%
  filter(year == 1969)
```

```
## # A tibble: 12 x 10
##    DriversKilled drivers front  rear   kms PetrolPrice VanKilled   law  year month
##            <dbl>   <dbl> <dbl> <dbl> <dbl>       <dbl>     <dbl> <dbl> <dbl> <chr>
##  1           107    1687   867   269  9059       0.103        12     0  1969 Jan
##  2            97    1508   825   265  7685       0.102         6     0  1969 Feb
##  3           102    1507   806   319  9963       0.102        12     0  1969 Mar
##  4            87    1385   814   407 10955       0.101         8     0  1969 Apr
##  5           119    1632   991   454 11823       0.101        10     0  1969 May
##  6           106    1511   945   427 12391       0.101        13     0  1969 Jun
##  7           110    1559  1004   522 13460       0.104        11     0  1969 Jul
##  8           106    1630  1091   536 14055       0.104         6     0  1969 Aug
##  9           107    1579   958   405 12106       0.104        10     0  1969 Sep
## 10           134    1653   850   437 11372       0.103        16     0  1969 Oct
## 11           147    2152  1109   434  9834       0.103        13     0  1969 Nov
## 12           180    2148  1113   437  9267       0.102        14     0  1969 Dec
```

Teilaufgabe 2 Innerhalb eines `filter()` Befehls können Sie mehrere Bedingungen per Komma aneinanderreihen.

```
seatbelt_data %>%
  filter(year == 1969, kms > 10000) %>%
  select(DriversKilled)
```

```
## # A tibble: 7 x 1
##   DriversKilled
##           <dbl>
## 1            87
## 2           119
## 3           106
## 4           110
## 5           106
## 6           107
## 7           134
```

Teilaufgabe 3 Wie alle anderen Befehle auch, müssen Operationen mit der *Pipe* durch den Zuweisungspfeil <- abgespeichert werden, um sie zu sichern. Ansonsten werden Sie nur angezeigt, aber nichts weiter geschieht.

```
seatbelt_data_69 <- seatbelt_data %>%
  filter(year == 1969)
```

Aufgabe 7: Mit `dplyr` umgehen

Hinweise

- Bitte nutzen Sie wann immer möglich die piping-Schreibweise mit %>% für diese und ähnliche Aufgaben. Dies ist in unser aller Interesse, da so die Lesbarkeit und Nachvollziehbarkeit ihres Codes maximiert wird.
- Lassen Sie sich über `?<funktion>` die Hilfe-Seite zu den vorgeschlagenen Funktionen anzeigen. Dort sehen sie alle Befehle, die Sie in einer Funktion verwenden können. Zum Beispiel zeigt Ihnen `?arrange` die Hilfe-Seite zum Befehl `arrange` an. (Da es mehrere Funktionen namens "arrange" gibt, müssen Sie zunächst per Klick auswählen, zu welcher Sie sich die Hilfe anzeigen lassen wollen.)

Aufgaben

1. Nutzen Sie den Befehl `filter()`, um sich alle Daten aus dem Monat *Januar* im Datensatz `seatbelt_data` anzeigen zu lassen.
2. Nutzen Sie den Befehl `select()`, um sich außerdem nur die Daten zu `DriversKilled`, `law`, `year` und `month` anzeigen zu lassen.
3. Nutzen Sie den Befehl `mutate()`, um eine neue Variable namens `id` zu erstellen, die die Zahlen von 1 bis 192 enthält. Überschreiben Sie `seatbelt_data` mit dem Ergebnis.
4. Nutzen Sie den Befehl `arrange()`, um den Datensatz anhand von `id` *abwärts* zu sortieren.
5. Nutzen Sie den Befehl `group_by()`, um im unbearbeiteten Datensatz `seatbelt_data` die Daten nach Jahren zu gruppieren. Nutzen Sie anschließend `summarize()`, um sich die mittlere Anzahl von Todesfällen pro Jahr anzeigen zu lassen.
6. Nutzen Sie den Befehl `rename()`, um die Variable `DriversKilled` in `drivers_killed` und `PetrolPrice` in `petrol_price` umzubennen.

Lösung

```
seatbelt_data %>%
  filter(month == "Jan")
```

Teilaufgabe 1

```
## # A tibble: 16 x 10
##    DriversKilled drivers front  rear   kms PetrolPrice VanKilled   law  year month
##            <dbl>   <dbl> <dbl> <dbl> <dbl>       <dbl>     <dbl> <dbl> <dbl> <chr>
##  1           107    1687   867   269  9059      0.103         12     0  1969 Jan
##  2           125    1752   925   316  9130      0.101         14     0  1970 Jan
##  3           134    2030   944   356 10266      0.0967        17     0  1971 Jan
##  4           159    2080  1005   359 10803      0.0907        14     0  1972 Jan
##  5           144    2097   903   354 11692      0.0864        13     0  1973 Jan
##  6           113    1608   731   262 11616      0.0924         8     0  1974 Jan
##  7           122    1577   664   278 11249      0.133         12     0  1975 Jan
##  8           102    1473   704   266 12177      0.114         14     0  1976 Jan
##  9           112    1648   714   291 11972      0.101         10     0  1977 Jan
## 10           148    1956   889   366 12387      0.0884        14     0  1978 Jan
```

```
## 11           114    1813   796   306 11196      0.0845        10     0  1979 Jan
## 12           115    1665   748   306 14027      0.104          7     0  1980 Jan
## 13           111    1474   704   284 15226      0.105          8     0  1981 Jan
## 14           115    1456   595   238 13601      0.113          4     0  1982 Jan
## 15           120    1494   619   281 16231      0.113          8     0  1983 Jan
## 16            92    1357   483   296 16224      0.118          5     1  1984 Jan
```

```
seatbelt_data %>%
  select(DriversKilled, law, year, month)
```

Teilaufgabe 2

```
## # A tibble: 192 x 4
##    DriversKilled   law  year month
##            <dbl> <dbl> <dbl> <chr>
##  1           107     0  1969 Jan
##  2            97     0  1969 Feb
##  3           102     0  1969 Mar
##  4            87     0  1969 Apr
##  5           119     0  1969 May
##  6           106     0  1969 Jun
##  7           110     0  1969 Jul
##  8           106     0  1969 Aug
##  9           107     0  1969 Sep
## 10           134     0  1969 Oct
## # ... with 182 more rows
```

```
seatbelt_data <- seatbelt_data %>%
  mutate(id = 1:192)
```

Teilaufgabe 3

```
seatbelt_data %>%
  arrange(desc(id))
```

Teilaufgabe 4

```
## # A tibble: 192 x 11
##    DriversKilled drivers front  rear   kms PetrolPrice VanKilled   law  year month    id
##            <dbl>   <dbl> <dbl> <dbl> <dbl>       <dbl>     <dbl> <dbl> <dbl> <chr> <int>
##  1           154    1763   721   491 18149       0.116         7     1  1984 Dec     192
##  2           137    1737   711   490 18564       0.116         4     1  1984 Nov     191
##  3           120    1575   641   408 19928       0.116         7     1  1984 Oct     190
```

```
##  4           122    1444   643   429 20195       0.114         7     1  1984 Sep     189
##  5            96    1284   644   521 21626       0.115         5     1  1984 Aug     188
##  6            79    1222   601   472 21486       0.115         7     1  1984 Jul     187
##  7            90    1185   522   465 19976       0.115         6     1  1984 Jun     186
##  8            87    1297   586   441 19584       0.115         6     1  1984 May     185
##  9            84    1110   548   375 19759       0.115         3     1  1984 Apr     184
## 10            81    1282   513   349 18539       0.116         4     1  1984 Mar     183
## # ... with 182 more rows
```

```
seatbelt_data %>%
  group_by(year) %>%
  summarise(mean_deaths = mean(DriversKilled))
```

Teilaufgabe 5

```
## # A tibble: 16 x 2
##     year mean_deaths
##    <dbl>       <dbl>
##  1  1969       117.
##  2  1970       133.
##  3  1971       138.
##  4  1972       147.
##  5  1973       144.
##  6  1974       129.
##  7  1975       118.
##  8  1976       120.
##  9  1977       119.
## 10  1978       127.
## 11  1979       123.
## 12  1980       112.
## 13  1981       112.
## 14  1982       123.
## 15  1983        99.8
## 16  1984       102.
```

```
seatbelt_data <- seatbelt_data %>%
  rename(drivers_killed = DriversKilled,
         petrol_price = PetrolPrice)
```

Teilaufgabe 6

Aufgabe 8: Long & Wide Data

Es ist für fast alle Vorgänge in R am besten, wenn Sie Daten im *long*-Format haben. Das liegt einfach an der Art, wie R funktioniet.

Wide-Data sieht so aus:

```
## # A tibble: 4 x 4
##    subj gender    t1    t2
##   <int> <chr>  <dbl> <dbl>
## 1     1 m          2     6
## 2     2 m          3     5
## 3     3 f          4     4
## 4     4 f          5     3
```

Long-Data sieht so aus (dies sind die gleichen Daten):

```
## # A tibble: 8 x 4
##    subj gender time  value
##   <int> <chr>  <chr> <dbl>
## 1     1 m      t1        2
## 2     2 m      t1        3
## 3     3 f      t1        4
## 4     4 f      t1        5
## 5     1 m      t2        6
## 6     2 m      t2        5
## 7     3 f      t2        4
## 8     4 f      t2        3
```

In der Psychologie sind unsere Roh-Daten oft im *wide*-Format. Das Paket `tidyr` bietet die Möglichkeit, diese Daten einfach in das *long*-Format zu bringen. Genaueres dazu können Sie hier finden.

Aufgaben

1. Führen Sie den unten stehenden Befehl aus, um den Datensatz `wide_data` zu erzeugen.

```
wide_data <- tibble(subj = 1:4,
                    gender = c("m", "m", "f", "f"),
                    t1 = c(2,3,4,5),
                    t2 = c(6,5,4,3))
```

2. Nutzen Sie den Befehl `gather()` mit den Optionen `key` und `value`, um einen Datensatz `long_data` zu erzeugen. Dieser sollte genau so aussehen, wie im Beispiel oben.

Lösung

```
wide_data <- tibble(subj = 1:4,
                    gender = c("m", "m", "f", "f"),
                    t1 = c(2,3,4,5),
                    t2 = c(6,5,4,3))
```

Teilaufgabe 1

```r
long_data <- wide_data %>%
  gather(t1:t2, key = time, value = value)
```

Teilaufgabe 2

Aufgabe 9: Rendern (knit)

Lassen Sie die Datei wie in Aufgabe 1 a)3 mit `Strg` + `Shift` + `K` (Windows) oder `Cmd` + `Shift` + `K` (Mac) rendern. Sie sollten nun im "Viewer" unten rechts eine "schön aufpolierte" Version ihrer Datei sehen. Falls das klappt: Herzlichen Glückwunsch! Ihr Code kann vollständig ohne Fehlermeldung gerendert werden. Falls nicht: Nur Mut, das wird schon noch! Vielleicht schaffen wir es ja in der Übung gemeinsam.

Literatur

Anmerkung: Diese Übungszettel basieren zum Teil auf den "Smart Alex" Aufgaben aus dem Lehrbuch *Discovering Statistics Using R* (Field, Miles & Field, 2012). Sie wurden für den Zweck dieser Übung modifiziert, und der verwendete R-Code wurde aktualisiert.

Field, A., Miles, J., & Field, Z. (2012). *Discovering Statistics Using R*. London: SAGE Publications Ltd.

Version: 21 April, 2022 08:40