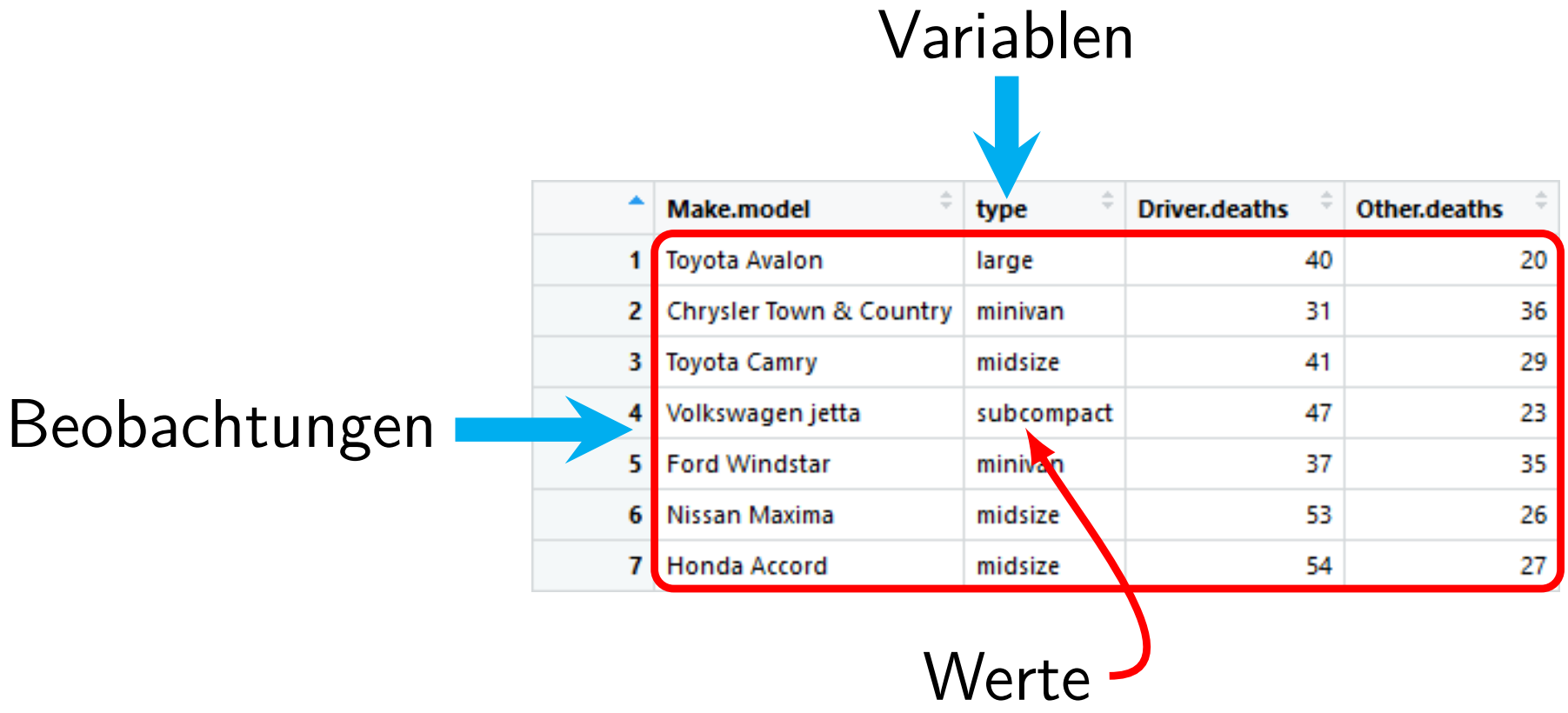
Spezielle Themen: tidy, plotly, shiny

Andreas Mändle

27.01.2020

R Tidy

Tidy data: Die Grundidee, dass der Datenaufbau dem Schema `Zeile=Beoachtung`, `Spalte=Variable` und `Zelle=Wert` folgt



	Make.model	type	Driver.deaths	Other.deaths
1	Toyota Avalon	large	40	20
2	Chrysler Town & Country	minivan	31	36
3	Toyota Camry	midsize	41	29
4	Volkswagen jetta	subcompact	47	23
5	Ford Windstar	minivan	37	35
6	Nissan Maxima	midsize	53	26
7	Honda Accord	midsize	54	27

Tidy packages

Installieren der wesentlichen Packages aus dem `tidyverse`:

```
install.packages("tidyverse")
install.packages("glue")
install.packages("broom")
install.packages("lubridate")
```

Die Pipe

Das `tidyverse`-Package lädt die zunächst wichtigsten Packages:

```
library(tidyverse)
```

Glue

Einfaches Verknüpfen von Zeichenketten:

```
library(glue)
vname <- "Fred"
nname <-"Feuerstein"
glue('Mein Name ist {vname} {nname}.')
```

```
## Mein Name ist Fred Feuerstein.
```

Siehe auch: https://github.com/tidyverse/glue

Broom

Säubern von non-tidy-Ausgaben:

```
library(broom)
# Erstelle Regressionsmodell:
lmfit <- lm(mpg ~ wt, mtcars)
# Sauber machen:
tidy(lmfit)
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)    37.3      1.88       19.9  8.24e-19
## 2 wt             -5.34     0.559      -9.56 1.29e-10
```

Dias Ergebnis ist ein `tibble`. Damit lässt sich leichter Arbeiten, als mit dem Output von `summary(lmfit)` – probiert es aus.

Siehe auch: https://cran.r-project.org/web/packages/broom/vignettes/broom.html

Lubridate

Arbeiten mit Datums- und Zeitangaben:

```
library(lubridate)
bday <- dmy("14/10/1990")
wday(bday, label = TRUE)
```

```
## [1] So
## Levels: So < Mo < Di < Mi < Do < Fr < Sa
```

```
year(bday) <- 2020
wday(bday, label = TRUE)
```

```
## [1] Mi
## Levels: So < Mo < Di < Mi < Do < Fr < Sa
```

Datumsangaben z.B. auch so: `dmy("14101979")` oder `ydm("79-14-10")`. Siehe auch: https://lubridate.tidyverse.org/

Die Pipe

Die Pipe %>% aus dem `magrittr`-Package erlaubt es, das Verschachteln von Funktionen und Argumenten als Pipeline auszudrücken:

```r
# Aus
plot(spline(aggregate(Temp ~ Month,
                      data = airquality,
                      mean)), type="l")
# wird
aggregate(Temp ~ Month, data = airquality, mean) %>%
  spline %>%
  plot(type="l")
```

- Die Pipe setzt das Objekt links als das erste Argument in der Funktion rechts ein.
- Um der logischen Struktur zu folgen liest man nun von links nach rechts (statt: von innen nach außen)

Das dplyr-Package

Alternatives Datenhandling:

- `select()`: wählt Spalten aus einem data.frame aus
  - `select(airquality, Wind)`
- `filter()`: wählt Zeilen anhand einer Bedingung aus
  - `filter(airquality, Temp > 90)`
- `mutate()`: Erstellt/überschreibt eine Spalte
  - `mutate(airquality, Temp = round((Temp-35)*5/9, 1))`
- `summarize`: aggregiert Zeilen mittels einer Funktion
- `group_by()`: gruppiert nach Werten einer Spalte
  - `summarize(group_by(airquality, Month), mean(Temp))`
- `drop_na()`: Entfernt `NA` aus gegebenen Spalten
  - `drop_na(airquality, Ozone, Solar.R)`

Vergleich: dplyr- und base-Befehle

dplyr-Befehl	base-Befehle
`select(...)`	`subset(select=...), [,...]`
`filter(...)`	`subset(subset=...), [...,]`
`mutate(...)`	`mydata$... <-...`

`group_by(...) %>%`
`  summarize(fun(...))`
	`aggregate(mydata,`
	`  by=..., FUN=fun),`
	`  apply-Befehle`

`drop_na(...)`	`na.omit(mydata, cols=...)`
`map(...)`	`lapply(mydata,...)`

dplyr-Beispiel

```
airquality %>%
  filter(Day<=15) %>%
  select(Wind, Temp, Month) %>%
  mutate(Temp = round((Temp-35)*5/9, 1)) %>%
  group_by(Month) %>%
  summarize(mw_Temp = mean(Temp, na.rm = TRUE),
            sd_Temp = sd(Temp, na.rm = TRUE))
```

- nimm den `airquality`-Datensatz
- wähle die Zeilen, in denen `Day<=15`
- wähle die Spalten `Wind`, `Temp` und `Month`
- Ersetze Temp durch entsprechende Werten in °C
- gruppiere nach `Month`
- berechne `mean` und `sd` (für die Gruppen)

dplyr-Beispiel

Month	mw_Temp	sd_Temp
5	17.08000	3.127573
6	26.44667	3.844080
7	27.52667	2.878856
8	27.82000	2.276338
9	25.81333	4.458197

dplyr-Beispiel 2

```r
data("survey", package = "MASS")

pulse_g <- survey %>%
  drop_na(Sex, Age, Pulse) %>%
  mutate(age_group = case_when(
                    Age <= 19 ~ "16-19",
                    Age <= 24 ~ "20-24",
                    Age <= 35 ~ "25-35"
                  )) %>%
  group_by(Sex, age_group) %>%
  summarize(mean_pulse = mean(Pulse),
            n = n(),
            se = sd(Pulse, na.rm = TRUE)/sqrt(n),
            ci_lower = mean_pulse - se * 1.96,
            ci_upper = mean_pulse + se * 1.96) %>%
  mutate_if(is.numeric, ~round(., 2))
```

dplyr-Beispiel 2

Wir haben gruppenweise den Mittelwert und Konfidenzintervalle berechnet:

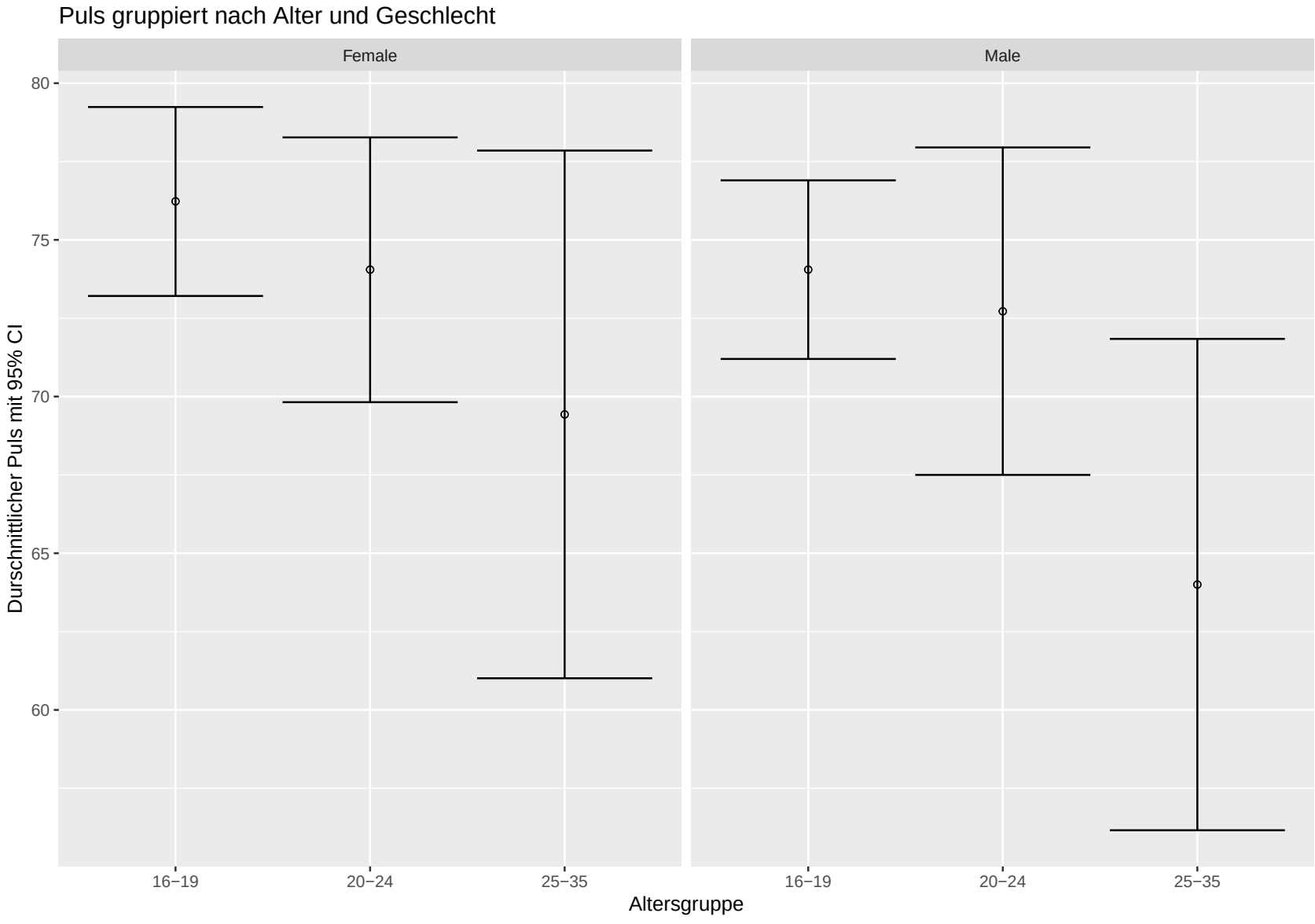
Sex	age_group	mean_pulse	n	se	ci_lower	ci_upper
Female	16-19	76.23	62	1.54	73.21	79.24
Female	20-24	74.05	22	2.15	69.82	78.27
Female	25-35	69.43	7	4.30	61.01	77.85
Female	NA	74.00	4	0.82	72.40	75.60
Male	16-19	74.05	59	1.45	71.20	76.90
Male	20-24	72.72	29	2.66	67.50	77.95
Male	25-35	64.00	4	4.00	56.16	71.84
Male	NA	73.25	4	2.87	67.63	78.87

dplyr-Beispiel 2

Die Daten können z.B. mit ggplot2 dargestellt werden: Wir enfernen vorher nur noch die NAs aus age_group.

```
pulse_g %>%
  drop_na(age_group) %>%
  ggplot(aes(x = age_group)) +
  geom_errorbar(aes(ymin = ci_lower,
                    ymax = ci_upper)) +
  geom_point(aes(y = mean_pulse), shape = 21) +
  facet_wrap(~Sex) +
  labs(title = "Puls gruppiert nach Alter/Geschlecht",
       x = "Altersgruppe",
       y = "Durschnittlicher Puls mit 95% CI")
```

dplyr-Beispiel 2



tibble-Beispiel

Ein `tibble` ist ein besonderer `data.frame` im Tidyversum. Bsp.:

```
tibble(
  x = 1:7,
  y = seq(0,1,length.out=7),
  r = rnorm(7)
)
```

x	y	r
1	0.0000000	1.6944323
2	0.1666667	-0.1228939
3	0.3333333	1.8584557
4	0.5000000	0.9385246
5	0.6666667	-0.7738027
6	0.8333333	-1.1479749
7	1.0000000	-1.1571860

tibble erzeugen mit tribble

Mit der Funktion `tribble` erzeugt man ein `tibble` aus Daten in Tabellenform:

```
dat <- tribble(
  ~Kategorie, ~Merkmal1, ~Merkmal2,
  "A",          1,         1/0,
  "B",          2,         1/Inf,
  "C",        3.5,         exp(1),
  "D",        4/7,         pi
)
```

Kategorie	Merkmal1	Merkmal2
A	1.0000000	Inf
B	2.0000000	0.000000
C	3.5000000	2.718282
D	0.5714286	3.141593

gather-Beispiel

Mit gather bringt man die Daten in eine gestapelte Struktur:

```
dat_long <- dat %>%
  gather(Gruppe, Wert)
```

Gruppe	Wert
Kategorie	A
Kategorie	B
Kategorie	C
Kategorie	D
Merkmal1	1
Merkmal1	2
Merkmal1	3.5
Merkmal1	0.571428571428571
Merkmal2	Inf
Merkmal2	0
Merkmal2	2.71828182845905
Merkmal2	3.14159265358979

gather-Beispiel

Der dritte Parameter legt fest, welche Spalten gestapelt werden sollen:

```
dat_long <- dat %>%
  gather(Gruppe, Wert, 2:3)
```

Kategorie	Gruppe	Wert
A	Merkmal1	1.0000
B	Merkmal1	2.0000
C	Merkmal1	3.5000
D	Merkmal1	0.5714
A	Merkmal2	Inf
B	Merkmal2	0.0000
C	Merkmal2	2.7183
D	Merkmal2	3.1416

gather-Beispiel 2

Das Gleiche nochmal anhand der `iris`-Daten:

```
head(iris)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

gather-Beispiel 2

Stapeln der Spalten 1 bis 4 als Paar von `Variable` und `Wert`:

```
iris_long <- iris %>%
  gather(Variable, Wert, 1:4)
```

Species	Variable	Wert
setosa	Sepal.Length	5.1
setosa	Sepal.Length	4.9
setosa	Sepal.Length	4.7
setosa	Sepal.Length	4.6
setosa	Sepal.Length	5.0
setosa	Sepal.Length	5.4

gather-Beispiel 2

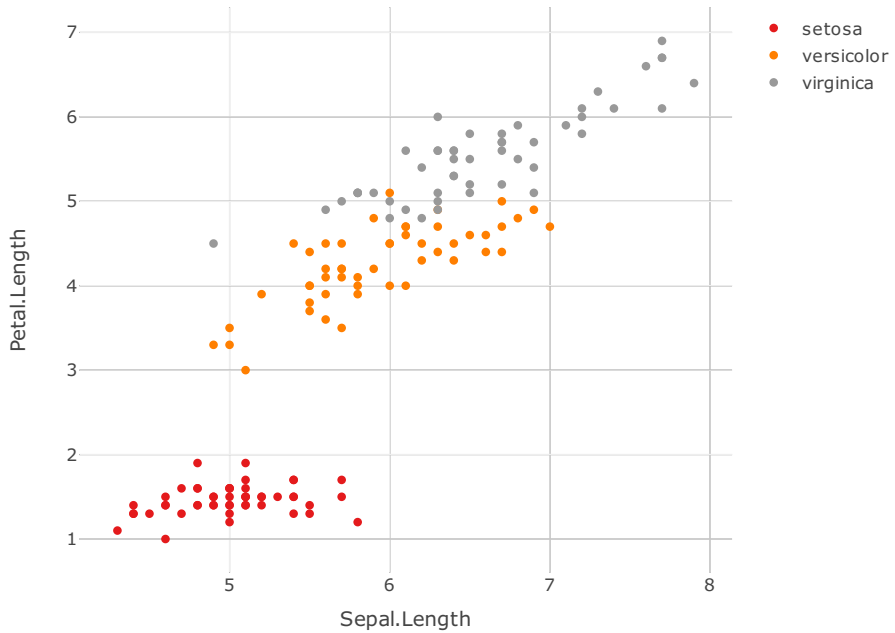
Dies Daten kann man nun weiter aggregieren, z.B:

```
iris_long %>%
  group_by(Species, Variable) %>%
  summarize(mean = mean(Wert))
```

Species	Variable	mean
setosa	Petal.Length	1.462
setosa	Petal.Width	0.246
setosa	Sepal.Length	5.006
setosa	Sepal.Width	3.428
versicolor	Petal.Length	4.260
versicolor	Petal.Width	1.326

plotly - ein erster Scatterplot

```
library(plotly)
plot_ly(data = iris,
        x = ~Sepal.Length, y = ~Petal.Length,
        color = ~Species, colors = "Set1")
```



Tipp: Zum Finden geeigneter Farbpaletten nutze http://colorbrewer2.org/.

3d-Scatterplot

Folgender Code lädt die `hills`-Daten und ergänzt diese um Angaben zu Leverege/potenziellen Ausreißern:

```
data(hills, package="MASS")
model3 <- lm(time~dist+climb-1,data=hills)
myhills <- hills %>%
  rownames_to_column(var = "hill") %>%
  mutate(cooksD=cooks.distance(model3),
         isOutlier=if_else(cooksD > 4/n(),
                           "Outlier", "kein Outlier"))
```

hill	dist	climb	time	cooksD	isOutlier
Greenmantle	2.5	650	16.083	0.0001001	kein Outlier
Carnethy	6.0	2500	48.350	0.0101948	kein Outlier
Craig Dunain	6.0	900	33.650	0.0029008	kein Outlier
Ben Rha	7.5	800	45.600	0.0016323	kein Outlier
Ben Lomond	8.0	3070	62.267	0.0243992	kein Outlier
Goatfell	8.0	2866	73.217	0.0001178	kein Outlier

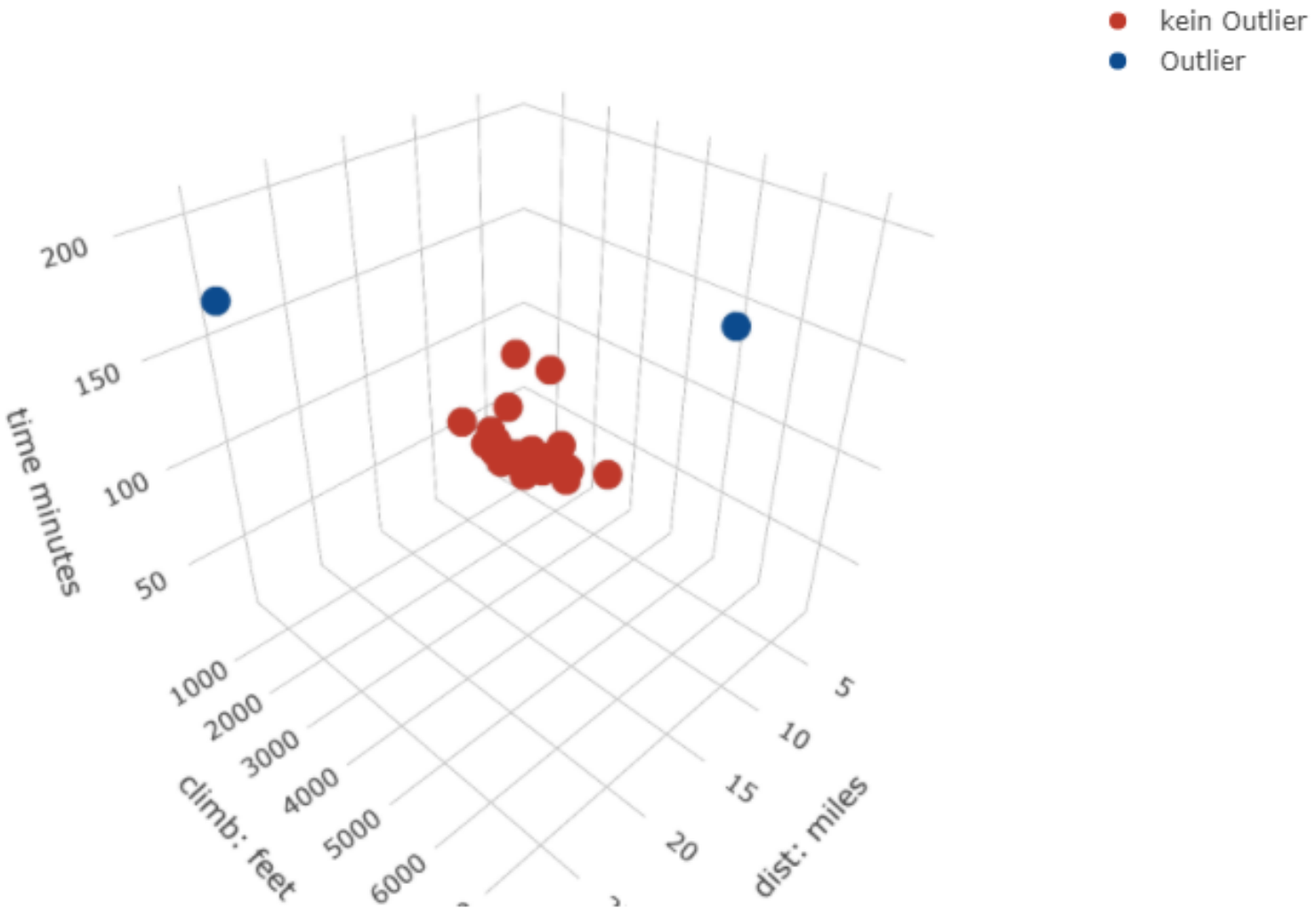
3d-Scatterplot

```
p <- plot_ly(myhills, x = ~dist, y = ~climb, z = ~time,
             color = ~isOutlier,
             colors = c('#BF382A', '#0C4B8E'),
             type = "scatter3d", mode='markers',
             text= rownames(hills)) %>%
  layout(scene = list(xaxis=list(title='dist: miles'),
                      yaxis=list(title='climb: feet'),
                      zaxis=list(title='time minutes')))
print(p) # oder nur p
```

Beachte die pipe aus dem Tidyversum. Probiere auch:

- `colors = c('green','yellow'),`
- `marker = list(color = ~cooksD, colorscale = c('white', 'red'), showscale = TRUE)`
  (die vorigen Zuweisungen für `color` und `colors` weglassen!)
- `mode='text'` (mit und ohne Nutzung von `marker`)

3d-Scatterplot



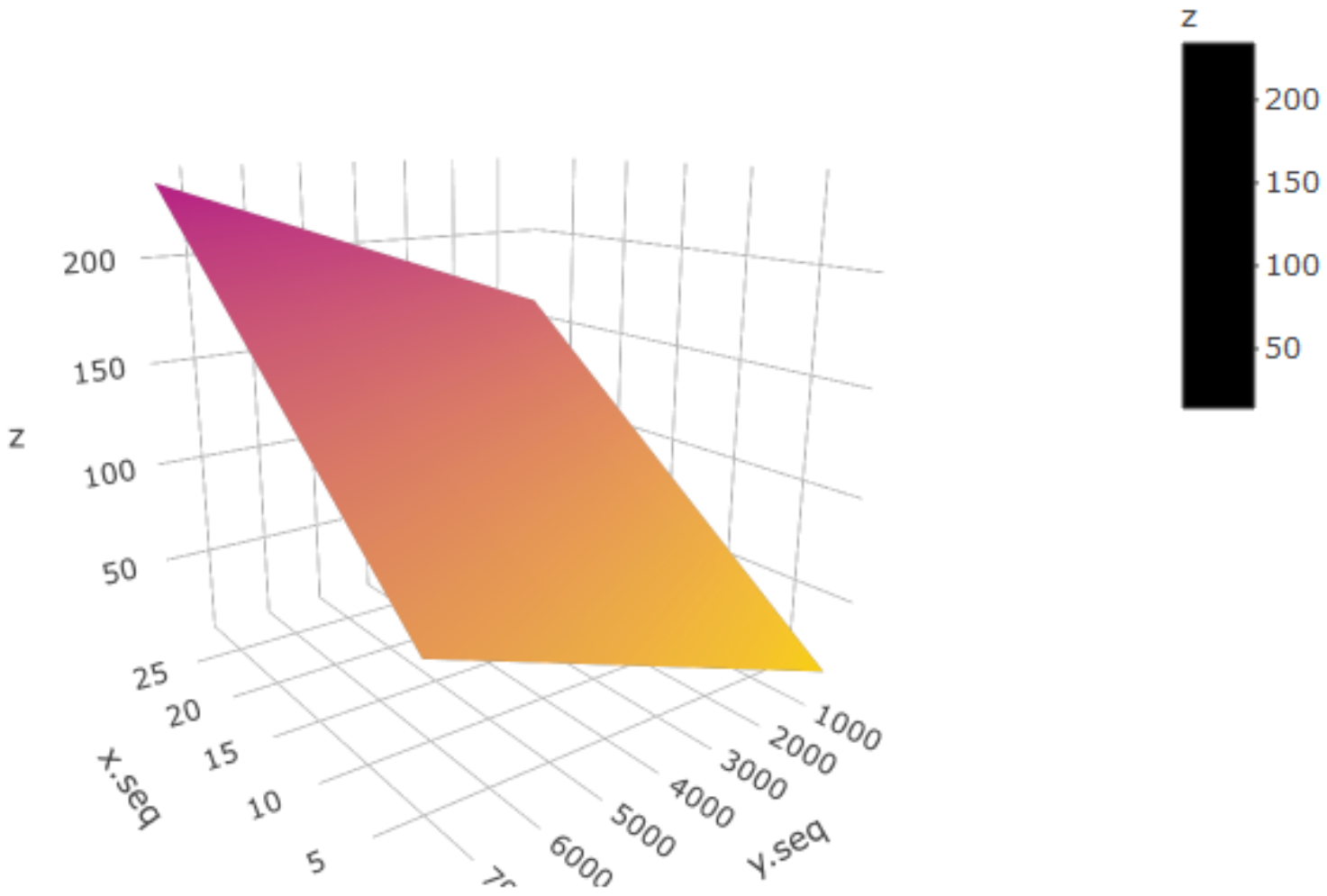
3d-Plot Regressionsebene

Datenpunkte für die Regressionsebene erstellen und surface-Plot erstellen:

```
mod.coef <- coef(model3); mod.coef
attach(myhills)
  x.seq <- seq(min(dist),max(dist),length.out=25)
  y.seq <- seq(min(climb),max(climb),length.out=25)
  z <- t(outer(x.seq, y.seq,
      function(x,y) 0+mod.coef[1]*x+mod.coef[2]*y))
detach(myhills)
p <- plot_ly(x=~x.seq, y=~y.seq, z=~z,
  colors = c("#f5cb11", "#b31d83"),type="surface")
p
```

3d-Plot Regressionsebene

Datenpunkte für die Regressionsebene erstellen und `surface`-Plot erstellen:



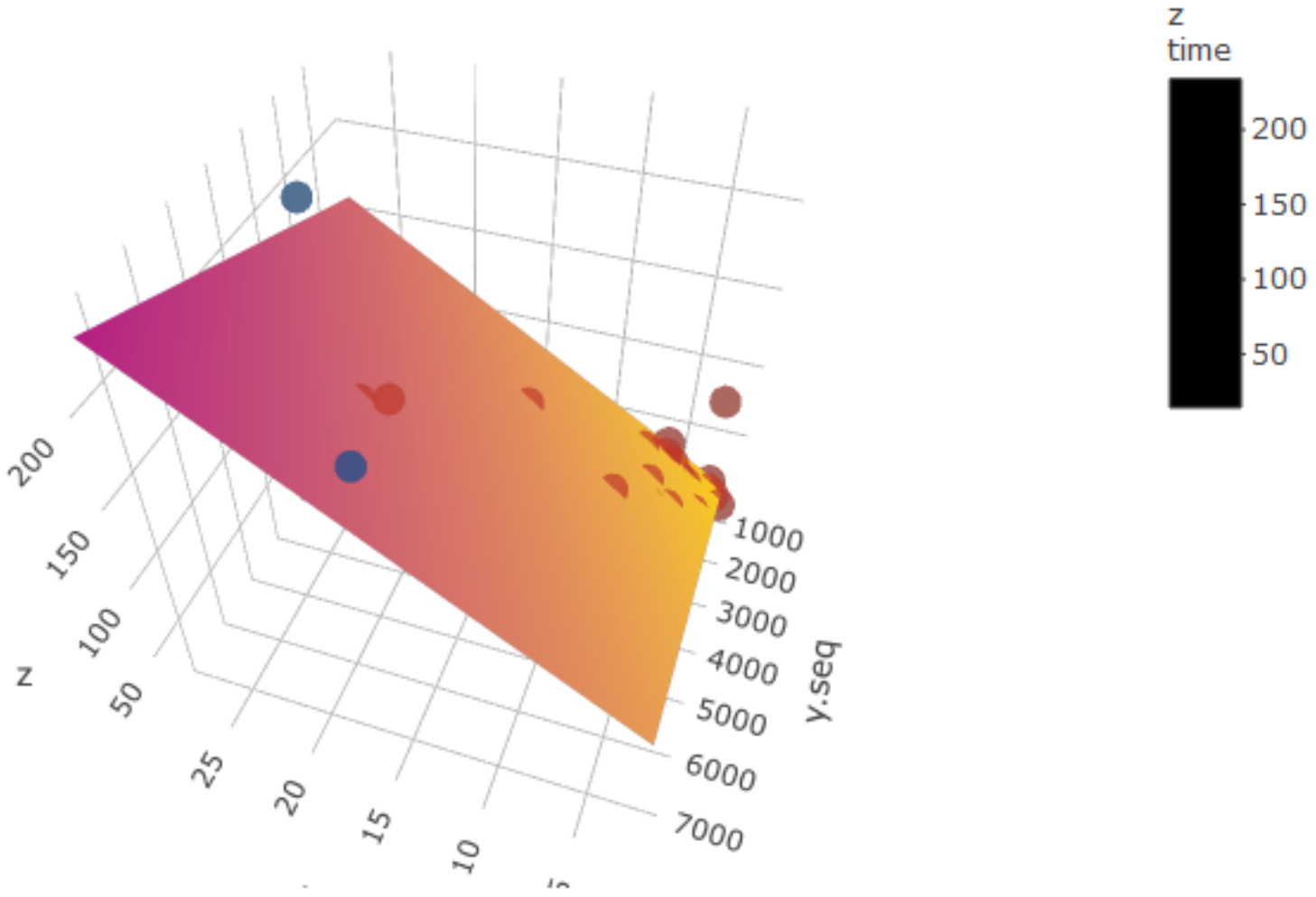
Regressionsebene + 3D-Punkte

Datenpunkte dem Plot hinzufügen und Plot ausgeben:

```
# Vektor der Farbindikatoren für "Outlier"
out.col <- c('#BF382A', '#0C4B8E')
out.col <- out.col[1+(myhills$isOutlier==T)]
p <- p %>%
  add_trace(data=myhills,
            x = ~dist, y = ~climb, z = ~time,
            text= myhills$hill, mode="markers",
            type="scatter3d",
            marker = list(color = out.col,
                          opacity=0.7,
                          symbol=105))
```

Regressionsebene + 3D-Punkte

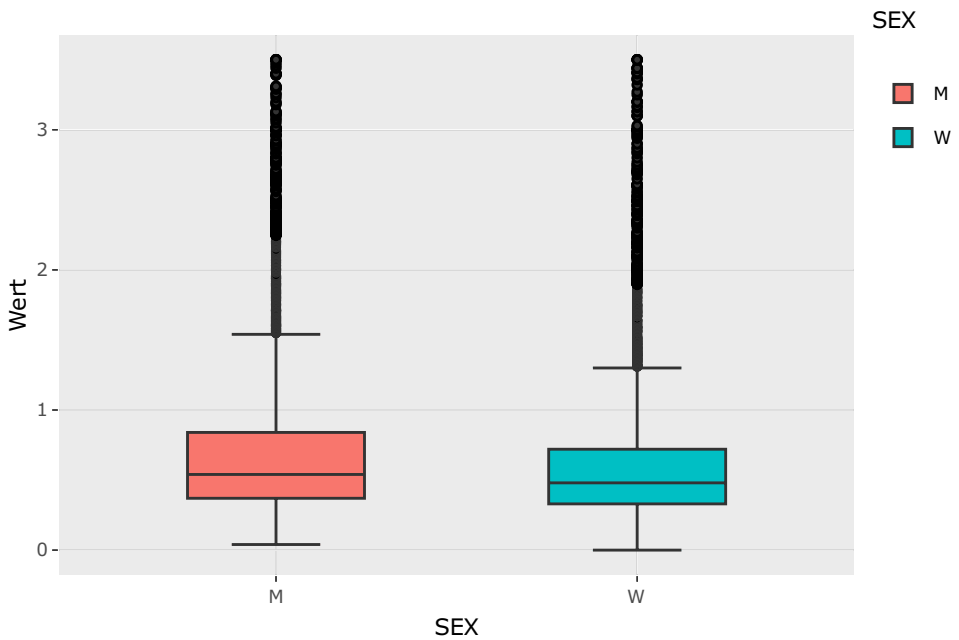
Datenpunkte dem Plot hinzufügen und Plot ausgeben:



ggplot2-Plot in plotly umwandeln

ggplotly wandelt ggplot2 in plotly-Grafiken um:

```
bilirubin <- read.table(file="Daten/bilirubin.txt",
                        head=TRUE, sep=";")
library(ggplot2); library(plotly)
p <- ggplot(bilirubin, aes(x=SEX, y=Wert, fill=SEX)) +
  geom_boxplot();
ggplotly(p) # wandelt ggplot2 in plotly um
```



Inspiration zum Plotten

- Plotly R
  https://plot.ly/r/basic-charts/
- R Graph Gallery (Hauptsächlich ggplot2)
  https://www.r-graph-gallery.com/
- Top 50 ggplot2 Visualizations http://r-statistics.co/
  Top50-Ggplot2-Visualizations-MasterList-R-Code.htlm
- nicht ganz ernst zu nehmen, die Chernoff-Gesichter:
  https://rdrr.io/cran/DescTools/man/PlotFaces.html

Shiny Apps

Was Ihr damit machen könnt: https://shiny.rstudio.com/gallery/

Es gibt Beispiele zum sofortigen Ausprobieren:

```
library(shiny)
runExample("01_hello")
```

Welche Beispiele es sonst noch gibt, seht ihr mit:

```
runExample()
```

Grundgerüst

Erstellt folgende Datei `app.R` (Dateiname wichtig) in einem neuen Ordner, den Ihr ausschließlich für dieses Shiny-App benutzt:

```
library(shiny)
# Platz für statischen Code, z.B. das Laden von Daten
ui <- fluidPage("Hier stehen Seiteninhalte:",
                "mehrere Objekte...",
                "können übergeben werden.")
server <- function(input, output) {
  # Hierhin kommt Code, der den Input verarbeitet
  # um Output zu erzeugen
  }
shinyApp(ui = ui, server = server) #startet die App
# hier darf kein Code mehr stehen
```

Grundgerüst bearbeiten

Aufgabe: Schreibt Folgendes als Argument in die Funktion `fluidPage`:

```
h6("R-Kurs", align = "center"),
h6("Letzter Termin", align = "center"),
h5("Wir haben gelernt:", align = "center"),
h4("Statistische Auswertungen.", align = "center"),
h3("Datenhandling mit ", em("base"),
   " und ", em("tidy"), align = "center"),
h2("Makrdown/knitR", align = "center"),
h1("und ",strong("Shiny."))
```

sidebar-Layout

Starte mit einem neuen Template. Die Oberfläche soll so definiert werden:

```
ui <- fluidPage( titlePanel("NBA Statistics 09/10"),
                 sidebarLayout(
                   sidebarPanel(
                   ),
                   mainPanel(
                     tableOutput("results")
                     )
                   )
                 )
```

`sidebarLayout` erzeugt eine schmale Fläche links und eine große Fläche rechts. Lasse die App einmal laufen.

Daten laden

Füge Folgendes an einer geeigneten Stelle für statischen Code ein:

```
library(tidyverse)
library(SportsAnalytics)
data(NBAPlayerStatistics0910)
nbadat <- as_tibble(NBAPlayerStatistics0910)
teams <- levels(nbadat$Team)
positions <- levels(nbadat$Position)
mplayed <- quantile(nbadat$TotalMinutesPlayed,
                    probs=c(0,1,.25,.75))
```

Installiere das benötigte Package.

League	Name	Team	Position	GamesPlayed	TotalMinutesPlayed	FieldGoalsMade	FieldGo
NBA	Arron Afflalo	DEN	SG	82	2228	272	
NBA	Alexis Ajinca	CHA	C	6	29	5	
NBA	Lamarcu Aldridge	POR	PF	78	2919	579	
NBA	Joe Alexander	CHI	SG	8	30	1	
NBA	Malik Allen	DEN	PF	51	458	46	
NBA	Ray Allen	BOS	SG	80	2812	464	

Bedienelemente

Schreibe Folgendes innerhalb des sidebarPanel():

```
sliderInput("mplayedInput", "gespielte Minuten",
            mplayed[1], mplayed[2],
            mplayed[3:4], pre = "$"),
radioButtons("posInput", "Position",
             choices = c("beliebig", positions),
             selected = "beliebig"),
selectInput("teamInput", "Team",
            choices = c("beliebig",teams))
```

Überprüfe ob die App soweit läuft und versuche zu verstehen, was du getan hast.

Filtern

Basierend auf der Auswahl wollen wir die Daten wie folgt filtern:

```
filtered <- reactive({
  nbadat %>%
  select(Name, Team, Position, TotalMinutesPlayed,
      FieldGoalsMade, PersonalFouls, TotalPoints) %>%
  filter(TotalMinutesPlayed >= input$mplayedInput[1],
      TotalMinutesPlayed <= input$mplayedInput[2],
      Position==input$posInput|input$posInput=="beliebig",
      Team==input$teamInput|input$teamInput=="beliebig"
  )
})
```

Füge den Code ein zwischen

`server <- function(input, output) { und }`

Output

Direkt hinter das eben Eingefügte ergänze

```
  output$results <- renderTable({
    filtered()
  })
```

Plot Objekte

erzeugt man ganz ähnlich wie Tabellen: ergänze den `mainPanel` um Objekte für *normale* Plots und `plotly`-Output:

```
mainPanel(
  plotOutput("myplot"),
  br(), br(),
  plotlyOutput('plot3d'),
  br(), br(),
  tableOutput("results")
    )))
```

Plots erzeugen

Damit die neuen Platzhalter befüllt werden, ergänze die Definition der Funktion `server` um folgende Zeilen:

```
output$myplot <- renderPlot({
  ggplot(filtered(), aes(FieldGoalsMade)) +
    geom_histogram()
})

output$plot3d <- renderPlotly({
  plot_ly(filtered(), x = ~TotalMinutesPlayed,
          y = ~PersonalFouls, z = ~TotalPoints,
          color = ~Position,
          colors = "Set1",
          type = "scatter3d", mode='markers',
          text= ~Name)
})
```

Selbstständig lernen

Falls du Zeit hast selbst weiter zu lernen:

- Shiny tutorials:https://shiny.rstudio.com/tutorial/
- Machine Learning: http://www.sthda.com/english/web/5-bookadvisor/54-machine-learning-essentials/
- Allgemeine Themen: Starte das interaktive Lernprogramm aus dem Package `swirl`

```
library("swirl")
swirl()
```

- Eigenes Package erstellen: https://support.rstudio.com/hc/en-us/articles/200486488-Developing-Packages-with-RStudio