R-Codes aller Übungen

Romina Behrend

Streudiagramm mit Regressionslinie

```
library(tidyverse)
library(haven)
df_votes = read_dta("aggregat2.dta")
# zuerst NAs filtern
df_votes = df_votes %>% filter(!is.na(srf2ja))

ggplot(df_votes, aes(x = nationalrat, y = acceptpo)) +
  geom_point() +
  scale_x_continuous(limits = c(0, 100)) +
  scale_y_continuous(limits = c(0, 100)) +
  theme_minimal() +
  xlab("Schlussabstimmung Nationalrat (Verhältnis Ja/Nein-Stimmen ohne Enthaltungen)") +
  ylab("Anteil Ja-Stimmen an der Urne") +
  geom_smooth(method = "lm", se = FALSE)
```

Beta 1: von Hand berechnen (Beta 0, Beta 1)

```
# NAs entfernen
df_votes = df_votes %>% filter(!is.na(UV) & !is.na(AV))

# Kovarianz
cov_xy = sum((df_votes$AV - mean(df_votes$AV)) *
               (df_votes$UV - mean(df_votes$UV))) / (nrow(df_votes)-1)
cov(df_votes$AV, df_votes$UV)

# Varianz
var_x = sum((df_votes$UV - mean(df_votes$UV))^2) / (nrow(df_votes)-1)
var(df_votes$UV)

# beta1
beta1 = cov_xy / var_x
beta1

# lineare Regression schätzen (Kontrolle)
model = lm(acceptpo ~ nationalrat, data = df_votes)
model
```

Beta 0: von Hand berechnen

```
# beta0
beta0 = mean(df_votes$acceptpo) - beta1 * mean(df_votes$nationalrat)
beta0

# lineare Regression schätzen (Kontrolle)
model = lm(acceptpo ~ nationalrat, data = df_votes)
model
```

s^2 & Standardfehler berechnen (für Beta 0, Beta 1)

```
# calculate residuals (e = y - yhat) (y: acceptpo, yhat: beta0 + beta1
# nationalrat)
e = df_votes$acceptpo - (beta0 + beta1 * df_votes$nationalrat)

# s^2
s2 = sum(e^2) / (nrow(df_votes) - 2)

# s.e. von beta0
se_beta0 = sqrt(s2 * (1/nrow(df_votes) +
    mean(df_votes$nationalrat)^2 /
    sum((df_votes$nationalrat - mean(df_votes$nationalrat))^2)))

# s.e. von beta1
se_beta1 = sqrt(s2 / sum((df_votes$nationalrat - mean(df_votes$nationalrat))^2))
```

lineare Regression rechnen (ohne Gewicht)

```
model1 = lm(acceptpo ~ nationalrat, data = df_votes)
summary(model1)
# wenn p-Wert unter (unserem gesetzten) Alpha (von 0.05), dann ist Effekt
# signifikant -> H0 verwerfen, HA annehmen [** als Orientierung]
```

lineare Regression rechnen (mit Gewicht)

```
# für den Datensatz
library(devtools)
devtools::install_github("benjaminschlegel/schlegel")

# mit Gewicht
model2 = lm(lr_self ~ age, data = schlegel::selects2015,
            weights = weight_total)
summary(model2)
```

Hypothesentests

```
# alpha-Wert von 0.05 (= 0.025 auf beiden Seiten)

# get betas
beta = coef(model1)

# get s.e.
se = sqrt(diag(vcov(model1)))

# 95% confidence interval
lower = beta + se * qt(0.025, summary(model1)$df[2])
upper = beta + se * qt(0.025, summary(model1)$df[2], lower.tail = FALSE)
cbind(lower, beta, upper)
summary(model)
# wenn innerhalb KI, dann H0 annehmen (kein Effekt, nicht signifikant),
# wenn ausserhalb KI, dann H0 ablehnen (Effekt, signifikant)

# Konfidenzinterval direkt berechnen
confint(model1)
```

Multivariate Regression

```
model = lm(acceptpo ~ suppshare2 + swjalager, data = df_votes)
summary(model)
# Effekte immer interpretieren mit ceteris-paribus!
```

Multivariate Regression: Beta 1 von Hand ausrechnen

```
# NAs entfernen, um von Hand rechnen zu können (damit wir gleich viele
# Residuals haben dann! Sonst können wir u1~v1 nicht rechnen! Aka am Anfang
# hier immer alle Variablen, die wir brauchen, einmal auf NAs filtern, damit
# wir ganz sicher bei allen die gleiche Fallzahl haben [da z.B. suppshare2
# nicht im model_y_x2 enthalten ist, werden da auch nicht ihre Fälle mit NAs
# herausgenommen, was die Fallzahl dann tendenziell anders ausfallen lassen
# kann verglichen zu model_x1_x2])
df_votes = df_votes %>% filter(!is.na(suppshare2) & !is.na(swjalager))
# alle Variablen, die im Modell sind
model_y_x2 = lm(acceptpo ~ swjalager, data = df_votes)
u1 = residuals(model_y_x2)

model_x1_x2 = lm(suppshare2 ~ swjalager, data = df_votes)
v1 = residuals(model_x1_x2)

summary(lm(u1 ~ v1))
```

Multivariate Regression: Beta 2 von Hand ausrechnen

```
model_y_x2 = lm(acceptpo ~ suppshare2, data = df_votes)
u2 = residuals(model_y_x2)

model_x1_x2 = lm(swjalager ~ suppshare2, data = df_votes)
v2 = residuals(model_x1_x2)

summary(lm(u2 ~ v2))
```

Variabeln Wichtigkeit: Level Importance (Durchschnittlicher Effekt)

```
coef(model)[-1] * colMeans(model.frame(model))[-1]
# [-1]: Entfernung von beta_0 und der AV
# je grösser, desto wichtiger ist Variable
```

Variablen Wichtigkeit: maximale Änderung

```
coef(model)[-1] * diff(apply(model.frame(model), 2, range))[-1]
# je grösser, desto wichtiger ist Variable
```

Variablen Wichtigkeit: Dispersion Importance (Standardisierte Koeffizienten)

```
library(QuantPsyc)
lm.beta(model)
# je grösser, desto wichtiger ist Variable
```

Variablen Wichtigkeit: Anteil erklärter Varianz R^2

```
library(relaimpo)
calc.relimp(model, type = "lmg") # Interpretation mit Prozentpunkten
calc.relimp(model, type = "lmg", rela = TRUE) # Interpretation mit Prozenten
# je grösser, desto wichtiger ist Variable
```

Modellevaluation: Varianzanalyse: R^2 von Hand berechnen (Multiple R-squared im summary(lm) Output)

```
v = anova(model)
SST = sum(v$`Sum Sq`)
SSR = sum(v$`Sum Sq`[-nrow(v)])
SSE = sum(v$`Sum Sq`[nrow(v)])

# R2 berechnen
R2 = SSR / SST
R2
# je grösser, desto besser (1: perfekt, 0: kein Zsmhang; kann nicht negativ
# werden)
# UV erklärt x% der Varianz der AV
```

Modelevaluation: angepasstes R2 von Hand berechnen (adjusted R-squared im summary(lm) Output)

```
v = anova(model)
MSE = v$`Mean Sq`[nrow(v)] # das nrow gibt Anzahl Zeilen zurück, aka sagen wir,
# dass wir den Wert in der letzten Zeile haben wollen (weil die letzte Zeile
# ja auch die Anzahl aller Zeilen ist)
MST = SST / sum(v$Df)
adjR2 = 1 - MSE / MST
adjR2
# je grösser, desto besser? (1: perfekt?, kann auch negativ werden)
```

Modelevaluation: RMSE von Hand berechnen (Residual Standard Error im summary(lm) Output)

```
v = anova(model)
MSE = v$`Mean Sq`[nrow(v)]
RMSE = sqrt(MSE)
RMSE
# RMSE: Fehler -> je kleiner, desto besser!
```

Modellevaluation: AIC / BIC

```
# WICHTIG: ZUERST ABCHECKEN, OB DIE FALLZAHLEN GLEICH SIND BEI BEIDEN MODELLEN
nobs(model1)
nobs(model2)
```

```r
# wichtig, hier alle NAs von allen Variablen  entfernen, weil
# wir gleiche Fallzahl brauchen, um zu vergleichen! Sonst haben wir andere
# AIC/BIC-Werte und können das alles nicht vergleichen
df_votes = df_votes %>% filter(!is.na(suppshare2) & !is.na(swjalager) &
                                 !is.na(srf2ja))
model1 = lm(acceptpo ~ suppshare2 + swjalager, data = df_votes)
model2 = lm(acceptpo ~ suppshare2 + srf2ja, data = df_votes)
# diese zwei Modelle wollen wir vergleichen, welches besser ist

AIC(model1, model2)
BIC(model1, model2)
# unterscheiden sich selten, daher egal, welche Methode verwendet wird
# NUR AIC und BIC untereinander vergleichen, nicht Werte von AIC zu
# Werten von BIC vergleichen!
```

Quadratisches Modell (nicht-monoton, u-förmig [keine konsistente Steigung])

```r
df_votes$knappheit = 1 - abs(df_votes$acceptpo - 50) / 50 # unnötig
model_quadrat = lm(partpo ~ swanzahl + I(swanzahl^2) + knappheit, data = df_votes)
texreg::screenreg(model_quadrat)
```

quadratisches Modell (andere Variante, haben wir nicht benutzt)

```r
model_quadrat = lm(partpo ~ swanzahl + poly(swanzahl, 2) + knappheit,
                   data = df_votes)
# Vorteil: ist orthogonal (wird speziell transformiert),
# führt dazu, dass Korrelation zw. swananzahl^2 & swananzahl kein Problem mehr,
# Nachteil: schwierigere Interpretation
```

Quadratisches Modell: Grafik

```r
ggplot(df_votes, aes(x = swanzahl, y = partpo)) +
  geom_point() +
  geom_smooth(method = "lm", formula = y ~ x + I(x^2)) +
  theme_minimal()
```

Discrete Change berechnen

```r
# von 1 bis 2
betas = coef(model_quadrat)[2:3]
```

```r
x_1 = c(1, 1^2)
x_2 = c(2, 2^2)
betas %*% x_2 - betas %*% x_1
```

Funktion: Discrete Change

```r
dc = function(v1, v2){
  betas = coef(model_quadrat)[2:3]
  x_1 = c(v1, v1^2)
  x_2 = c(v2, v2^2)
  betas %*% x_2 - betas %*% x_1
}
dc(1, 2) # Discrete Change von 1 zu 2, jeweilige Werte hier einfach eingeben
```

Beispiel: Marginale Effekte von swanzahl (Steigung)

```r
# abhängig von der jeweiligen Ableitung!
# Ableitung von 52.63 - 5.384 * swanzahl + 0.502 * swanzahl^2 + 6.582 *
# knappheit
# auf swanzahl ist: -5.384 + 2 * 0.502 * swanzahl
# aka, zuerst Gleichung ableiten!

# deswegen ist marginaler Effekt bei (X=) 1:
coef(model_quadrat)[2] + 2 * coef(model_quadrat)[3] * 1

# für andere Werte von X dann halt jeweils einsetzen (hier müssten wir
# ganz rechts den Wert ändern, der Rest bleibt)
```

Marginale Effekte: Funktion

```r
library(margins)
marginal_effects(model_quadrat,
                 data = data.frame(swanzahl = 1:9, knappheit = 0.7))
# Knappheit kann irgendetwas sein
# hier kriegen wir dann die marginalen Effekte für Werte 1 bis 9
# Zur Erinnerung: Beim linearen Modell sind marginale Effekte für alle Werte
# gleich
```

Grafik: linlin (X: normal, Y: normal)

```
# WICHTIG: immer bei allen verwendeten Variablen zuerst die NAs entfernen bevor
# wir plotten!

world = world %>% filter(!is.na(pcgdp) & !is.na(homicide))
linlin = ggplot(world, aes(x = pcgdp, y = homicide)) +
  geom_point() + theme_bw() +
  geom_smooth(method = "lm") +
  ggtitle("lin-lin: y ~ x")
linlin
```

Grafik: loglin (X: normal, Y: log)

```
loglin = ggplot(world, aes(x = pcgdp, y = log(homicide))) +
  geom_point() + theme_bw() +
  geom_smooth(method = "lm") +
  ggtitle("log-lin: log(y) ~ x")
loglin
```

Grafik: linlog (X: log, Y: normal)

```
linlog = ggplot(world, aes(x = log(pcgdp), y = homicide)) +
  geom_point() + theme_bw() +
  geom_smooth(method = "lm") +
  ggtitle("lin-log: y ~ log(x)")
linlog
```

Grafik: loglog (X: log, Y: log)

```
loglog = ggplot(world, aes(x = log(pcgdp), y = log(homicide))) +
  geom_point() + theme_bw() +
  geom_smooth(method = "lm") +
  ggtitle("log-log: log(y) ~ log(x)")
loglog
```

alle Grafiken zusammen darstellen lassen

```
library(ggpubr)
ggarrange(linlin, linlog, loglin, loglog, ncol = 2, nrow = 2)
```

linlin Modell (X: normal, Y: normal)

```
m_linlin = lm(homicide ~ pcgdp, data = world)
```

linlog Modell (X: log, Y: normal)

```
m_linlog = lm(homicide ~ log(pcgdp), data = world)
```

loglin Modell (X: normal, Y: log)

```
m_loglin = lm(log(homicide) ~ pcgdp, data = world)
```

loglog Modell (X: log, Y: log)

```
m_loglog = lm(log(homicide) ~ log(pcgdp), data = world)
```

alle Modelle zusammen darstellen

```
library(texreg)
screenreg(list(m_linlin, m_linlog, m_loglin, m_loglog),
          custom.model.names = c("homecide", "homecide",
                                 "log(homecide)", "log(homecide)"))
```

aus kategorieller Variable eine Dummy Variable machen

```
df_votes = df_votes %>% mutate(
  rechtsform = as_factor(formec2),  # beachte, das formec2 ist im Datensatz!
  initiative = recode(rechtsform, "I." = "Initiative",
                      .default = "keine Initiative")
)
```

lm-Modell mit binärer Variable

```
model = lm(acceptpo ~ initiative + nationalrat,
           data = df_votes)
summary(model)
```

Grafik: lm-Modell mit binärer UV

```
ggplot(df_votes, aes(x = nationalrat, y = acceptpo,
                     col = initiative)) +
  geom_point() + theme_bw() +
  geom_smooth(method="lm", fullrange=TRUE, se = FALSE)
```

lm-Modell mit diskreter UV (mehrere Ausprägungen)

```
model = lm(acceptpo ~ rechtsform + nationalrat,
           data = df_votes)
summary(model)
```

lm-Modell mit diskreter UV: andere Kategorien untereinander vergleichen

```
library(car)
linearHypothesis(model, "rechtsformI. - rechtsformObl.")
```

lm-Modell mit diskreter UV: andere Kategorien untereinander vergleichen, indem die Basiskategorie geändert wird

```
df_votes$rechtsform = relevel(df_votes$rechtsform, ref = "Obl.")
# Obl. = gewünschte Basiskategorie
```

Interaktion: diskrete Variablen

```
model = lm(acceptpo ~ initiative * fromSVP, data = df_votes)
summary(model)
# ODER:
model = lm(acceptpo ~ initiative * fromSVP + nationalrat, data = df_votes)
summary(model)
```

Interaktion: diskrete Variablen - Grafik

```
library(sjPlot)
plot_model(model, type = "int") + theme_minimal() +
  scale_color_manual(values = c("blue", "red"), name="Urheber") +
  xlab("") +
  scale_y_continuous(labels = scales::percent_format(scale=1))
```

Interaktion: Hypothesen testen (diskrete Variablen)

```
library(car)
linearHypothesis(model, "(Intercept) +
                 initiativeInitiative +
                 initiativeInitiative:fromSVPSVP
                 + fromSVP")
# hier testen wir, ob der vorausgesagte Wert (Achsenabschnitt für eine
# Initiative der SVP) signifikant ist
linearHypothesis(model, "initiativeInitiative +
                 initiativeInitiative:fromSVPSVP")
# hier testen wir, ob der Effekt der Initiative, wenn sie von der SVP kommt,
# einen signifikantn Effekt (auf acceptpo) hat
```

Interaktion: kontinuierliche Variablen

```
model = lm(acceptpo ~ importance * suppshare2 + nationalrat, data = df_votes)
summary(model)
```

Interaktion: kontinuierliche Variablen - Grafik

```
plot_model(model, type = "int") + theme_minimal() +
  scale_color_manual(values = c("blue", "red"),
                     name="Anteil Ja-Propaganda") +
  scale_fill_manual(values = c("blue", "red"),
                     name="Anteil Ja-Propaganda") +
  scale_y_continuous(labels = scales::percent_format(scale=1)) +
  xlab("Nationale Bedeutung Vorlage")
```

Interaktion: kontinuierliche Variablen - Hypothesen testen

```
linearHypothesis(model, "100*suppshare2 + 900*importance:suppshare2")
linearHypothesis(model, "100*suppshare2 + 400*importance:suppshare2")
# da die UVs nicht nur die Werte 0/1 annehmen können, wählen wir hier
# gezielt die für uns interessanten Werte der UVs aus (100/9 oder 100/4)
```

Multikollinearität: bivariate Kollinearitäten/Korrelationen herausfinden mit Pearson's R

```
myvars <- c("nationalrat", "swjalager", "difficul") # entsprechende
# Variablen angeben
df_model = na.exclude(df_votes[myvars])
# NAs entfernen (müssen wir NUR machen, wenn die angegebenen Variablen
# nicht in einem lm-Modell gebündelt sind)
cor(df_model$nationalrat, df_model$swjalager)
cor(df_model)

# wenn alle Variablen bereits in einem Modell gespeichert sind,
# können wir das auch so schreiben:
df_model = model.frame(model)[,-1]
cor(df_model$nationalrat, df_model$swjalager)
cor(df_model) # hier rechnet man es gleich für gesamte Matrix (alle Variablen
# untereinander korreliert)

# wenn absoluter (!) Wert > 0.8, dann Multikollinearitätsproblem
```

Multikollinearität: bivariate Kollinearitäten/Korrelationen herausfinden mit Pearson's R: Grafik

```
library(PerformanceAnalytics)
chart.Correlation(df_model, histogram=TRUE, method = "pearson")
summary(lm(acceptpo ~ nationalrat + difficul, data = df_votes))
```

Multikollinearität: Kollinearität herausfinden mit VIF

```
library(car)
vif(model)
vif(lm(acceptpo ~ nationalrat + difficul, data = df_votes))
sqrt(vif(model)) # gibt an, um wie viel sich der Standardfehler von Beta der
# jeweiligen Variablen bei Multikollinearität  verändert, wenn sie nicht
# mehr mit den anderen Variablen im Modell korreliert wäre
# -> um so viel ist Standardfehler von Beta grösser bei Multikollinearität,
# als wenn er unkorreliert (mit den anderen Prädiktoren im Modell) wäre
```

Einfluss: Hebelwirkung berechnen (mit Plot!)

```
hat = hatvalues(model)
hat_bar = mean(hat)
df_hebel = data.frame(h = hat, index = names(hat))
```

```r
ggplot(df_hebel, aes(as.numeric(index), h)) +
  geom_point() +
  geom_hline(yintercept = 2 * hat_bar,
             linetype = "dashed") +
  geom_hline(yintercept = 3 * hat_bar,
             linetype = "dashed") +
  theme_minimal()
# hier sehen wir dann die Punkte, die 2x oder 3x über den Durchschnittswert
# aller hatvalues liegen
```

Einfluss: Ausreisser identifizieren: intern studentisierte Residuen

```r
r = rstandard(model)
```

Einfluss: Ausreisser identifizieren: extern studentisierte Residuen

```r
rstudent(model)
```

Einfluss: Ausreisser identifizieren

```r
df_model = model.frame(model)
df_model$h = hat
df_model$r = r
df_model$t = t
df_model %>% mutate(
  grosse_Hebelwirkung2 = h > 2 * hat_bar,
  grosse_Hebelwirkung3 = h > 3 * hat_bar,
  multivariater_Ausreisser = abs(r) > 3,
  moeglicher_Problemfall = grosse_Hebelwirkung2 & multivariater_Ausreisser
) %>% filter(moeglicher_Problemfall)
# dieser Output sollte uns angeben, ob (und welche?) wir Problemfälle haben
# in den Daten
```

Einfluss: problematische Werte berechnen & anzeigen lassen

```r
sum(hatvalues(model) > 2 * mean(hatvalues(model)) & abs(rstudent(model)) > 3)
# zeigt uns an, wie viele Werte problematisch sind (hohe Hebelwirkung +
# Ausreisser) [am liebsten hätten wir hier = 0]

df_votes[which(hatvalues(model) > 2 * mean(hatvalues(model)) & abs(rstudent(model)) > 3),]
# um herauszufinden, welcher Wert genau problematisch ist
```

Einfluss: einflussreiche Beobachtungen identifizieren: DFFITS

```
sum(dffits(model)>1) # oder > 2
# zeigt uns Summe aller einflussreichen Beobachtungen im Datensatz an
```

Einfluss: einflussreiche Beobachtungen identifizieren: Cooks Distance

```
sum(cooks.distance(model)>1)
# zeigt uns Summe aller einflussreichen Beobachtungen im Datensatz an
```

Normalität: Normalität überprüfen (qqPlot)

```
library(car)
qqPlot(model)
# wenn viele Punkte ausserhalb des blauen Bandes, dann nicht normalverteilt
```

Normalität: Normalität überprüfen (Histogramm)

```
# alles zusammen ausführen lassen!

x = rstudent(model)
h = hist(x, breaks=10, col="red")
xfit = seq(min(x), max(x), length = 40)
yfit = dnorm(xfit) * diff(h$mids[1:2]) * length(x)
lines(xfit, yfit, col = "blue", lwd=2)

# zeichnet automatisch Histogramm + blaue Linie, muss nur Model angeben
# wenn blaue Linie + Histogramm kaum identisch sind, dann nicht normalverteilt
```

Normalität: Zusatz: Normalität überprüfen (prettier)

```
library(MASS)
stud_res = studres(model3)
ggplot() +
  geom_density(aes(x = stud_res), color = "red") +
  geom_line(aes(x = seq(min(stud_res), max(stud_res),
                        length.out = 50),
                y = dnorm(seq(min(stud_res), max(stud_res),
                              length.out = 50)))) +
  theme_minimal()
```

Heteroskedastizität diagnostizieren: Streudiagramm schätzen

```
data.frame(x = predict(model), y = resid(model)) %>%
  ggplot(aes(x, y)) + geom_point() + theme_minimal() +
  geom_hline(yintercept = 0, size = 1)
# x-Achse: geschätzte Y, y-Achse: Residuen
```

Heteroskedastizität diagnostizieren: Goldfisch-Quandt-Test

```
library(lmtest)
gqtest(model, order.by = ~nationalrat+swjalager+difficul, data = df_votes)
# bei order.by schreiben wir einfach unsere Formel wie im Modell
# wenn signifikant: Heterskedastizität (das ist doof)
# wenn nicht signifikant: Homoskedastizität (das wollen wir!)
```

Heteroskedastizität diagnostizieren: Breusch-Pagan-Test

```
library(lmtest)
bptest(model)
# bei order.by schreiben wir einfach unsere Formel wie im Modell
# wenn signifikant: Heterskedastizität
# wenn nicht signifikant: Homoskedastizität
```

Heteroskedasitizität korrigieren: Huber-White-Korrektur

```
library(sandwich)
r_vcov = vcovHC(model)  # korrigierte Varianz-Kovarianz-Matrix
r_se = sqrt(diag(r_vcov)) # Standardabweichungen
r_t = coef(model) / r_se # t-Werte ausrechnen
r_p = 2*pnorm(-abs(r_t)) # p-Werte ausrechnen
r_p # korrigierte p-Werte
# wir benutzen die BETAS aus dem ursprünglichen Output, um die jeweiligen
# Effekte auf die UV zu interpretieren.
# die p-Werte hier zeigen uns einfach an, ob die Betas aus dem
# ursprünglichen Output signifikant sind oder nicht!
```

Spezifikationsfehler: RESET Test

```
library(lmtest)
resettest(model)
# wenn signifikant, dann Fehlspezifikation
```

Fixed Effects Model

```
model = lm(lr_self ~ opinion_eu_membership + canton, data = df_selects)
summary(model)
# wichtig: Levelvariable MUSS ein Faktor sein (hier: canton), um ganz
# sicher zu gehen, kann man auch folgendes Modell benutzen:
model = lm(lr_self ~ opinion_eu_membership + as.factor(canton),
           data = df_selects)
summary(model)
```

Fixed Effects Model: alternatives Modell ohne overall Achsenabschnitt

```
model = lm(lr_self ~ canton - 1 + opinion_eu_membership, data = df_selects)
summary(model)
# hier sollten nun für alle Kantone der Achsenabschnitt direkt angegeben sein
# wenn man die Achsenabschnitte der Meinung zur EU angegeben sein sollen, dann
# folgender Code:
model = lm(lr_self ~ opinion_eu_membership + canton - 1, data = df_selects)
summary(model)
# ich glaube, dass Faktorvariable immer relativ weit vorne stehen muss,
# irgendwie halt hinter der anderen Variable oder so. (Bzw. wahrsch. bestimmt
# Position, wie wir sie hier ins Modell angeben dann die Position im Output
# daher ist es zu raten, die Level2-Variable eher am Anfang zu positionieren)
```

Intraclass Correlation (ICC)

```
library(lme4)
model = lmer(lr_self ~ 1 + (1 | canton), data = df_selects) # WICHTIG
# HIER IST KEINE ERKLÄRENDE VARIABLE, KEINE UV DRIN!
summary(model)
Intercept Variance / (Intercept Variance + Residual Variance)
```

Intraclass Correlation (ICC): Funktion

```
icc = function(model){
  sigma_alpha = unlist(VarCorr(model)[1])
  sigma_epsilon = attr(VarCorr(model), "sc")^2
  sigma_alpha / (sigma_alpha + sigma_epsilon)
}
icc(model)
# Lösung ist Anteil der Varianz der AV, der durch Level 2 erklärt wird (i guess)
```

Random Intercept Modell

```
library(lme4)
model = lmer(lr_self ~ age + gender + (1 | canton), data = df_selects)
summary(model)
```

Random Intercept Modell: Fixed Effects darstellen

```
library(lme4)
model = lmer(lr_self ~ age + gender + (1 | canton), data = df_selects)
library(texreg)
screenreg(model)
# zeigt die durchschn. Effekte unabhängig vom Level 2 (glaube ich)
```

Random Intercept Modell: alpha_j anzeigen

```
library(lme4)
model = lmer(lr_self ~ age + gender + (1 | canton), data = df_selects)
ranef(model)
```

Random Intercept Modell: Grafik: Fixed Effects

```
library(sjPlot)
library(tidyverse)
# optional: library(glmmTMB)
plot_model(model, type = "est") + theme_minimal()
```

Random Intercept Modell: Grafik: Random Intercepts (aka Random Effects) darstellen

```
library(sjPlot)
library(tidyverse)
# optional: library(glmmTMB)
plot_model(model, type = "re", sort.est = "(Intercept)") +
  theme_minimal()
```

Random Intercept Modell: Grafik: Annahmen

```r
library(sjPlot)
library(tidyverse)
# optional: library(glmmTMB)
plot_model(model, type = "diag")
```

Random Intercept Modell: *vorausgesagter* Wert berechnen

```r
# Vorgehen:
# 1. Ganze Gleichung des Modells aufschreiben
# BEISPIEL: AV = beta_0j + beta_1 * age + beta_2 * female (+ Epsilonj)
# BEISPIEL KANN MAN AUCH SO SCHREIBEN:
# *(gamma_00 + alpha_j) + beta_1 * age + beta_2 * female*
# 2. Werte für UVs in Gleichung schreiben.
# BEISPIEL: gamma_00 + alpha_ZH + beta_1 * 23 + beta_2 * 1
# 3. Die jeweiligen Werte holen.

fixef(model) # hier können wir die Koeffizienten für Gamma und die Betas ablesen
# Gamma = (Intercept), Betas jeweils die Koeffizienten bei den jeweiligen UVs
# Alpha_j ist der Random Effekt dieser spezifischen Ausprägung des Levels 2.
# Alpha_j können wir daher aus ranef(model) ablesen.
ranef(model) # Alpha_j ablesen
# jetzt alle Werte zsmrechnen

# ODER: ALLES SO ANWÄHLEN:
fixef(model)[1] + ranef(model)[[1]][1,] +
  fixef(model)[2] * x1 + fixef(model)[3] * x2 # [x] ist für Beta 1 etc.
# bei ranef in[x,] gewünschte Zeile anwählen
# immer 1 + 1, dann 2 + 1 etc...

# BEISPIEL: Berechnung mit Skalarprodukt
fixef(model_random) %*% c(1, 10, 5, 28) + ranef(model_random)$cntry["CH",]

# BEISPIEL: Berechnung ohne Skalarprodukt
fixef(model_random)[1] + 10 * fixef(model_random)[2]
+ 5 * fixef(model_random)[3]
+ 28 * fixef(model_random)[4] + ranef(model_random)$cntry["CH",]
```

Random Intercept Modell: *Durchschnittlicher* Wert berechnen

```r
# Vorgehen gleich wie oben, *HIER OHNE ALPHA_J*:
# 1. Ganze Gleichung des Modells aufschreiben
# BEISPIEL: *(gamma_00 + beta_1 * age + beta_2 * female*
# 2. Werte für UVs in Gleichung schreiben.
# BEISPIEL: gamma_00 + beta_1 * 23 + beta_2 * 1
# 3. Die jeweiligen Werte holen.

fixef(model) # hier können wir die Koeffizienten für Gamma und die Betas ablesen
```

```r
# Gamma = (Intercept), Betas jeweils die Koeffizienten bei den jeweiligen UVs
# jetzt alle Werte zsmrechnen

# ODER: ALLES SO ANWÄHLEN:
fixef(model)[1] +  fixef(model)[2] * 23 + fixef(model)[3] * 1
# [x] ist für Beta 1 etc. immer 1 + 1, dann 2 + 1 etc...

# ODER: Beispiel:
fixef(model_random) %*% c(1, 3, 0, 25)
```

Autokorrelation

```r
library(haven)
library(tidyverse)
library(lmtest)
dwtest(model)
# H0: keine Autokorrelation
# H1: Autokorrelation
# wenn nicht signifikant, keine Autokorrelation
# wenn signifikant, Autokorrelation
```

Autokorrelation korrigieren (New-Wey-West-Methode)

```r
library(sandwich)
r_vcov = NeweyWest(model)
r_se = sqrt(diag(r_vcov))
r_t = coef(model) / r_se
r_p = 2*pnorm(-abs(r_t))
m = cbind(coef(model), r_se, r_t, r_p)
m
```

Autoregressives Modell erster Stufe (AR1): Modell schätzen

```r
# AR1
ar1 = lm(AV ~ lag(AV), data = df_parteistaerke)
summary(ar1)

# ODER:

library(tseries)
ar1_ = arma(df_parteistaerke$FDP, order = c(1, 0))
summary(ar1_)
```

Autoregressives Modell zweiter Stufe (AR2): Modell schätzen

```
ar2 = lm(FDP ~ lag(FDP) + lag(FDP, 2), data = df_parteistaerke)
summary(ar2)

# ODER:

library(tseries)
ar2_ = arma(df_parteistaerke$FDP, order = c(2, 0))
summary(ar2_)
```

Autoregressives Modell: schätzen mit weiterer UV (x)

```
ar1_eco = lm(FDP ~ kofbarometer + lag(FDP), data = df_parteistaerke2)
summary(ar1_eco)
```

kumulative Querschnittsdaten: Model (normal)

```
model = lm(UV ~ AV1 + AV2 + as.factor(year), data = df_selects,
           weights = weighttot) # evt. auch nicht mit Gewichten
summary(model)
```

kumulative Querschnittsdaten: Model (mit Interaktion)

```
model = lm(lr1 ~ age * as.factor(year) + sex , data = df_selects,
           weights = weighttot) # evt. auch nicht mit Gewichten
summary(model)
```

Paneldaten im Longformat

```
pivot_longer(df, selection, values_to = "name", names_to = "name")
# WICHTIG! DAS BRAUCHT MAN EIG. WIE IMMER FÜR PANELDATEN!
```

Pooled/Pooling Effects Panel Modell

```
library(plm)
model_pooling = plm(lr_self ~ sex + edyear + political_interest,
                        data=df_shp, index = c("idpers","year"),
```

```
                      model = "pooling")
summary(model_pooling)
```

First Difference Panel Modell

```
library(plm)
model_fd = plm(lr_self ~ sex + edyear + political_interest,
                    data=df_shp, index = c("idpers","year"),
                    model = "fd")
summary(model_fd)
```

Fixed Effects Panel Modell (within): Unterschied für EIN Individuum über Zeit

```
library(plm)
model_fixed = plm(lr_self ~ sex + edyear + political_interest,
               data=df_shp, index = c("idpers","year"),
               model = "within")
summary(model_fixed)
```

Random Effects Panel Modell

```
library(plm)
model_random = plm(lr_self ~ sex + edyear + political_interest,
                    data=df_shp, index = c("idpers","year"),
                    model = "random")
summary(model_random)
```

Random Effects Panel Modell: Intercepts einzelner Fälle betrachten

```
library(plm)
model_random = plm(lr_self ~ sex + edyear + political_interest,
                    data=df_shp, index = c("idpers","year"),
                    model = "random")
summary(model_random)
ranef(model_random)
```

Hausman-Test (auf Endogenität)

```
# HA: Endogenität besteht -> wenn signifikant, Fixed Effects Modell ODER
# First Difference benutzen
# HO: keine Endogenität -> Random Effects Modell benutzen
library(plm)
phtest(model_fixed, model_random)
```

Breusch-Pagan-Lagrange-Multipler Test (auf Random Effects)

```
# HO: keine Varianz zw. Einheiten
# HA: Varianz zw. Einheiten
# wenn signifkant, dann Random Effects Modell benutzen (weil Pooling Modell
# NICHT geeignet)
# wenn nicht signifikant,d ann Pooling Modell benutzen (weil Random Effects
# Modell NICHT geeignet)
library(plm)
plmtest(model_pooling, type="bp")
```

Loop für Discrete Changes (für Interpretation mit x^2)

```
for(lr in 1:10){
  cat(lr-1, "->", lr, ":",
      round(coef(model)[2:3] %*% c(lr, lr^2) - coef(model)[2:3] %*% c(lr-1,
                                                    (lr-1)^2), digits = 3),
      "\n")
}
```

Logit-Modell (logistisches Modell)

```
library(tidyverse)
library(haven)
model_logit = glm(AV ~ UV1 + UV2 + UV3,
                  data = df_voto, family = binomial)
summary(model_logit)
```

Probit-Modell

```
library(tidyverse)
library(haven)
model_probit = glm(participation ~ polint + age + sex,
```

```
                  data = df_voto, family = binomial(link=probit))
summary(model_probit)
```

Logit- & Probit-Modell: Ergebnisse zusammen darstellen

```
texreg::screenreg(list(model_logit, model_probit),
                  custom.model.names = c("Logit", "Probit"))
```

Logit-Modell (logistisches Modell): Interpretation mit Odds Ratio

```
exp(coef(model_logit))
# dann jeweils den Koeffizienten, der auch im Modell oben signifikant war,
# interpretieren
# am besten rechnet man noch (Wert, der hier ausgegeben wird-1)*100
# um dann die Prozent Wahrscheinlichkeit zu bekommen
(Wert, der hier ausgegeben wird-1)*100
# man vergleicht dann hier mit der Ausprägung, die vor der angegebenen
# Ausprägung kommt (aka wenn hier im Output sex2 woman steht, dann
# wird das verglichen mit sozusagen Ausprägung woman, die wahrsch. = 2 ist,
# mit 2-1 = 1 = man)
```

Logit-Modell (logistisches Modell): Interpretation mit vorausgesagter Wahrscheinlichkeit (predicted probability)

```
yhat = c(1, 0, 0, 0, 0, 23, 0) %*% coef(model_logit)
# wichtig: der Vektor (1, 0, 0 etc.) definiert sozusagen die Werte, die für
# die jeweiligen Variablen gelten soll (also von den spezifischen Fall, für
# den wir etwas voraussagen wollen, wie bspw. ein 23-jähriger Mann)
# ABER: die erste 1 MUSS IMMER STEHEN!
# zum besseren Verständnis: für Model:
# logit(P(participation = 1)) = beta0 + beta1 * agei + beta2 * genderi +
# beta3 * agei * genderi + epsiloni
# wäre: yhat = beta[1] (* 1)  + beta[2] * 20 + beta[3] * 1 + beta[4] * 20 * 1
# das kann man aber ja eben schneller schreiben, daher mit Skalarprodukt hier
p = exp(yhat) / (1 + exp(yhat))
p
# das *100 = Prozent
```

Logit-Modell (logistisches Modell): Interpretation mit vorausgesagter Wahrscheinlichkeit (predicted probability) + Discrete Change

```
# hier rechnen wir einfach zwei versch. vorausgesagte Wahrscheinlichkeiten (die
# wir berechnet haben, so wie wir das oben mit yhat etc. gemacht haben), und
# rechnen minus
p2 - p
# Ergebnis wird dann von p2-Sicht aus interpretiert: p2 ist weniger/mehr
# likely um xy zu machen als p
# ACHTUNG: HIER IST DANN DAS ERGEBNIS IN PROZENTPUNKTEN ANGEGEBEN!
```

Logit-Modell (logistisches Modell): Simulieren: Konfidenzintervalle hinzufügen (zu vorausgesagten Wahrscheinlichkeiten)

```
# zu benutzen bei VIELEN Fällen
library(MASS)
sim_betas = mvrnorm(1000, coef(model_logit), vcov(model_logit))
yhat = sim_betas %*% c(1, 25, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0)
p = exp(yhat) / (1 + exp(yhat))
mean(p)
quantile(p, probs = c(0.025, 0.975))
```

Logit-Modell (logistisches Modell): Bootstrap: Konfidenzintervalle hinzufügen (zu vorausgesagten Wahrscheinlichkeiten)

```
# kann bei jedem Datensatz benutzt werden
library(tidyverse)
library(haven)
boot = function(x, model){
  data = model.frame(model)
  sample_data = data[sample(nrow(data), nrow(data), replace = TRUE),]
  coef(update(model, data = sample_data, weights = sample_data$`(weights)`))
}

boot_betas = lapply(1:1000, boot, model = model_logit)
boot_betas = do.call("rbind", boot_betas)

# Warning message:
# In rbind(c(`(Intercept)` = 1.95840499454745, `polint2 rather interested`
# = -0.27851226684004, : number of columns of result is not a multiple of
# vector length (arg 11)
# diese Warnmeldung ist NORMAL und kann egal sein

yhat = boot_betas %*% c(1, 0, 0, 0, 0, 23, 0) # bei c(..) gewünschte Werte eingeben
```

```
quantile(yhat, probs = c(0.025, 0.975)) # Konfidenzintervall für vorausgesagter Wert
mean(yhat) # = der vorausgesagte Wert
```

Logit-Modell (logistisches Modell): Simulieren (mit Package glm.predict)

```
library(glm.predict)
predicts(model_logit, "F(1);23;F(1)", type = "simulation")
# F(1): erste Ausprägung der ersten UV (im Bsp. pol. Interesse = KATEGORIELL)
# 23: Ausprägung = 23 bei zweiter UV (im Bsp. Alter = NUMERISCH!)
# F(1): erste Ausprägung der dritten UV (im Bsp. Geschlecht)
# wenn man type nicht spezifiziert, wird entweder Bootstrap (bei unter 500
# Fällen) oder Simulation (bei über 500 Fällen) gerechnet
# im Output haben wir dann die vorausgesagten Wahrscheinlichkeiten
# für genau diese Kombination an Ausprägungen (inkl.
# Konfidenzintervall (mean, lower, upper))
```

Logit-Modell (logistisches Modell): Bootstrap (mit Package glm.predict)

```
library(glm.predict)
predicts(model_logit, "F(1);23;F(1)", type = "bootstrap")
# F(1): erste Ausprägung der ersten UV (im Bsp. pol. Interesse)
# 23: Ausprägung = 23 bei zweiter UV (im Bsp. Alter)
# F(1): erste Ausprägung der dritten UV (im Bsp. Geschlecht)
# Wenn man type nicht spezifiziert, wird entweder Bootstrap (bei unter 500
# Fällen) oder Simulation (bei über 500 Fällen) gerechnet.
# Im Output haben wir dann die vorausgesagten Wahrscheinlichkeiten
# für genau diese Kombination an Ausprägungen (inkl.
# Konfidenzintervall (mean, lower, upper)).
```

Logit-Modell (logistisches Modell): Simulieren + Discrete Changes (mit Package glm.predict)

```
library(glm.predict)
predicts(model_logit, "F(1,2);23;F(1)", type = "simulation",
         position = 1)
# hier machen wir ein Discrete Changes bei der ersten Variable
# Im Output haben wir dann jeweils val1_mean, was die vorausgesagte Wahrschein-
# lichkeit meint für die erste gewünschte Ausprägung auf der ersten Variable
# (F(1,2), im Beispiel war das für sehr Interessierte), inkl. dem KI-Intervall
# (val1_lower, val1_upper)
# Same goes für die vorausgesagte Wahrscheinlichkeit für val2_mean, das ist
```

```
# die zweite gewünschte Ausprägung auf der ersten Variable (F(1,2), bei
# der wir ja den Discrete Change ausrechnen).
# dc_mean ist dann der Unterschied beider Ausprägungen in PROZENTPUNKTEN,
# inkl. deren KI-Intervall (anscheinend signifikant, wenn BEIDE Zahlen dc_lower
# & dc_upper negativ oder positiv sind; anscheinend insignifikant, wenn
# EINE der beiden Zahlen positiv und die andere negativ ist)

# anderes Beispiel but same logic:
predicts(model_logit, "F(1);23;F", type = "simulation",
         position = 3)
# hier machen wir ein Discrete Change bei der dritten Variable (wegen
# position = 3)
```

Logit-Modell (logistisches Modell): vorausgesagte Wahrscheinlichkeiten (mit glm.predicts) - komplexere Modelle

```
library(glm.predict)
df_pred = predicts(model_logit, "F;mean;mode")
df_pred
# erstellt einen data-frame (df_pred)
# schauen uns hier alle Werte von erster UV an, dann den Mittelwert von
# UV2, und den Modus von UV3
# wenn man (wie hier) für wenige Werte die vorausgesagte Wahrscheinlichkeit
# angeben will ODER wenn man ein Remote-Fehler bekommt, kann man hinten noch
# ein doPar = FALSE einsetzen (dann rechnet R schneller anscheinend,
# but I wouldn't risk that in the exam)

df_pred = predicts(model_logit, "F;Q4;F")
df_pred
# hier schauen wir uns alle Ausprägungen von erster & dritter UV an,
# und die Quantile von zweiter UV
```

Logit-Modell (logistisches Modell): vorausgesagte Wahrscheinlichkeiten (mit glm.predicts) - komplexere Modelle: Graphisch darstellen

```
library(glm.predict)
df_pred = predicts(model_logit, "mode;18-75;F")
ggplot(df_pred, aes(x = age, y = mean, ymin = lower, ymax = upper)) +
  geom_ribbon(alpha = 0.6) + geom_line() + facet_wrap(~sex) +
  theme_minimal() + scale_y_continuous(labels = scales::percent)

# macht eine etwas grusige Grafik, daher noch ein weiterer Code,
# um die Anzahl Simulationen zu erhöhen

df_pred = predicts(model_logit, "mode;18-75;F", sim.count = 100000)
```

```r
ggplot(df_pred, aes(x = age, y = mean, ymin = lower, ymax = upper)) +
  geom_ribbon(alpha = 0.6) + geom_line() + facet_wrap(~sex) +
  theme_minimal() + scale_y_continuous(labels = scales::percent)

# hier noch graphische Darstellung, bei der eine diskrete Variable auf der
# x-Achse dargestellt ist
ggplot(df_pred, aes(x = bildung, y = mean, ymin = lower, ymax = upper)) +
  geom_pointrange() +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5)) +
  xlab("Bildung") + ylab("Vorausgesagte Wahrscheinlichkeit (Ja)") +
  ggtitle("Anschaffung neuer Kampfjets") +
  scale_y_continuous(labels = scales::percent)
```

Logit-Modell (logistisches Modell): Discrete Changes (mit glm.predict) - komplexere Modelle

```r
library(glm.predict)
df_dc = predicts(model_logit, "F;18-75;F", sim.count = 100000,
                 position = 3)
# hier mit 100'000 Simulationen statt Standardanzahl Simulationen (nicht
# unbedingt mega nötig, aber für grafische Darstellung ganz nice)
```

Logit-Modell (logistisches Modell): Discrete Changes (mit glm.predict) - komplexere Modelle: Graphisch darstellen

```r
library(glm.predict)
df_dc = predicts(model_logit, "F;18-75;F", sim.count = 100000,
                 position = 3)
ggplot(df_dc, aes(x = age, y = dc_mean, ymin = dc_lower, ymax = dc_upper)) +
  geom_ribbon(alpha = 0.6) + geom_line() + facet_wrap(~polint) +
  theme_minimal() + geom_hline(yintercept = 0, col = "red",
                               linetype = "dashed") +
  ylab("discrete change")
# Alpha = 0.6 can be whatever though
```