

Computational Statistics

Robin Baudisch

Merlin Kopfmann

Maximilian Neudert

17. Juni 2019

Inhaltsverzeichnis

1 Lineare Regression	4
1.1 A1	4
1.2 A2	5
1.3 A3	5
1.4 A4	6
1.5 A5	6
1.6 A6	11
2 Cross Validation	12
2.1 A1	12
2.2 A2	16
2.3 A3	17
3 Bootstrap	19
3.1 A1	19
3.2 A2	25
4 Shrinking Methoden	47
4.1 A1	47
4.2 A2	53
5 PCA/PLS Regression	58
5.1 A1	58
5.2 A2	62
6 Zufallszahlen	64
6.1 A1	64
6.2 A2	67
7 Monte-Carlo-Simulation	75
7.1 A1	75
7.2 A2	77
7.3 A3	80

Einführung

Aufgaben werden mit folgendem Seed bearbeitet:

```
set.seed(42)
```

Und es werden folgende Libraries benutzt:

```
usepackage = function(name) {  
  x = deparse(substitute(name))  
  if (!require(x, character.only = TRUE)) {  
    install.packages(x, dep=TRUE)  
    if(!require(x, character.only = TRUE)) stop("Package not found")  
  }  
}  
usepackage(komadown)  
usepackage(knitr)  
usepackage(ggplot2)  
usepackage(tidyr)  
usepackage(lm.beta)  
usepackage(car)  
usepackage(magrittr)  
usepackage(boot)  
usepackage(tidyverse)  
usepackage(reshape)  
usepackage(grid)  
usepackage(gridExtra)  
usepackage(glmnet)  
usepackage(ElemStatLearn)  
usepackage(pls)
```

1 Lineare Regression

1.1 A1

```
# Lade Daten
load(file='res/Donald.RData')
data <- Donald_1

# Fritte die Regression
fit <- lm(
  Trump ~ Alter + Geschlecht + Minderheit + Fremdenfeindlich + IQ,
  data = data
)

# Ergebnis
summary(fit)
#>
#> Call:
#> lm(formula = Trump ~ Alter + Geschlecht + Minderheit + Fremdenfeindlich +
#>      IQ, data = data)
#>
#> Residuals:
#>    Min      1Q   Median      3Q     Max
#> -15.6835 -4.5286 -0.0023  4.1231 13.0974
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)    
#> (Intercept) 29.73834  3.60973  8.238 9.73e-14 ***
#> Alter        0.18153  0.03049  5.954 1.91e-08 ***
#> Geschlecht   5.75572  0.99977  5.757 4.97e-08 ***
#> Minderheit  -6.57586  1.82843 -3.596 0.000443 ***
#> Fremdenfeindlich 9.34984  0.16325 57.272 < 2e-16 ***
#> IQ          -0.40135  0.03016 -13.309 < 2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 6.065 on 144 degrees of freedom
#> Multiple R-squared:  0.9713, Adjusted R-squared:  0.9703 
#> F-statistic: 973.6 on 5 and 144 DF,  p-value: < 2.2e-16
```

1.2 A2

```
# standardisiere die Parameter des Regressionsmodells
fit.beta <- lm.beta(fit)
print(fit.beta)
#>
#> Call:
#> lm(formula = Trump ~ Alter + Geschlecht + Minderheit + Fremdenfeindlich +
#>      IQ, data = data)
#>
#> Standardized Coefficients:
#>             (Intercept)          Alter        Geschlecht       Minderheit
#>             0.000000000  0.08513441  0.08190695 -0.05790182
#> Fremdenfeindlich           IQ
#>             0.91808061 -0.19120337
```

Der Parameter “Fremdenfeindlich” hat den größten Effekt auf die abhängige Variable. Je höher der Parameterwert, desto größer die Zustimmung zu Trump (in %). Der Parameter “IQ” hat einen moderaten negativen Effekt auf die Ausprägung der abhängigen Variable. Je höher der Parameterwert, desto geringer die Zustimmung zu Trump (in %). Die Parameter “Geschlecht”, “Minderheit” und “Alter” haben jeweils einen geringen Effekt auf die Ausprägung der abhängigen Variable.

1.3 A3

```
KI <- confint(object = fit, level = 0.95)
kable(KI, format = 'pandoc', align = 'c', digits = 3)
```

	2.5 %	97.5 %
(Intercept)	22.603	36.873
Alter	0.121	0.242
Geschlecht	3.780	7.732
Minderheit	-10.190	-2.962
Fremdenfeindlich	9.027	9.673
IQ	-0.461	-0.342

In der Ausgabe sehen wir die Intervallgrenzen für das 95%-Konfidenzintervall.

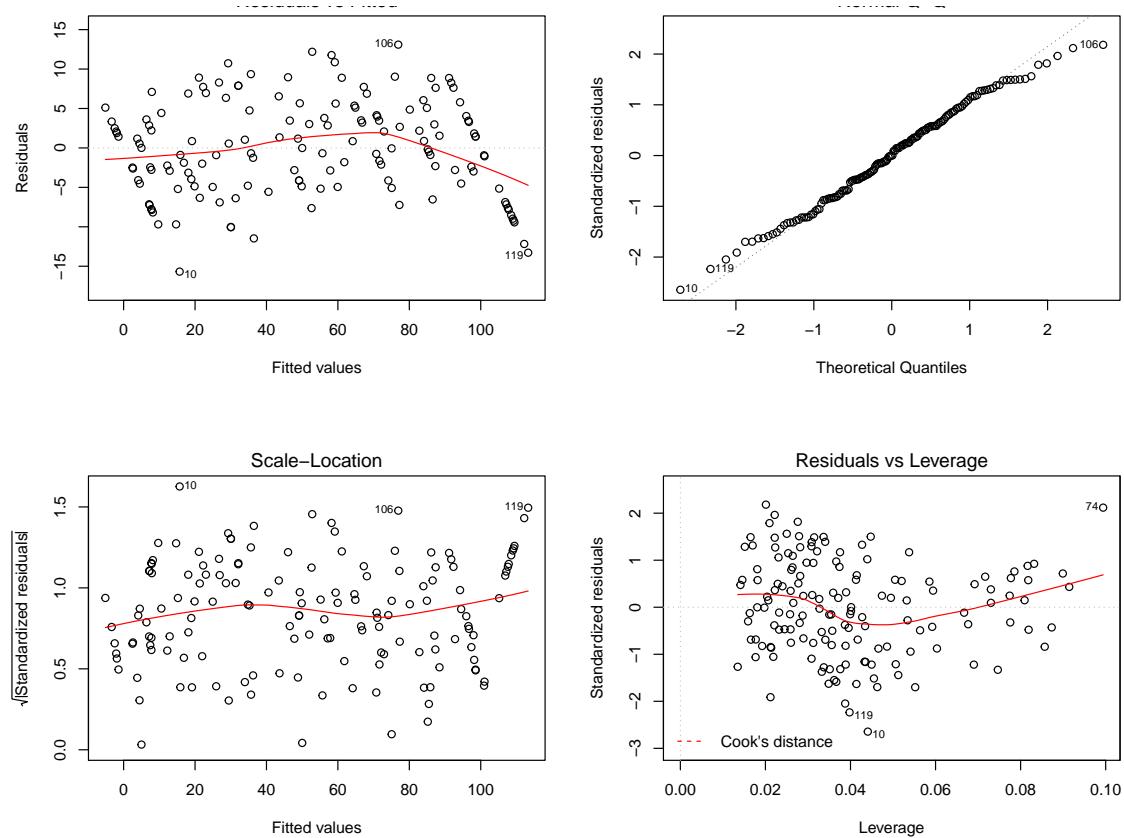
1.4 A4

```
vif_fit <- vif(mod = fit)
vif_fit
#>      Alter      Geschlecht      Minderheit Fremdenfeindlich
#> 1.024821 1.014460 1.299054 1.287851
#>          IQ
#> 1.034409
```

Die VIF-Werte liegen alle deutlich unter 5(10), es liegen also keine Hinweise für Multikollinearität zwischen den Modellparametern vor.

1.5 A5

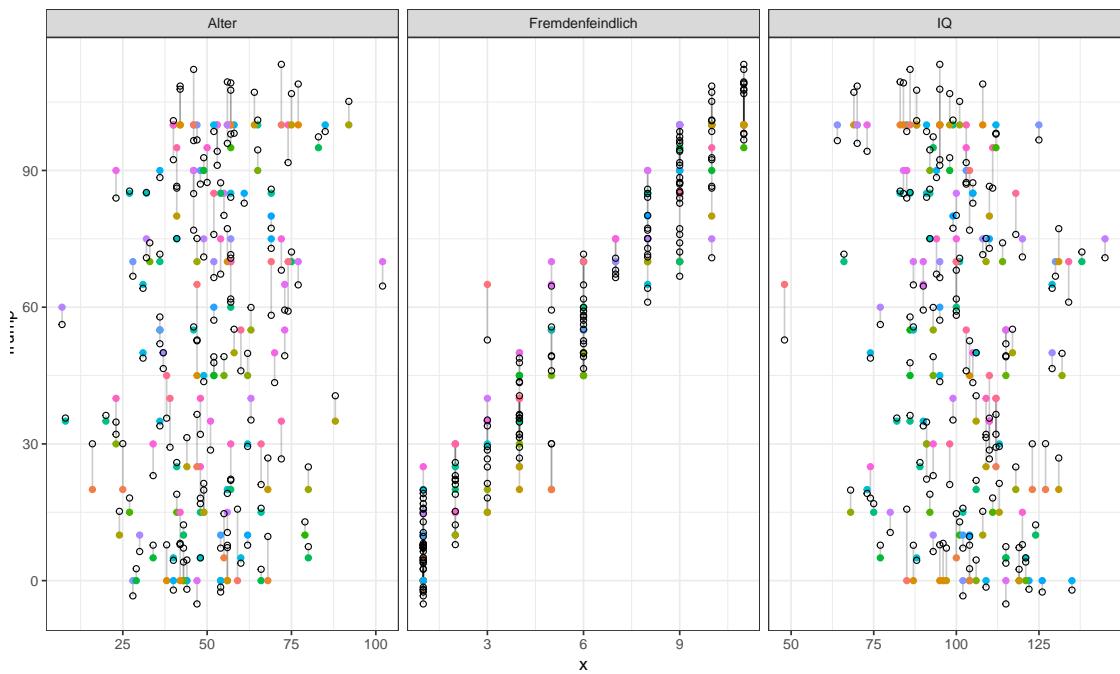
```
par(mfrow = c(2, 2))
plot(fit)
```



```

data$predicted <- predict(fit)
data$residuals <- residuals(fit)

data %>%
  gather(key = "iv", value = "x", -Trump, -predicted,
         -residuals, -Minderheit, -Geschlecht) %>%
  ggplot(aes(x = x, y = Trump)) +
  geom_segment(aes(xend = x, yend = predicted), alpha = .2) +
  geom_point(aes(color = factor(residuals))) +
  guides(color = FALSE) +
  geom_point(aes(y = predicted), shape = 1) +
  facet_grid(~ iv, scales = 'free_x') +
  theme_bw()
    
```

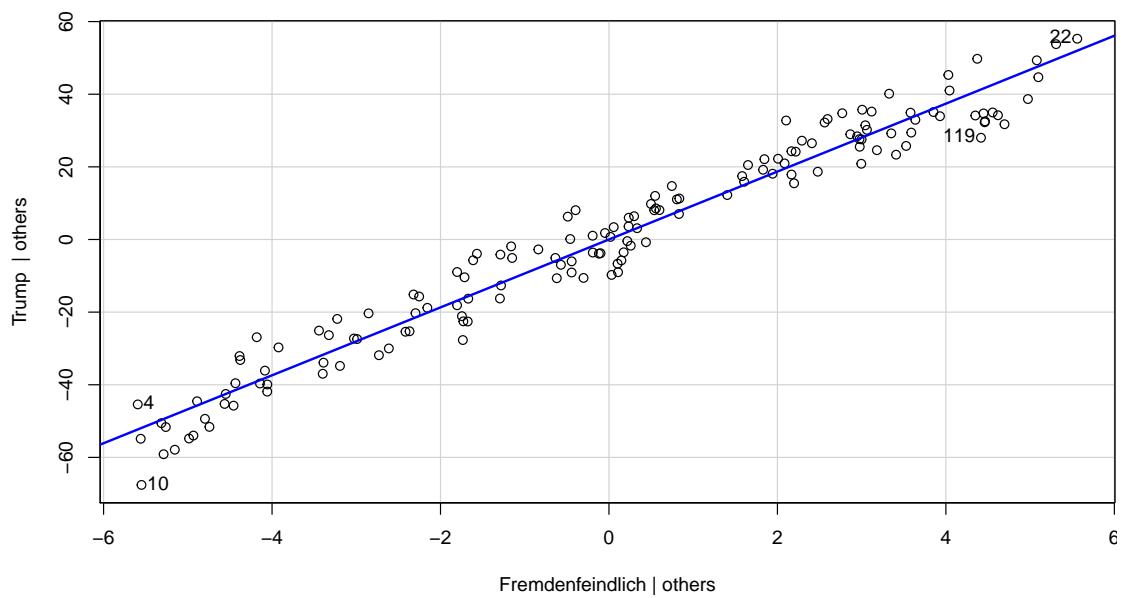


Die Residuenanalyse ergibt, dass die Residuen näherungsweise normalverteilt sind und es keine klar erkennbaren Muster in den Residuenplots gibt.

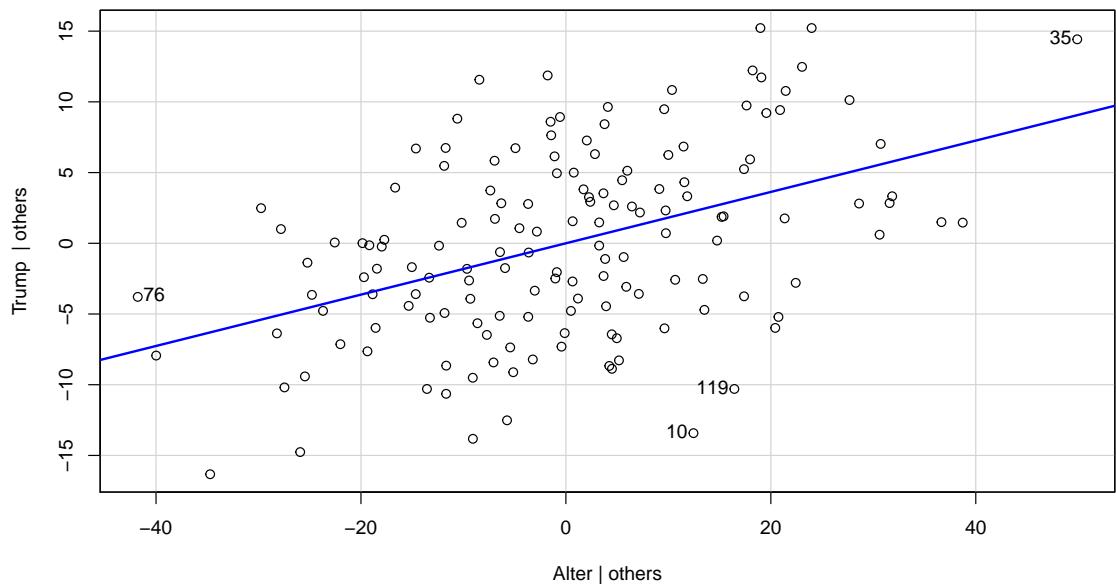
Zudem wurden die Ausprägungen der (nicht-binären) unabhängigen Parameter den Ausprägungen der abhängigen Variablen durch Scatterplots gegenübergestellt. Lediglich der Parameter “Fremdenfeindlich” weist einen klar erkennbaren linearen Zusammenhang zur abhängigen Variablen auf. Dies bestätigt das Ergebnis aus 2.

```
avPlot(fit, "Fremdenfeindlich")
```

1 Lineare Regression

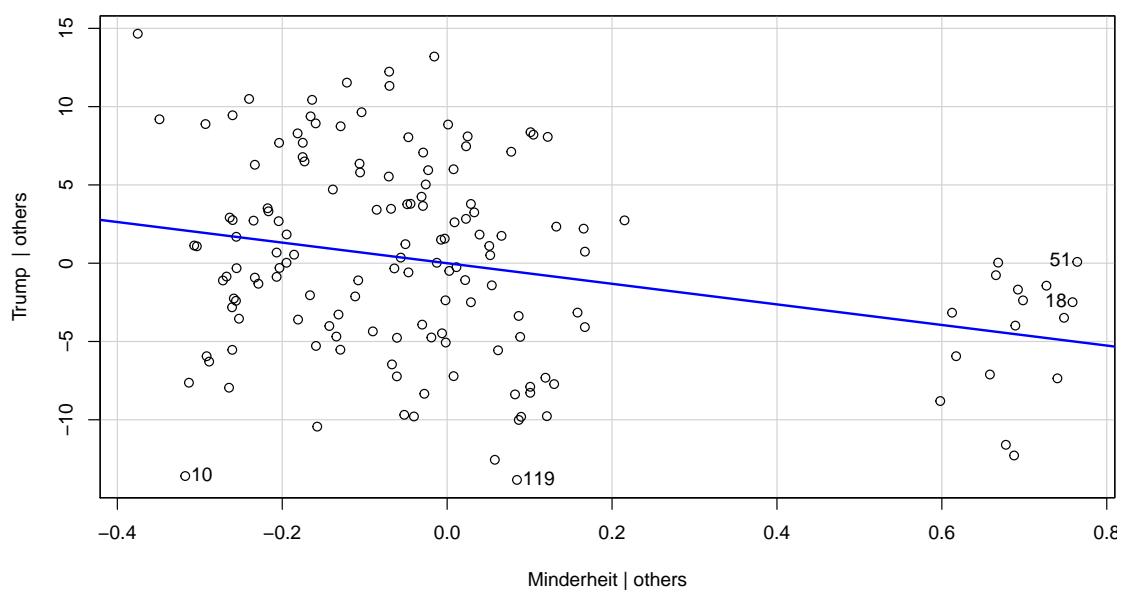


```
avPlot(fit, "Alter")
```

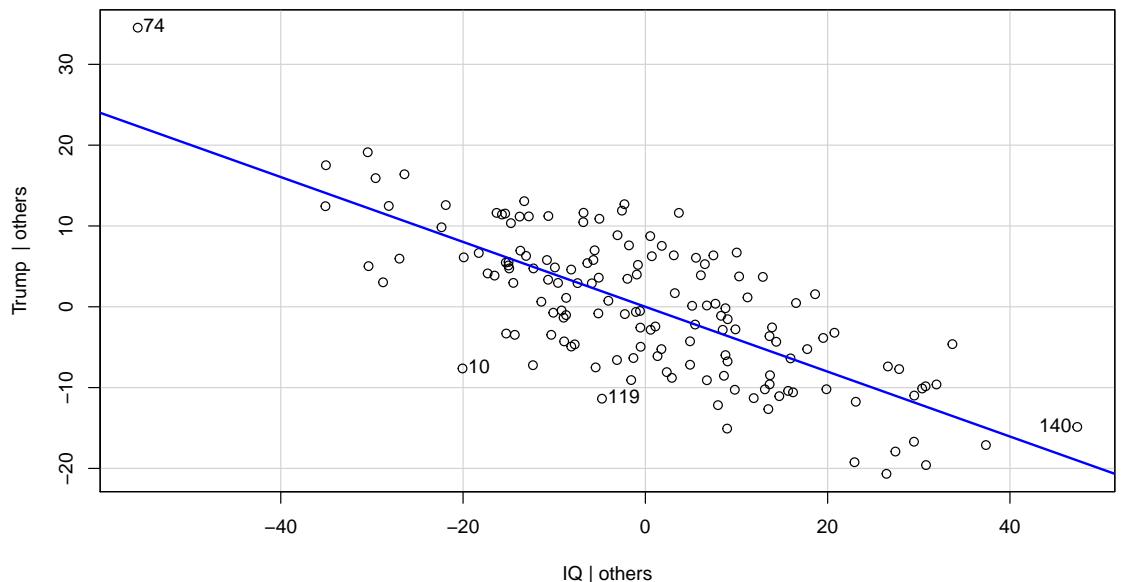


```
avPlot(fit, "Minderheit")
```

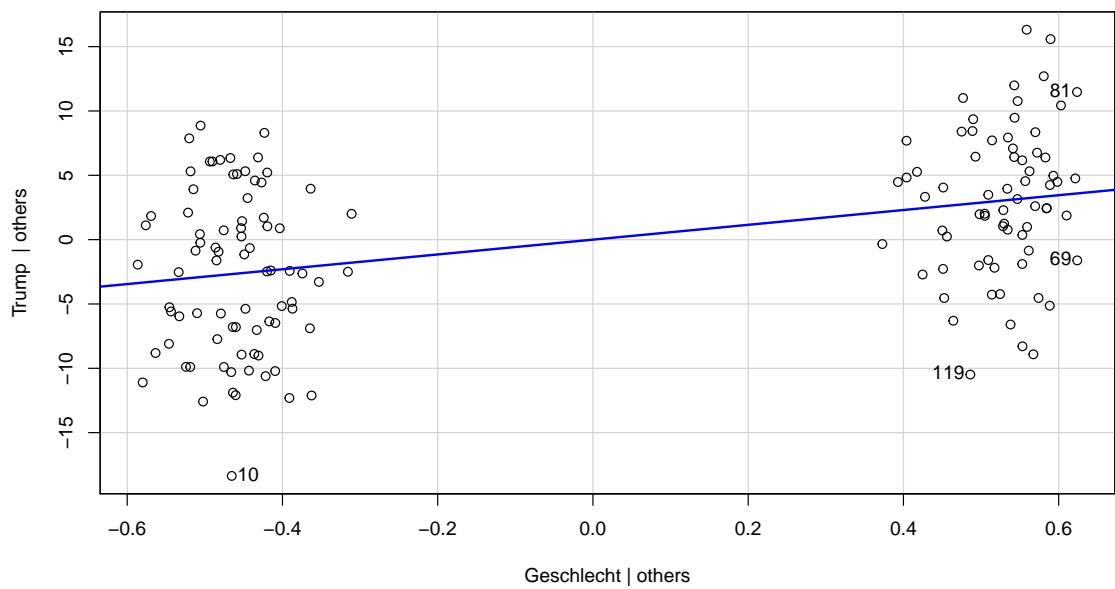
1 Lineare Regression



```
avPlot(fit, "IQ")
```



```
avPlot(fit, "Geschlecht")
```



An den Added-Variable Plot wird nochmals deutlich, dass “Fremdenfeindlich”, “IQ” und “Alter” einen linearen Trend aufweisen. Die beiden binären Variablen “Minderheit” und “Geschlecht” kann man nicht linear erklären.

1.6 A6

```
my_data <- data.frame("Geschlecht" = 1, "Alter" = 24,
                      "Minderheit" = 0, "Fremdenfeindlich" = 5, "IQ" = 100)
my_data$predicted <- predict(fit, newdata = my_data)

my_data2 <- data.frame("Geschlecht" = 1, "Alter" = 27,
                       "Minderheit" = 1, "Fremdenfeindlich" = 3, "IQ" = 100)
my_data2$predicted <- predict(fit, newdata = my_data2)

my_data3 <- data.frame("Geschlecht" = 1, "Alter" = 29,
                       "Minderheit" = 0, "Fremdenfeindlich" = 8, "IQ" = 90)
my_data3$predicted <- predict(fit, newdata = my_data3)

pred_sum <- rbind(my_data, my_data2, my_data3)
pred_sum
#>   Geschlecht Alter Minderheit Fremdenfeindlich IQ predicted
#> 1       1     24          0                  5 100 46.46492
#> 2       1     27          1                  3 100 21.73396
#> 3       1     29          0                  8  90 79.43557
```

Das Modell prognostiziert uns unterschiedliche Ergebnisse. Man erkennt klar, dass “Fremdenfeindlich” die größte Auswirkung auf die Zustimmungsrate hat.

2 Cross Validation

2.1 A1

```

load(file = "res/Donald.RData")

## Vergleich der Variablenverteilung zwischen
## Trainings- und Testdatensatz/Lineare Regression (Drei Mal)
for (i in 1:3){
  smp_size <- 100
  train_ind <- sample(seq_len(nrow(Donald_1)), size = smp_size)

  train <- Donald_1[train_ind, ]
  test <- Donald_1[-train_ind, ]
  print(i)
  cat("train: \n\n")
  print(summary(train))
  cat("\n")
  cat("test: \n\n")
  print(summary(test))
  linreg = lm(
    Trump ~ Alter + Geschlecht + Minderheit + Fremdenfeindlich + IQ,
    data = train)
  pred <- predict.lm(linreg, test)
  cat("\n")
  cat("MSE: \n")
  print(mean((test$Trump - pred) ^ 2))
  cat("----- \n\n")
}

#> [1] 1
#> train:
#>
#>   Geschlecht      Alter      Minderheit  Fremdenfeindlich
#>   Min.   :0.00   Min.   : 7.00   Min.   :0.00   Min.   : 1.00
#>   1st Qu.:0.00   1st Qu.:40.00   1st Qu.:0.00   1st Qu.: 2.00
#>   Median :0.00   Median :50.50   Median :0.00   Median : 5.00
#>   Mean   :0.43   Mean   :49.79   Mean   :0.12   Mean   : 5.35
#>   3rd Qu.:1.00   3rd Qu.:58.50   3rd Qu.:0.00   3rd Qu.: 9.00
#>   Max.   :1.00   Max.   :92.00   Max.   :1.00   Max.   :11.00
#>   IQ          Trump
#>   Min.   : 66.0   Min.   :  0.00
#>   1st Qu.: 92.0   1st Qu.: 15.00

```

```

#> Median :103.5   Median : 45.00
#> Mean   :102.9   Mean   : 49.35
#> 3rd Qu.:112.0   3rd Qu.: 81.25
#> Max.    :145.0   Max.    :100.00
#>
#> test:
#>
#>   Geschlecht      Alter      Minderheit Fremdenfeindlich
#>   Min.    :0.00    Min.    : 23.00   Min.    :0.00   Min.    : 1.00
#>   1st Qu.:0.00    1st Qu.: 42.00   1st Qu.:0.00   1st Qu.: 3.00
#>   Median  :1.00    Median  : 53.00   Median  :0.00   Median  : 6.00
#>   Mean    :0.54    Mean    : 54.14   Mean    :0.08   Mean    : 5.82
#>   3rd Qu.:1.00    3rd Qu.: 66.50   3rd Qu.:0.00   3rd Qu.: 9.00
#>   Max.    :1.00    Max.    :102.00   Max.    :1.00   Max.    :11.00
#>   IQ          Trump
#>   Min.    : 48.00  Min.    :  0.00
#>   1st Qu.: 87.25  1st Qu.: 21.25
#>   Median  : 95.00  Median  : 70.00
#>   Mean    : 96.68  Mean    : 57.40
#>   3rd Qu.:106.00  3rd Qu.: 88.75
#>   Max.    :138.00  Max.    :100.00
#>
#> MSE:
#> [1] 35.98927
#> -----
#>
#> [1] 2
#> train:
#>
#>   Geschlecht      Alter      Minderheit Fremdenfeindlich
#>   Min.    :0.00    Min.    :  7.00   Min.    :0.0    Min.    : 1.00
#>   1st Qu.:0.00    1st Qu.: 41.00   1st Qu.:0.0    1st Qu.: 2.00
#>   Median  :0.00    Median  : 52.00   Median  :0.0    Median  : 6.00
#>   Mean    :0.42    Mean    : 51.61   Mean    :0.1    Mean    : 5.58
#>   3rd Qu.:1.00    3rd Qu.: 60.50   3rd Qu.:0.0    3rd Qu.: 9.00
#>   Max.    :1.00    Max.    :102.00   Max.    :1.0    Max.    :11.00
#>   IQ          Trump
#>   Min.    : 48.00  Min.    :  0.00
#>   1st Qu.: 89.75  1st Qu.: 20.00
#>   Median  :100.00  Median  : 57.50
#>   Mean    :100.26  Mean    : 53.15
#>   3rd Qu.:111.25  3rd Qu.: 85.00
#>   Max.    :145.00  Max.    :100.00

```

```

#>
#> test:
#>
#>   Geschlecht      Alter      Minderheit Fremdenfeindlich
#>   Min.    :0.00  Min.    :16.00  Min.    :0.00  Min.    : 1.00
#>   1st Qu.:0.00  1st Qu.:41.25  1st Qu.:0.00  1st Qu.: 1.25
#>   Median   :1.00  Median   :49.50  Median   :0.00  Median   : 5.00
#>   Mean     :0.56  Mean     :50.50  Mean     :0.12  Mean     : 5.36
#>   3rd Qu.:1.00  3rd Qu.:60.75  3rd Qu.:0.00  3rd Qu.: 9.00
#>   Max.     :1.00  Max.     :85.00  Max.     :1.00  Max.     :11.00
#>   IQ        Trump
#>   Min.    : 66  Min.    :  0.00
#>   1st Qu.: 92  1st Qu.: 15.00
#>   Median   :102  Median   : 47.50
#>   Mean     :102  Mean     : 49.80
#>   3rd Qu.:112  3rd Qu.: 88.75
#>   Max.     :135  Max.     :100.00
#>
#> MSE:
#> [1] 40.43078
#> -----
#>
#> [1] 3
#> train:
#>
#>   Geschlecht      Alter      Minderheit Fremdenfeindlich
#>   Min.    :0.00  Min.    : 7.00  Min.    :0.00  Min.    : 1.00
#>   1st Qu.:0.00  1st Qu.:42.00  1st Qu.:0.00  1st Qu.: 1.00
#>   Median   :0.00  Median   :52.50  Median   :0.00  Median   : 4.50
#>   Mean     :0.46  Mean     :51.76  Mean     :0.13  Mean     : 5.02
#>   3rd Qu.:1.00  3rd Qu.:62.25  3rd Qu.:0.00  3rd Qu.: 8.00
#>   Max.     :1.00  Max.     :83.00  Max.     :1.00  Max.     :11.00
#>   IQ        Trump
#>   Min.    : 48.0  Min.    :  0.0
#>   1st Qu.: 93.0  1st Qu.: 15.0
#>   Median   :102.0  Median   : 47.5
#>   Mean     :102.4  Mean     : 47.4
#>   3rd Qu.:113.0  3rd Qu.: 75.0
#>   Max.     :135.0  Max.     :100.0
#>
#> test:
#>
#>   Geschlecht      Alter      Minderheit Fremdenfeindlich

```

```
#> Min.   :0.00  Min.   :16.0  Min.   :0.00  Min.   : 1.00  
#> 1st Qu.:0.00  1st Qu.:36.0  1st Qu.:0.00  1st Qu.: 3.25  
#> Median :0.00  Median :48.0  Median :0.00  Median : 7.00  
#> Mean    :0.48  Mean   :50.2  Mean   :0.06  Mean   : 6.48  
#> 3rd Qu.:1.00  3rd Qu.:57.0  3rd Qu.:0.00  3rd Qu.: 9.00  
#> Max.    :1.00  Max.   :102.0 Max.   :1.00  Max.   :11.00  
#>          IQ            Trump  
#> Min.   :66.0  Min.   : 0.00  
#> 1st Qu.:86.0  1st Qu.:31.25  
#> Median :94.5  Median :70.00  
#> Mean   :97.7  Mean   :61.30  
#> 3rd Qu.:108.8 3rd Qu.:95.00  
#> Max.   :145.0  Max.   :100.00  
#>  
#> MSE:  
#> [1] 43.26809  
#> -----
```

Durch die zufällige Aufteilung der Datenpunkte in zwei disjunkte Datensätze (train, test) entstehen Diskrepanzen zwischen den Verteilungen der Variablen. Diese Schwankungen wirken sich dann auch auf die Modellgüte aus. Wir wissen aus der vergangenen Übung, dass das Merkmal “Fremdenfeinlich” die Kovariate mit dem stärksten Einfluss auf die abhängige Variable ist. Vergleichen wir die Verteilung dieses Merkmals zwischen Train- und Testdatensatz in 1 mit dem aus 3, fällt auf, dass die Diskrepanzen zwischen train und test in 3 eindeutig stärker ausfallen, als in 1. Dies spiegelt sich im jeweiligen MSE wieder: Das Modell, welches mit den Daten aus 1 trainiert und getestet wurde, weist einen deutlich niedrigeren MSE auf, als das Modell aus 3.

2.2 A2

a)

```
mse <- c()
for (i in 1:nrow(Donald_1)){
  lou = lm(
    Trump ~ Alter + Geschlecht + Minderheit + Fremdenfeindlich + IQ,
    data = Donald_1[-i,])
  pred <- predict.lm(lou, Donald_1[i,])
  mse[i] <- mean((Donald_1[i,]$Trump - pred) ^ 2)
}
mse <- sum(mse)/nrow(Donald_1)
kable(mse, format = 'pandoc', align = 'c', digits = 3)
```

$$\overline{\overline{x}} \\ \underline{\underline{38.191}}$$

Der MSE für die Leave-one-out Cross-Validation liegt zwischen den berechneten MSE's aus Aufgabe 1. Dies ist zu erwarten, da wir hier untersuchen, wie gut das lineare Regressionsmodell auf "ungesehenen" Daten performt. In dem wir jeweils nur einen Datenpunkt als "Test"-Datensatz verwenden, bekommen wir so viele MSE-Werte, wie Datenpunkte im Datensatz. Diese MSE-Werte werden gemittelt, um eine "robustere" Einschätzung über die Generalisierbarkeit des Modells auf unsere Daten zu erhalten.

b)

```
model <- glm(
  Trump ~ Alter + Geschlecht + Minderheit + Fremdenfeindlich + IQ,
  data = Donald_1
)

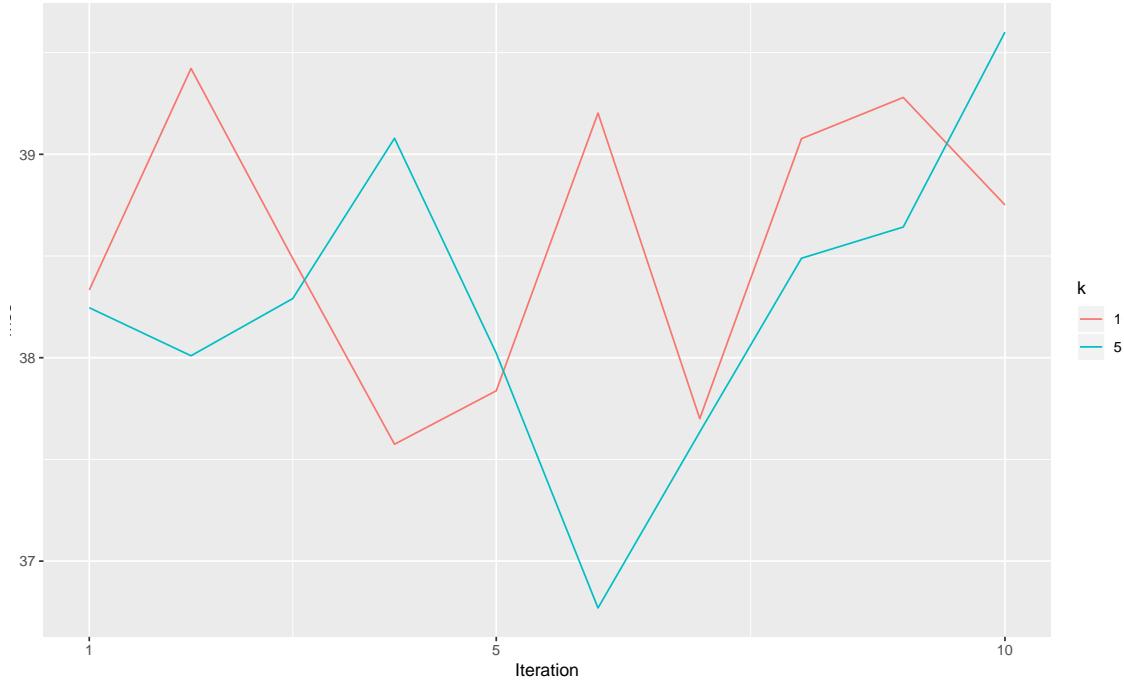
cv_model <- cv.glm(Donald_1, model, K = nrow(Donald_1))
mse1 <- cv_model$delta
kable(mse1, format = 'pandoc', align = 'c', digits = 3)
```

$$\overline{\overline{x}} \\ \underline{\underline{38.191}} \\ \underline{\underline{38.181}}$$

Die Implementierung des “Leave-one-out cross-validation”-Verfahrens durch cv.glm gibt zwei Modellgütemetriken zurück. Der erste Wert ist der durchschnittliche MSE über die Zeilen. Dieser ist identisch mit dem Wert aus a). Man kann vermuten, dass die Implementierung des “Leave-one-out cross-validation”-Verfahrens durch cv.glm identisch mit unserer manuellen Implementierung aus a) ist. Der zweite Wert ist laut Dokumentation ein Bias-korrigierter MSE.

2.3 A3

```
mse <- matrix(NA, nrow = 20, ncol = 3)
for (k in c(5,10)){
  for (i in 1:10){
    val = c(i, cv.glm(Donald_1, model, K = k)$delta[1], k)
    mse[ifelse(k == 5, i, i + 10),] = val
  }
}
mse <- data.frame(mse)
names(mse) <- c("index", "mse", "k")
mse$k <- as.character(mse$k)
gg = ggplot(
  data = mse,
  mapping = aes(
    x = index,
    y = mse,
    color = k
  )
)
gg = gg + xlab("Iteration")
gg = gg + scale_x_continuous(breaks = c(1,5,10))
gg + geom_line()
```



Bei der k -fachen Kreuzvalidierung wird die ursprüngliche Stichprobe zufällig in k gleich große Teilstichproben aufgeteilt. Von den k Teilstichproben wird eine einzige Teilstichprobe als Validierungsdaten für den Test des Modells aufbewahrt und die restlichen $k - 1$ Teilstichproben werden als Trainingsdaten verwendet. Der Kreuzvalidierungsprozess wird dann k -mal wiederholt, wobei jede der k Teilstichproben genau einmal als Validierungsdaten verwendet wird. Die k -Ergebnisse können dann gemittelt werden, um eine einzige Schätzung zu erhalten. Der Vorteil dieser Methode gegenüber des wiederholt zufälligen Subsampling besteht darin, dass alle Beobachtungen sowohl für das Training als auch für die Validierung verwendet werden und jede Beobachtung genau einmal für die Validierung verwendet wird. Die k -fache Kreuzvalidierung mit $k = 10$ wird häufig verwendet, aber im Allgemeinen bleibt k ein nicht fixierter Parameter.

Untereinander schwanken die MSE-Werte zufällig. Als Trend ist zu beobachten, dass für $k = 5$ die MSE-Werte niedriger sind. Alles in allem variieren die MSE-Werte um einen Wert von 38. Dies deckt sich mit den Ergebnissen aus Aufgabe 2. Die Schwankungen sind ebenfalls im Bereich der generierten MSE-Werte aus Aufgabe 1. Interessant ist, dass wir in Aufgabe 1 mit $\text{MSE} \sim 35$ einen, im Vergleich, relativ niedrigen MSE-Wert erzielt haben.

3 Bootstrap

3.1 A1

a)

```
means50 = matrix(NA, nrow = 50, ncol = 1)
for (i in 1:50){
  summe = runif(500,0,1)
  means50[i,] = mean(summe)
}
means1000 = matrix(NA, nrow = 1000, ncol = 1)
for (i in 1:1000){
  summe = runif(500,0,1)
  means1000[i,] = mean(summe)
}
kable(head(means1000), align = 'c', digits = 3)
```

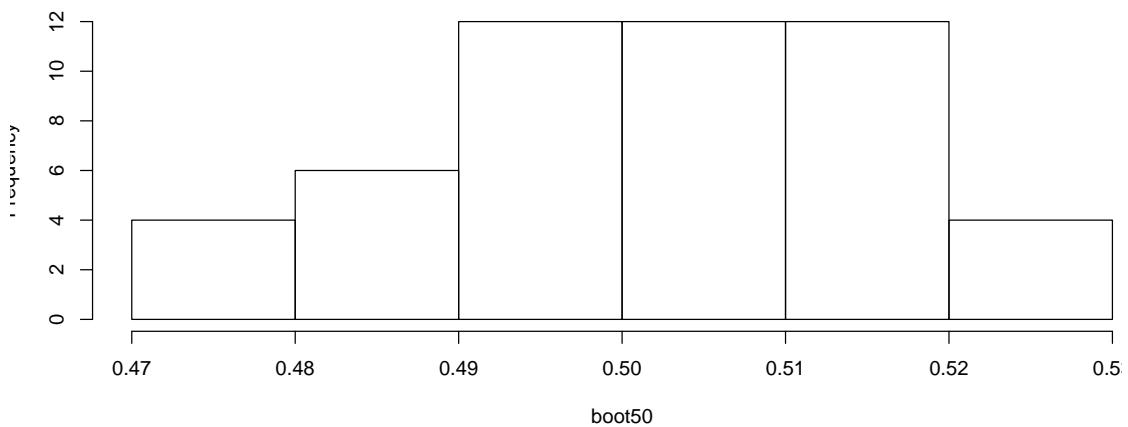
0.494
0.522
0.494
0.499
0.519
0.502

b) & c)

```
bootdata <- runif(500,0,1)
bootfunc <- function(data,i) {
  d <- data[i]
  return(mean(d))
}
boot50 <- boot(bootdata, bootfunc, R = 50)
boot50 <- boot50$t
boot1000 <- boot(bootdata, bootfunc, R = 1000)
boot1000 <- boot1000$t
hist(boot50)
```

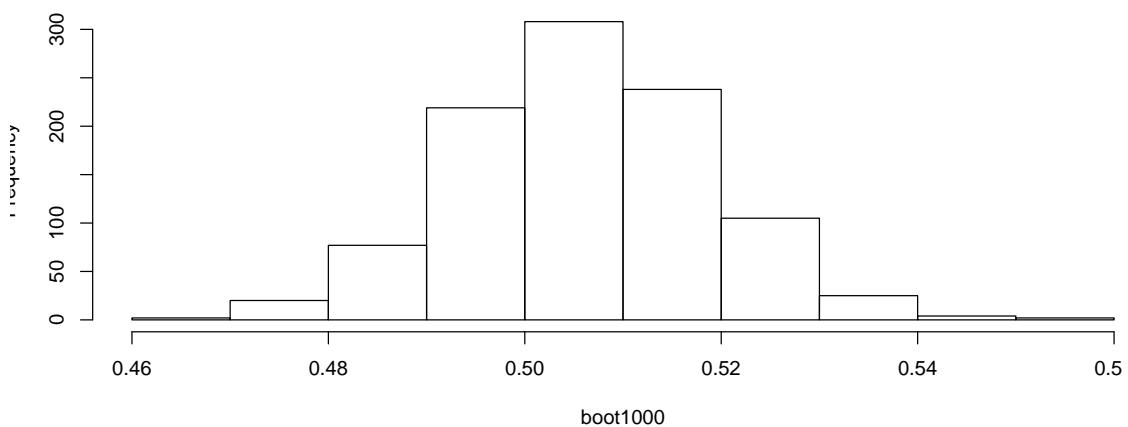
3 Bootstrap

histogram of boot50



```
hist(boot1000)
```

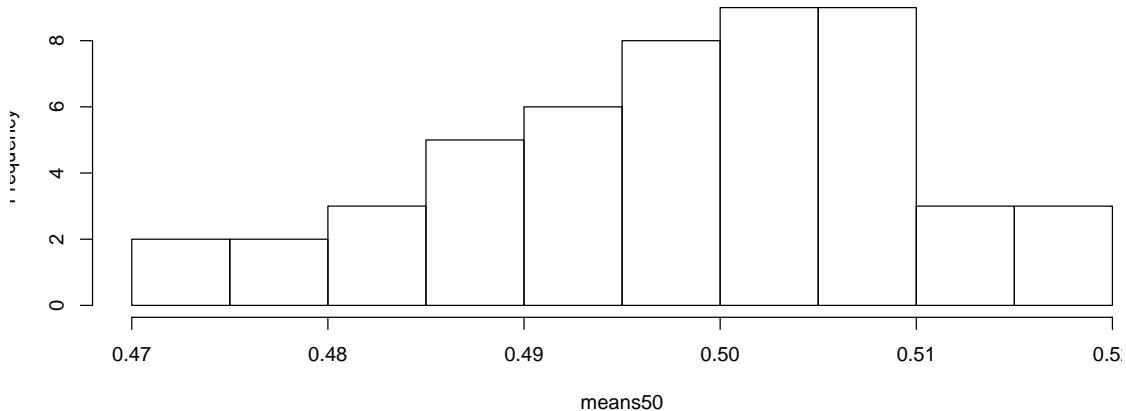
histogram of boot1000



```
hist(means50)
```

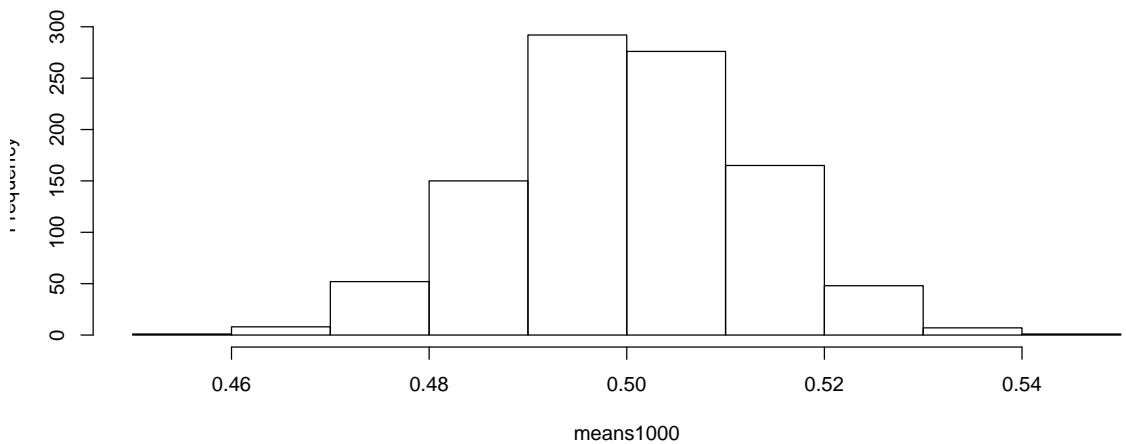
3 Bootstrap

histogram of means50



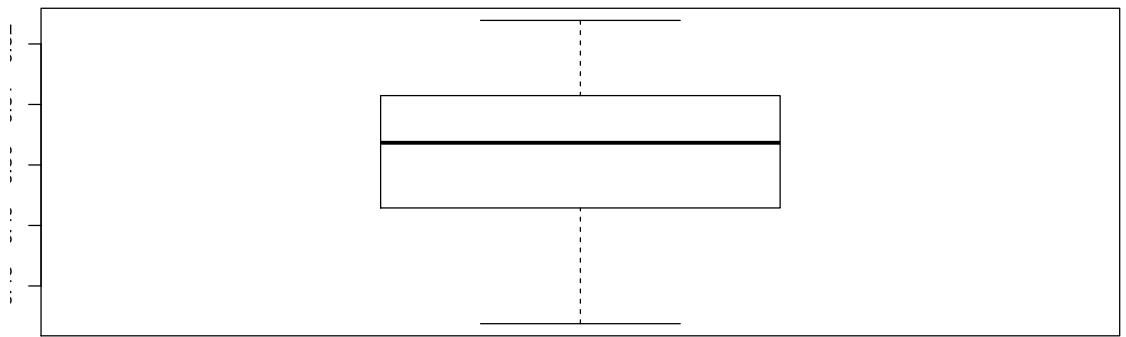
```
hist(means1000)
```

histogram of means1000

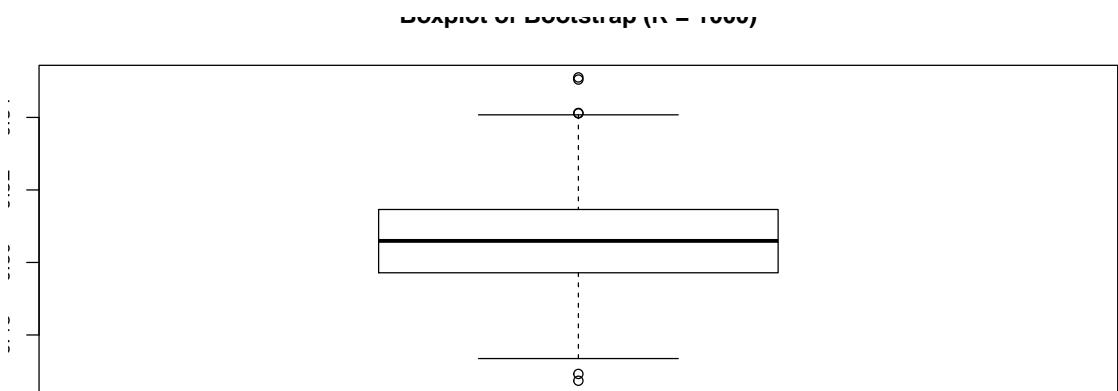


```
boxplot(boot50, main='Boxplot of Bootstrap (R = 50)')
```

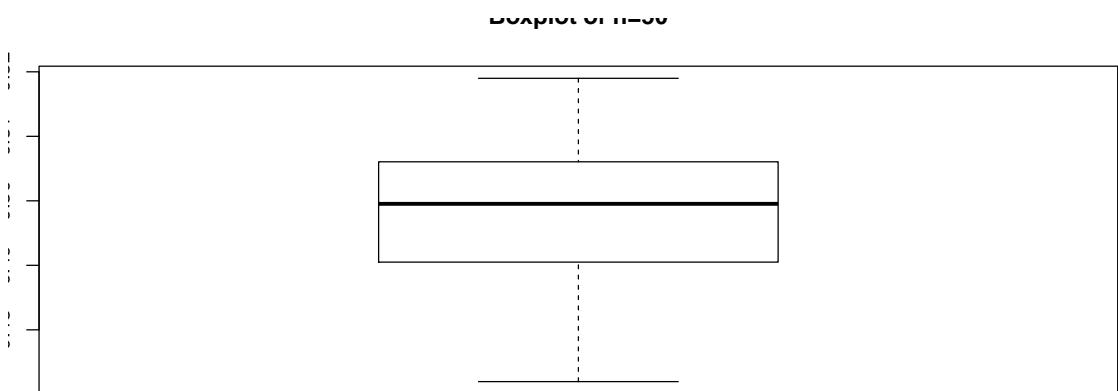
Boxplot of Bootstrap (n = 50)



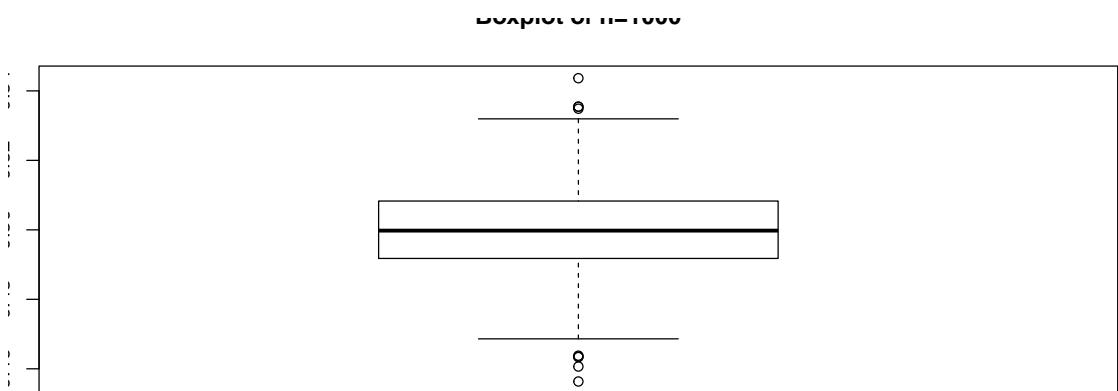
```
boxplot(boot1000, main='Boxplot of Bootstrap (R = 1000)')
```



```
boxplot(means50, main='Boxplot of n=50')
```

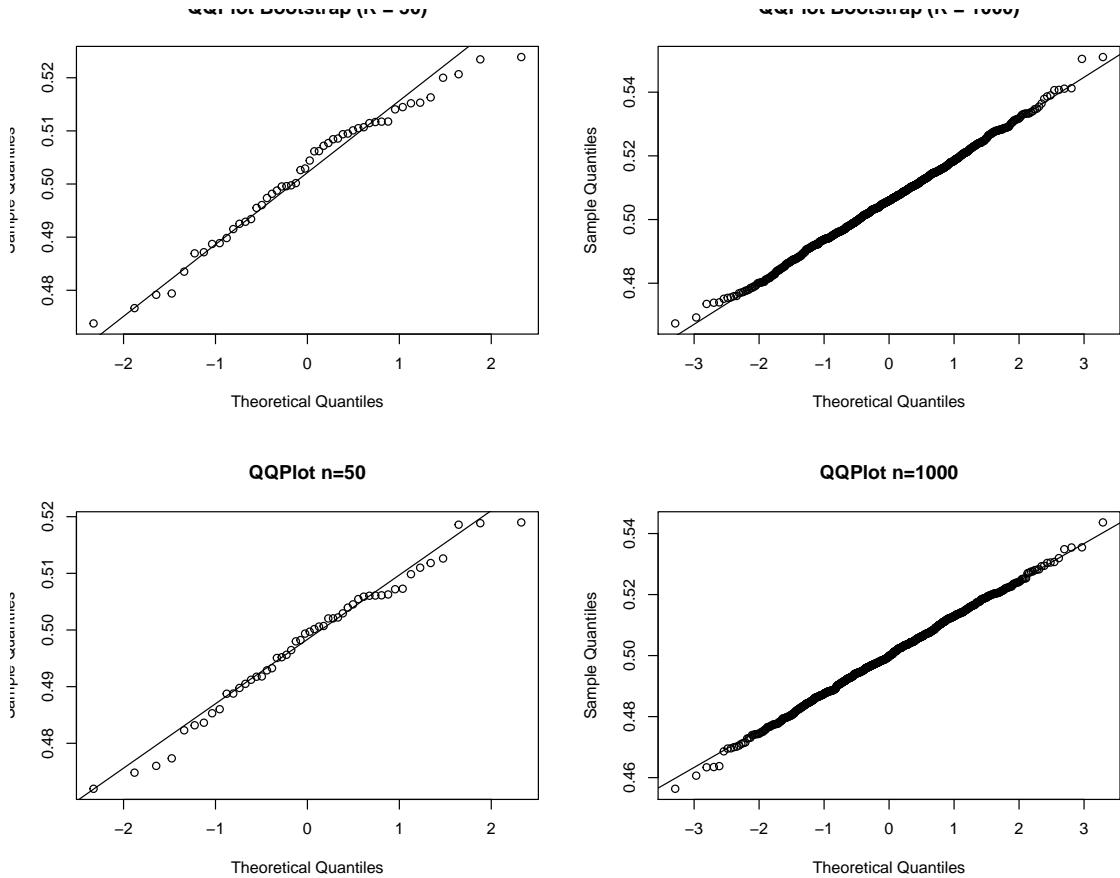


```
boxplot(means1000, main='Boxplot of n=1000')
```



```
par(mfrow=c(2,2))
qqnorm(boot50, main='QQPlot Bootstrap (R = 50)')
```

```
qqline(boot50)
qqnorm(boot1000, main='QQPlot Bootstrap (R = 1000)')
qqline(boot1000)
qqnorm(means50, main='QQPlot n=50')
qqline(means50)
qqnorm(means1000, main='QQPlot n=1000')
qqline(means1000)
```



```
shapiro.test(means50)
#>
#> Shapiro-Wilk normality test
#>
#> data: means50
#> W = 0.9789, p-value = 0.5064
shapiro.test(means1000)
#>
#> Shapiro-Wilk normality test
#>
```

```
#> data: means1000
#> W = 0.9992, p-value = 0.9559
shapiro.test(boot50)
#>
#> Shapiro-Wilk normality test
#>
#> data: boot50
#> W = 0.97108, p-value = 0.256
shapiro.test(boot1000)
#>
#> Shapiro-Wilk normality test
#>
#> data: boot1000
#> W = 0.99878, p-value = 0.7409
```

Laut Shapiro-Wilks-Test kann für keine der vier Stichproben die Nullhypothese (Stichprobe ist normalverteilt) verworfen werden, auch wenn die Histogramme von boot50 und means50 nicht nach Normalverteilung aussehen.

d)

```
levene50 <- c(means50,boot50)
levene1000 <- c(means1000,boot1000)

val = c(rep(1, length(means50)), rep(2, length(boot50)))
levgroup50 <- as.factor(val)
val = c(rep(1, length(means1000)), rep(2, length(boot1000)))
levgroup1000 <- as.factor(val)

leveneTest(levene50, levgroup50)
#> Levene's Test for Homogeneity of Variance (center = median)
#>      Df F value Pr(>F)
#> group  1  0.7377 0.3925
#>       98
leveneTest(levene1000, levgroup1000)
#> Levene's Test for Homogeneity of Variance (center = median)
#>      Df F value Pr(>F)
#> group    1  0.0106 0.9181
#>       1998

t.test(means50, boot50, var.equal = TRUE)
```

```
#>
#> Two Sample t-test
#>
#> data: means50 and boot50
#> t = -1.8056, df = 98, p-value = 0.07405
#> alternative hypothesis: true difference in means is not equal to 0
#> 95 percent confidence interval:
#> -0.009042639 0.000426705
#> sample estimates:
#> mean of x mean of y
#> 0.497763 0.502071

t.test(means1000, boot1000, var.equal = TRUE)
#>
#> Two Sample t-test
#>
#> data: means1000 and boot1000
#> t = -10.524, df = 1998, p-value < 2.2e-16
#> alternative hypothesis: true difference in means is not equal to 0
#> 95 percent confidence interval:
#> -0.007088453 -0.004861553
#> sample estimates:
#> mean of x mean of y
#> 0.5000432 0.5060182
```

Um t-Tests durchführen zu können, müssen die Gruppen erst auf Varianzhomogenität geprüft werden. Diese wird mit dem Levene-Test geprüft. Laut diesem ist die Varianzhomogenität sowohl für Means_{sim50} und Means_{boot50} und Means_{sim1000} und Means_{boot1000} gegeben.

Der t-Test zur Überprüfung der Mittelwerte von Means_{sim50} und Means_{boot50} lieferte ein insignifikantes Ergebnis. Aus diesem Grund kann die Nullhypothese ("Differenz der Mittelwerte gleich 0.") nicht verworfen werden.

Für die Mittelwerte von Means_{sim1000} und Means_{boot1000} kann die Nullhypothese mit einem Siginifkanzniveau von 95% verworfen werden.

3.2 A2

a)

```
load("res/Donald.RData")
vergleich <- lm(
  data = Donald_1,
```

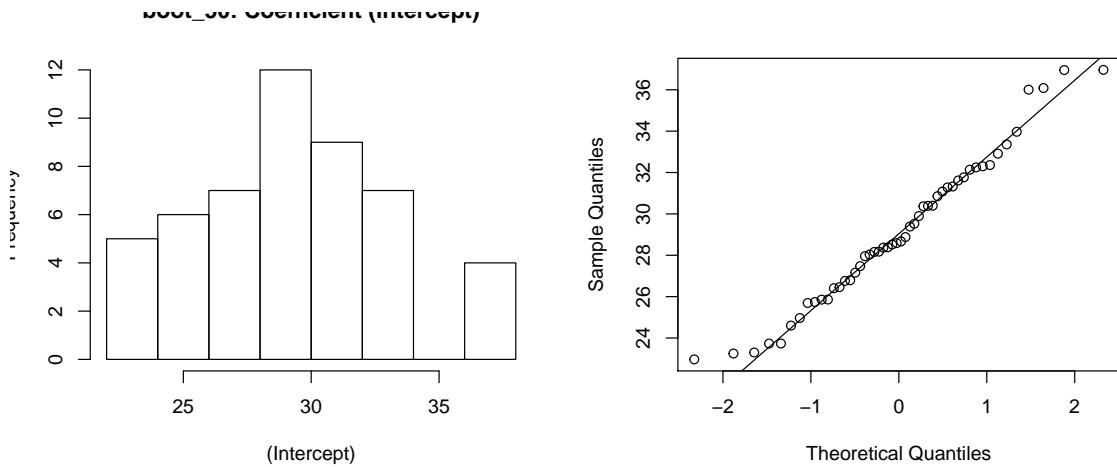
```

Trump ~ Geschlecht + Alter + Minderheit + Fremdenfeindlich + IQ
theta <- function(formula, data, indices){
  d <- data[indices,]
  fit <- lm(formula, data = d)
  return(coef(fit))
}

coef_names <- names(vergleich$coefficients)

boot50_1 <- boot(data=Donald_1, statistic=theta, R=50, formula=Trump~.)
par(mfrow=c(1,2))
for (i in 1:6){
  hist(boot50_1$t[,i],
    main=paste("boot_50: Coefficient",
    coef_names[i]), xlab = coef_names[i])
  qqnorm(boot50_1$t[,i], main = NULL)
  qqline(boot50_1$t[,i])
  print(t.test(boot50_1$t[,i], mu=vergleich$coefficients[i]))
}

```

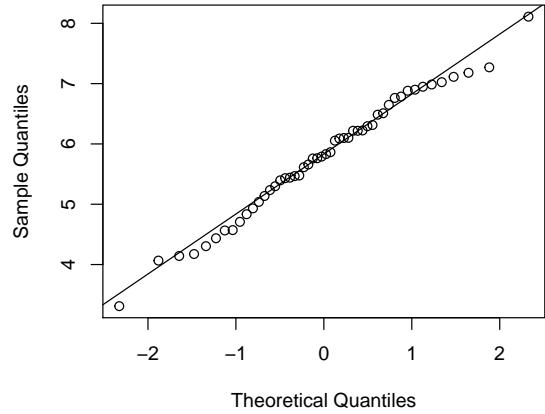
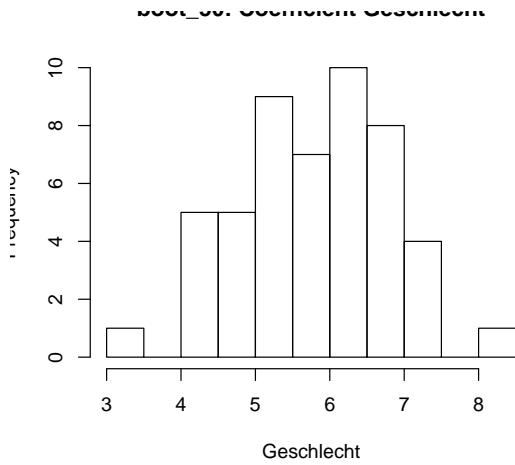


```

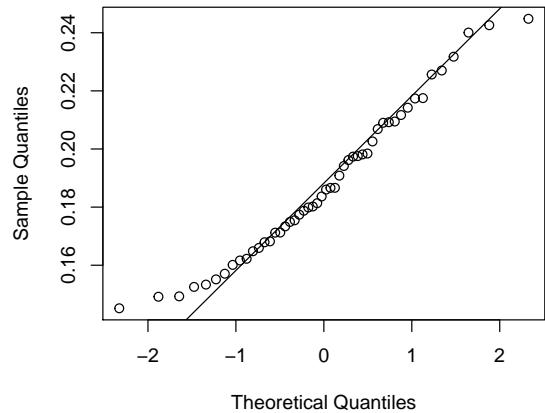
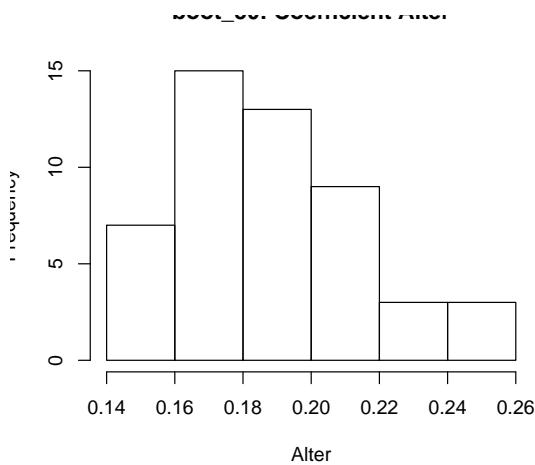
#>
#> One Sample t-test
#>
#> data: boot50_1$t[, i]
#> t = -1.1499, df = 49, p-value = 0.2558
#> alternative hypothesis: true mean is not equal to 29.73834
#> 95 percent confidence interval:
#> 28.13126 30.17565
#> sample estimates:

```

```
#> mean of x
#> 29.15346
```

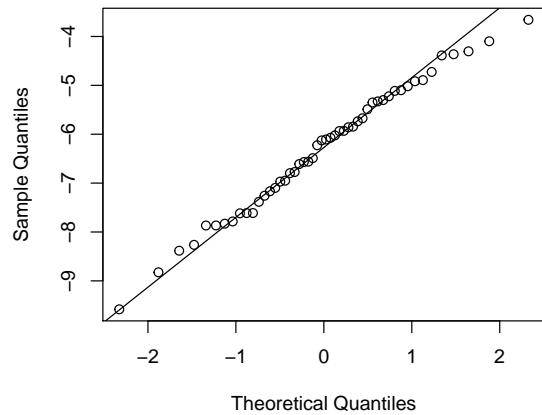
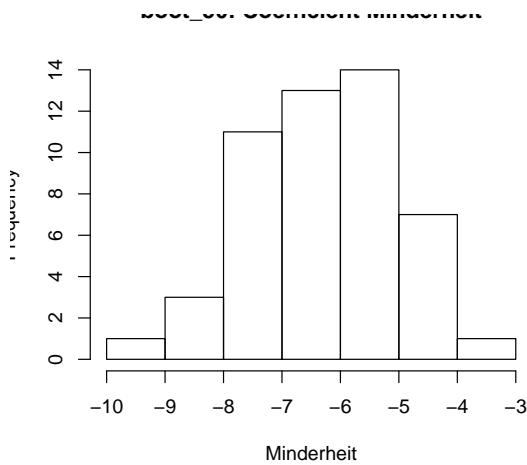


```
#>
#> One Sample t-test
#>
#> data: boot50_1$t[, i]
#> t = 0.23409, df = 49, p-value = 0.8159
#> alternative hypothesis: true mean is not equal to 5.75572
#> 95 percent confidence interval:
#> 5.503144 6.074898
#> sample estimates:
#> mean of x
#> 5.789021
```

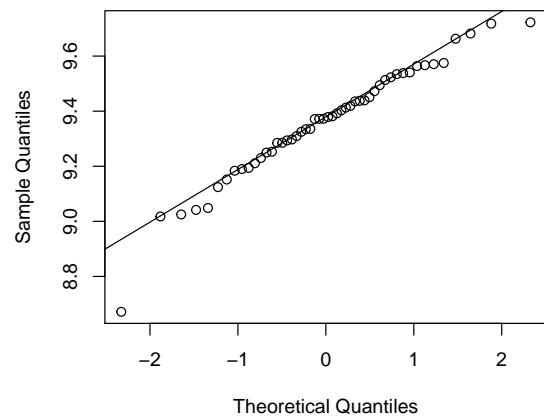
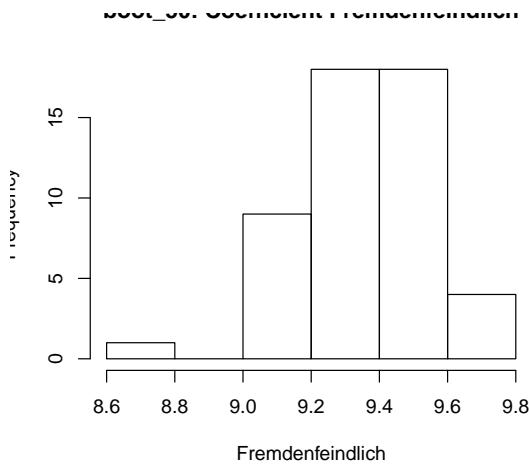


```
#>
```

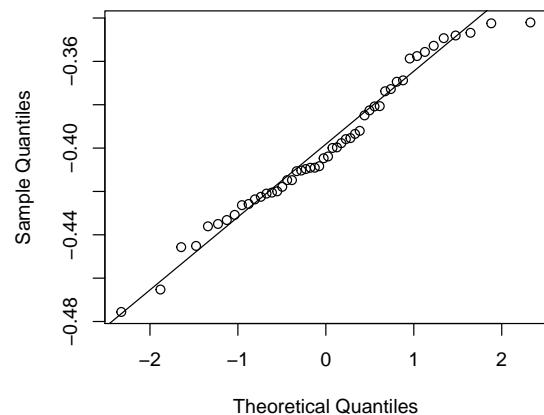
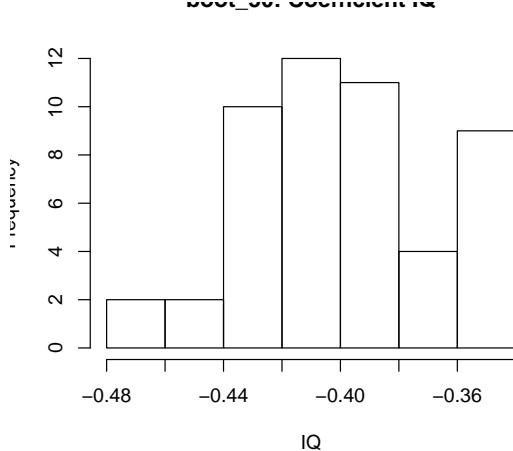
```
#> One Sample t-test
#>
#> data: boot50_1$t[, i]
#> t = 1.748, df = 49, p-value = 0.08673
#> alternative hypothesis: true mean is not equal to 0.1815267
#> 95 percent confidence interval:
#> 0.1805527 0.1955171
#> sample estimates:
#> mean of x
#> 0.1880349
```



```
#>
#> One Sample t-test
#>
#> data: boot50_1$t[, i]
#> t = 1.5126, df = 49, p-value = 0.1368
#> alternative hypothesis: true mean is not equal to -6.575863
#> 95 percent confidence interval:
#> -6.668653 -5.918275
#> sample estimates:
#> mean of x
#> -6.293464
```



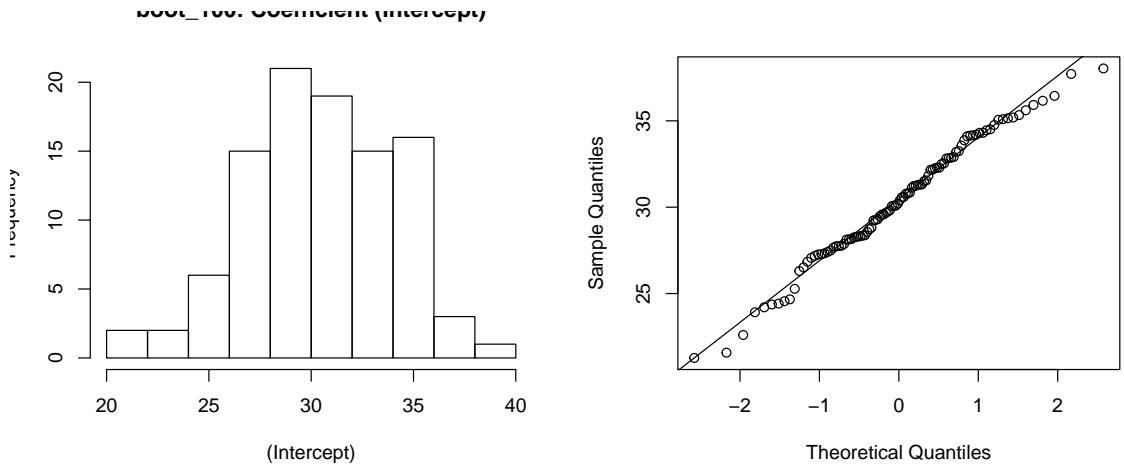
```
#>
#> One Sample t-test
#>
#> data: boot50_1$t[, i]
#> t = 0.35607, df = 49, p-value = 0.7233
#> alternative hypothesis: true mean is not equal to 9.349838
#> 95 percent confidence interval:
#> 9.302384 9.417730
#> sample estimates:
#> mean of x
#> 9.360057
```



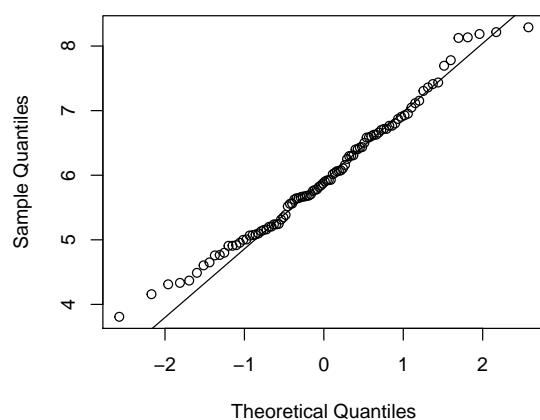
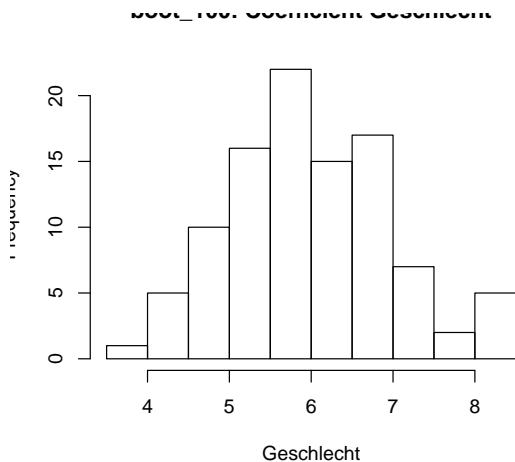
```
#>
#> One Sample t-test
#>
#> data: boot50_1$t[, i]
```

```
#> t = 0.38054, df = 49, p-value = 0.7052
#> alternative hypothesis: true mean is not equal to -0.4013497
#> 95 percent confidence interval:
#> -0.4087766 -0.3904530
#> sample estimates:
#> mean of x
#> -0.3996148

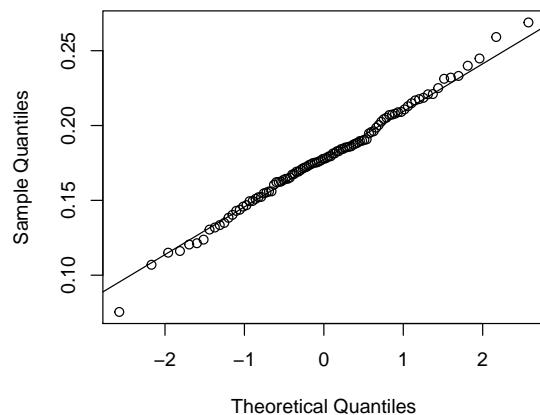
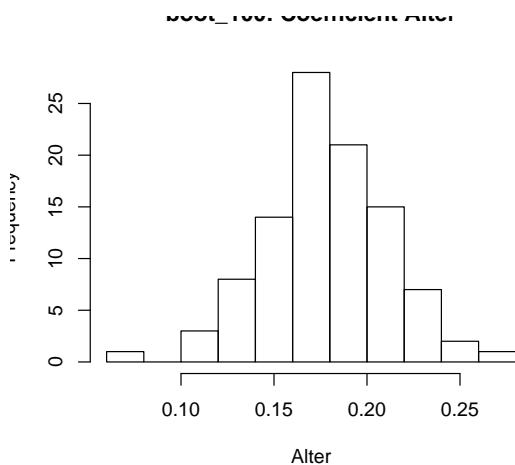
boot100_1 <- boot(data=Donald_1, statistic=theta, R=100, formula=Trump~.)
par(mfrow=c(1, 2))
for (i in 1:6){
  hist(boot100_1$t[,i], main=paste("boot_100: Coefficient", coef_names[i]), xlab = coef_name[i])
  qqnorm(boot100_1$t[,i], main = NULL)
  qqline(boot100_1$t[,i])
  print(t.test(boot100_1$t[,i], mu=vergleich$coefficients[i]))
}
```



```
#>
#> One Sample t-test
#>
#> data: boot100_1$t[, i]
#> t = 1.7239, df = 99, p-value = 0.08784
#> alternative hypothesis: true mean is not equal to 29.73834
#> 95 percent confidence interval:
#> 29.64540 31.06252
#> sample estimates:
#> mean of x
#> 30.35396
```

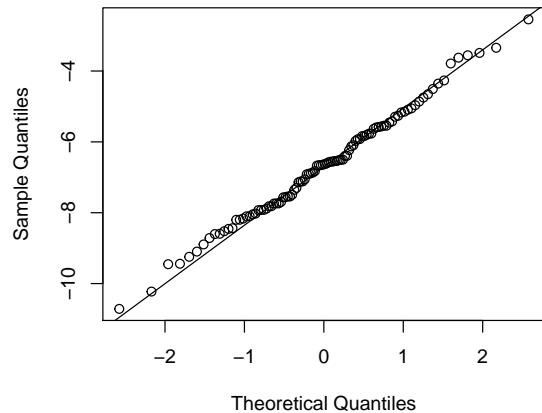
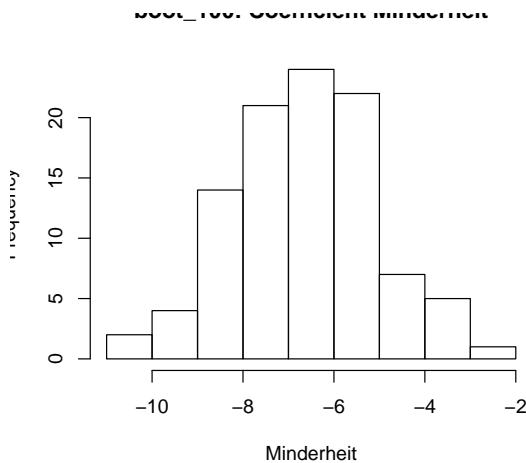


```
#>
#> One Sample t-test
#>
#> data: boot100_1$t[, i]
#> t = 2.2386, df = 99, p-value = 0.02742
#> alternative hypothesis: true mean is not equal to 5.75572
#> 95 percent confidence interval:
#> 5.780898 6.173650
#> sample estimates:
#> mean of x
#> 5.977274
```

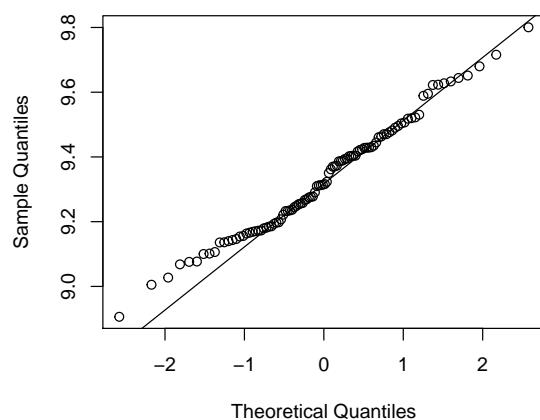
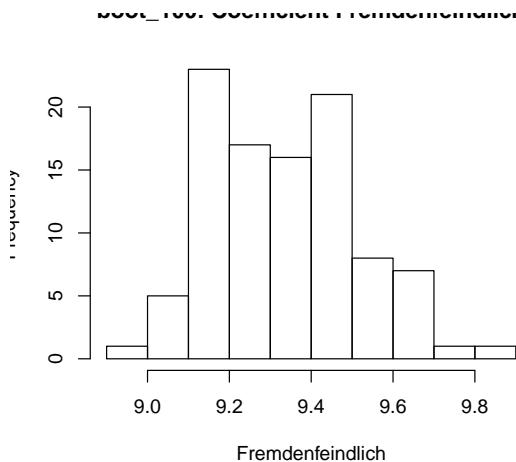


```
#>
#> One Sample t-test
#>
#> data: boot100_1$t[, i]
```

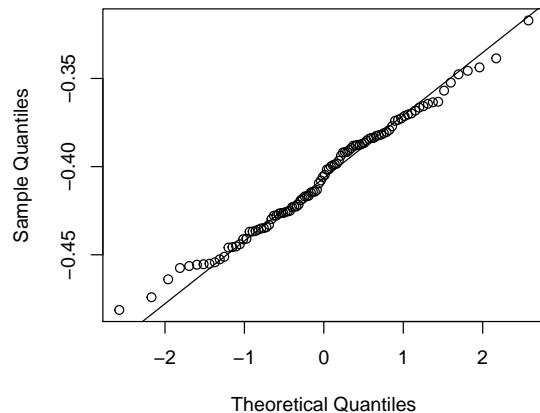
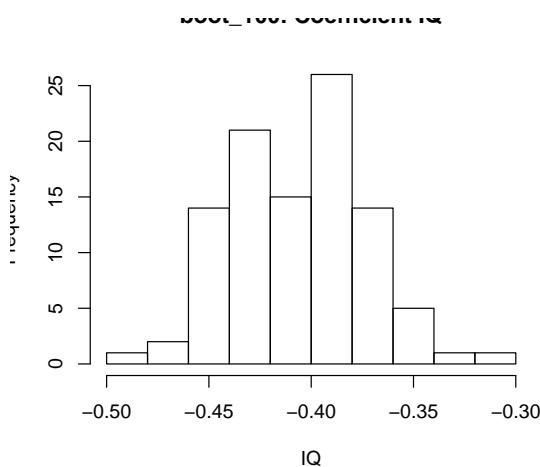
```
#> t = -1.0572, df = 99, p-value = 0.293
#> alternative hypothesis: true mean is not equal to 0.1815267
#> 95 percent confidence interval:
#> 0.1712778 0.1846506
#> sample estimates:
#> mean of x
#> 0.1779642
```



```
#>
#> One Sample t-test
#>
#> data: boot100_1$t[, i]
#> t = -0.5942, df = 99, p-value = 0.5537
#> alternative hypothesis: true mean is not equal to -6.575863
#> 95 percent confidence interval:
#> -6.979361 -6.358339
#> sample estimates:
#> mean of x
#> -6.66885
```



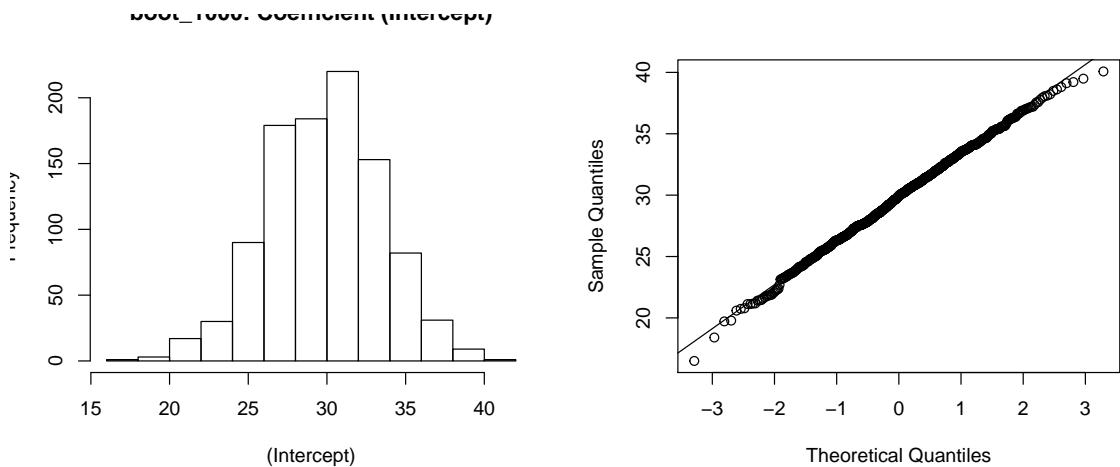
```
#>
#> One Sample t-test
#>
#> data: boot100_1$t[, i]
#> t = -0.94105, df = 99, p-value = 0.349
#> alternative hypothesis: true mean is not equal to 9.349838
#> 95 percent confidence interval:
#> 9.298257 9.368232
#> sample estimates:
#> mean of x
#> 9.333245
```



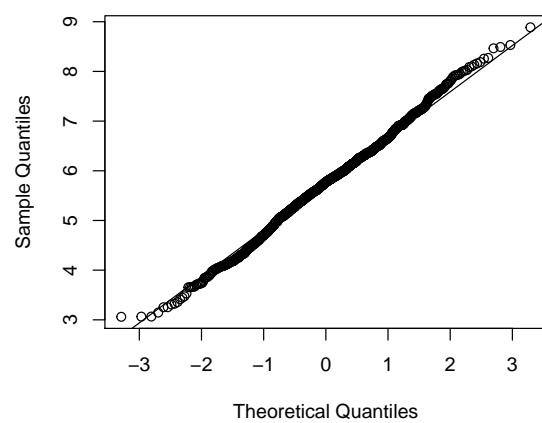
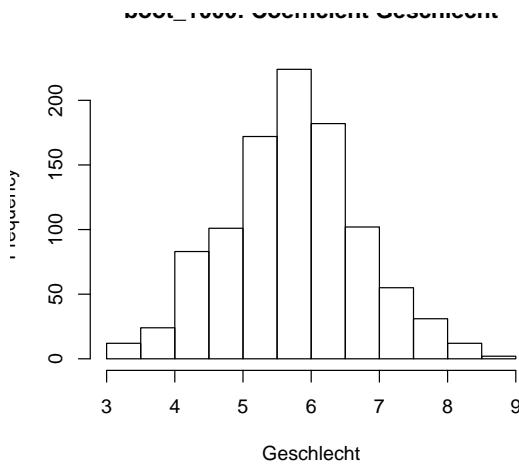
```
#>
#> One Sample t-test
#>
#> data: boot100_1$t[, i]
```

```
#> t = -1.406, df = 99, p-value = 0.1629
#> alternative hypothesis: true mean is not equal to -0.4013497
#> 95 percent confidence interval:
#> -0.4126399 -0.3994242
#> sample estimates:
#> mean of x
#> -0.406032

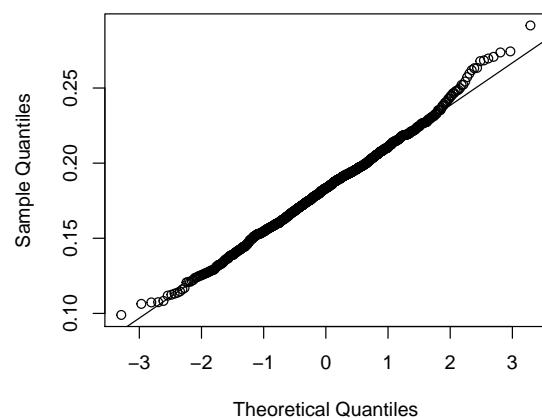
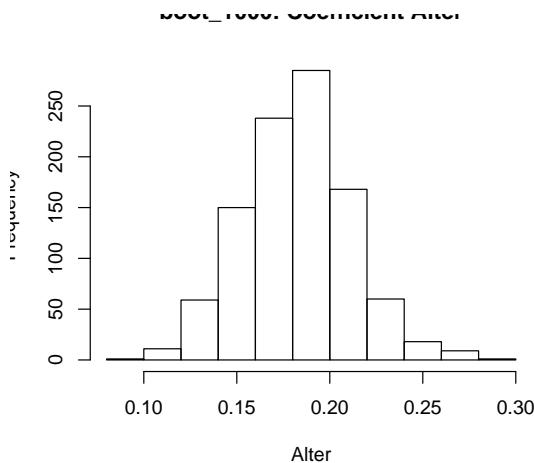
boot1000_1 <- boot(data=Donald_1, statistic=theta, R=1000, formula=Trump~.)
par(mfrow=c(1,2))
for (i in 1:6){
  hist(boot1000_1$t[,i], main=paste("boot_1000: Coefficient", coef_names[i]), xlab = coef_names[i])
  qqnorm(boot1000_1$t[,i], main = NULL)
  qqline(boot1000_1$t[,i])
  print(t.test(boot1000_1$t[,i], mu=vergleich$coefficients[i]))
}
```



```
#>
#> One Sample t-test
#>
#> data: boot1000_1$t[, i]
#> t = 0.55418, df = 999, p-value = 0.5796
#> alternative hypothesis: true mean is not equal to 29.73834
#> 95 percent confidence interval:
#> 29.57863 30.02377
#> sample estimates:
#> mean of x
#> 29.8012
```

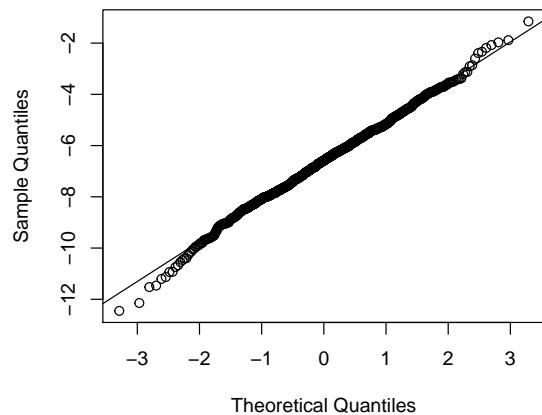
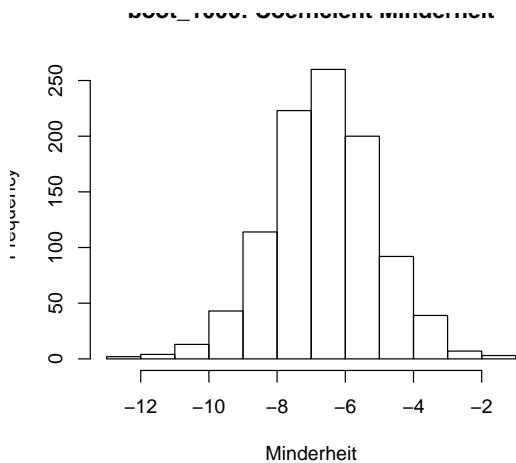


```
#>
#> One Sample t-test
#>
#> data: boot1000_1$t[, i]
#> t = -0.69897, df = 999, p-value = 0.4847
#> alternative hypothesis: true mean is not equal to 5.75572
#> 95 percent confidence interval:
#> 5.672561 5.795196
#> sample estimates:
#> mean of x
#> 5.733879
```

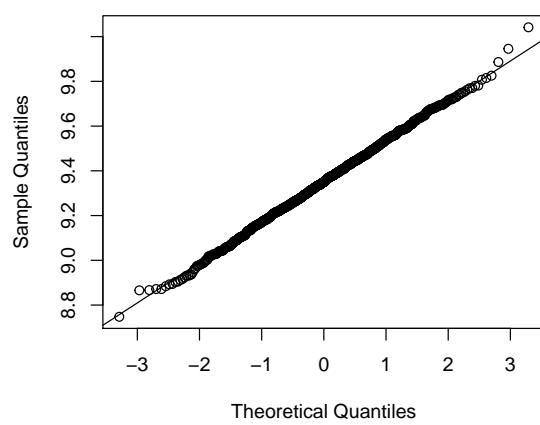
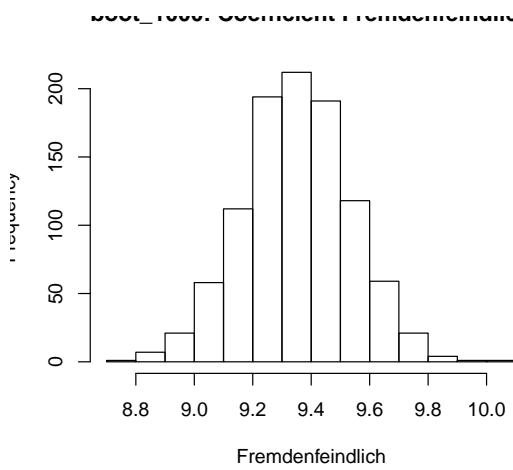


```
#>
#> One Sample t-test
#>
#> data: boot1000_1$t[, i]
```

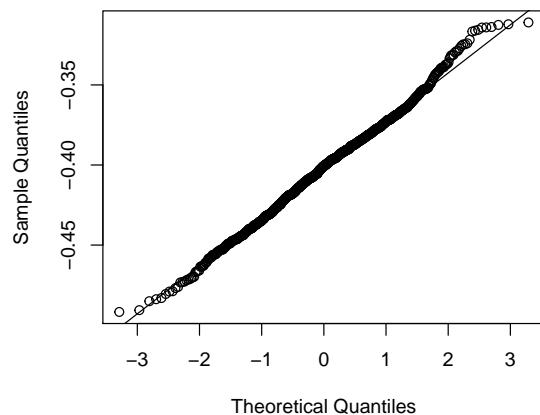
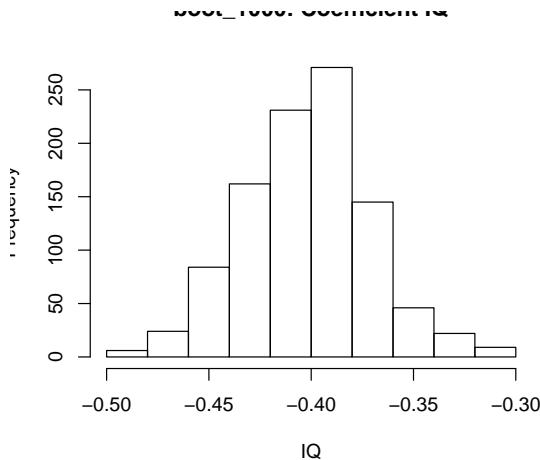
```
#> t = 1.4219, df = 999, p-value = 0.1554
#> alternative hypothesis: true mean is not equal to 0.1815267
#> 95 percent confidence interval:
#> 0.1810316 0.1846272
#> sample estimates:
#> mean of x
#> 0.1828294
```



```
#>
#> One Sample t-test
#>
#> data: boot1000_1$t[, i]
#> t = -0.86717, df = 999, p-value = 0.3861
#> alternative hypothesis: true mean is not equal to -6.575863
#> 95 percent confidence interval:
#> -6.715782 -6.521706
#> sample estimates:
#> mean of x
#> -6.618744
```



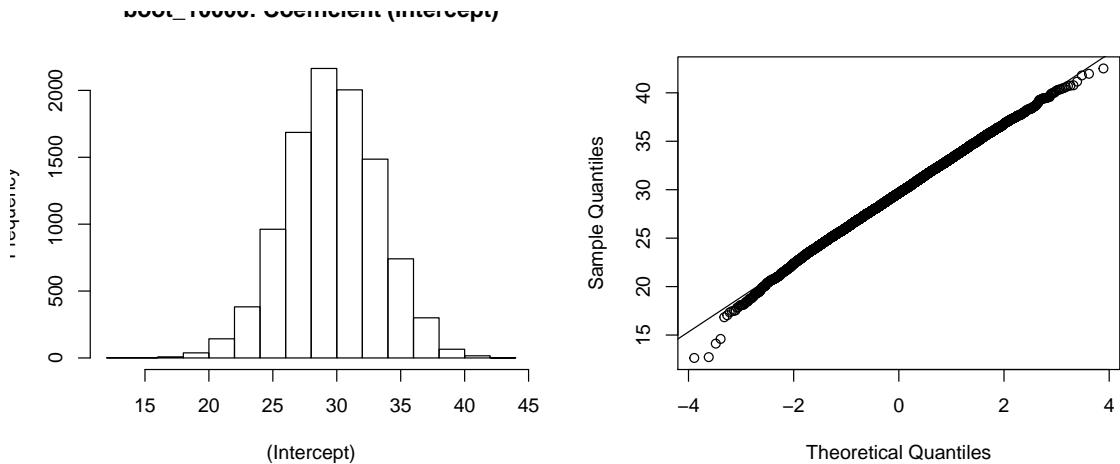
```
#>
#> One Sample t-test
#>
#> data: boot1000_1$t[, i]
#> t = 0.22419, df = 999, p-value = 0.8227
#> alternative hypothesis: true mean is not equal to 9.349838
#> 95 percent confidence interval:
#> 9.339718 9.362569
#> sample estimates:
#> mean of x
#> 9.351143
```



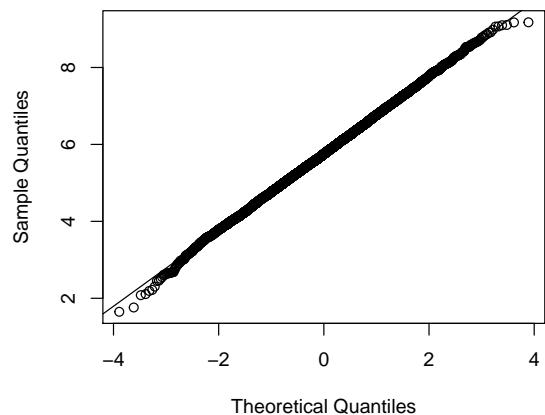
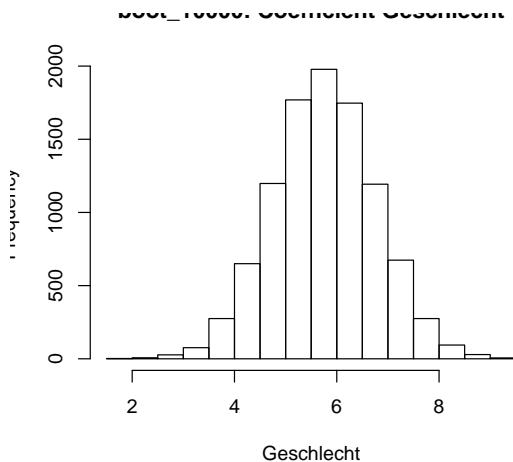
```
#>
#> One Sample t-test
#>
#> data: boot1000_1$t[, i]
```

```
#> t = -0.93855, df = 999, p-value = 0.3482
#> alternative hypothesis: true mean is not equal to -0.4013497
#> 95 percent confidence interval:
#> -0.4042033 -0.4003426
#> sample estimates:
#> mean of x
#> -0.4022729

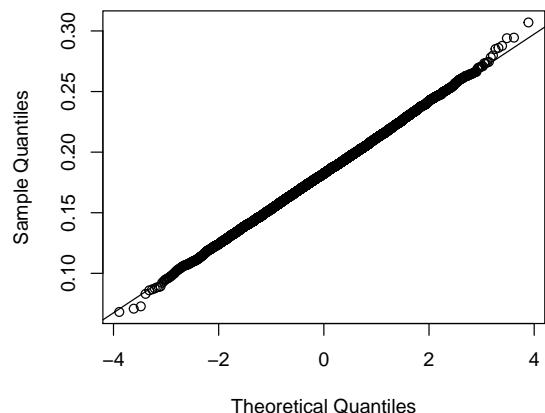
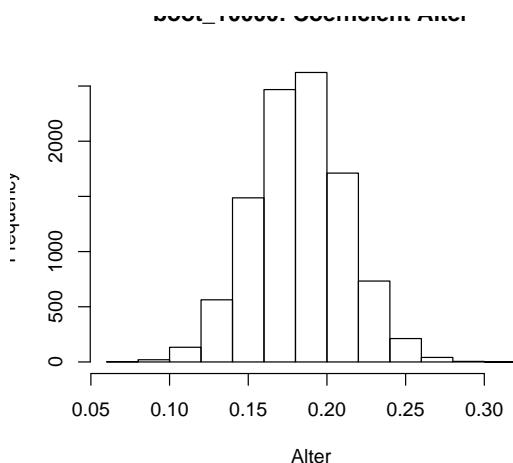
boot10000_1 <- boot(data=Donald_1, statistic=theta, R=10000, formula=Trump~.)
par(mfrow=c(1,2))
for (i in 1:6){
  hist(boot10000_1$t[,i], main=paste("boot_10000: Coefficient", coef_names[i]), xlab = coef_names[i])
  qqnorm(boot10000_1$t[,i], main = NULL)
  qqline(boot10000_1$t[,i])
  print(t.test(boot10000_1$t[,i], mu=vergleich$coefficients[i]))
}
```



```
#>
#> One Sample t-test
#>
#> data: boot10000_1$t[, i]
#> t = -2.4243, df = 9999, p-value = 0.01536
#> alternative hypothesis: true mean is not equal to 29.73834
#> 95 percent confidence interval:
#> 29.58034 29.72162
#> sample estimates:
#> mean of x
#> 29.65098
```

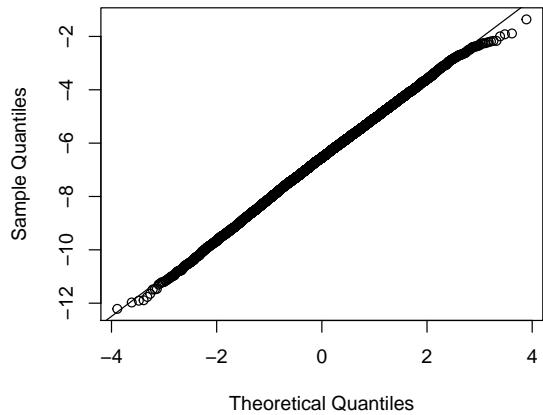
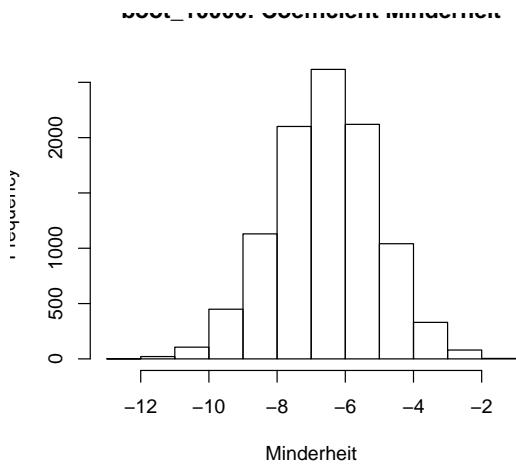


```
#>
#> One Sample t-test
#>
#> data: boot10000_1$t[, i]
#> t = 0.047807, df = 9999, p-value = 0.9619
#> alternative hypothesis: true mean is not equal to 5.75572
#> 95 percent confidence interval:
#> 5.736572 5.775825
#> sample estimates:
#> mean of x
#> 5.756198
```

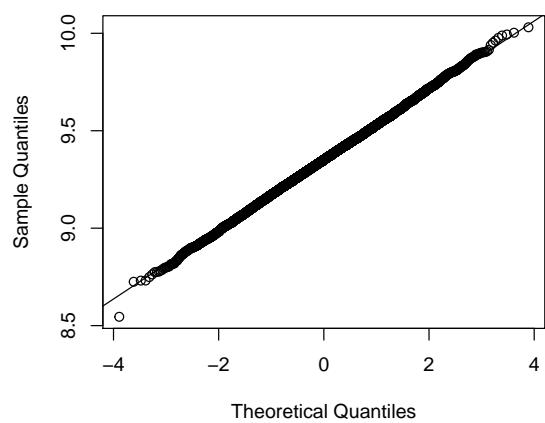
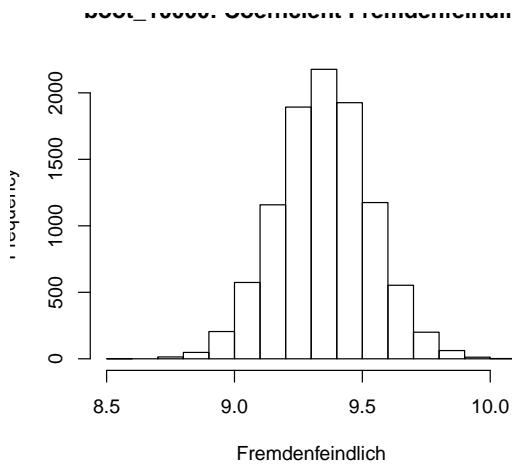


```
#>
#> One Sample t-test
#>
#> data: boot10000_1$t[, i]
```

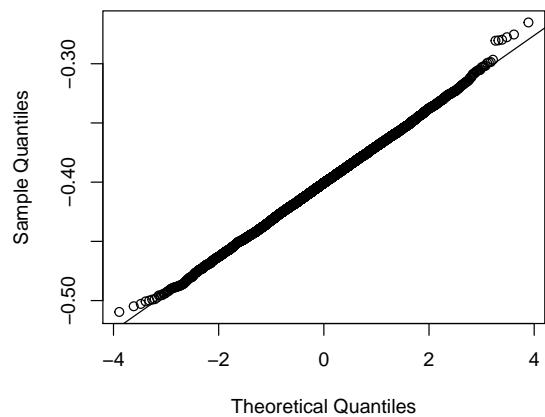
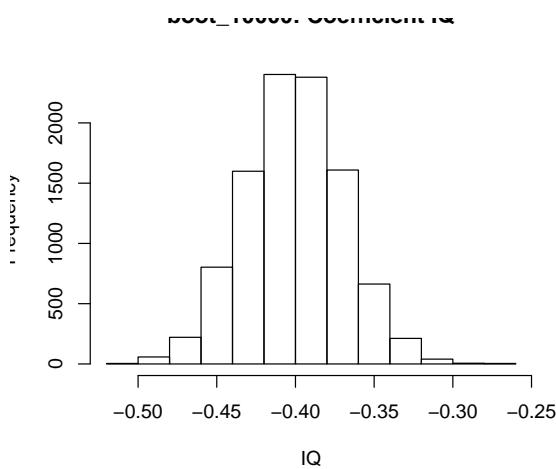
```
#> t = 3.1769, df = 9999, p-value = 0.001493
#> alternative hypothesis: true mean is not equal to 0.1815267
#> 95 percent confidence interval:
#> 0.1818826 0.1830295
#> sample estimates:
#> mean of x
#> 0.1824561
```



```
#>
#> One Sample t-test
#>
#> data: boot10000_1$t[, i]
#> t = 0.52339, df = 9999, p-value = 0.6007
#> alternative hypothesis: true mean is not equal to -6.575863
#> 95 percent confidence interval:
#> -6.597539 -6.538394
#> sample estimates:
#> mean of x
#> -6.567966
```



```
#>
#> One Sample t-test
#>
#> data: boot10000_1$t[, i]
#> t = 0.23136, df = 9999, p-value = 0.817
#> alternative hypothesis: true mean is not equal to 9.349838
#> 95 percent confidence interval:
#> 9.346706 9.353808
#> sample estimates:
#> mean of x
#> 9.350257
```



```
#>
#> One Sample t-test
#>
#> data: boot10000_1$t[, i]
```

```
#> t = 1.5753, df = 9999, p-value = 0.1152
#> alternative hypothesis: true mean is not equal to -0.4013497
#> 95 percent confidence interval:
#> -0.4014698 -0.4002468
#> sample estimates:
#> mean of x
#> -0.4008583
```

Hier untersuchen wir, ob wir durch die Bootstrappingverfahren auf das gleiche Ergebnis kommen wie die lineare Regression. Außerdem vergleichen wir die Verteilungen der Parameterschätzer für verschiedene Replikationsgrößen. Zu den Verteilungen lässt sich, wie bei Aufgabe 1, sagen, dass sich die Verteilungen mit mehr Replikationen immer mehr an die Normalverteilung annähern. Dies lässt sich an den Histogrammen, sowie den qqnorm-Plots ablesen. Die t-Tests sind tendenziell insignifikant ($p\text{-Wert} > 0.05$), die Nullhypotesen können deshalb nicht verworfen werden. Es gibt vereinzelt Ausnahmen (Beispiel: Bei boot_100 Geschlecht).

b)

```
Konfi50 <- matrix(NA, 6, 2)
rnames = c("Intercept", "Geschlecht", "Alter", "Minderheit", "Fremdenfeindlich", "IQ")
rownames(Konfi50) <- rnames
colnames(Konfi50) <- c("2,5%", "97,5%")
for(i in 1:6)
{
  Konfi50[i,1] <- boot.ci(boot50_1, type="basic", index = i)$basic[4]
  Konfi50[i,2] <- boot.ci(boot50_1, type="basic", index = i)$basic[5]
}
Konfi100 <- matrix(NA, 6, 2)
rnames = c("Intercept", "Geschlecht", "Alter", "Minderheit", "Fremdenfeindlich", "IQ")
rownames(Konfi100) <- rnames
colnames(Konfi100) <- c("2,5%", "97,5%")
for(i in 1:6)
{
  Konfi100[i,1] <- boot.ci(boot100_1, type="basic", index = i)$basic[4]
  Konfi100[i,2] <- boot.ci(boot100_1, type="basic", index = i)$basic[5]
}
Konfi1000 <- matrix(NA, 6, 2)
rnames = c("Intercept", "Geschlecht", "Alter", "Minderheit", "Fremdenfeindlich", "IQ")
rownames(Konfi1000) <- rnames
colnames(Konfi1000) <- c("2,5%", "97,5%")
for(i in 1:6)
{
```

```

Konfi1000[i,1] <- boot.ci(boot1000_1, type="basic", index = i)$basic[4]
Konfi1000[i,2] <- boot.ci(boot1000_1, type="basic", index = i)$basic[5]
}
Konfi10000 <- matrix(NA, 6, 2)
rnames = c("Intercept", "Geschlecht", "Alter", "Minderheit", "Fremdenfeindlich", "IQ")
rownames(Konfi10000) <- rnames
colnames(Konfi10000) <- c("2,5%", "97,5%")
for(i in 1:6)
{
  Konfi10000[i,1] <- boot.ci(boot10000_1, type="basic", index = i)$basic[4]
  Konfi10000[i,2] <- boot.ci(boot10000_1, type="basic", index = i)$basic[5]
}
kable(Konfi50, align = 'c', digits = 3)

```

	2,5%	97,5%
Intercept	22.518	36.413
Geschlecht	3.685	7.946
Alter	0.119	0.217
Minderheit	-9.344	-3.825
Fremdenfeindlich	8.979	9.911
IQ	-0.461	-0.331

```
kable(Konfi100, align = 'c', digits = 3)
```

	2,5%	97,5%
Intercept	22.481	37.319
Geschlecht	3.312	7.267
Alter	0.112	0.252
Minderheit	-9.726	-3.364
Fremdenfeindlich	9.004	9.682
IQ	-0.461	-0.334

```
kable(Konfi1000, align = 'c', digits = 3)
```

	2,5%	97,5%
Intercept	22.613	37.216
Geschlecht	3.755	7.724
Alter	0.120	0.237
Minderheit	-9.523	-3.350
Fremdenfeindlich	8.993	9.716
IQ	-0.465	-0.339

```
kable(Konfi10000, align = 'c', digits = 3)
```

	2,5%	97,5%
Intercept	22.849	36.953
Geschlecht	3.783	7.693
Alter	0.122	0.238
Minderheit	-9.488	-3.571
Fremdenfeindlich	8.989	9.706
IQ	-0.464	-0.341

c)

```
confi = confint(vergleich, level=0.95)
kable(confi, align = 'c', digits = 3)
```

	2,5 %	97,5 %
(Intercept)	22.603	36.873
Geschlecht	3.780	7.732
Alter	0.121	0.242
Minderheit	-10.190	-2.962
Fremdenfeindlich	9.027	9.673
IQ	-0.461	-0.342

```
kable(Konfi50, align = 'c', digits = 3)
```

	2,5%	97,5%
Intercept	22.518	36.413
Geschlecht	3.685	7.946
Alter	0.119	0.217
Minderheit	-9.344	-3.825
Fremdenfeindlich	8.979	9.911
IQ	-0.461	-0.331

```
kable(Konfi100, align = 'c', digits = 3)
```

	2,5%	97,5%
Intercept	22.481	37.319
Geschlecht	3.312	7.267
Alter	0.112	0.252
Minderheit	-9.726	-3.364
Fremdenfeindlich	9.004	9.682
IQ	-0.461	-0.334

```
kable(Konfi1000, align = 'c', digits = 3)
```

	2,5%	97,5%
Intercept	22.613	37.216
Geschlecht	3.755	7.724
Alter	0.120	0.237
Minderheit	-9.523	-3.350
Fremdenfeindlich	8.993	9.716
IQ	-0.465	-0.339

```
kable(Konfi10000, align = 'c', digits = 3)
```

	2,5%	97,5%
Intercept	22.849	36.953
Geschlecht	3.783	7.693
Alter	0.122	0.238
Minderheit	-9.488	-3.571
Fremdenfeindlich	8.989	9.706
IQ	-0.464	-0.341

Die Unterschiede der Konfidenzintervalle durch Bootstrapping mit dem Konfidenzintervall der linearen Regression sind verschwindend gering, deshalb ist eine Empfehlung schwierig. Wir würden uns für ein bootstrapping mit Replikationsgröße 10000 entscheiden, da hier die Unterschiede wohl am geringsten sind.

Interessanterweise ist bei boot_100 das Konfidenzintervall für den Parameterschätzer der Kovariaten "Geschlecht" am unterschiedlichsten zum Konfidenzintervall der linearen Regression. Im Gegensatz zu den anderen Replikationsgrößen ist dieses Intervall nach unten verschoben. Dies passt zu der in Aufgabe 2a) aufgetretenen Anomalie.

c)

```
Konfi1000perc <- matrix(NA, 6, 2)
rnames = c("Intercept", "Geschlecht", "Alter", "Minderheit", "Fremdenfeindlich", "IQ")
rownames(Konfi1000perc) <- rnames
colnames(Konfi1000perc) <- c("2,5%", "97,5%")
for(i in 1:6)
{
  Konfi1000perc[i,1] <- boot.ci(boot1000_1, type="perc", index = i)$perc[4]
  Konfi1000perc[i,2] <- boot.ci(boot1000_1, type="perc", index = i)$perc[5]
}
Konfi1000bca <- matrix(NA, 6, 2)
rnames = c("Intercept", "Geschlecht", "Alter", "Minderheit", "Fremdenfeindlich", "IQ")
```

```

rownames(Konfi1000bca) <- rnames
colnames(Konfi1000bca) <- c("2,5%","97,5%")
for(i in 1:6)
{
  Konfi1000bca[i,1] <- boot.ci(boot1000_1, type="bca", index = i)$bca[4]
  Konfi1000bca[i,2] <- boot.ci(boot1000_1, type="bca", index = i)$bca[5]
}
kable(Konfi1000, align = 'c', digits = 3)

```

	2,5%	97,5%
Intercept	22.613	37.216
Geschlecht	3.755	7.724
Alter	0.120	0.237
Minderheit	-9.523	-3.350
Fremdenfeindlich	8.993	9.716
IQ	-0.465	-0.339

```
kable(Konfi1000perc, align = 'c', digits = 3)
```

	2,5%	97,5%
Intercept	22.261	36.864
Geschlecht	3.788	7.756
Alter	0.126	0.243
Minderheit	-9.801	-3.628
Fremdenfeindlich	8.983	9.707
IQ	-0.463	-0.337

```
kable(Konfi1000bca, align = 'c', digits = 3)
```

	2,5%	97,5%
Intercept	22.330	36.883
Geschlecht	3.740	7.732
Alter	0.124	0.235
Minderheit	-9.804	-3.635
Fremdenfeindlich	8.987	9.711
IQ	-0.470	-0.341

Wir sehen hier keine großen Unterschiede in den Konfidenzintervallen.

4 Shrinking Methoden

4.1 A1

a)

```
linreg <- lm(lpsa~.-train, data = prostate)
linreg_coeff <- linreg$coefficients
```

b)

```
ridgereg_0 <- glmnet(x = model.matrix(linreg),
                      y = prostate$lpsa, alpha = 0, lambda = 0)
ridgereg_0_coeff = as.data.frame(as.matrix(ridgereg_0$beta))

ridgereg_10 <- glmnet(x = model.matrix(linreg),
                       y = prostate$lpsa, alpha = 0, lambda = 10)
ridgereg_10_coeff = as.data.frame(as.matrix(ridgereg_10$beta))

ridgereg_coeffs <- data.frame(ridgereg_0_coeff,
                               ridgereg_10_coeff, linreg_coeff)
colnames(ridgereg_coeffs) <- c("lambda_0", "lambda_10", "linreg")

lambda_0_mean <- mean(ridgereg_coeffs$lambda_0)
lambda_10_mean <- mean(ridgereg_coeffs$lambda_10)

kable(ridgereg_coeffs, align = 'c', digits = 3)
```

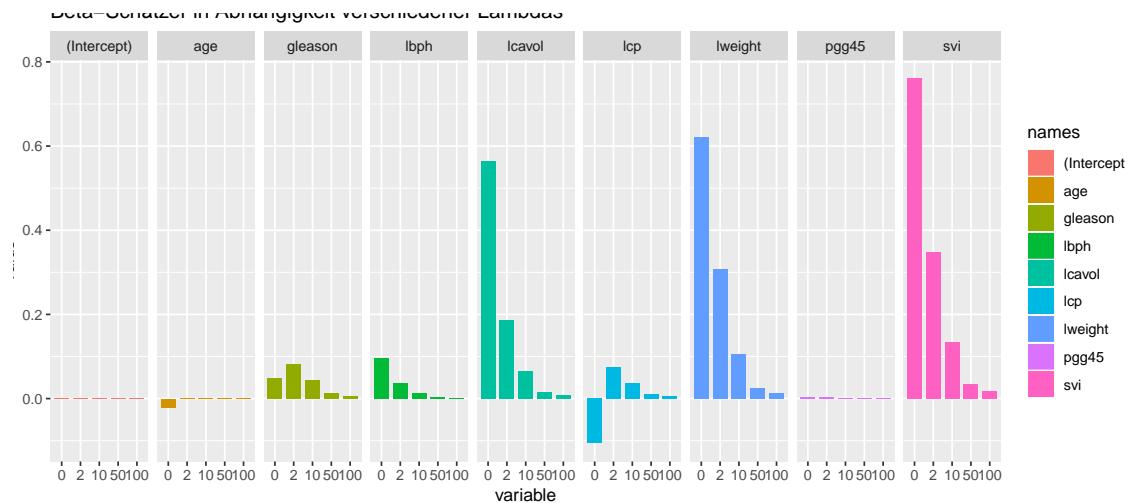
	lambda_0	lambda_10	linreg
(Intercept)	0.000	0.000	0.182
lcavol	0.564	0.064	0.564
lweight	0.622	0.107	0.622
age	-0.021	0.002	-0.021
lbph	0.097	0.013	0.097
svi	0.762	0.135	0.762
lcp	-0.106	0.036	-0.106
gleason	0.049	0.045	0.049
pgg45	0.004	0.001	0.004

Man sieht deutlich, dass ein größeres λ die Koeffizienten stärker “schrumpft” als ein kleines λ (Mittelwert der Koeffizienten für $\lambda = 0$: 0.219044; für $\lambda = 10$: 0.0448175). Bei $\lambda = 0$ sind die Koeffizienten (bis auf den Intercept) nahezu identisch mit den geschätzten Koeffizienten des

linearen Regressionsmodells. Im Falle $\lambda = 10$ sind zudem alle Koeffizienten positiv.

c)

```
coeff_plot <- function(lambda_list, y, linear_model, alpha){  
  coeff_list <- matrix(NA, nrow = length(names(linear_model$coefficients)),  
                        ncol = length(lambda_list))  
  
  for (i in 1:length(lambda_list)){  
    ridgereg <- glmnet(x = model.matrix(linear_model), y = y,  
                         alpha = alpha, lambda = lambda_list[i])  
    coeff <- as.vector(ridgereg$beta)  
    coeff_list[,i] <- coeff  
  }  
  coeff_list <- as.data.frame(coeff_list)  
  colnames(coeff_list) <- lambda_list  
  coeff_list$names <- names(linear_model$coefficients)  
  return(coeff_list)  
}  
  
coeffs <- coeff_plot(lambda_list = c(0, 2, 10, 50, 100), y = prostate$lpsa,  
                      linear_model = linreg, alpha = 0)  
  
df1.m <- melt(coeffs, id.vars = "names")  
  
p = ggplot(df1.m, aes(x=variable, y=value, fill=names))  
p = p + geom_bar(stat="identity", width = 0.75)  
p = p + facet_grid(. ~ names)  
p = p + ggttitle("Beta-Schätzer in Abhängigkeit verschiedener Lambdas")  
p
```



Negative β werden mit größer werdendem λ positiv. Außerdem konvergieren die β -Schätzer Richtung 0, werden aber nie genau 0 (anders als beim Lasso-Verfahren).

d)

```

train = prostate[prostate$train == TRUE, ]
test = prostate[prostate$train == FALSE, ]

lambdas <- c(0, 0.09, 2)
for (lambda in lambdas){
  ridgereg <- glmnet(x = as.matrix(within(train, rm(lpsa))),
                      y = train$lpsa, alpha = 0, lambda = lambda)
  y.true <- test$lpsa
  y.pred <- predict.glmnet(ridgereg, newx = as.matrix(within(test, rm(lpsa))),
                           lambda = lambda, alpha = 0)
  cat('\n')
  print(lambda)
  cat(mean((y.true-y.pred)^2))
  cat('\n')
}
#>
#> [1] 0
#> 0.52125
#>
#> [1] 0.09
#> 0.4940847
#>
#> [1] 2

```

```
#> 0.5691183
```

Das optimale λ liegt im Bereich um 0.09.

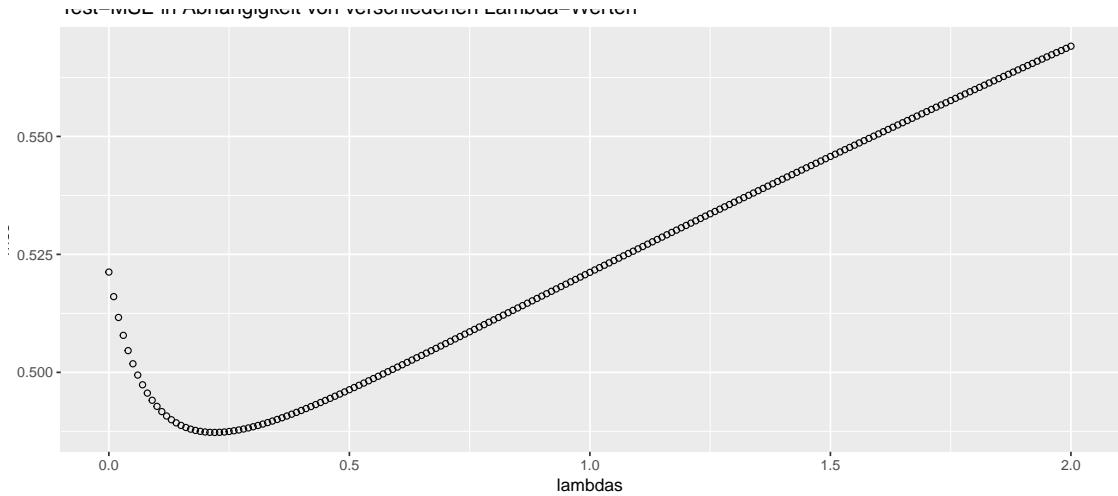
e)

```
lambdas <- seq(0, 2, 0.01)
cv_glm_fit <- cv.glmnet(x = as.matrix(within(prostate, rm(lpsa))),
                         y = prostate$lpsa, alpha = 0, lambda = lambdas)
opt_lambda <- cv_glm_fit$lambda.min

mse <- c()
for (lambda in lambdas){
  ridgereg <- glmnet(x = as.matrix(within(train, rm(lpsa))),
                       y = train$lpsa, alpha = 0, lambda = lambda)
  y.true <- test$lpsa
  y.pred <- predict.glmnet(ridgereg, newx = as.matrix(within(test, rm(lpsa))),
                           lambda = lambda, alpha = 0)
  mse <- append(mse, mean((y.true-y.pred)^2))
}

lambda_mse <- NULL
lambda_mse$lambdas <- lambdas
lambda_mse$mse <- mse

lambda_mse <- as.data.frame(lambda_mse)
ggplot(lambda_mse, aes(x=lambda_mse, y=mse)) +
  geom_point(shape=1) +
  ggtitle("Test-MSE in Abhängigkeit von verschiedenen Lambda-Werten")
```



```
opt_lambda_test <- lambda_mse$lambdas[lambda_mse$mse == min(lambda_mse$mse)]
```

Der optimale Schätzer für λ des kompletten Datensatzes beträgt laut cv.glmnet 0.04. Dieser liegt nicht weit vom Lambda-Wert aus d) mit dem niedrigsten Test-MSE (0.09). Der Plot des Test-MSE in Abhängigkeit von verschiedenen λ -Werten spricht allerdings für ein optimales Lambdas von 0.22. Diese Diskrepanz lässt sich durch zwei Faktoren erklären:

1. cv.glmnet führt standardmäßig eine 10-fache Kreuzvalidierung durch, um das beste λ zu finden. Das optimale λ laut Test-MSE wird hier nicht über Kreuzvalidierung ermittelt.
2. Wir ermitteln den Test-MSE ‘out-of-sample’, das heißt die Daten, mit denen wir die Ridge-Regression fitten, sind unterschiedlich.

f)

```
linreg <- lm(lpsa~.-train, data=prostate)

ridgereg <- glmnet(x = model.matrix(linreg), y = prostate$lpsa, alpha = 0,
                     lambda = opt_lambda)

ridgereg_coeff <- as.vector(ridgereg$beta)
coeff_df <- NULL
coeff_df$ridgereg_coeffs <- ridgereg_coeff
coeff_df$linreg_coeffs <- linreg_coeff
coeff_df <- as.data.frame(coeff_df)

coeff_df
#>          ridgereg_coeffs linreg_coeffs
#> (Intercept)      0.00000000  0.181560862
```

```
#> lcavol      0.52610716  0.564341279  
#> lweight     0.61452336  0.622019787  
#> age        -0.01887969 -0.021248185  
#> lbph        0.09095463  0.096712523  
#> svi         0.72086081  0.761673403  
#> lcp         -0.06951544 -0.106050939  
#> gleason    0.05805273  0.049227933  
#> pgg45      0.00383986  0.004457512
```

Vergleicht man die Koeffizienten der Ridge-Regression mit den Koeffizienten der linearen Regression so a) wird der “Shrinking”-Effekt der Ridge-Regression deutlich. Nahezu alle Koeffizienten der Ridge-Regression liegen näher an der 0 als die Koeffizienten der linearen Regression (mit Ausnahme des Koeffizienten der Kovariaten “gleason”).

4.2 A2

a)

```
lasso_0 <- glmnet(x = model.matrix(linreg), y = prostate$lpsa, alpha = 1,
                      lambda = 0)
lasso_10 <- glmnet(x = model.matrix(linreg), y = prostate$lpsa, alpha = 1,
                      lambda = 10)

coeff_df$lasso_0_coeffs <- as.vector(lasso_0$beta)

coeff_df$lasso_10_coeffs <- as.vector(lasso_10$beta)

kable(coeff_df %>% select(2:4), align = 'c', format = 'pandoc', digits = 3)
```

	linreg_coeffs	lasso_0_coeffs	lasso_10_coeffs
(Intercept)	0.182	0.000	0
lcavol	0.564	0.564	0
lweight	0.622	0.622	0
age	-0.021	-0.021	0
lbph	0.097	0.097	0
svi	0.762	0.762	0
lcp	-0.106	-0.106	0
gleason	0.049	0.049	0
pgg45	0.004	0.004	0

Das Lasso-Verfahren mit $\lambda = 0$ generiert nahezu identische β -Schätzer wie die lineare Regression. Bei $\lambda = 10$ liegen alle “geschrumpften” β -Schätzer bereits bei 0.

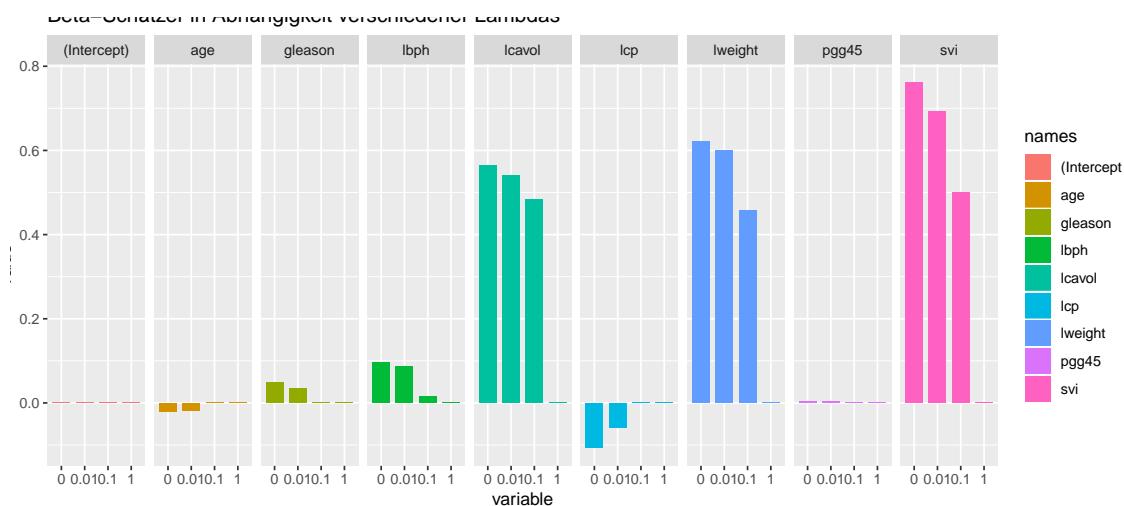
b)

```
coeffs <- coeff_plot(lambda_list = c(0, 0.01, 0.1, 1), y = prostate$lpsa,
                      linear_model = linreg, alpha = 1)

df1.m <- melt(coeffs, id.vars = "names")

p = ggplot(df1.m, aes(x=variable, y=value, fill=names))
p = p + geom_bar(stat="identity", width = 0.75)
p = p + facet_grid(. ~ names)
p = p + ggtitle("Beta-Schätzer in Abhängigkeit verschiedener Lambdas")
```

p



Auch beim Lasso-Verfahren zeichnet sich ein ähnliches Bild ab wie bei Aufgabe 1 c). Bei größer werdendem λ „schrumpfen“ die β -Schätzer, konvergieren gegen 0 und werden, anders als bei Ridge-Regression, ab einem bestimmten λ sogar 0.

c)

```

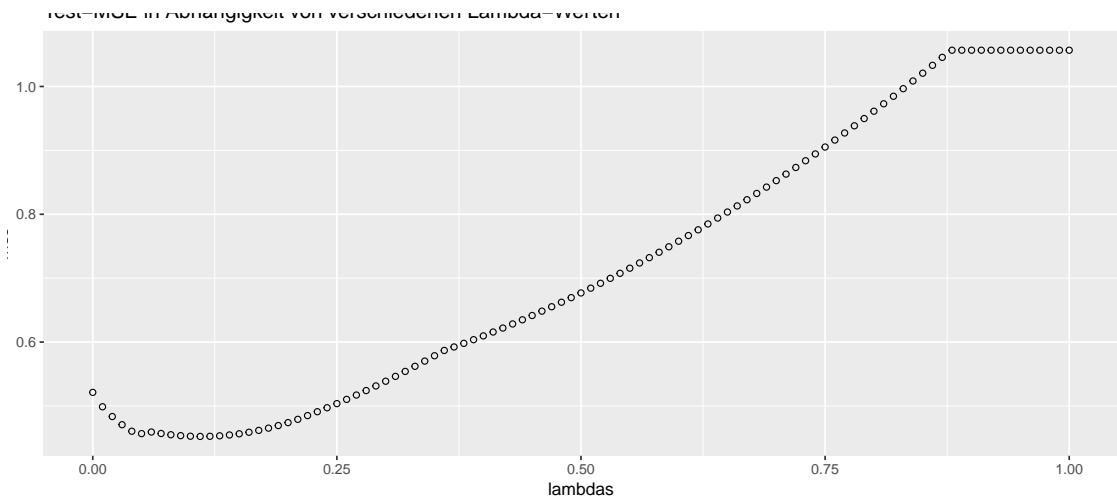
lambdas <- c(0, 0.002, 1)
for (lambda in lambdas){
  ridgereg <- glmnet(x = as.matrix(within(train, rm(lpsa))),
                      y = train$lpsa, alpha = 1, lambda = lambda)
  y.true <- test$lpsa
  y.pred <- predict.glmnet(
    ridgereg, newx = as.matrix(within(test, rm(lpsa))),
    lambda = lambda, alpha = 1)
  cat('\n')
  print(lambda)
  cat(mean((y.true-y.pred)^2))
  cat('\n')
}
#> [1] 0
#> 0.52125
#>
#> [1] 0.002
#> 0.5154957
#>
```

```
#> [1] 1  
#> 1.056733
```

Hier scheint der “Sweet Spot” für λ um 0.002 zu liegen.

d)

```
lambdas <- seq(0, 1, 0.01)  
cv_glm_fit <- cv.glmnet(x = as.matrix(within(prostate, rm(lpsa))),  
                           y = prostate$lpsa, alpha = 1, lambda = lambdas)  
opt_lambda <- cv_glm_fit$lambda.min  
  
mse <- c()  
for (lambda in lambdas){  
  ridgereg <- glmnet(x = as.matrix(within(train, rm(lpsa))),  
                      y = train$lpsa, alpha = 1, lambda = lambda)  
  y.true <- test$lpsa  
  y.pred <- predict.glmnet(ridgereg, newx = as.matrix(within(test, rm(lpsa))),  
                           lambda = lambda, alpha = 1)  
  mse <- append(mse, mean((y.true-y.pred)^2))  
}  
  
lambda_mse <- NULL  
lambda_mse$lambdas <- lambdas  
lambda_mse$mse <- mse  
  
lambda_mse <- as.data.frame(lambda_mse)  
ggplot(lambda_mse, aes(x=lambda_mse, y=mse)) +  
  geom_point(shape=1) +  
  ggtitle("Test-MSE in Abhängigkeit von verschiedenen Lambda-Werten")
```



```
opt_lambda_test <- lambda_mse$lambda[mse == min(lambda_mse$mse)]
```

Hier liegt der optimale Wert für λ für den kompletten Datensatz bei 0.03. Dieser Wert liegt wieder sehr nahe am ermittelten Optimum aus c). Plottet man den Test-MSE (out-of-sample) in Abhängigkeit verschiedener λ -Werte, kommt man allerdings auf ein optimalen λ -Wert von 0.11. Dies liegt wieder an den gleichen Gründen, wie in Aufgabe 1 e).

e)

```
lasso <- glmnet(x = model.matrix(linreg), y = prostate$lpsa, alpha = 1,
                  lambda = opt_lambda)

lasso_coeff <- as.vector(lasso$beta)
coeff_df$lasso_coeff <- lasso_coeff

kable(coeff_df %>% select(2, 5), format='pandoc', align = 'c', digits = 3)
```

	linreg_coeffs	lasso_coeff
(Intercept)	0.182	0.000
lcavol	0.564	0.509
lweight	0.622	0.558
age	-0.021	-0.010
lbph	0.097	0.067
svi	0.762	0.598
lcp	-0.106	0.000
gleason	0.049	0.008

	linreg_coeffs	lasso_coeff
pgg45	0.004	0.002

Wie bereits in Aufgabe 1 ist beim Lasso-Verfahren gut zu erkennen, dass die Koeffizienten im Vergleich zur linearen Regression aus Aufgabe 1 a) “geschrumpft” werden und näher an der 0 liegen. Anders als bei der Ridge-Regression sind beim Lasso-Verfahren zwei β -Schätzer tatsächlich 0 geworden (lcp und gleason).

5 PCA/PLS Regression

5.1 A1

a)

```
data = subset(prostate, select=c(lcavol,lweight,age,lbph,svi,lcp,gleason,pgg45))
cmat = cor(data)
kable(cmat, align = 'c', digits = 3)
```

	lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45
lcavol	1.000	0.281	0.225	0.027	0.539	0.675	0.432	0.434
lweight	0.281	1.000	0.348	0.442	0.155	0.165	0.057	0.107
age	0.225	0.348	1.000	0.350	0.118	0.128	0.269	0.276
lbph	0.027	0.442	0.350	1.000	-0.086	-0.007	0.078	0.078
svi	0.539	0.155	0.118	-0.086	1.000	0.673	0.320	0.458
lcp	0.675	0.165	0.128	-0.007	0.673	1.000	0.515	0.632
gleason	0.432	0.057	0.269	0.078	0.320	0.515	1.000	0.752
pgg45	0.434	0.107	0.276	0.078	0.458	0.632	0.752	1.000

Man erkennt, dass "pgg45" und "lcavol" am stärksten mit anderen Variablen korrelieren. Insbesondere "pgg45" und "gleason" weisen eine hoh positive Korrelation von 0.752 auf. Wenn wir Variablen mit hoher Korrelation als Prädiktor in das Modell hinzufügen, dann werden die Regressionskoeffizienten davon stark beeinflusst. Diese Multikollinearität ist insofern schlecht, da die Prognose dadurch in Richtungen gezerrt wird, die vom eigentlichen Ergebnis abweicht. Um dies zu vermeiden gibt es neben Variablen aus dem Modell nehmen verschiedene Verfahren. In Übung 4 nutze man dazu Ridge/Lasso Regression. Hier nun andere Methoden.

b)

```
data = subset(
  prostate,
  select = c(lcavol, lweight, age, lbph, svi, lcp, gleason, pgg45, lpsa)
)
pcr = pcr(
  lpsa ~ lcavol + lweight + age + lbph + svi + lcp + gleason + pgg45,
  data = data,
  validation = "CV",
  scale = TRUE
)
summary(pcr)
#> Data:    X dimension: 97 8
```

```
#> Y dimension: 97 1
#> Fit method: svdpc
#> Number of components considered: 8
#>
#> VALIDATION: RMSEP
#> Cross-validated using 10 random segments.
#>          (Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps
#> CV           1.16   0.8519  0.8645  0.7788  0.7686  0.7805  0.7691
#> adjCV        1.16   0.8501  0.8630  0.7762  0.7665  0.7760  0.7659
#>          7 comps 8 comps
#> CV           0.7635  0.7390
#> adjCV        0.7585  0.7352
#>
#> TRAINING: % variance explained
#>          1 comps 2 comps 3 comps 4 comps 5 comps 6 comps 7 comps
#> X            42.01   62.61   74.81   82.71   88.75   94.28   97.56
#> lpsa         47.04   47.67   58.61   59.45   60.73   61.66   64.21
#>          8 comps
#> X            100.00
#> lpsa         66.34
```

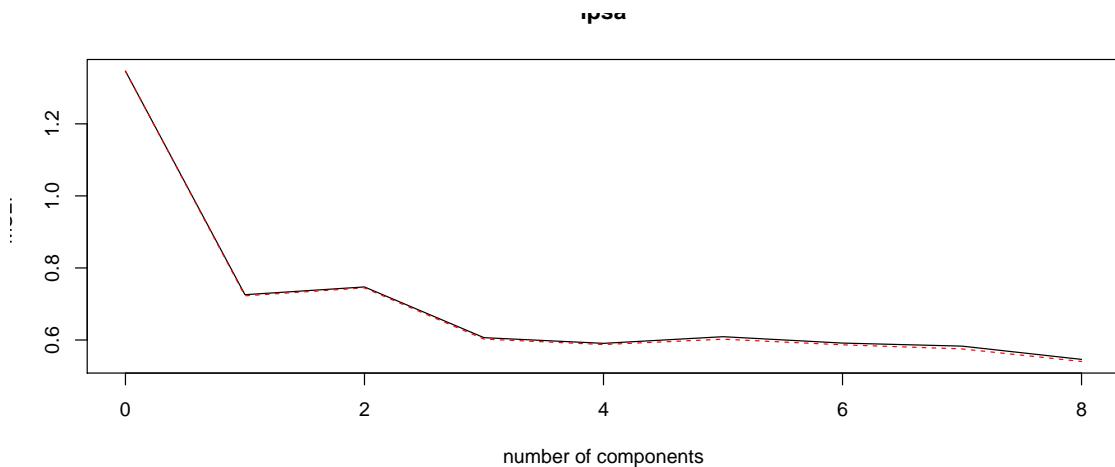
Wir sehen hier, dass wir mit 4 Hauptkomponenten schon 82.71% der Varianz erklären können. Danach steigt die Zunahme der erklärten Varianz nur noch langsam. Bis zu 6 Hauptkomponenten könnte man noch sinnvoll in das Modell mit aufnehmen. Mehr würde dem eigentlich Sinn widersprechen die Anzahl an Prädiktoren zu senken, um Multikollinearität zu reduzieren.

c)

In der Summary bekommen wir zu den Cross-Validations den RMSEP (Root Mean Squared Error) mit ausgegeben. Wir erhalten den kleinsten Fehler mit 0.7583 für 8 Hauptkomponenten. Dies ist verständlich, da mit allen Hauptkomponenten auch 100% der Varianz erklärt werden. Dies heißt aber nicht, dass das Modell dann auf einem anderen (unter Umständen stark unterschiedlichen) Datensatz immer noch die besten Ergebnisse liefert.

d)

```
validationplot(pcr, val.type = "MSEP")
```



Sinnvoll ist es eine Anzahl an Hauptkomponenten zu wählen, wo sich nicht mehr viel ändert. Dies kann man anhand eines Plot mittels Elbow-Methode erreichen. Sprich wir schauen, wo auf der Kurve etwa der Ellbogen liegt und nehmen dann den Wert an dieser stelle. Schauen wir uns den Plot an, so erscheint es sinnvoll sich für 3 Hauptkomponenten zu entscheiden.

e)

```
train = prostate[prostate$train == TRUE, ]
test = prostate[prostate$train == FALSE, ]
pcr = pcr(
  lpsa ~ lcavol + lweight + age + lbph + svi + lcp + gleason + pgg45,
  data = train,
  scale = TRUE
)

y.true = test$lpsa
y.pred = predict(pcr, test, ncomd = 3)

MSE = mean((y.true - y.pred)^2)
MSE = round(MSE, 3)
MSE
#> [1] 0.538
```

Wir erhalten einen MSE von 0.538. Vergleichen wir den Wert mit den Ergebnissen aus der letzten Übung, so stellen wir fest, dass der MSE etwa auf gleichem Niveau ist, wie der von Ridge/Lasso-Regression. Mittels Parametertuning lässt sich bei den anderen Regressionsverfahren ein besseres Ergebnis erreichen, aber dafür ist dafür auch in deutlich höherer Aufwand nötig. Die Ergebnisse der PCA sind für den gegebenen Aufwand sehr gut.

5.2 A2

a)

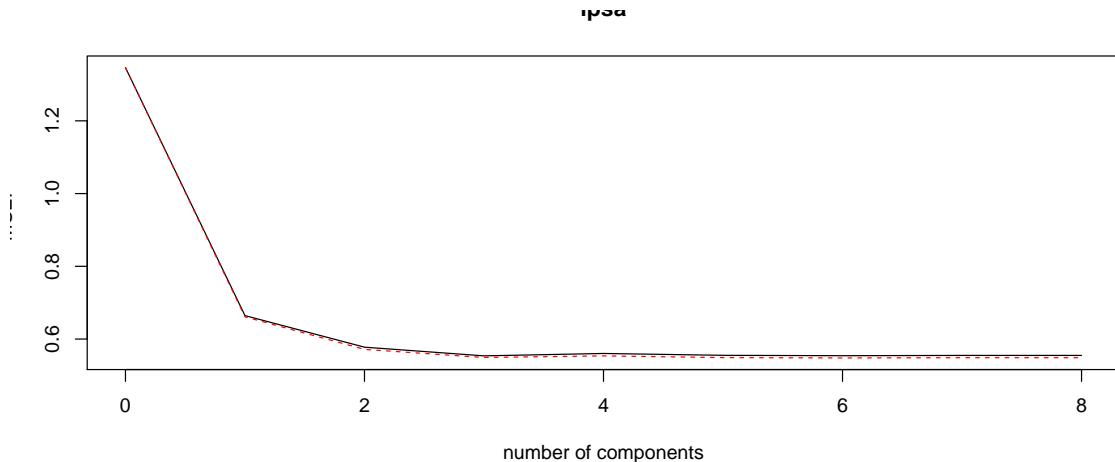
```
data = subset(
  prostate,
  select = c(lcavol, lweight, age, lbph, svi, lcp, gleason, pgg45, lpsa)
)
plsr = plsr(
  lpsa ~ lcavol + lweight + age + lbph + svi + lcp + gleason + pgg45,
  data = data,
  validation = "CV",
  scale = TRUE
)
summary(plsr)
#> Data:    X dimension: 97 8
#> Y dimension: 97 1
#> Fit method: kernelpls
#> Number of components considered: 8
#>
#> VALIDATION: RMSEP
#> Cross-validated using 10 random segments.
#>           (Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps
#> CV          1.16   0.8152   0.760   0.7442   0.7486   0.7452   0.7445
#> adjCV       1.16   0.8131   0.756   0.7411   0.7441   0.7409   0.7403
#>           7 comps 8 comps
#> CV          0.7450   0.7450
#> adjCV       0.7408   0.7408
#>
#> TRAINING: % variance explained
#>           1 comps 2 comps 3 comps 4 comps 5 comps 6 comps 7 comps
#> X          41.30   53.96   66.37   78.64   84.35   90.34   94.36
#> lpsa       54.62   63.55   65.31   66.12   66.32   66.34   66.34
#>           8 comps
#> X          100.00
#> lpsa       66.34
```

b)

Hier sinkt der Fehler schneller im Vergleich zur PCA. Der geringste Fehler ist 0.7411 für 3 Komponenten.

c)

```
validationplot(plsr, val.type = "MSEP")
```



Nach der Elbow-Methode bieten sich hier 2 Komponenten an.

d)

```
train = prostate[prostate$train == TRUE, ]
test = prostate[prostate$train == FALSE, ]
plsr = plsr(
  lpsa ~ lcavol + lweight + age + lbph + svi + lcp + gleason + pgg45,
  data = train,
  scale = TRUE
)

y.true = test$lpsa
y.pred = predict(plsr, test, ncomd = 2)

MSE = mean((y.true - y.pred)^2)
MSE = round(MSE, 3)
MSE
#> [1] 0.509
```

Wir erhalten einen MSE von 0.509. Dieser Wert ist besser als das Ergebnis von Aufgabe 1 und näher an den Ridge/Lasso-Regression Werten aus der vorherigen Übung aber nicht besser als diese. Das macht auch Sinn, da PLSR vom Aufwand her zwischen PSA und Ridge/Lasso-Regression liegt.

6 Zufallszahlen

6.1 A1

a)

```
gv <- runif(10000,min=0,max=1)
lambda <- 3
invexp <- -(1/lambda)*log(gv)
exp <- rexp(10000,lambda)

invpois = NULL
for (i in 1:10000){
  j <- 0
  U <- runif(1,0,1)
  Y <- -(1/lambda)*log(U)
  sum <- Y

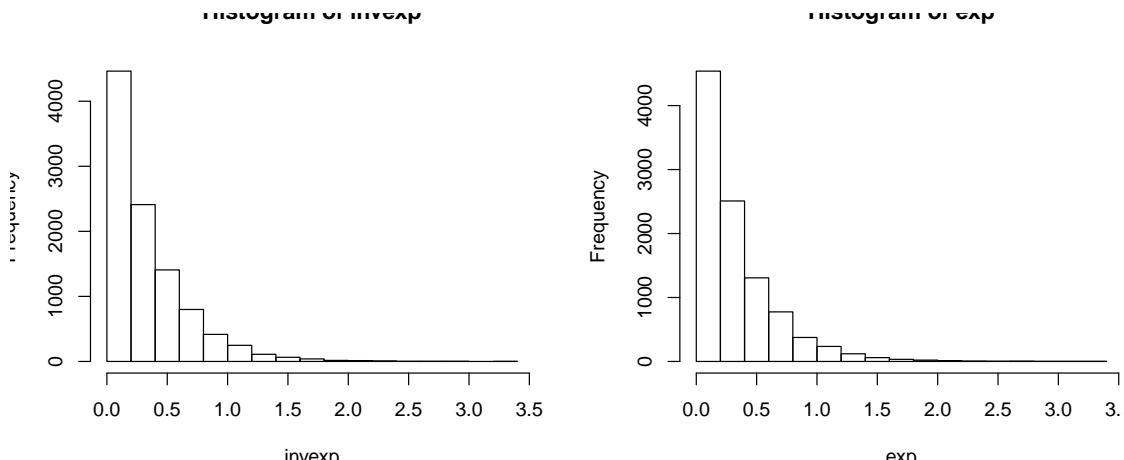
  while(sum < 1){
    U <- runif(1,min = 0,max = 1)
    Y <- -(1/lambda)*log(U)
    sum <- sum + Y
    j <- j + 1;
  }

  invpois <- rbind(invpois,j)
}

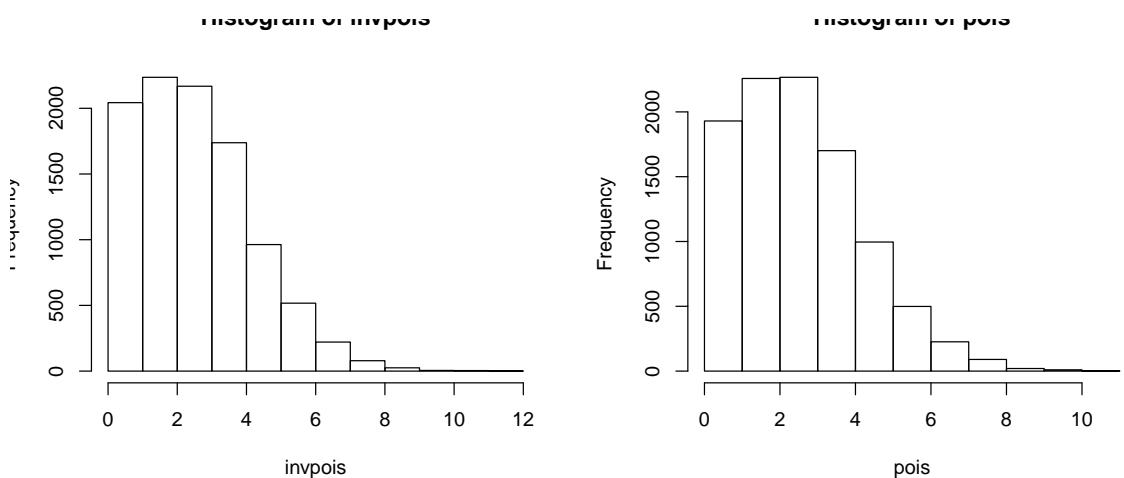
pois <- rpois(10000,lambda)

par(mfrow=c(1,2))

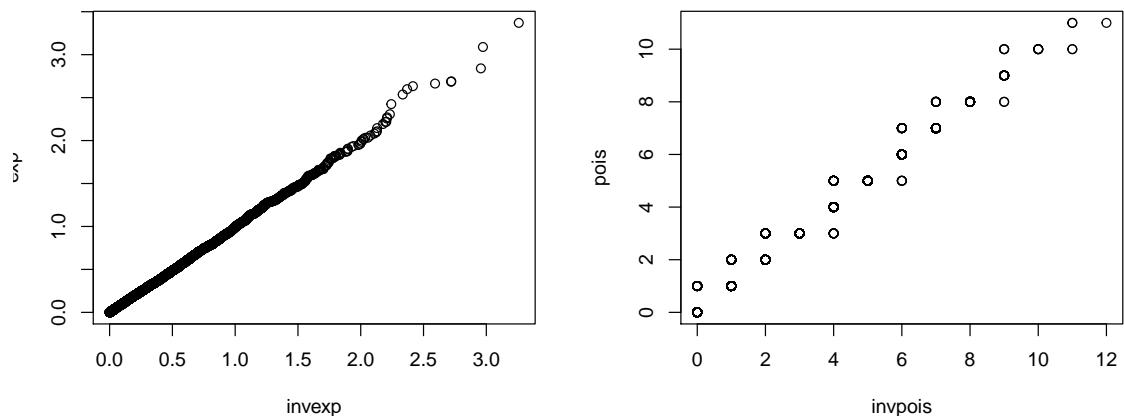
hist(invexp)
hist(exp)
```



```
hist(invexp)
hist(exp)
```



```
qqplot(invexp, exp)
qqplot(invpois, pois)
```



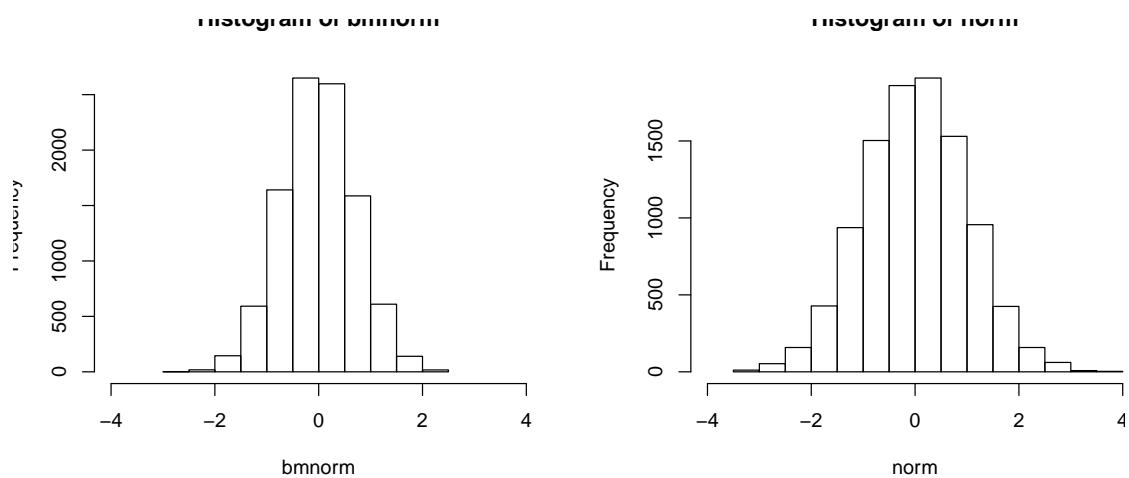
Es zeigt sich dass sich die Inversionsverfahren bei $n = 10000$ relativ gut den zugrundeliegenden Verteilungen annähern.

b)

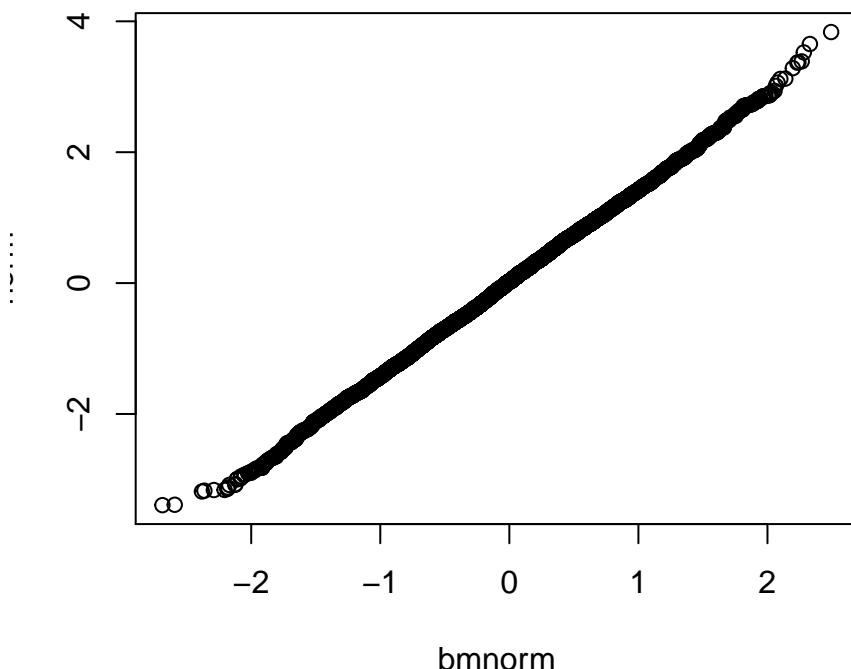
```
u <- runif(10000,0,1)
v <- runif(10000,0,1)

bmnorm = cos(2*pi*u)*sqrt(-log(v))
norm <- rnorm(10000)

par(mfrow=c(1,2))
hist(bmnorm,xlim=c(-4,4))
hist(norm, xlim=c(-4,4))
```



```
qqplot(bmnorm, norm)
```



Es zeigt sich eine Normalverteilung der transformierten gleichverteilten Zufallszahlen durch die Transformationsformel. Da die Standardabweichung der durch die Box-Müller Methode erzeugten Zufallszahlen 0.6987197 ist, ist das Histogramm im Vergleich zu den von R erzeugten normalverteilten Zufallszahlen "schmäler". Da rnorm standardmäßig standardnormalverteilte Zufallszahlen erzeugt, sieht man hier deutliche Unterschiede.

6.2 A2

a)

```
randomorg <- read.csv('res/randomorg.csv', header=FALSE)  
randomorg <- randomorg[, "V1"]
```

b)

```
midsqr = function(seed, length) {  
  ergebnis = c()  
  for(i in 1:length) {  
    value = seed * seed
```

```
    seed = (value %% 10000) %% 10000000
    ergebnis = c(ergebnis, seed/10000000)
}
return(ergebnis)
}
midsqr = midsqr(222222222222,10000)
```

c)

```
standardize <- function(x){(x-min(x))/(max(x)-min(x))}

baumannikov_zwirbler <- function(seed, n){
  if((seed %% 2) == 0){
    seed <- seed +1
  }
  start <- as.character((seed*n)^2)
  save <- c()
  for (i in 1:(n)){
    if (i == 1){
      x <- start
      k <- nchar(x)
      y <- as.numeric(substr(as.numeric(x), round(0.25*k), round(0.75*k)))
      save[i] <- y
    }
    else{
      x <- as.character((save[i-1]*n)^2)
      k <- nchar(x)
      y <- as.numeric(substr(as.numeric(x), round(0.25*k), round(0.75*k)))
      save[i] <- y
    }
  }
  return(standardize(save%%100))
}

baumannikov_zwirbler <- baumannikov_zwirbler(22222222, 10000)
```

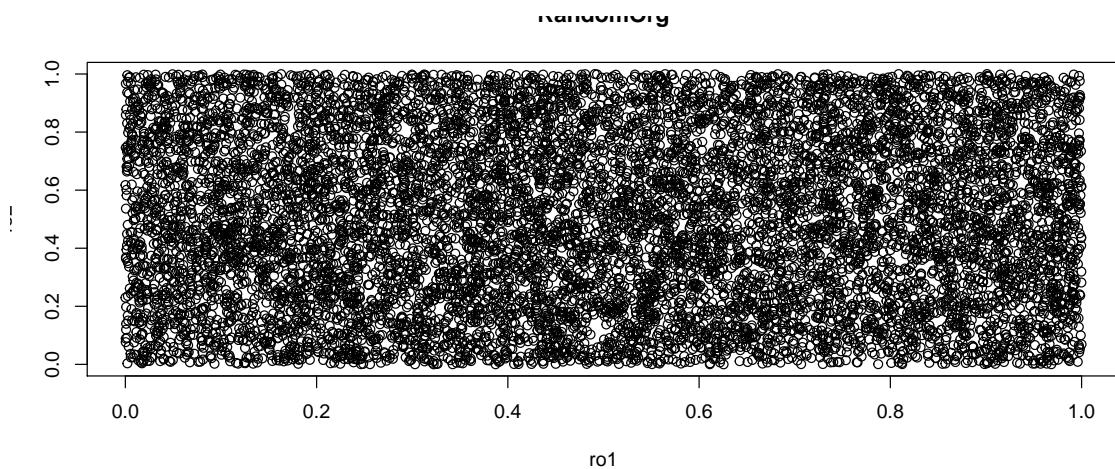
d)

```
set.seed(42, kind='Mersenne-Twister')
mersenne <- runif(10000)
```

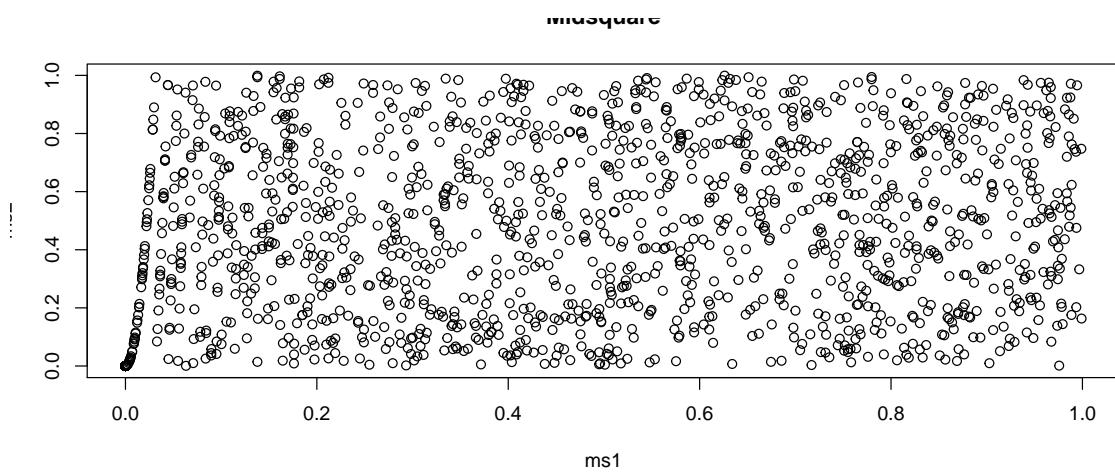
```
set.seed(42, kind='Super-Duper')
superduper <- runif(10000)
```

e)

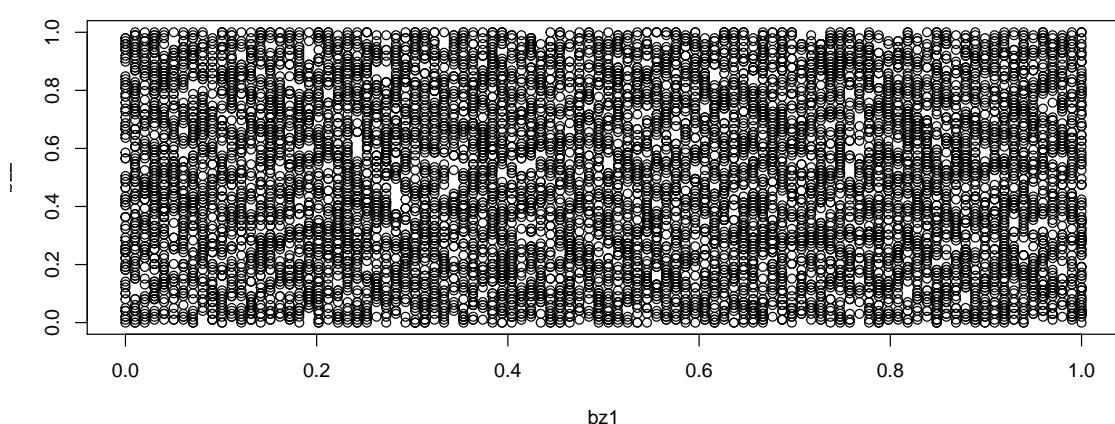
```
ro1 <- randomorg[-length(randomorg)]
ro2 <- randomorg[-1]
plot(ro1,ro2, main="RandomOrg")
```



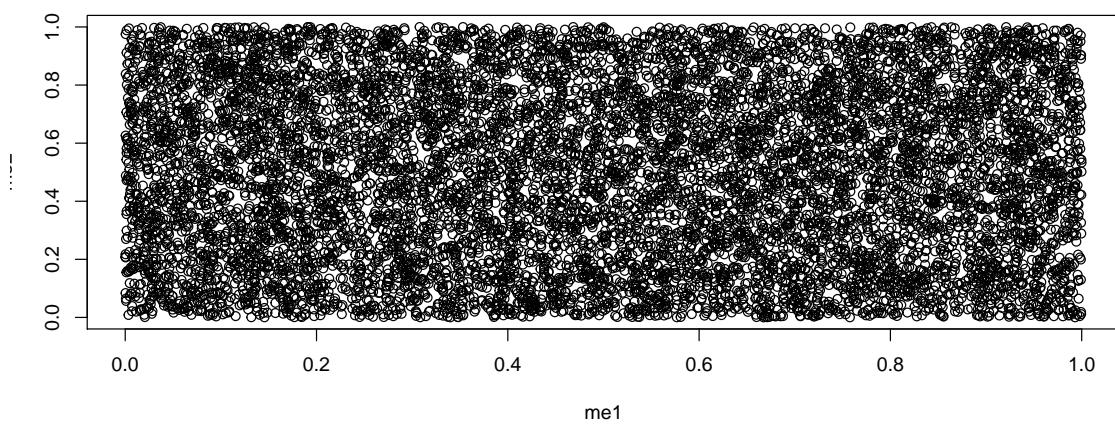
```
ms1 <- midsqr[-length(midsqr)]
ms2 <- midsqr[-1]
plot(ms1,ms2, main="Midsquare")
```



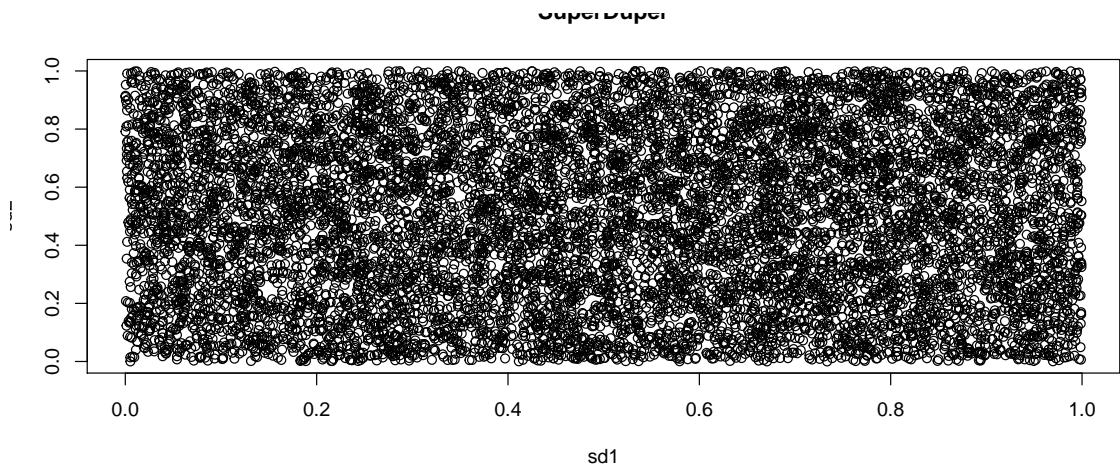
```
bz1 <- baumannikov_zwirbler[-length(baumannikov_zwirbler)]  
bz2 <- baumannikov_zwirbler[-1]  
plot(bz1, bz2, main="Baumannikov-Zwirbler")
```



```
me1 <- mersenne[-length(mersenne)]  
me2 <- mersenne[-1]  
plot(me1,me2, main="Mersenne")
```



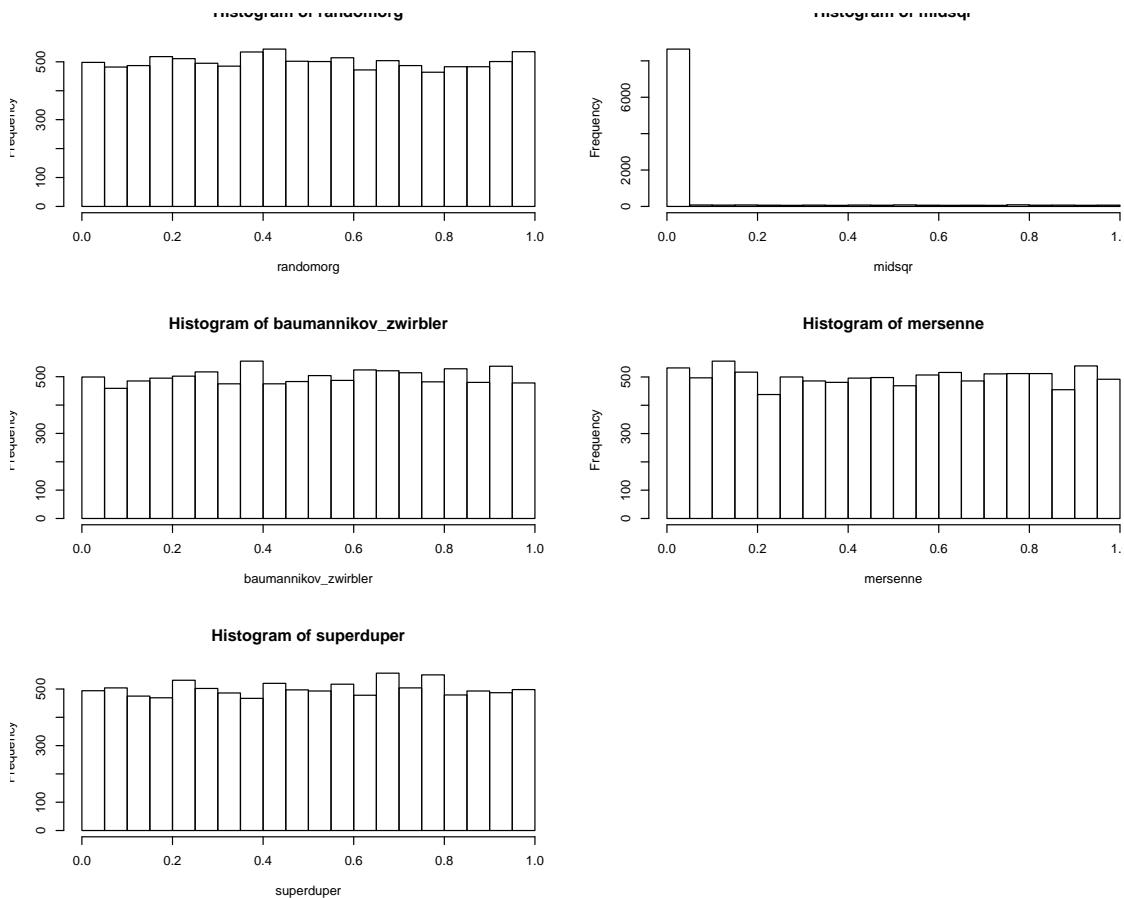
```
sd1 <- superduper[-length(superduper)]  
sd2 <- superduper[-1]  
plot(sd1,sd2, main="SuperDuper")
```



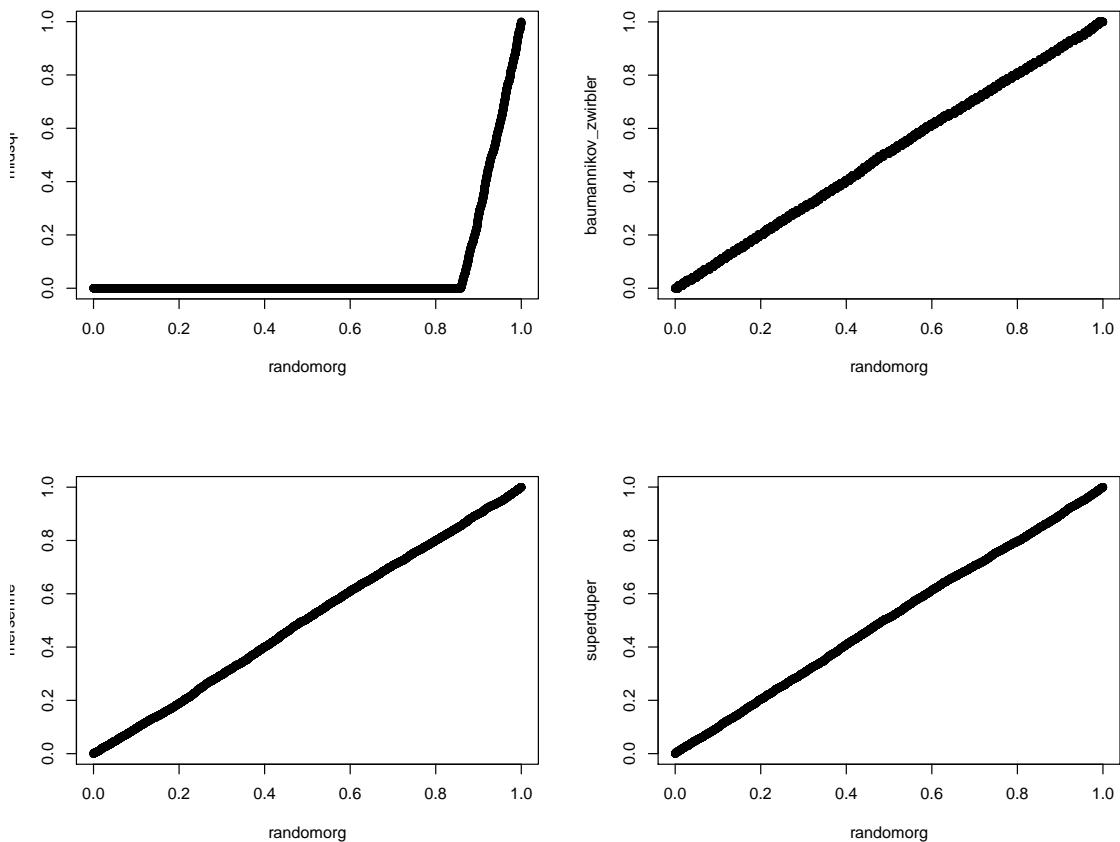
```
par(mfrow=c(3,2))
hist(randomorg)
hist(midsqr)
hist(baumannikov_zwirbler)
hist(mersenne)
hist(superduper)

par(mfrow=c(2,2))
```

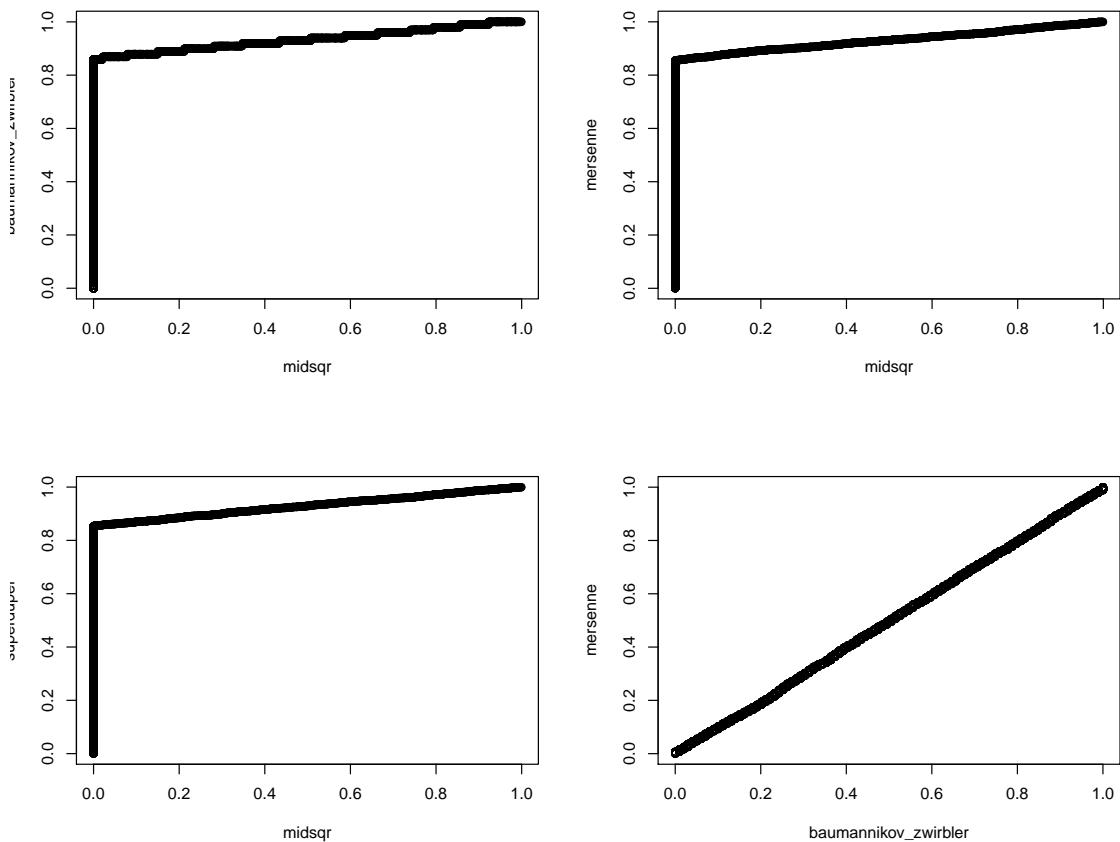
6 Zufallszahlen



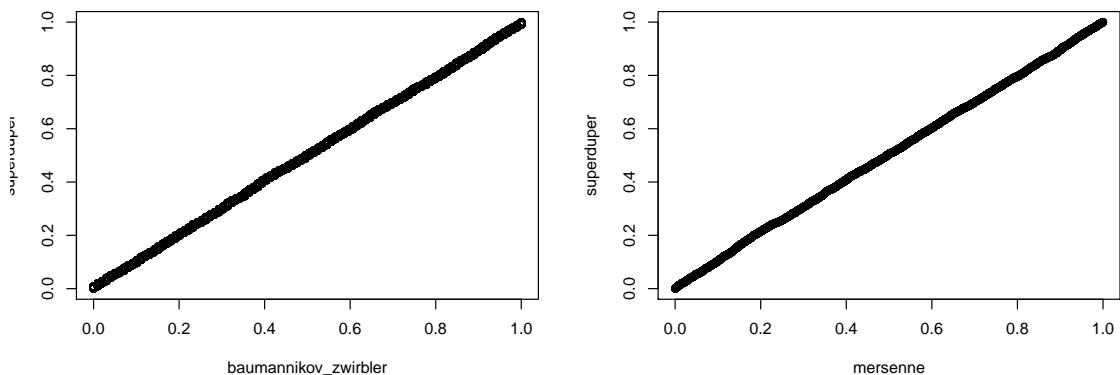
```
qqplot(randomorg,midsqr)
qqplot(randomorg,baumannikov_zwirbler)
qqplot(randomorg,mersenne)
qqplot(randomorg,superduper)
```



```
qqplot(midsqr,baumannikov_zwirbler)
qqplot(midsqr,mersenne)
qqplot(midsqr,superduper)
qqplot(baumannikov_zwirbler,mersenne)
```



```
qqplot(baumannikov_zwirbler,superduper)
qqplot(mersenne,superduper)
```



Die Scatterplots der Zweier-Tupel zeigen eine zufällige Verteilung der Zufallszahlen bis auf die Methode des Mittsquadrat Verfahrens von Neumann. Bei diesem zeigt sich eine deutliche Überverteilung der Zahlen bei (0,0). Die Histogramme bestätigen dieses Bild. Auch die QQPlots zeigen die Schwäche des Mittsquadrat Verfahrens von Neumann im Vergleich zu den anderen Zufallsgeneratoren.

7 Monte-Carlo-Simulation

7.1 A1

```

zi = function(xi, yi) {
  out = 4 * ifelse(xi^2 + yi^2 <= 1, 1, 0)
  return(out)
}

simpi = function(n) {
  xi = runif(n, -1, 1)
  yi = runif(n, -1, 1)
  z = zi(xi, yi)
  out = mean(z)
  return(out)
}

pimat = matrix(NA, nrow = 8, ncol = 3)
for (k in seq(8)) {
  z = simpi(10^k)
  pimat[k, 1] = k
  pimat[k, 2] = z
  pimat[k, 3] = round(abs(pi - z), 6)
}
df_pi = data.frame(pimat)
colnames(df_pi) = c("k", "z", "e")
kable(df_pi, align = "c")

```

k	z	e
1	3.600000	0.458407
2	3.200000	0.058407
3	3.200000	0.058407
4	3.180800	0.039207
5	3.149520	0.007927
6	3.141424	0.000169
7	3.142315	0.000722
8	3.141721	0.000129

Um mit Chebyshev abschätzen zu können benötigen wir Erwartungswert und Varianz der Zufallsvariablen z_i . Der Erwartungswert sollte mit π übereinstimmen. Wir rechnen nach. Sei dazu z_1 die Approximation von π mittels des Einheitskreises im Einheitsquadrat eingeschränkt auf den 1. Quadranten.

$$E(z_i) = E(4z_1) = 4 E(z_1) = 4 \int_0^1 \sqrt{1-x^2} dx = 4 \frac{\pi}{4} = \pi \quad (1)$$

$$V(z_i) = V(4z_1) = 16 E(z_i) \quad (2)$$

$$= 16 \int_0^1 \sqrt{1-x^2}^2 dx - 16 \left(\int_0^1 \sqrt{1-x^2} dx \right)^2 \quad (3)$$

$$= 16 \int_0^1 1-x^2 dx - 16 \frac{\pi^2}{16} = 16 \left[x - \frac{1}{3}x^3 \right]_0^1 - \pi^2 \quad (4)$$

$$= \frac{32}{3} - \pi^2 \approx 0.797 \quad (5)$$

Damit erhalten wir dann für \bar{z} :

$$E(\bar{z}) = E\left(\frac{1}{n} \sum_{i=1}^n z_i\right) = \frac{1}{n} n E(z_i) = E(z_i) = \pi \quad (6)$$

$$V(\bar{z}) = V\left(\frac{1}{n} \sum_{i=1}^n z_i\right) = \frac{1}{n^2} \left(\sum_{i=1}^n z_i \right)^2 = \frac{n}{n^2} V(z_i) = \frac{1}{n} V(z_i) \approx \frac{0.797}{n} \quad (7)$$

Nach Chebyshev gilt nun für einen beliebigen Fehler ε und $\sigma^2 = V(z_i)$:

$$P(|\bar{z} - \pi| \geq \varepsilon) \leq \frac{V(\bar{z})}{\varepsilon^2} \quad (8)$$

$$\implies P(|\bar{z}(n) - \pi| \geq \varepsilon) \leq \frac{\sigma^2}{n^2 \varepsilon^2} = \frac{\sigma}{n \varepsilon} \quad (9)$$

Nun gilt es zu bedenken, dass die Abschätzung zwar für alle $\varepsilon > 0$ gilt, aber nur für $n\varepsilon^2 > \sigma^2$ wirklich Sinn macht. Sprich es macht nur Sinn zu prüfen, wie hoch die Chance ist, dass man ein Vielfaches der Standardabweichung vom Mittelwert abweicht. Wie man an der Abschätzung erkennen kann, sinkt die Genauigkeit, dass wir um mehr als einen gegebenen Fehler abweichen mit zunehmenden n . Das heißt insbesondere, dass mit höheren n der Fehler sinkt. Dies können wir anhand der Tabelle bestätigen.

7.2 A2

a)

```
simsnd = function(n) {  
  x = runif(n, -1.96, 1.96)  
  phi = dnorm(x)  
  out = 2 * 1.96 * mean(phi)  
  return(out)  
}  
  
sndmat = matrix(NA, nrow = 8, ncol = 3)  
for (k in seq(8)) {  
  z = simsnd(10^k)  
  sndmat[k, 1] = k  
  sndmat[k, 2] = z  
  sndmat[k, 3] = round(abs(0.95 - z), 6)  
}  
df_snd = data.frame(sndmat)  
colnames(df_snd) = c("k", "z", "e")  
kable(df_snd, align="c")
```

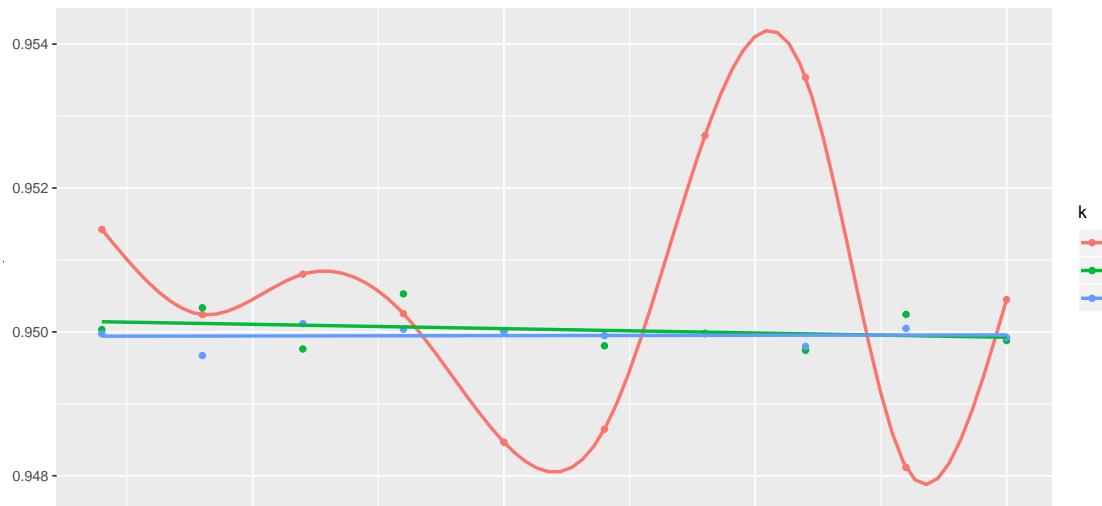
k	z	e
1	1.1504552	0.200455
2	0.9626102	0.012610
3	0.9450319	0.004968
4	0.9482917	0.001708
5	0.9508406	0.000841
6	0.9499435	0.000056
7	0.9499549	0.000045
8	0.9500460	0.000046

Hier sehen wir , dass es ab etwa 10^6 Versuchen sehr präzise wird.

b)

```
nexp = 10
M = matrix(NA, nrow = nexp, ncol = 4)
for (k in seq(5,7)) {
  for (i in seq(nexp)) {
    z = simsnd(10^k)
    M[i, 1] = i
    M[i,k - 3] = z
  }
}
df = data.frame(M)
colnames(df) = c("i", "5", "6", "7")
ggdf = gather(df, "k", "P", seq(2,4))
gg = ggplot(
  data = ggdf,
  mapping = aes(x = i, y = P, group = k, col = k)
)
gg = gg + xlab("Iteration") + ylab("P")
gg = gg + geom_point()
gg = gg + stat_smooth(method = "gam", formula = y ~ s(x, bs='cr'), se = FALSE)
gg = gg + theme(
  axis.title.x=element_blank(),
  axis.text.x=element_blank(),
  axis.ticks.x=element_blank()
)
```

gg



In der Abbildung können wir erkennen, dass für $k = 5 \implies n = 10^5$ es noch eine signifikante Streuung um den erwarteten Wert von $P = 0.95$ gibt. Um dies graphisch besser darzustellen haben wir einen kubischen Regressionspline durch die Punkte gezogen. Man erkennt für $k = 5$ deutliche Abweichungen, während für $k = \{6, 7\}$ der Spline eher eine Gerade am erwarteten Wert ist. Erinnern wir uns an die Abschätzung mit Chebyshev aus Aufgabe 1, so sollten wir auch hier eine ähnliche Abschätzung erhalten (da analoges Vorgehen) und da die Wahrscheinlichkeit von Mittelwert abzuweichen mit dem Produkt aus Fehler und n sinkt, braucht es schon ein hohes n , um möglichst kleine Fehler zu garantieren.

7.3 A3

```

u1 = function(x,y) {
  if (x^2 / 49 + y^2 / 9 - 1 <= 0) {
    if (abs(x) >= 4 && y >= -(3 * sqrt(33))/7 && y <= 0) {
      return(1)
    } else {
      if (abs(x) >= 3 && y >= 0) {
        return(1)
      }
    }
  }
  return(0)
}

u2 = function(x,y) {
  if (y >= -3 && y <= 0 && -4 <= x && x <= 4) {
    if (-(3*sqrt(33)-7)*x^2/112 + abs(x)/2
        + sqrt(1-(abs(abs(x)-2)-1)^2) - y - 3 <= 0) {
      return(1)
    }
  }
  return(0)
}

u3 = function(x,y) {
  if (y >= 0 && 3/4 <= abs(x) && abs(x) <= 1) {
    if (-8*abs(x) - y + 9 >= 0) {
      return(1)
    }
  }
  return(0)
}

u4 = function(x,y) {
  if (1/2 <= abs(x) && abs(x) <= 3/4 && 3 * abs(x) - y + 3/4 >= 0 && y >= 0) {
    return(1)
  }
  return(0)
}

u5 = function(x,y) {
  if (abs(x) <= 1/2 && y >= 0 && 9/4 - y >= 0) {
    return(1)
  }
  return(0)
}

```

```
u6 = function(x,y) {
  if (1 <= abs(x) && abs(x) <= 3 && y >= 0) {
    if (-abs(x)/2 - 3/7 * sqrt(10)*sqrt(4 - (abs(x) -1)^2) -y + 6*sqrt(10)/7 + 3/2 >= 0)
      return(1)
    }
  return(0)
}

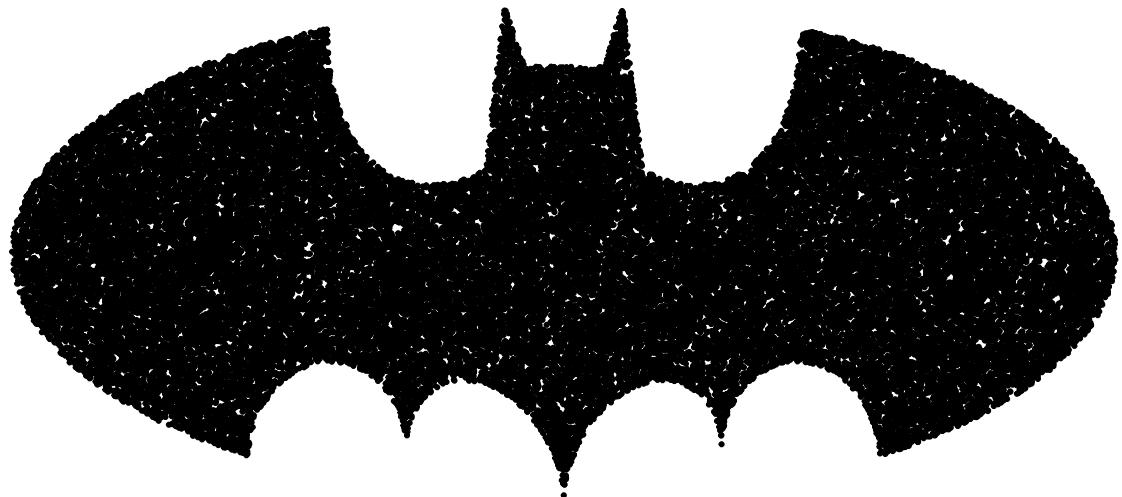
u = function(x,y) {
  if (u1(x,y) == 1) {
    return(1)}
  if (u2(x,y) == 1) {
    return(1)}
  if (u3(x,y) == 1) {
    return(1)}
  if (u4(x,y) == 1) {
    return(1)}
  if (u5(x,y) == 1) {
    return(1)}
  if (u6(x,y) == 1) {
    return(1)}
  return(0)
}

batman = function(n) {
  x = runif(n, -7, 7)
  y = runif(n, -4, 4)
  df = data.frame(x = double(), y = double(), z = double())
  for (i in seq(n)) {
    z = u(x[i], y[i])
    df = rbind(df, c(x[i], y[i], z))
  }
  colnames(df) = c("x", "y", "z")
  return(df)
}

n = 10^5
df = batman(n)
```

a)

```
gg = ggplot(  
  data = df[df$z == 1],  
  mapping = aes(  
    x = x,  
    y = y  
)  
)  
gg = gg + geom_point(size = 1)  
gg = gg + theme(  
  axis.title.x=element_blank(),  
  axis.text.x=element_blank(),  
  axis.ticks.x=element_blank(),  
  axis.title.y=element_blank(),  
  axis.text.y=element_blank(),  
  axis.ticks.y=element_blank(),  
  panel.grid.major = element_blank(),  
  panel.grid.minor = element_blank(),  
  panel.background = element_blank()  
)  
gg
```



Es ist Batman. Für `geom_point(size = 1)` sogar Space Batman.

b)

```
meanz = mean(df$z)
A = 14 * 8 * meanz
A = round(A, 2)
```

Der Space Batman hat einen approximativen Flächeninhalt von $A = 48.47$.