h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

- Booklet -

Computational Statistics

Von:
Büsra KARAOGLAN
Matrikelnummer: 754331

Prüferin:
Prof. Dr. Jutta GROOS

Fachbereich Mathematik und Naturwissenschaften
Masterstudiengang Business Mathematics

Sommersemester 2018

27. Juli 2018

Inhaltsverzeichnis

Einleitung

Diese Ausarbeitung dient der Vorleistung des Fachbereichs Business Mathematics an der Hochschule Darmstadt für das Modul *Computational Statistics*, gelesen von Prof. Dr. Jutta Groos. Sie beinhaltet Aufgaben die mit Hilfe der Vorlesungsunterlagen und mit der Softwareprogramm R Markdown bearbeitet wurden. Es werden in dieser Ausarbeitung unterschiedliche Themenbereiche erarbeitet. Diese Themen werden in der Vorlesung sehr ausführlich erläutert. Das theoretische beziehungsweise mathematische Hintergrundwissen der Themen sind somit Voraussetzungen für das Verständnis der Aufgaben. Allerdings werden an erforderlichen Stellen ergänzende Bemerkungen bezüglich der Thematik gemacht. Die folgende Arbeit bezieht sich hauptsächlich auf die Vorlesungunterlagen von Prof. Dr. Jutta Groos, Prof. Dr. Horst Zisgen und Dr. Antje Jahn. Falls es andere Quellen verwendet wurden, werden diese an den ensprechenden Stellen gekennzeichnet.

1 Aufgabenblatt 1: Lineare Regression

Für das erste und zweite Aufgabenblatt wird der in der Vorlesung ausgeteilte *Donald* Datensatz verwendet. Der Datensatz umfasst Messdaten von 150 Beobachtungen mit fünf unabhängige Variablen Geschlecht, Alter, Minderheit, Fremdenfeindlich und IQ. Das Ziel ist eine Vorhersage über die abhängige Variable Trump bzw. die Wahlentscheidung einer Personen zu treffen. Dabei wird die Zustimmung für Trump als Presidenten der Vereinigten Staaten in Prozent angegeben. Da hier die abhängige Variable nicht nur von einer unabhängigen Variablen sondern von mehreren unabhängigen Variablen beschrieben wird, handelt es sich um eine **multiple lineare Regression**.

Zu Beginn müssen erstmal die Daten in R eingelesen werden.

```
load("Donald.RData")
```

Der Modellansatz für multiple lineare Regression lautet für $i = 150$ hier:

$$Y_i := \beta_0 + \beta_1 X_{i1} + \cdots + \beta_5 X_{i5} + \varepsilon_i$$

```
MLR <- lm(Donald_1$Trump ~ Donald_1$Geschlecht + Donald_1$Alter
          + Donald_1$Minderheit + Donald_1$Fremdenfeindlich + Donald_1$IQ)

summary(MLR)
##
## Call:
## lm(formula = Donald_1$Trump ~ Donald_1$Geschlecht + Donald_1$Alter +
##     Donald_1$Minderheit + Donald_1$Fremdenfeindlich + Donald_1$IQ)
##
## 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 ***
## Donald_1$Geschlecht         5.75572    0.99977   5.757 4.97e-08 ***
## Donald_1$Alter              0.18153    0.03049   5.954 1.91e-08 ***
## Donald_1$Minderheit        -6.57586    1.82843  -3.596 0.000443 ***
## Donald_1$Fremdenfeindlich   9.34984    0.16325  57.272  < 2e-16 ***
## Donald_1$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
```

Die p-Werte sind kleiner als jedes vernünftige Signifikanzniveau (man hätte evtl. $\alpha = 5\%$ nehmen können). Die Koeffizienten von Geschlecht, Alter, Minderheit, Fremdenfeindlich und IQ sind daher signifikant von 0 verschieden. Dies beruht daher, weil man auf die folgenden Hypothesen getestet hat:

$$\begin{aligned} H0 &: \beta_1 = \beta_2 = \cdots = \beta_5 = 0 \\ H1 &: \textit{Mindestens ein } \beta_l \neq 0,\ 1 \leq l \leq 5 \end{aligned}$$

Mit Hilfe des Bestimmtheitsmaßes R^2 kann gesagt werden wie gut das Modell auf die Daten passt. Das R^2 beträgt hier über 97 %, das heißt man kann also 97 % der Varianz der Variable Trump durch das oben beschriebene Modell erklären. Zusätzlich kann noch das korrigierte R^{2*} = 97,03 % betrachtet werden. Dabei besteht das aus R^2 und einem Strafterm, welche durch die Hinzunahme der unabhängigen Variablen steigt.

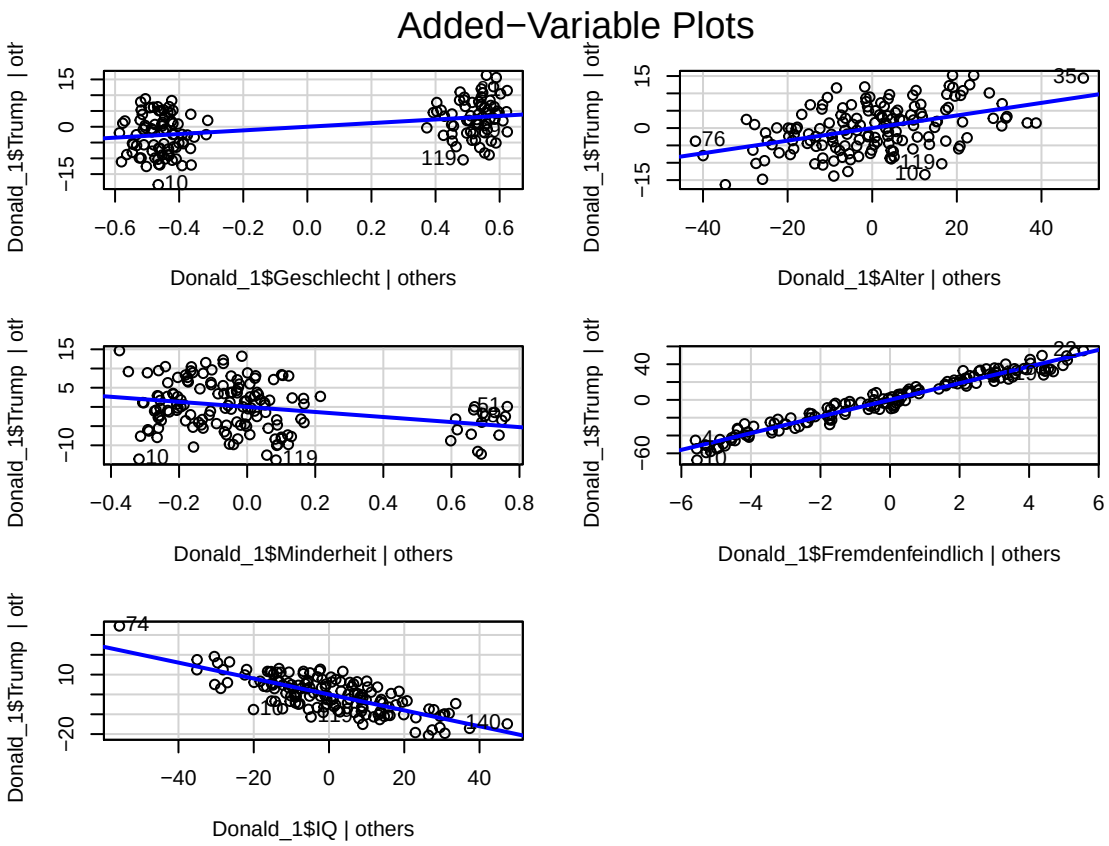
Zusätzlich sollten die Parameter, also die Koeffizienten β, standardisert werden. Da die Konstante, hier Intercept, bei der Analyse der Regressionsgleichung nicht interessant ist, wird sie bei der Standardisierung auf Null gesetzt und die Variablen können nun einfacher miteinander verglichen werden. Es ist erkennbar, dass die Variable Fremdenfeindlichkeit den größten (positiven) Einfluss erreicht hat.

```
library(lm.beta)

MLR.beta <- lm.beta(MLR)

coef(MLR.beta)
##                (Intercept)       Donald_1$Geschlecht
##                 0.00000000                0.08190695
##             Donald_1$Alter       Donald_1$Minderheit
##                 0.08513441               -0.05790182
## Donald_1$Fremdenfeindlich                Donald_1$IQ
##                 0.91808061               -0.19120337
```

Zuletzt werden die Voraussetzungen bzw. Modellannahmen geprüft, da die Schätzer der multiple lineare Regression mit Hilfe der kleinsten Quadrate Methode bestimmt wird und somit bestimmte Optimalitätsbedingugen (siehe Best Linear Unbiased Estimator) erfüllen muss. Wenn die Annahmen nicht stimmen deutet dies darauf hin, dass das gewählte Regressionsmodell den tatsächlichen Zusammenhang nicht vollständig erklärt.

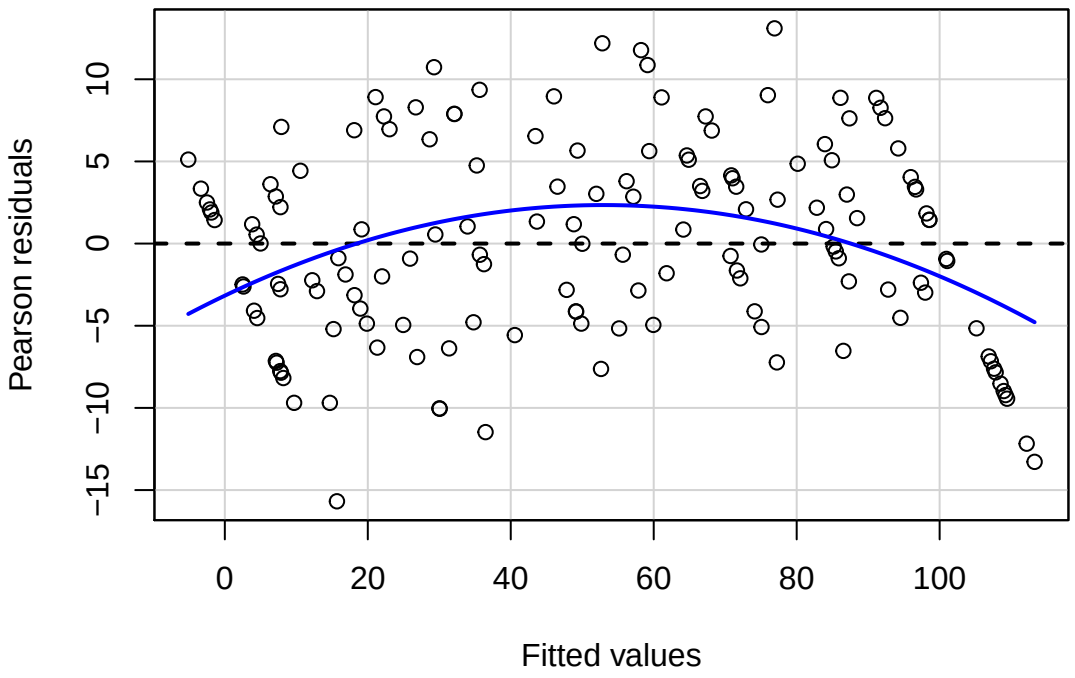
Zunächst wird betrachtet ob die Annahme eines linearen Zusammenhangs verletzt ist. Da hier fünf unabhängige Variablen einen Einfluss auf die abhängige Variable haben, kann diese nicht in einem Streudiagramm betrachtet werden. Nichtsdestotrotz kann das partielles Regressionsdiagramm, in dem der Einfluss von einer unabhängigen Variable auf die Zielvariable abgebildet wird, betrachten. Hier ist es deutlich, dass nur bei der Variable Fremdenfeindlich ein klarer Zusammenhang erkennbar ist.

```
library(car)

avPlots(MLR)
```



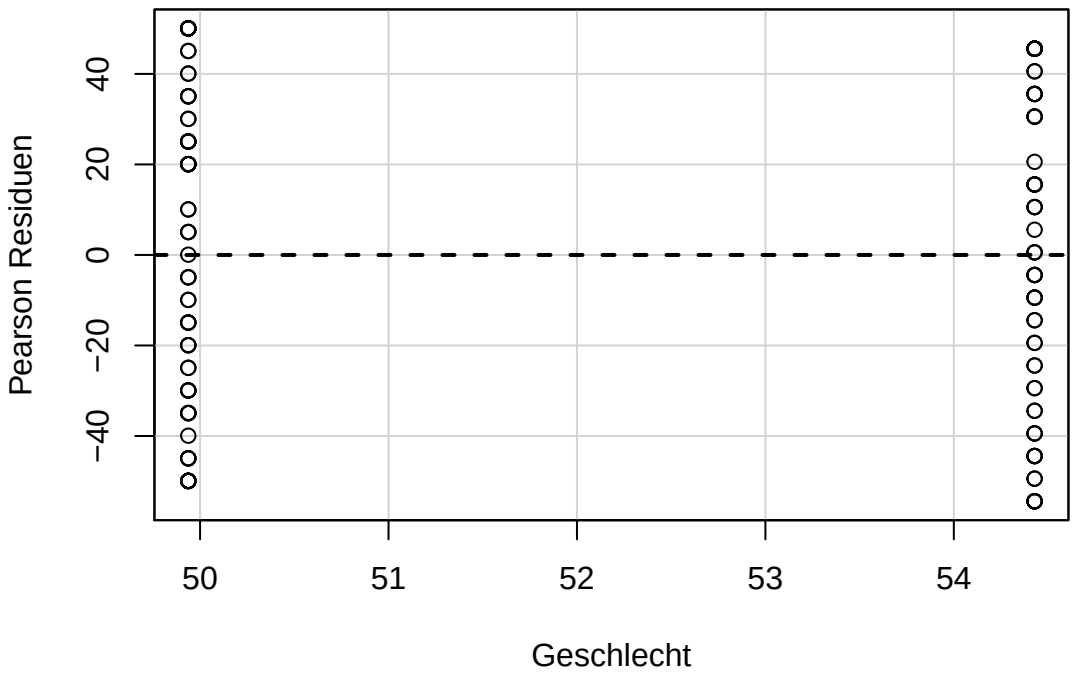
Jedoch kann auch mit Hilfe der Residuenplot (Fitted values gegen Pearson Residuen) die Linearitätsannahme überprüft werden. Hier wird eine zweidimensionale Grafik mit $\hat{Y}_i := \hat{\beta}_0 + \hat{\beta}_1 X_{i1} + \cdots + \hat{\beta}_5 X_{i5}$ auf der x-Achse und die Residuen $\varepsilon_i = Y_i - \hat{Y}_i$ auf der y-Achse erstellt. Hier ist keine lineare oder nichtlineare Trend erkennbar und die Punkte sind offensichtlich unsystematisch gestreut. Zudem verteilen sich die Residuen ungefähr in einem gleichbleibend dickem horizontalen Band und somit keine Heteroskedastizität erkennbar. Zudem kann jede Einflussgröße $X_1, \ldots, X_5$ einzeln abgebildet und auf Homoskedastizität überprüft werden.

```
residualPlots(MLR)
```

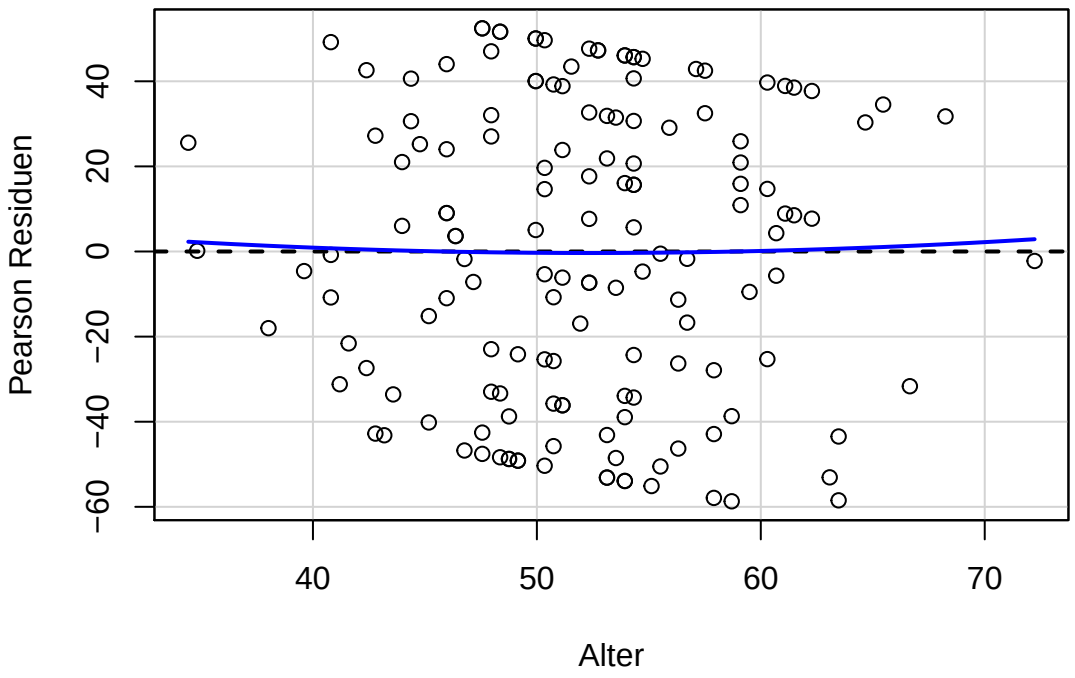



```
##            Test stat Pr(>|Test stat|)
## Tukey test   -4.6752        2.936e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

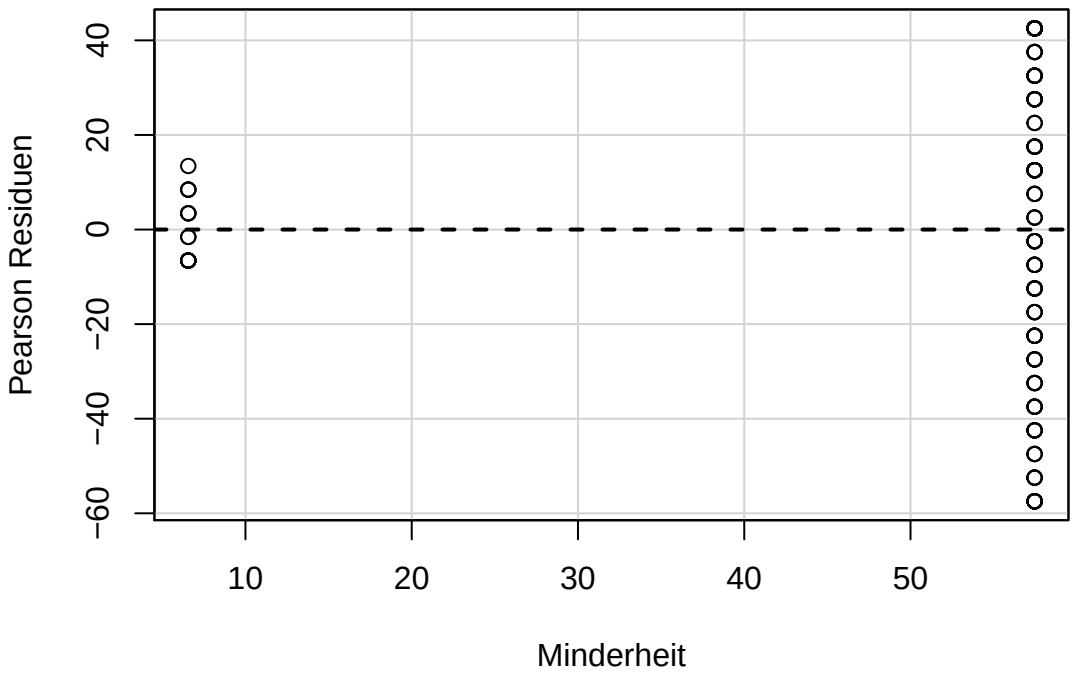
```
MLR1 <- lm(Donald_1$Trump ~ Donald_1$Geschlecht)
MLR2 <- lm(Donald_1$Trump ~ Donald_1$Alter)
MLR3 <- lm(Donald_1$Trump ~ Donald_1$Minderheit)
MLR4 <- lm(Donald_1$Trump ~ Donald_1$Fremdenfeindlich)
MLR5 <- lm(Donald_1$Trump ~ Donald_1$IQ)

residualPlots(MLR1, xlab="Geschlecht", ylab="Pearson Residuen")
```



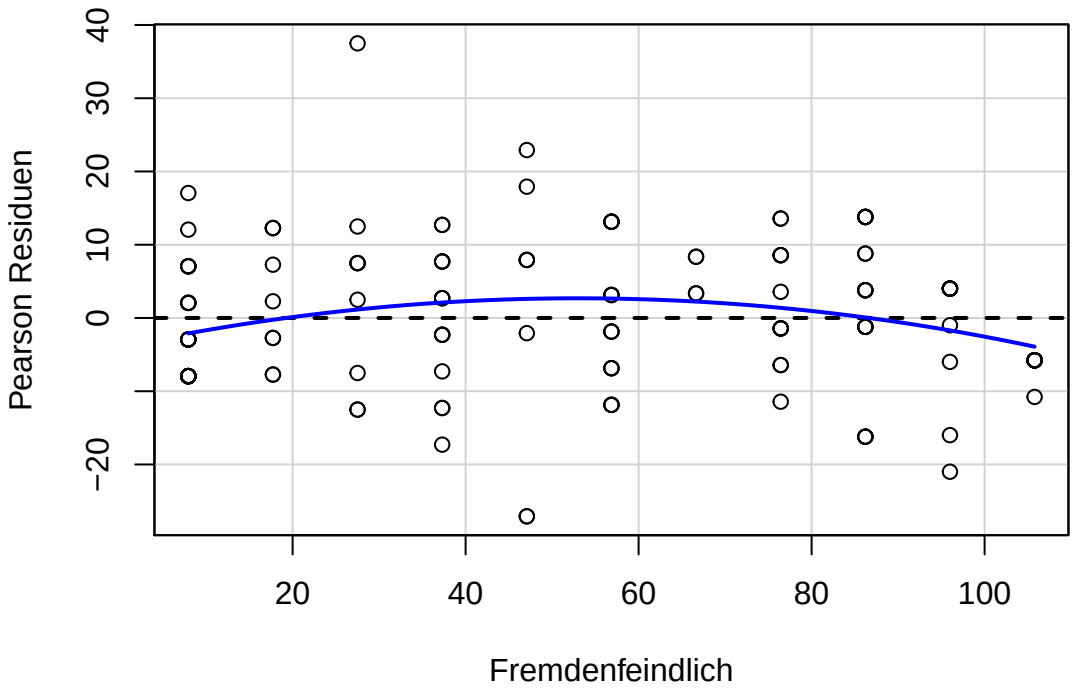

```
residualPlots(MLR2, xlab="Alter", ylab="Pearson Residuen")
```



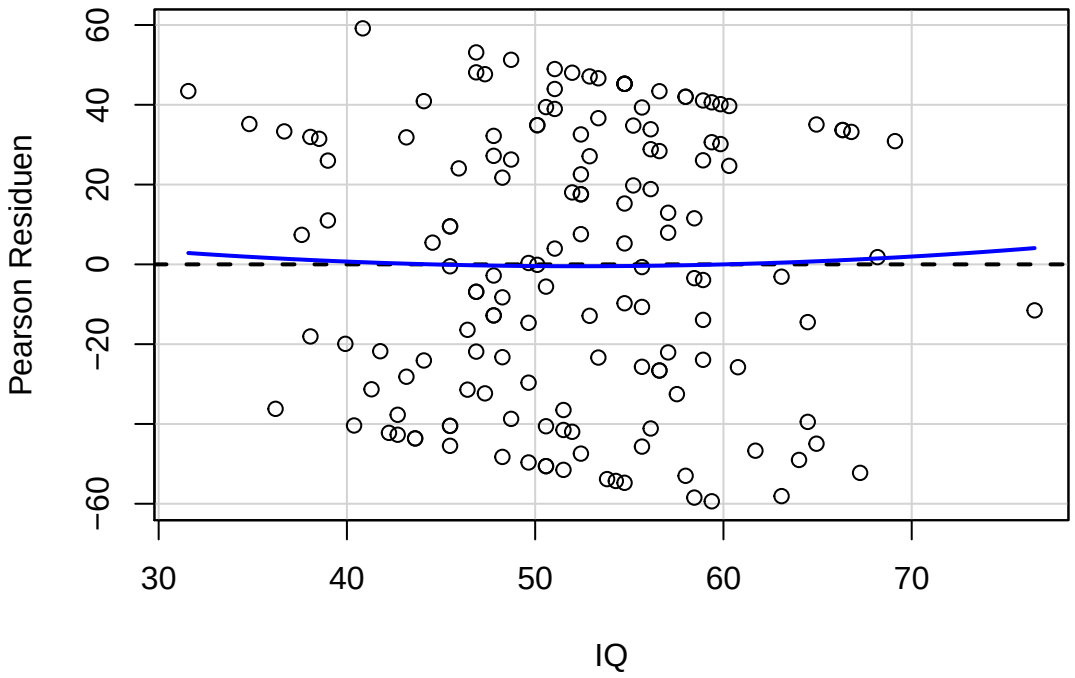

```
##            Test stat Pr(>|Test stat|)
## Tukey test    0.1853            0.853
residualPlots(MLR3, xlab="Minderheit", ylab="Pearson Residuen")
```




```
residualPlots(MLR4, xlab="Fremdenfeindlich", ylab="Pearson Residuen")
```



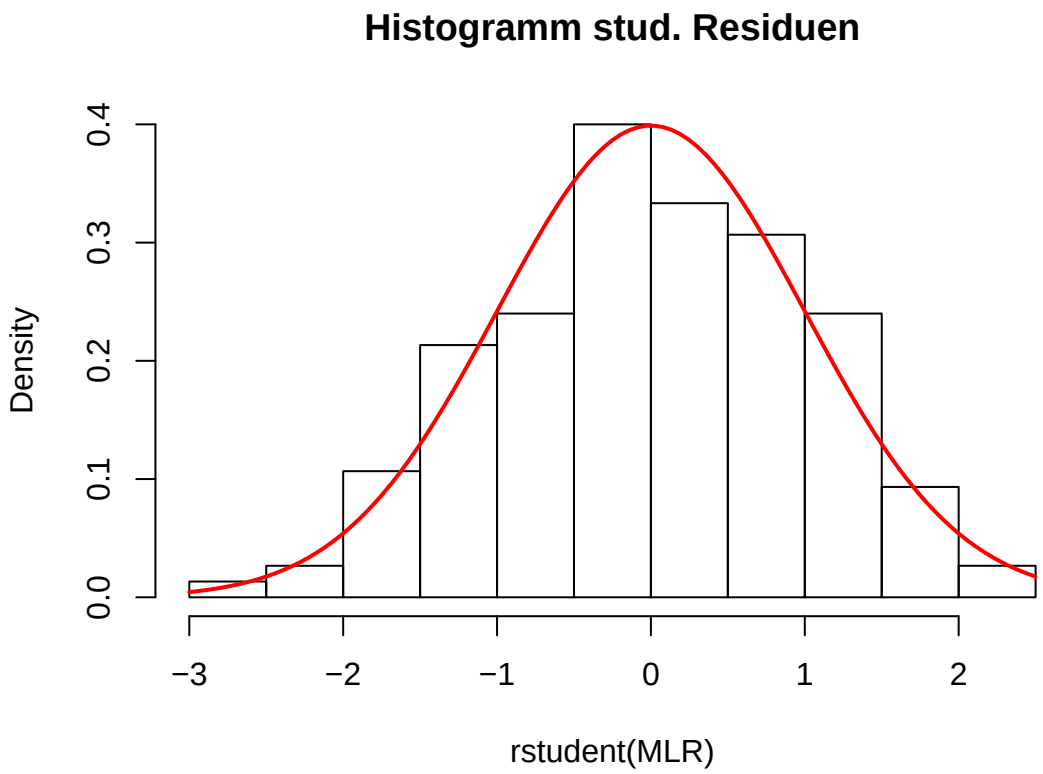

```
##            Test stat Pr(>|Test stat|)
## Tukey test   -2.6275         0.008602 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
residualPlots(MLR5, xlab="IQ", ylab="Pearson Residuen")
```




```
##            Test stat Pr(>|Test stat|)
## Tukey test    0.2378            0.812
```

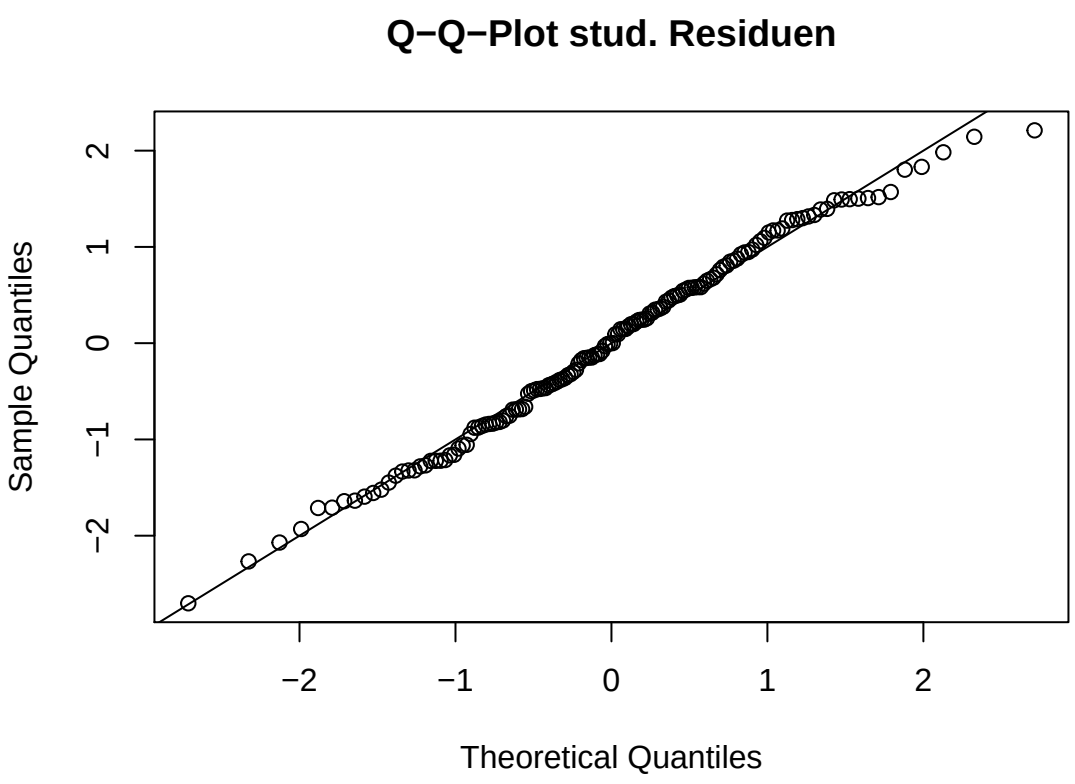
Nun wird die Normalverteilungsannahme der Residuen geprüft. Dies kann auf sehr unterschiedliche Weisen erfolgen. Hier wird zunächst Histogramm und QQ-Plot erstellt und mit Hilfe dieser Graphiken argummentiert. Es gibt in R unterschiedliche Residuen und einer von denen sind die studentisierte Residuen. Diese werden in intern und extern studentisierte Residuen unterteilt. Die R-Funktionen rstudent() bei der Erstellung der Histogrammvgibt die extern studentisierten Residuen wieder. Näheres zustudentisierte Residuen kann in [HT13] nachgelesen werden. Die Form des Histogramms sollte nun möglichst der standard Normalverteilungskurve entsprechen.

```
hist(rstudent(MLR), main="Histogramm stud. Residuen", freq=FALSE)
# Dichtefunktion der Standardnormalverteilung hinzufügen
curve(dnorm(x, mean=0, sd=1), col="red", lwd=2, add=TRUE)
```



Zusätzlich wird nun der Quantil-Quantil (Q-Q) Plot angeschaut. Falls die Daten aus der Normalverteilung stammen sollten die empirischen und die theoretischen Quantile annähernd überein stimmen und somit auch die Datenpunkte auf einer Geraden liegen.

```
qqnorm(rstudent(MLR), main="Q-Q-Plot stud. Residuen")
abline(0,1)
```



Da anhand der Graphiken nicht ganz deutlich gesagt werden kann, ob die Normalverteilungsannahme stimmt oder nicht, wird nun zusätzlich das Shapiro-Wilk Test angewendet.

```
shapiro.test(rstudent(MLR))
##
##  Shapiro-Wilk normality test
##
## data:  rstudent(MLR)
## W = 0.99125, p-value = 0.4833
```

Hier ist die Nullhypothese dass es eine Normalverteilung der Grundgesamtheit vorliegt. Zusätzlich wird ein Signifikanzniveau, üblicherweise $\alpha = 5\ \%$, gewählt und die Nullhypothese wird abgelehnt wenn der p-Wert kleiner ist als das Signifikanzniveau. Somit kann man hier die Normalverteilungshypothese nicht verwerfen.

Bei der multiplen linearen Regression wird davon ausgegangen, dass die Residuen unabhängig voneinander sind und diese Annahme wird nun mit Durbin-Watson Test auf Richtigkeit geprüft. Auf was dieser Test beruht kann in [JW08] nachgelesen werden. Die Nullhypothese für den Durbin-Watson-Test ist, dass die Residuen nicht korreliert sind. Laut [JW08] existiert bei einem Wert von 2 keine Autokorrelation zwischen den Residuen.

```
dwt(MLR)
##  lag Autocorrelation D-W Statistic p-value
##    1   -0.0007675521      1.974829   0.876
##  Alternative hypothesis: rho != 0
```

Zum Schluss wird die Multikollinearität geprüft, das heißt es wird geprüft ob die unabhängigen Variablen als lineare Funktion der anderen unabhängigen Variable sich darstellen lassen können. Ein Maß für die

Multikollinearität ist der Varianzinflationsfaktor (kurz VIF) $VIF_j = \frac{1}{1-R_j^2}$. Als Daumenregel sollte der VIF-Wert nicht über 5 oder auch 10 gehen.

```
vif(MLR)
##       Donald_1$Geschlecht            Donald_1$Alter
##                  1.014460                  1.024821
##       Donald_1$Minderheit Donald_1$Fremdenfeindlich
##                  1.299054                  1.287851
##                Donald_1$IQ
##                  1.034409
```

Wie man aus der Tabelle erkennen kann, ist eine Multikollinearit ausgeschlossen da alle Variablen nicht linear abhängig von den anderen Variablen sind.

2 Aufgabenblatt 2: Cross Validation

In dieser Übung wird wieder der Datensatz *Donald* vewendet, wobei hier multiple lineare Regression mit Cross Validation (Kreuzvalidierung) durchgeführt. In dem ersten Aufgabenblatt beruhte das Regressionsmodell auf dem kompletten Datensatz. Bei der Kreuzvalidierung wird jedoch der Datensatz in mehrere Teile aufgeteilt, wobei auf Basis eines Teils des Datensatzes die Modellparameter geschätzt werden und auf den Rest wird dann der Modellfehler berechnet. Während in dem ersten Aufgabenblatt das (korrigierte) Bestimmtheitsmaß R^2 für die Anpassungsgüte des Regressionsmodells verwendet wurde, kann nun mit Hilfe der Cross Validation eine Aussage über die Modellgüte bwz. Prognosegüte des Modells getroffen werden.

Der Datensatz, bestehend aus n Datenpunkte, werden in Trainingsdaten N_T und in Testdaten bzw. Validierungsdaten N_V mit einem Daumenregel $N_V < N_T$ aufgeteilt. Auf der Trainingsdatensatz wird die Regressionsanalyse durchgeführt und die Modellparameter geschätzt. Das Regressionsmodell wird daraufhin auf die Testdaten angewendet wobei die Prognose des Modells mit den tatsächlichen Werten vergleicht wird. Dies kann mit Hilfe der mittlere quadratische Fehler, also Mean Squared Error (MSE)

$$MSE = \frac{1}{N_v}\sum_{i=1}^{N_V}(Y_i - \hat{Y}_i)^2$$

erreicht werden. Bei der k-fold Cross Validation (k-fache Kreuzvalidierung) werden zunächst die Daten in k disjunkte Teilmengen bzw. Blöcke partitioniert. Diese werden dann als Testdatensatz bzw. jeder Block wird einmal als Testset verwendet, wobei die verbleibenden $k-1$ Daten für die Trainings des Modells benutzt. Mit $k = n$ tritt das Spezialfall der k-fachen Kreuzvalidierung als die Leave-One-Out-Kreuzvalidierung (LOOCV) auf. Hier wird jede Beobachtung einzeln als Testdatensatz für das Modell verwendet, wobei die übrigen Beobachtungen als Trainingsset deklariert wird.

Die Daten werden zunächst zufällig in zwei Teildatensätze vom Unfang 50 für Testdaten und 100 für Trainingsdaten aufgeteilt. Diese Teilung wird dreimal mit unterschiedlichen seed-Werte durchgeführt. Zudem wird hier wieder einmal die multiple lineare Regressionsmodell angepasst. Der MSE Wert von Testdaten ist für alle drei Durchgänge ungefähr im gleichen (zwischen 35 und 40) Bereich. Hier sollte kein MSE Wert von 0 erwartet werden, da hier nur eine multiple lineare Regression - einer der einfachsten Modelle - für die Prognose verwendet wurde und es bessere Modelle für die Schätzung der Vorhersage existiert. Nichtsdestotrotz gilt auch hier je größer der MSE, desto schlechter wird die Prognose. Zusätzlich ist es erkennbar, dass es bei allen Teilungen die Variable Minderheit einen signifikanzniveau von 1 % hat während die anderen Variablen mit $\alpha = 0,1$ % einen signifikanten Einfluss auf das Modell haben.

```
set.seed(1234)
indexa<-sample(1:nrow(Donald_1),50)
testa<-Donald_1[indexa,]
traina<-Donald_1[-indexa,]

MLRa <- lm(Trump ~ Geschlecht + Alter + Minderheit + Fremdenfeindlich + IQ, data = traina)

summary(MLRa)
##
## Call:
## lm(formula = Trump ~ Geschlecht + Alter + Minderheit + Fremdenfeindlich +
##     IQ, data = traina)
##
## Residuals:
## <Labelled double>
##      Min       1Q   Median       3Q      Max
## -15.3957  -4.0922  -0.2413   4.5776  12.6798
##
```

```
## Labels:
##  value label
##      0    0%
##    100  100%
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      30.37149    4.53444   6.698 1.53e-09 ***
## Geschlecht        5.83994    1.25243   4.663 1.03e-05 ***
## Alter             0.13875    0.03853   3.601 0.000508 ***
## Minderheit       -6.76480    2.38405  -2.838 0.005570 **
## Fremdenfeindlich  9.30780    0.20487  45.432  < 2e-16 ***
## IQ               -0.38200    0.03834  -9.964  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.201 on 94 degrees of freedom
## Multiple R-squared:  0.9693, Adjusted R-squared:  0.9677
## F-statistic: 594.2 on 5 and 94 DF,  p-value: < 2.2e-16

predMLRa <- predict(MLRa, testa)

#predMLRa

#mean((testa$Trump-predMLRa)^2)
#(sum((testa$Trump-predMLRa)^2))/length(testa)
library(Metrics)
mse(testa$Trump, predMLRa)
## [1] 35.594
```

```
set.seed(4545)
indexb<-sample(1:nrow(Donald_1),50)
testb<-Donald_1[indexb,]
trainb<-Donald_1[-indexb,]

MLRb <- lm(Trump ~ Geschlecht + Alter + Minderheit + Fremdenfeindlich + IQ, data = trainb)

summary(MLRb)
##
## Call:
## lm(formula = Trump ~ Geschlecht + Alter + Minderheit + Fremdenfeindlich +
##     IQ, data = trainb)
##
## Residuals:
## <Labelled double>
##      Min       1Q   Median       3Q      Max
## -16.5584  -4.4862   0.3828   4.4972  11.3950
##
## Labels:
##  value label
##      0    0%
##    100  100%
##
## Coefficients:
```

```
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      35.49797    4.46405   7.952 4.03e-12 ***
## Geschlecht        6.03173    1.24244   4.855 4.79e-06 ***
## Alter             0.15362    0.03618   4.246 5.12e-05 ***
## Minderheit       -9.02295    2.28428  -3.950 0.000151 ***
## Fremdenfeindlich  9.26377    0.20205  45.848  < 2e-16 ***
## IQ               -0.43843    0.03698 -11.857  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.041 on 94 degrees of freedom
## Multiple R-squared:  0.9716, Adjusted R-squared:  0.9701
## F-statistic: 642.4 on 5 and 94 DF,  p-value: < 2.2e-16

predMLRb <- predict(MLRb, testb)

mse(testb$Trump, predMLRb)
## [1] 40.09959
```

```
set.seed(2018)
indexc<-sample(1:nrow(Donald_1),50)
testc<-Donald_1[indexc,]
trainc<-Donald_1[-indexc,]

MLRc <- lm(Trump ~ Geschlecht + Alter + Minderheit + Fremdenfeindlich + IQ, data = trainc)

summary(MLRc)
##
## Call:
## lm(formula = Trump ~ Geschlecht + Alter + Minderheit + Fremdenfeindlich +
##     IQ, data = trainc)
##
## Residuals:
## <Labelled double>
##      Min       1Q   Median       3Q      Max
## -14.1312  -4.4718   0.3780   4.8497  11.7575
##
## Labels:
##  value label
##      0    0%
##    100  100%
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      24.24341    4.64692   5.217 1.08e-06 ***
## Geschlecht        6.31104    1.25617   5.024 2.40e-06 ***
## Alter             0.16254    0.03613   4.498 1.96e-05 ***
## Minderheit       -7.01234    2.19776  -3.191  0.00193 **
## Fremdenfeindlich  9.51777    0.20676  46.033  < 2e-16 ***
## IQ               -0.34549    0.03867  -8.934 3.40e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.178 on 94 degrees of freedom
```

```
## Multiple R-squared:  0.9713, Adjusted R-squared:  0.9698
## F-statistic: 637.2 on 5 and 94 DF,  p-value: < 2.2e-16

predMLRc <- predict(MLRc, testc)

#predMLRc

mse(testc$Trump, predMLRc)
## [1] 38.51793
```

Es wird hier zunächst eine manuelle Leave-One-Out Cross Validation mit der ersten Zeile als Testset durchgeführt und dazugehörige Test MSE berechnet. MSE Wert ist deutlich höher als die Werte aus der Aufgabe 1.

```
indexaa<-2:nrow(Donald_1)
trainaa<-Donald_1[indexaa,]
testaa<-Donald_1[-indexaa,]

MLRaa <- lm(Trump ~ Geschlecht + Alter + Minderheit + Fremdenfeindlich + IQ, data = trainaa)

predMLRaa <- predict(MLRaa, testaa)

#predMLRaa

#summary(MLRaa)

(testaa[1,6]-predMLRaa)^2
##         Trump
## [1,] 61.72219
```

Nun wird das ganze manuell mit for Schleife automatisiert und für die Zeilen 2 bis 150 durchgeführt.

```
#leere Vektoren definieren
quad_abw <- vector(,150)
pred_for <- vector(,150)
data<- data.frame(Donald_1)

#for-Schleife definieren
for(i in 2:150) {
  daten_i <- data[i,]
  MLR_for <- lm(Trump ~ Geschlecht + Alter + Minderheit + Fremdenfeindlich + IQ, data = data[-i,])
  pred_for[i] <- predict(MLR_for, daten_i)
  quad_abw[i] <- (daten_i[6] - pred_for[i])^2
}
sum(quad_abw)/149
## [1] 38.03259
```

Mit dem delta Befehl wird ein Vektor mit MSE Werten der Cross Validation, wobei die erste Komponente die rohe und zweite die angepasste Vorhersagefehler der Kreuzvalidierungsschätzung, ausgegeben. Weiteres dazu kann in [BOOT] auf der Seite 42 nachgelesen werden.

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

library(boot)
```

```
cv <- cv.glm(Donald_1, model, K=150)

cv$delta
## [1] 38.19052 38.18070
```

Es ist erkennbar, dass beide MSE Werte, für for Schleife und cv.glm Befehl, sehr nahe beieinander und auch wieder zwischen 35 und 40 liegen.

Zuletzt wird 10 k-fache Cross Validations mit $k = 5$ und $k = 10$ durchgeführt. Es ist wieder deutlich, dass die MSE Werte zwischen 35 und 40 liegen.

```
cv1 <- cv.glm(Donald_1, model, K=5)
cv2 <- cv.glm(Donald_1, model, K=5)
cv3 <- cv.glm(Donald_1, model, K=5)
cv4 <- cv.glm(Donald_1, model, K=5)
cv5 <- cv.glm(Donald_1, model, K=5)
cv6 <- cv.glm(Donald_1, model, K=5)
cv7 <- cv.glm(Donald_1, model, K=5)
cv8 <- cv.glm(Donald_1, model, K=5)
cv9 <- cv.glm(Donald_1, model, K=5)
cv10 <- cv.glm(Donald_1, model, K=5)

cv11 <- cv.glm(Donald_1, model, K=10)
cv12 <- cv.glm(Donald_1, model, K=10)
cv13 <- cv.glm(Donald_1, model, K=10)
cv14 <- cv.glm(Donald_1, model, K=10)
cv15 <- cv.glm(Donald_1, model, K=10)
cv16 <- cv.glm(Donald_1, model, K=10)
cv17 <- cv.glm(Donald_1, model, K=10)
cv18 <- cv.glm(Donald_1, model, K=10)
cv19 <- cv.glm(Donald_1, model, K=10)
cv20 <- cv.glm(Donald_1, model, K=10)


cv1$delta
## [1] 37.15799 36.95285
cv2$delta
## [1] 38.15927 37.83333
cv3$delta
## [1] 37.52493 37.27199
cv4$delta
## [1] 38.81293 38.41119
cv5$delta
## [1] 40.14587 39.59908
cv6$delta
## [1] 38.54088 38.17879
cv7$delta
## [1] 40.15597 39.60076
cv8$delta
## [1] 37.60051 37.33899
cv9$delta
## [1] 38.24750 37.91196
cv10$delta
```

```
## [1] 39.38923 38.92922

cv11$delta
## [1] 38.08775 37.93810
cv12$delta
## [1] 37.66769 37.54196
cv13$delta
## [1] 37.93579 37.79597
cv14$delta
## [1] 37.88499 37.74664
cv15$delta
## [1] 38.55582 38.38218
cv16$delta
## [1] 38.64419 38.46447
cv17$delta
## [1] 38.48826 38.31801
cv18$delta
## [1] 39.20187 38.99206
cv19$delta
## [1] 37.73715 37.60810
cv20$delta
## [1] 38.90833 38.71437
```

3 Aufgabenblatt 3: Bootstrap

Das Bootstrap Verfahren ist eine Resampling Methode mit der man unter anderem die Verteilung von Parameterschätzern einer linearen Regression überprüfen kann. Es wird also verwendet, wenn man bei Verletzung der Voraussetzungen (zum Beispiel bei der linearen Regression) oder bei komplexeren Schätzern Aussagen über deren Verteilung machen möchte. Bei der Bootstrap-Algorithmus werden aus einer Stichprobe der Länge n R Replikationen, der wiederum die Länge n hat, erzeugt. Das heißt hier wird immer wieder neue Stichprobe durch Ziehen mit Zurucklegen (der Länge n) erzeugt. Aus dieser neu erzeugten Stichprobe wird dann der Schätzwert $\hat{\theta}$ berechnet und dieser Vorgang wird R mal wiederholt.

In dieser Aufgabe wird die Aussage des Zentralen Grenzwertsatzes für Bootstrap-Daten überprüft. Zunächst werden 50 bzw. 1000 Datensätze mit jeweils 500 gleichverteilten Daten auf [0,1] simuliert und davon die Mittelwerte gespeichert.

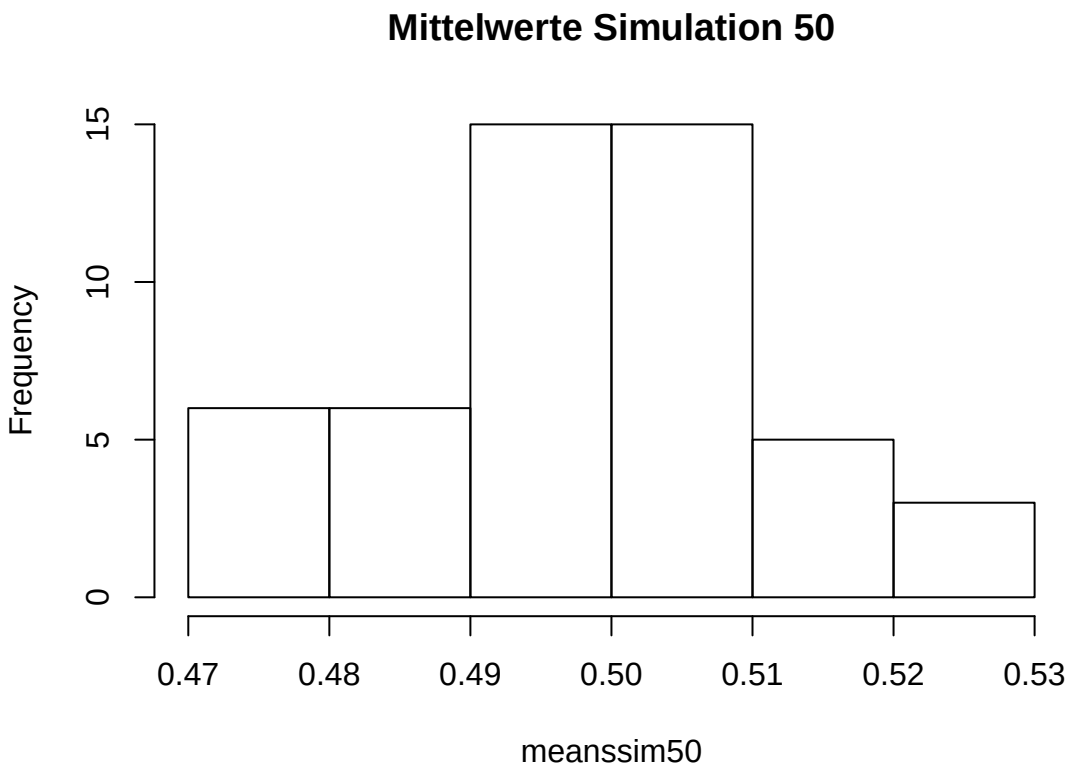
```
set.seed(2018)

datensatz1 <- matrix(runif(500*50,0,1), nrow=50)
meanssim50 <- apply(datensatz1,1,mean)

datensatz2 <- matrix(runif(500*1000,0,1), nrow=1000)
meanssim1000 <- apply(datensatz2,1,mean)
```

Es wird nun ein weiterer Datensatz mit 500 gleichverteilten Daten auf [0,1] simuliert, wobei hier noch mit Bootstrap 50 bzw. 1000 Replikationen erzeugt und die Mittelwerte der Replikationen gespeichert werden.

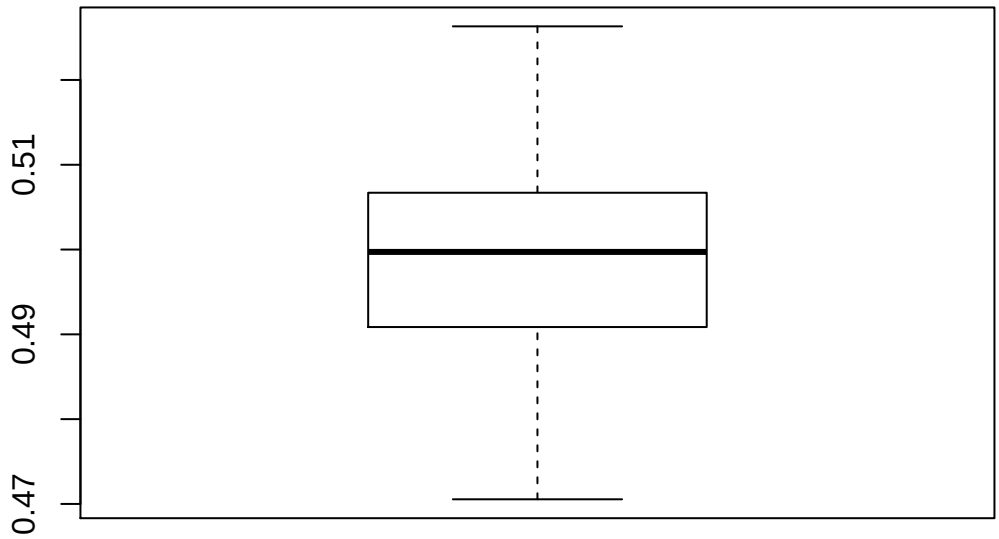
```
#library(boot)
x<- data.frame(runif(500,min=0,max=1))
#x[,1]

helpfunction<- function(data,index){
  d <- data[index,]
  return(mean(d))
}

meansboot50<- boot(x, statistic = helpfunction, R = 50)

meansboot1000<- boot( x, statistic = helpfunction, R = 1000)
```

Nun werden die Mittelwerte der simulierten Datensätze mit Mittelwerte der Bootstrap Replikationen mit Hilfe der Histogramme und Boxplots miteinander verglichen. Mit den untenstehenden Histogrammen ist es deutlich, dass die Verteilung der Mittelwerte der simulierten Daten und Bootstrap Daten, zum mindestens für $R = 1000$, sehr ähnlich sind. Zusätlich wird mit dem Shapiro-Wilk Test auf die Normalverteilung getestet. Hier ist es zu erkennen, dass die Mittelwerte aus dem simulierten Daten und auch aus dem replizierten Datensatz normalverteilt sind.

```
hist(meanssim50, breaks = "FD", main = "Mittelwerte Simulation 50")
```

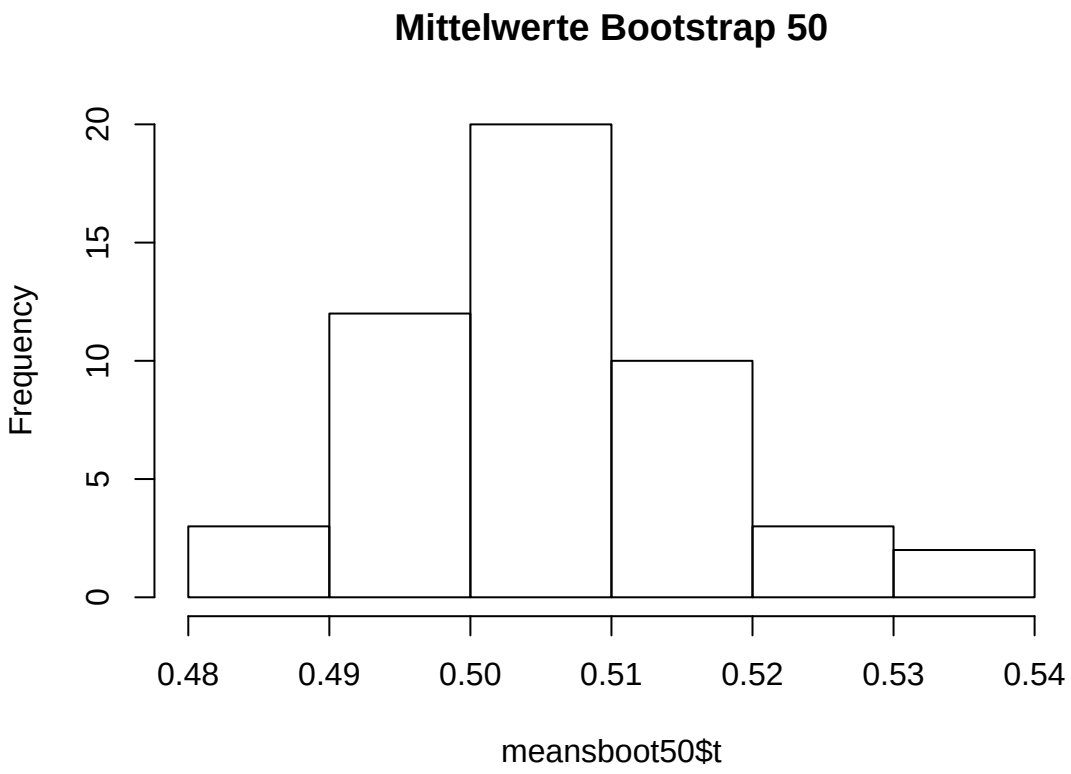



```
boxplot(meanssim50, main="Boxplot der Mittelwerte Simulation 50")
```

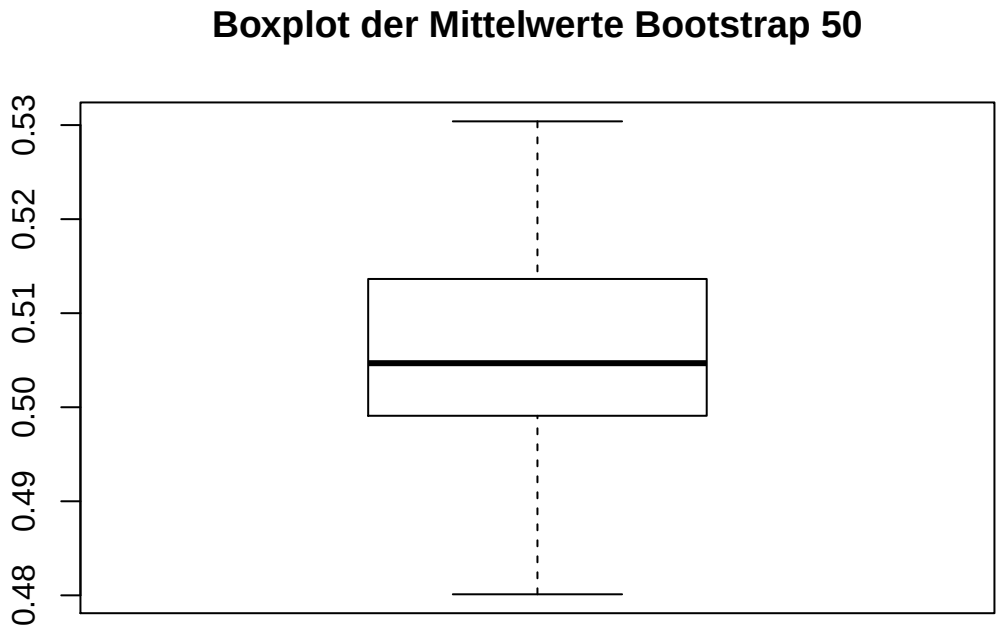



```
shapiro.test(meanssim50)
##
##  Shapiro-Wilk normality test
##
## data:  meanssim50
## W = 0.97497, p-value = 0.364
```

```
hist(meansboot50$t, breaks = "FD", main = "Mittelwerte Bootstrap 50")
```

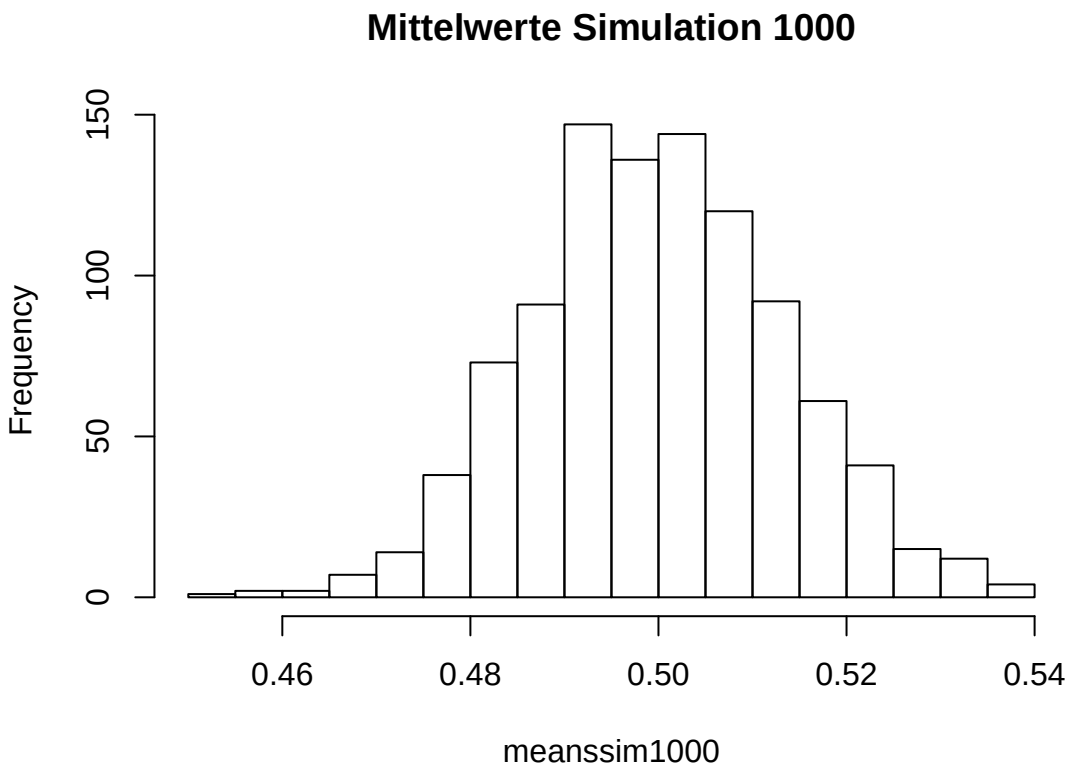



```
boxplot(meansboot50$t, main="Boxplot der Mittelwerte Bootstrap 50")
```

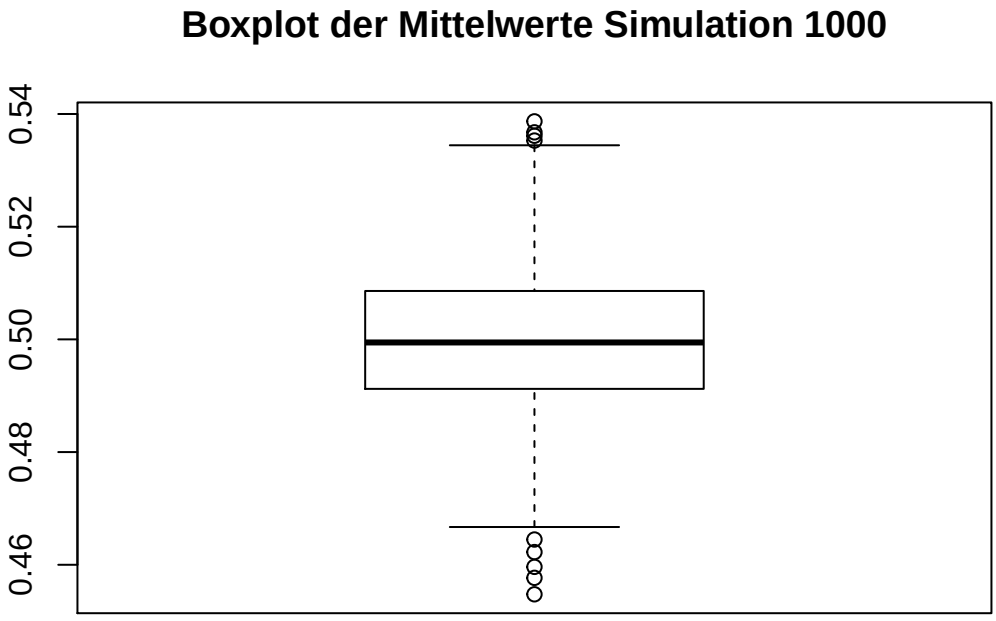



```
shapiro.test(meansboot50$t)
##
##  Shapiro-Wilk normality test
##
## data:  meansboot50$t
## W = 0.98418, p-value = 0.736
```

```
hist(meanssim1000, breaks = "FD", main = "Mittelwerte Simulation 1000")
```

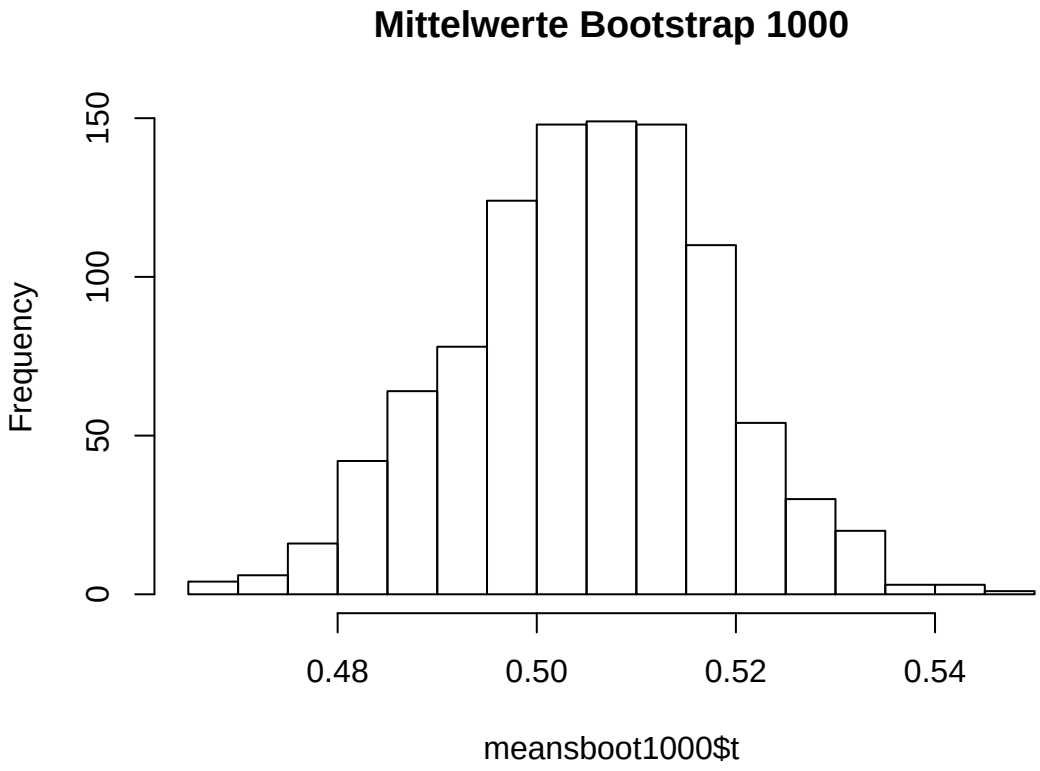



```
boxplot(meanssim1000, main="Boxplot der Mittelwerte Simulation 1000")
```

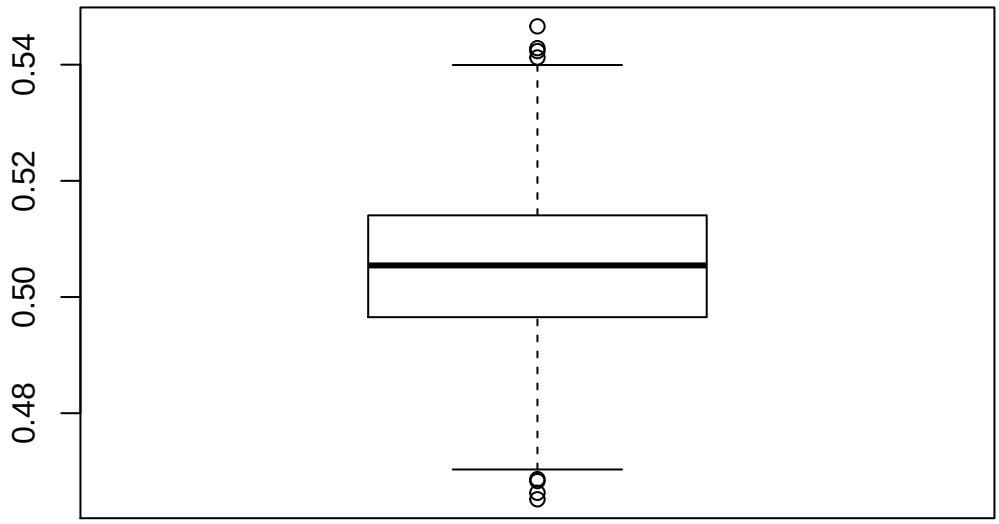



```
shapiro.test(meanssim1000)
##
##  Shapiro-Wilk normality test
##
## data:  meanssim1000
## W = 0.99848, p-value = 0.5446
```

```
hist(meansboot1000$t, breaks = "FD", main = "Mittelwerte Bootstrap 1000")
```




```r
boxplot(meansboot1000$t, main="Boxplot der Mittelwerte Bootstrap 1000")
```




```
shapiro.test(meansboot1000$t)
##
##  Shapiro-Wilk normality test
##
## data:  meansboot1000$t
## W = 0.99826, p-value = 0.4112
```

Mit dem Zweistichproben-t-Test wird Mittelwerte zweier Stichproben, hier simulierten und mit Bootstrap berechnteten Mittelwerte, auf die Gleichheit (Nullhypothese) oder Ungleichheit (Alternativhypothese) der Erwartungswerte geprüft. Da hier bei beiden Datensätze (50 bzw. 1000) die p-Werte deutlich kleiner als Signifikanzniveau $\alpha = 5\ \%$ sind, kann gesagt werden, dass die Mittelwerte signikant unterschiedlich sind.

```
t.test(meanssim50,meansboot50$t)
##
##  Welch Two Sample t-test
##
## data:  meanssim50 and meansboot50$t
## t = -2.7425, df = 96.385, p-value = 0.007271
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.01166519 -0.00186941
## sample estimates:
## mean of x mean of y
## 0.4988354 0.5056027
t.test(meanssim1000,meansboot1000$t)
##
##  Welch Two Sample t-test
```

```
##
## data:  meanssim1000 and meansboot1000$t
## t = -9.0196, df = 1994.7, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.006466103 -0.004156421
## sample estimates:
## mean of x mean of y
## 0.4998730 0.5051842
```

4 Aufgabenblatt 4: Shrinkage Methoden

In diesem Teil der Ausarbeitung werden die Shrinkage Methoden, Ridge und Lasso Regressionen, für die Prognose der Zielgröße angewendet. Bei zu vielen Einflussvariablen kann es vorkommen, dass man den Einfluss der geschätzten Modellparameter (Koeffizienten) künstlich mit Hilfe eines Straftermes schrumpfen (Ridge) bzw. die Anzahl dieser Variablen reduzieren und somit einen Variablenselektion durchführen (Lasso). Es wird der Datensatz *prostate* aus dem Paket *ElemStatLearn* verwendet. Es wird der Einfluss von acht Variablen (lcavol, lweight, age, lbph, svi, lcp, gleason und pgg45) von Prostata-Krebs-Patienten auf deren PSA-Wert (die abhängige Variable lpsa) untersucht.

Zunächst werden die Daten eingelesen und eine lineare Regression durchgeführt.

```
library(ElemStatLearn)
library(glmnet)
data("prostate", package = "ElemStatLearn")

trainsh <- prostate[prostate$train,]
testsh <- prostate[!prostate$train,]

X <- data.matrix(prostate[,1:8]) #anstatt model.matrix hier data.matrix genommen oder as.matrix()
Y <- prostate$lpsa

X.train <- data.matrix(trainsh[,1:8])
Y.train <- trainsh$lpsa

X.test <- data.matrix(testsh[,1:8])
Y.test <- testsh$lpsa
```

Zudem werden die Koeffizienten der linearen Regressionsmodell in einem Vektor gespeichert um später diese mit Ridge und Lasso Regressionskoeffizienten zu vergleichen.

```
lineareReg <- lm(lpsa ~ lcavol + lweight + age + lbph + svi + lcp + gleason + pgg45,
                 data=trainsh)

summary(lineareReg)
##
## Call:
## lm(formula = lpsa ~ lcavol + lweight + age + lbph + svi + lcp +
##     gleason + pgg45, data = trainsh)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.64870 -0.34147 -0.05424  0.44941  1.48675
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.429170   1.553588   0.276  0.78334
## lcavol       0.576543   0.107438   5.366 1.47e-06 ***
## lweight      0.614020   0.223216   2.751  0.00792 **
## age         -0.019001   0.013612  -1.396  0.16806
## lbph         0.144848   0.070457   2.056  0.04431 *
## svi          0.737209   0.298555   2.469  0.01651 *
## lcp         -0.206324   0.110516  -1.867  0.06697 .
## gleason     -0.029503   0.201136  -0.147  0.88389
## pgg45        0.009465   0.005447   1.738  0.08755 .
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7123 on 58 degrees of freedom
## Multiple R-squared:  0.6944, Adjusted R-squared:  0.6522
## F-statistic: 16.47 on 8 and 58 DF,  p-value: 2.042e-12

koeff = coef(lineareReg)
koeff
##  (Intercept)       lcavol      lweight          age         lbph
##  0.429170133  0.576543185  0.614020004 -0.019001022  0.144848082
##          svi          lcp      gleason        pgg45
##  0.737208645 -0.206324227 -0.029502884  0.009465162
```

Bei der Ridge Regression mit $\lambda = 0$ erhält man wieder die lineare Regression.

```
# alpha = 0 gibt RR und alpha=1 gibt Lasso zurück
ridge0 <- glmnet(X,Y, alpha=0, lambda=0)
ridge0
##
## Call:  glmnet(x = X, y = Y, alpha = 0, lambda = 0)
##
##      Df   %Dev Lambda
## [1,]  8 0.6634      0
coef(ridge0)
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept)  0.179531072
## lcavol       0.564271417
## lweight      0.622218451
## age         -0.021251155
## lbph         0.096678106
## svi          0.761615376
## lcp         -0.106066349
## gleason      0.049475128
## pgg45        0.004454923
```

Vergleicht man nun die lineare Regression bzw. Ridge Regression mit $\lambda = 0$ mit Ridge Regression mit $\lambda = 10$, so ist es deutlich erkennbar, dass die Einfüsse bzw. die Koeffizienten der Variablen schrumpfen.

```
ridge10 <- glmnet(X,Y, alpha=0, lambda=10)
ridge10
##
## Call:  glmnet(x = X, y = Y, alpha = 0, lambda = 10)
##
##      Df  %Dev Lambda
## [1,]  8 0.254     10
coef(ridge10)
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept) 1.536807719
## lcavol      0.064394837
## lweight     0.106582457
## age         0.001718267
## lbph        0.012817952
```

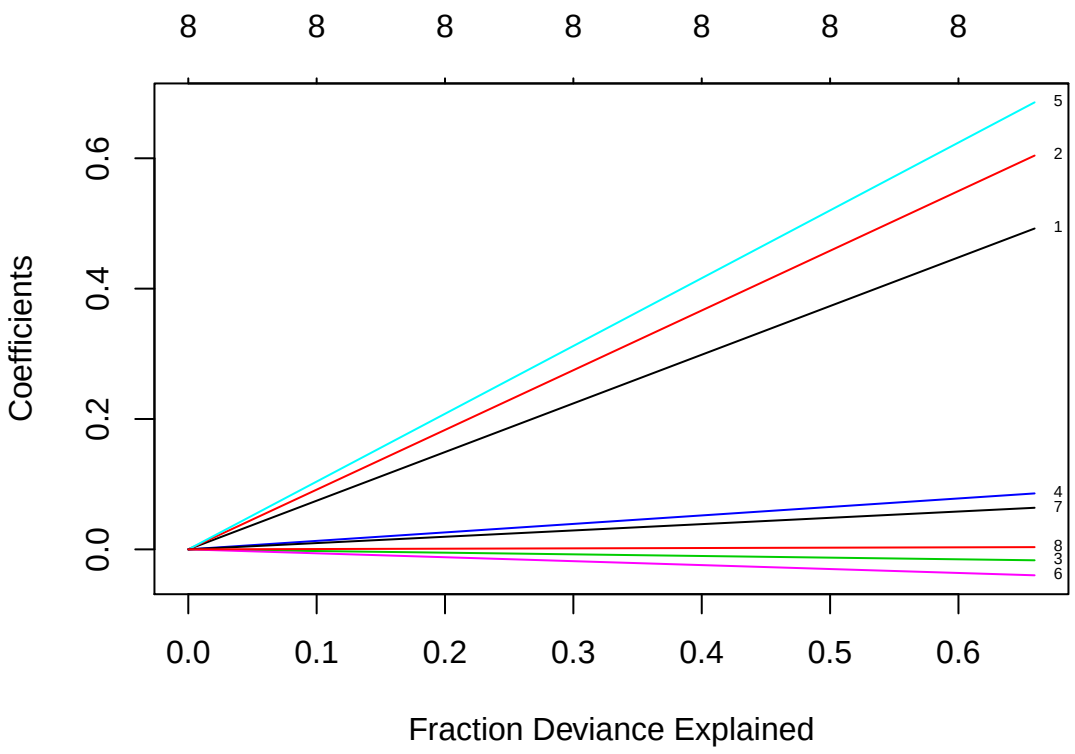
```
## svi         0.135335780
## lcp         0.036490103
## gleason     0.044694072
## pgg45       0.001324128
```

Möchte man das nun in die Höhe treiben, so kann auch ein Wert von $\lambda = 100$ betrachtet werden.

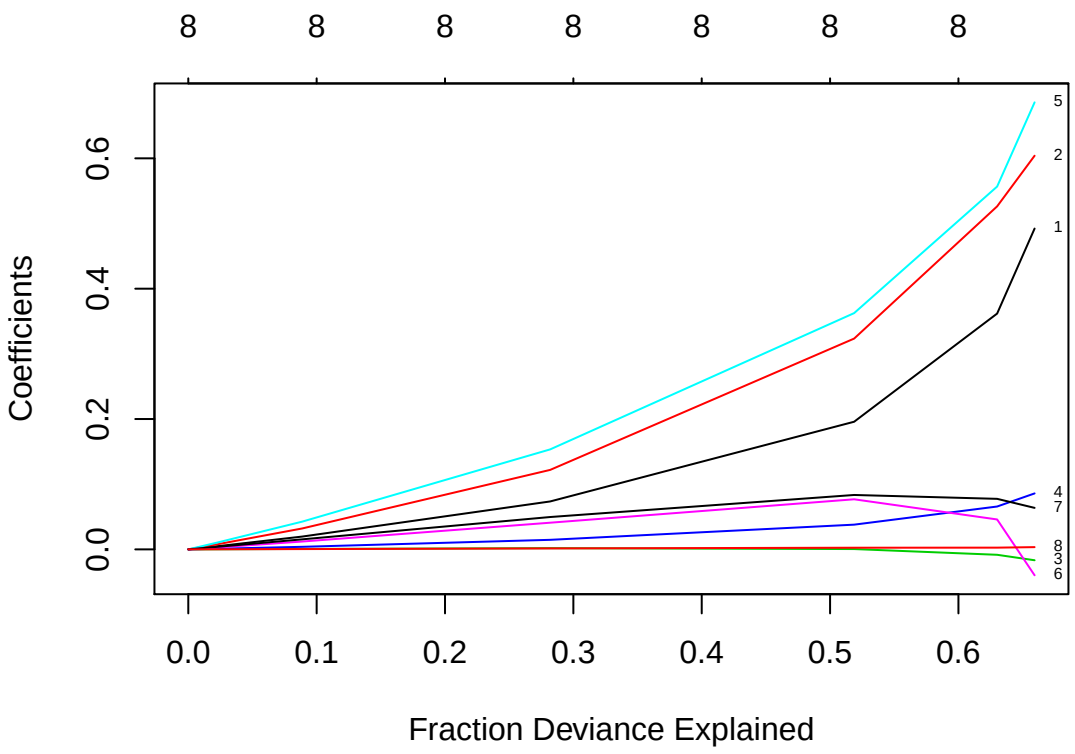
```
ridge100 <- glmnet(X,Y, alpha=0, lambda=100)
ridge100
##
## Call:  glmnet(x = X, y = Y, alpha = 0, lambda = 100)
##
##      Df    %Dev Lambda
## [1,]  8 0.03756    100
coef(ridge100)
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                         s0
## (Intercept) 2.3507302368
## lcavol      0.0080214768
## lweight     0.0130560364
## age         0.0002842558
## lbph        0.0015962024
## svi         0.0175222400
## lcp         0.0049959420
## gleason     0.0064591024
## pgg45       0.0001895105
```

Die farbigen Kurven zeigen wie die Koeffizientenschätzungen für verschiedene unabhängige Variablen sich durch verschiedene ??-Werten entwickeln. Zudem misst die horizontale Achse den durch das angepasste Modell erklärten Abweichungsanteil.

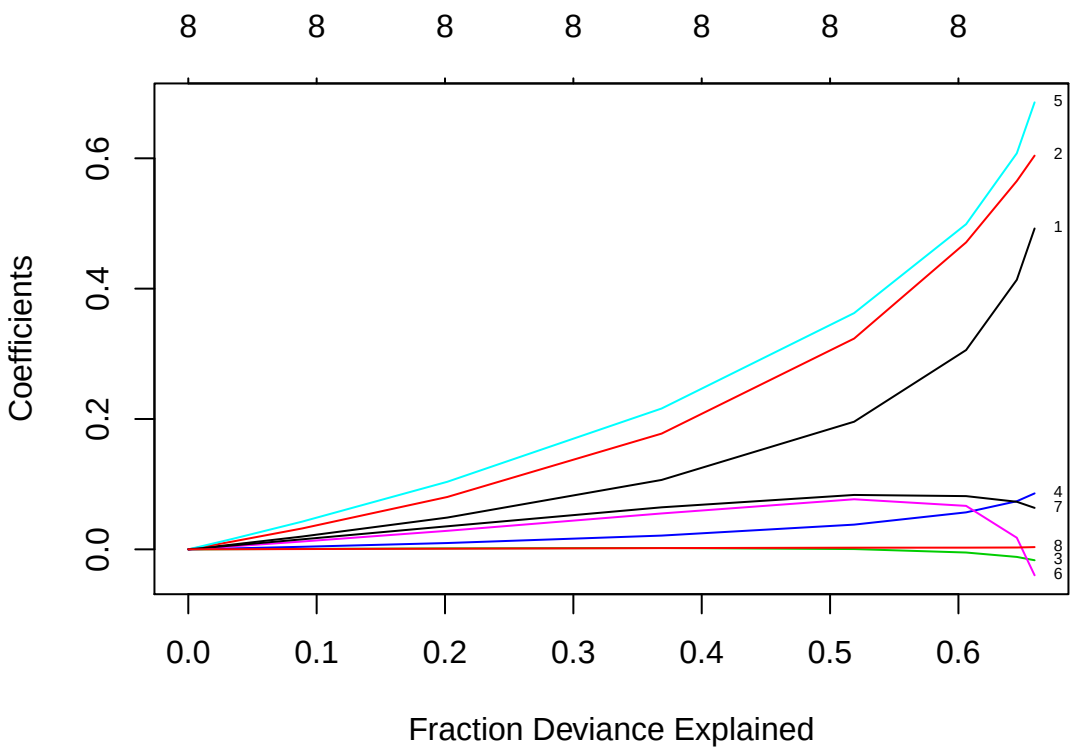
```
plot(glmnet(X,Y, alpha=0, nlambda=2), xvar='dev', label=TRUE)
```



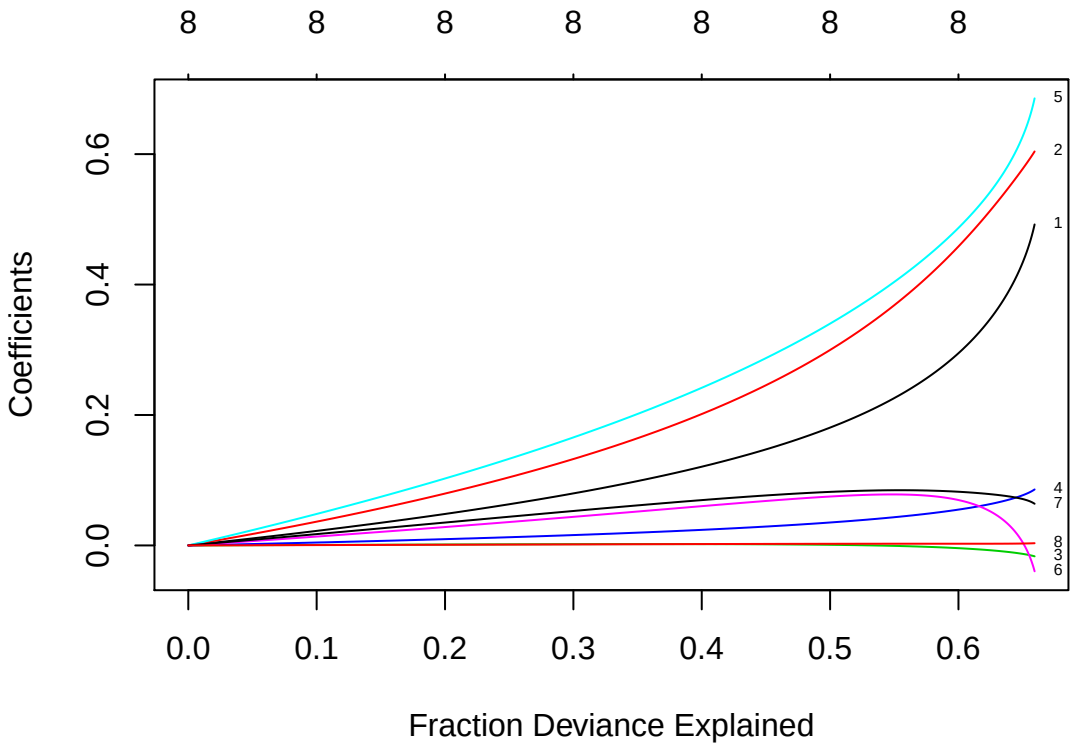

```
plot(glmnet(X,Y, alpha=0, nlambda=7), xvar='dev', label=TRUE)
```




```
plot(glmnet(X,Y, alpha=0, nlambda=10), xvar='dev', label=TRUE)
```

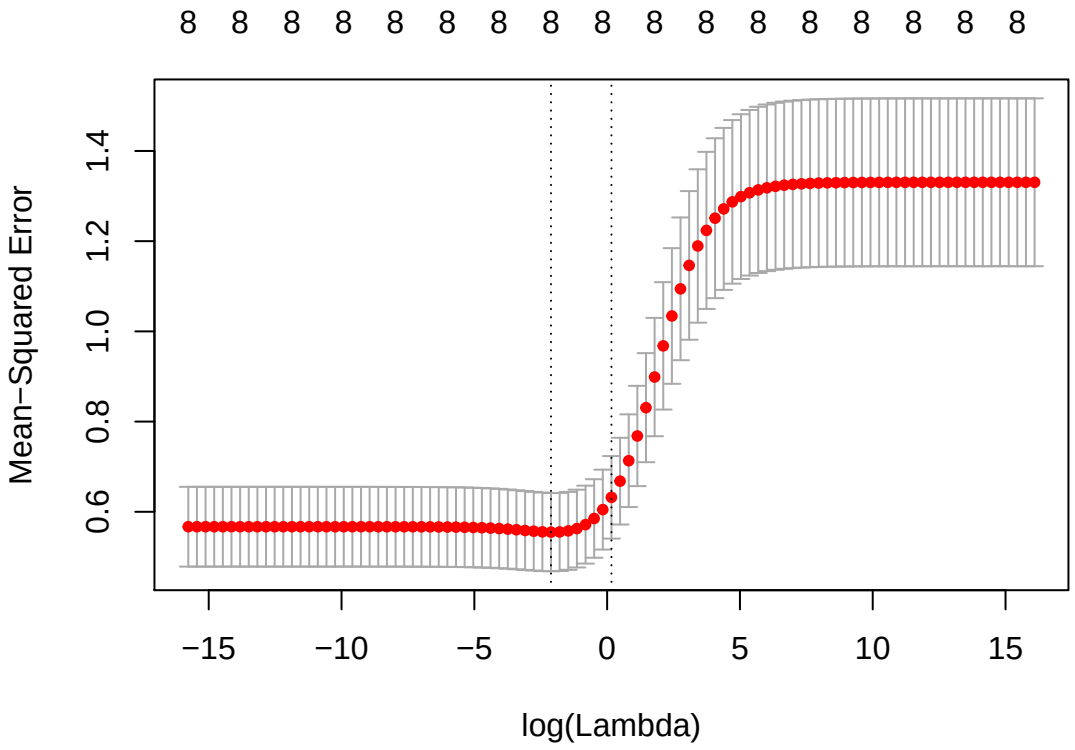



```
plot(glmnet(X,Y, alpha=0, nlambda=1000), xvar='dev', label=TRUE)
```



Nun wird das beste λ über Kreuzvalidierung gewählt und ein Plot für den Test-MSE-Schätzer in Anhängigkeit von λ erzeugt.

```
set.seed(1234)
ridge.cv <- cv.glmnet(X, Y, alpha=0, lambda = 5^seq(10,-10, length =100))
plot(ridge.cv)
```




```
ridge.cv$lambda.min
## [1] 0.1208261
```

Der optimale Schätzer ist hier $\lambda = 0.1208261$ und die dazugehörigen Koeffizienten für die Ridge Regression sehen wie folgt aus:

```
coef(ridge.cv, s = "lambda.min")
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept) -0.025847924
## lcavol       0.468992419
## lweight      0.594897341
## age         -0.015233373
## lbph         0.082299975
## svi          0.662214431
## lcp         -0.021010212
## gleason      0.066972280
## pgg45        0.003178942
```

Nun vergleicht man diese Koeffizienten mit den Koeffizienten aus der linearen Regressionsmodell. Es fällt auf, dass alle Variablen kleiner sind als vorher.

```
koeff
##  (Intercept)       lcavol      lweight          age         lbph
##  0.429170133  0.576543185  0.614020004 -0.019001022  0.144848082
##          svi          lcp      gleason        pgg45
##  0.737208645 -0.206324227 -0.029502884  0.009465162
```

Am Ende kann nun ein Test MSE berechnet werden.

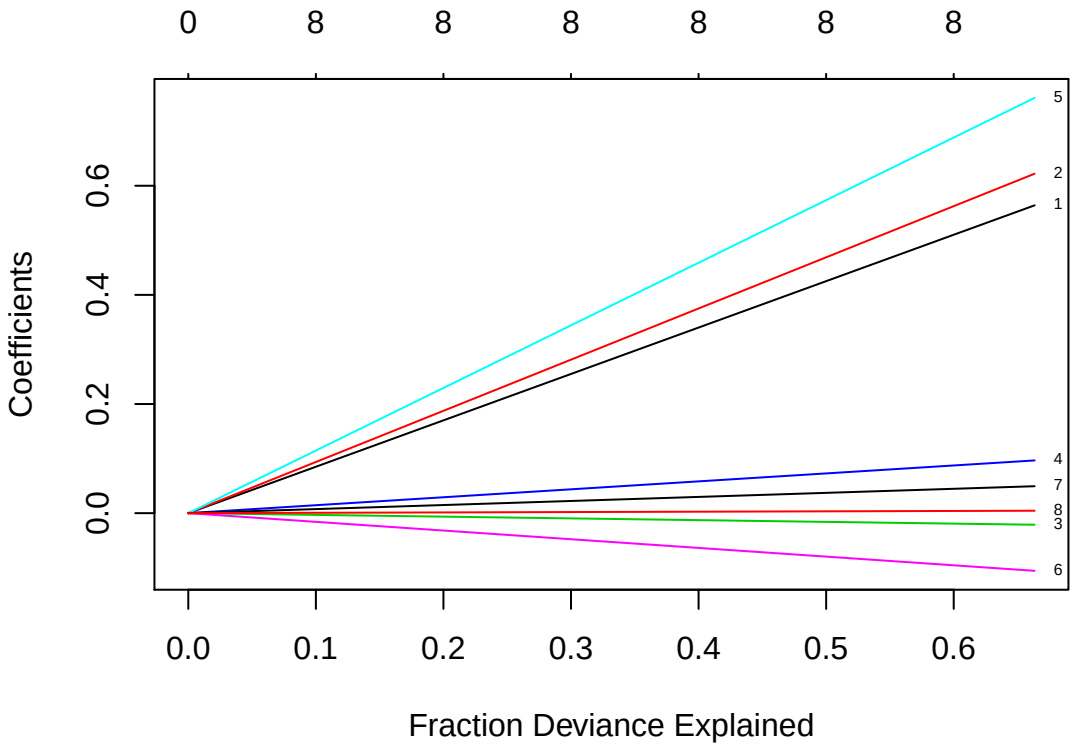
```
ridge.fit <- glmnet(X,Y, alpha=0, lambda=ridge.cv$lambda.min )
ridge.pred <- predict(ridge.fit, X.test)

mse(ridge.pred, Y.test)
## [1] 0.4045433
```

Jetzt werden die Koeffizienten der Variablen nicht nur geschrumpft sondern es findet direkt eine Variablenselektion statt, wobei die Einflüsse der Variablen 0 werden und somit auch nicht mehr in das Modell aufgenommen werden. Das ganze geschieht mit Lasso Methode.

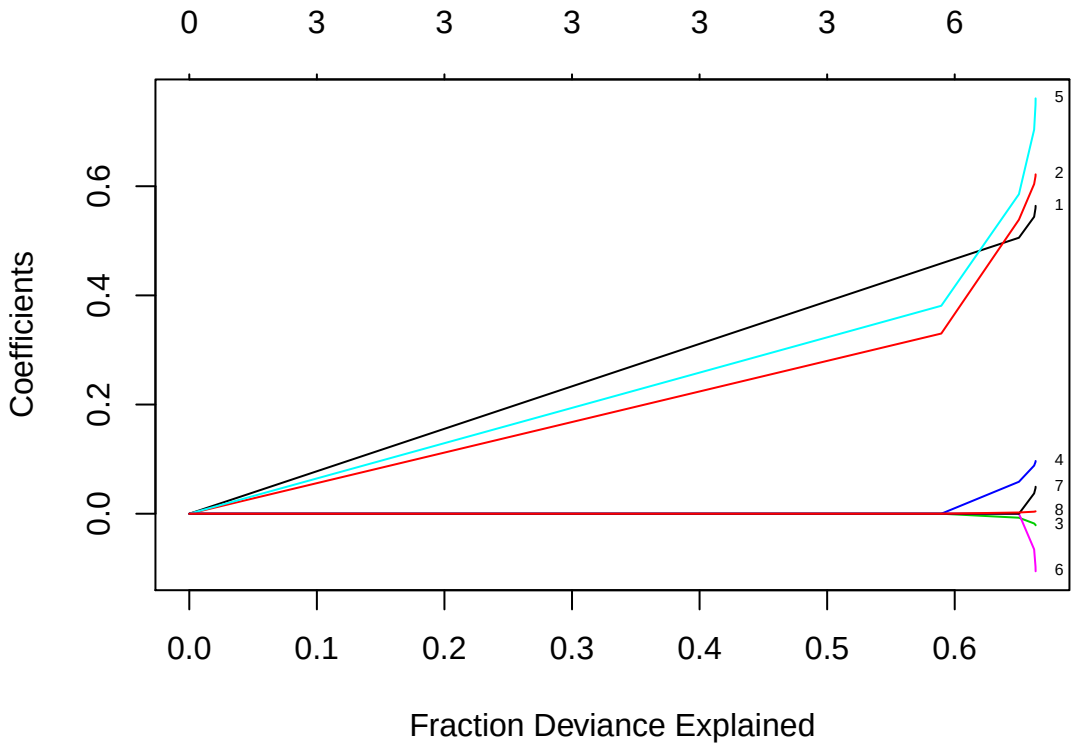
```
lasso0 <- glmnet(X,Y, alpha=1, lambda=0)
lasso0
##
## Call:  glmnet(x = X, y = Y, alpha = 1, lambda = 0)
##
##      Df   %Dev Lambda
## [1,]  8 0.6634      0
coef(lasso0)
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept)  0.179531072
## lcavol       0.564271417
## lweight      0.622218451
## age         -0.021251155
## lbph         0.096678106
## svi          0.761615376
## lcp         -0.106066349
## gleason      0.049475128
## pgg45        0.004454923
```

```
lasso10 <- glmnet(X,Y, alpha=1, lambda=10)
lasso10
##
## Call:  glmnet(x = X, y = Y, alpha = 1, lambda = 10)
##
##      Df %Dev Lambda
## [1,]  0    0     10
coef(lasso10)
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                    s0
## (Intercept) 2.478387
## lcavol      0.000000
## lweight     .
## age         .
## lbph        .
## svi         .
## lcp         .
## gleason     .
## pgg45       .
```

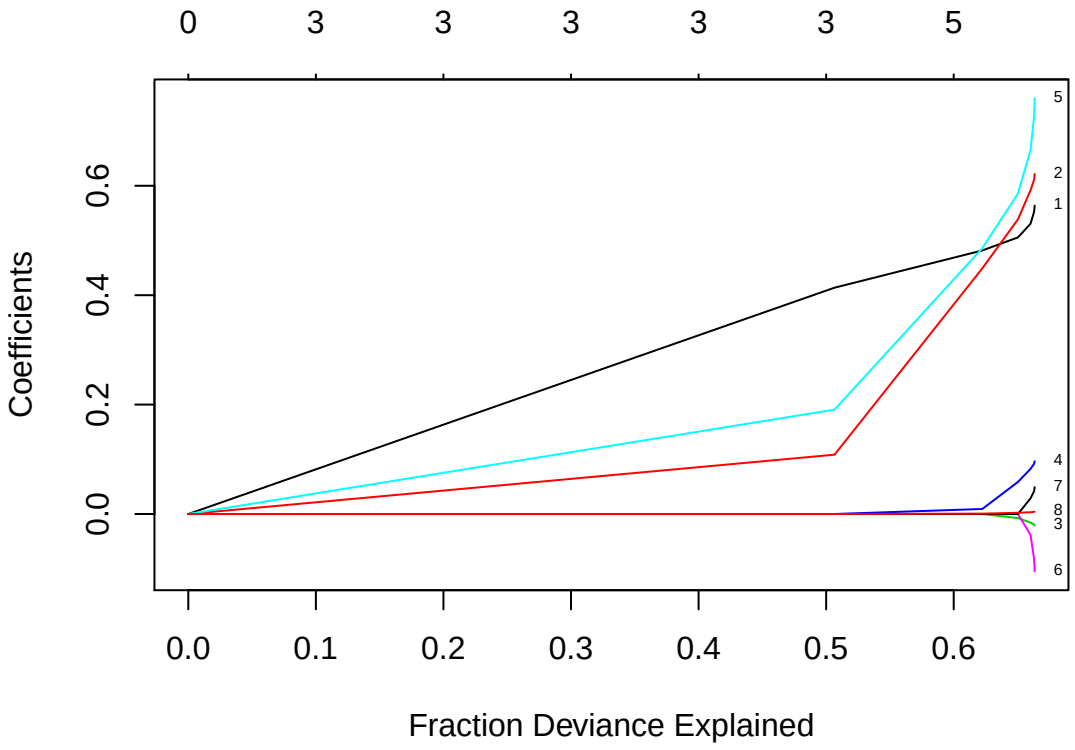
```
plot(glmnet(X,Y, alpha=1, nlambda=2), xvar='dev', label=TRUE)
```



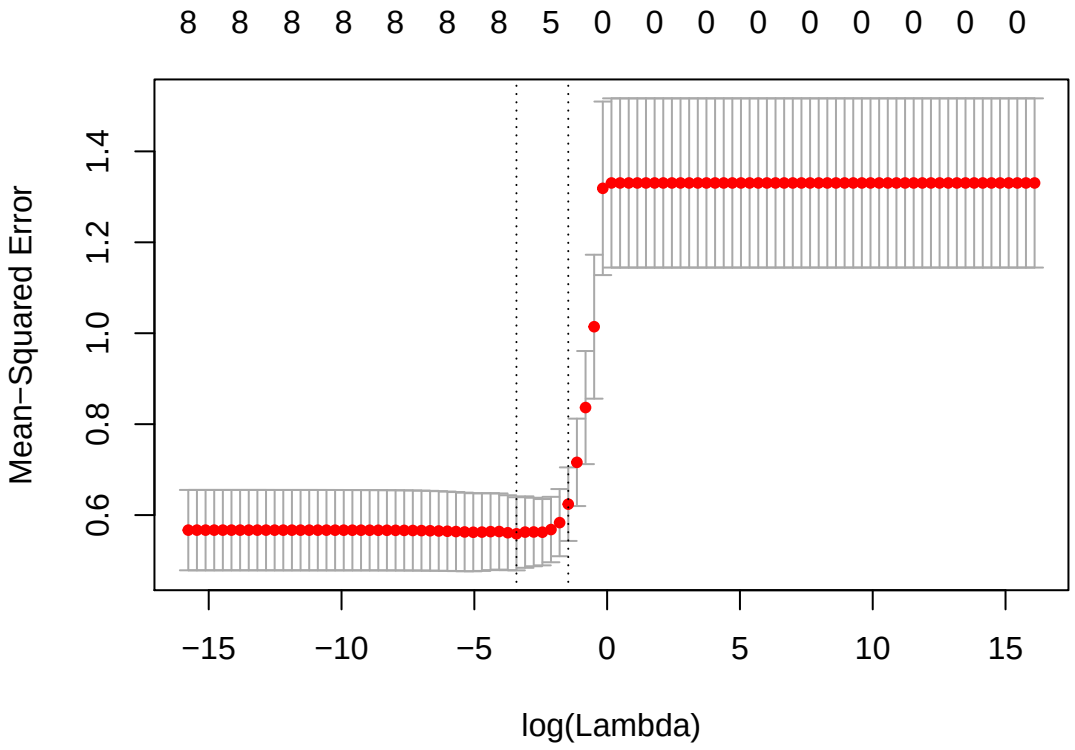

```
plot(glmnet(X,Y, alpha=1, nlambda=7), xvar='dev', label=TRUE)
```




```
plot(glmnet(X,Y, alpha=1, nlambda=10), xvar='dev', label=TRUE)
```




```
set.seed(1234)
lasso.cv <- cv.glmnet(X, Y, alpha=1, lambda = 5^seq(10,-10, length =100))
plot(lasso.cv)
```




```
lasso.cv$lambda.min
## [1] 0.03291064
```

```
coef(lasso.cv, s = "lambda.min")
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept)  0.162538929
## lcavol       0.508065496
## lweight      0.551528027
## age         -0.009250399
## lbph         0.064355329
## svi          0.593849976
## lcp          .
## gleason      0.003978919
## pgg45        0.002353636
```

```
koeff
##  (Intercept)       lcavol      lweight          age         lbph
##  0.429170133  0.576543185  0.614020004 -0.019001022  0.144848082
##          svi          lcp      gleason        pgg45
##  0.737208645 -0.206324227 -0.029502884  0.009465162
```

Das Lasso Verfahren mit der minimalen $\lambda = 0.03291064$ eliminiert die Variable lcp und verringert zusätzlich die Einflüsse der restlichen Variablen.

```
lasso.fit <- glmnet(X,Y, alpha=1, lambda=lasso.cv$lambda.min )
lasso.pred <- predict(lasso.fit, X.test)
```

```r
mse(lasso.pred, Y.test)
## [1] 0.392165
```

5 Aufgabenblatt 5: PCA- und PLS-Regression

```
library(ElemStatLearn)
library(glmnet)
data("prostate", package = "ElemStatLearn")

trainsh <- prostate[prostate$train,]
testsh <- prostate[!prostate$train,]

X <- data.matrix(prostate[,1:8]) #anstatt model.matrix hier data.matrix genommen oder as.matrix()
Y <- prostate$lpsa

X.train <- data.matrix(trainsh[,1:8])
Y.train <- trainsh$lpsa

X.test <- data.matrix(testsh[,1:8])
Y.test <- testsh$lpsa
```

In diesem Aufgabenblatt werden wieder die *prostate* Daten verwendet und die PCA- und PLS Regressionen durchgeführt.

Zunächst wird die Korrelationsmatrix für unabhängigen Variablen erstellt. Es ist hier zu erkennen, dass eine hohe Korreltaion bei den Variablen lcp und lcavol mit 0.6753, lcp und svi mit 0.6731, lcp und pgg45 mit 0.6315 und zuletzt gleason und pgg45 mit 0.7519 existiert. Da es anscheinend hier hohe Multikollinearität vorliegt, können dementsprechend PCA- und PLS-Regressionen als Modellansätze verwendet werden. Somit ist es auch nicht verwunderlich, dass die Variable lcp bei der Lasso Regression eliminiert wurde.

```
pro <- prostate[, c(-9, -10)]
round(cor(pro),4)
##         lcavol lweight    age    lbph     svi     lcp gleason  pgg45
## lcavol  1.0000  0.2805 0.2250  0.0273  0.5388  0.6753  0.4324 0.4337
## lweight 0.2805  1.0000 0.3480  0.4423  0.1554  0.1645  0.0569 0.1074
## age     0.2250  0.3480 1.0000  0.3502  0.1177  0.1277  0.2689 0.2761
## lbph    0.0273  0.4423 0.3502  1.0000 -0.0858 -0.0070  0.0778 0.0785
## svi     0.5388  0.1554 0.1177 -0.0858  1.0000  0.6731  0.3204 0.4576
## lcp     0.6753  0.1645 0.1277 -0.0070  0.6731  1.0000  0.5148 0.6315
## gleason 0.4324  0.0569 0.2689  0.0778  0.3204  0.5148  1.0000 0.7519
## pgg45   0.4337  0.1074 0.2761  0.0785  0.4576  0.6315  0.7519 1.0000
```

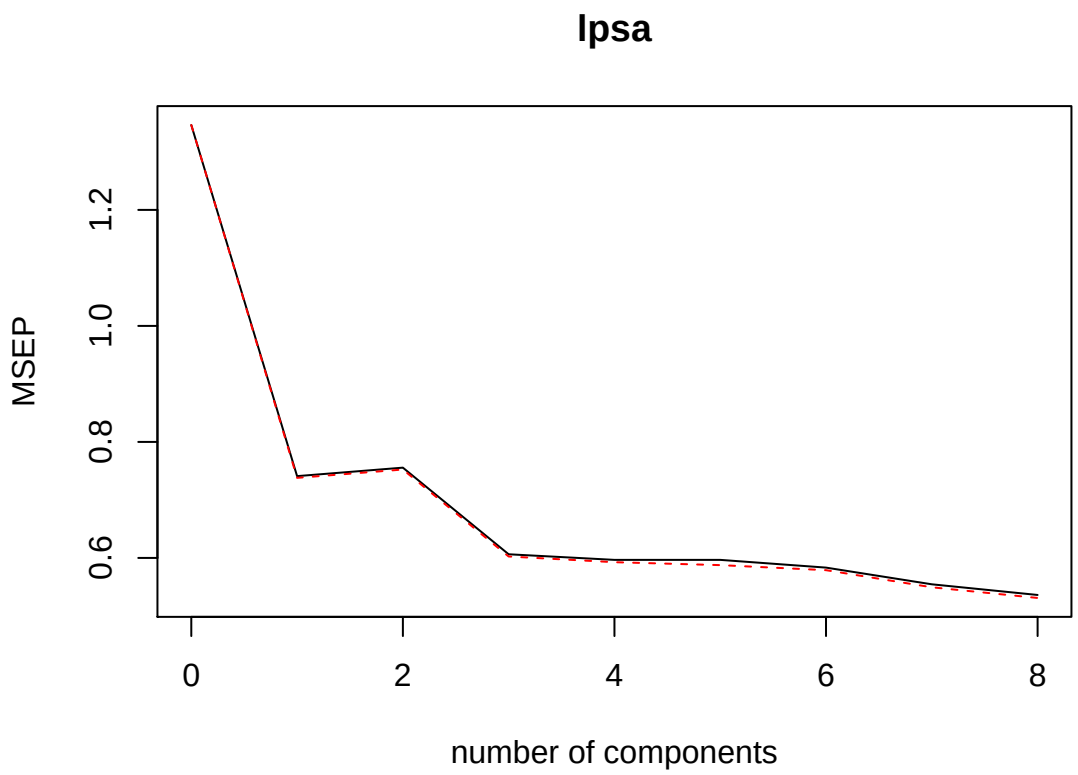
Nun folgt die PCA-Regression, wobei die Variablen vor der PCA-Regression standardisiert werden und zusätzlich wird der Fehler einer 10-fachen Cross-Validation (hier in pls Paket als default) für jede Anzahl von Hauptkomponenten verwendet.

```
library(pls)
library(Metrics)
pcr <- pcr(lpsa ~ lcavol + lweight + age + lbph + svi + lcp + gleason + pgg45,
           data=prostate, scale=TRUE, validation="CV")
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   0.8608   0.8693   0.7786   0.7724   0.7724   0.7637
## adjCV         1.16   0.8591   0.8674   0.7762   0.7698   0.7666   0.7609
##        7 comps  8 comps
## CV      0.7447   0.7322
## adjCV   0.7412   0.7287
##
## 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
```

Anhand der Ergebnisse und des Plots liegt zwar das Minimum der RMSEP bzw. MSEP Werte bei 8 Hauptkomponenten aber dennoch kann man zum Beispiel einer Komponentenanzahl von 4 ein sehr gutes Modell entwickeln. Dies ist möglich, da schon bei 4 Hauptkomonenten ein RMSEP Wert von 0.7666 erreicht werden konnte und zudem ein großer Teil, hier 82.71 %, der abhängigen Variablen erklärt werden kann.

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



Zusätzlich wird der Test MSE-Wert für die PCA-Regression berechnet.

```
pcr_model <- pcr(lpsa ~ lcavol + lweight + age + lbph + svi + lcp + gleason + pgg45,
                 data = trainsh, scale=TRUE, validation = "CV")
```

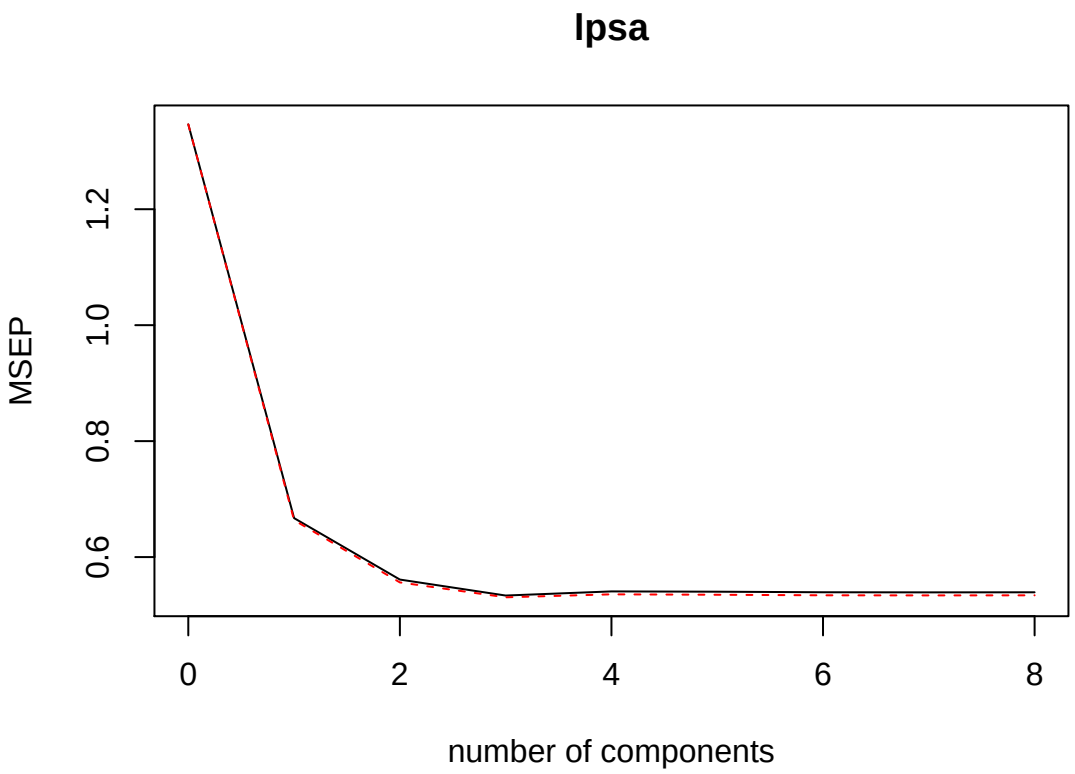
```
pcr_pred <- predict(pcr_model, testsh, ncomp = 4)
library(Metrics)
mse(pcr_pred, Y.test)
## [1] 0.5369458
```

Nun wird eine PLS-Regression durchgeführt.

```
pls <- plsr(lpsa ~ lcavol + lweight + age + lbph + svi + lcp + gleason + pgg45,
            data=prostate, scale=TRUE, validation="CV")
summary(pls)
## 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   0.8168   0.7492   0.7305   0.7354   0.7351   0.7344
## adjCV         1.16   0.8147   0.7459   0.7285   0.7319   0.7314   0.7308
##        7 comps  8 comps
## CV      0.7343   0.7343
## adjCV   0.7307   0.7308
##
## 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
```

Hier ist es ganz eindeutig das Minimum der RMSEP Werte mit einer Anzahl von 3 PLS-Komponenten erreicht werden kann. Dies kann auch im untenstehenden Plot betrachtet werden.

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



Natürlich wird auch hier ein MSE Wert für die Vorhersagegenauigkeit des Modells berechnet.

```
pls_model <- plsr(lpsa~lcavol + lweight + age + lbph + svi + lcp + gleason + pgg45,
                  data = trainsh, scale=TRUE, validation = "CV")

pls_pred <- predict(pls_model, testsh, ncomp = 3)
library(Metrics)
mse(pls_pred, Y.test)
## [1] 0.4284326
```

Zuletzt können die Test MSE Werte von Ridge-, Lasso-, PCA-, und PLS-Regression verglichen werden. Anscheinend ergabe bei der Lasso-Regression der beste MSE Wert - welche durch das Weglassen der mit anderen Variablen korrelierte Variable lcp begründet ist.

```
mse(ridge.pred, Y.test)
## [1] 0.4045433
mse(lasso.pred, Y.test)
## [1] 0.392165
mse(pcr_pred, Y.test)
## [1] 0.5369458
mse(pls_pred, Y.test)
## [1] 0.4284326
```

6 Aufgabenblatt 6: Erzeugung von Zufallszahlen

Es werden nun Zufallszahlen mit Hilfe der Pseudozufallszahlengeneratoren erzeugt. Die Zufallszahlen heißen auch Pseudozufallszahlen, da diese Zahlen durch einen deterministischen Algorithmus berechnet wird und zufällig aussieht. Es gibt mehrere Pseudozufallszahlengeneratoren und hier werden nur die beiden lineare Kongruenz Generator und Mersenne Twister Generator angewendet. Beim ersten Aufruf des Generators muss vorher immer ein Startwert (englisch seed) gewählt werden. Der lineare Kongruenz Generator

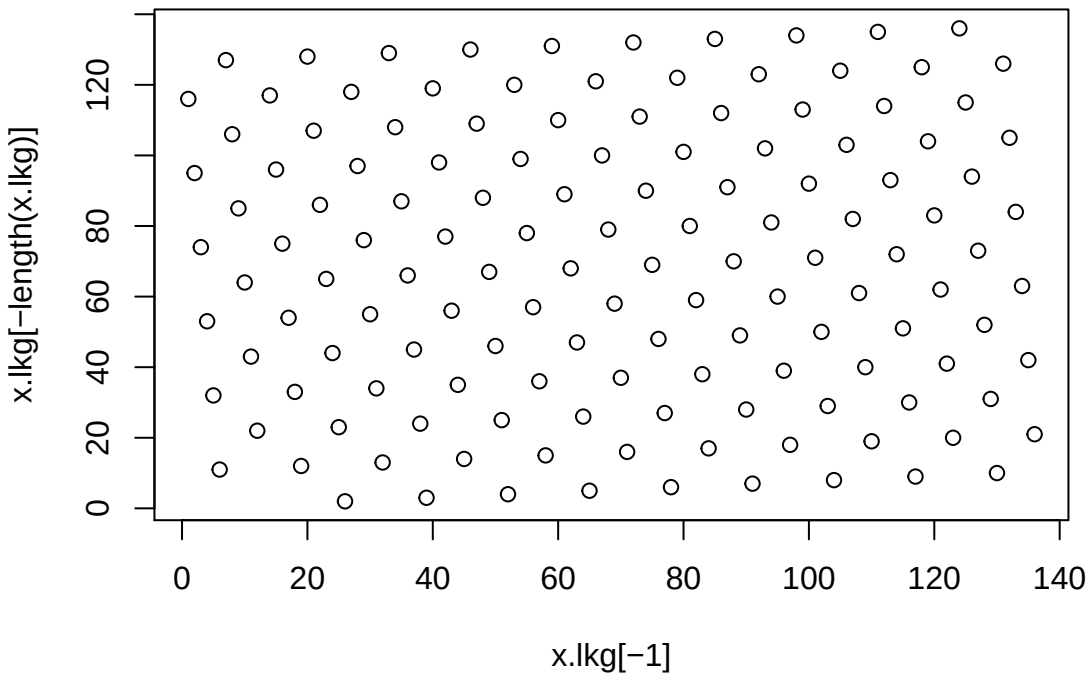
$$x_{i+1} = a \cdot x_i + b \quad (mod\ m)$$

ist ein rekursiver arithmetischer Zufallszahlengenerator, da die neuen Zufallszahlen aus den vorhergehenden Zahlen berechnet werden. Es muss ein Anfansgswert (oder seed) x_0, Modulo m und die Parameter a und b gewählt werden. Zunächst wird der vorgegebene LKG $x_{i+1} = 13 \cdot x_i \ \ mod\ 137$ implementiert. Die neu erzeugten Pseudozufallszahlen werden zusätzlich unten ausgegeben.

```
#hier d als Startwert x0
lcg.rand <- function(n,d=1) {
  rng <- vector(length = n)
  m <- 137
  a <- 13
  c <- 0

  for (i in 1:n) {
    d <- (a * d + c) %% m
    rng[i] <- d
  }
  return(rng)
}

lcg.rand(n=136)
##   [1]  13  32   5  65  23  25  51 115 125 118  27  77  42 135 111  73 127
##  [18]   7  91  87  35  44  24  38  83 120  53   4  52 128  20 123  92 100
##  [35]  67  49  89  61 108  34  31 129  33  18  97  28  90  74   3  39  96
##  [52]  15  58  69  75  16  71 101  80  81  94 126 131  59  82 107  21 136
##  [69] 124 105 132  72 114 112  86  22  12  19 110  60  95   2  26  64  10
##  [86] 130  46  50 102  93 113  99  54  17  84 133  85   9 117  14  45  37
## [103]  70  88  48  76  29 103 106   8 104 119  40 109  47  63 134  98  41
## [120] 122  79  68  62 121  66  36  57  56  43  11   6  78  55  30 116   1
```

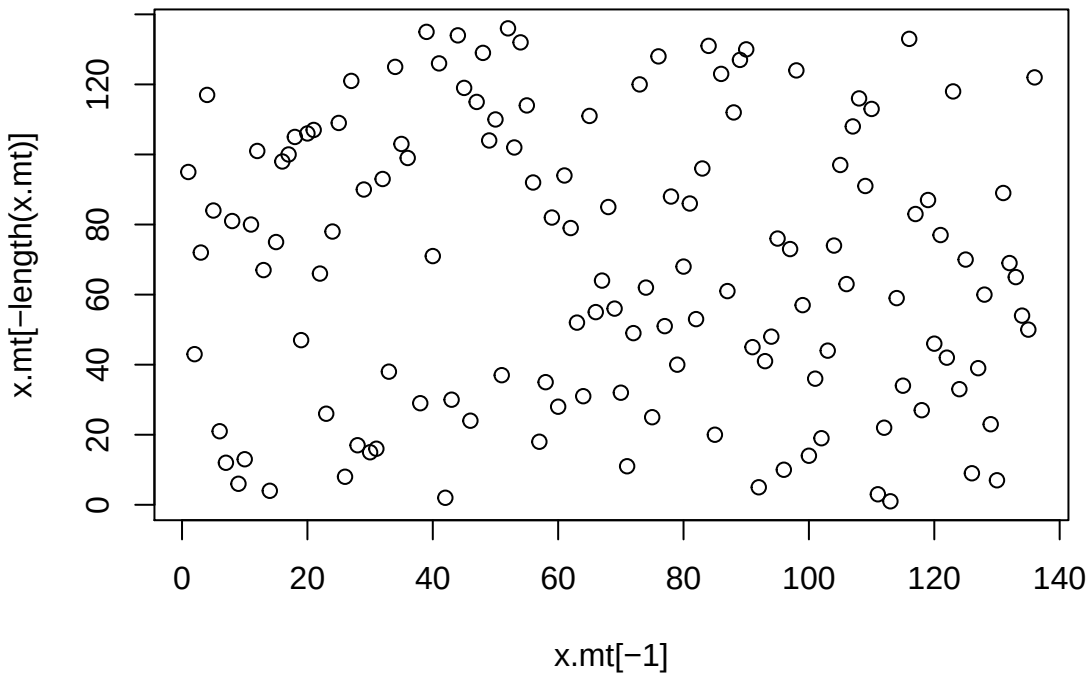
Die Zufallszahlen x_i werden nun als Paare von aufeinanderfolgenden Zufallszahlen $(x_1, x_2), (x_2, x_3), \ldots, (x_{135}, x_{136})$ als Punkte in einer zweidimensionalen Ebende dargestellt.

```
x.lkg <- lcg.rand(n=136)
plot(x.lkg[-1],x.lkg[-length(x.lkg)])
```



Ein besserer Zufallsgenerator ist der Mersenne Twister Generator und ist auch daher der Standard-Generator für viele Softwareprogramme (auch hier in R). Näheres zu Mersenne Twister Generator kann in [SG16] nachgelesen werden. Es wird hier nun Zufallszahlen y_i mit Hilfe der Mersenne Twister Generator erstellt und wie bei der LKG als Paare von aufeinanderfolgenden Zufallszahlen $(y_1, y_2), (y_2, y_3), \ldots, (y_{135}, y_{136})$ geplotet.

```
set.seed(kind="Mersenne-Twister",seed=1) #hier d=1 Startwert
vec <- 1:136
x.mt <- sample(vec, 136)
plot(x.mt[-1],x.mt[-length(x.mt)])
```



Es ist hier eindeutig sichtbar, dass bei der LKG Muster und bei der Mersenne Twister Generator keine Muster erkennbar ist. Daher kann auch der Mersenne Twister Generator als der bessere Pseudozufallszahlengenerator erklärt werden.

Nun sollen zuerst die beiden folgenden LKG

$$\begin{aligned} x_{i+1}^{(1)} &= 40014 \cdot x_i \ mod\ 2147483563 \\ x_{i+1}^{(2)} &= 40692 \cdot x_i \ mod\ 2147483399 \end{aligned}$$

berechnet und dann sollen die Ergebnisse wie folgt

$$z_i = \left(x_i^{(1)} + x_i^{(2)} \right) \ mod\ 2147483563$$

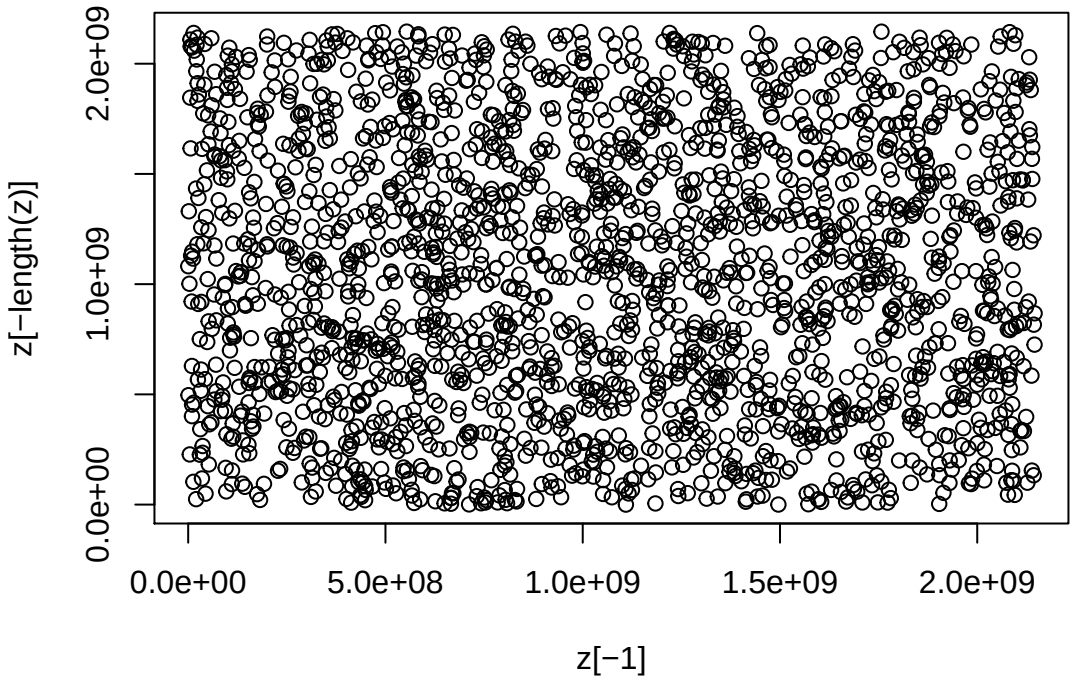
kombiniert werden. Diese sollen dann im Anschluss noch mit dem Mersenne Twister Generator verglichen werden.

```
# lcgneu.rand(n=136, a=13, m=137, d= 1)ergibt gleiche Ergbenis

#alle Variablen frei gesetzt-damit man später ändern kann

#hier d als Startwert x0
lcgneu.rand <- function(n,a,m,d) {
  rng <- vector(length = n)
  for (i in 1:n) {
```

```
    d <- (a * d ) %% m
    rng[i] <- d
  }
  return(rng)
}

x1 <- lcgneu.rand(n=2147, a=40014, m=2147483563, d=1)
x2 <- lcgneu.rand(n=2147, a=40692, m=2147483399, d=1)

hilfe <- function(x1,x2){
  (x1+x2)%%2147483563
}
```

Diese neue Zufallszahl z_i wird, wie oben schon einmal durchgeführt, geplotet.

```
z<- hilfe(x1,x2)
plot(z[-1],z[-length(z)])
```



Es ist hier erkennbar, dass die Zufallszahlen gegen 0 konvergieren, wobei dies bei dem Mersenne Twister Generator nicht der Fall ist. Somit ist dies wieder ein Grund das auch beim optimierten bzw. verbesserten LKG Modellen der Mersenne Twister Generator die deutlich bessere Auswahl ist.

7 Aufgabenblatt 7: Erzeugung von Gammaverteilten Zufallszahlen

Es wird hier (nicht gleichverteilte) 1000 Zufallszahlen mit Hilfe der Verwerfungsmethode für die Dichte der Gammaverteilung $Gamma\left(\frac{3}{2},1\right)$

$$f(x) = \frac{1}{\Gamma\left(\frac{3}{2}\right)} \cdot x^{\frac{1}{2}} \cdot e^{-x} = \frac{2}{\pi} \cdot x^{\frac{1}{2}} \cdot e^{-x}, \quad x > 0$$

mit

$$h(x) = \frac{2}{3} \cdot e^{\frac{-2x}{3}} \quad \text{und} \quad c = \frac{3^{\frac{3}{2}}}{(2\pi e)^{\frac{1}{2}}}$$

erzeugt.

Für die Anwendung der Verwerfungsmethode wird zunächst $h(x)$ integriert

$$\begin{aligned} H(x) &= \int_0^x h(x)dx = \int_0^x \frac{2}{3} \cdot e^{\frac{-2x}{3}} dx = \frac{2}{3} \cdot \int_0^x e^{\frac{-2x}{3}} dx \\ &= \frac{2}{3} \cdot \int e^z \cdot \left(-\frac{3}{2}\right) dz = \frac{2}{3} \cdot \left(-\frac{3}{2}\right) \cdot \int e^z dz \\ &= -\int e^z dz = -e^z| = -e^{\frac{-2x}{3}}|_0^x = 1 - e^{\frac{-2x}{3}} \end{aligned}$$

und anschließend die Umkehrfunktion gebildet $H^{-1}(y)$.

```
f <- function(x){
  o <- (2/pi)*x^(1/2)*exp(-x)
  return(o)
}

h <- function(x){
  z <- (2/3)*exp(-2*x/3)
  return(z)
}

c <- (3^(3/2))/(2*pi*exp(1))^(1/2)

h_inf<- function(x){
  l <- (-3/2)*log(1-x, base = exp(1))
  return(l)
}
```

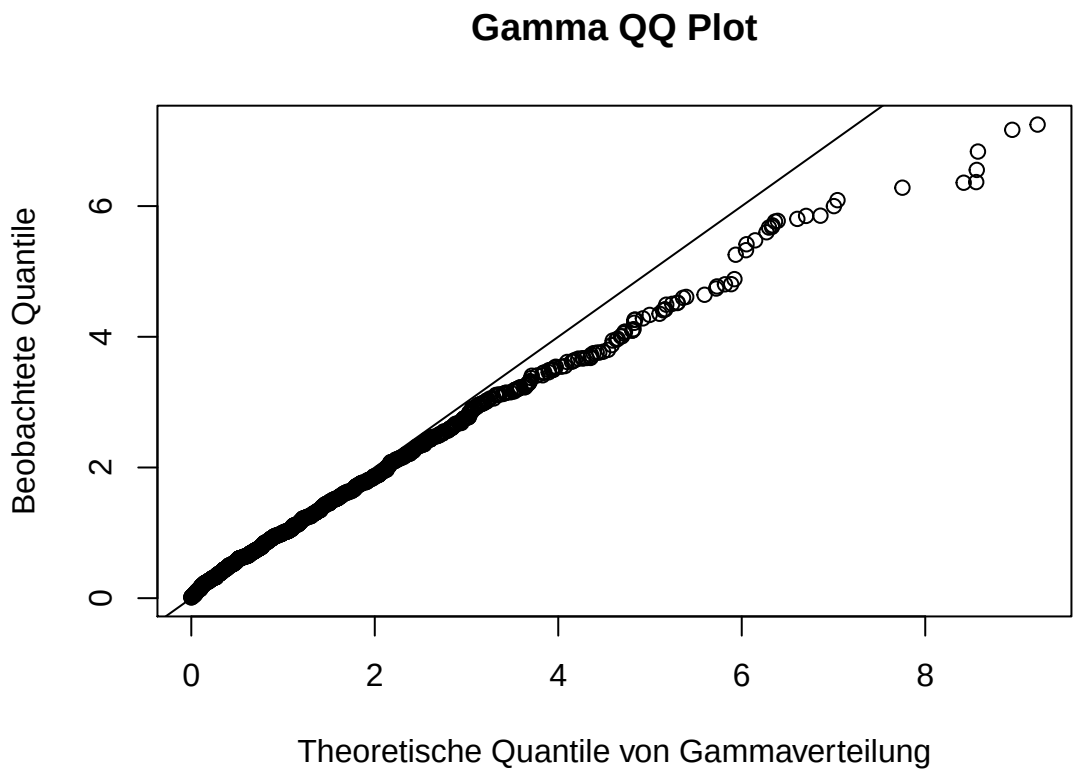
Jetzt kann die Verwerfungsmethode implementiert werden. Es wird zunächst Zufallszahlen Y aus h und U gleichverteilte Zufallszahlen erzeugt. Falls die Bedingung $U \leq \frac{f(y)}{c \cdot h(y)}$ gilt, werden diese erzeugte Zufallszahl akzeptiert. Dieser Vorgang wird dann so oft wiederholt bist 1000 Zufallszahlen erzeugt wurden.

```
x <- NULL
while(length(x) < 1000){
  loop = TRUE
  while(loop){
    s <- runif(1,0,1)
    y <- h_inf(s)
    u<- runif(1,0,1)
```

```
    abfrage <- u<= f(y)/(c*h(y))
    if(abfrage){
      x <- c(x,y) # in das leere Vektor werden jetzt die y drangehängt,
                  # die die Abfrage erfüllt haben
      loop = FALSE
    }
  }
}
```

Zusätzlich wird die Erzeugung der gammaverteilte Zufallszahlen mit der Verwerfungsmethode anhand der QQ Plot betrachtet. Hier ist es zu sehen, dass die Punkte für die höheren Quantile deutlich von der Referenzlinie abweichen.

```
rr<- rgamma (n=1000, shape=1, scale=3/2)
qqplot(rr,x, xlab = 'Theoretische Quantile von Gammaverteilung', ylab = 'Beobachtete Quantile',
       main = 'Gamma QQ Plot')
abline(0,1)
```



8 Aufgabenblatt 8: Monte Carlo

ALs letztes wird die Monte Carlo Integration mit Samples Mean Verfahren der Standardnormalverteilung

$$P\left(-1,96 \leq X \leq 1,96\right) = \int_{-1,96}^{1,96} \frac{1}{\sqrt{2\pi}} \cdot e^{-0,5x^2} dx$$

implementiert. Gesucht ist hier

$$I = \int_a^b g(x)dx = \int_a^b \frac{g(x)}{f(x)} f(x)dx$$

mit $g(x) = \frac{1}{\sqrt{2\pi}} \cdot e^{-0,5x^2}$ und der Wahrscheinlichkeitsdichte

$$f(x) = \begin{cases} >0 & \text{für alle } x \in [a,b] \text{ mit } g(x) \neq 0 \\ = 0 & \text{für alle } x \in [a,b] \text{ mit } g(x) = 0 \text{ und für alle } x \in \mathbb{R} \setminus [a,b] \end{cases} \tag{1}$$

Dann gilt für jede nach $f(x)$ verteilte Zufallsgröße, dass der Integral als Erwartungswert einer Zufallsgröße bestimmt werden kann:

$$\mathbb{E}\left(\frac{g(x)}{f(x)}\right)$$

Sei x nun auf $[a,b]$ gleichverteilt, das heißt

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{für alle } x \in [a,b] \\ 0 & \text{sonst} \end{cases} \tag{2}$$

somit gilt dann für $\mathbb{E}\left(\frac{g(x)}{f(x)}\right) = \mathbb{E}\left((b-a) \cdot g(x)\right) = (b-a) \cdot \mathbb{E}(g(x)) = I$. Zum Schluss wird I mit Hilfe des arithmetischen Mittels einer Stichprobe geschätzt:

$$\hat{I} = \frac{b-a}{N} \cdot \sum_{i=1}^{N} g(x_i)$$

```
t <-runif(1000,0,1)

g <- function(x){
  z <- (1/sqrt(2*pi))*exp(-0.5*x^2)
  return(z)
}

a = -1.96
b = 1.96

((b-a)/1000)*sum(g(t))
## [1] 1.341738
```

Literatur

[BOOT] Angelo Canty, Brian Ripley: https://cran.r-project.org/web/packages/boot/boot.pdf, aufgerufen am 21.06.2018

[HT13] Helge Toutenburg: Lineare Modelle, Springer-Verlag, 2013

[JW08] Jeffrey M. Wooldridge: Introductory Econometrics: A Modern Approach, Cengage Learning, 2008

[SG16] Stefan Gerlach: Computerphysik - Einführung, Beispiele und Anwendungen, Springer-Verlag, 2016