Exercise 6 - Cross Validation of Models

Konstantinos Vakalopoulos 12223236

2023-11-22

Contents

Task 1 2
Task 1.1 3
Task 1.2 5
Task 1.3 7
Task 1.4 9
Task 2 10
Task 2.1 12
Task 2.2 15
Task 2.3 18
Task 2.4 24

Task 1

We will work with the data set Auto in the ISLR package. Obtain information on the data set, its structure and the real world meaning of its variables from the help page.

In order to get an idea about the data, the command ?Auto must be executed. Therefore, the data consists of 9 variables and 392 observations. The variables are: mpg (miles per gallon), cylinders (Number of cylinders between 4 and 8), displacement (Engine displacement (cu. inches)), horsepower (Engine horsepower), weight (Vehicle weight (lbs.)), acceleration (Time to accelerate from 0 to 60 mph (sec.)), year (Model year (modulo 100)), origin (Origin of car (1. American, 2. European, 3. Japanese)) and name (Vehicle name). The original data contained 408 observations but 16 observations with missing values were removed.

For additional information, to get an intuition about the data, the following commands were executed.

```
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 4.2.3
```

```
data(Auto)
head(Auto)
```

```
##   mpg cylinders displacement horsepower weight acceleration year origin
## 1  18         8          307        130   3504         12.0   70      1
## 2  15         8          350        165   3693         11.5   70      1
## 3  18         8          318        150   3436         11.0   70      1
## 4  16         8          304        150   3433         12.0   70      1
## 5  17         8          302        140   3449         10.5   70      1
## 6  15         8          429        198   4341         10.0   70      1
##                       name
## 1 chevrolet chevelle malibu
## 2         buick skylark 320
## 3        plymouth satellite
## 4             amc rebel sst
## 5               ford torino
## 6          ford galaxie 500
```

```
summary(Auto)
```

```
##       mpg          cylinders      displacement     horsepower        weight
##  Min.   : 9.00   Min.   :3.000   Min.   : 68.0   Min.   : 46.0   Min.   :1613
##  1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0   1st Qu.: 75.0   1st Qu.:2225
##  Median :22.75   Median :4.000   Median :151.0   Median : 93.5   Median :2804
##  Mean   :23.45   Mean   :5.472   Mean   :194.4   Mean   :104.5   Mean   :2978
##  3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8   3rd Qu.:126.0   3rd Qu.:3615
##  Max.   :46.60   Max.   :8.000   Max.   :455.0   Max.   :230.0   Max.   :5140
##
##   acceleration        year           origin                      name
##  Min.   : 8.00   Min.   :70.00   Min.   :1.000   amc matador       :  5
##  1st Qu.:13.78   1st Qu.:73.00   1st Qu.:1.000   ford pinto        :  5
##  Median :15.50   Median :76.00   Median :1.000   toyota corolla    :  5
##  Mean   :15.54   Mean   :75.98   Mean   :1.577   amc gremlin       :  4
##  3rd Qu.:17.02   3rd Qu.:79.00   3rd Qu.:2.000   amc hornet        :  4
##  Max.   :24.80   Max.   :82.00   Max.   :3.000   chevrolet chevette:  4
##                                                  (Other)           :365
```

```
cat("The size of the data is: ", dim(Auto))
```

```
## The size of the data is:  392 9
```

Task 1.1

Fit the following models
mpg ~ horsepower
mpg ~ poly(horsepower,2)
mpg ~ poly(horsepower,3)

Visualize all 3 models in comparison added to a scatterplot of the input data.

For all the data set, we specifically took the variables horsepower and mpg, which are the independent and dependent variables, respectively. Afterwards, we fitted the models using the lm() function and a scatterplot pf the input data and the three models was created.

```
df <- Auto[c("mpg", "horsepower")] # subset

linear.model <- lm(mpg ~ horsepower, data=df) # linear
quadratic.model <- lm(mpg ~ poly(horsepower,2), data=df) # quadratic
cubic.model <- lm(mpg ~ poly(horsepower,3), data=df) # cubic
```

We import the library ggplot.

```
if (!require("ggplot2")) install.packages("ggplot2")
```

```
## Loading required package: ggplot2
```

```
library(ggplot2)
```

Finally, the scatter plot was created.

```
ggplot(df, aes(x = horsepower, y = mpg)) +
  geom_point() +
  labs(title = "Scatterplot",
       x = "Horsepower",
       y = "Miles Per Gallon")+
  theme(plot.title = element_text(hjust = 0.5))+
  geom_smooth(method='lm', formula= y ~ x, se = F, aes(color='Linear')) +
  geom_smooth(method='lm', formula= y ~ poly(x,2), se = F, aes(color='Quadratic')) +
  geom_smooth(method='lm', formula= y ~ poly(x,3), se = F, aes(color='Cubic')) +
  scale_color_manual(name='Models',
                     breaks=c('Linear', 'Quadratic', 'Cubic'),
                     values=c('Cubic'='green', 'Quadratic'='blue', 'Linear'='red'))
```

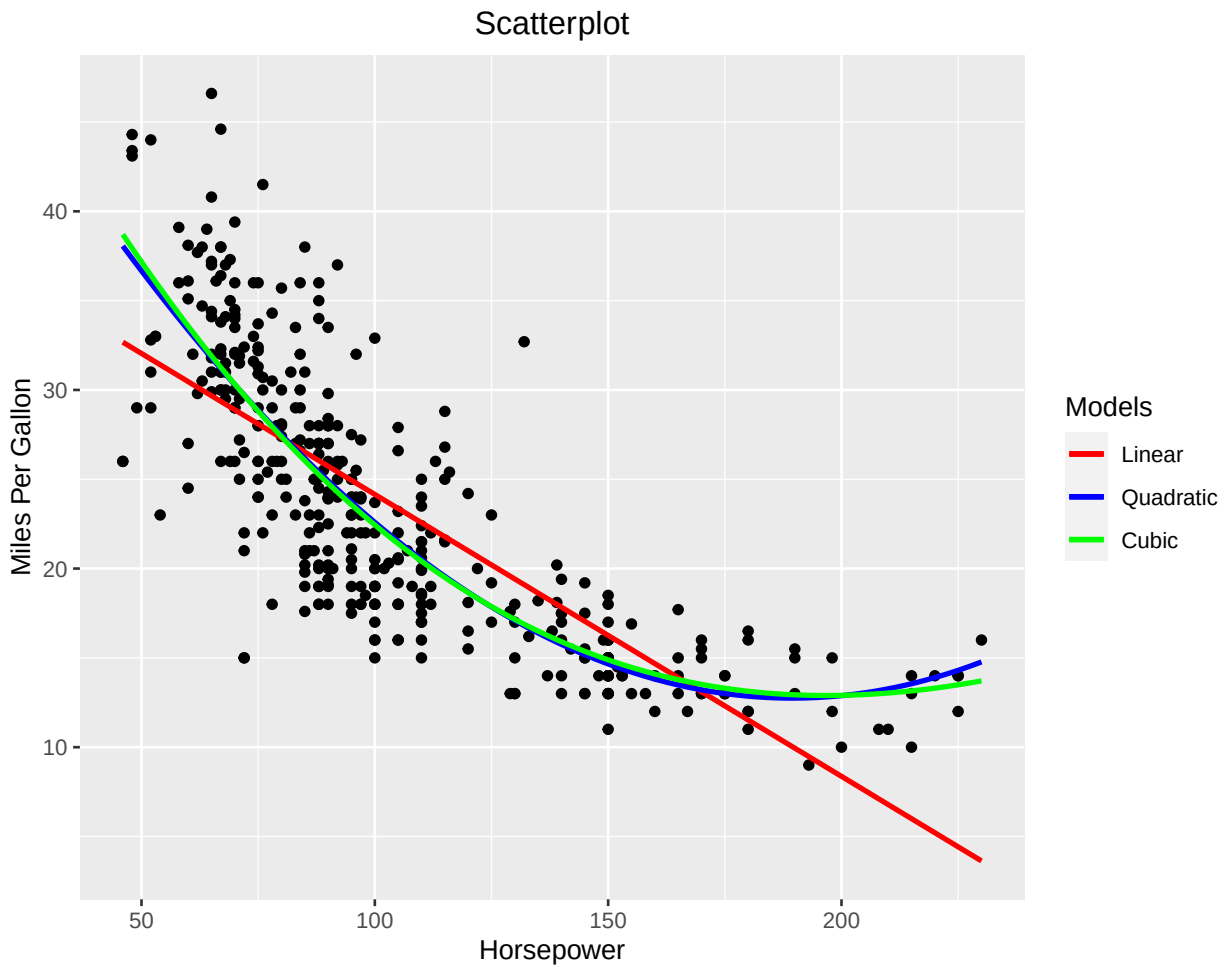


Figure 1: Scatterplot with the 3 models

Task 1.2

Use the validation set approach to compare the models. Use once a train/test split of 50%/50% and once 70%/30%. Choose the best model based on Root Mean Squared Error, Mean Squared Error and Median Absolute Deviation.

For this task, a function performance() was created. The specific function takes as input the model (linear, quadratic or cubic), the data and the percentage of the split (50%/50% or 70%/30%). Based on a seed (12223236), for reproducibility purposes, split the data, iusing the sample() command, into train and test set. With the train data, the corresponding model is created and then according to the test data, the yhat (y_pred) are calculated using the command predict(). Finally, the Root Mean Squared Error, Mean Squared Error and Median Absolute Deviation are calculated and returned from the function.

```
performance <- function(model, data, per){
  # Split the data
  set.seed(12223236)
  n <- nrow(data)
  train <- sample(1:n,round(n*per))
  test <- (1:n)[-train]

  # Check the model
  if (model == "Linear Model"){
    model <- lm(mpg ~ horsepower, data=data[train,])
  }
  else if (model == "Quadratic Model"){
    model <- lm(mpg ~ poly(horsepower,2), data=data[train,])
  }
  else {
    model <- lm(mpg ~ poly(horsepower,3), data=data[train,])
  }

  # Predict the test data according to the corresponding model
  y_test <- data$mpg[test]
  y_pred <- predict(model, newdata = data[test,])

  # Performance Metrics
  RMSE <- sqrt(mean((y_test-y_pred)^2))
  MSE <- mean((y_test-y_pred)^2)
  MAD <- median(abs((y_test-y_pred-median(y_test-y_pred))))

  result <- data.frame(RMSE = RMSE, MSE = MSE, MAD = MAD)
  return(result)
}
```

The models are:

```
models <- c("Linear Model", "Quadratic Model", "Cubic Model")
```

For the train/test split of 50%/50%:

```
# 50% train and test split
result.50 <- rbind(performance(models[1],df,0.5),
                   performance(models[2],df,0.5),
                   performance(models[3],df,0.5))
```

```
rownames(result.50) <- c("Linear_model", "Quadratic_model", "Cubic_model")
colnames(result.50) <- c("MSE", "RMSE", "MAD")
```

```
cat("The best model based on Root Mean Squared Error is: ",
    models[which.min(result.50$RMSE)])
```

```
## The best model based on Root Mean Squared Error is:  Quadratic Model
```

```
cat("\nThe best model based on Mean Squared Error is: ",
    models[which.min(result.50$MSE)])
```

```
##
## The best model based on Mean Squared Error is:  Quadratic Model
```

```
cat("\nThe best model based on Median Absolute Deviation is: ",
    models[which.min(result.50$MAD)])
```

```
##
## The best model based on Median Absolute Deviation is:  Quadratic Model
```

Based on the results from the 3 performance metrics, the best model is the quadratic model.

For the train/test split of 70%/30%:

```
# 70% train and test split
result.70 <- rbind(performance(models[1],df,0.7),
                   performance(models[2],df,0.7),
                   performance(models[3],df,0.7))
rownames(result.70) <- c("Linear_model", "Quadratic_model", "Cubic_model")
colnames(result.70) <- c("MSE", "RMSE", "MAD")
```

```
cat("The best model based on Root Mean Squared Error is: ",
    models[which.min(result.70$RMSE)])
```

```
## The best model based on Root Mean Squared Error is:  Quadratic Model
```

```
cat("\nThe best model based on Mean Squared Error is: ",
    models[which.min(result.70$MSE)])
```

```
##
## The best model based on Mean Squared Error is:  Quadratic Model
```

```
cat("\nThe best model based on Median Absolute Deviation is: ",
    models[which.min(result.70$MAD)])
```

```
##
## The best model based on Median Absolute Deviation is:  Quadratic Model
```

Once again, the best model is the quadratic model.

Task 1.3

Use the cv.glm function in the boot package for the following steps.

From the package "boot", the cv.glm function is introduced. The three models were created, using this time the function glm() and not the lm(). Finally, three functions, for the Root Mean Squared Error, Mean Squared Error and Median Absolute Deviation, were created.

```
if (!require("boot")) install.packages("boot")
```

```
## Loading required package: boot
```

```
library(boot)
```

```
mse <- function(y_test,y_pred){mean((y_test-y_pred)^2)}
rmse <- function(y_test,y_pred){sqrt(mean((y_test-y_pred)^2))}
mad <- function(y_test,y_pred){median(abs((y_test-y_pred-median(y_test-y_pred))))}
```

```
glm.linear.model <- glm(mpg ~ horsepower, data=df)
glm.quadratic.model <- glm(mpg ~ poly(horsepower,2), data=df)
glm.cubic.model <- glm(mpg ~ poly(horsepower,3), data=df)
```

Task 1.3.1 Use cv.glm for Leave-one-out Cross Validation to compare the models above.

First we create two lists for the three models and the three performance metrics.

```
all.models <- list(glm.linear.model, glm.quadratic.model, glm.cubic.model)
cost.functions <- list(mse,rmse,mad)
```

For those, the Leave-one-out Cross Validation is performed using the cv.glm function. The function takes as input the data, the model and the cost function. The first element of delta is the raw cross-validation estimate of prediction error and the second is the adjusted one. We want the $delta[1].

```
# LOOCV
startTime <- Sys.time()
LOOCV <- matrix(0, nrow = 3, ncol = 3)

for (i in 1:nrow(LOOCV)){
  for (j in 1:ncol(LOOCV)){
    set.seed(12223236)
    LOOCV[i,j] <- cv.glm(df, all.models[[i]], cost = cost.functions[[j]])$delta[1]
  }
}
endTime <- Sys.time()
# prints recorded time
print(endTime - startTime)
```

```
## Time difference of 18.42236 secs
```

Task 1.3.2 Use cv.glm for 5-fold and 10-fold Cross Validation to compare the models above.

The same procedure is done for the 5-fold and 10-fold Cross Validation. The only difference is that we have to initialize the K (5 or 10) in the cv.glm function, which indicates the number of folds.

For the 5-fold Cross Validation:

```
# 5-fold CV
startTime <- Sys.time()
Fold_5_CV <- matrix(0, nrow = 3, ncol = 3)

for (i in 1:nrow(Fold_5_CV)){
  for (j in 1:ncol(Fold_5_CV)){
    set.seed(12223236)
    Fold_5_CV[i,j] <- cv.glm(df, all.models[[i]], cost = cost.functions[[j]],
                             K = 5)$delta[1]
  }
}
endTime <- Sys.time()
# prints recorded time
print(endTime - startTime)
```

```
## Time difference of 0.2285221 secs
```

For the 10-fold Cross Validation:

```
# 10-fold CV
startTime <- Sys.time()
Fold_10_CV <- matrix(0, nrow = 3, ncol = 3)

for (i in 1:nrow(Fold_10_CV)){
  for (j in 1:ncol(Fold_10_CV)){
    set.seed(12223236)
    Fold_10_CV[i,j] <- cv.glm(df, all.models[[i]], cost = cost.functions[[j]],
                              K = 10)$delta[1]
  }
}
endTime <- Sys.time()
# prints recorded time
print(endTime - startTime)
```

```
## Time difference of 0.433495 secs
```

We, finally, transform the results into data frames.

```
LOOCV <- as.data.frame(LOOCV)
rownames(LOOCV) <- c("Linear_model", "Quadratic_model", "Cubic_model")
colnames(LOOCV) <- c("MSE", "RMSE", "MAD")

Fold_5_CV <- as.data.frame(Fold_5_CV)
rownames(Fold_5_CV) <- c("Linear_model", "Quadratic_model", "Cubic_model")
colnames(Fold_5_CV) <- c("MSE", "RMSE", "MAD")
```

```
Fold_10_CV <- as.data.frame(Fold_10_CV)
rownames(Fold_10_CV) <- c("Linear_model", "Quadratic_model", "Cubic_model")
colnames(Fold_10_CV) <- c("MSE", "RMSE", "MAD")
```

It is worth mentioning that he differences in calculation time are evident, with Leave-One-Out Cross-Validation (LOOCV) taking more time and 5-Fold Cross-Validation taking less. This occurs because in the case of LOOCV, the number of folds is equal to the number of observations in the data.

Task 1.4

Compare all results from 2 and 3. in a table and draw your conclusions.

Bellow, all the results are presented in the tables.

```
knitr::kable(result.50, format = "markdown",caption = "Train/Test split of 50%/50%")
```

Table 1: Train/Test split of 50%/50%

	MSE	RMSE	MAD
Linear_model	4.789772	22.94191	2.821861
Quadratic_model	4.261677	18.16189	2.289694
Cubic_model	4.359549	19.00566	2.435924

```
knitr::kable(result.70, format = "markdown",caption = "Train/Test split of 70%/30%")
```

Table 2: Train/Test split of 70%/30%

	MSE	RMSE	MAD
Linear_model	4.872403	23.74031	2.869381
Quadratic_model	4.588216	21.05173	2.413048
Cubic_model	4.807300	23.11014	2.478854

```
knitr::kable(LOOCV, format = "markdown",caption = "Leave-One-Out Cross Validation")
```

Table 3: Leave-One-Out Cross Validation

	MSE	RMSE	MAD
Linear_model	24.23151	3.848748	0
Quadratic_model	19.24821	3.272041	0
Cubic_model	19.33498	3.276807	0

```
knitr::kable(Fold_5_CV, format = "markdown",caption = "5-Fold Cross Validation")
```

Table 4: 5-Fold Cross Validation

	MSE	RMSE	MAD
Linear_model	24.14306	4.909338	3.175597
Quadratic_model	19.13983	4.344416	2.424981
Cubic_model	19.18256	4.347780	2.317324

```
knitr::kable(Fold_10_CV, format = "markdown",caption = "10-Fold Cross Validation")
```

Table 5: 10-Fold Cross Validation

	MSE	RMSE	MAD
Linear_model	24.28011	4.902112	3.016887
Quadratic_model	19.18829	4.329869	2.509581
Cubic_model	19.30521	4.341145	2.479193

According to the results above, the optimal model is the quadratic model because in all cases, the performance metrics are the lowest. The only difference is observed in the Mean Absolute Deviation (MAD) for the 10-Fold and 5-Fold cross-validation, where the best model based on this metric is the cubic model. Furthermore, it's worth noting that in the case of Leave-One-Out Cross Validation, the MAD is zero because there is no median for a single value. In Leave-One-Out Cross Validation, essentially, the performance metric is computed for each individual observation.

Task 2

Load the data set 'economics' from the package 'ggplot2'.

From the package 'ggplot2', we load the data set 'economics'. Below, an intuition about the data is presented. The data set 'economics' consists of 574 observations and 6 variables. The variables are: date (Month of data collection), pce (personal consumption expenditures, in billions of dollars), pop (total population, in thousands), psavert (personal savings rate), uempmed (median duration of unemployment, in weeks) and unemploy (number of unemployed in thousands).

```
data(economics)
head(economics)
```

```
## # A tibble: 6 x 6
##   date         pce    pop psavert uempmed unemploy
##   <date>     <dbl>  <dbl>   <dbl>   <dbl>    <dbl>
## 1 1967-07-01  507. 198712    12.6     4.5     2944
## 2 1967-08-01  510. 198911    12.6     4.7     2945
## 3 1967-09-01  516. 199113    11.9     4.6     2958
## 4 1967-10-01  512. 199311    12.9     4.9     3143
## 5 1967-11-01  517. 199498    12.8     4.7     3066
## 6 1967-12-01  525. 199657    11.8     4.8     3018
```

```
cat("The size of the data is: ", dim(economics))
```

```
## The size of the data is:  574 6
```

```
summary(economics)
```

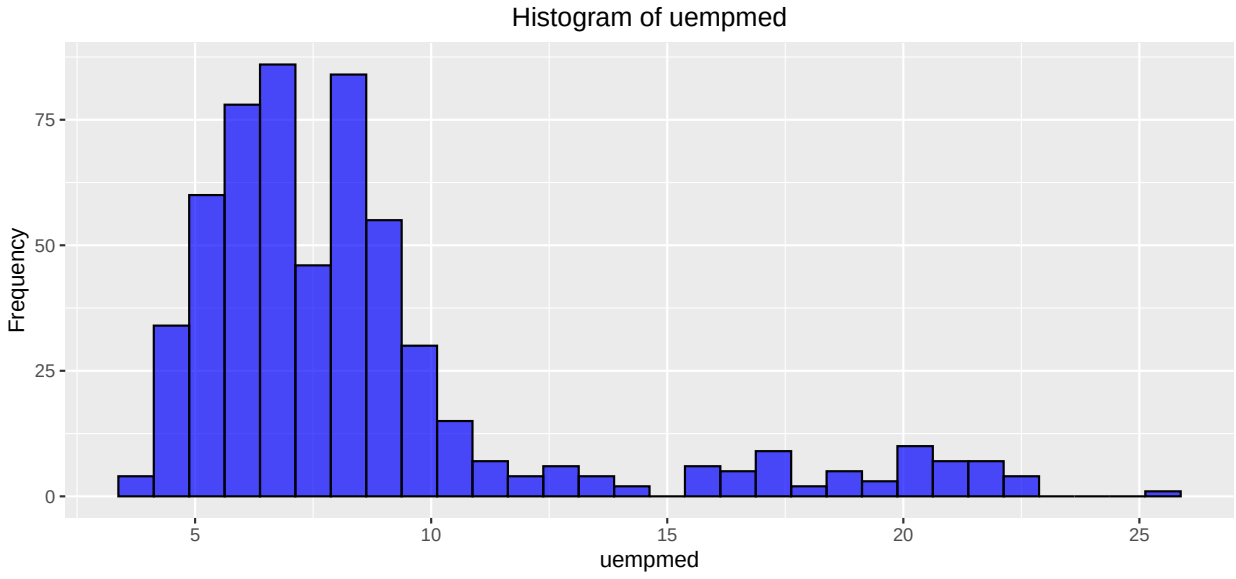
```
##       date                 pce              pop            psavert
##  Min.   :1967-07-01   Min.   :  506.7   Min.   :198712   Min.   : 2.200
##  1st Qu.:1979-06-08   1st Qu.: 1578.3   1st Qu.:224896   1st Qu.: 6.400
##  Median :1991-05-16   Median : 3936.8   Median :253060   Median : 8.400
##  Mean   :1991-05-17   Mean   : 4820.1   Mean   :257160   Mean   : 8.567
##  3rd Qu.:2003-04-23   3rd Qu.: 7626.3   3rd Qu.:290291   3rd Qu.:11.100
##  Max.   :2015-04-01   Max.   :12193.8   Max.   :320402   Max.   :17.300
##     uempmed          unemploy
##  Min.   : 4.000   Min.   : 2685
##  1st Qu.: 6.000   1st Qu.: 6284
##  Median : 7.500   Median : 7494
##  Mean   : 8.609   Mean   : 7771
##  3rd Qu.: 9.100   3rd Qu.: 8686
##  Max.   :25.200   Max.   :15352
```

We keep the two variables unemploy and uempmed.

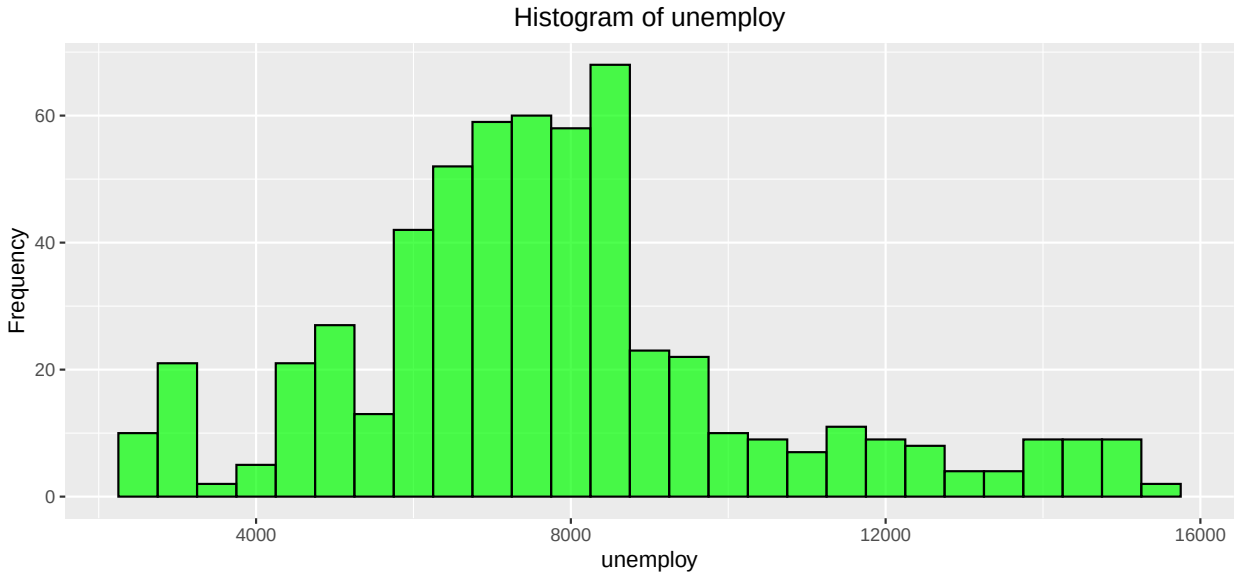
```
df_economics <- data.frame(economics[c("uempmed", "unemploy")])
```

The two histograms are used for demonstration purposes only in order to get an idea about how the data are distributed.

```
par(mfrow = c(1, 2))
ggplot(df_economics, aes(x = uempmed)) +
  geom_histogram(binwidth = 0.75, fill = "blue", color = "black", alpha = 0.7) +
  labs(title = "Histogram of uempmed", x = "uempmed", y = "Frequency") +
  theme(plot.title = element_text(hjust = 0.5))
```




```
ggplot(df_economics, aes(x = unemploy)) +
  geom_histogram(binwidth = 500, fill = "green", color = "black", alpha = 0.7) +
  labs(title = "Histogram of unemploy", x = "unemploy", y = "Frequency") +
  theme(plot.title = element_text(hjust = 0.5))
```



Task 2.1

Fit the following models to explain the number of unemployed persons 'unemploy' by the median number of days unemployed 'uempmed' and vice versa:

- linear model
- an appropriate exponential or logarithmic model (which one is appropriate depends on which is the dependent or independent variable)
- polynomial model of 2nd, 3rd and 10th degree

The dependent variable is the uempmed and independent is the unemploy. Here, we take the exponential model and at the same time we scale the independent variable unemploy. The explanation is given in the next task. The models are:

```
linear <- lm(uempmed ~ unemploy, data=df_economics)
print(linear)
```

```
##
## Call:
## lm(formula = uempmed ~ unemploy, data = df_economics)
##
## Coefficients:
## (Intercept)     unemploy
##   -1.892269     0.001351
```

```
exponential <- lm(uempmed ~ exp(scale(unemploy)), data=df_economics)
print(exponential)
```

```
##
## Call:
## lm(formula = uempmed ~ exp(scale(unemploy)), data = df_economics)
##
```

```
## Coefficients:
##          (Intercept)  exp(scale(unemploy))
##                6.459                 1.155
```

```
polynomial.2 <- lm(uempmed ~ poly(unemploy,2), data=df_economics)
print(polynomial.2)
```

```
##
## Call:
## lm(formula = uempmed ~ poly(unemploy, 2), data = df_economics)
##
## Coefficients:
##        (Intercept)  poly(unemploy, 2)1  poly(unemploy, 2)2
##              8.609              85.455              25.738
```

```
polynomial.3 <- lm(uempmed ~ poly(unemploy,3), data=df_economics)
print(polynomial.3)
```

```
##
## Call:
## lm(formula = uempmed ~ poly(unemploy, 3), data = df_economics)
##
## Coefficients:
##        (Intercept)  poly(unemploy, 3)1  poly(unemploy, 3)2  poly(unemploy, 3)3
##              8.609              85.455              25.738              -2.484
```

```
polynomial.10 <- lm(uempmed ~ poly(unemploy,10), data=df_economics)
print(polynomial.10)
```

```
##
## Call:
## lm(formula = uempmed ~ poly(unemploy, 10), data = df_economics)
##
## Coefficients:
##          (Intercept)   poly(unemploy, 10)1   poly(unemploy, 10)2
##               8.6087               85.4553               25.7382
##  poly(unemploy, 10)3   poly(unemploy, 10)4   poly(unemploy, 10)5
##              -2.4841               -8.8197               -7.0505
##  poly(unemploy, 10)6   poly(unemploy, 10)7   poly(unemploy, 10)8
##              -2.0052                3.7767                3.7564
##  poly(unemploy, 10)9  poly(unemploy, 10)10
##               0.2841                1.4818
```

The dependent variable is the unemploy and independent is the uempmed. This time, we take the logarithmic model. The models are:

```
linear <- lm(unemploy ~ uempmed, data=df_economics)
print(linear)
```

```
##
## Call:
```

```
## lm(formula = unemploy ~ uempmed, data = df_economics)
##
## Coefficients:
## (Intercept)      uempmed
##      2956.8        559.3
```

```
logarithmic <- lm(unemploy ~ log(uempmed), data=df_economics)
print(logarithmic)
```

```
##
## Call:
## lm(formula = unemploy ~ log(uempmed), data = df_economics)
##
## Coefficients:
##  (Intercept)  log(uempmed)
##        -4936          6144
```

```
polynomial.2 <- lm(unemploy ~ poly(uempmed,2), data=df_economics)
print(polynomial.2)
```

```
##
## Call:
## lm(formula = unemploy ~ poly(uempmed, 2), data = df_economics)
##
## Coefficients:
##       (Intercept)  poly(uempmed, 2)1  poly(uempmed, 2)2
##              7771              54977             -12797
```

```
polynomial.3 <- lm(unemploy ~ poly(uempmed,3), data=df_economics)
print(polynomial.3)
```

```
##
## Call:
## lm(formula = unemploy ~ poly(uempmed, 3), data = df_economics)
##
## Coefficients:
##       (Intercept)  poly(uempmed, 3)1  poly(uempmed, 3)2  poly(uempmed, 3)3
##              7771              54977             -12797               7004
```

```
polynomial.10 <- lm(unemploy ~ poly(uempmed,10), data=df_economics)
print(polynomial.10)
```

```
##
## Call:
## lm(formula = unemploy ~ poly(uempmed, 10), data = df_economics)
##
## Coefficients:
##         (Intercept)   poly(uempmed, 10)1   poly(uempmed, 10)2
##              7771.3              54976.6             -12796.9
##  poly(uempmed, 10)3   poly(uempmed, 10)4   poly(uempmed, 10)5
##              7003.6              -6532.0               2910.4
```

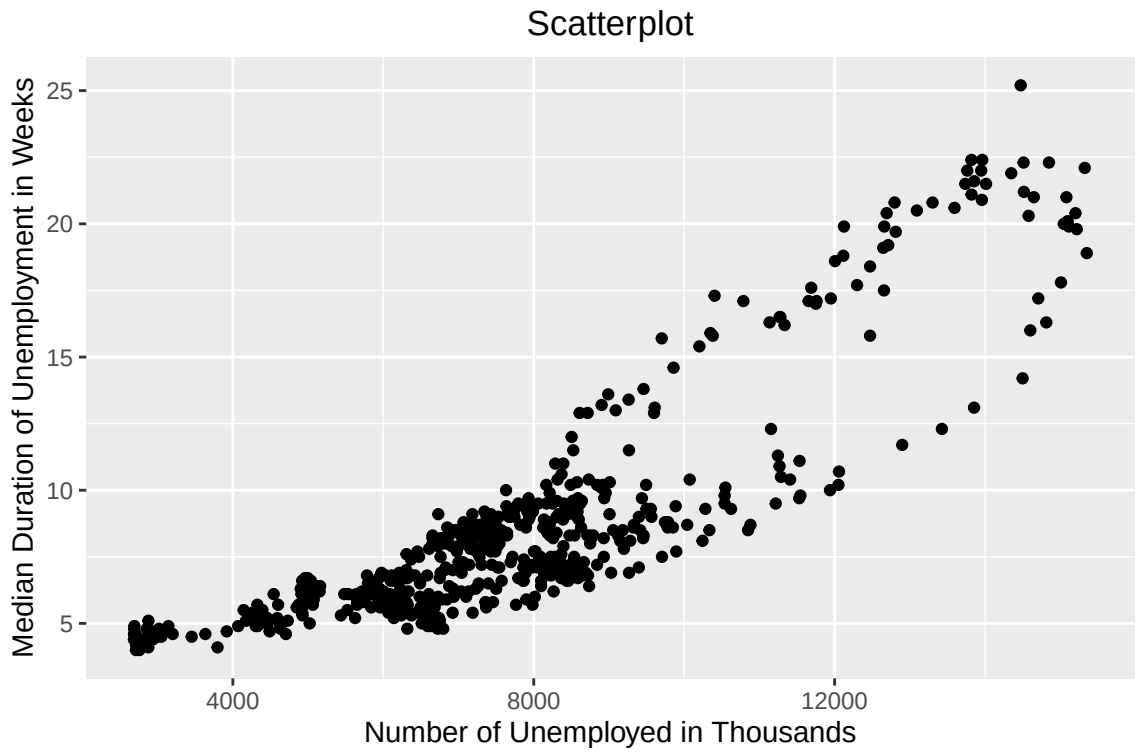
```
##  poly(uempmed, 10)6   poly(uempmed, 10)7   poly(uempmed, 10)8
##              -2355.6               2425.5                661.1
##  poly(uempmed, 10)9  poly(uempmed, 10)10
##              -2363.2               1047.9
```

Task 2.2

Plot the corresponding data and add all the models for comparison.

The dependent variable is the uempmed and independent is the unemploy. Here, a scatter plot is created.

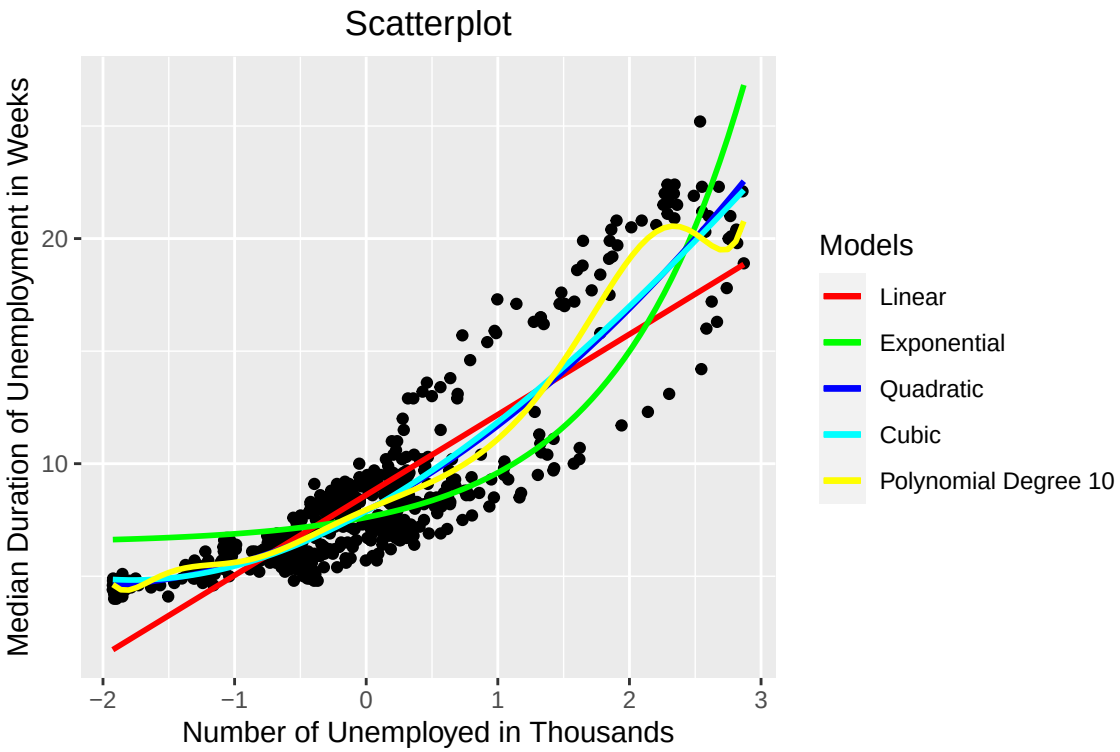
```
ggplot(df_economics, aes(x = unemploy, y = uempmed)) +
  geom_point() +
  labs(title = "Scatterplot",
       x = "Number of Unemployed in Thousands",
       y = "Median Duration of Unemployment in Weeks")+
  theme(plot.title = element_text(hjust = 0.5))
```



In order to include the corresponding models, we have to scale the independent variable, which is the number of unemployed in thousands because in case we take the exponential model and calculate the exponential of the variable unemploy, the result will be infinity. Thus, the scatter plot, including the models, will be:

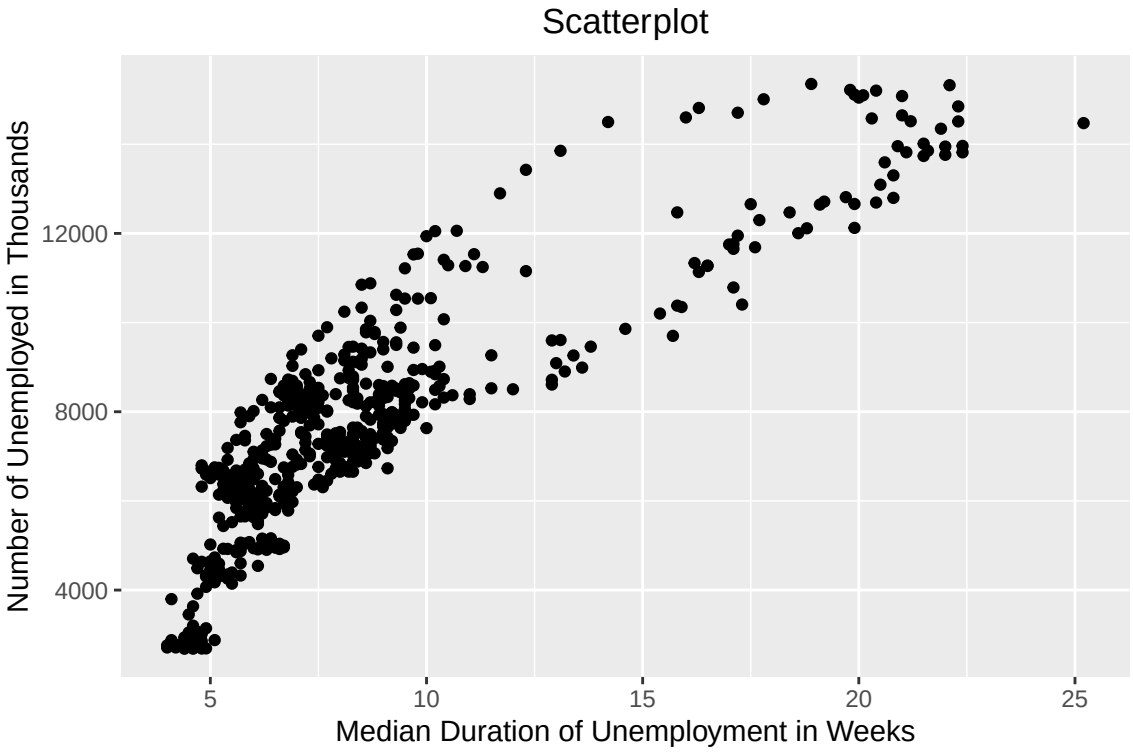
```
ggplot(df_economics, aes(x = scale(unemploy), y = uempmed)) +
  geom_point() +
  labs(title = "Scatterplot",
       x = "Number of Unemployed in Thousands",
       y = "Median Duration of Unemployment in Weeks")+
  theme(plot.title = element_text(hjust = 0.5)) +
  geom_smooth(method='lm', formula= y ~ x, se = F,
```

```
            aes(color='Linear')) +
  geom_smooth(method='lm', formula= y ~ exp(x), se = F,
            aes(color='Exponential')) +
  geom_smooth(method='lm', formula= y ~ poly(x,2), se = F,
            aes(color='Quadratic')) +
  geom_smooth(method='lm', formula= y ~ poly(x,3), se = F,
            aes(color='Cubic')) +
  geom_smooth(method='lm', formula= y ~ poly(x,10), se = F,
            aes(color='Polynomial Degree 10')) +
  scale_color_manual(name='Models',
                     breaks=c('Linear', 'Exponential',
                              'Quadratic',
                              'Cubic', 'Polynomial Degree 10'),
                     values=c('Cubic'='cyan', 'Quadratic'='blue',
                              'Linear'='red',
                              'Exponential' = "green",
                              'Polynomial Degree 10' = "yellow"))
```



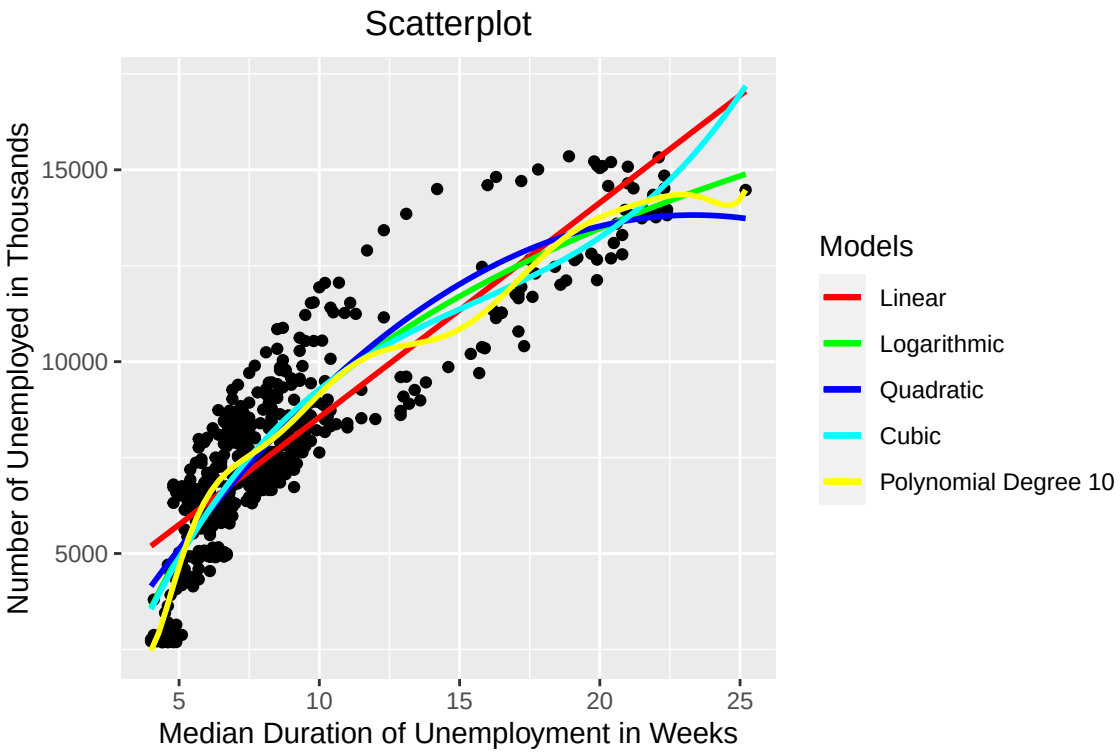
Now, the dependent variable is the unemploy and independent is the uempmed. Here, a scatter plot is created.

```
ggplot(df_economics, aes(x = uempmed, y = unemploy)) +
  geom_point() +
  labs(title = "Scatterplot",
       x = "Median Duration of Unemployment in Weeks",
       y = "Number of Unemployed in Thousands")+
  theme(plot.title = element_text(hjust = 0.5))
```



Including the models, the scatter plot will be:

```
ggplot(df_economics, aes(x = uempmed, y = unemploy)) +
  geom_point() +
  labs(title = "Scatterplot",
       x = "Median Duration of Unemployment in Weeks",
       y = "Number of Unemployed in Thousands")+
  theme(plot.title = element_text(hjust = 0.5)) +
  geom_smooth(method='lm', formula= y ~ x, se = F,
              aes(color='Linear')) +
  geom_smooth(method='lm', formula= y ~ log(x), se = F,
              aes(color='Logarithmic')) +
  geom_smooth(method='lm', formula= y ~ poly(x,2), se = F,
              aes(color='Quadratic')) +
  geom_smooth(method='lm', formula= y ~ poly(x,3), se = F,
              aes(color='Cubic')) +
  geom_smooth(method='lm', formula= y ~ poly(x,10), se = F,
              aes(color='Polynomial Degree 10')) +
  scale_color_manual(name='Models',
                     breaks=c('Linear', 'Logarithmic',
                              'Quadratic',
                              'Cubic', 'Polynomial Degree 10'),
                     values=c('Cubic'='cyan', 'Quadratic'='blue',
                              'Linear'='red',
                              'Logarithmic' = "green",
                              'Polynomial Degree 10' = "yellow"))
```



Task 2.3

Use the cv.glm function in the boot package for the following steps. Compare the Root Mean Squared Error and Mean Squared Error.

1. Use cv.glm for Leave-one-out Cross Validation to compare the models above.
2. Use cv.glm for 5-fold and 10-fold Cross Validation to compare the models above.

For this task, we are going to do the same procedure twice as we did in task 1.3, once for the dependent variable being the uempmed and independent the unemploy and once for the dependent variable being the unemploy and independent the uempmed. The models are created using the glm() function and a list with these models is created. For each model, Leave-One-Out, 5-Fold and 10-Fold cross validation is performed and the Root Mean Squared Error and Mean Squared Error are compared for each case. All the relusts are presented in tables.

Case 1 where the dependent variable is the uempmed and independent is the unemploy. Again we are going to scale the variable unemploy when the exponential model is created.

```
# Case 1 dependent = uempmed / independent = unemploy
# scaling
df_economics$unemploy <- scale(df_economics$unemploy)[1:length(df_economics$unemploy)]

#model creation
glm.linear.case.1 <- glm(uempmed ~ unemploy, data=df_economics)
glm.exponential.case.1 <- glm(uempmed ~ exp(unemploy), data=df_economics)
glm.polynomial.2.case.1 <- glm(uempmed ~ poly(unemploy,2), data=df_economics)
glm.polynomial.3.case.1<- glm(uempmed ~ poly(unemploy,3), data=df_economics)
```

```
glm.polynomial.10.case.1 <- glm(uempmed ~ poly(unemploy,10), data=df_economics)

all.models.case.1 <- list(glm.linear.case.1,
                          glm.exponential.case.1,
                          glm.polynomial.2.case.1,
                          glm.polynomial.3.case.1,
                          glm.polynomial.10.case.1)
```

For the Leave-One-Out cross validation:

```
# LOOCV
startTime <- Sys.time()
LOOCV <- matrix(0, nrow = 5, ncol = 2)

for (i in 1:nrow(LOOCV)){
  for (j in 1:ncol(LOOCV)){
    set.seed(12223236)
    LOOCV[i,j] <- cv.glm(df_economics, all.models.case.1[[i]],
                         cost = cost.functions[[j]])$delta[1]
  }
}
endTime <- Sys.time()
# prints recorded time
print(endTime - startTime)
```

```
## Time difference of 30.3409 secs
```

For the 5-Fold cross validation:

```
# 5-fold CV
startTime <- Sys.time()
Fold_5_CV <- matrix(0, nrow = 5, ncol = 2)

for (i in 1:nrow(Fold_5_CV)){
  for (j in 1:ncol(Fold_5_CV)){
    set.seed(12223236)
    Fold_5_CV[i,j] <- cv.glm(df_economics, all.models.case.1[[i]],
                             cost = cost.functions[[j]], K = 5)$delta[1]
  }
}
endTime <- Sys.time()
# prints recorded time
print(endTime - startTime)
```

```
## Time difference of 0.26739 secs
```

For the 10-Fold cross validation:

```
# 10-fold CV
startTime <- Sys.time()
Fold_10_CV <- matrix(0, nrow = 5, ncol = 2)
```

```
for (i in 1:nrow(Fold_10_CV)){
  for (j in 1:ncol(Fold_10_CV)){
    set.seed(12223236)
    Fold_10_CV[i,j] <- cv.glm(df_economics, all.models.case.1[[i]],
                              cost = cost.functions[[j]], K = 10)$delta[1]
  }
}
endTime <- Sys.time()
# prints recorded time
print(endTime - startTime)
```

```
## Time difference of 0.5062308 secs
```

In the next code chunk, we rename the rows and columns of each validation case.

```
LOOCV <- as.data.frame(LOOCV)
rownames(LOOCV) <- c("Linear_model", "Exponential_model",
                     "Quadratic_model", "Cubic_model",
                     "Polynomial_Degree_10")
colnames(LOOCV) <- c("MSE", "RMSE")

Fold_5_CV <- as.data.frame(Fold_5_CV)
rownames(Fold_5_CV) <- c("Linear_model", "Exponential_model",
                         "Quadratic_model", "Cubic_model",
                         "Polynomial_Degree_10")
colnames(Fold_5_CV) <- c("MSE", "RMSE")

Fold_10_CV <- as.data.frame(Fold_10_CV)
rownames(Fold_10_CV) <- c("Linear_model", "Exponential_model",
                          "Quadratic_model", "Cubic_model",
                          "Polynomial_Degree_10")
colnames(Fold_10_CV) <- c("MSE", "RMSE")
```

The results are shown in the below tables.

```
knitr::kable(LOOCV, format = "markdown",
             caption = "Leave-One-Out Cross Validation")
```

Table 6: Leave-One-Out Cross Validation

	MSE	RMSE
Linear_model	4.159797	1.602083
Exponential_model	4.925250	1.620154
Quadratic_model	3.005112	1.296986
Cubic_model	3.009934	1.309659
Polynomial_Degree_10	2.832303	1.211443

```
knitr::kable(Fold_5_CV, format = "markdown",
             caption = "5-Fold Cross Validation")
```

Table 7: 5-Fold Cross Validation

	MSE	RMSE
Linear_model	4.258705	2.053934
Exponential_model	4.968097	2.217411
Quadratic_model	3.075680	1.745838
Cubic_model	3.078449	1.746858
Polynomial_Degree_10	2.885243	1.687805

```
knitr::kable(Fold_10_CV, format = "markdown",
             caption = "10-Fold Cross Validation")
```

Table 8: 10-Fold Cross Validation

	MSE	RMSE
Linear_model	4.172859	2.033460
Exponential_model	4.983096	2.210892
Quadratic_model	3.036005	1.740171
Cubic_model	3.060614	1.746524
Polynomial_Degree_10	2.860747	1.684342

Case 2 where the dependent variable is the unemploy. and independent is the uempmed.

```
# Case 2 dependent = unemploy. / independent = uempmed.
df_economics <- data.frame(economics[c("uempmed", "unemploy")])

#model creation
glm.linear.case.2 <- glm(unemploy ~ uempmed, data=df_economics)
glm.logarithmic.case.2 <- glm(unemploy ~ log(uempmed), data=df_economics)
glm.polynomial.2.case.2 <- glm(unemploy ~ poly(uempmed,2), data=df_economics)
glm.polynomial.3.case.2 <- glm(unemploy ~ poly(uempmed,3), data=df_economics)
glm.polynomial.10.case.2 <- glm(unemploy ~ poly(uempmed,10), data=df_economics)


all.models.case.2 <- list(glm.linear.case.2,
                          glm.logarithmic.case.2,
                          glm.polynomial.2.case.2,
                          glm.polynomial.3.case.2,
                          glm.polynomial.10.case.2)
```

For the Leave-One-Out cross validation:

```
# LOOCV
startTime <- Sys.time()
LOOCV <- matrix(0, nrow = 5, ncol = 2)

for (i in 1:nrow(LOOCV)){
  for (j in 1:ncol(LOOCV)){
    set.seed(12223236)
    LOOCV[i,j] <- cv.glm(df_economics, all.models.case.2[[i]],
```

```
                          cost = cost.functions[[j]])$delta[1]
  }
}
endTime <- Sys.time()
# prints recorded time
print(endTime - startTime)
```

```
## Time difference of 30.01952 secs
```

For the 5-Fold cross validation:

```
# 5-fold CV
startTime <- Sys.time()
Fold_5_CV <- matrix(0, nrow = 5, ncol = 2)

for (i in 1:nrow(Fold_5_CV)){
  for (j in 1:ncol(Fold_5_CV)){
    set.seed(12223236)
    Fold_5_CV[i,j] <- cv.glm(df_economics, all.models.case.2[[i]],
                             cost = cost.functions[[j]], K = 5)$delta[1]
  }
}
endTime <- Sys.time()
# prints recorded time
print(endTime - startTime)
```

```
## Time difference of 0.260272 secs
```

For the 10-Fold cross validation:

```
# 10-fold CV
startTime <- Sys.time()
Fold_10_CV <- matrix(0, nrow = 5, ncol = 2)

for (i in 1:nrow(Fold_10_CV)){
  for (j in 1:ncol(Fold_10_CV)){
    set.seed(12223236)
    Fold_10_CV[i,j] <- cv.glm(df_economics, all.models.case.2[[i]],
                              cost = cost.functions[[j]], K = 10)$delta[1]
  }
}
endTime <- Sys.time()
# prints recorded time
print(endTime - startTime)
```

```
## Time difference of 0.495636 secs
```

In the next code chunk, we rename the rows and columns of each validation case.

```
LOOCV <- as.data.frame(LOOCV)
rownames(LOOCV) <- c("Linear_model", "Logarithmic_model",
                     "Quadratic_model", "Cubic_model",
                     "Polynomial_Degree_10")
colnames(LOOCV) <- c("MSE", "RMSE")

Fold_5_CV <- as.data.frame(Fold_5_CV)
rownames(Fold_5_CV) <- c("Linear_model", "Logarithmic_model",
                         "Quadratic_model", "Cubic_model",
                         "Polynomial_Degree_10")
colnames(Fold_5_CV) <- c("MSE", "RMSE")

Fold_10_CV <- as.data.frame(Fold_10_CV)
rownames(Fold_10_CV) <- c("Linear_model", "Logarithmic_model",
                          "Quadratic_model", "Cubic_model",
                          "Polynomial_Degree_10")
colnames(Fold_10_CV) <- c("MSE", "RMSE")
```

The results are shown in the below tables.

```
knitr::kable(LOOCV, format = "markdown",caption = "Leave-One-Out Cross Validation")
```

Table 9: Leave-One-Out Cross Validation

	MSE	RMSE
Linear_model	1715211	1040.0170
Logarithmic_model	1333997	980.4186
Quadratic_model	1432531	1012.2549
Cubic_model	1366405	984.5664
Polynomial_Degree_10	4530738	996.6290

```
knitr::kable(Fold_5_CV, format = "markdown",caption = "5-Fold Cross Validation")
```

Table 10: 5-Fold Cross Validation

	MSE	RMSE
Linear_model	1753052	1323.660
Logarithmic_model	1347965	1160.884
Quadratic_model	1443238	1201.278
Cubic_model	1378047	1173.864
Polynomial_Degree_10	32631244	3426.296

```
knitr::kable(Fold_10_CV, format = "markdown",caption = "10-Fold Cross Validation")
```

Table 11: 10-Fold Cross Validation

	MSE	RMSE
Linear_model	1727717	1310.204
Logarithmic_model	1342639	1157.811
Quadratic_model	1442845	1200.345
Cubic_model	1369452	1169.381
Polynomial_Degree_10	2631414	1409.873

According to the table results, in the first case the optimal model is the polynomial degree of 10 and in the second case the optimal model is the logarithmic. It is worth noting that the values of RMSE and MSE, in the second case, are large because there is a difference in the range of values of the two variables.

Task 2.4

Explain based on the CV and graphical model fits the concepts of Underfitting, Overfitting and how to apply cross-validation to determine the appropriate model fit. Also, describe the different variants of cross validation in this context.

Underfitting occurs when a model is too simple to capture the underlying patterns in the data. It results in a model that has high bias and low variance. An underfit model performs poorly on both the training data and new, unseen data because it cannot learn the relationships within the data effectively. On the other hand, overfitting occurs when a model is too complex and fits the training data too closely, capturing noise and random fluctuations in the data. It results in a model with low bias and high variance. An overfit model performs very well on the training data but poorly on new, unseen data because it has essentially memorized the training data rather than learned meaningful patterns.

In the case of each task, the linear model is unable to capture the underlying patterns in the data. This particular model essentially represents a straight line while the data is logarithmic and exponentially distributed. In the case of task 2, it is observed that the model with a polynomial degree of 10 overfits the data as it is more complex than the others. This is also evident from the scatter plot in task 2.2.

Below are presented the steps on how to apply cross-validation to determine the appropriate model fit.

1. Split Data: Divide your data set into a training set and a test set.
2. Select Models: Choose the models you want to evaluate.
3. K-Fold Cross-Validation: Use k-fold cross-validation, where 'k' is typically 5 or 10. This involves dividing the training data into 'k' subsets or folds.
4. Training and Testing: Perform the following steps 'k' times:
    - Train each model on 'k-1' of the folds.
    - Test the model on the remaining fold.
    - Record the model's performance metric.
5. Evaluate Models: After 'k' iterations, you will have 'k' performance scores for each model.
6. Average and Compare: Calculate the average performance score for each model across all 'k' iterations. The model with the best average score is the most appropriate fit.
7. Final Testing: Once the best model has been chosen, we can perform a final evaluation on your validation or test set to ensure it generalizes well.

In our case, we had 5-Fold, 10-Fold, and Leave-One-Out cross-validation, which means k = 5, k = 10, and k = the length of the data set, respectively. It is worth noting that, in each validation case, the execution time was calculated, and it is evident that LOOCV is the most computationally expensive procedure.