Homework Assignment 1

Data Science and Machine Learning 1 - CEU 2018

Kristof Menyhert

2018-01-27

Load the required packages:

```
library(data.table)
library(ggplot2)
library(GGally)
library(caret)
library(rpart)
library(scales)
theme_set(theme_bw()) #globally set ggplot theme to black & white
```

1. Model selection with a validation set

Load the data and tidy it, then create log_price variable, and set the seed:

```
data <- fread("C:/Users/Chronos/OneDrive - Central European University/R/machine_learning1/hw1/kc_house

data[, `:=`(floors = as.numeric(floors), zipcode = factor(zipcode))]
data[, log_price := log(price)]

data[, c("id", "date", "sqft_living15", "sqft_lot15", "price") := NULL]

set.seed(1234)
```

a) Using createDataPartition, cut your data into three parts: 50% should be your training data, 25% each your validation and test sets (hint: cut data into two parts, then further cut one part into two):

```
# 1st cut:
cut <- createDataPartition(y = data$log_price, times = 1, p = 0.5, list = FALSE)

data_train <- data[cut, ]
data_validation_test <- data[-cut, ]

# Further cut:
cut2 <- createDataPartition(y = data_validation_test$log_price, times = 1, p = 0.5,
    list = FALSE)

data_validation <- data_validation_test[cut2, ]
data_test <- data_validation_test[-cut2, ]

### check spliting:
length(data$log_price) == (length(data_train$log_price) + length(data_validation$log_price) +
    length(data_test$log_price))
```

```
## [1] TRUE
```

```
# If true we are likely did a good spliting
```

b) Train three models on the training data via caret, without cross validation (method = "none"):

Now I am filling in the missing details to train 3 models:

```
# fill in the missing details
train_control <- trainControl(method = "none")

tune_grid <- data.frame("cp" = c(0.0001))

# simple lm:
simple_linear_fit <- train(log_price ~ sqft_living,
                    data = data_train,
                    method = "lm",
                    trControl = train_control)

# multivariable lm:
linear_fit <- train(log_price ~ .,
                    data = data_train,
                    method = "lm",
                    trControl = train_control)

# regression tree:
rpart_fit <- train(log_price ~ .,
                    data = data_train,
                    method = "rpart",
                    trControl = train_control,
                    tuneGrid = tune_grid)
```

c) Compare your models on the validation set and choose the one with the best performance (using RMSE). Use predict.train for prediction just like we used predict in class.

Define RMSE:

```
RMSE <- function(x, true_x) sqrt(mean((x - true_x)^2))
```

Calculate RMSE:

```
# predict results based on the models above:
simple_linear_fit_data_validation<- predict.train(simple_linear_fit, newdata = data_validation)

linear_fit_data_validation<- predict.train(linear_fit, newdata = data_validation)

rpart_fit_fit_data_validation<- predict.train(rpart_fit, newdata = data_validation)

# calculate RMSE:
simple_linear_rmse <- RMSE(data_validation$log_price, simple_linear_fit_data_validation)
linear_rmse <- RMSE(data_validation$log_price, linear_fit_data_validation)
rpart_rmse <- RMSE(data_validation$log_price, rpart_fit_fit_data_validation)

# put RMSEs into a table
data.table(simple_linear_rmse, linear_rmse, rpart_rmse)
```

```
##    simple_linear_rmse linear_rmse rpart_rmse
## 1:          0.3797396    0.190462  0.2192006
```

d) Evaluate the final model on the test set. Why is it important to have this final set of observations set aside for evaluation? (Hint: think about what we used the validation set for.)

We can see from the table above that the linear_RMSE is the lowest and that is why I choose the linear model for further prediction.

In the following lines I calculate the RMSE on the test set:

```
linear_fit_data_validation<- predict.train(linear_fit, newdata = data_test)

final_performance_measure <- RMSE(data_test$log_price, linear_fit_data_validation)

final_performance_measure
```

```
## [1] 0.1885708
```

I think, if we get nearly the same results on the validation and on the test dataset (this is what happened in our case), we can be more sure that our model which was trained on the training set performs the same and it was not a coincidence (other words: random thing) that we got these results.

Test set is meant to be no intaraction for choosing any model just for check our decisions.

e) Do you think it makes more sense to use this method rather than the one used in class? What can be advantages or disadvantages of one or the other?

It is safer to use this approach compared that we used on the class. The disadvantage is that it is more time consuming and if the sample is small and we divide our data into 3 parts we will get so little amount of observation that we are not able to train a general model on the training sample.

2) Predicting developer salaries

Load the dateset and do some preparation on it:

```
data <- fread("C:/Users/Chronos/OneDrive - Central European University/R/machine_learning1/hw1/survey_r

data <- data[!is.na(Salary) & Salary > 0]
data <- data[complete.cases(data)]
data <- data[, Gender := ifelse(Gender == "Male", "Male",
                                ifelse(Gender == "Female", "Female", "Other"))]
large_countries <- data[, .N, by = "Country"][N > 60][["Country"]]
data <- data[, Country := ifelse(Country %in% large_countries, Country, "Other")]
```

a) Describe what the data cleansing steps mean:

I am going line by line:

- We read in the data with fread()
- We drop observations where Salary is missing or less then 0
- Convert the gender variable to Male/Female/Other variable. (if Male we assign Male, if Female we assign Female and the other goes to the Other category)

- we define a vector which consists the large countries by name. Large countries where we have greater than 60 observations in the dataset.

- Based on the vector in the previous point we select those countries which is not in the large country group and assign 'Other' for those observations. Observations with large_country keep their country name.

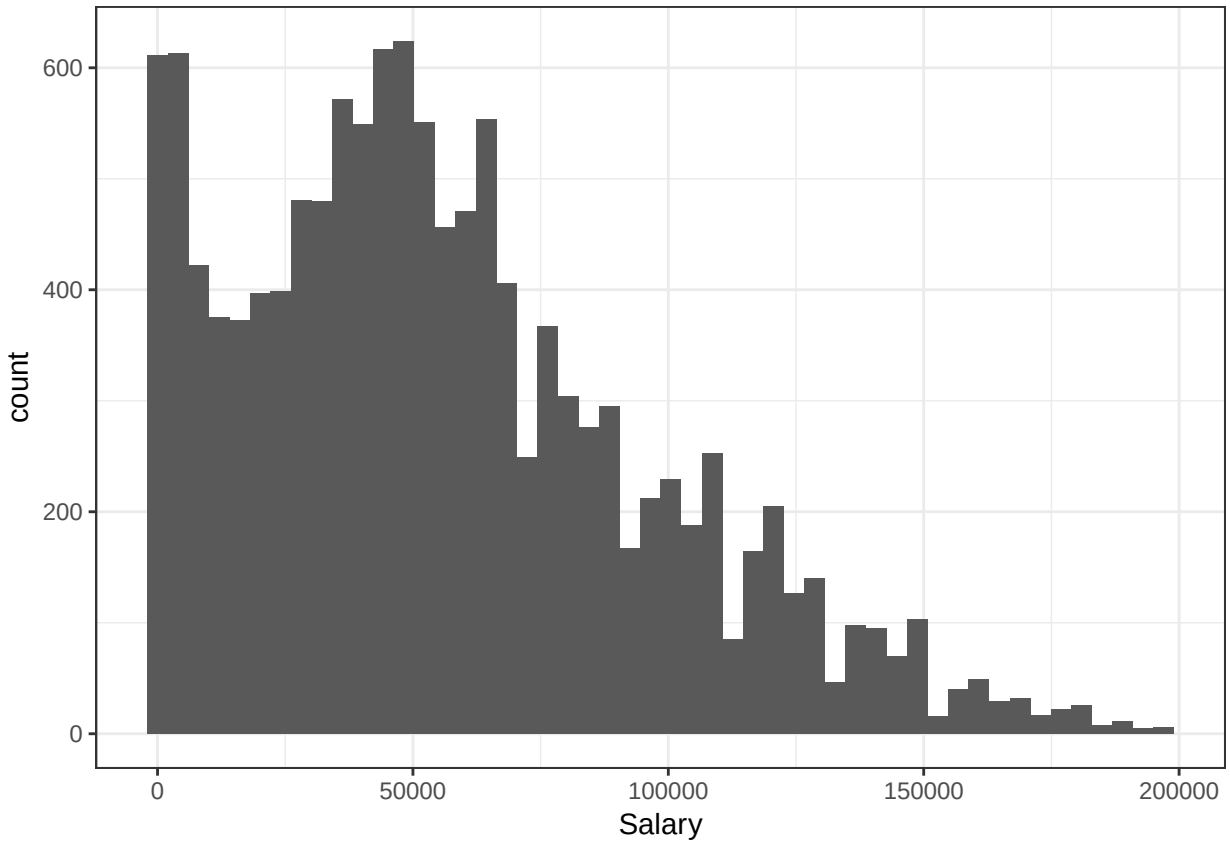
b) Using graphs, find at least two interesting features that can contribute to understanding developer salaries.

1st plot:

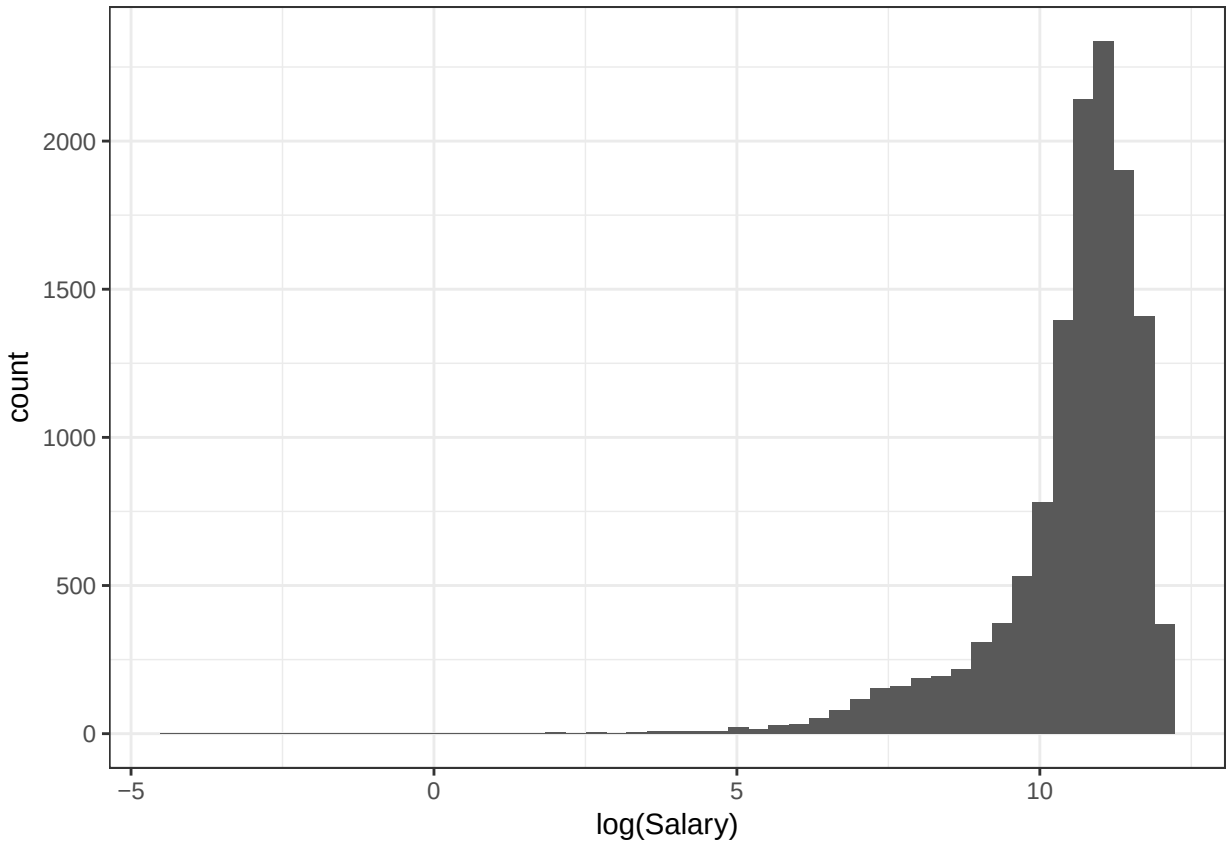
I examined the distribution of the salaries. In the first case we do not yield a normal distribution as we excepted regarding this variable, so I tried taking the log of it, which also not resulted a normally distributed variable, but we got closer to a normal distributed variable.

At this point I don't know yet, which variable I would like to use, but I keep both of them.

```
ggplot(data, aes(Salary)) + geom_histogram(bins = 50)
```




```
ggplot(data, aes(log(Salary))) + geom_histogram(bins = 50)
```

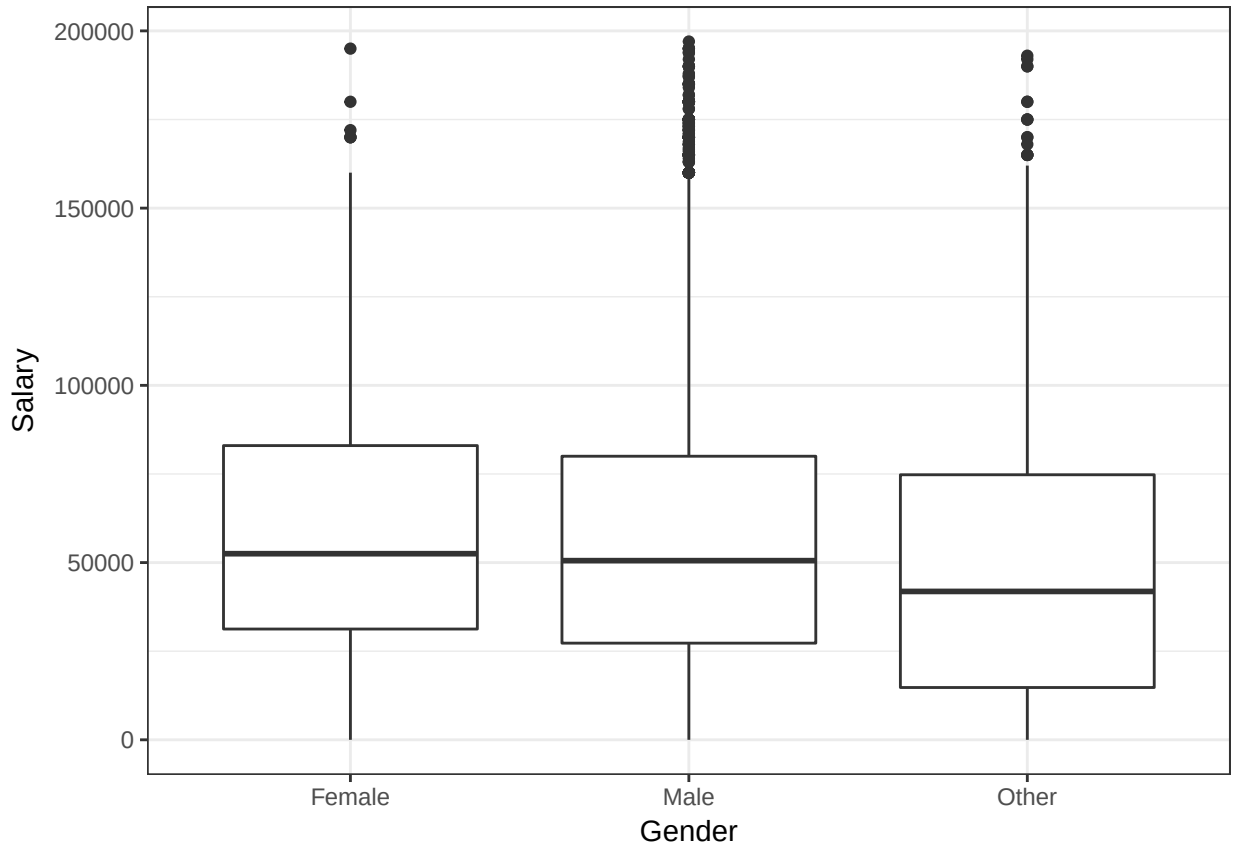



```
data[, log_salary := log(Salary)] #create log_salary
```

2st plot:

Who earns better? Male or Female or Other?

```
ggplot(data, aes(Gender,Salary)) + geom_boxplot()
```

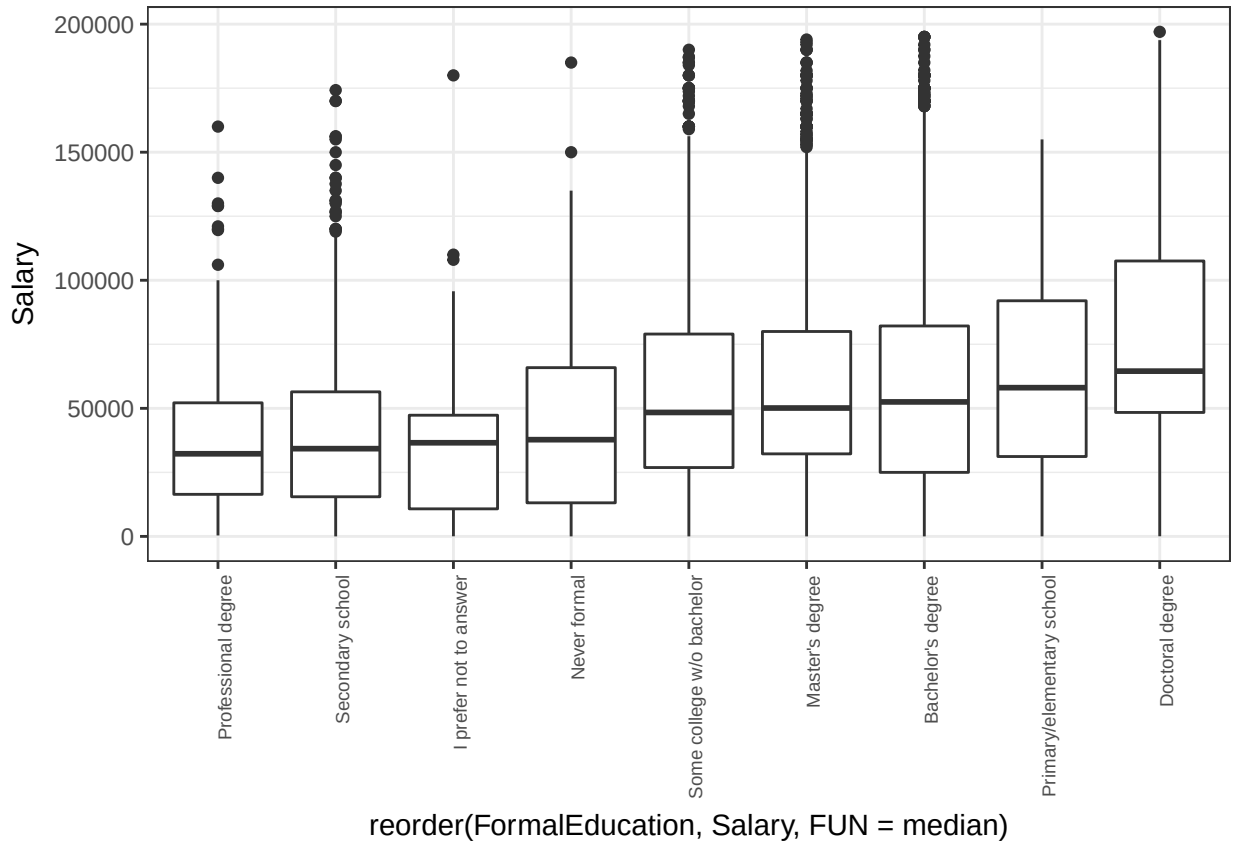


Female and Male people earns almost the same, however Other is a little bit less on average.

3rd plot:

After this, I tried to plot a box plot based on the Salary and the FromalEducation variables.

```
#unique(data$FormalEducation)

# Make some modification to nicer plots:
data[FormalEducation == "Some college/university study without earning a bachelor's degree", FormalEduca
data[FormalEducation == "I never completed any formal education", FormalEducation := "Never formal"]

#plot it:
ggplot(data, aes(reorder(FormalEducation, Salary, FUN=median),Salary)) + geom_boxplot() +  theme(axis.te
```



We can see that people with Doctoral Degrees earn the best. A surprising thing can be that on the second place of average earnings there are the people with only Primary/secondary school of education. Note: Most of the cases the averages are close to each other and the IQRs are wide and overlapping so based on this it is hard to conclude a significant result, but it tells us something.

4th plot:

Then I plotted the experience variable (YearsProgram) vs. Salary on a box plot too.

```
ggplot(data, aes(reorder(YearsProgram, Salary, FUN=median), Salary)) + geom_boxplot() +  theme(axis.tex
```

On average we can see the following: The more the experience, the higher the salary. Which is not surprising.

c) Create a training and a test set assigning 70% to the training set and 30% as the test set.

```
cut <- createDataPartition(y = data$Salary,
                                     times = 1,
                                     p = 0.7,
                                     list = FALSE)

data_train <- data[cut, ]
data_test <- data[-cut, ]

### check spliting:
length(data$Salary) == (length(data_train$Salary) + length(data_test$Salary))
```

```
## [1] TRUE
```

d) Using caret train at least two predictive models to predict the logarithm of Salary (they can be of the same family but with different hyperparameters or they can be of different families like we used lm and rpart in the first exercise). Make sure NOT to include Salary as a predictor variable. Also, just before calling train, remember to use set.seed. Then: choose the best model based on cross-validation estimation on the training set and evaluate its performance on the test set

A linear model:

```r
set.seed(1234)

fit_control <- trainControl(method = "cv", number = 10)

linear_model <- train(log_salary ~ as.factor(Country) + Gender + FormalEducation + YearsProgram, data =

linear_model
```

```
## Linear Regression
##
## 9020 samples
##    4 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 8118, 8118, 8118, 8119, 8117, 8117, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.9817047  0.4776789  0.5508753
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

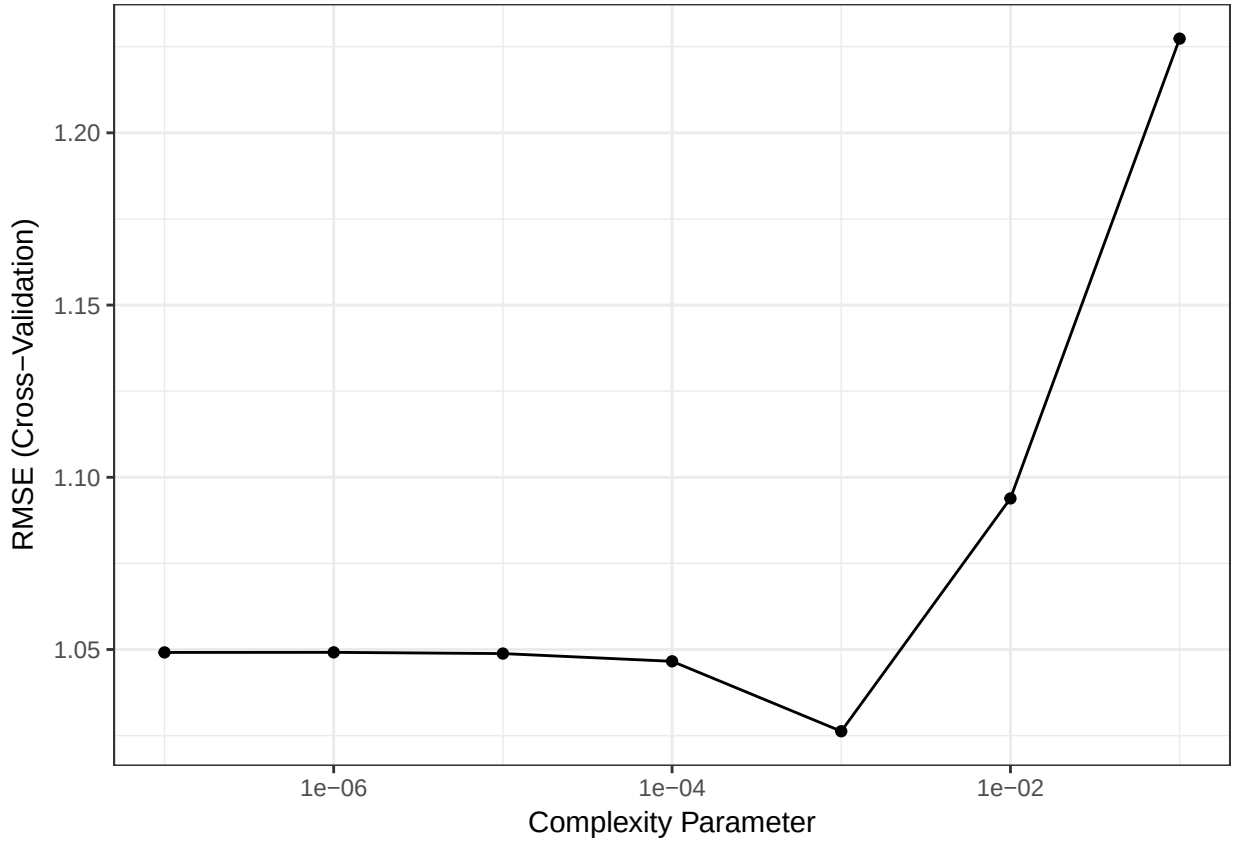
A tree model:

```r
tune_grid <- data.frame("cp" = c(0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001))

rpart_model <- train(log_salary ~ . - Salary,
                     data = data_train,
                     method = "rpart",
                     trControl = fit_control,
                     tuneGrid = tune_grid)

rpart_model
```

```
## CART
##
## 9020 samples
##    8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 8118, 8118, 8119, 8117, 8117, 8117, ...
## Resampling results across tuning parameters:
##
##   cp     RMSE      Rsquared   MAE
##   1e-07  1.049154  0.4145084  0.5982461
##   1e-06  1.049173  0.4144928  0.5982755
##   1e-05  1.048821  0.4148111  0.5975747
##   1e-04  1.046573  0.4161700  0.5926445
##   1e-03  1.026269  0.4315649  0.5839614
##   1e-02  1.093861  0.3508670  0.6532347
##   1e-01  1.227352  0.1821817  0.7863944
##
## RMSE was used to select the optimal model using the smallest value.
```

```
## The final value used for the model was cp = 0.001.
```

```
ggplot(rpart_model) + scale_x_log10()
```



The best model is with CP = 0.001, but gave worse results then the linear model so I try using that one for the test set.

```
data_test$predicted_linear <- predict.train(linear_model, newdata = data_test)

# investigate RMSE of the test set:
linear_fit_test_data_rmse <- RMSE(data_test$predicted_linear, data_test$log_salary)
linear_fit_test_data_rmse
```

```
## [1] 0.9279307
```

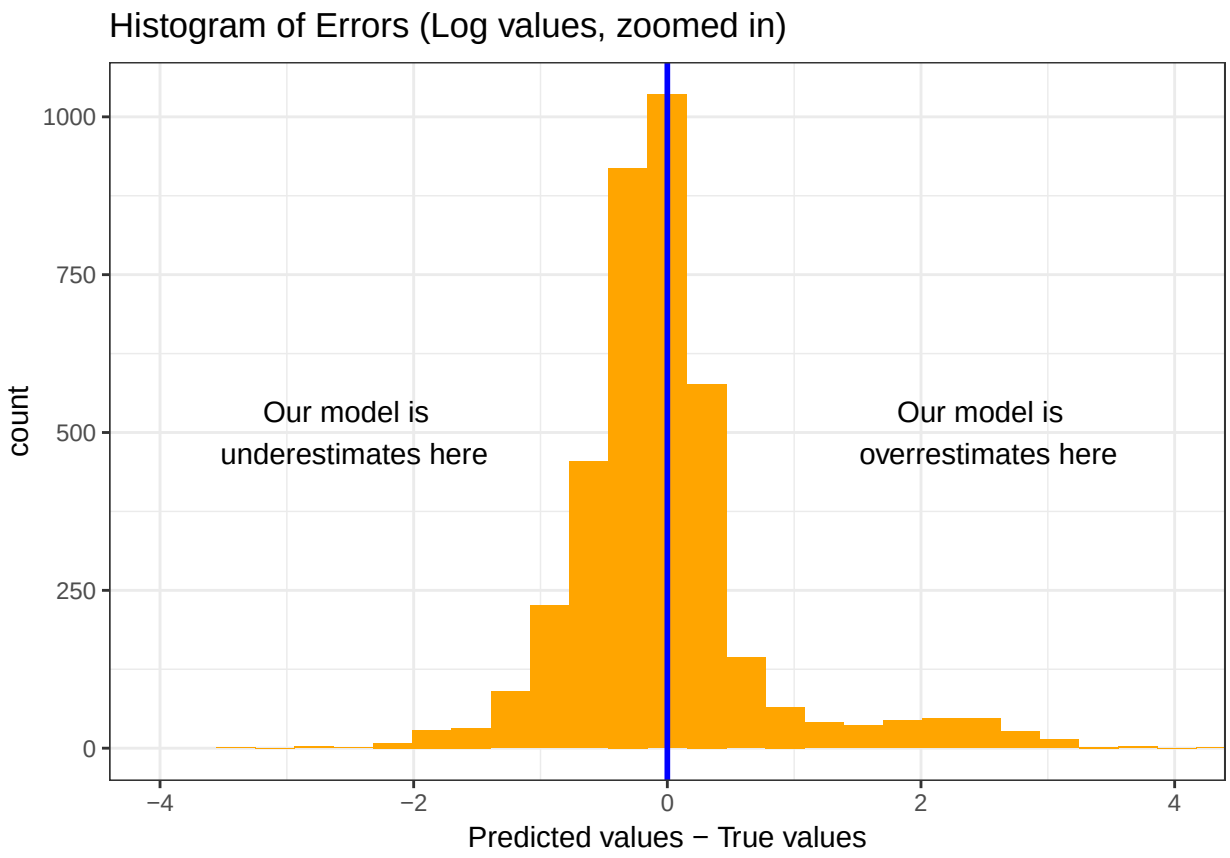
```
data.table("data_train_RMSE" = c(0.9818)," " = c("VS."), "data_test_RMSE" = linear_fit_test_data_rmse)
```

```
##    data_train_RMSE     data_test_RMSE
## 1:          0.9818 VS.      0.9279307
```

Surprisingly our model performed better on the test data

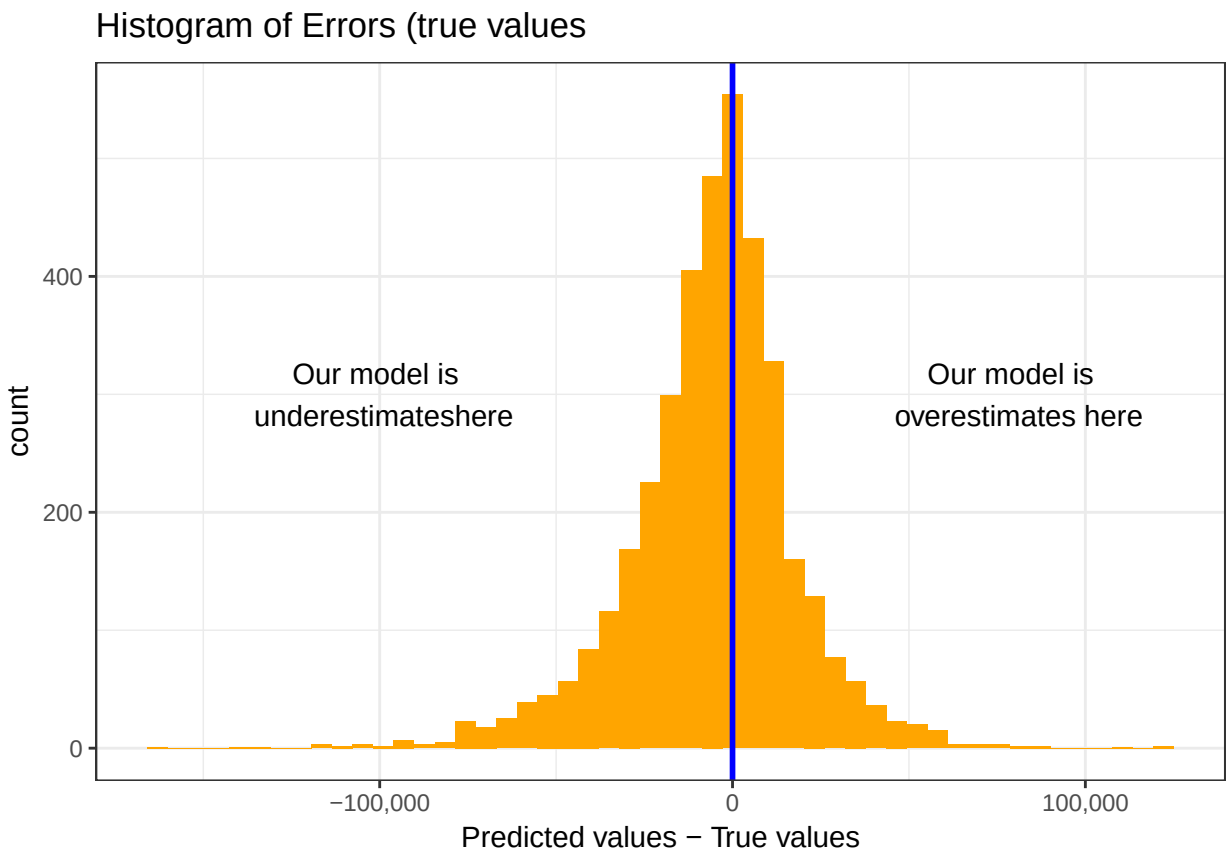
Then plotting the predicted results vs. the real results in LOG:

```
ggplot(data_test, aes(predicted_linear - log_salary)) + geom_histogram(bins = 50,
    fill = "orange") + coord_cartesian(xlim = c(-4, 4)) + geom_vline(xintercept = 0,
    color = "blue", size = 1) + annotate("text", x = 2.5, y = 500, label = "Our model is \n overrestima
    annotate("text", x = -2.5, y = 500, label = "Our model is \n underestimates here") +
    ggtitle("Histogram of Errors (Log values, zoomed in)") + labs(x = "Predicted values - True values")
```



Then plotting the predicted results vs. the real results in real values:

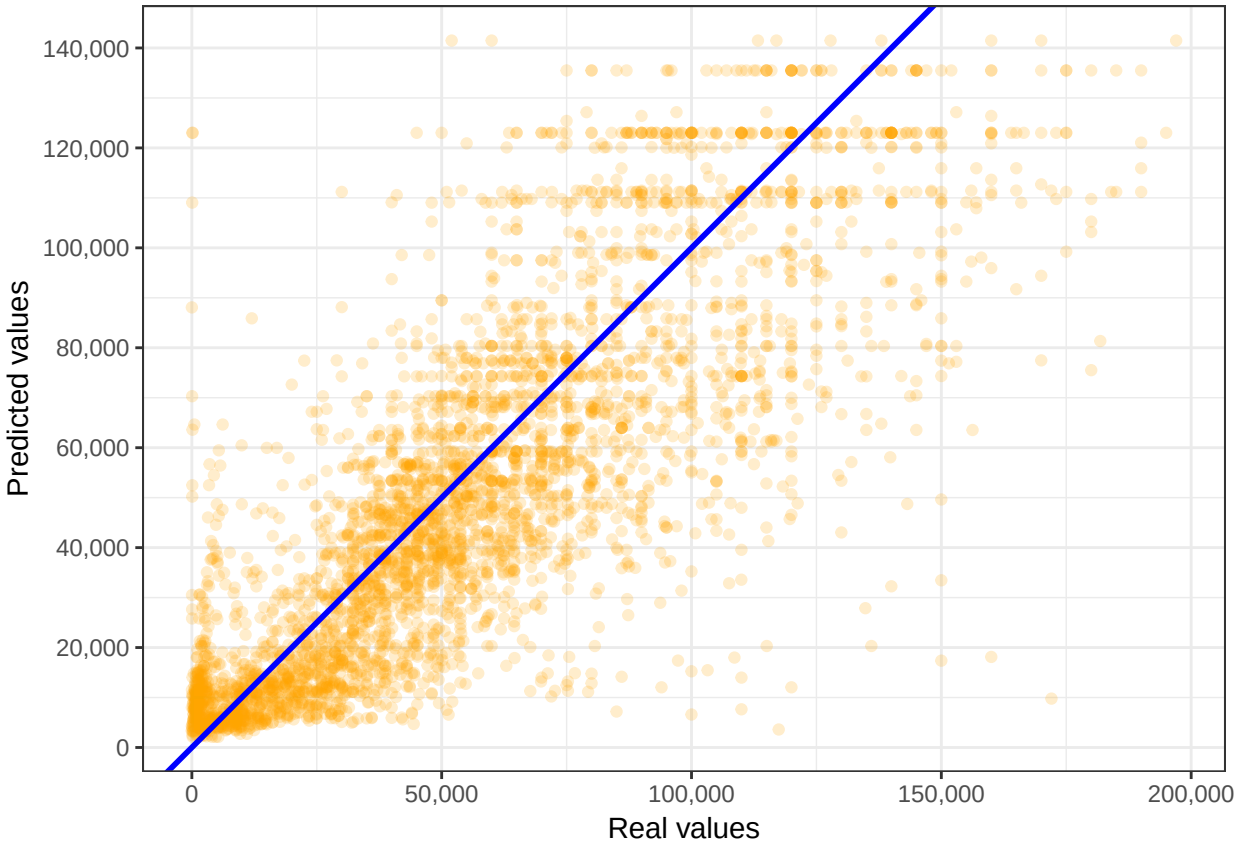
```
ggplot(data_test, aes(exp(predicted_linear) - exp(log_salary))) + geom_histogram(bins = 50,
    fill = "orange") + geom_vline(xintercept = 0, color = "blue", size = 1) +
    scale_x_continuous(labels = comma) + annotate("text", x = 80000, y = 300,
    label = "Our model is \n overestimates here") + annotate("text", x = -1e+05,
    y = 300, label = "Our model is \n underestimateshere") + ggtitle("Histogram of Errors (true values"
    labs(x = "Predicted values - True values")
```



We can see that our errors giving us a nice normally distributed graph which is good, indicates homoscedasticity.

Last graph, true values vs. predicted values:

```
number_ticks <- function(n) {
    function(limits) pretty(limits, n)
}

ggplot(data_test, aes(y = exp(predicted_linear), x = exp(log_salary))) + geom_point(color = "orange",
    alpha = 0.2) + scale_x_continuous(labels = comma, breaks = number_ticks(5)) +
    scale_y_continuous(labels = comma, breaks = number_ticks(6)) + geom_abline(intercept = 0,
    slope = 1, color = "blue", size = 1) + coord_fixed() + labs(x = "Real values",
    y = "Predicted values")
```



It shows us that there are observations above the 1slope line and under the line. Sometimes even big errors (likely, concerning lower values), but on average our model is indicating that it is working normally.

3 LOOCV on the Titanic data

Load the data:

```
library(titanic)
library(data.table)

data_train <- data.table(titanic_train)
# recode Survived to factor - needed for binary prediction
data_train[, Survived := factor(ifelse(Survived == 1, "survived", "died"))]
```

a) Name a disadvantage of this method compared to using a moderate value (say, 10) for k?

- LOOCV is a special case of k-fold CV The disadvantage is that LOOCV has a very high resource requirements compared to 10 fold method.
- From the ISLR book: "The most obvious advantage is computational. LOOCV requires fitting the statistical learning method n times. This has the potential to be computationally expensive. But cross-validation is a very general approach that can be applied to almost any statistical learning method."

b) Why do you think it can still make sense to compute this measure? In what way can this measure be closer to the "real" performance of the model?

- k-fold CV is often gives more accurate estimates of the test error rate than does LOOCV
- LOOCV will give approximately unbiased estimates of the test error. Therefore, from the perspective of bias reduction, it is clear that LOOCV is to be preferred to k-fold CV. However, we know that bias is not the only source for concern in an estimating procedure; we must also consider the procedure's variance. It turns out that LOOCV has higher variance than does k-fold CV with k < n.
- To summarize it: there is a bias-variance trade-off associated with the choice of k in k-fold cross-validation. Typically, given these considerations, one performs k-fold cross-validation using k = 5 or k = 10, as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance.

c) You can implement LOOCV with caret by setting an option in trainControl: method = "loocv". and use a simple logit model glm for prediction.

LOOCV logit model:

```
train_control <- trainControl(method = "loocv", classProbs = TRUE)

logit_loocv <- train(Survived ~ Fare + Sex,
                     data = data_train,
                     method = "glm",
                     trControl = train_control)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
## trainInfo, : There were missing values in resampled performance measures.
```

10 fold CV model:

```
train_control <- trainControl(method = "cv", number = 10, classProbs = TRUE)

logit_cv <- train(Survived ~ Fare + Sex,
                     data = data_train,
                     method = "glm",
                     trControl = train_control)
```

d) Compare the accuracy of the model estimated by the two resampling methods via summary(fitted_model$resample). Accuracy is the share of cases predicted correctly.

```
# LOOCV model:
summary(logit_loocv$resample)
```

```
##     Accuracy          Kappa       Resample
##  Min.   :0.0000   Min.   :0     Length:891
##  1st Qu.:1.0000   1st Qu.:0     Class :character
##  Median :1.0000   Median :0     Mode  :character
##  Mean   :0.7823   Mean   :0
##  3rd Qu.:1.0000   3rd Qu.:0
##  Max.   :1.0000   Max.   :0
##                   NA's   :697
```

```
# 10 fold CV model:
summary(logit_cv$resample)
```

```
##     Accuracy          Kappa          Resample
##  Min.   :0.7386   Min.   :0.4303   Length:10
##  1st Qu.:0.7444   1st Qu.:0.4624   Class :character
##  Median :0.7598   Median :0.4887   Mode  :character
```

```
##  Mean   :0.7811   Mean   :0.5322
##  3rd Qu.:0.8180   3rd Qu.:0.6096
##  Max.   :0.8652   Max.   :0.7112
```

Means are almost the same

Quantiles and median are so extreme (0/1) in the case of LOOCV, because we always left out only one observation when we train the model and therefore the accuracy there is either 100% or 0% (in the case of LOOCV). But the averages of all the models accuracy are almost the same as we can see from the above table.

In the last lines I am comparing the predicted results of the two models:

```
#predict values: survived/died:
data_train$Predicted_loocv <- predict.train(logit_loocv, newdata = data_train)
data_train$Predicted_cv <- predict.train(logit_cv, newdata = data_train)

#create table:
table(data_train$Predicted_loocv, data_train$Survived)
```

```
##
##            died survived
##   died      462      107
##   survived   87      235
```

```
table(data_train$Predicted_cv, data_train$Survived)
```

```
##
##            died survived
##   died      462      107
##   survived   87      235
```

As we see the two models gave the same results.