Lab 2

Assignment 1

1. Fit linear regression to the training data

1. Assume that Fat can be modeled as a linear regression in which absorbance characteristics (Channels) are used as features. Report the underlying probabilistic model, fit the linear regression to the training data and estimate the training and test errors. Comment on the quality of fit and prediction and therefore on the quality of model.

The probabilistic model for linear regression is:

$$y = \theta_0 + \theta_i x_i + \ldots + \theta_p x_p + \varepsilon \text{ , where } \varepsilon \sim N(0, \sigma^2), \text{which we can write as } y \sim N(\boldsymbol{\theta^T x}, \sigma^2)$$

For our problem, assuming fat can be modelled as a linear regression, the underlying probabilistic model is:

$$fat = \theta_0 + \sum_{i=1}^{100} \theta_i \times channel_i + \varepsilon, \qquad where\ \varepsilon \sim N(0, \sigma^2)$$

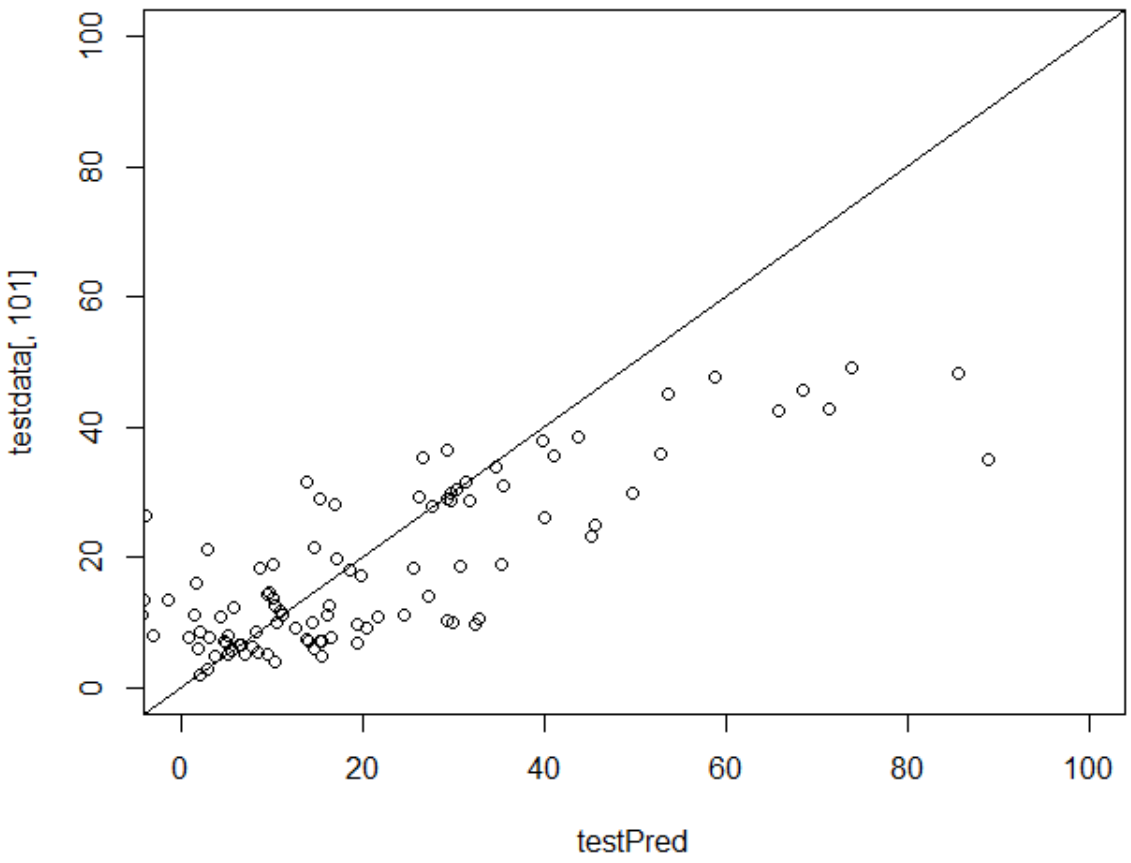
Which we also can write as:

$$fat \sim N(\theta_0 + \sum_{i=1}^{100} \theta_i \times channel_i, \sigma^2)$$

```
> MSETrain
[1] 0.005709117
> MSETest
[1] 722.4294
>
```

If we calculate MSE for our training data and test data, we get MSEtrain = 0.0057 and MSEtest = 722.4294. We can see that our test MSE is very large compared to the train MSE (which is very close to 0), and both indicate that the model is too complex and thus overfitted.

We can also plot the predictions on the test set and the true fat-levels from the test set and we can see that the model performs badly when making predictions.



2. LASSO cost function to be optimized

2. Assume now that Fat can be modeled as a LASSO regression in which all Channels are used as features. Report the cost function that should be optimized in this scenario.

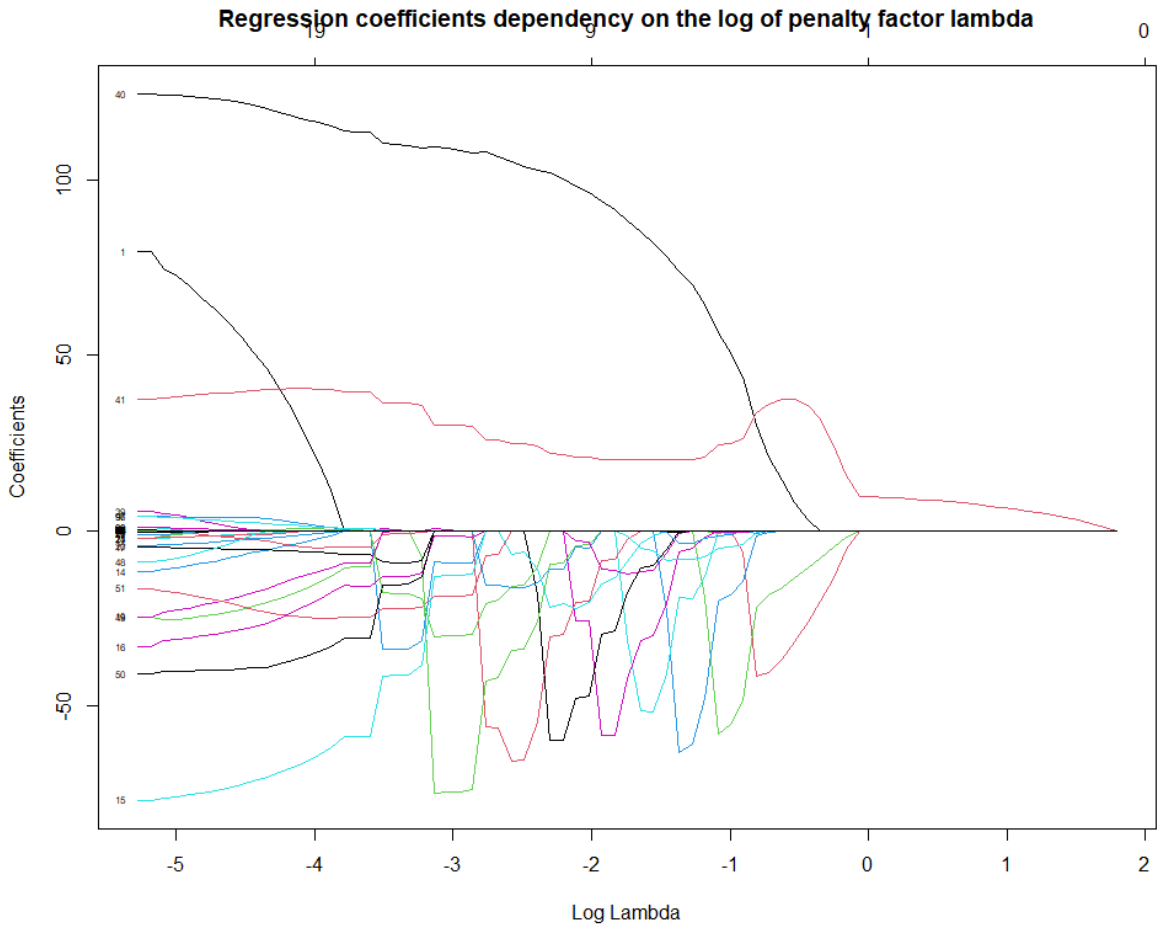
$$\theta hat^{lasso} = argmin(\frac{1}{n}\sum_{i=1}^{n}(fat_i - (\theta_0 + \theta_1 \times channel_{1,i} + \cdots + \theta_p \times channel_{p,i})^2 + \lambda \sum_{j=1}^{p} |\theta_j|$$

Where n is number of observations (nrow(traindata) if we use our training data), and p is the number of features (100 in our case).

3. Fit the LASSO regression model to the training data

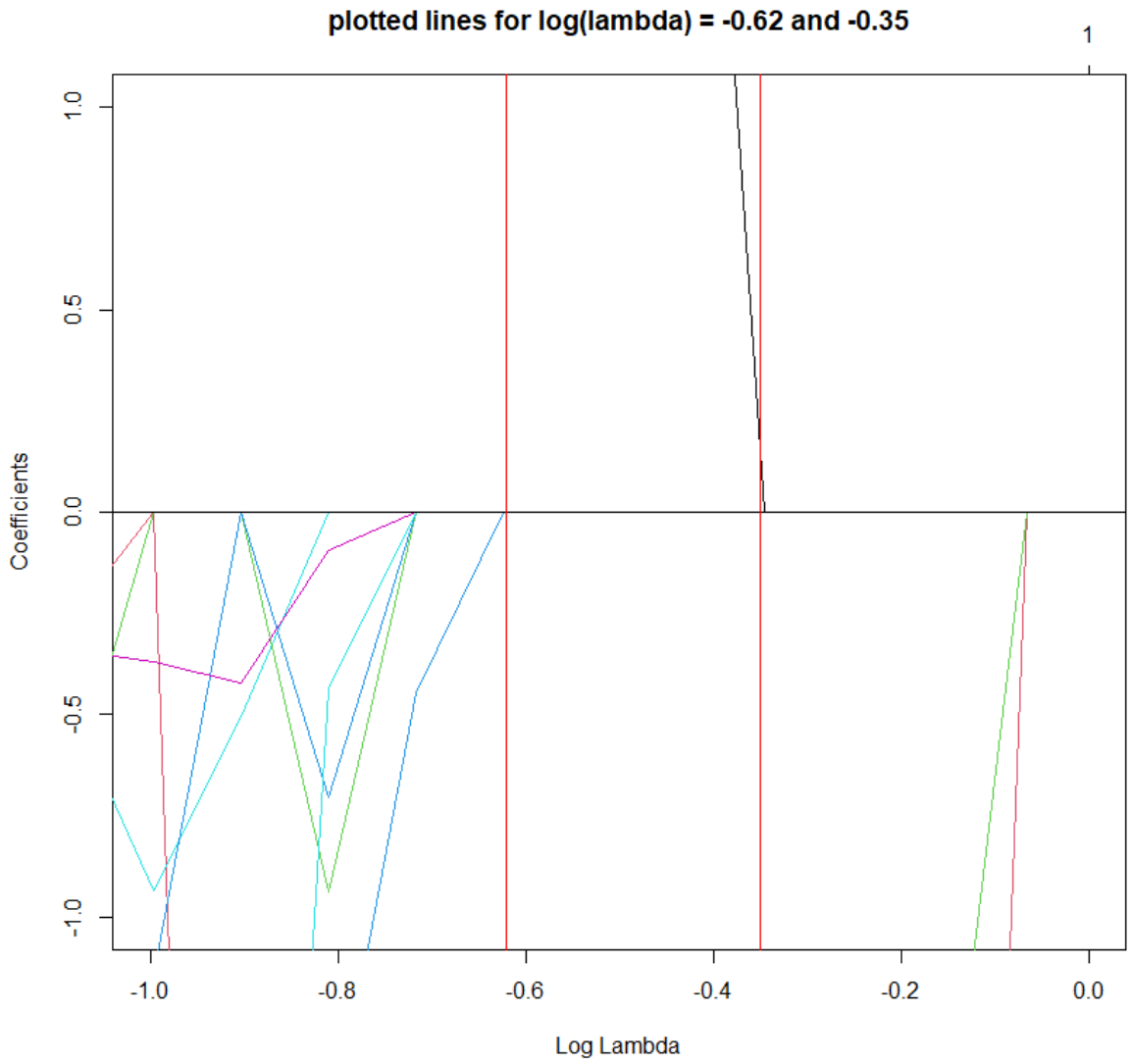
3. Fit the LASSO regression model to the training data. Present a plot illustrating how the regression coefficients depend on the log of penalty factor ($\log \lambda$) and interpret this plot. What value of the penalty factor can be chosen if we want to select a model with only three features?

After training our LASSO regression model using glmnet, we can plot how the regression coefficients depend on the log of lambda which can be seen below:



On the top of the y-axis, we can see how many features in our model are non-zero. We can see that as we increase our penalty factor lambda, more coefficients for our features become 0. Lambda = 0, i.e. no regularization, would mean that all features that were originally in the model will remain, and for lambda=e^2 (i.e. log of lambda = 2) all coefficients are 0, thus we have no features left in the model and only the intercept term.

We know that the more regularization we apply to a model (i.e. larger lambda) and thus more penalty for large coefficients, the less flexible and less complex the model becomes since it gets punished for having many and/or large theta-coefficients, and this is in line with what we see in the graph, larger lambda leads to fewer non-zero features.



If we zoom in and plot red lines for log(lambda)= -0.62 and -0.35, we get above figure.

We can then see that for log(lambda) ≈ -0.62 which corresponds to lambda ≈ 0.5379 we go from having 4 to 3 non-zero features, and for log(lambda) ≈ -0.35 which corresponds to lambda ≈ 0.7047 we go from having 3 to 2 non-zero features. However, since between two lambda values the number of model variables can NOT be assumed to remain monotonic. The number of non-zero variables can increase and then decrease again as we increase lambda, due to how the optimization works.

Instead, lets not try to find an interval, but lets look at the degrees of freedom of the model together with corresponding lambdas.

```
> lambda_df_deviance
          lambda df   deviance
 [1,] 6.018065119  0 0.00000000
 [2,] 5.483436801  1 0.03913045
 [3,] 4.996303388  1 0.07161724
 [4,] 4.552445566  1 0.09858834
 [5,] 4.148018849  1 0.12098023
 [6,] 3.779520287  1 0.13957036
 [7,] 3.443758121  1 0.15500422
 [8,] 3.137824141  1 0.16781768
 [9,] 2.859068493  1 0.17845563
[10,] 2.605076728  1 0.18728745
[11,] 2.373648891  1 0.19461979
[12,] 2.162780466  1 0.20070722
[13,] 1.970645010  1 0.20576111
[14,] 1.795578339  1 0.20995694
[15,] 1.636064108  1 0.21344039
[16,] 1.490720682  1 0.21633241
[17,] 1.358289165  1 0.21873342
[18,] 1.237622499  1 0.22072678
[19,] 1.127675527  1 0.22238170
[20,] 1.027495941  1 0.22375565
[21,] 0.936216034  1 0.22489632
[22,] 0.853045182  3 0.29236134
[23,] 0.777262999  3 0.38343618
[24,] 0.708213096  3 0.45907582
[25,] 0.645297397  4 0.52215118
[26,] 0.587970955  4 0.57473776
[27,] 0.535737235  5 0.61858822
[28,] 0.488143816  5 0.65529196
[29,] 0.444778465  9 0.69562806
[30,] 0.405265572  8 0.73159735
[31,] 0.369262895  8 0.76063691
[32,] 0.336458597 10 0.78447889
[33,] 0.306568543  8 0.80592817
[34,] 0.279333839  7 0.82369588
[35,] 0.254518590 11 0.83764891
[36,] 0.231907860 10 0.85161527
[37,] 0.211305805  8 0.86356832
[38,] 0.192533980  9 0.87192896
[39,] 0.175429792 10 0.88007156
[40,] 0.159845092  7 0.88760782
[41,] 0.145644895  8 0.89270615
[42,] 0.132706204  9 0.89843326
```

And, we can see, that for 3 of the lambda values used we have models with 3 non-zero features, namely the lambda-values 0.7082, 0.7773, and 0.8530. Thus I choose lambda=0.8530, since this lambda regularizes the most, and we want to keep our theta-values small if possible.

The following MSE's were obtained from the LASSO Model:

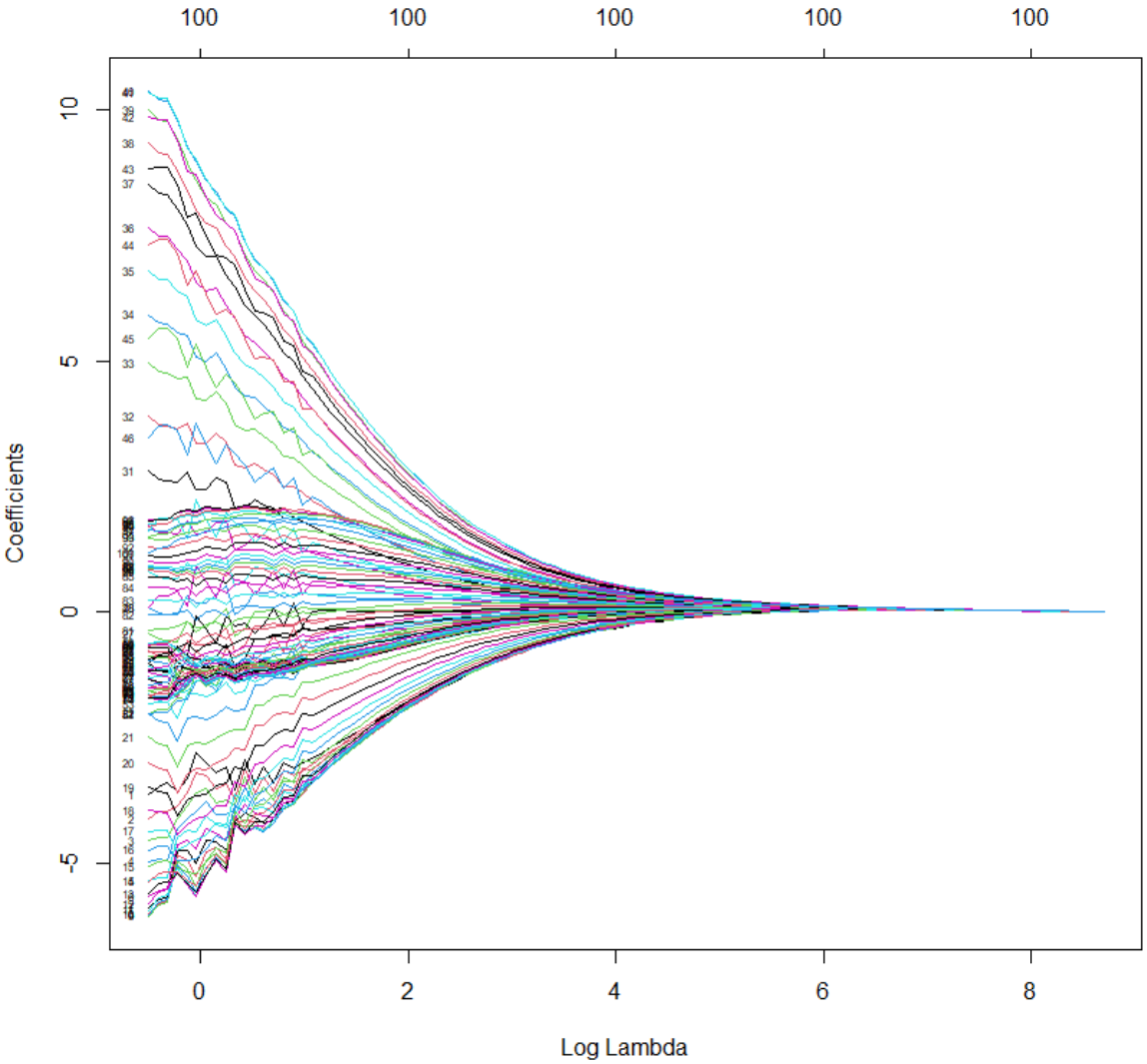
```
> MSETrainLasso
[1] 52.44821
> MSETestLasso
[1] 52.1125
```

Since the MSE for training and testing set is so similar, this could be an indication that our model is underfitted. Meaning, although the MSE for test set is much better than in part 1 where we used regular linear regression and had a MSE of 722.4, we can probably find a more suitable model.

4. Fit a Ridge regression model to the training data

4. Repeat step 3 but fit Ridge instead of the LASSO regression and compare the plots from steps 3 and 4. Conclusions?

After training our Ridge regression model using glmnet, we can plot how the regression coefficients depend on the log of lambda which can be seen in below figure:



With same reasoning as for lasso regression, with ridge regression, as lambda increases the coefficients decrease. However one important difference is that here no coefficients become 0, only very close to 0 for larger lambdas, compared to the LASSO regression where some coefficients

actually reach 0, i.e. the LASSO regression automatically does parameter selection for us. Thus in the ridge model all 100 original features are non-zero for any value of lambda.

The following MSE's were obtained from the Ridge Model:

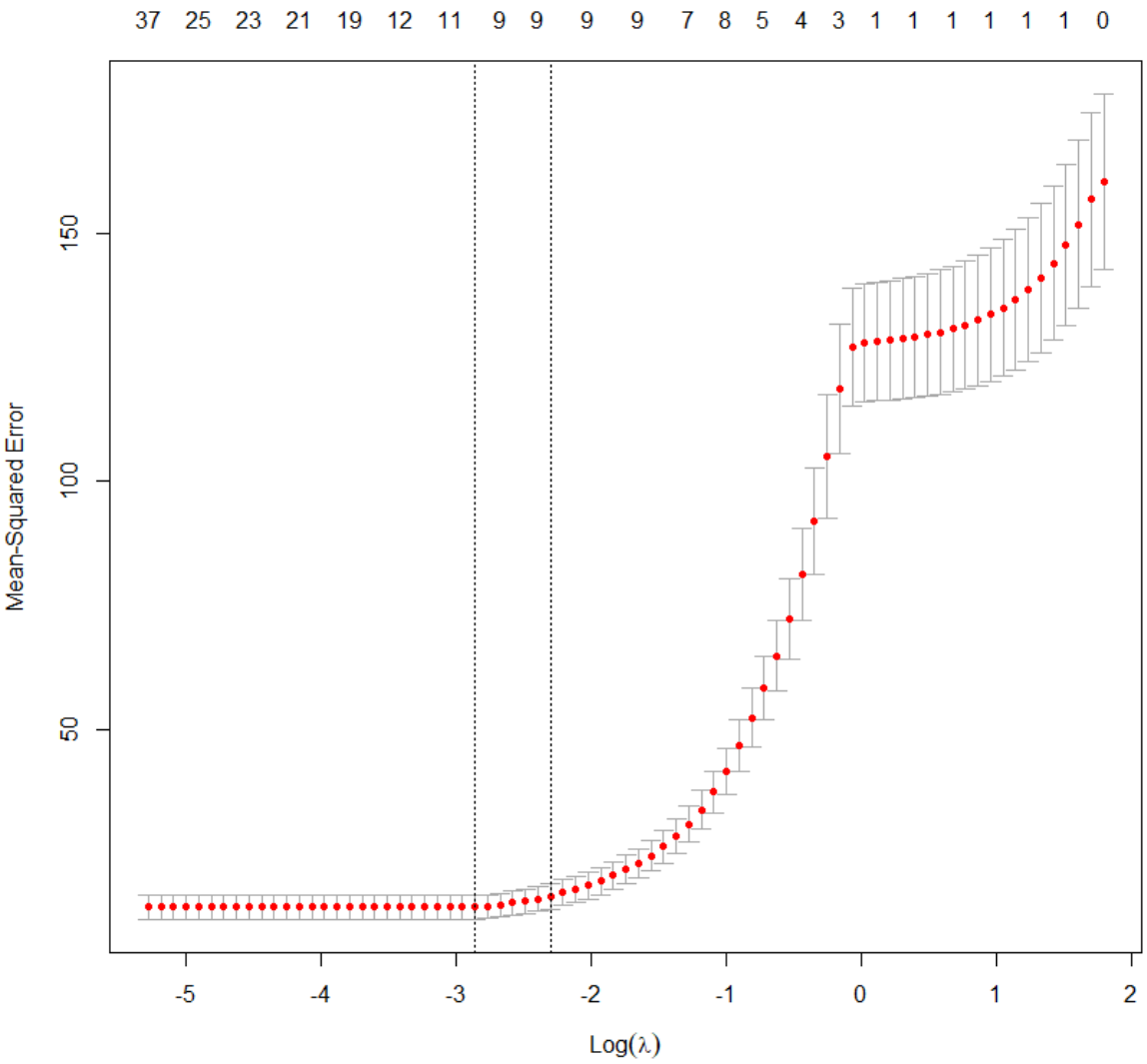
```
> MSETrainRidge
[1] 102.7454
> MSETestRidge
[1] 100.4839
```

As for the LASSO model, since the MSE for training and testing set is so similar, this could be an indication that our model is underfitted, so we can probably find a more suitable model. The MSE for the test set is also higher than for the LASSO model, so the Ridge model performs worse.

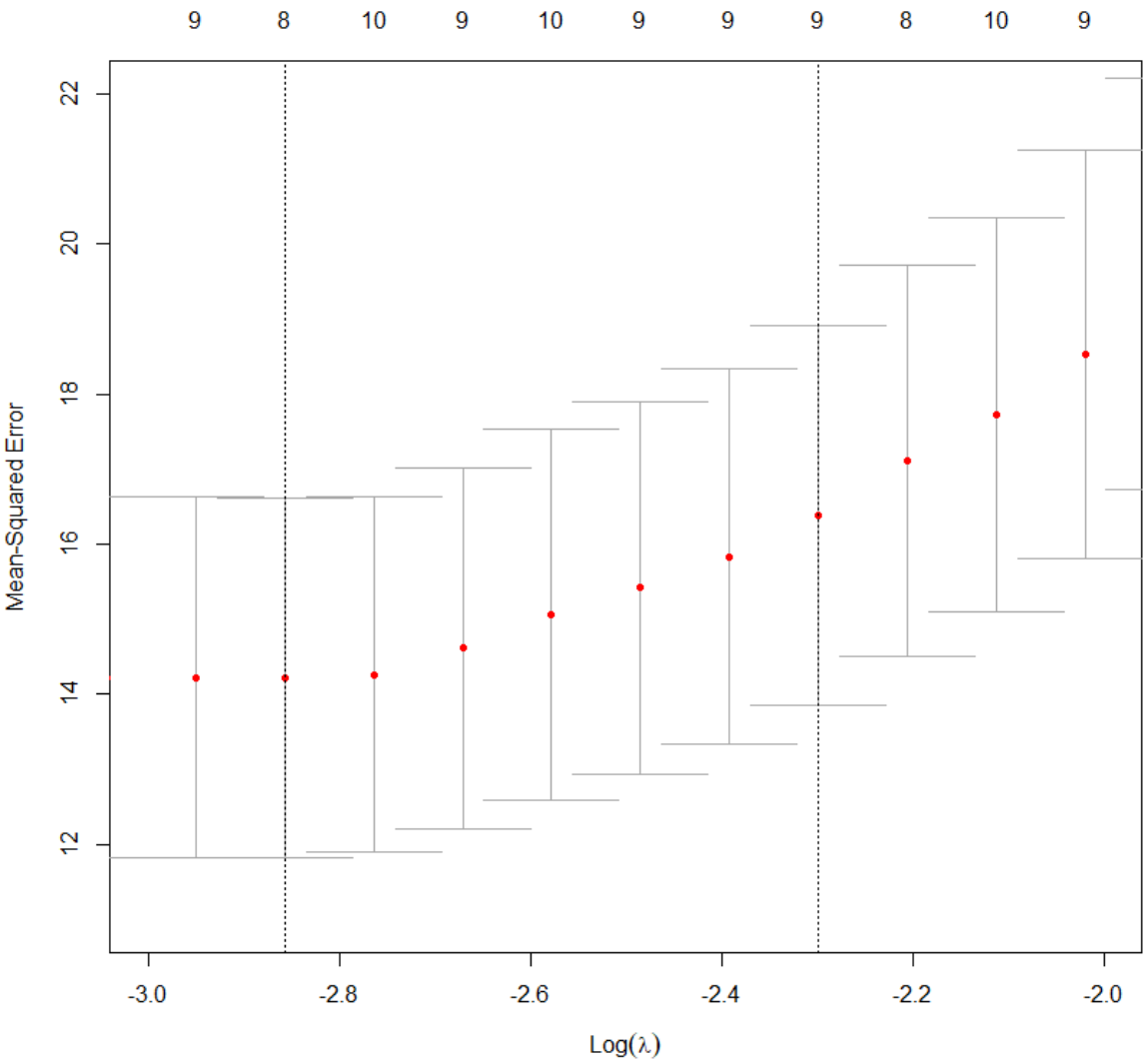
5. 10-fold Cross-Validation to compute the optimal LASSO model

5. Use cross-validation with default number of folds to compute the optimal LASSO model. Present a plot showing the dependence of the CV score on $\log \lambda$ and comment how the CV score changes with $\log \lambda$. Report the optimal λ and how many variables were chosen in this model. Does the information displayed in the plot suggests that the optimal λ value results in a statistically significantly better prediction than $\log \lambda = -4$? Finally, create a scatter plot of the original test versus predicted test values for the model corresponding to optimal lambda and comment whether the model predictions are good.

After training our optimal LASSO regression model using cv.glmnet with 10 folds (the default), we can plot how the average cross-validation error depends on the log of lambda, see below:



As lambda increases, more regularization is applied to the model,and it becomes less flexible and less complex. We can see that as lambda increases, the larger the CV error becomes because the model becomes less complex and eventually could become underfitted. As lambda decreases, the CV error becomes smaller, but this also leads to a more complex model, and eventually the model could overfit. So when selecting an optimal lambda, there is generally a trade-off between a complex model with a small CV error or a less complex model with a slightly higher CV error.



The plotted line on the left corresponds to lambda_min, and is the lambda which minimizes the CV error, and the plotted line to the right corresponds to lambda_1se, which is the largest lambda value within 1 standard error of lambda_min.

```
> lasso_optimal_model$lambda.min
[1] 0.05744535
```

```
> lasso_optimal_model$cvm[51]
[1] 14.21175
```

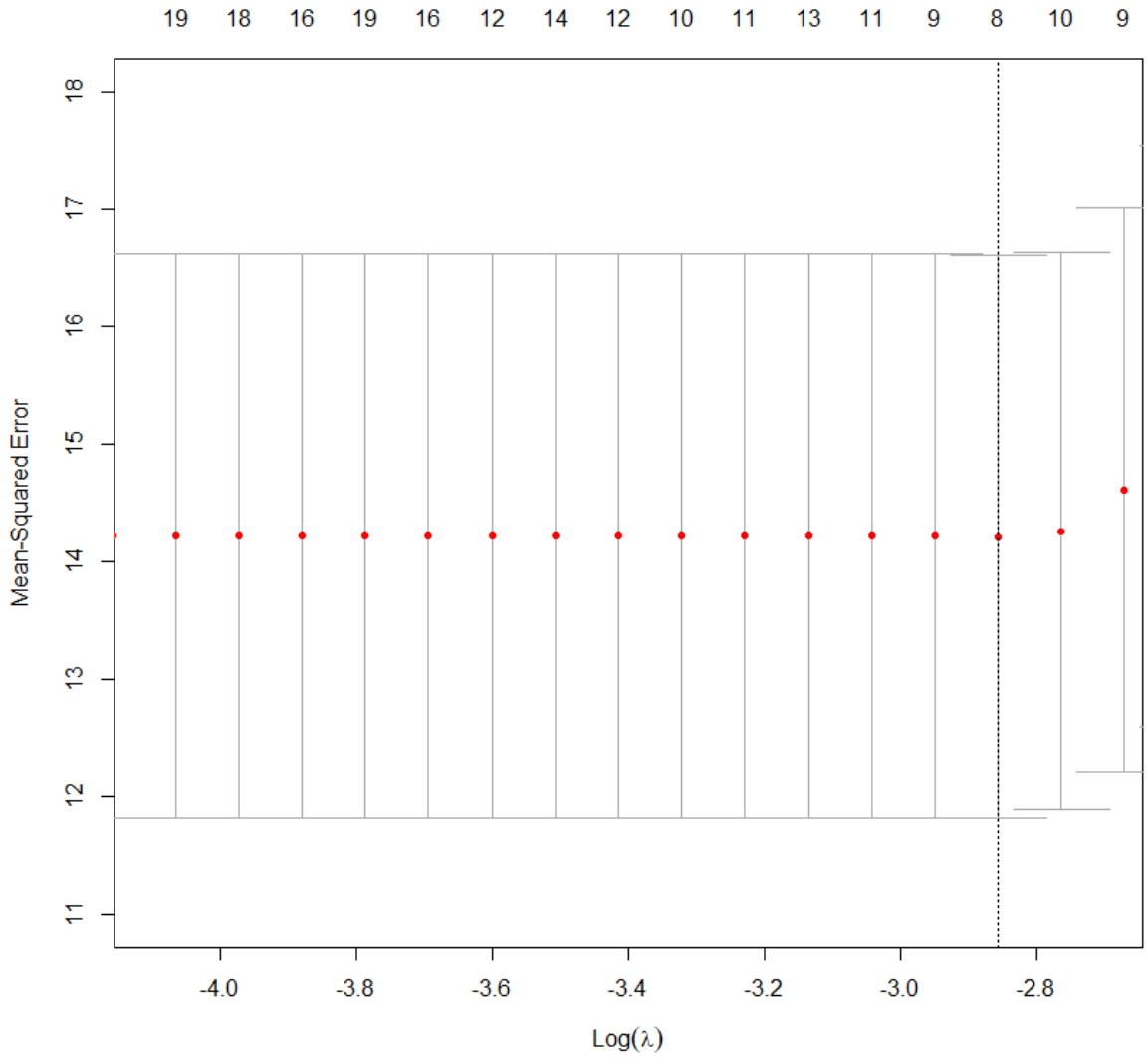
```
> lasso_optimal_model$nzero[51]
s50
  8
```

In this case lambda_min = 0.05744535, with mean cross-validation error 14.21175, and the model has has 8 non-zero variables. Compared to lambda_1se = 0.1003874, with mean cross-validation error 16.39023, and the model has 9 non-zero variables. In this case, there seems to be nothing that indicates that the model corresponding to lambda_1se is less complex than the model corresponding to lambda_min, since it has 9 non-zero features compared to 8, although it is possible that these 9 features could have significantly smaller coefficients (if we assume that model

complexity can be measured by looking at number of variables, then the model with 9 non-zero features is more complex). At the same time, the model corresponding to lambda_1se has a slightly higher CV error as expected. So in this case, there is nothing which is telling us not to choose lambda_min, thus we come to the conclusions that lambda = 0.05744535 results in the optimal LASSO regression model.

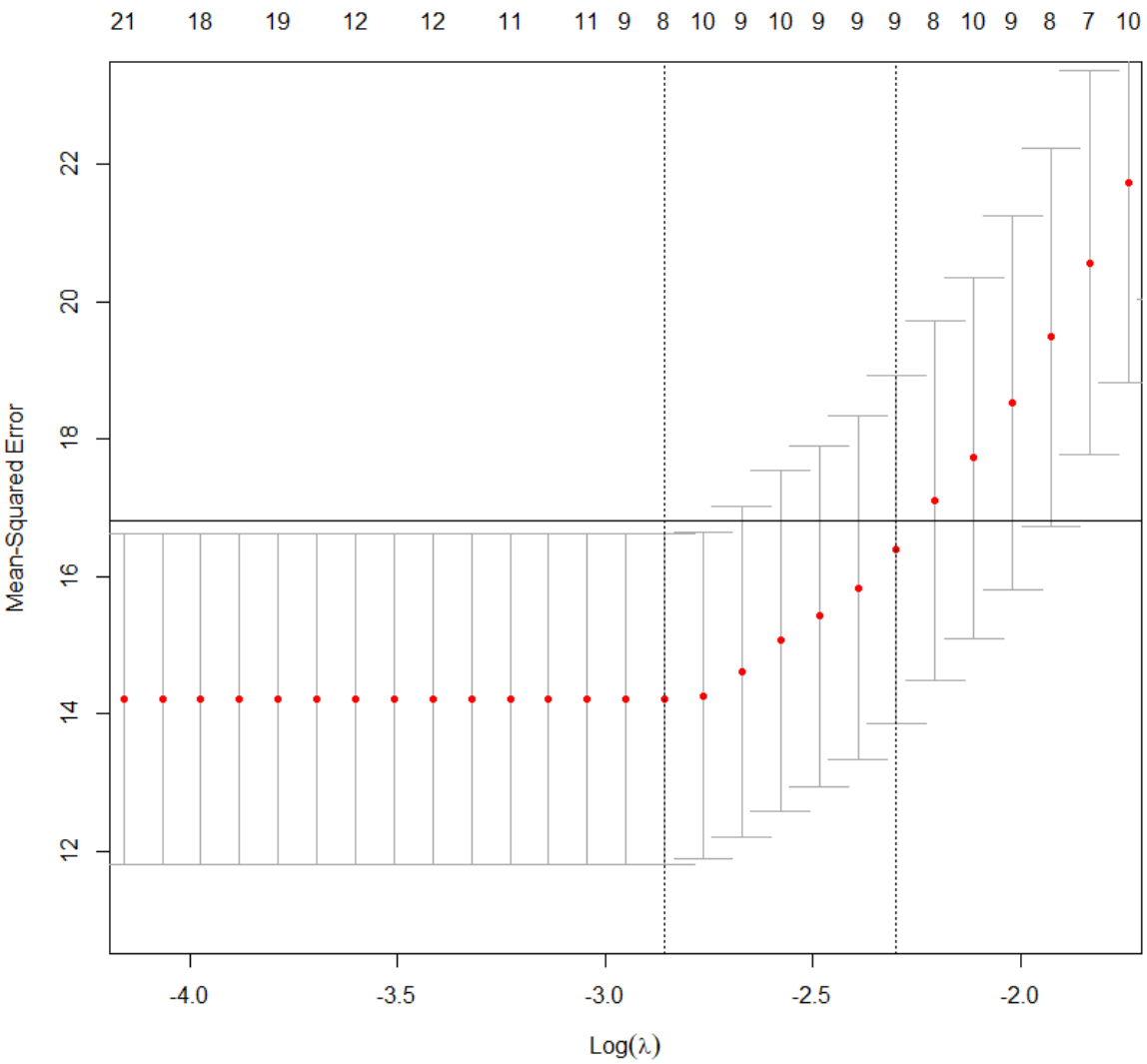
Does the information displayed in the plot suggests that the optimal lambda value results in a statistically significantly better prediction than log(lambda) = 4?

To answer this we zoom in on the graph where we can see both log(lambda) = -4 and our selected optimal lambda, see below:



The red dots are the mean CV errors for different lambdas, and the grey intervals are the confidence bounds for each lambda showing the variance of the CV error. We see that hose bounds overlap entirely for log(lambda) = -2,85 which we chose as our optimal lambda, and log(lambda) = -4.0. So no, the plot does not suggest that the optimal lambda results in a statistically significantly better

prediction than log (lambda) = −4. Since the complexity is lower for the optimal lambda and also has a lower CV error, the selected optimal lambda is obviously a better choice than log(lambda) = -4.

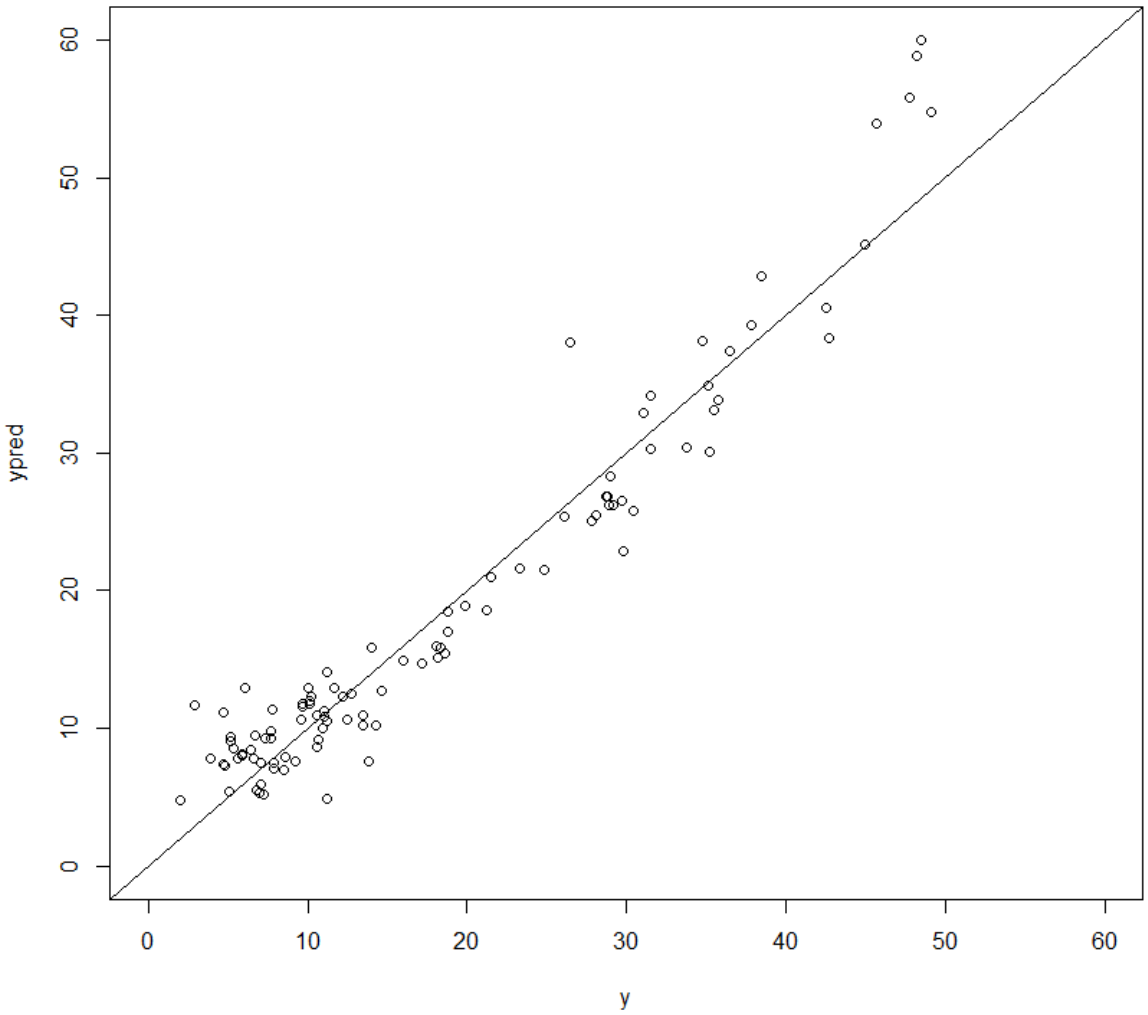


For clarification of the previous conclusion: If we would look at log (lambda) -2.85 and log (lambda) -1.8 or so, we would see that we can say that log (lambda) -2.85 results in a statistically significantly better prediction than log (lambda) = -1.8 since the CV error for lambda -2.85 statistically will be better than the CV error for log (lambda) -1.8.

LASSO regression model evaluation with the optimal lambda:

If we train a new model using our optimal lambda using all train data, we get a slightly more accurate model, since we now use all of our training data for training, compared to using 9/10 parts for training and 1/10 parts for validation as we do when using 10-fold cross-validation.

We can then predict the fat-levels in the test set using our model, and plot this together with the actual fat-levels in the test data, see below plot.



We can see that the model seems to preform well, especially on lower fat-levels. For fat-levels around 15 to 30 the model seems to slightly underestimate the fat-levels, and for fat-levels above 45 the model seem to slightly overestimate the fat-levels.

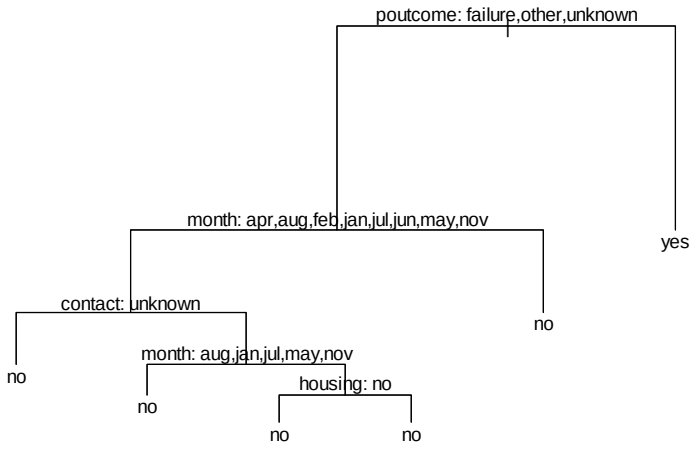
The MSE on training data for this model is 11.244, and the MSE on test data for this model is 13.2998, which is significantly better than the MSE on test data for any of the previous models (test MSE = 52.11 for LASSO without optimizing hyperparameter lambda, test MSE = 100.48 for Ridge

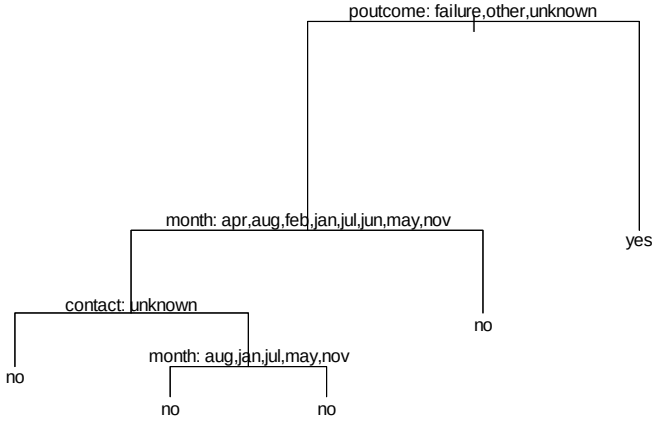
model). Since the train MSE is not close to 0 and the test MSE is only slightly larger than the training MSE, we can see that the model has no indication of being overfitted (and also no indication of underfitting since there exists a generalization gap of around 2.05). Thus, we can draw the conclusion that the LASSO model which uses the optimal lambda-value discovered through cross-validation performs well.

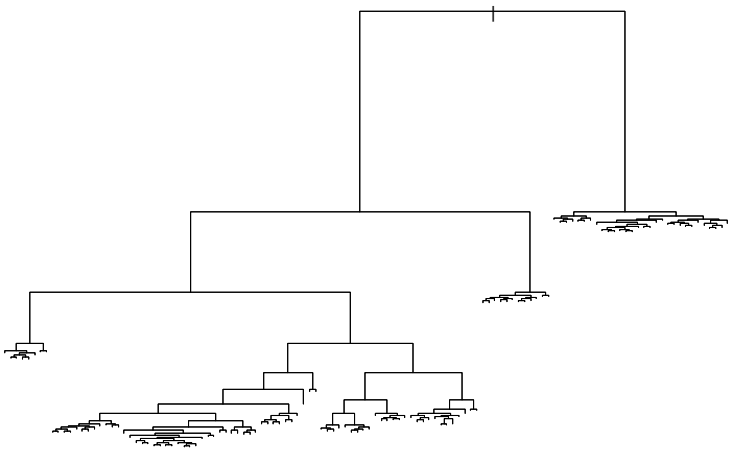
Assignment 2

2. Fit decision trees to the training data so that you change the default settings one by one (i.e. not simultaneously):
    a. Decision Tree with default settings.
    b. Decision Tree with smallest allowed node size equal to 7000.
    c. Decision trees minimum deviance to 0.0005.

    and report the misclassification rates for the training and validation data. Which model is the best one among these three? Report how changing the deviance and node size affected the size of the trees and explain why.







We get the above 3 trees.

The misclassification rates for the training and validation data was then calculated together with the number of terminal nodes (NTN) and the residual mean variance (RMV). These are presented in Table 1 below.

Decision tree	NTN	RMV	Train misclass	Validation misclass
1. Default	6	0.6022	0.1048	0.1092679
2. Node size	5	0.6097	0.1048	0.1092679
3. Min deviance	122	0.5213	0.09362	0.1115535

As we can see, the misclassification rates for tree 1 and 2 are the same for train and validation data. This can be explained by looking at the plots, and we see that the extra split in the second tree has the value “no” in both the left and right leaf node of the split, meaning that this split does not affect any of the predictions. Altough the label is “no” for both, the probabilities are different which is why this split is done, and if we continue splitting we one of the leaf nodes could later on produce a leaf node with the prediction “yes”.

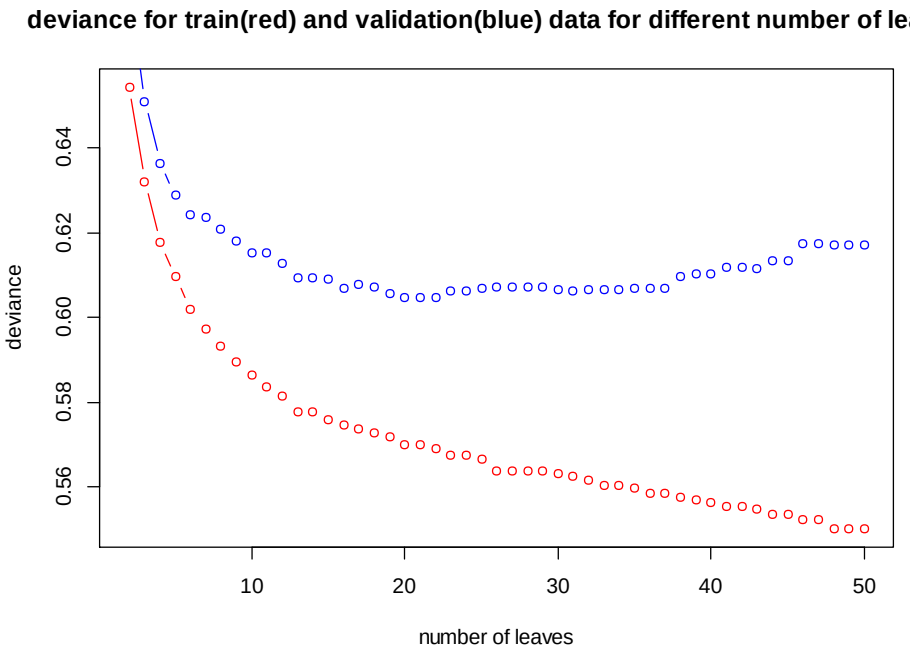
As we can see, tree number 3 is heavliy overfitted with 122 terminal nodes, and we can also see that it has a lower training error as can be espected for a overfitted model, and also a worse validation error as can also be expected since an overfitted model generalizes worse. Thus tree 1 and 2 are both good options, but since the extra leaf node in tree 1 does not contribute with anything (and is also slightly more complex than tree 2), then I would choose tree 2 as the optimal model here.

When we force the tree to have a larger minimum node size in tree 2 (from default of 10 to 7000), we restrict how small the nodes can be, thus we get one less leaf node since in the default tree, one of the 2 leaf nodes in the extra split must have had a node size which was smaller than 7000.

When we allow the minimum deviance to be smaller (the within-node deviance must be at least this time that of the root node for the node to be split – from 0.0005 to 0.01), then we relax the tree, which will result in more splits, since now even a very small deviance will make the tree split.

3. Use training and validation sets to choose the optimal tree depth in the model 2c: study the trees up to 50 leaves. Present a graph of the dependence of deviances for the training and the validation data on the number of leaves and interpret this graph in terms of bias-variance tradeoff. Report the optimal amount of leaves and which variables seem to be most important for decision making in this tree. Interpret the information provided by the tree structure (not everything but most important findings).

When calculating training error and validation error for 2:50 leaves and plotting the average errors, we get the following plot:



For the training error, the more leaf nodes, the more complex the model becomes, and thus the training error decreases since we are overfitting the model more and more byby allowing it to be more flexible and adapt better to the training data. Thus this is not relevant to look at.

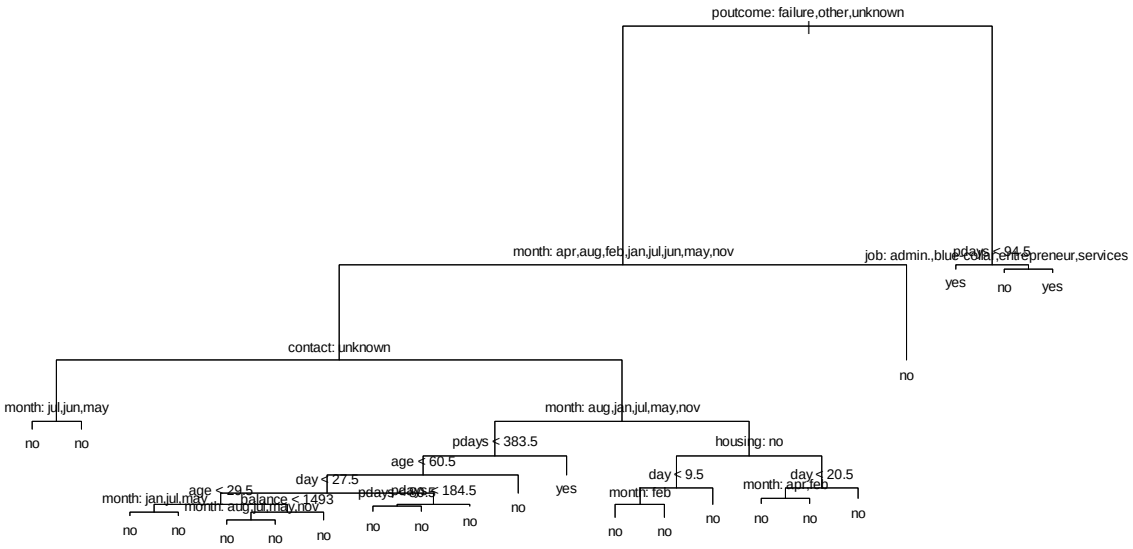
For the validation deviance, for a very simple modle with a low number of leaves, the model underfits and we have al arge bias error, but the variance is small, since the model is simple and not flexible enough which will result in a small variance if we repeat this with new data. But for a large number of leaf nodes, the model becomes complex and overfitted, and we will have a smaller bias, but a larger variance, since now the model is more flexible and adapts to the data very will, i.e. the mapping of input to output is very good due to the flexibilty of the model, and thus we get a lower bias, and if we repeat this for other data sets we will get a large variance due to the high flexibility.

The bias-variance tradeoff is even easier to see when analyzing the training errors, since there the bias goes towards 0 as model complexity increases and at the same time the variance increases.

```
> which.min(validationScore)
[1] 22
```

The optimal number of leaves, i.e. the number of leaves that minimzes the validation error, is 22. Here the model is neither underfitted or overfitted, and has both moderate bias and variance.

If we prune tree number 3 from assignment 1 to have 22 leaf nodes which we found to be the optimal number of leaves, we get the following tree:



And we can see that the variables that are actually used in the tree construction are the following:

```
Variables actually used in tree construction:
[1] "poutcome" "month"    "contact"  "pdays"    "age"      "day"      "balance"  "housing"  "job"
```

Furthermore we can interpret the information provided by the tree. First of all, since the first split is on variable “previous outcome”, this is most likely the most important feature. This does indeed seem to play a very important role, since if failure/other/unkown we take the left branch where a majority of leaf nodes are “no”, whereas if success we take the right branch, where we based on pdays and job quickly can determine what the outcome will be, and we also seem to have a high chance of getting a terminal node with “yes”. E.g. if previous outcome is success, and pdays > 94.5, we get a high chance of “yes”.

If we take the left branch, we can see that the only leaf node with a “yes” is the one at splitting criterion pdays < 383.5, thus if this is false, i.e. pdays > 383.5, we end up in the “yes” node, so this is also an important feature for us.

By analyzing the tree like this, we can go through the branches of the tree and see where we have the highest chance of getting a “yes”, e.g. if we go left at the first split, if contact is known, and for certain months, if pdays > 383.5 we have a high probabiliy of getting a “yes”, compared to the other outcomes in the left part of the tree where we most likely will get a “no”.

4. Estimate the confusion matrix, accuracy and F1 score for the test data by using the optimal model from step 3. Comment whether the model has a good predictive power and which of the measures (accuracy or F1-score) should be preferred here.

Confusion matrix, true no/yes-labels on the y-axis (left), and predicted no/yes on the x-axis(top).

```
> confusionMatrix
      pred
true     no   yes
  no  11872   107
  yes  1371   214
```

Accuracy and F1-score:

```
> accuracy
[1] 0.8910351
> F1
[1] 0.224554
```

As we can see from the confusion matrix, we have some imbalance in the data, where we have a lot of more no-labels(11872+107=11979) than yes-labels (1371+214=1585). If we just look at the accuracy, this is a bit misleading regarding model performance due to the imbalance. The model predicts much fewer “yes” than are actually true “yes”-labels. As we can see, predicting a true yes-label as “no” happens often here, and 1371 out of the total 1585 true “yes”-labels are predicted “no”, i.e. a lot of FN. The accuracy is only slightly better than what the model would perform if simply classifying everything as “no”, 11979/(11979+1585)=0.88314, which is due to the imbalance in the data.

Therefore it would be much more suitable to look at the F!”-accuracy instead, since it takes the imbalance into account, since it considers both the precision and recall. We have a rather low F1-score here at 0.224554 or 22.455%, which means that our model does not perform very well. Much of this is due to as discussed above, that the model predicts many true “yes” as “no”, i.e. many FN.

5. Perform a decision tree classification of the test data with the following loss matrix,

$$L = \begin{matrix} \\ Observed \end{matrix} \begin{matrix} Predicted \\ yes \\ no \end{matrix} \begin{pmatrix} 0 & 5 \\ 1 & 0 \end{pmatrix}$$

and report the confusion matrix for the test data. Compare the results with the results from step 4 and discuss how the rates has changed and why.

We now want to punish FN more than FP (FN punished 5 times harder than FP). Thus, the model must be 5 times as certain classifying a no (thus reducing the number of FN), meaning we only classify as "no" if we are very certain it is a "no". And in this case, we of course would prefer to have more FP than less FN, since we rather have more potential customers which in the end turn out to be "no"'s, than to have many customers that could have been "yes" predicted as no.

Thus we get this new confusion matrix:

```
> confusionMatrix_NewLoss
      pred
true      no   yes
  no   11030   949
  yes    771   814
```

And we now predict more "yes" which was our goal. Thus, our predicitons for "yes" are now much better, and we now have 814 TP's compared to 214. However, what we lose by doing this is of course an increased misclassification of "no", and now we have 11030 TN's, which is fewer than the previous 11872, but as discussed above, in our application this tradeoff is advantageous for us.

```
> accuracy_newLossMatrix
[1] 0.8731937
> F1_newLossMatrix
[1] 0.4862605
```

As discussed earlier, the accuracy is slightly lower but this is irrelevant.

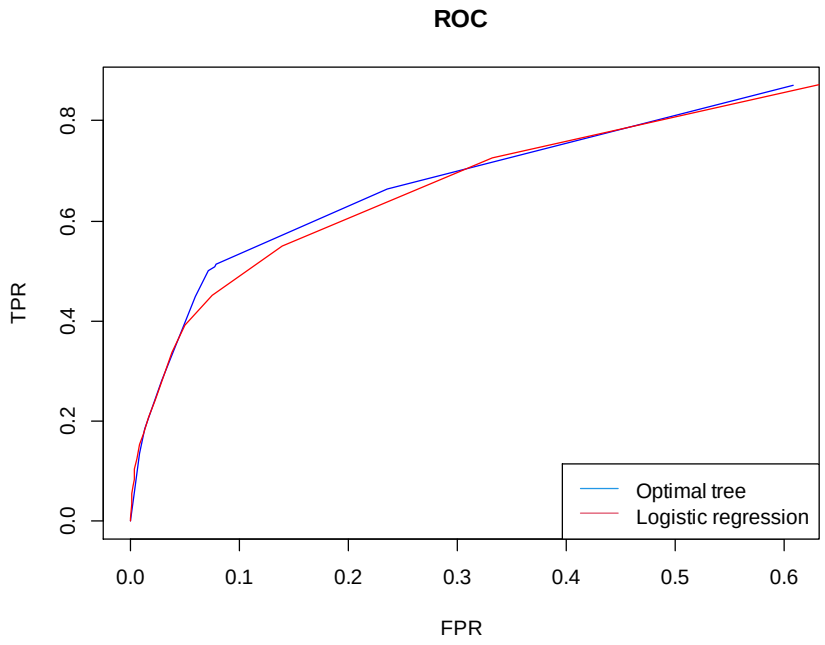
However the F1-score has increased to 0.48626 or 48.626% which is a large improvement from 22.455%. So we get a much better model now, which can be seen by comparing the F1-scores.

6. Use the optimal tree and a logistic regression model to classify the test data by using the following principle:

$$\hat{Y} = 1 \; if \; p(Y = 'good'|X) > \pi, otherwise \; \hat{Y} = 0$$

where $\pi = 0.05,0.1,0.15, \ldots 0.9,0.95$. Compute the TPR and FPR values for the two models and plot the corresponding ROC curves. Conclusion? Why precision-recall curve could be a better option here?

By this, and classify using the optimal tree and above values, as well as training a logistic regression model and classify using above values, we get the following ROC-curve:



The models have very similar performance. The area under the curve (AUC) can by simply looking at the plot barely be differentiated. But it appears as if the tree has a slightly larger AUC, thus the tree model has better predictive performance than the logistic regression model.

A Precision-Recall-Curve (PR-Curve) would in this case be better due to the imbalanced data, like with the discussion regarding accuracy vs F1-score, F1-score is better for imbalanced data, and the same goes for ROC-curve vs PR-curve, where PR-curve in this case would be a better way to evaluate the models.

This is due to, similarily to the argument with accuracy vs F1-score, the ROC-curve will still show good performance if we would just classify everything as the majority class, i.e. "no", whereas a PR-curve would take this imbalance into account.

Assignment 3

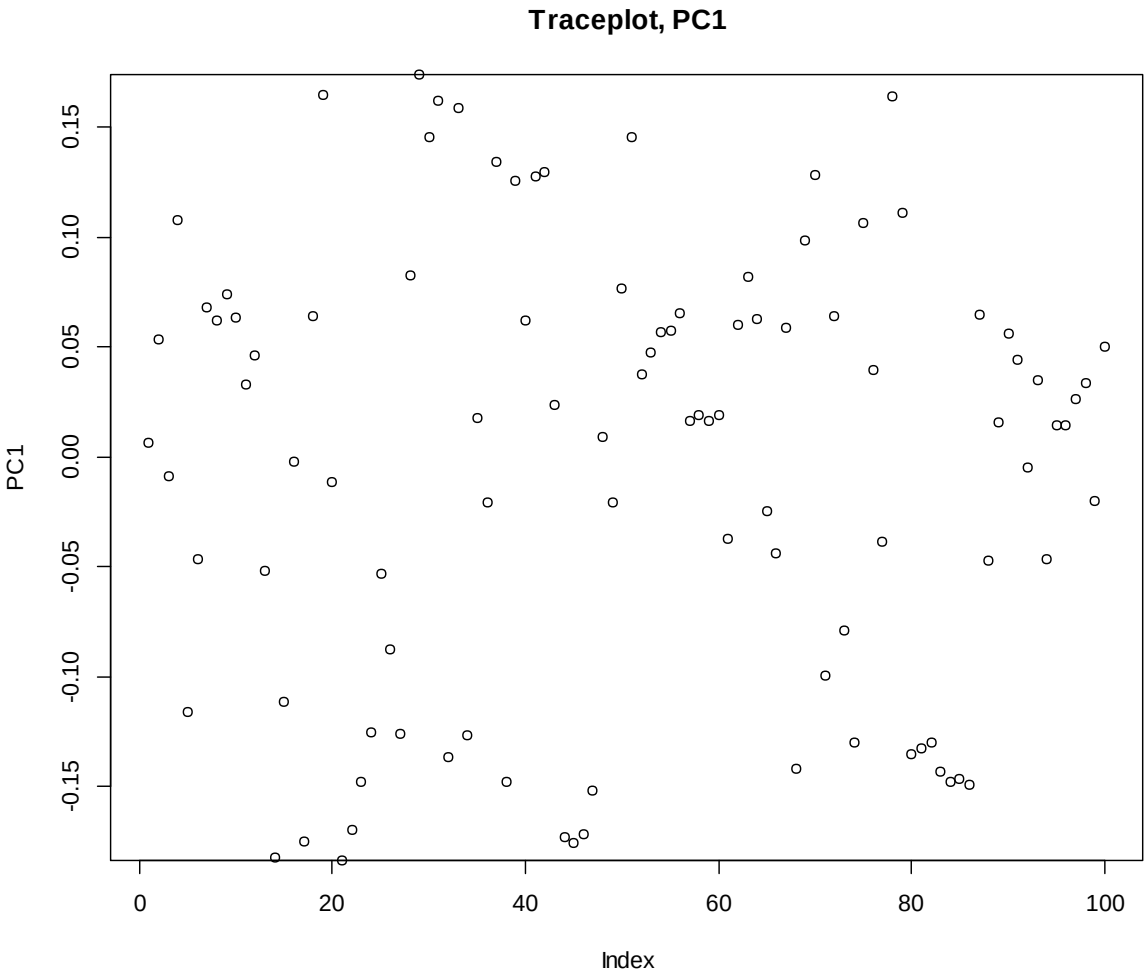
1. Scale all variables except of ViolentCrimesPerPop and implement PCA by using function eigen(). Report how many features are needed to obtain at least 95% of variance in the data. What is the proportion of variation explained by each of the first two principal components?

Scaling the data and calculating the covariance matrix, we can then use function eigen() to obtain the principal components (i.e. the "directions") which correspond to the eigenvectors of the sample covariance, and we obtain how much variation each direction corresponds to (i.e. the "amount" in each PC direction) which coresponds to the eigenvalues of the sample covariance.

By analyzing these eigenvalues, we can discover that out of the original 100 features, 35 features are needed to obtain at least 95% of the variance in the data. By looking at the first eigenvalue, we see that the first PC explains 25.02% of the total variation, whereas by looking at the second eigenvalue we see that the second PC explains 16.94% of the variation.

2. Repeat PCA analysis by using princomp() function and make the score plot of the first principle component. Do many features have a notable contribution to this component? Report which 5 features contribute mostly (by the absolute value) to the first principle component. Comment whether these features have anything in common and whether they may have a logical relationship to the crime level. Also provide a plot of the PC scores in the coordinates (PC1, PC2) in which the color of the points is given by ViolentCrimesPerPop. Analyse this plot (hint: use **ggplot2** package).

If we repeat the PCA we can perform a trace plot using the first PC. The trace plot shows us how the PC is expressed in terms of original features, so in this case it will tell us how much each of the original 100 features ($x_1,\ldots,x_{100}$) contribute to PC1. See below plot.



We can see that the contribution of the original features is quite scattered along the y-axis, only a few feaures have a contribution close to 0, and as we can see quite many features have a notable contribution to the PC and there are many features with a contribution of and absolute value larger than 0.15.

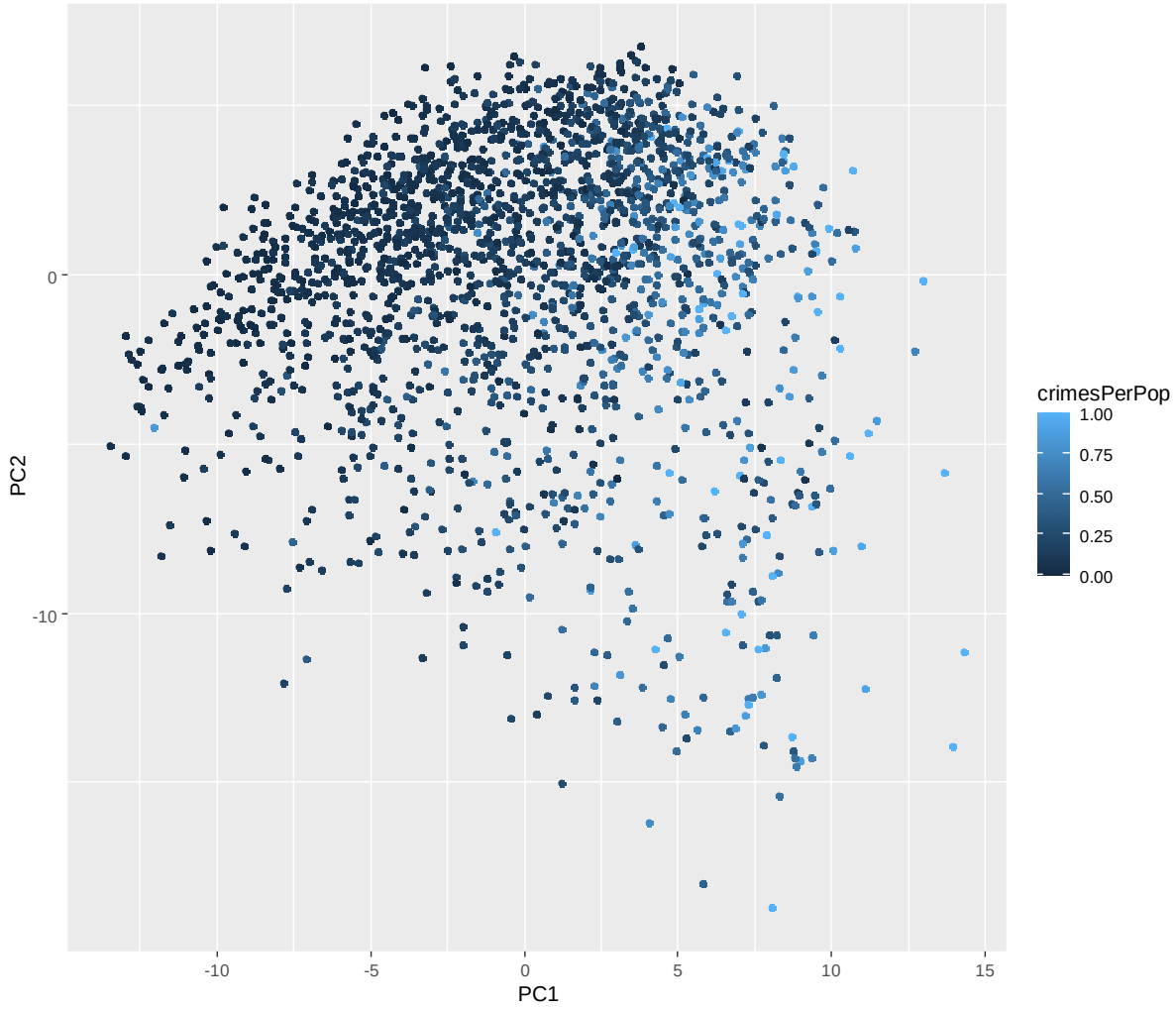
If we extract the 5 original features which contribute the most to the first PC, we can see that it is the following 5: Median family income, median household income, percentage of kids in family housing with two parents, percentage of households with investment / rent income and percentage of people under the poverty level.

```
> PC1_orig_val
    medFamInc     medIncome    PctKids2Par    pctWInvInc PctPopUnderPov
   -0.1833080    -0.1819830     -0.1755423     -0.1748683     0.1737978
```

One can quite easily see that all those features are related to the economical situation of a certain area. The first, second and fifth feature are quite clearly connected since the explicitly have to do with the income level. The same goes for the fourth feature, where one can assume that there is also a high correlation between having low income or living in poverty and not getting any investment/rent income. The third feature is also connected, since a family with 1 or 2 parents will have a vastly different family income. Thus, all the 5 features which contributed the most to PC1 (and since PC1 corresponds to the largest variance in the original data, thus these 5 features overall

correspond to the largest variance), are related to the economical sitation, and it makes sense that this would have a high effect on crime rates.

We can also plot the PC scores in the coordinates PC1, PC2. The scores represent what the coordinates of our data looks like in the coordinates of the first 2 principal components, so each point in our data is transferred from the original 100-dimensional coordinate system with $x_1,\ldots,x_{100}$ to this 2-dimensional coordinate system of only PC1 and PC2. See below plot.



It seems as if for high values on the PC1-axis, the rate of crimes increases. However, it is hard to see any large difference along the PC2-axis, and it is rather hard to compare due to the clustering at the top of the graph as compared to the bottom. From earlier, we saw that the 2 PC's are enough to explain 25+17 = 42% of the variation, and the plot confirms that PC1 has a larger impact than PC2.

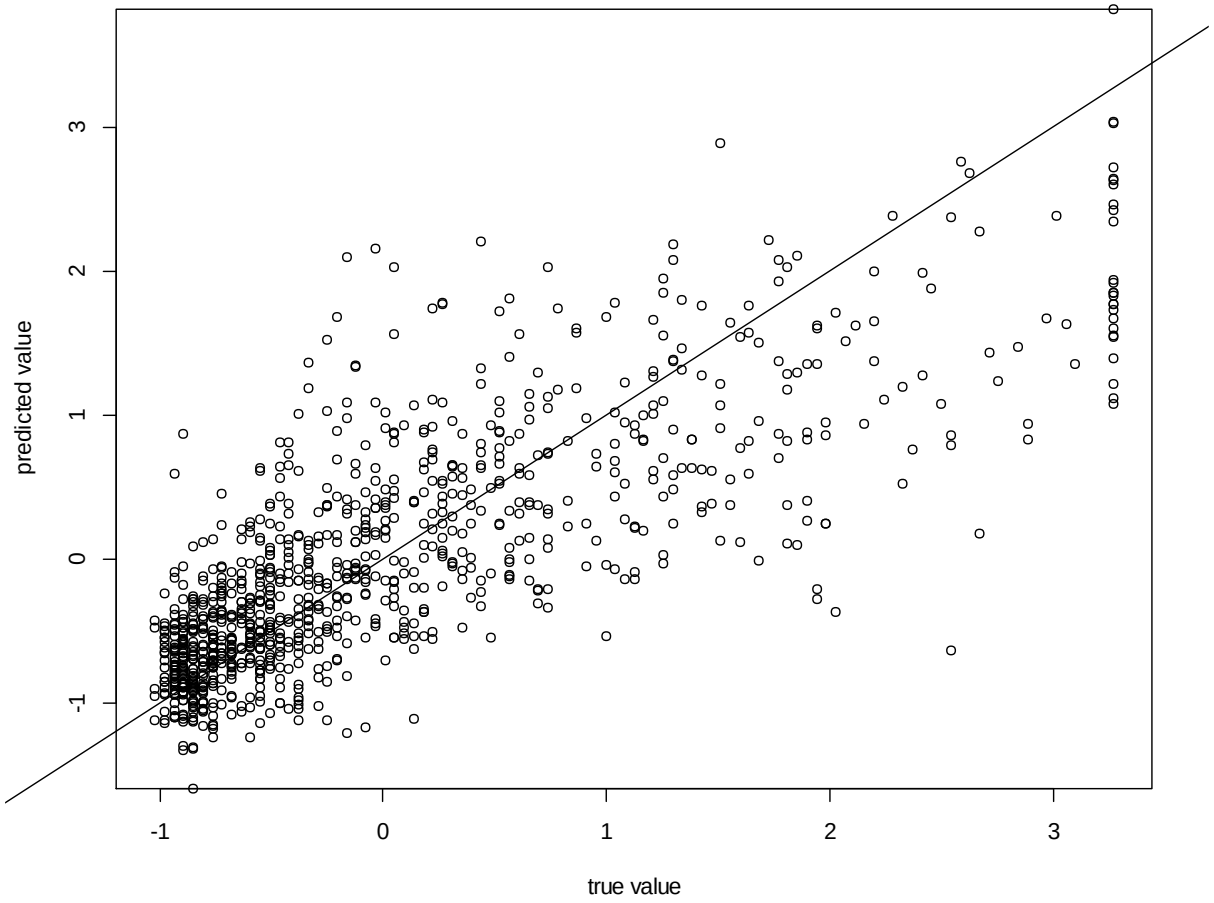
3. Scale the original data: both features and response, split data into training and test (50/50) and estimate a linear regression model from training data in which ViolentCrimesPerPop is target and all other data columns are features. Compute training and test errors for these data and comment on the quality of model.

Training a linear regression model and calculating MSE for training and test data, as well as R-squared, we get the following:

```
> MSETrain
[1] 0.2591772
> MSETest
[1] 0.4000579
> R2
[1] 0.000137232
```

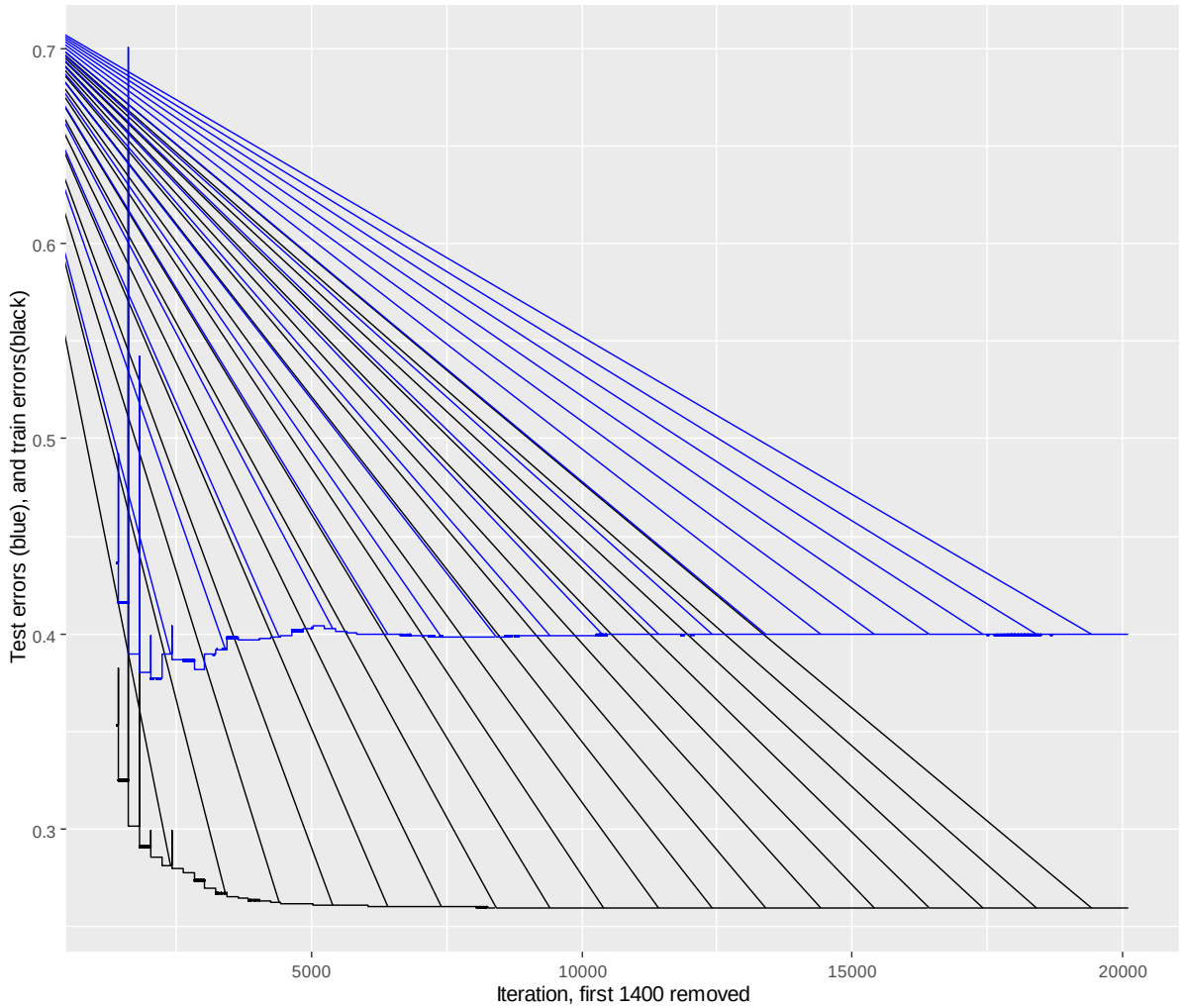
As expected, MSE for training set is lower than MSE for test set, but it is hard to tell if there are indications that the model is overfitting (MSE train not very close to 0), and no indication that the model is underfitting (the two MSE values are not very close to eachother). It is hard to draw any conclusion from the MSE values, as this should rather be used for comparing different models and not evaluate performance of a single model. However if we calculate the R-squared value, we will see that it is 0.0137%, which indicates that the model performs very poorly, since this value tells us the proportion of the variation in the dependent variable (y) that is predictable from the independent variables (x_1,…,x_{100}).

We can also plot the true labels with the predictions, and we can see that the model deos indeed not perform very well:



4. Implement a function that depends on parameter vector θ and represents the cost function for linear regression without intercept on the training data set. Afterwards, use BFGS method (optim() function without gradient specified) to optimize this cost with starting point $\theta^0 = 0$ and compute training and test errors for every iteration number. Present a plot showing dependence of both errors on the iteration number and comment which iteration number is optimal according to the early stopping criterion. Compute the training and test error in the optimal model, compare them with results in step 3 and make conclusions.
    a. **Hint 1**: don't store parameters from each iteration (otherwise it will take a lot of memory), instead compute and store test errors directly.
    b. **Hint 2**: discard some amount of initial iterations, like 500, in your plot to make the dependences visible.

If we optimize our model to find the optimal theta which minimizes test error, we implement a function which calculates test and training MSE and store them in vectors, and optimize this using the optim() function and initial theta = 0. If we do so, we can plot how training and test errors depend on number of iterations:



As we can see, as the iterations go on, the train error decreases due to overfitting. We can also see, that in early iterations testing and training error are similar, and this is an indication of underfitting. As our test error is increasing, we must apply early stopping in order to get the optimal model, and this increase in test error as training error is decreasing is well visualized in the beginning of above graph. If we select the iteration number which gave us the lowest MSE for testing data, this is iteration number 2183 which is the optimal number of iterations.

For iteration number 2183, we get the following MSE for train and test data:

```
> MSETrain
[[1]]
[1] 0.2858249

> MSETest
[[1]]
[1] 0.3769468
```

Compared to the previous part, MSE for training error is slightly higher (was 0.259), which is expected since we apply early stopping and the model is thus less overfitted. The MSE for test data is what we care about, and is slightly lower than in previous part (was 0.400), due to the fact that we stop before the MSE test starts increasing again due to overfitting. So overall, we have improved our model by applying early stopping. Had we not stopped early, we would have arrived in a similar

situation as the previous model i.e. we would overfit, which we can see in the graph where training error goes towards 0.4, and testing error towards 0.26.