Compulsory exercise 2: Group 10

TMA4268 Statistical Learning V2019

Martin Kvisvik Larsen

12 april, 2019

```
install.packages("knitr") #probably already installed
install.packages("rmarkdown") #probably already installed
install.packages("bestglm")# for subset selection with categorical variables
install.packages("glmnet")# for lasso
install.packages("tree") #tree
install.packages("randomForest") #for random forest
install.packages("ElemStatLearn") #dataset in Problem 2
#install.packages("BiocManager") #allows the use of BiocManager
#BiocManager::install(c("pheatmap")) #heatmap in Problem 2
```

```
install.packages("ggplot2")
install.packages("GGally") # for ggpairs
install.packages("caret") #for confusion matrices
install.packages("pROC") #for ROC curves
install.packages("e1071") # for support vector machines
install.packages("nnet") # for feed forward neural networks
```

Problem 1: Regression [6 points]

```
all=dget("https://www.math.ntnu.no/emner/TMA4268/2019v/data/diamond.dd")
dia_train=all$dtrain
dia_test=all$dtest
```

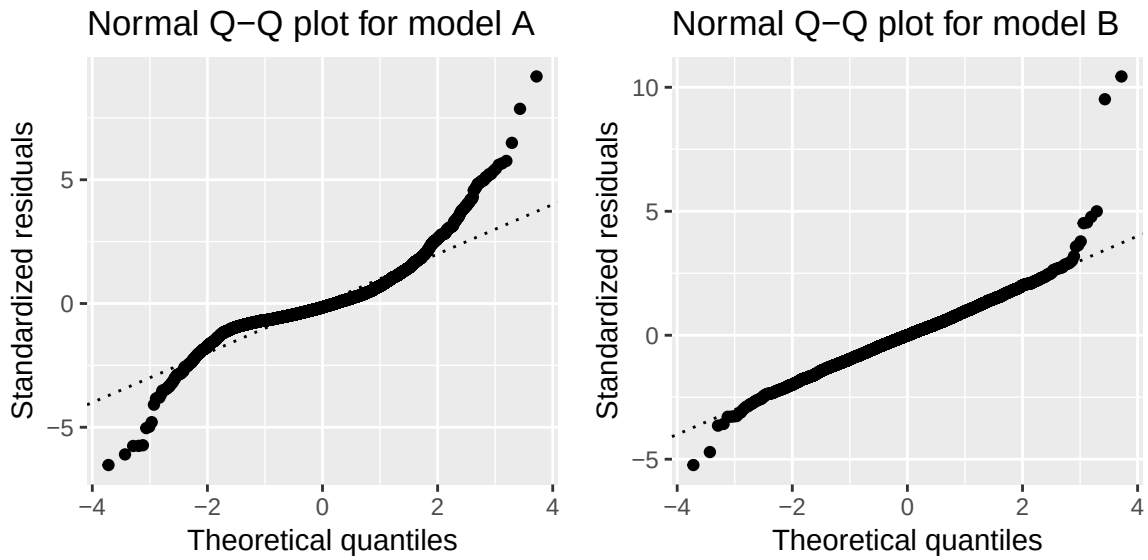
Q1: Would you choose `price` or `logprice` as response variable? Justify your choice. Next, plot your choice of response pairwise with `carat`, `logcarat`, `color`, `clarity` and `cut`. Comment.

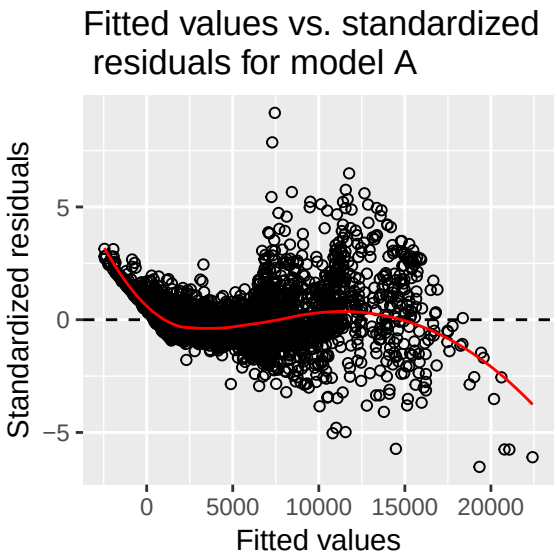
In order to access whether price or logprice is the better response variable in order to do regression, two linear models with the least square estimator are created. All the covariates in the dataset are used for both models, but model A has price as its response variable, while model B has logprice as its response variable.

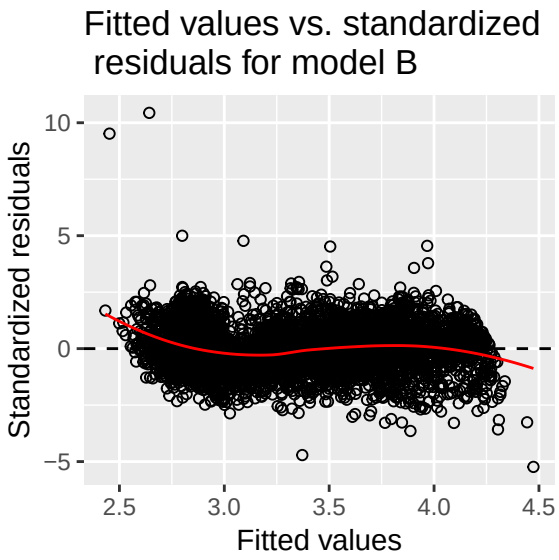
```
library(ggplot2)
library(grid)

# Setting up a linear regression model for each of the response variables
model_A = lm(formula = price ~ logcarat + carat + cut + color +
               clarity + depth + table + xx + yy + zz, data=dia_train)
model_B = lm(formula = logprice ~ logcarat + carat + cut + color +
               clarity + depth + table + xx + yy + zz, data=dia_train)

# Creating QQ-plot for model A
qq_plot_A = ggplot(model_A, aes(sample = .stdresid)) +
```

```
  stat_qq(pch = 19) +
  geom_abline(intercept = 0, slope = 1, linetype = "dotted") +
  labs(x = "Theoretical quantiles", y = "Standardized residuals",
       title = "Normal Q-Q plot for model A")

# Creating QQ-plot for model B
qq_plot_B = ggplot(model_B, aes(sample = .stdresid)) +
  stat_qq(pch = 19) +
  geom_abline(intercept = 0, slope = 1, linetype = "dotted") +
  labs(x = "Theoretical quantiles", y = "Standardized residuals",
       title = "Normal Q-Q plot for model B")

# Plotting the fitted values vs. standardized residuals for model A
res_plot_A = ggplot(model_A, aes(.fitted, .stdresid)) +
  geom_point(pch = 21) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  geom_smooth(se = FALSE, col = "red", size = 0.5, method = "loess") +
  labs(x = "Fitted values", y = "Standardized residuals",
       title = "Fitted values vs. standardized \n residuals for model A")

# Plotting the fitted values vs. standardized residuals for model B
res_plot_B = ggplot(model_B, aes(.fitted, .stdresid)) +
  geom_point(pch = 21) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  geom_smooth(se = FALSE, col = "red", size = 0.5, method = "loess") +
  labs(x = "Fitted values", y = "Standardized residuals",
       title = "Fitted values vs. standardized \n residuals for model B")
```

```
qq_plot_A
qq_plot_B
res_plot_A
res_plot_B
```

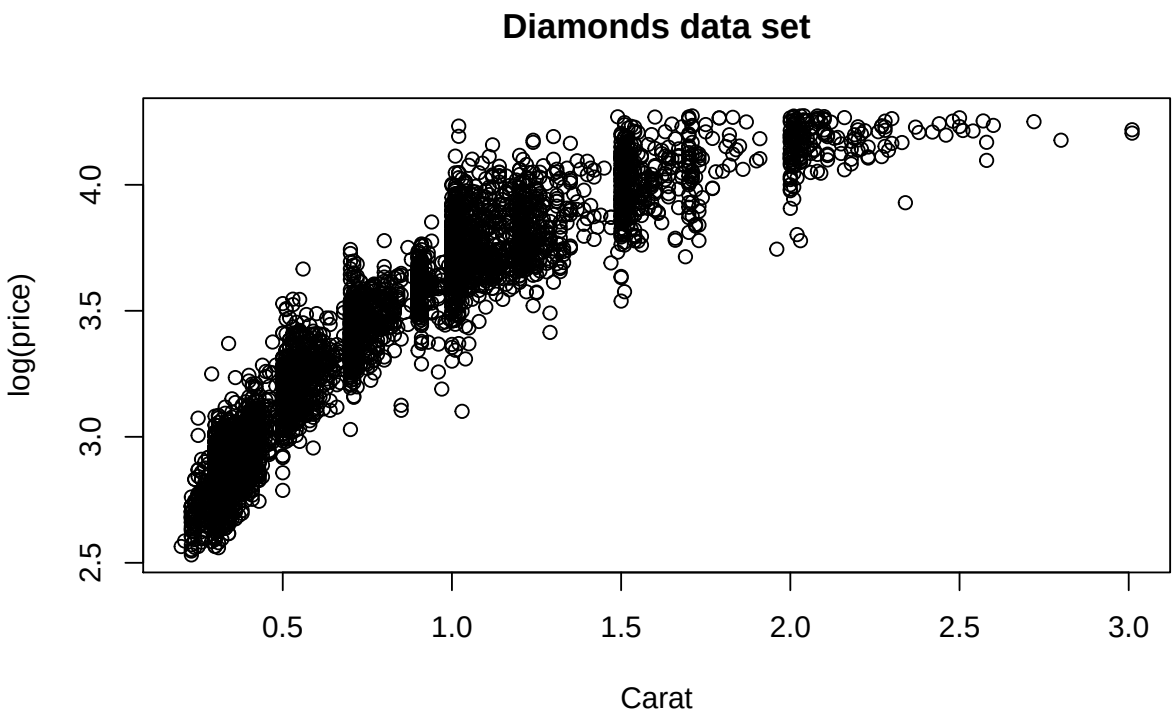


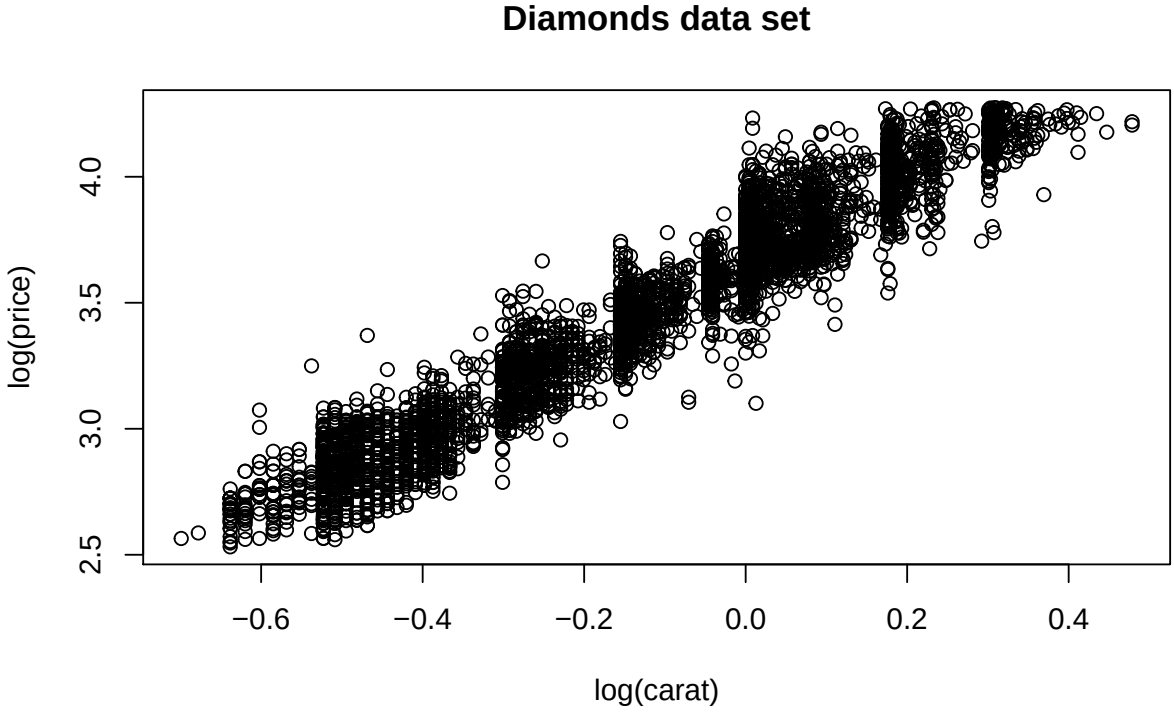


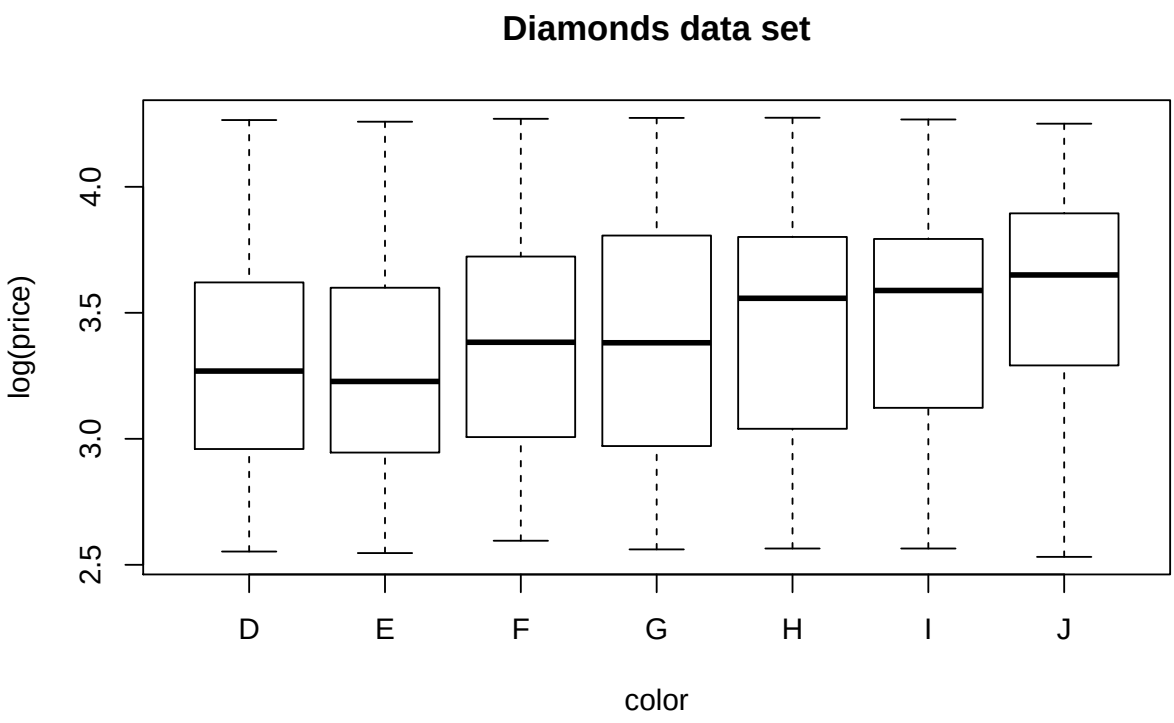


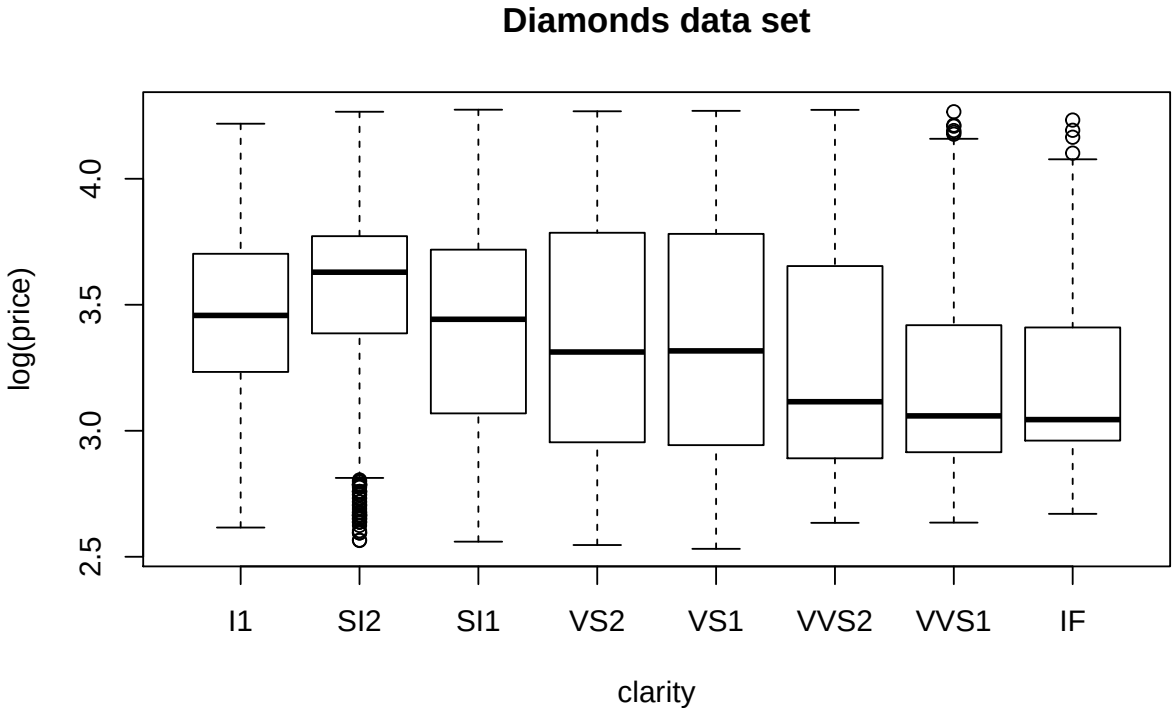
Based on the QQ-plots and the fitted values vs. standardized residuals plots for the two models, I choose to use logprice as my response variable. From model B's QQ-plot one can see that the plot is much closer to a straight line than that for model A, meaning that model B is closer to being normally distributed than model A. Additionally from the fitted values vs. standardized residuals for the two model one can see that the standardized residuals of model B have a closer to constant spread around 0 than that those of model A.

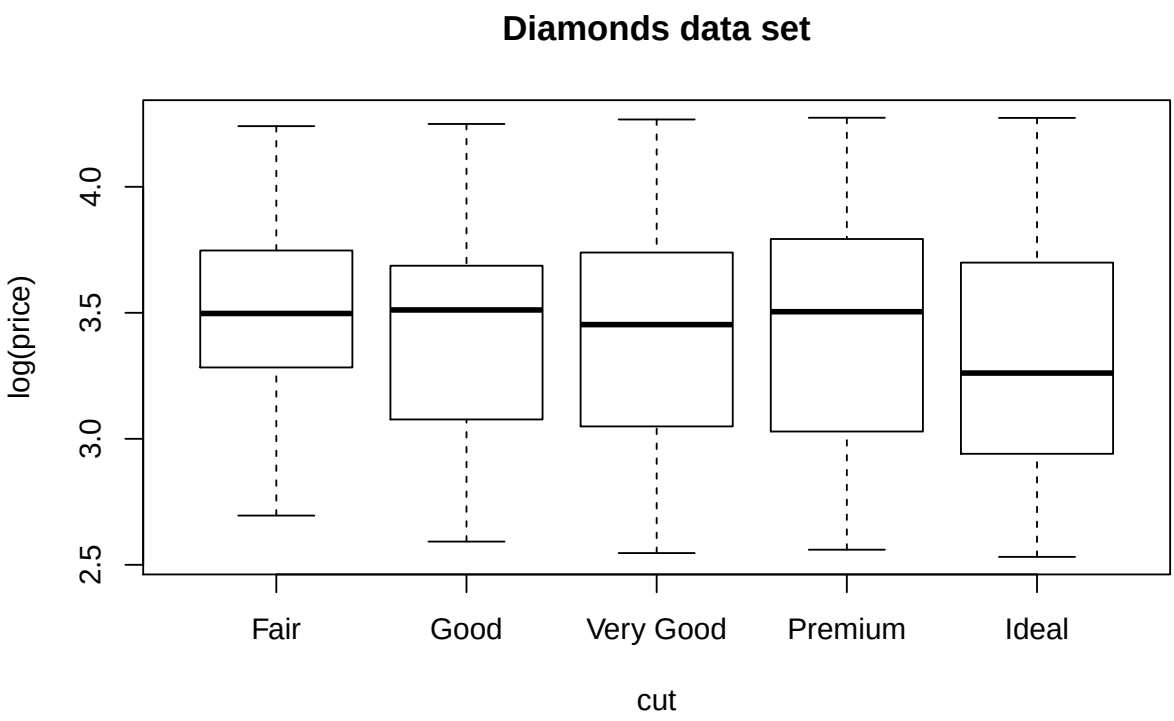
```
plot(dia_train$carat, dia_train$logprice, xlab="Carat", ylab="log(price)",
     main="Diamonds data set")
plot(dia_train$logcarat, dia_train$logprice, xlab="log(carat)", ylab="log(price)",
     main="Diamonds data set")
plot(dia_train$color, dia_train$logprice, xlab="color", ylab="log(price)",
     main="Diamonds data set")
plot(dia_train$clarity, dia_train$logprice, xlab="clarity", ylab="log(price)",
     main="Diamonds data set")
plot(dia_train$cut, dia_train$logprice, xlab="cut", ylab="log(price)",
     main="Diamonds data set")
```


Diamonds data set
log(price)
2.5
3.0
3.5
4.0
0.5
1.0
1.5
2.0
2.5
3.0
Carat


Diamonds data set
log(price)
2.5
3.0
3.5
4.0
−0.6
−0.4
−0.2
0.0
0.2
0.4
log(carat)


Diamonds data set
log(price)
2.5
3.0
3.5
4.0
D
E
F
G
H
I
J
color


Diamonds data set
log(price)
2.5
3.0
3.5
4.0
I1
SI2
SI1
VS2
VS1
VVS2
VVS1
IF
clarity



From the pairwise plots we can see that there is a increasing, yet non-linear relationship between logprice and carat, while there is increasing but close to linear relationship between logprice and logcarat. From the logprice vs. color box plot one can see that there is a slight upward trend in logprice from the "best" color index D to the "worst" color index J. This could indicate that a diamond's color is not a deciding factor on its own for the diamond's logprice (and thus price), seeing as the diamonds with the best color index are in general less expensive than the ones with worse indices. A similar trend can be seen from the logprice vs. clarity box plot, where the diamonds with the better clarity indices IF and VVS2 are not the ones with the highest logprice and the diamonds with the worse clarity indices I1 and SI1 are not the one with the lowest logprice, in general. This can indicate that clarity does not have a strong direct correlation on a diamond's logprice. A similar argument can be made for the cut covariate. It is also possible that these trends are caused by a correlation between color, clarity and cut and other covariates. This could for instance have been examined by plotting color, clarity and cut against carat, where a clear positive correlation with logprice was seen.

Use the local regression model $\texttt{logprice} = \beta_0 + \beta_1 \texttt{carat} + \beta_2 \texttt{carat}^2$ weighted by the tricube kernel K_{i0}.

Q2: What is the predicted price of a diamond weighting 1 carat. Use the closest 20% of the observations.

```
x0 = 1

# R uses the tricubic kernel for weighting as default, hence the weights
# parameter is unspecified. The 20% closest observations are used by specifying
# span = 0.2.
dia_locreg = loess(formula = logprice ~ carat, degree=2, data=dia_train, span=0.2)
dia_logprice_pred_locreg = predict(dia_locreg, x0)

dia_price_pred_locreg = 10^dia_logprice_pred_locreg
```

The predicted price for a diamond of 1 carat(s) is: 5101

Q3: What choice of β_1, β_2 and K_{i0} would result in KNN-regression?

KNN-regression estimates $\hat{f}(x_0)$ based on the mean of the response of the K near neighbours. KNN-regression does not use the covariates for sample i, x_i, directly in the regression but rather the value of the response variable y_i. x_i is only used to calculate which sample points are in the neighbourhood of x_0. Hence the following parameters would result in KNN-regression:

$$\beta_1 = 0 \tag{1}$$

$$\beta_2 = 0 \tag{2}$$

$$K_{i0} = \begin{cases} \frac{1}{K} & x_i \in \mathcal{N}_0, \\ 0 & x_i \notin \mathcal{N}_0 \end{cases} \tag{3}$$

Q4: Describe how you can perform model selection in regression with AIC as criterion.

For each model the AIC, defined as follows, is calculated:

$$\text{AIC} = \frac{1}{n\sigma^2}(\text{RSS} + 2d\hat{\sigma}^2) \tag{4}$$

The AIC criterion is an estimate of the test MSE, which makes it useful if there is not enough data to split the dataset into a training, validation and test set. By using the AIC as criterion one can skip the validation set by calculating the AIC over the training set and do model selection based it. Since AIC includes the number of predictors in the model, d, more complex models are penalized more. Thus one can simply select the model with the lowest AIC.

Q5: What are the main differences between using AIC for model selection and using cross-validation (with mean squared test error MSE)?

The main difference between using AIC for model selection and using cross-validation is that using the AIC the mean squared test error is estimated based on the fitted models and the training set, while with cross-validation the mean squared test error is estimated by calculating the mean squared test error over the validation set. Additionally the AIC approach of doing model selection makes more assumptions about the true underlying model than the cross-validation approach, which makes it less flexible. Another difference is that the cross-validation approach is much more computationally expensive than the AIC approach.

Q6: See the code below for performing model selection with `bestglm()` based on AIC. What kind of contrast is used to represent `cut`, `color` and `clarity`? Write down the final best model and explain what you can interpret from the model.

```
library(bestglm)

ds_train=as.data.frame(within(dia_train,{
  y=logprice    # setting reponse
  logprice=NULL # not include as covariate
  price=NULL    # not include as covariate
  carat=NULL    # not include as covariate
  }))

dia_glm = bestglm(Xy=ds_train, IC="AIC")$BestModel
summary(dia_glm)
coef(dia_glm)
```

```
dia_glm$contrasts$cut
dia_glm$contrasts$color
dia_glm$contrasts$clarity
```

```
##
## Call:
## lm(formula = y ~ ., data = data.frame(Xy[, c(bestset[-1], FALSE),
##     drop = FALSE], y = y))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.31283 -0.03717  0.00033  0.03564  0.58991
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.985969   0.042993  69.452  < 2e-16 ***
## logcarat      1.580675   0.029042  54.428  < 2e-16 ***
## cutGood       0.028558   0.005747   4.970 6.93e-07 ***
## cutVery Good  0.040480   0.005285   7.660 2.22e-14 ***
## cutPremium    0.042984   0.005295   8.118 5.90e-16 ***
## cutIdeal      0.054829   0.005223  10.498  < 2e-16 ***
## colorE       -0.021787   0.003031  -7.189 7.51e-13 ***
## colorF       -0.038892   0.003107 -12.519  < 2e-16 ***
## colorG       -0.067999   0.003024 -22.488  < 2e-16 ***
## colorH       -0.112815   0.003235 -34.877  < 2e-16 ***
## colorI       -0.165072   0.003686 -44.783  < 2e-16 ***
## colorJ       -0.223651   0.004543 -49.233  < 2e-16 ***
## claritySI2    0.174988   0.007933  22.060  < 2e-16 ***
## claritySI1    0.251631   0.007876  31.950  < 2e-16 ***
## clarityVS2    0.315556   0.007924  39.824  < 2e-16 ***
## clarityVS1    0.346409   0.008051  43.027  < 2e-16 ***
## clarityVVS2   0.402528   0.008233  48.894  < 2e-16 ***
## clarityVVS1   0.433326   0.008549  50.684  < 2e-16 ***
## clarityIF     0.473681   0.009154  51.745  < 2e-16 ***
## xx            0.069331   0.006610  10.489  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05978 on 4980 degrees of freedom
## Multiple R-squared:  0.9815, Adjusted R-squared:  0.9814
## F-statistic: 1.39e+04 on 19 and 4980 DF,  p-value: < 2.2e-16
##
##  (Intercept)     logcarat      cutGood cutVery Good   cutPremium
##   2.98596920   1.58067481   0.02855836   0.04048016   0.04298389
##     cutIdeal       colorE       colorF       colorG       colorH
##   0.05482931  -0.02178654  -0.03889196  -0.06799861  -0.11281494
##       colorI       colorJ   claritySI2   claritySI1   clarityVS2
##  -0.16507179  -0.22365118   0.17498764   0.25163144   0.31555599
##   clarityVS1  clarityVVS2  clarityVVS1    clarityIF           xx
##   0.34640865   0.40252834   0.43332618   0.47368068   0.06933053
## [1] "contr.treatment"
## [1] "contr.treatment"
## [1] "contr.treatment"
```

The treatment contrast is used for `cut`, `color` and `clarity`. This means that the different levels of `cut`, `color` and `clarity` are contrasted relative to a baseline level, which in this case are `Fair`, `D` and `I1` respectively for the three covariates. Additionally the `Intercept` is the mean of the baseline group. In the regression problem R introduces dummy variables (indicator variables) for each of levels (excluding the baseline levels) of the categorical covariates. For example for the `cut` covariate the following dummy variables are introduced: `cutGood`, `cutVery Good`, `cutPremium` and `cutIdeal`. Here the `cutGood` variable would be equal to 1 if the `cut` covariate was equal to `Good` and 0 otherwise.

The final best model is from the model selection is:

$$\hat{\text{logprice}} = \begin{bmatrix} 1 \\ \text{logcarat} \\ \text{cutGood} \\ \text{cutVery Good} \\ \text{cutPremium} \\ \text{cutIdeal} \\ \text{colorE} \\ \text{colorF} \\ \text{colorG} \\ \text{colorH} \\ \text{colorI} \\ \text{colorJ} \\ \text{claritySI2} \\ \text{claritySI1} \\ \text{clarityVS2} \\ \text{clarityVS1} \\ \text{clarityVVS2} \\ \text{clarityVVS1} \\ \text{clarityIF} \\ \text{xx} \end{bmatrix}^\top \begin{bmatrix} 2.985969 \\ 1.580675 \\ 0.028558 \\ 0.040480 \\ 0.042984 \\ 0.054829 \\ -0.021787 \\ -0.038892 \\ -0.067999 \\ -0.112815 \\ -0.165072 \\ -0.223651 \\ 0.174988 \\ 0.251631 \\ 0.315556 \\ 0.346409 \\ 0.402528 \\ 0.433326 \\ 0.473681 \\ 0.069331 \end{bmatrix} \tag{5}$$

From the best model one can see that the covariates `depth`, `table`, `yy` and `zz` has been left out of the model, which indicates that they are not strong predictors. From the model one can see that there is a strong positive correlation between `logprice` and `logcarat`. From the coefficients of the `cut` dummy variables one can see that in general higher qualities of `cut` relative to the baseline level (`Fair`) leads to a higher `logprice`, even though the correlation is not that strong. From the coefficients of the `color` dummy variables one can see that a decreasing quality of `color` relative to the baseline level (`D`) leads to a decrease in `logprice`. The decrease in `logprice` due to relative differences in `color` is stronger than the increase in `logprice` due to relative differences in `cut`. From the `clarity` dummy variable coefficients one can in general see an increase in `logprice` for increasing levels (`SI1`, `SI2`, `VS1`, `VVS1`, `VVS2` and `IF`) relative to the baseline level (`I1`). The change in `logprice` due to relative changes in `clarity` is stronger than the changes in `logprice` due to relative changes in `cut` and `color`. However, one can also see a slight decrease in the standard error for increasing levels of `cut`, a slight increase for decreasing levels of `color` and a slight increase for increasing levels of `clarity`.

Q7: Calculate and report the MSE of the test set using the best model (on the scale of `logprice`).

```
dia_mse_test_glm = mean((dia_test$logprice - predict(dia_glm, dia_test))^2)
```

The mean squared test error of the best model is: 0.0036

Q8: Build a model matrix for the covariates `~logcarat+cut+clarity+color+depth+table+xx+yy+zz-1`. What is the dimension of this matrix?

```
# Model matrix for training set
dia_matrix_train = model.matrix(~ logcarat + cut + clarity + color + depth
                                + table + xx + yy + zz - 1, data = dia_train)
# Model matrix for test set
dia_matrix_test = model.matrix(~ logcarat + cut + clarity + color + depth
                               + table + xx + yy + zz - 1, data = dia_test)

dia_matrix_dims = dim(dia_matrix_train)
```

The dimensions of the model matrix are: []

Q9: Fit a lasso regression to the diamond data with `logprice` as the response and the model matrix given in Q8. How did you find the value to be used for the regularization parameter?

```
library(glmnet)

# alpha=1 means that Lasso regression is utilized
# Using cross validation to select lambda
dia_cv_lasso = cv.glmnet(x=dia_matrix_train, y=dia_train$logprice, alpha=1)
dia_cv_lasso_best_lambda = dia_cv_lasso$lambda.min

# Creating the Lasso regressor with the lambda value that minimizes the cross validation
# error
dia_lasso = glmnet(x=dia_matrix_train, y=dia_train$logprice, alpha=1,
                   lambda = dia_cv_lasso_best_lambda)
coef(dia_lasso)
```

```
## 25 x 1 sparse Matrix of class "dgCMatrix"
##                           s0
## (Intercept)   3.050230e+00
## logcarat      1.588901e+00
## cutFair      -4.731752e-02
## cutGood      -1.489853e-02
## cutVery Good -2.337768e-03
## cutPremium    .
## cutIdeal      1.279676e-02
## claritySI2    1.405974e-01
## claritySI1    2.168782e-01
## clarityVS2    2.806773e-01
## clarityVS1    3.112062e-01
## clarityVVS2   3.671705e-01
## clarityVVS1   3.976273e-01
## clarityIF     4.377507e-01
## colorE       -1.981794e-02
## colorF       -3.662724e-02
## colorG       -6.542357e-02
## colorH       -1.102818e-01
## colorI       -1.625192e-01
## colorJ       -2.207556e-01
## depth         3.545491e-05
## table         4.281761e-04
## xx            6.169559e-02
## yy            3.615818e-03
## zz            2.412922e-03
```

The regularization parameter λ is selected by performing cross-validation and picking the value of λ from the model with the minimum mean squared cross validation error. The best fit value of λ based on the mean squared cross validation error is: 0.0001293

Q10: Calculate and report the MSE of the test set (on the scale of `logprice`).

```
dia_mse_test_lasso = mean((dia_test$logprice - predict(dia_lasso, dia_matrix_test))^2)
```

The mean squared test error for the best fit Lasso regressor is: 0.003637

Q11: A regression tree to model is built using a *greedy* approach. What does that mean? Explain the strategy used for constructing a regression tree.

A *greedy* approach means that one takes a top-down approach to a problem in which one selects the best solution to each subproblem of the problem. The solution to the problem is thus created from the selected solutions to the subproblems. If the problem has certain attributes such an approach to creating a solution is optimal, however this is in general not the case.

A greedy approach is taken when constructing a regression tree because it is not computationally feasible to check every partition of non-overlapping regions of the predictor space, $R_1, ..., R_j$. A recursive binary splitting approach is taken instead. One starts at the top of the regression tree and split the predictor space into two regions $R_1(j, s)$ and $R_2(j, s)$ defined as:

$$R_1(j, s) = \{x | x_j < s\} \tag{6}$$

$$R_2(j, s) = \{x | x_j \geq s\} \tag{7}$$

The partitioning, i.e. the selection of the predictor x_j and the threshold s is found by minimizing the RSS for the two regions, given by:

$$\sum_{i:x_i \in R_1(j,s)} \left(y_i - \hat{y}_{R_1}\right)^2 + \sum_{i:x_i \in R_2(j,s)} \left(y_i - \hat{y}_{R_2}\right)^2 \tag{8}$$

The splitting yields two non-overlapping regions or "branches". The binary splitting procedure is then continued in order to build the regression tree. For each iteration each split is only allowed to depend on one predictor, such that two new branches are introduced in the regression tree. For each iteration only the best split at that particular iteration is performed, that is the split that gives the smallest RSS. Splits that might lower the overall RSS are not considered. This successive splitting is performed until some stopping criterion is met, for example based on the number of observations in each region or on some RSS threshold.

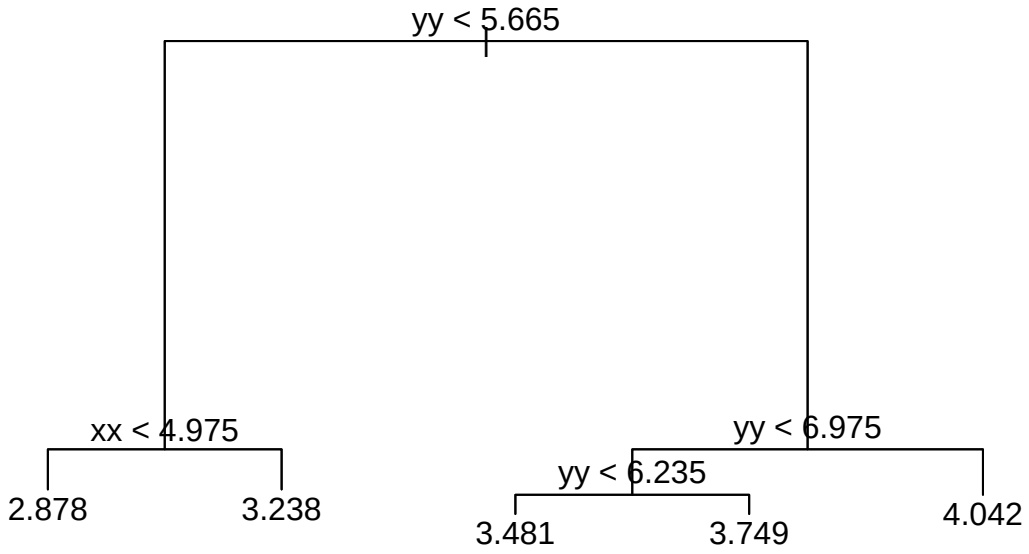
Q12: Is a regression tree a suitable method to handle both numerical and categorical covariates? Elaborate.

A regression tree is a suitable method to handle both numerical and categorical covariates even though the splitting procedure is computationally infeasible if the categorical covariates have q possible unordered values. However one can order the outcomes of the categorical covariates such that they have increasing correlation with the response variable. One can split as if the categorical covariate values were ordered and still perform an optimal split (Fisher 1958).

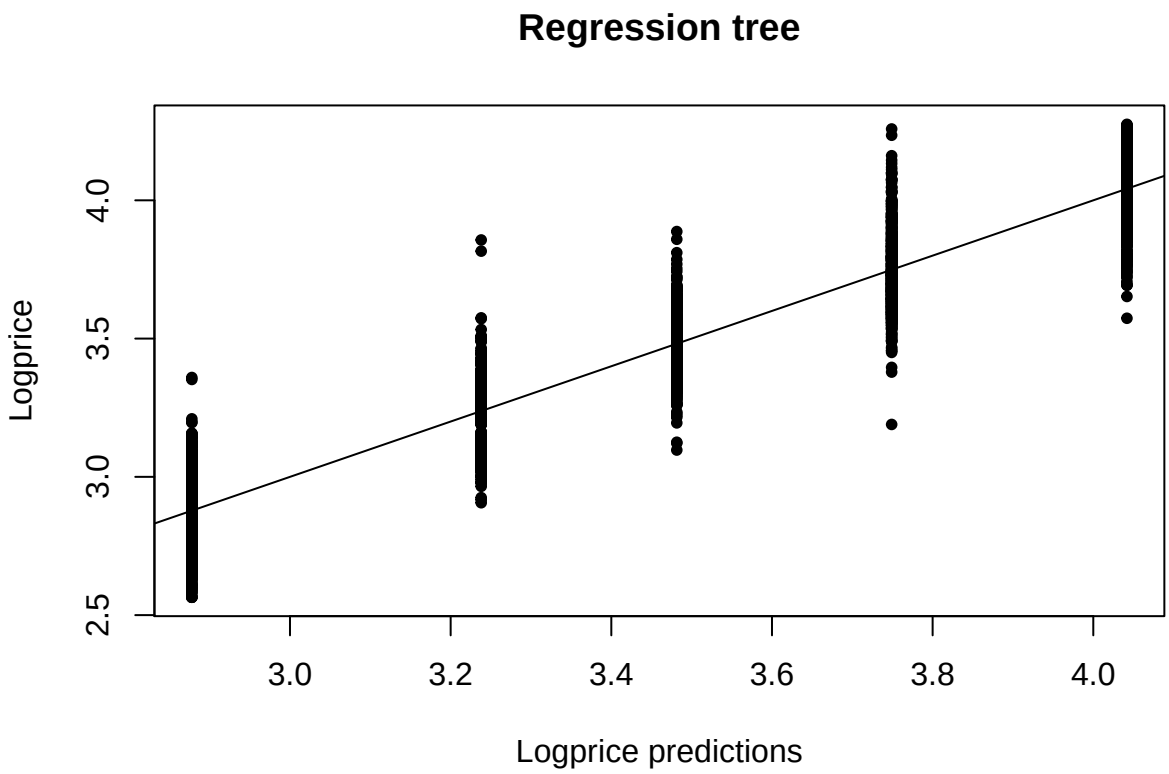
Q13: Fit a (full) regression tree to the diamond data with `logprice` as the response (and the same covariates as for c and d), and plot the result. Comment briefly on you findings.

```
library(tree)

dia_tree = tree(formula = logprice ~ logcarat + cut + clarity + color
                + depth + table + xx + yy + zz - 1, data=dia_train)
```

```r
dia_logprice_pred_tree = predict(dia_tree, dia_test)

# Plot of the regression tree
plot(dia_tree, type = "proportional")
text(dia_tree, pretty = 1)
```




```r
# Plot of prediction of response variable vs. response variables
plot(x = dia_logprice_pred_tree, y = dia_test$logprice, pch = 20,
     main = "Regression tree",
     xlab = "Logprice predictions",
     ylab = "Logprice")
abline(0, 1)
```




```
summary(dia_tree)
```

```
##
## Regression tree:
## tree(formula = logprice ~ logcarat + cut + clarity + color +
##     depth + table + xx + yy + zz - 1, data = dia_train)
## Variables actually used in tree construction:
## [1] "yy" "xx"
## Number of terminal nodes:  5
## Residual mean deviance:  0.01623 = 81.08 / 4995
## Distribution of residuals:
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -0.5595000 -0.0916200 -0.0004152  0.0000000  0.0879300  0.4925000
```

From the plot of the fitted regression tree structure one can see that it is quite simple with only 5 leaf nodes, which makes the tree simple to interpret. Another thing that is worth noticing is that only two covariates, `xx` and `yy`, are used to divided the predictor space of the model. This is peculiar as the `yy` predictor was left out of the best model from subset selection, indicating that it was not a strong predictor. However, `yy` is the variable that is used in the top split of the tree, which would indicate that it is the strongest predictor for the regression tree.

From the plot of the logprice predictions made by the regression tree vs. logprices on the test set one can see that the regression tree does a does an ok job at predicting logprices, despite the fact that it is very simple. The straight line indicates perfect predictions and one can see that the logprices are evenly spread around the predictions with reasonably small changes in the variance throughout the prediction interval.

Q14: Calculate and report the MSE of the test set (on the scale of `logprice`).

```r
dia_mse_test_tree = mean((dia_test$logprice - predict(dia_tree, dia_test))^2)
```

The mean squared test error for the regression tree is: 0.01716

Q15: Explain the motivation behind bagging, and how bagging differs from random forest? What is the role of bootstrapping?

The motivation behind bagging is that decision trees usually suffer from high variance, meaning that small changes in the predictors can lead to large changes in the fitted model. Bagging or bootstrap aggregating is a method to reduce the variance of decision trees, where one uses the average of the predictions of B fitted decision trees instead of the prediction of just a single fitted decision tree. Assuming that one has B fitted decision trees; $\hat{f}^{*1}(\mathbf{x}), ..., \hat{f}^{*B}(\mathbf{x})$ the prediction when using bagging is given by the following equation:

$$\hat{f}_{\text{bag}}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(\mathbf{x}) \tag{9}$$

Bagging differs from random forest in that the decision trees used in bagging are allowed to split at any of the p predictors in each split, while the decision trees used in random forests are only allow to split at a random subset of $m < p$ predictors in each split. This reduces the correlation between the different trees and reduces the variance of the prediction more than regular bagging.

Bootstrapping plays a crucial role in bagging and random forests because they require the availability of B independent data sets of observations of a random variable X with the same mean μ and variance σ^2. In general one do not have access to many independent data sets, so one uses bootstrapping to create B data sets from a single data set instead.

Q16: What are the parameter(s) to be set in random forest, and what are the rules for setting these?

The parameters to be set in a random forest is the number of trees, B, and the number of predictors that are allowed to consider at each splitting, m. The number of trees is not a tuning parameter and the best choice is to choose it large enough so that random forest stabilizes. The rules for setting m is different for classification and regression problems. For classification the rule for setting m is $m \approx \sqrt{p}$, while for regression the rule for setting m is $m = \frac{p}{3}$.

Q17: Boosting is a popular method. What is the main difference between random forest and boosting?

Boosting, like in random forest, contains B decision trees, but the difference is that each individual tree in boosting is grown sequentially using information from the previous version of that particular tree. This is done by fitting residual trees with $d + 1$ leaf nodes to the residuals of the model and then updating the trees based on the residual trees weighted with a parameter λ, also known as slow learning. The procedure is repeated until some stopping criterion is met. The tuning parameters of boosting are different from the random forest. In boosting the number of trees B the learning rate λ and the number of splits d have to be tuned, while as in random forest the number of predictors used for splitting m is the only tuning parameter.

If the residual trees in the algorithm are denoted as $\hat{f}^b(\mathbf{x})$ the update rule for the model $\hat{f}(\mathbf{x})$ is:

$$\hat{f}(\mathbf{x}) \leftarrow \hat{f}(\mathbf{x}) + \lambda\hat{f}^b(\mathbf{x}) \tag{10}$$

The update rule for the residuals is:

$$r_i \leftarrow r_i - \lambda\hat{f}^b(\mathbf{x}) \tag{11}$$

The equation for the boosted model is:

$$\hat{f}(\mathbf{x}) = \sum_{b=1}^{B}\lambda\hat{f}^b(\mathbf{x}) \tag{12}$$

Q18: Fit a random forest to the diamond data with `logprice` as the response (and the same covariates as before). Comment on your choice of parameter (as decribed in Q16).

```
library(randomForest)

p = 9

# Random forest parameters
rf_B = 500
rf_m = floor(p/3)    # Set rule for random forest for regression

dia_rf = randomForest(formula = logprice ~ logcarat + cut + clarity + color
                      + depth + table + xx + yy + zz - 1, data = dia_train,
                      mtry = rf_m, ntree = rf_B, importance = TRUE)
```

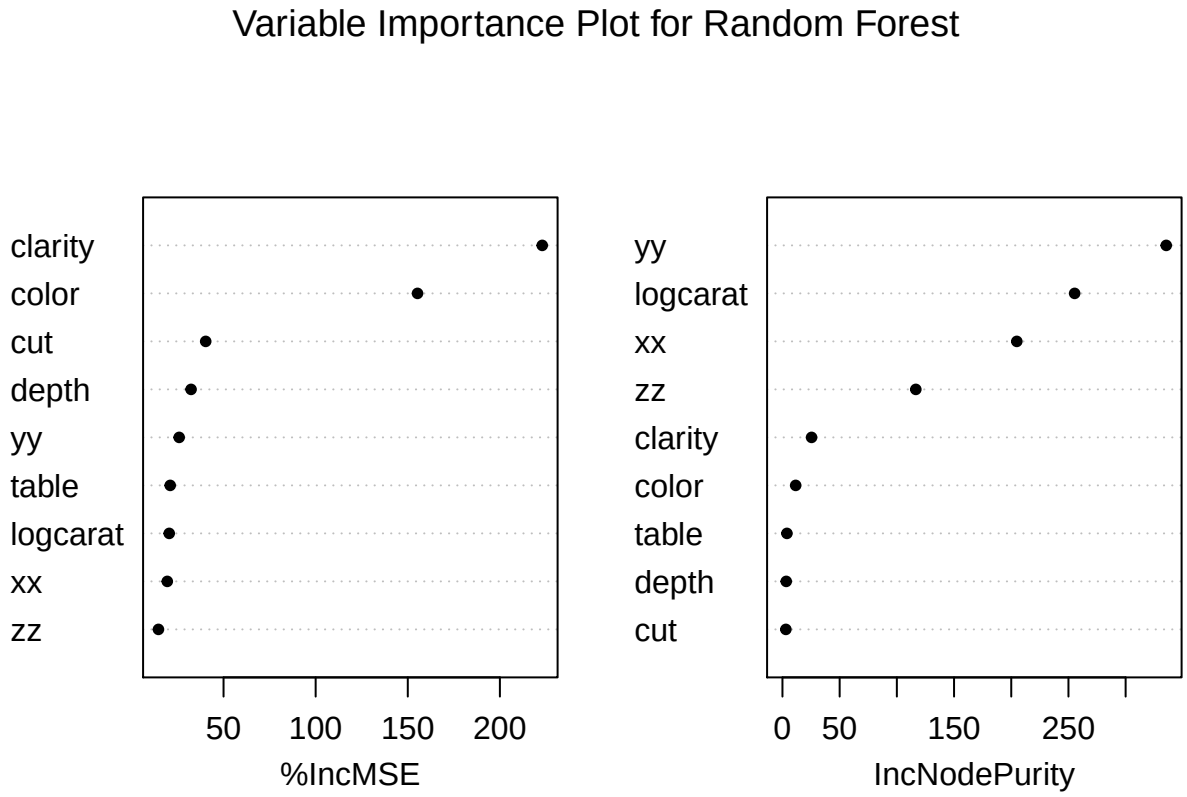
Since the prediction of `logprice` is a regression problem, the number of predictors to be used in the splitting procedure of the trees m is set according to the rule:

$$m = \frac{p}{3} \tag{13}$$

The number of trees B is not a tuning parameter, it just has to be large enough. Hence B is set equal to the default value of 500, which is based on a rule of thumb.

Q19: Make a variable importance plot and comment on the plot. Calculate and report the MSE of the test set (on the scale of `logprice`).

```
library(randomForest)

varImpPlot(dia_rf, pch = 20, main = "Variable Importance Plot for Random Forest")
```




```
dia_mse_test_rf = mean((dia_test$logprice - predict(dia_rf, dia_test))^2)
```

From the increase in node purity vs. predictor subplot of the variable importance plot for the random forest one can see that there is a significant leap in increase in node purity for the predictors `yy`, `logcarat`, `xx` and `zz` compared to the other predictors. This indicates that `yy`, `logcarat`, `xx` and `zz` are the stronger predictors for performing splitting, with `yy` being the strongest one. This result agrees with the results from the single regression tree, which only used `yy` and `xx` for splitting.

However, this does not agree with the results from the percentage increase in the mean squared error plot. From the plot one can see that the order of importance of the variables has changed quite a lot from the increase in node purity plot. From the percentage increase in mean squared error plot one can see that `clarity` and `color` are the most important variables in terms of the increase in mean squared error. This agrees somewhat with the results from the best model of subset selection, which excluded `depth`, `table`, `yy` and `zz` and predicted rather strong positive correlation between `clarity` and `color` and `logprice`. It also agrees somewhat with the results from the Lasso regression model, which predicted very small correlations between `depth`, `table`, `yy` and `zz` and `logprice`. However both the model from the subset selection and the Lasso regression model predicted strong positive correlation between `logcarat` and `logprice`, which has been rendered as one of the less important predictors in the random forest.

The mean squared test error for the random forest with $B = $500 and $m = $3 is: 0.002795

Q20: Finally, compare the results from c (subset selection), d (lasso), e (tree) and f (random forest): Which method has given the best performance on the test set and which method has given you the best insight into the relationship between the price and the covariates?

Summary of the mean least square for the different methods:

- Subset selection: 0.0036
- Lasso regression: 0.003637
- Regression tree: 0.01716
- Random forest: 0.002795

From the mean squared test errors of the four models one can see that the random forest achieved the best performance on the test set. The model from the subset selection did however give me the most insight into the relationship between the price and the covariates as it left unimportant covariates out of the model.

Problem 2: Unsupervised learning [3 points]

Q21: What is the definition of a principal component score, and how is the score related to the eigenvectors of the matrix $\hat{\mathbf{R}}$.

Denoting the standardized data as $\mathbf{Z} \in \mathbb{R}^{n \times p}$, consisting of n data samples of p predictors, each sample of the standardized data as $\mathbf{Z}_i$ and each loading vector used in the decomposition as $\mathbf{\Phi}_j$, the full principal components decomposition can be denoted as:

$$\begin{gathered} \mathbf{T} = \mathbf{Z}\mathbf{\Phi} \\ \begin{bmatrix} \mathbf{T}_1 & \mathbf{T}_2 & \dots & \mathbf{T}_m \end{bmatrix} = \begin{bmatrix} \mathbf{Z}_1 \\ \mathbf{Z}_2 \\ \vdots \\ \mathbf{Z}_n \end{bmatrix} \begin{bmatrix} \mathbf{\Phi}_1 & \mathbf{\Phi}_2 & \dots & \mathbf{\Phi}_m \end{bmatrix} \end{gathered} \tag{14}$$

The definition of a principal component score is the project of the data points along a specific loading vector $\mathbf{\Phi}_j$. For example in the case of the first principal component the scores of the principal component $t_{11}, t_{21}, ..., t_{n1}$ are given by the following equations:

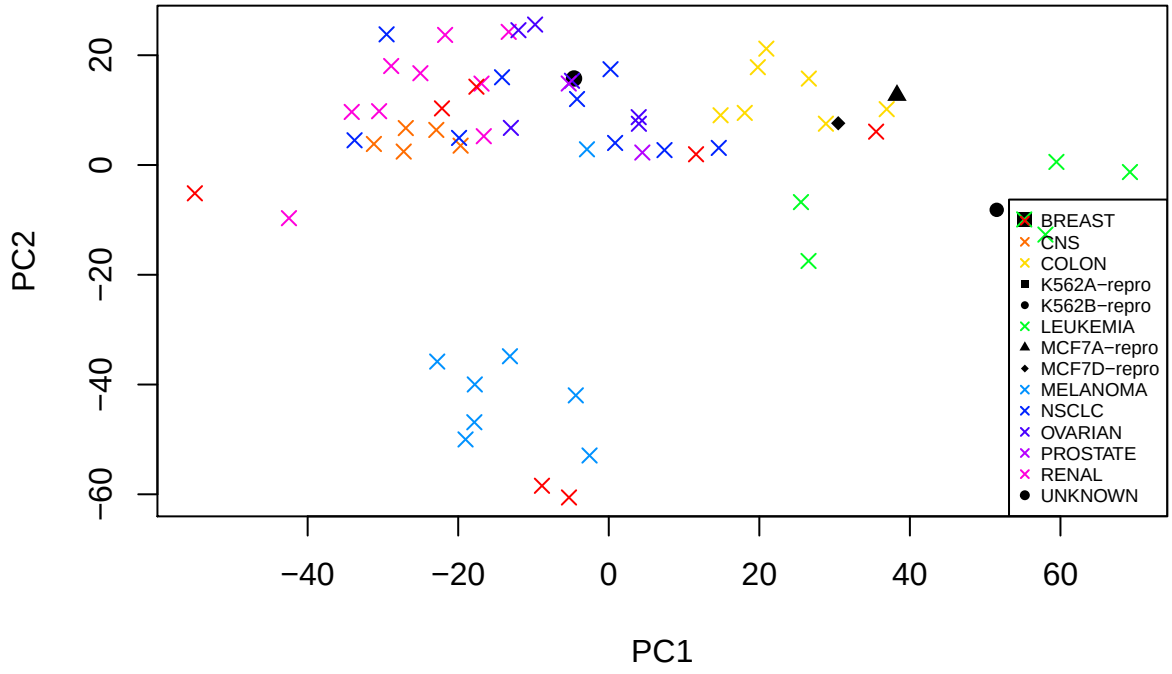
$$\begin{gathered} \mathbf{T}_1 = \mathbf{Z}\mathbf{\Phi}_1 \\ \begin{bmatrix} t_{11} \\ t_{21} \\ \vdots \\ t_{n1} \end{bmatrix} = \begin{bmatrix} z_{11} & z_{12} & \dots & z_{1p} \\ z_{21} & z_{22} & \dots & z_{2p} \\ \vdots & & \ddots & \vdots \\ z_{n1} & z_{n2} & \dots & z_{np} \end{bmatrix} \begin{bmatrix} \phi_{11} \\ \phi_{21} \\ \vdots \\ \phi_{n1} \end{bmatrix} \\ \begin{bmatrix} t_{11} \\ t_{21} \\ \vdots \\ t_{n1} \end{bmatrix} = \begin{bmatrix} z_{11}\phi_{11} + z_{12}\phi_{21} + \ldots + z_{1p}\phi_{p1} \\ z_{21}\phi_{11} + z_{22}\phi_{21} + \ldots + z_{2p}\phi_{p1} \\ \vdots \\ z_{n1}\phi_{11} + z_{n2}\phi_{21} + \ldots + z_{np}\phi_{p1} \end{bmatrix} \end{gathered} \tag{15}$$

The score of the principal components are related to the eigenvectors of the matrix $\hat{\mathbf{R}}$ through the fact that for scores $\mathbf{T}_i$ the corresponding loading vector $\mathbf{\Phi}_i$ on which the data points are projected is the eigenvector corresponding to i-th largest eigenvalue of $\hat{\mathbf{R}}$.

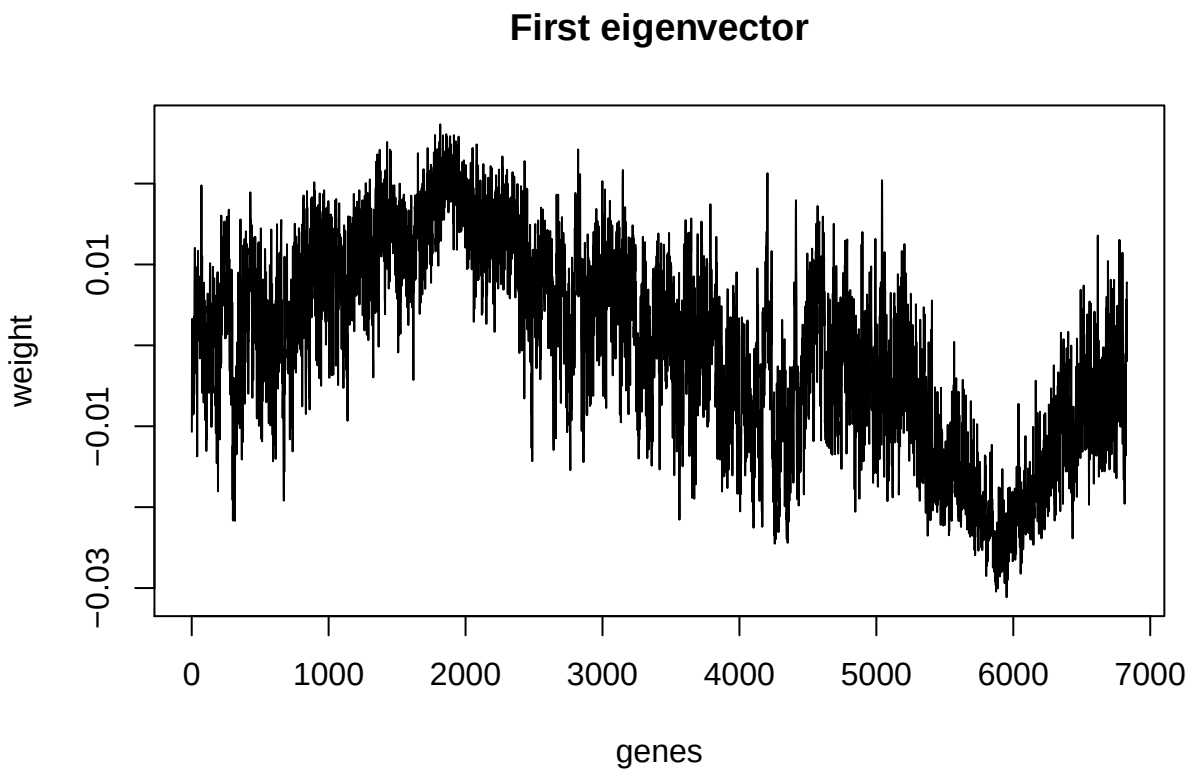
Q22: Explain what is given in the plot with title "First eigenvector". Why are there only $n = 64$ eigenvectors and $n = 64$ principal component scores?

```
library(ElemStatLearn)
X=t(nci) #n times p matrix
table(rownames(X))
ngroups=length(table(rownames(X)))
cols=rainbow(ngroups)
```

```
cols[c(4,5,7,8,14)] = "black"
pch.vec = rep(4,14)
pch.vec[c(4,5,7,8,14)] = 15:19

colsvsnames=cbind(cols,sort(unique(rownames(X))))
colsamples=cols[match(rownames(X),colsvsnames[,2])]
pchvsnames=cbind(pch.vec,sort(unique(rownames(X))))
pchsamples=pch.vec[match(rownames(X),pchvsnames[,2])]

Z=scale(X)

pca=prcomp(Z)
plot(pca$x[,1],pca$x[,2],xlab="PC1",ylab="PC2",pch=pchsamples,col=colsamples)
legend("bottomright",legend = colsvsnames[,2],cex=0.55,col=cols,pch = pch.vec)
```




```
plot(1:dim(X)[2],pca$rotation[,1],type="l",xlab="genes",ylab="weight",main="First eigenvector")
```




```
##
##      BREAST         CNS       COLON K562A-repro K562B-repro    LEUKEMIA
##           7           5           7           1           1           6
## MCF7A-repro MCF7D-repro    MELANOMA       NSCLC     OVARIAN    PROSTATE
##           1           1           8           9           6           2
##       RENAL     UNKNOWN
##           9           1
```

The elements that are plotted in the plot titled "First eigenvectors" are the elements of the first principal loading vector (eigenvector) corresponding to each of the predictors, i.e. $\phi_{11}, \phi_{21}, ..., \phi_{p1}$. There are only $n = 64$ eigenvectors because there are only 64 data samples in the data set. This means that if one were to use more eigenvectors, one would increase the dimensionality of the data and thus remove the primary benefit of using principal component decomposition.

Q23: How many principal components are needed to explain 80% of the total variance in $\mathbf{Z}$? Why is `sum(pca$sdev^2)=p`?

```
# Since pca contains m=n=64 eigenvectors, they contain all the variance in the data.
# Therefore, one can just use iterate over the list of variance and return the index
# when the cumulative sum of variance is larger than 80% of the total variance.
find_n_elems_needed = function(list, frac){
  len = length(list)
  list = sort(list, decreasing = TRUE)

  sum_needed = frac * sum(list)
  cumulative_sum = 0
```

```
  for(i in 1:len){
    cumulative_sum = cumulative_sum + list[i]
    if(cumulative_sum >= sum_needed){
      return(i)
    }
  }
  return(-1)
}

pca_total_var = sum(pca$sdev^2)
pca_m_needed = find_n_elems_needed(pca$sdev^2, 0.8)
```

The number of eigenvectors needed in order to explain 80 % of the total variance in $\mathbf{Z}$ is: 32 `sum(pca$sdev^2)=p` because the data has been normalized, so that each column vector of $\mathbf{Z}$ has a standard deviation equal to 1, and because when the loading vectors $\mathbf{\Phi}_i$ were calculated they were constrained with the following constraint:

$$\sum_{j=1}^{p} \phi_{ji}^2 = 1 \tag{16}$$

This constraint makes it so that the loading vectors do not change the variance of the data. Therefore if the maximum number of eigenvectors $min(n, p)$ is utilized, the total variance (and thus the standard deviation) across all the principle components will be the same as the total variance in the original data, i.e. the sum of variance of each column $\mathbf{Z}_i$; $\sum_{i=1}^{p} Var(\mathbf{Z}_i) = p \cdot 1 = p$)

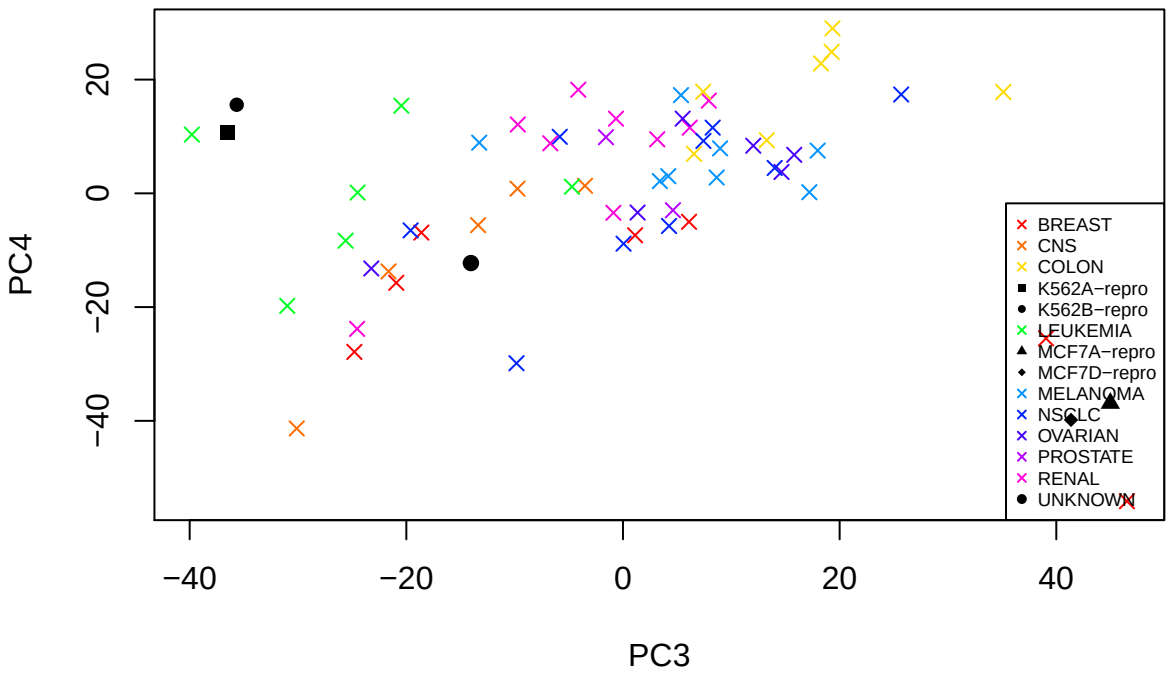
Q24: Study the PC1 vs PC2 plot, and comment on the groupings observed. What can you say about the placement of the `K262`, `MCF7` and `UNKNOWN` samples? Produce the same plot for two other pairs of PCs and comment on your observations.

I cannot see the `K262` anywhere in the plot, so I am assuming that there is an error in the task text and that it should say `K526` instead.

From the PC1 vs PC2 plot one can see that the sample classes are more spread in the direction of the first principal component than the second one, which was as expected because the data has higher variance along the direction of the first principal component. For the samples of `COLON`, `NSCLC` and `RENAL` one can see quite clear groupings along PC1, while they are indistinguishable along PC2. For the samples of `MELANOMA` and `NSCLC` the opposite is the case. They are clealy distinguishable along PC2, but indistinguishable along PC1. The `LEUKEMIA` samples are quite distinguishable both along PC1 and PC2.

For the two `K526` samples one can see that they are group close together, both along PC1 and PC2 and that they are clearly distinguishable from the two `MCF7` samples, both along PC1 and PC2. As for the two `MCF7` samples one can see that they are fairly close to each other, both along PC1 and PC2. Additionally one can see that they are indistinguishable from the `UNKNOWN` sample along PC2, but clearly distinguishable along PC1. However, the `ỦNKNOWN` sample is very hard to distinguish from a lot of the other sample groups.

```
plot(pca$x[,3],pca$x[,4],xlab="PC3",ylab="PC4",pch=pchsamples,col=colsamples)
legend("bottomright",legend = colsvsnames[,2],cex=0.55,col=cols,pch = pch.vec)
```



From the PC3 vs PC4 plot one can see that there is generally less overall spread and groupings of the samples, which was expected as the variance of the data is lower than along PC1 and PC2. The `LEUKEMIA`, `CNS` and `COLON` are the only clear groupings that stand apart.

As for the `K526`, `MCF7` and `UNKNOWN` samples, the `K526` samples are very close together along PC3 and PC4. The `MCF7` samples are also very closely grouped together both along PC3 and PC4. There is a very clear distinction between `K526` and `MCF7`, both along PC3 and PC4. The `UNKNOWN` sample is still relatively close to a large cluster of sample classes, but now seems rather distinguishable from the `K526` and `MCF7` samples, both along PC3 and PC4.

Q25:: Explain what it means to use Euclidean distance and average linkage for hierarchical clustering.

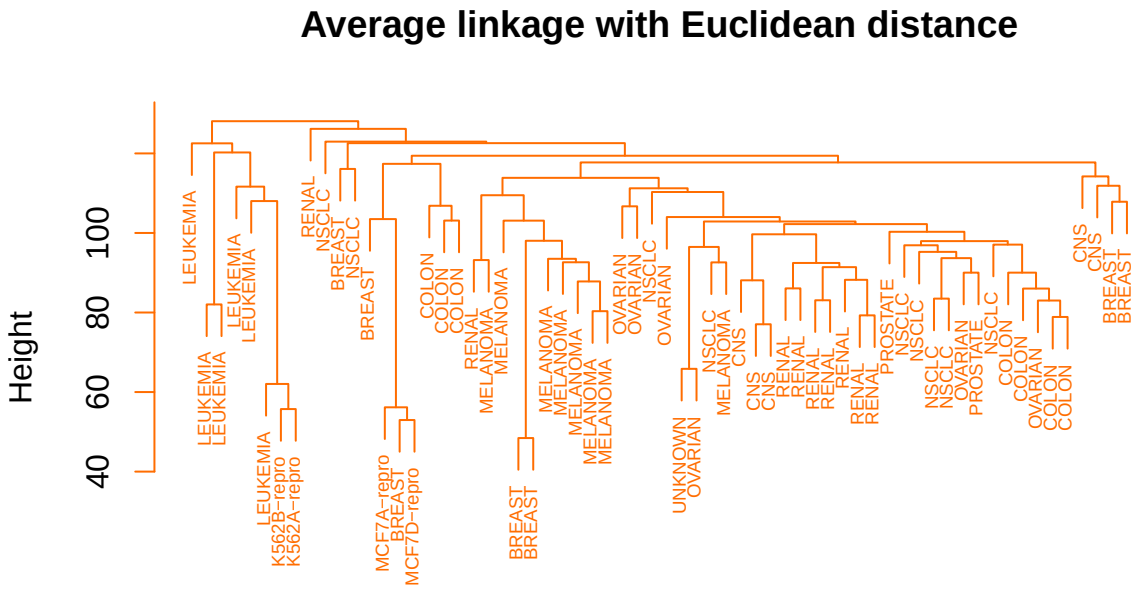
Euclidean distance measure is used as a measure of dissimilarity between pairs of observations. The dissimilarity measure between observations is needed when two observations are to be fused into a cluster. The greater the Euclidean distance between two observations, the more dissimilar they are. The Euclidean distance is defined as:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{k=1}^{p}(x_{ki} - x_{kj})^2} \tag{17}$$

Average linkage is a method of measuring dissimilarity between pairs of groups of observations. The linkage is needed when two clusters are to be fused into one cluster. The average linkage the mean intercluster dissimilarity is used as a dissimilarity measure. Consider two clusters, A and B. The mean intercluster dissimilarity is defined as the average of all the pairwise dissimilarities between the observations in cluster A and the observations in cluster B. The two clusters with the lowest mean intercluster dissimilarity are considered to be the most similar and are therefore the best candidates for fusing.

Q26:: Perform hierarchical clustering with Euclidean distance and average linkage on the scaled gene expression in `Z`. Observe where our samples labelled as K562, MCF7 and `UNKNOWN` are placed in the dendrogram. Which conclusions can you draw from this?

```
hc_euclid_avg = hclust(dist(Z[,-1], method = "euclidean"), method = "average")
plot(hc_euclid_avg,
     main="Average linkage with Euclidean distance",
     pch = pchsamples,
     col = colsamples,
     xlab="", sub="", cex=0.6)
```



From the dendrogram one can see that `K526`, `MCF7` and `UNKNOWN` are fused quite high. The cluster with `K526` is fused very late with the cluster that contains `MCF7` and `UNKNOWN`, which can indicate that it is quite dissimilar to them, or rather that there are other samples that are more similar. `MCF7` and `UNKNOWN` are also fused relatively high up in the dendrogram, which indicate that there are quite a lot of other samples that are more similar to `UNKNOWN` than what `MCF7` is.

Q27:: Study the R-code and plot below. Here we have performed hierarchical clustering based on the first 64 principal component instead of the gene expression data in `Z`. What is the difference between using all the gene expression data and using the first 64 principal components in the clustering? We have plotted the dendrogram together with a heatmap of the data. Explain what is shown in the heatmap. What is given on the horizontal axis, vertical axis, value in the pixel grid?

```
#library(pheatmap)
#npcs=64
#pheatmap(pca$x[,1:npcs],scale="none",cluster_col=FALSE,cluster_row=TRUE,clustering_distance_rows = #"e
```

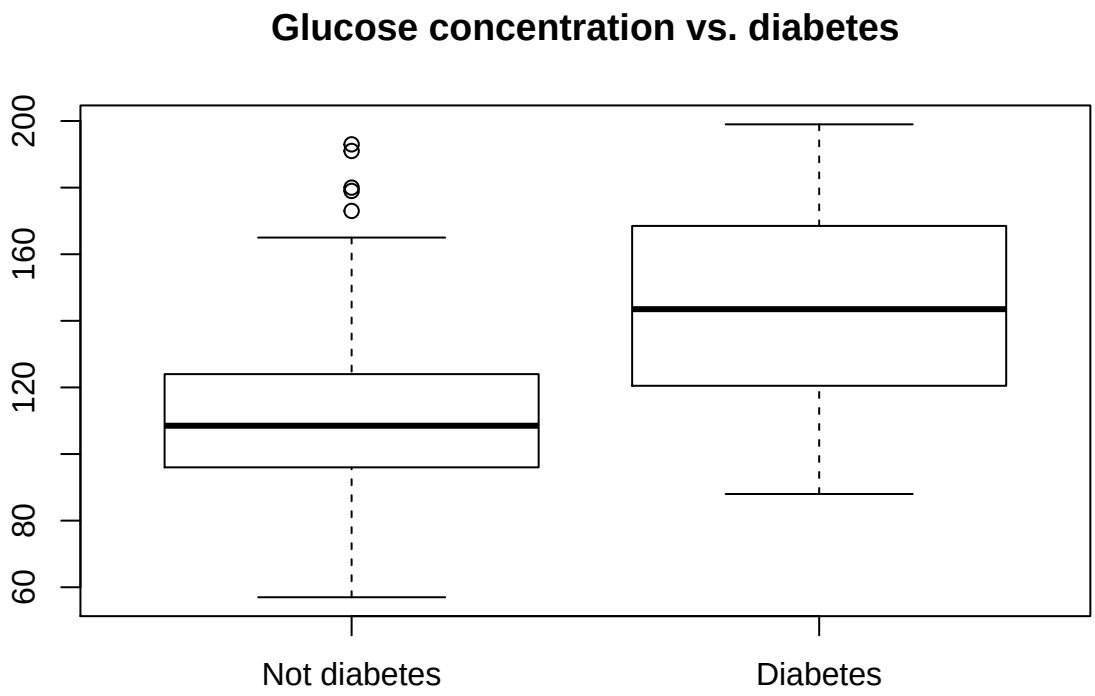
There is no difference between performing clustering using all the gene expression data and the first $n = m = 64$ principal component because the distance between individual observations remain the same. In the heatmap the different principal components are plotted along the x-axis, while the different cancer types are plotted along the y-axis. The value in the pixel grid are the principal component scores for each cancer type sample.

Problem 3: Flying solo with diabetes data [6 points]

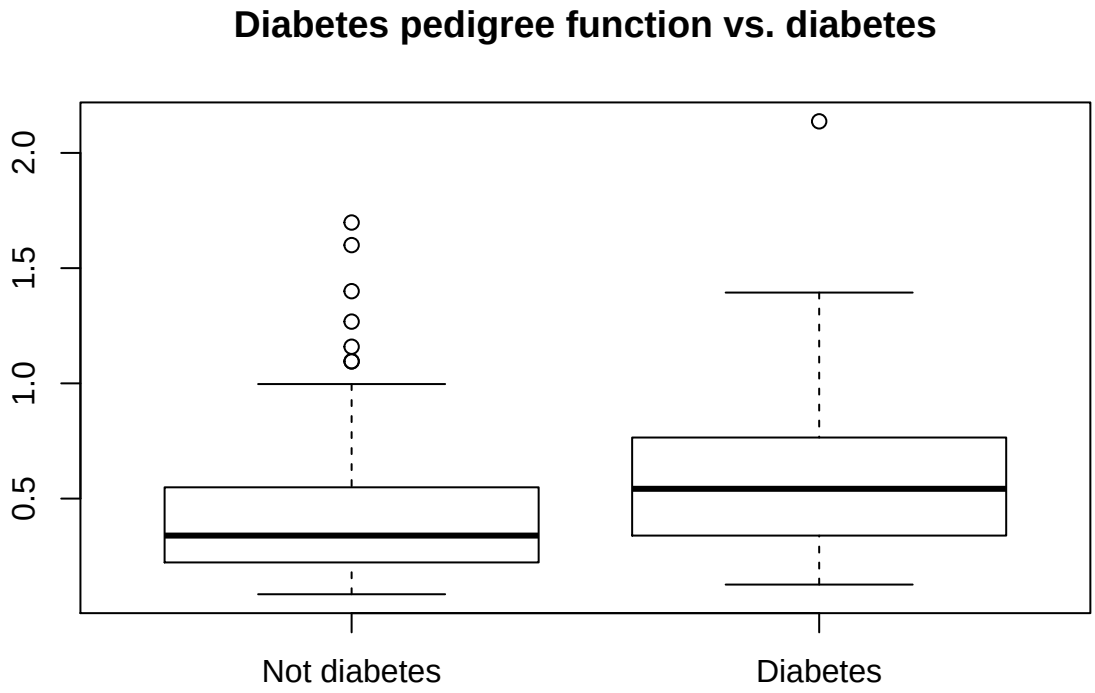
```
flying=dget("https://www.math.ntnu.no/emner/TMA4268/2019v/data/flying.dd")
diabetes_train=flying$ctrain
diabetes_test=flying$ctest

diabetes_p = ncol(diabetes_train)
diabetes_n_tr = nrow(diabetes_train)
diabetes_n_te = nrow(diabetes_test)

diabetes_train_fac = diabetes_train
diabetes_train_fac$diabetes = factor(diabetes_train_fac$diabetes, labels = c("0", "1"))

diabetes_test_fac = diabetes_test
diabetes_test_fac$diabetes = factor(diabetes_test_fac$diabetes, labels = c("0", "1"))

# Cutoff probability
diabetes_co_prob = 0.5
```

Q28: Start by getting to know the *training data*, by producing summaries and plots. Write a few sentences about what you observe and include your top 3 informative plots and/or outputs.

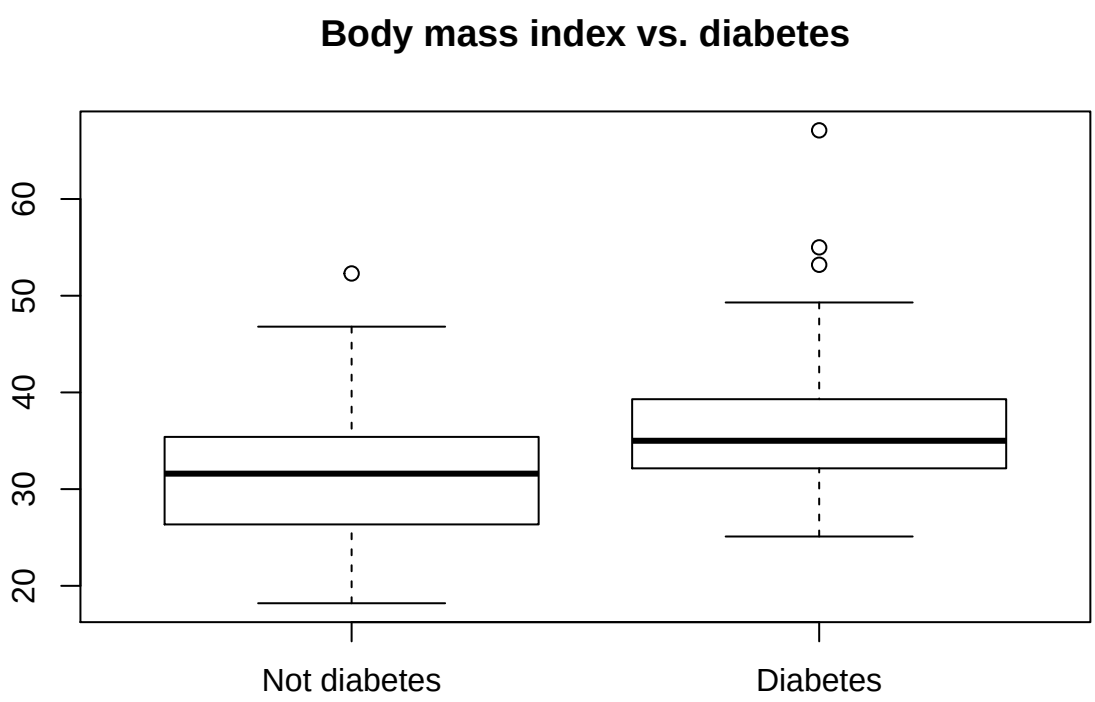
```
# Boxplot for glu
boxplot(formula = glu ~ diabetes, data = diabetes_train,
        main = "Glucose concentration vs. diabetes",
        names = c("Not diabetes", "Diabetes"))
```




```
# Boxplot for ped
boxplot(formula = ped ~ diabetes, data = diabetes_train,
        main = "Diabetes pedigree function vs. diabetes",
        names = c("Not diabetes", "Diabetes"))
```




```
# Boxplot for bmi
boxplot(formula = bmi ~ diabetes, data = diabetes_train,
        main = "Body mass index vs. diabetes",
        names = c("Not diabetes", "Diabetes"))
```



From the boxplot "Glucose concentration vs. diabetes" one can see that there is a distinct difference in the glucose concentration measured for people with diabetes. For people with diabetes the glucose concentrations have a much higher median, but much wider Q1 and Q3 quantiles, indicating that there is more variance in the `glu` predictor for people with diabetes. However the large difference in median suggests that `glu` is a strong predictor for the presence of `diabetes`.

From the boxplot "Diabetes pedigree function vs. diabetes" one can see that the median of `ped` is slightly higher in cases where `diabetes` is present. The difference is not as distinctive as that for `glu`, but it is still visible. One can also see that the Q4 quantile is much wider than the Q1 quantile, indicating that there is larger variance among the measurements in the upper end of the spectra. However it seems that `ped` can be a decent predictor for predicting the presence of `diabetes`, but not as strong as `glu`.

From the boxplot "Body mass index vs. diabetes" one can, like with `glu` and `ped`, see a distinctive difference in `bmi` for samples where `diabetes` is present and not. The median of `bmi` is slightly higher and the variance lower when `diabetes` is present. This indicates that `bmi` could be a good predictor for predicting the presence of `diabetes`.

Q29: Use different methods to analyse the data. In particular use

- one method from Module 4: Classification
- one method from Module 8: Trees (and forests)
- one method from Module 9: Support vector machines and, finally
- one method from Module 11: Neural networks

For each method you

- clearly write out the model and model assumptions for the method

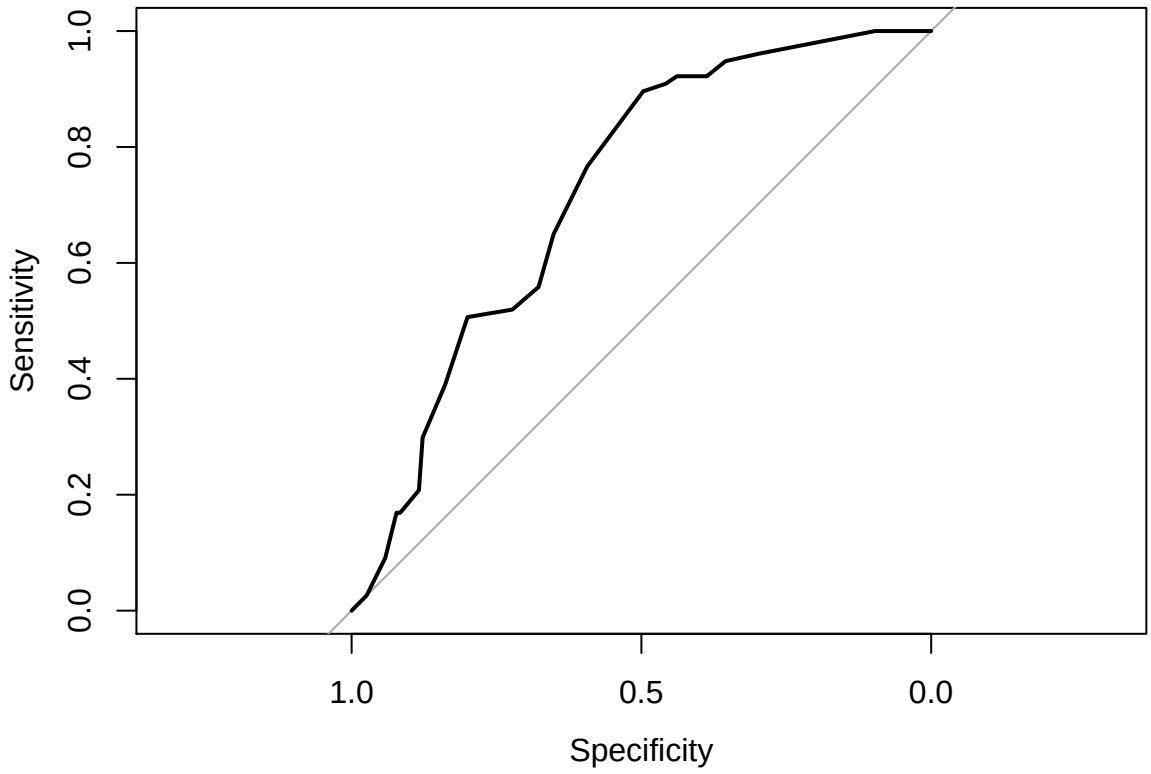
- explain how any tuning parameters are chosen or model selection is performed
- report (any) insight into the interpretation of the fitted model
- evaluate the model using the test data, and report misclassifiation rate (cut-off 0.5 on probability) and plot ROC-curves and give the AUC (for method where class probabilities are given).

Method from Module 4: Classification

```
library(class)
library(caret)

# Setting seed for reproducability
set.seed(0)

# The different values of k to be considered
ks = 1:100

# The number of folds
n_folds = 5

# Using k-fold cross-validation to do model selection
diabetes_fold_covariates = subset(diabetes_train_fac, select = c(-diabetes))
diabetes_fold_labels = subset(diabetes_train_fac, select = c(diabetes))

# Creating folds
valid_idxs = 1:nrow(diabetes_fold_labels)

folds = createFolds(valid_idxs, k = n_folds)

cv = sapply(ks, function(k){
  sapply(seq_along(folds), function(j) {
    yhat = knn(train = diabetes_fold_covariates[ -folds[[j]], ],
               cl = diabetes_fold_labels[ -folds[[j]], ],
               test = diabetes_fold_covariates[ folds[[j]], ],
               k = k)
    mean(diabetes_fold_labels[ folds[[j]], ] != yhat)
  })
})
```

```
library(caret)
library(pROC)
# Picking the K that minimizes the cross-validation error
cv_error = colMeans(cv)
cv.standard_error = (1/sqrt(n_folds)) * apply(cv, 2, sd)
knn_cv_best_k = which.min(cv_error)


diabetes_knn_pred = knn(train = subset(diabetes_train_fac, select = c(-diabetes))[,],
                        cl = subset(diabetes_train_fac, select = c(diabetes))[,],
                        test = subset(diabetes_test_fac, select = c(-diabetes))[,],
                        k = knn_cv_best_k,
                        prob = TRUE)
```

```r
# Calculating test misclassification rate
diabetes_knn_test_mr = mean(diabetes_test$diabetes != as.numeric(as.character(diabetes_knn_pred)))

# Calculating ROC and AUC
diabetes_knn_roc = roc(response = diabetes_test$diabetes,
                       predictor = attr(diabetes_knn_pred, "prob"))
diabetes_knn_auc = diabetes_knn_roc$auc
plot(diabetes_knn_roc)
```



The method from module 4 that I choose to use was the KNN-classifier. The KNN-classifier does not assume anything about the underlying model of the data. Therefore it does not necessarily give that much insight into the data.

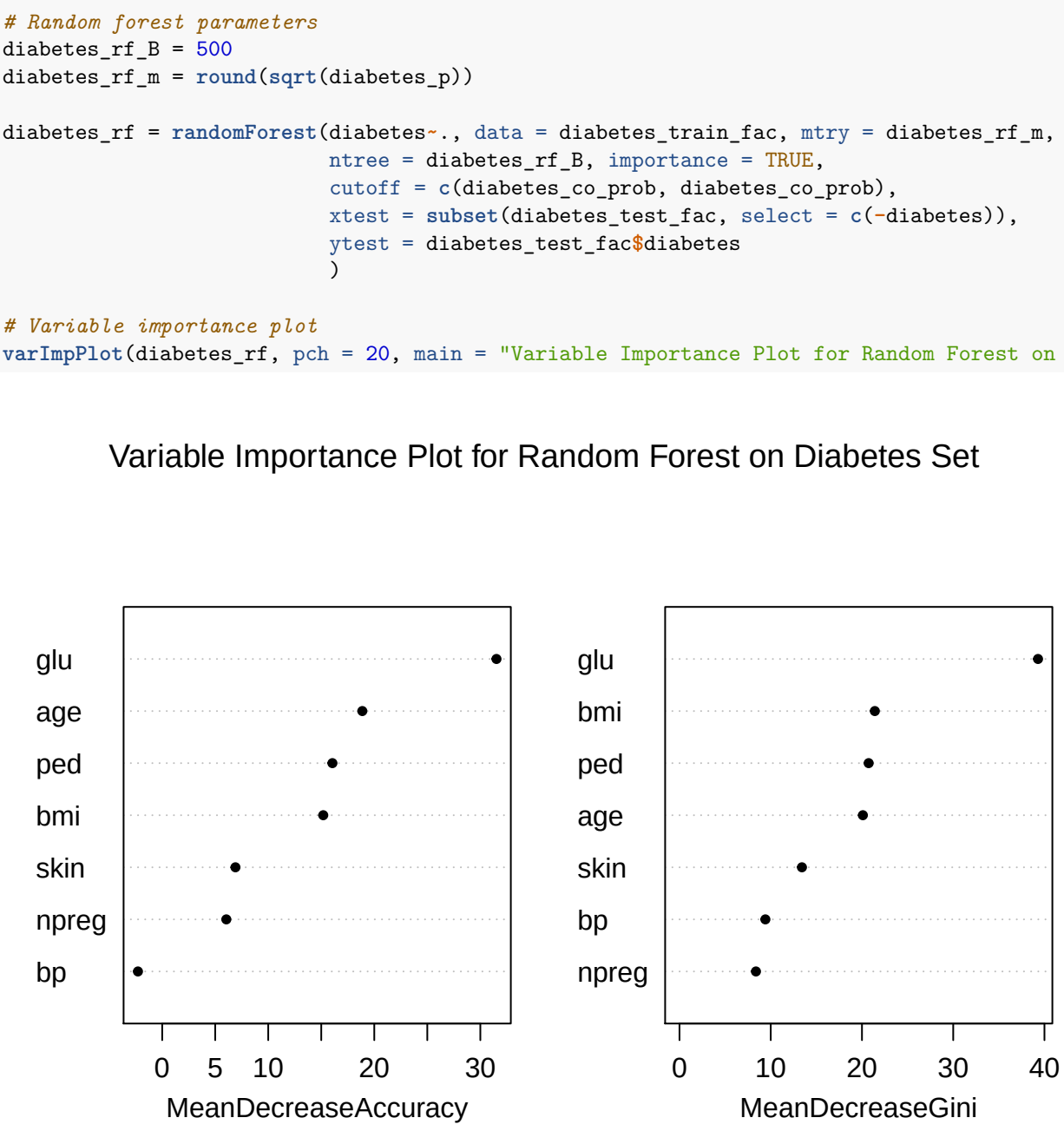
The only tuning parameter for the KNN-classifier is the number of neighbours that it considers K. In order to do model selection, i.e. selecting the parameter K, I have chosen to perform k-fold cross-validation over $k = 5$ folds for K-values ranging from 1 to 100. Then the value of K that yielded the minimum 0/1 cross validation loss was picked for the final model. The final value for K was found to be 41.

AUC for the KNN-classifier: 0.7241

Misclassification rate on the test set for the KNN-classifier: 0.2198

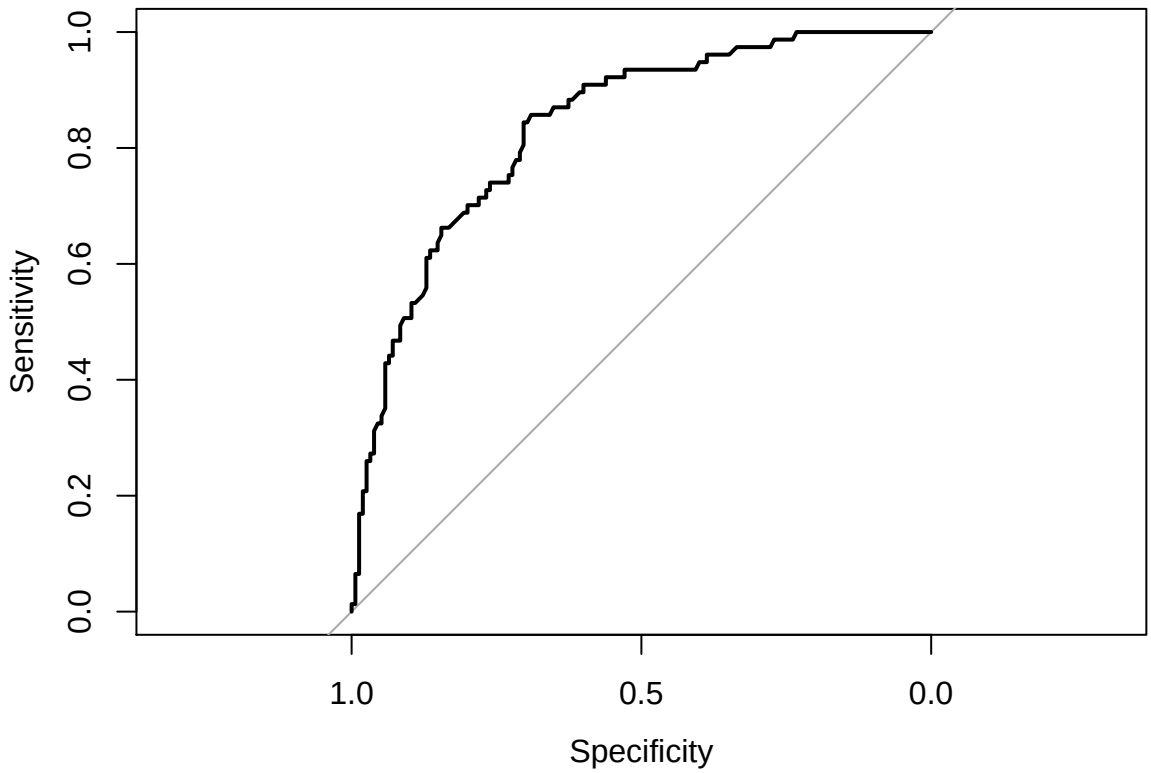
Method from Module 8: Trees

```r
library(randomForest)
library(caret)
library(pROC)

# Random forest parameters
diabetes_rf_B = 500
diabetes_rf_m = round(sqrt(diabetes_p))

diabetes_rf = randomForest(diabetes~., data = diabetes_train_fac, mtry = diabetes_rf_m,
                           ntree = diabetes_rf_B, importance = TRUE,
                           cutoff = c(diabetes_co_prob, diabetes_co_prob),
                           xtest = subset(diabetes_test_fac, select = c(-diabetes)),
                           ytest = diabetes_test_fac$diabetes
                           )

# Variable importance plot
varImpPlot(diabetes_rf, pch = 20, main = "Variable Importance Plot for Random Forest on Diabetes Set")
```

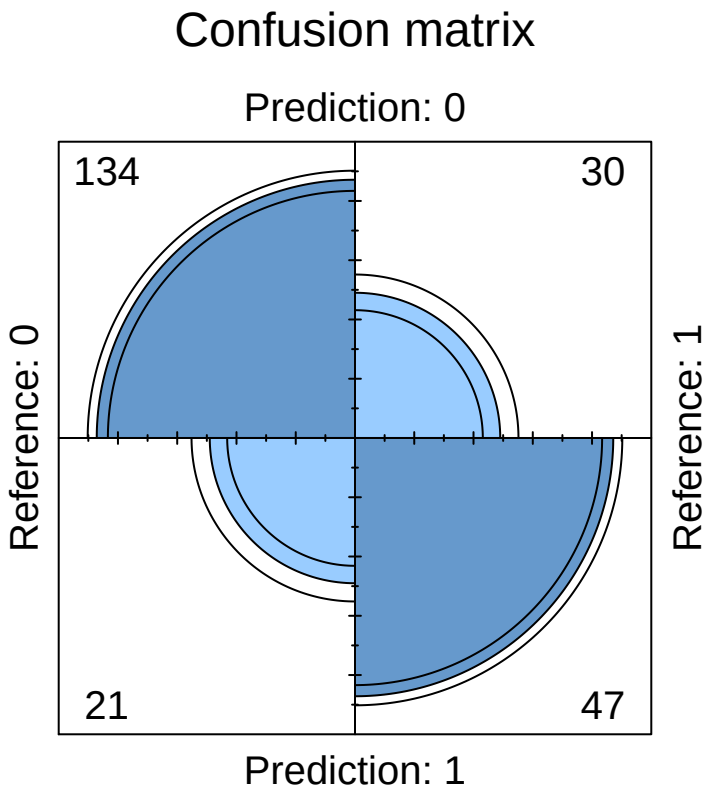



```r
# ROC and AUC
# The proportion of votes for the classes are equal to the predicted probability
# of the classes. Since this is a binary case and the classes are hot-one encoded
# one can simply compare the
diabetes_rf_roc = roc(response = diabetes_test$diabetes,
                      predictor = diabetes_rf$test$votes[,2])
```

```r
diabetes_rf_auc = diabetes_rf_roc$auc
plot(diabetes_rf_roc)
```




```r
# Confusion matrix
diabetes_rf_cm = confusionMatrix(diabetes_rf$test$predicted, diabetes_test_fac$diabetes)
fourfoldplot(diabetes_rf_cm$table, main = "Confusion matrix")
```




```
diabetes_rf_test_acc = diabetes_rf_cm$overall[1]
diabetes_rf_test_mr = 1 - diabetes_rf_test_acc
```

The number of trees in the random forest B is not tuned, but is set large, i.e. $B = 500$. The number of predictors that are used for splitting m is set according to the following rule for classification:

$$m \approx \sqrt{p} \tag{18}$$

From the variable importance plot for the random forest on the diabetes set one can see that the mean decrease accuracy is affected most by `glu`, `ped`, `age` and `bmi`. This goes along with the initial inspection of the data set and supports the observations that was made then.

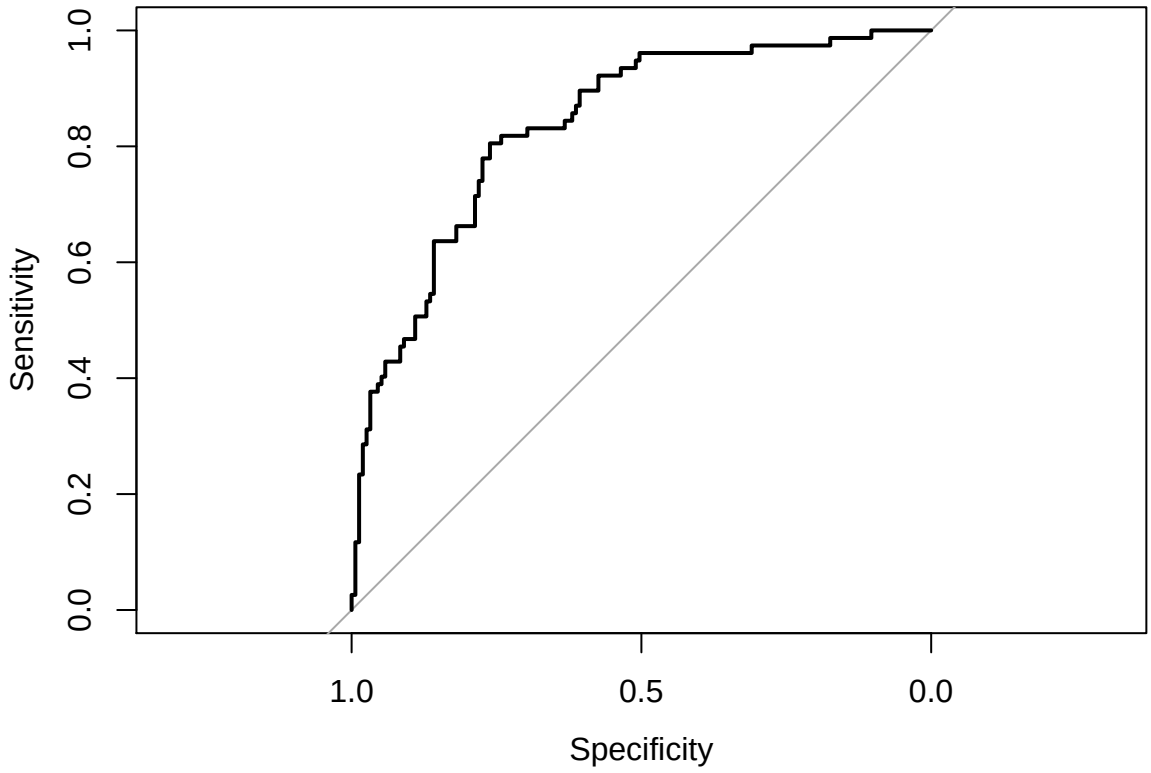
AUC for the random forest: 0.8365

Misclassification rate on the test set for the random forest: 0.2198

Method from Module 9: Support vector machines

```
library(e1071)

svm_kernel = "radial"

# Using cross validation to select the best cost parameter
diabetes_svm_cv = tune(svm, diabetes ~ ., data = diabetes_train, kernel = svm_kernel)
```

```r
diabetes_best_svm = diabetes_svm_cv$best.model

# Calculating predicted classes and their respective probabilities
diabetes_svm_pred_probs = predict(diabetes_best_svm, diabetes_test)
diabetes_svm_pred_classes = as.numeric(diabetes_svm_pred_probs > diabetes_co_prob)

# Calculating misclassification rate on the test set
diabetes_svm_test_mr = mean(diabetes_test$diabetes != diabetes_svm_pred_classes)

# Calculating ROC and AUC
diabetes_svm_roc = roc(response = diabetes_test$diabetes,
                       predictor = diabetes_svm_pred_probs)
diabetes_svm_auc = diabetes_svm_roc$auc
plot(diabetes_svm_roc)
```



From module 9 I have chosen to use a support vector machine with a radial kernel. Support vector machines do not offer much interpretability in terms of the underlying model of the data, hence it is hard to do any inference from the fitted model. The tuning parameters for a support vector machine with a radial kernel is the cost C and the kernel parameter γ. To do model selection, i.e. select the values for C and γ, I have utilized k-fold cross-validation with $k = 10$ folds. The following parameter values were found from performing cross-validation:

C: 1

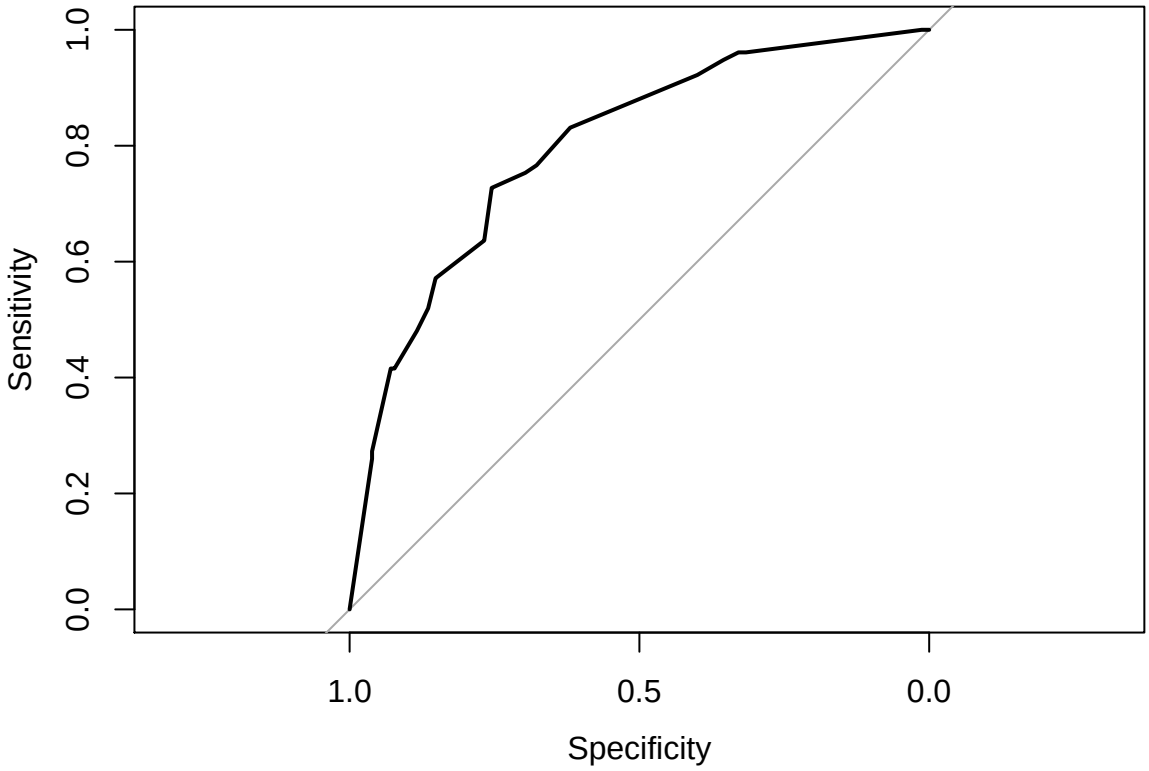
γ: 0.1429

The AUC for the SVM is: 0.8358

The misclassification rate of the SVM on the test set is: 0.25

Method from Module 11: Neural networks

```
library(nnet)

diabetes_train_data = subset(diabetes_train, select = c(-diabetes))
diabetes_train_labels = subset(diabetes_train, select = c(diabetes))
diabetes_test_data = subset(diabetes_test, select = c(-diabetes))
diabetes_test_labels = subset(diabetes_test, select = c(diabetes))

mean = apply(diabetes_train_data, 2, mean)
std = apply(diabetes_train_data, 2, sd)

diabetes_scaled_train_data = scale(diabetes_train_data, center = mean, scale = std)
diabetes_scaled_test_data = scale(diabetes_test_data, center = mean, scale = std)

diabetes_nnet = nnet(x = diabetes_scaled_train_data, y = diabetes_train_labels,
                     size=5, entropy = TRUE, maxit = 1000)

diabetes_nnet_pred = predict(diabetes_nnet, diabetes_scaled_test_data)

diabetes_nnet_pred_probs = diabetes_nnet_pred
diabetes_nnet_pred_classes = as.numeric(diabetes_nnet_pred_probs > diabetes_co_prob)
diabetes_nnet_test_mr = mean(diabetes_test_labels$diabetes != diabetes_nnet_pred_classes)

diabetes_nnet_roc = roc(response = diabetes_test_labels$diabetes,
                        predictor = diabetes_nnet_pred_probs)
diabetes_nnet_auc = diabetes_nnet_roc$auc
plot(diabetes_nnet_roc)
```




```
## # weights:  46
## initial  value 186.475826
## iter  10 value 120.922998
## iter  20 value 104.638621
## iter  30 value 93.590047
## iter  40 value 85.782183
## iter  50 value 84.121409
## iter  60 value 83.669464
## iter  70 value 83.649143
## final  value 83.647628
## converged
```

From module 11 I have chosen to use a regular feedforward neural network with one hidden layer with 5 nodes. The model assumption of a single layered feedforward neural network with one output node is that the response for each observation $\mathbf{x}_i$ is:

$$\hat{y}_i(\mathbf{x}_i) = \phi_0(w_0 + w_1 x_{i1} + \ldots + w_p x_{ip}) \tag{19}$$

Here ϕ_0 is an activation function, that is a non-linear function, w_0 is the bias of the output node and $w_j x_{ij}$ are the weighted activations from the previous layer. The choice of activation function ϕ_0 depends on the layer of the network (for multilayered neural networks) and whether the network is used for regression or classification. For binary class classification the sigmoid function defined as follows is popular:

$$\phi_{\text{sigmoid}} = \frac{1}{1 + e^{(w_0 + w_1 x_{i1} + \ldots + w_p x_{ip})}} \tag{20}$$

The only parameters that I have changed are the number of nodes in the hidden layer. They were chosen based on trial and error.

The AUC for the neural network is: 0.7982

The misclassification of the neural network on the test set is: 0.2414

Q30: Conclude with choosing a winning method, and explain why you mean that this is the winning method.

The winning method that I am choosing is the random forest. This is because the random forest does allow for some inference about the data through the variable importance plot. Additionally it has the high AUC at the same time it has a misclassification rate that is lower than the SVM and the neural network and comparable to the KNN-classifier.