EBC1-3

Albiyan Aldo Natapraja (596621); Soopin Oh (604440);
Dyon Pacheco (465935); Maximiliaan Stenfert (472291)

06/02/2022

Assignment 1 *Part 1*: Split the sample dataset into a training dataset (80%) and a testing dataset (20%).

```
#split data into train (80%) & test (20%)

set.seed(123)

data_split <- initial_split(BostonHousing2,prop=4/5)
train_set <- training(data_split)
test_set <- testing(data_split)
```

Part 2: Estimate three linear OLS regression models that explain cmedv, the corrected median value of owner-occupied homes in a given census tract, using the training data.

```
#set the regession model
lm_model <- linear_reg() %>%
            set_engine('lm') %>%
            set_mode('regression')

#first with the training set
Model1 <- lm_model %>% fit(cmedv~crim+nox+age+lstat, data=train_set)

Model2 <- lm_model %>% fit(cmedv~crim+nox+age+lstat+rm+chas+ptratio+dis+rad+tax,
                           data=train_set)

Model3 <- lm_model %>% fit(cmedv~crim+crim^2+nox+nox^2+age+age^2+lstat+lstat^2+rm+rm^2+chas+
    ptratio+ptratio^2+dis+dis^2+dis*lstat+rad+rad^2+tax+tax^2, data=train_set)
```

Predict cmedv for all three models, first for the training data itself, then for the testing data

```
#run the forecast for train_set first
Forecast1 <- predict(Model1, train_set)
colnames(Forecast1) <- c("Forecast1")

Forecast2 <- predict(Model2, train_set)
colnames(Forecast2) <- c("Forecast2")

Forecast3 <- predict(Model3, train_set)
colnames(Forecast3) <- c("Forecast3")

Actual_train <- select(train_set, "cmedv")

#this shows the data frame of predicted values from models with train data.
df_train <- cbind(Actual_train, Forecast1, Forecast2, Forecast3)
```

```
colnames(df_train)[1] <- c("Actual")
```

```
#now, repeat the predicted values with test set
Forecast4 <- predict(Model1, test_set)
colnames(Forecast4) <- c("Forecast4")

Forecast5 <- predict(Model2, test_set)
colnames(Forecast5) <- c("Forecast5")

Forecast6 <- predict(Model3, test_set)
colnames(Forecast6) <- c("Forecast6")

Actual_test <- select(test_set, "cmedv")

#this shows the predicted values of three models with test data.
df_test <- cbind(Actual_test, Forecast4, Forecast5, Forecast6)
colnames(df_test)[1] <- c("Actual")
```

Part 3: Plot the actual values against the predicted values of cmedv for all three models, both for the training and testing data.

```
melt_train <- melt(df_train, id.vars="Actual", variable.name="model")

ggplot(melt_train) +
  geom_point(aes(Actual, value, colour = model), size=1) +
  geom_abline(intercept=0, slope=1) + facet_wrap(~model) +
  xlab("Actual cmedv value") + ylab("Predicted cmedv value") +
  labs(title = "Forecast model with training data") +
  scale_color_discrete(name = "Model", labels = c("Model 1", "Model 2", "Model 3"))
```


Forecast model with training data
Forecast1
Forecast2
Forecast3
Predicted cmedv value
Actual cmedv value
0
10
20
30
40
50
Model
Model 1
Model 2
Model 3

The black line across the graph shows the regression line. The further the predicted points are from the line, the further it is from the actual value. The codes for test data are the same as training data, they are not shown for better presentation in pdf. As can be seen from the two sets of graphs (with training vs. test data), the models are shown to predict similarly.

Part 4: Calculate RMSE, MAE, and MAPE for the three models, both for the training and testing data. Which model provides the most accurate predictions? Are there noticeable differences between the within-sample and out-of-sample performance that suggest overfitting?

```
#now we measure the rmse, mae and mape for training set first.
#first, calculate the root mean square error for each forecast model.
rmse1 <- rmse(df_train, Actual, Forecast1, na_rm=TRUE)
rmse2 <- rmse(df_train, Actual, Forecast2, na_rm=TRUE)
rmse3 <- rmse(df_train, Actual, Forecast3, na_rm=TRUE)
#combine all the rmse values into one tibble.
rmse_tr <- rbind(rmse1, rmse2, rmse3)
rmse_tr <- select(rmse_tr, .estimate)

#then, calculate the mean absolute error for all.
mae1 <- mae(df_train, Actual, Forecast1, na_rm=TRUE)
mae2 <- mae(df_train, Actual, Forecast2, na_rm=TRUE)
mae3 <- mae(df_train, Actual, Forecast3, na_rm=TRUE)
mae_tr <- rbind(mae1, mae2, mae3)
mae_tr <- select(mae_tr, .estimate)

#calculate the mean absolute percentage.
mape1 <- mape(df_train, Actual, Forecast1, na_rm=TRUE)
```

```
mape2 <- mape(df_train, Actual, Forecast2, na_rm=TRUE)
mape3 <- mape(df_train, Actual, Forecast3, na_rm=TRUE)
mape_tr <- rbind(mape1, mape2, mape3)
mape_tr <- select(mape_tr, .estimate)

model <- tibble(Model=1:3)

#combine all the estimates of metrics into one table for all models.
metrics_train <- cbind(model, rmse_tr, mae_tr, mape_tr)
names(metrics_train)[2] <- "RMSE"
names(metrics_train)[3] <- "MAE"
names(metrics_train)[4] <- "MAPE"
```

For training set:

```
##   Model     RMSE      MAE     MAPE
## 1     1 5.992481 4.334368 20.75314
## 2     2 4.692123 3.306053 16.86228
## 3     3 4.667253 3.288865 16.69975
```

For training data, Model 1 has the highest RMSE, MAE and MAPE, which means that it is not the best model to predict the value of cmedv. Model 3 is the best model that predicts the cmedv value, as it has the lowest RMSE, MAE and MAPE. This means that the predicted values of model 3 would have closest to the actual values generally.

For test set: (ran the same way as previously, codes are hidden for better presentation)

```
##   Model     RMSE      MAE     MAPE
## 1     1 6.381678 4.844224 22.79922
## 2     2 5.039124 3.836126 18.92912
## 3     3 5.014147 3.797786 18.58756
```

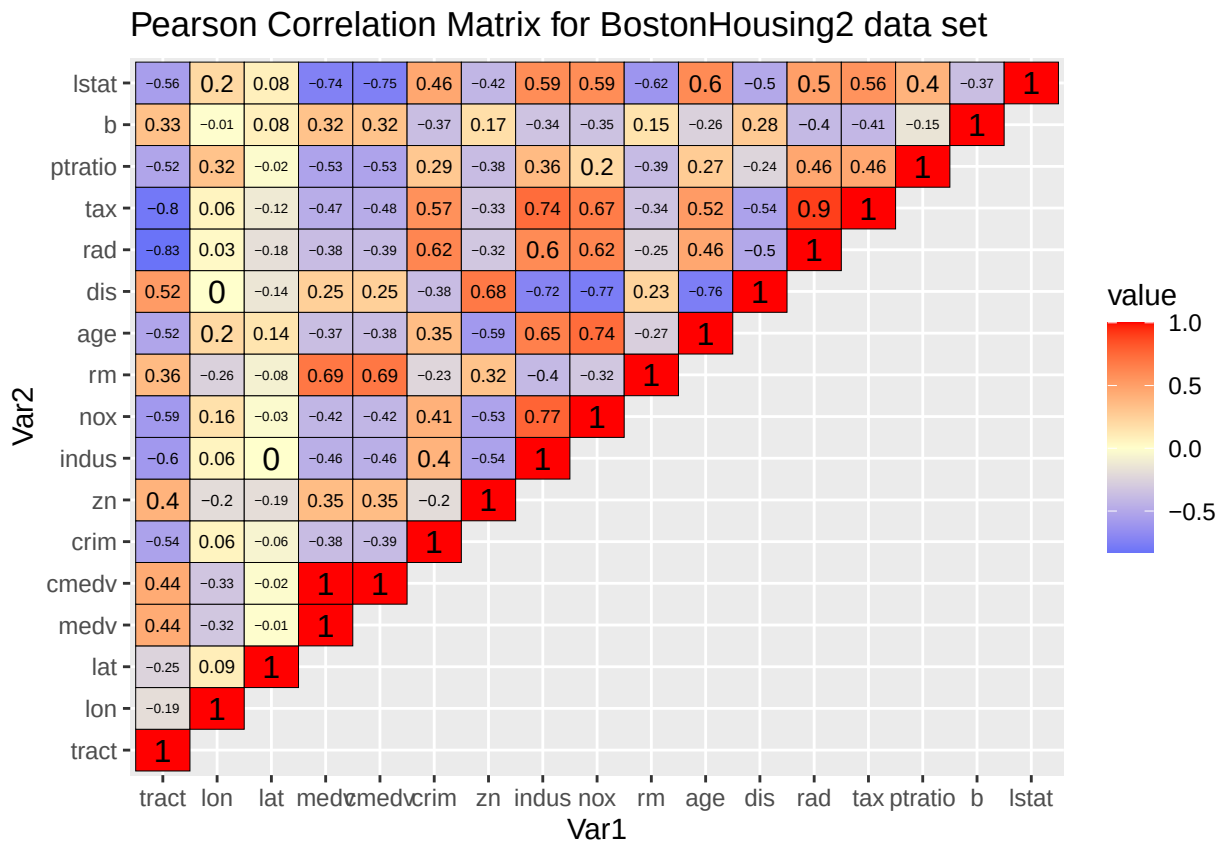
For test data, the best model is also Model 3, as it has the lowest values for all RMSE, MAE and MAPE measurements.

The RMSE, MAE and MAPE values are generally higher for the forecasts run with test data, compared to the forecasts run with training data. The RMSE, MAE and MAPE shows that the models are better fit when using the within-sample (training data), instead of out-of-sample (test data). There is a slight chance of over-fitting however not so much as the differences of RMSE, MAE and MAPE between within- and out-of-sample is not so extreme.

Part 5: Can you come up yourself with an alternative model that generates better predictions than the models above?

```
#first, we want to look at the correlations for the variables to each other by using correlation matrix
#choose the numeric columns in the data set
train_set_num <- select_if(train_set, is.numeric)

#make the correlation matrix
cormat <- round(cor(train_set_num),2)

#function to only get the half of correlation for better look
get_upper_tri <- function(cormat){
    cormat[lower.tri(cormat)]<- NA
    return(cormat)
  }
upper_tri <- get_upper_tri(cormat)
melted_cormat <- melt(upper_tri, na.rm = TRUE)
```

```
ggplot(melted_cormat, aes(x=Var1, y=Var2, fill=value, label=value)) +
  geom_tile(color = "black") +
  scale_fill_gradient2(low = "#075AFF",
                       mid = "#FFFFCC",
                       high = "#FF0000") +
  geom_fit_text(color="black") +
  labs(title = "Pearson Correlation Matrix for BostonHousing2 data set")
```



From the heatmap, we see that variable cmedv does not have strong correlation with most of the other variables. cmedv, the outcome variable, is the corrected version of medv, thus we will not look at medv as our predictor. "lstat", "rm" and "ptratio" are between 0.5-1.0 (absolute value) in correlation values, which would be added into our model to predict cmedv.

what the variables mean:

-RM: Average number of rooms per dwelling

-LSTAT: Percentage of lower status of the population

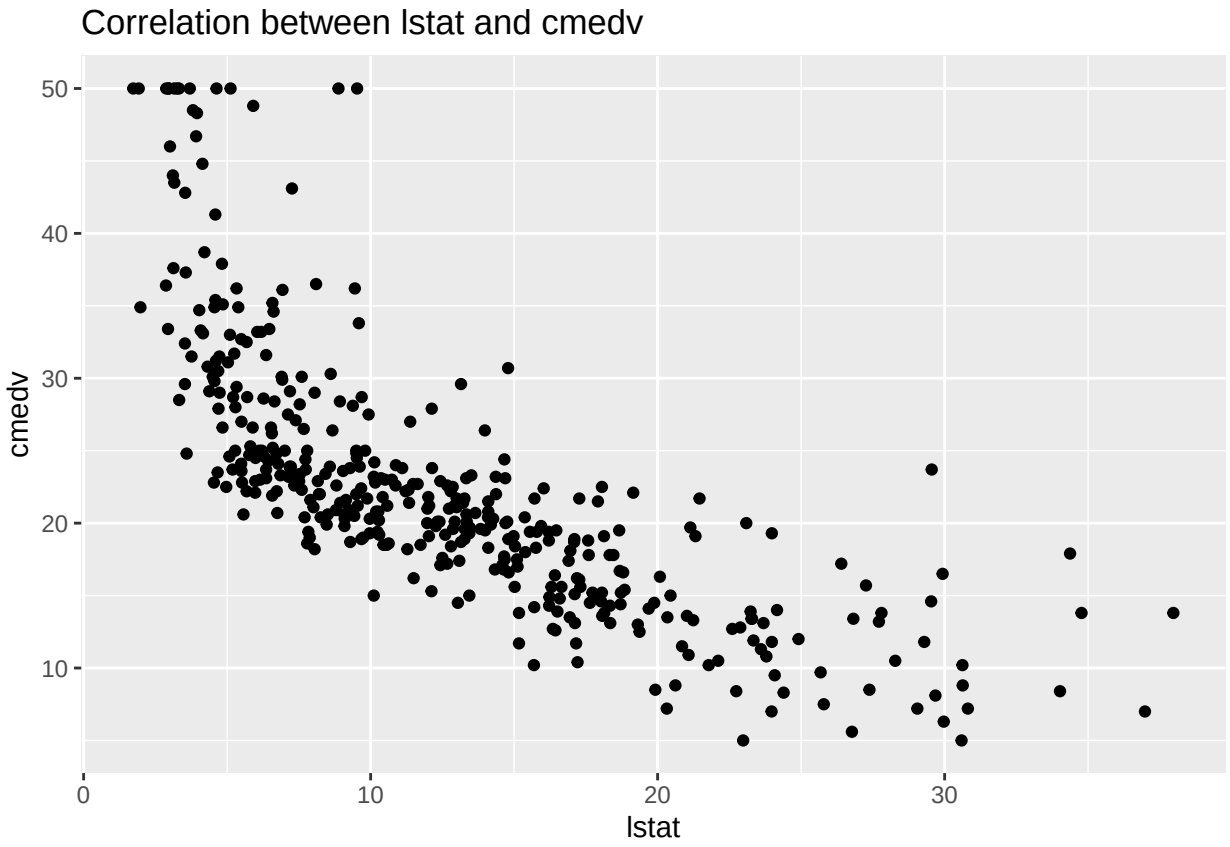
-PTRATIO: pupil-teacher ratio by town

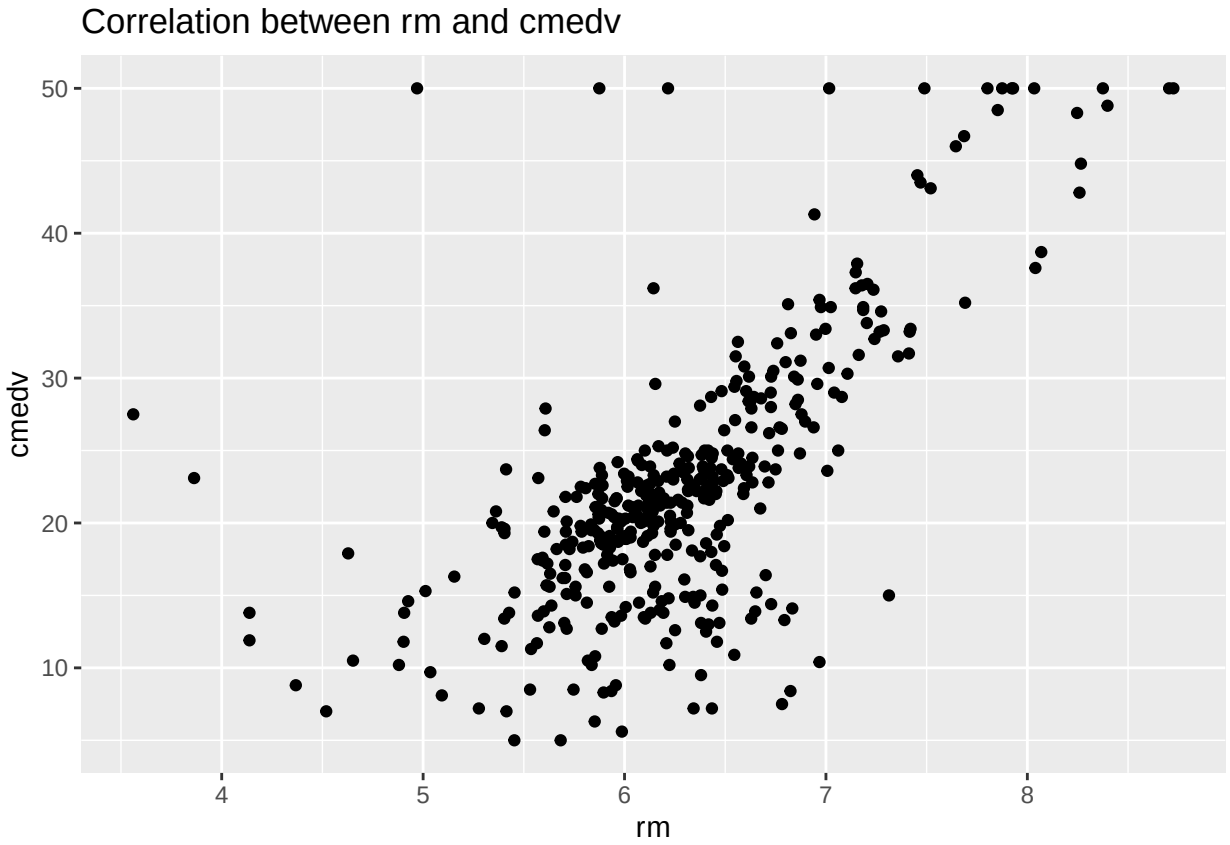
-CMEDV: corrected median value of owner-occupied homes in USD 1000's

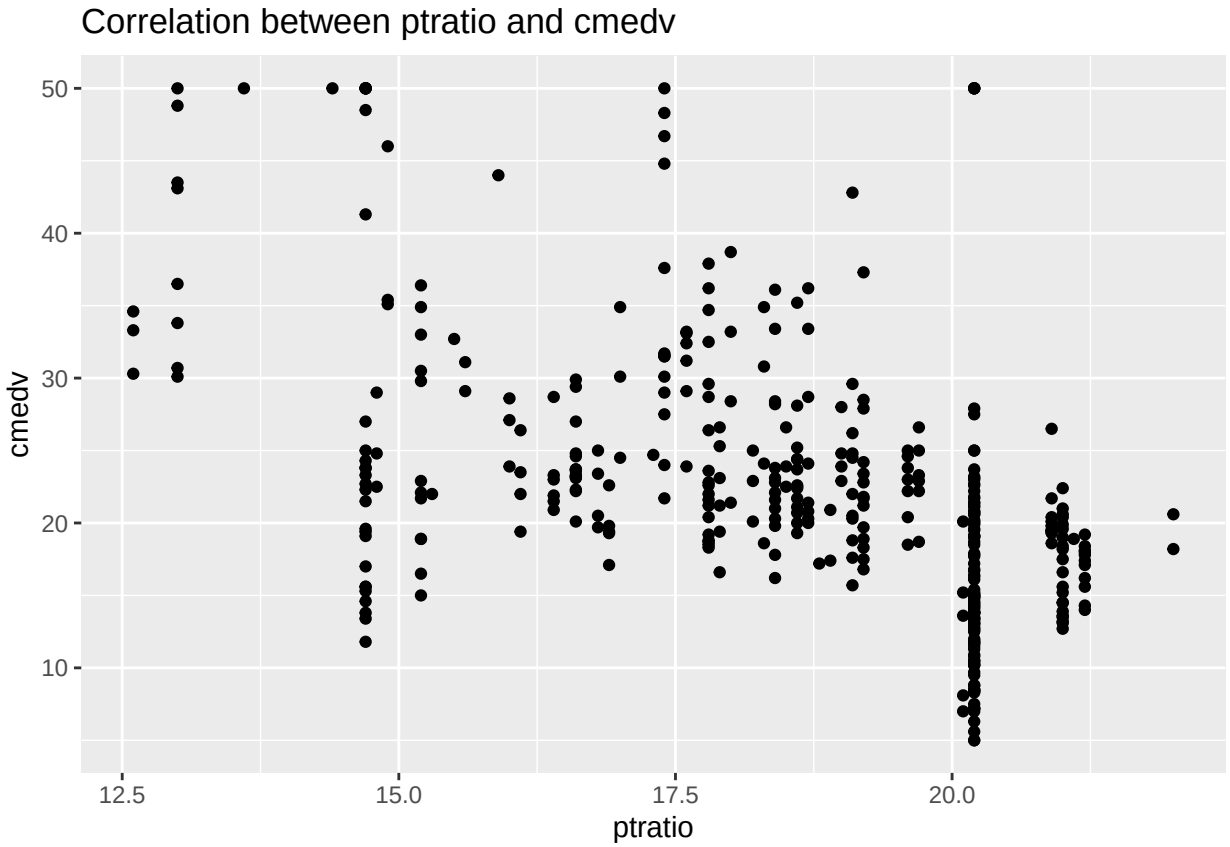
Random Forest models are immune to outliers, which is present in our data, and they completely ignore statistical issues because unlike other machine learning models which perform much better after being normalized.

```
gg_lstat <- ggplot(data=train_set) + geom_point(aes(x=lstat, y=cmedv)) +
  labs(title = "Correlation between lstat and cmedv")
```

```
gg_rm <- ggplot(data=train_set) + geom_point(aes(x=rm, y=cmedv)) +
  labs(title = "Correlation between rm and cmedv")
gg_ptratio <- ggplot(data=train_set) + geom_point(aes(x=ptratio, y=cmedv)) +
  labs(title = "Correlation between ptratio and cmedv")
```

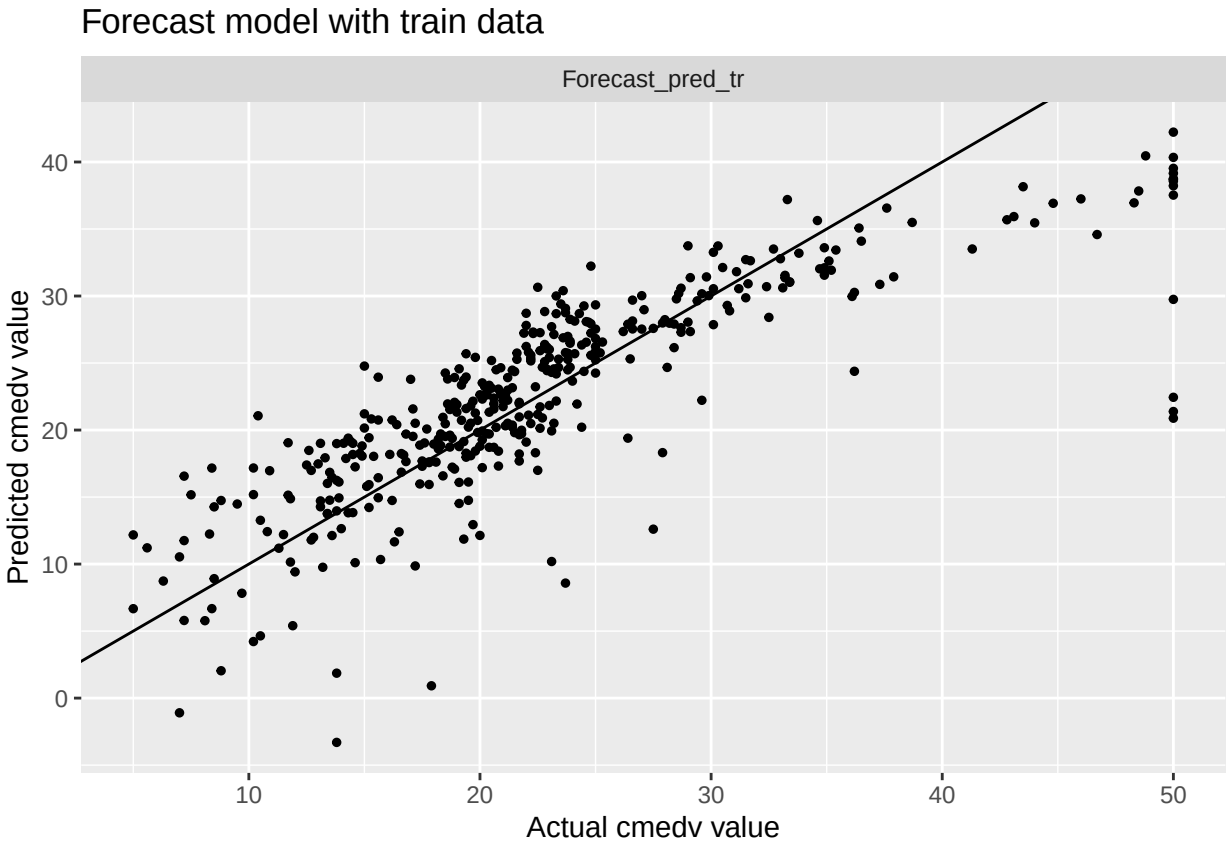



Correlation between rm and cmedv
cmedv
rm
50
40
30
20
10
4
5
6
7
8



For our own prediction model, lstat, rm and ptratio are run through linear regression.

```
Model_pred <- lm_model %>% fit(cmedv~lstat+rm+ptratio, data=train_set)

Forecast_pred_tr <- predict(Model_pred, train_set)
colnames(Forecast_pred_tr) <- c("Forecast_pred_tr")
Actual_test_tr <- select(train_set, "cmedv")
df_pred_tr <- cbind(Actual_test_tr, Forecast_pred_tr)
colnames(df_pred_tr)[1] <- c("Actual")

melt_pred_tr <- melt(df_pred_tr, id.vars="Actual", variable.name="model")

ggplot(melt_pred_tr) +
  geom_point(aes(Actual, value), size=1) +
  geom_abline(intercept=0, slope=1) + facet_wrap(~model) +
  xlab("Actual cmedv value") + ylab("Predicted cmedv value") +
  labs(title = "Forecast model with train data")
```



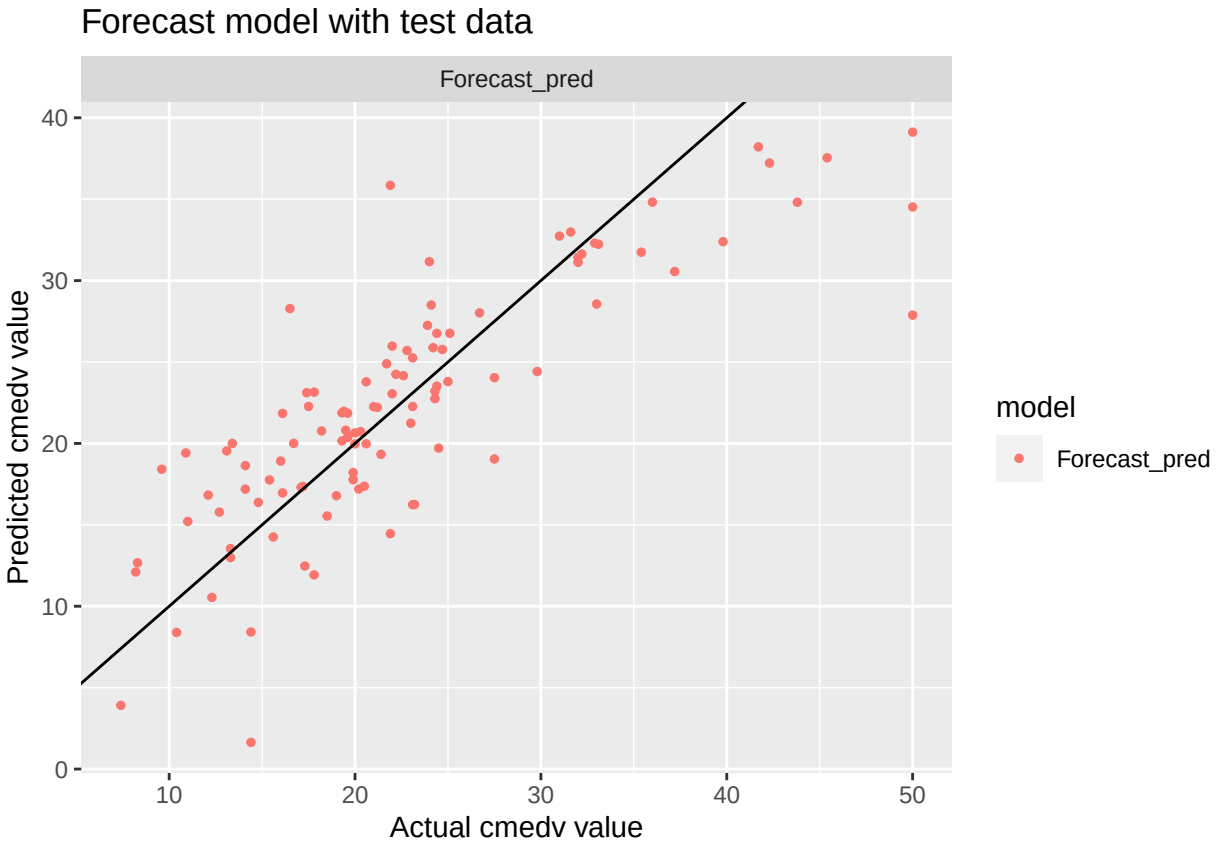

```
rmse7 <- rmse(df_pred_tr, Actual, Forecast_pred_tr, na_rm=TRUE)
rmse_pr_tr <- select(rmse7, .estimate)
mae7 <- mae(df_pred_tr, Actual, Forecast_pred_tr, na_rm=TRUE)
mae_pr_tr <- select(mae7, .estimate)
mape7 <- mape(df_pred_tr, Actual, Forecast_pred_tr, na_rm=TRUE)
mape_pr_tr <- select(mape7, .estimate)

metrics_pred_tr <- cbind(rmse_pr_tr, mae_pr_tr, mape_pr_tr)

names(metrics_pred_tr)[1] <- "RMSE"
names(metrics_pred_tr)[2] <- "MAE"
names(metrics_pred_tr)[3] <- "MAPE"
```

For train data:

```
##       RMSE      MAE     MAPE
## 1 5.167271 3.571508 18.30466
```

We get RMSE, MAE and MAPE which is lower than Model 1 but higher than Model 2 and 3.

For test data, the same codes are run.



For test data:

```
##       RMSE      MAE     MAPE
## 1 5.23833 3.796428 19.06718
```

The predicted values ran with training and test set (thus, within-sample and out-of-sample) have similar coefficients of RMSE, MAE and MAPE. This shows that our regression model is not overfitting, but still is rather effective in predicting the model (compared to Model 1).

Part 6: Can you say something about which variables appear to be the most important predictors of cmedv?

The lstat and rm seems to be the most important predictors of cmedv, as when they are plotted against the outcome variable of cmedv, the graphs show correlation between the variables. Furthermore, the correlation value between each of the predictors and cmedv is relatively high, as it is above 0.5 (as can be seen from the correlation matrix).

Part 7: Do you think any of the variables reflect true causal relationships?

The predictors chosen for our model make causal relationships with the cmedv intuitively. For percentage of lower status in the population (lstat), lower lstat value would mean that there are less people with "lower status", meaning that more people in the area would be relatively in "higher status", thus increasing the median value of the homes (cmedv). Also, average number of rooms per house (rm) is intuitively related to the median value of the houses, as the more rooms the houses have on average, the higher the houses would value. It is interesting for ptratio to be correlated with cmedv, as the plot shows that the lower the pupil to teacher ratio is, the higher cmedv value becomes. It may be interpreted that the "richer" or better-off areas with cmedv would have less children, thus less number of pupils per teacher. However, this may not be so intuitive so we cannot confirm a causal relationship between ptratio and cmedv.

Assignment 2

Reading and Pre-processing the data for the model

```
#restart environment
rm(list=ls())

#Read dataset
churn_df <- read_csv("customer_churn.csv")
```

```
## Rows: 7043 Columns: 21
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (17): customerID, gender, Partner, Dependents, PhoneService, MultipleLin...
## dbl  (4): SeniorCitizen, tenure, MonthlyCharges, TotalCharges
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
#convert vector into factor
churn_df <- churn_df %>%
  mutate(Churn = as.factor(Churn)) %>% #the output should be a factor in a classification
  mutate_if(is.character, as.factor) #Convert character vector to a factor
```

Split the sample dataset into a training dataset (80%) and a testing dataset (20%).

```
#split data into train (80%) & test (20%)
set.seed(123)

data_split <- initial_split(churn_df, prop=4/5)
train_set <- training(data_split)
test_set <- testing(data_split)
```

Ensuring the training dataset and testing dataset have similiar proportion

```
#to test for the proportion of churn for training set and testing set
train_set %>%
  count(Churn) %>%
  mutate(prop= n/sum(n))
```

```
## # A tibble: 2 x 3
##   Churn     n  prop
##   <fct> <int> <dbl>
## 1 No     4116 0.731
## 2 Yes    1518 0.269
```

```
#26.9% churn and 73.1% non-churn


test_set %>%
  count(Churn) %>%
  mutate(prop= n/sum(n))
```

```
## # A tibble: 2 x 3
##   Churn     n  prop
##   <fct> <int> <dbl>
## 1 No     1058 0.751
## 2 Yes     351 0.249
```

```
#24.9% churn and 75.1% non-churn
```

Why it is important to keep the proportions similiar? We noted that churn proportion are similiar between the testing and training data set that we created from the original data, i.e. 24.9% and 26.9%. It is important to have a similiar proportion of churn between the training and test data set to confirm that the samples taken from original data through the process of data splitting does represent the original data.

Estimate two linear OLS regression models that explain Churn, the telco customer churn rate, using the training data and testing data set.

```
#set up the logistic regression model for binary classification model
glm_model <-
  logistic_reg() %>%
  set_engine("glm")

#adjusting variables to use as prediction
Model1 <- glm_model %>% fit(Churn~gender+Partner+Dependents+tenure, data=train_set)

Model2 <- glm_model %>% fit(Churn~gender+SeniorCitizen+Partner+Dependents+tenure+MonthlyCharges+Contrac
```

Predict Churn for all two models, first for the training data itself, then for the testing data

```
#create dataframe for actual data for train_set
Actual_train <- select(train_set, "Churn")

#run prediciton for train_set
Forecast1 <- predict(Model1, train_set)
colnames(Forecast1) <- c("Forecast1")

Forecast2 <- predict(Model2, train_set)
colnames(Forecast2) <- c("Forecast2")

#create dataframe for train_set
df_train_1 <- cbind(Actual_train, Forecast1)
df_train_2 <- cbind(Actual_train, Forecast2)
```

```
#create dataframe for actual data for test_set
Actual_test <- select(test_set, "Churn")

#run prediciton for test_set
Forecast3 <- predict(Model1, test_set)
colnames(Forecast3) <- c("Forecast3")

Forecast4 <- predict(Model2, test_set)
colnames(Forecast4) <- c("Forecast4")

#create dataframe for test_set
df_test_1 <- cbind(Actual_test, Forecast3)
df_test_2 <- cbind(Actual_test, Forecast4)
```

Create confusion matrix for churn preidction with 40% threshold and accuracy, sensitivity & specificity for train data set for both models

```
#set same leveling for training dataset for the confusion matrix Model 1
df_train_1$Forecast1 <- ifelse(df_train_1$Forecast1 == "Yes", 1, 0)
df_train_1$Churn <- ifelse(df_train_1$Churn == "Yes", 1, 0)
```

```r
#creating confusion matrix for the training set Model 1
confusionMatrix(table(df_train_1$Forecast1, df_train_1$Churn), threshold = 0.4)
```

```
## Confusion Matrix and Statistics
##
##
##        0    1
##   0 3805 1058
##   1  311  460
##
##                Accuracy : 0.757
##                  95% CI : (0.7456, 0.7682)
##     No Information Rate : 0.7306
##     P-Value [Acc > NIR] : 3.313e-06
##
##                   Kappa : 0.2693
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9244
##             Specificity : 0.3030
##          Pos Pred Value : 0.7824
##          Neg Pred Value : 0.5966
##              Prevalence : 0.7306
##          Detection Rate : 0.6754
##    Detection Prevalence : 0.8632
##       Balanced Accuracy : 0.6137
##
##        'Positive' Class : 0
##
```

```r
#Visualisasion of model 1 train set confusion matrix
CM_df_train_1 <- data.frame(p = c(3805, 311, 4116),
                 n = c(1058, 460, 1518), Total = c(4863, 771, 5634))
rownames(CM_df_train_1) <-  c("Y", "N", "Total")
knitr::kable(CM_df_train_1, "pipe")
```

	p	n	Total
Y	3805	1058	4863
N	311	460	771
Total	4116	1518	5634

```r
#set same leveling for training dataset for the confusion matrix Model 2
df_train_2$Forecast2 <- ifelse(df_train_2$Forecast2 == "Yes", 1, 0)
df_train_2$Churn <- ifelse(df_train_2$Churn == "Yes", 1, 0)

#creating confusion matrix for the training set Model 2
confusionMatrix(table(df_train_2$Forecast2, df_train_2$Churn), threshold = 0.4)
```

```
## Confusion Matrix and Statistics
##
##
##        0    1
```

```
##   0 3681  773
##   1  435  745
##
##                Accuracy : 0.7856
##                  95% CI : (0.7746, 0.7962)
##     No Information Rate : 0.7306
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4142
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8943
##             Specificity : 0.4908
##          Pos Pred Value : 0.8264
##          Neg Pred Value : 0.6314
##              Prevalence : 0.7306
##          Detection Rate : 0.6534
##    Detection Prevalence : 0.7906
##       Balanced Accuracy : 0.6925
##
##        'Positive' Class : 0
##
```

```
#Visualisasion of model 2 train set confusion matrix
CM_df_train_2 <- data.frame(p = c(3681, 435, 4116),
                 n = c(773, 745, 1518), Total = c(4454, 1180, 5634))
rownames(CM_df_train_2) <-  c("Y", "N", "Total")
knitr::kable(CM_df_train_2, "pipe")
```

	p	n	Total
Y	3681	773	4454
N	435	745	1180
Total	4116	1518	5634

Create confusion matrix for churn prediction with 40% threshold and accuracy, sensitivity & specificity for test data set for both models

```
#set same leveling for testing data set for the confusion matrix
df_test_1$Forecast3 <- ifelse(df_test_1$Forecast3 == "Yes", 1, 0)
df_test_1$Churn <- ifelse(df_test_1$Churn == "Yes", 1, 0)

#creating confusion matrix for the testing set
confusionMatrix(table(df_test_1$Forecast3, df_test_1$Churn), threshold = 0.4)
```

```
## Confusion Matrix and Statistics
##
##
##       0   1
##   0 981 239
##   1  77 112
##
##                Accuracy : 0.7757
##                  95% CI : (0.753, 0.7973)
```

```
##     No Information Rate : 0.7509
##     P-Value [Acc > NIR] : 0.01601
##
##                   Kappa : 0.2912
##
##  Mcnemar's Test P-Value : < 2e-16
##
##             Sensitivity : 0.9272
##             Specificity : 0.3191
##          Pos Pred Value : 0.8041
##          Neg Pred Value : 0.5926
##              Prevalence : 0.7509
##          Detection Rate : 0.6962
##    Detection Prevalence : 0.8659
##       Balanced Accuracy : 0.6232
##
##        'Positive' Class : 0
##
```

```
#Visualisasion of model 1 test set confusion matrix
CM_df_test_1 <- data.frame(p = c(981, 77, 1058),
                 n = c(239, 112, 351), Total = c(1220, 189, 1409))
rownames(CM_df_test_1) <-  c("Y", "N", "Total")
knitr::kable(CM_df_test_1, "pipe")
```

	p	n	Total
Y	981	239	1220
N	77	112	189
Total	1058	351	1409

```
#set same leveling for testing data set for the confusion matrix
df_test_2$Forecast4 <- ifelse(df_test_2$Forecast4 == "Yes", 1, 0)
df_test_2$Churn <- ifelse(df_test_2$Churn == "Yes", 1, 0)

#creating confusion matrix for the testing set
confusionMatrix(table(df_test_2$Forecast4, df_test_2$Churn), threshold = 0.4)
```

```
## Confusion Matrix and Statistics
##
##
##       0   1
##   0 954 170
##   1 104 181
##
##                Accuracy : 0.8055
##                  95% CI : (0.7839, 0.8259)
##     No Information Rate : 0.7509
##     P-Value [Acc > NIR] : 6.630e-07
##
##                   Kappa : 0.4454
##
##  Mcnemar's Test P-Value : 8.609e-05
##
##             Sensitivity : 0.9017
```

```
##             Specificity : 0.5157
##          Pos Pred Value : 0.8488
##          Neg Pred Value : 0.6351
##              Prevalence : 0.7509
##          Detection Rate : 0.6771
##    Detection Prevalence : 0.7977
##       Balanced Accuracy : 0.7087
##
##        'Positive' Class : 0
##
```

```
#Visualisasion of model 1 test set confusion matrix
CM_df_test_2 <- data.frame(p = c(954, 104, 1058),
                 n = c(170, 181, 351), Total = c(1124, 285, 1409))
rownames(CM_df_test_2) <-  c("Y", "N", "Total")
knitr::kable(CM_df_test_2, "pipe")
```

	p	n	Total
Y	954	170	1124
N	104	181	285
Total	1058	351	1409

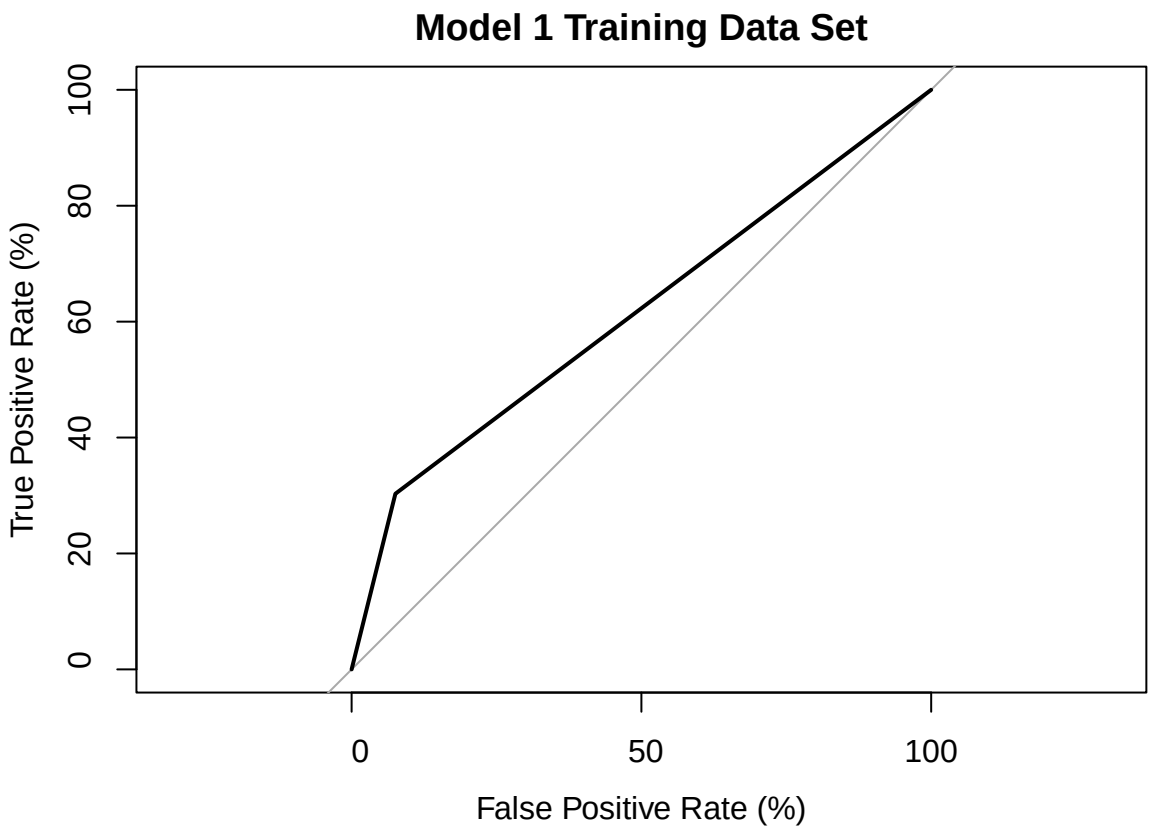
Create ROC curve, the cumulative response curve and the lift curve for both models, both for training and testing dataset. also calculate the AUC for each ROC curve

Plotting ROC and AUC calculation for model 1 and comparison between training and testing dataset

```
#ROC for model 1 training and testing set
roc(df_train_1$Churn, df_train_1$Forecast1, plot=TRUE, legacy.axes=TRUE, percent=TRUE, main = "Model 1 T
```

```
## Setting levels: control = 0, case = 1
```

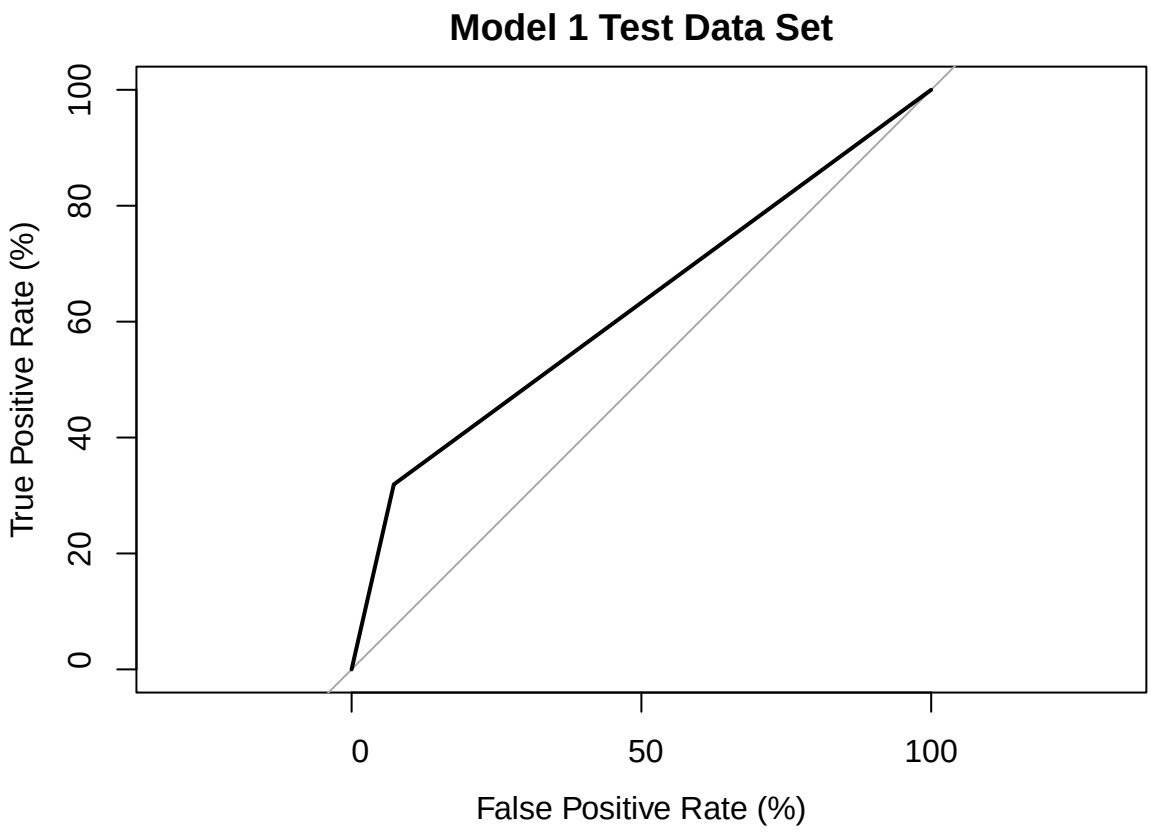
```
## Setting direction: controls < cases
```




```
##
## Call:
## roc.default(response = df_train_1$Churn, predictor = df_train_1$Forecast1,     percent = TRUE, plot =
##
## Data: df_train_1$Forecast1 in 4116 controls (df_train_1$Churn 0) < 1518 cases (df_train_1$Churn 1).
## Area under the curve: 61.37%
```

```
roc(df_test_1$Churn, df_test_1$Forecast3, plot=TRUE, legacy.axes=TRUE, percent=TRUE,main = "Model 1 Tes
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```




```
##
## Call:
## roc.default(response = df_test_1$Churn, predictor = df_test_1$Forecast3,     percent = TRUE, plot = T
##
## Data: df_test_1$Forecast3 in 1058 controls (df_test_1$Churn 0) < 351 cases (df_test_1$Churn 1).
## Area under the curve: 62.32%
```

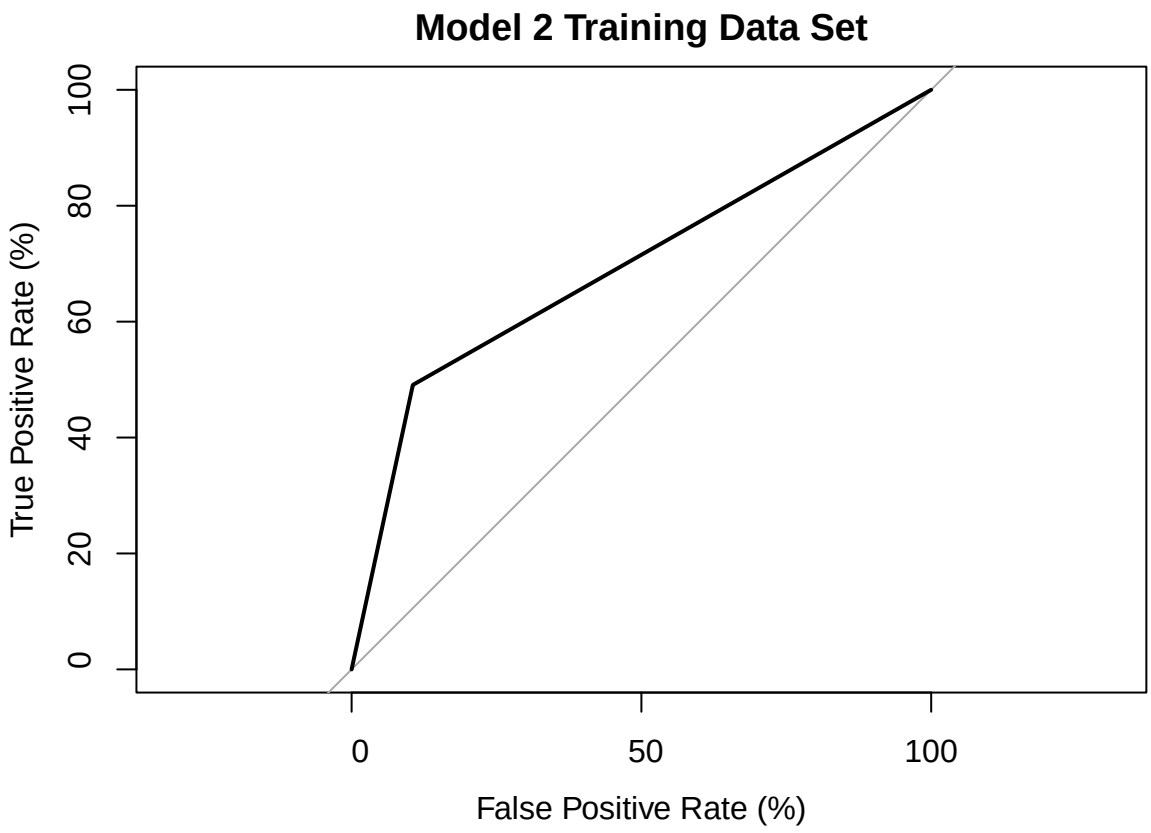
Model 1 ROC interpretation: From the result above we could conclude that the model is relatively able to distuingsh class on a decent level (although not so accurate), as we noted the AUC for both training and testing dataset are 61.37% and 62.32% respectively, just above the 50% cut-off (which means the model randomly guess the outcome/class). Moreover, we could also see that AUC for training set is marginally lower as compared to test set, hence the steeper curve for test data set ROC. This also indicates that model 1 ability in predicting the actual outcome becomes lower as you add more data.

Plotting ROC curve and AUC calculation for model 2 and comparison between training and testing dataset

```
#ROC curve for model 2 training and testing set
roc(df_train_2$Churn, df_train_2$Forecast2, plot=TRUE, legacy.axes=TRUE, percent=TRUE, main = "Model 2 T
```

```
## Setting levels: control = 0, case = 1
```

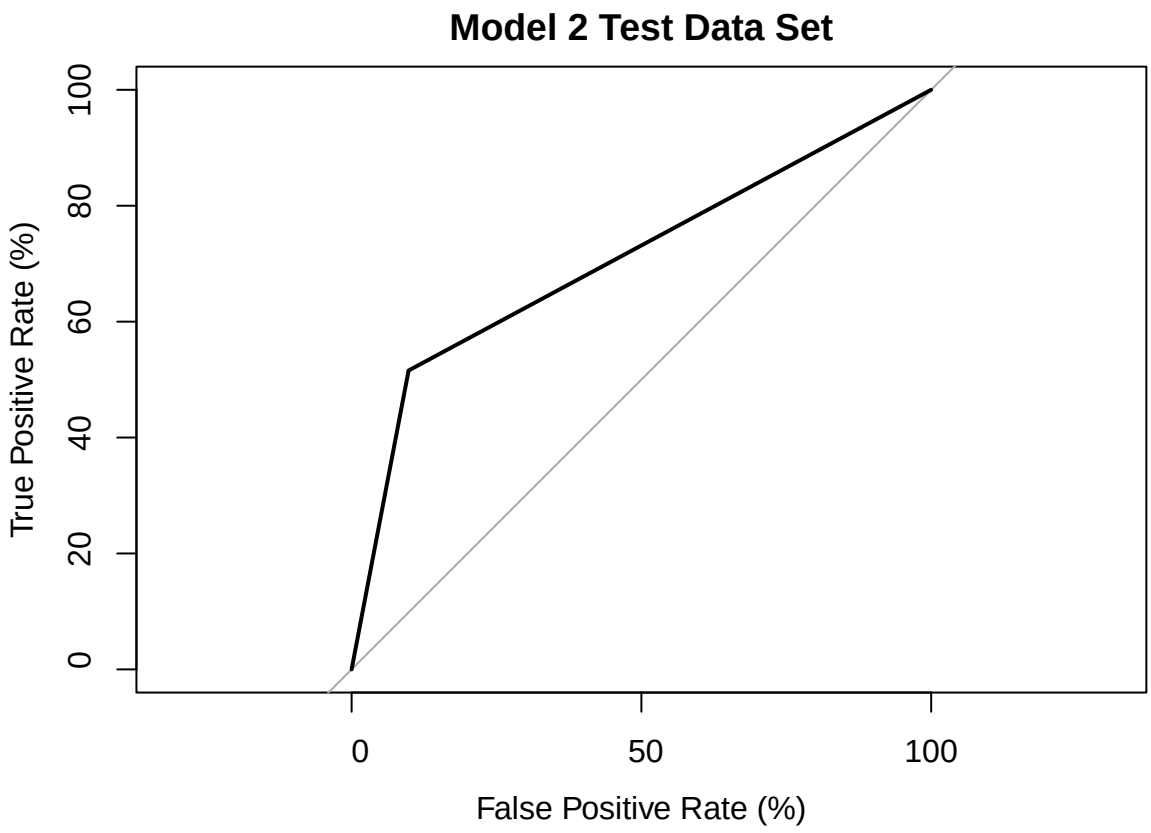
```
## Setting direction: controls < cases
```




```
##
## Call:
## roc.default(response = df_train_2$Churn, predictor = df_train_2$Forecast2,     percent = TRUE, plot =
##
## Data: df_train_2$Forecast2 in 4116 controls (df_train_2$Churn 0) < 1518 cases (df_train_2$Churn 1).
## Area under the curve: 69.25%
```

```
roc(df_test_2$Churn, df_test_2$Forecast4, plot=TRUE, legacy.axes=TRUE, percent=TRUE, main = "Model 2 Tes
```

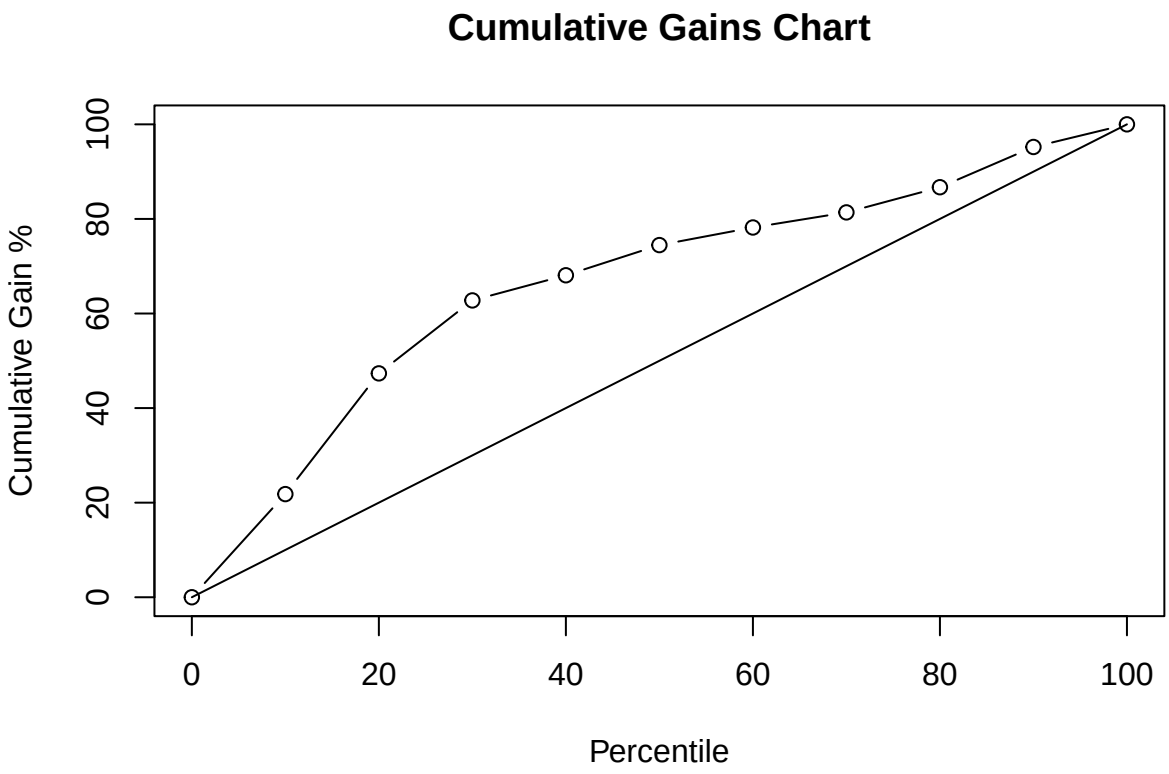
```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```



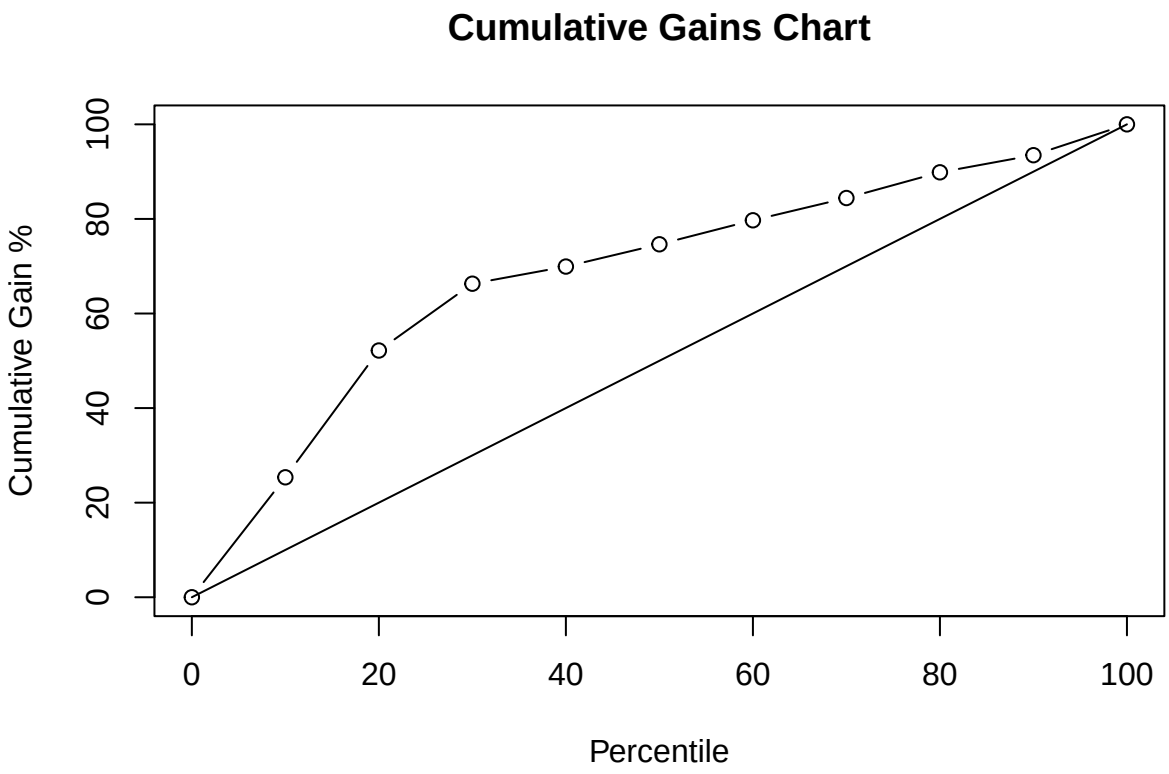

```
##
## Call:
## roc.default(response = df_test_2$Churn, predictor = df_test_2$Forecast4,     percent = TRUE, plot = T
##
## Data: df_test_2$Forecast4 in 1058 controls (df_test_2$Churn 0) < 351 cases (df_test_2$Churn 1).
## Area under the curve: 70.87%
```

Model 2 ROC interpretation: By adding additional predictor variables in the second model, we noted that accuracy of the model increased by circa 8-9% as compared to model 1. Under model 2, the AUC for training and test dataset are 69.25% and 70.87% repectively. Similiar to the previous model however, the accuracy fell as you add more data as indicated by lower AUC for training dataset as per above result. In overall, model 2 is a better model that could predict the actual outcome as compared to previous one.

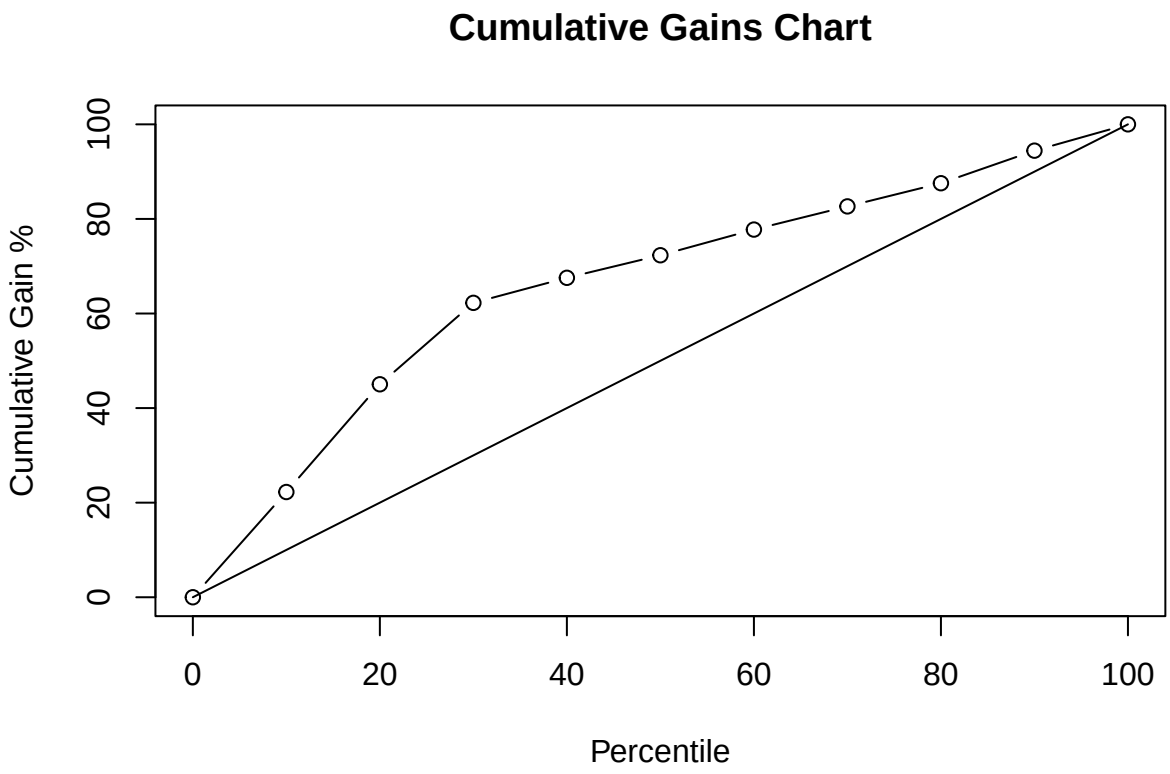
```
##Cummulative response curve

##trainingset 1

cumGainsChart(df_test_1$Churn, df_test_1$Forecast3, resolution = 1/10)
```



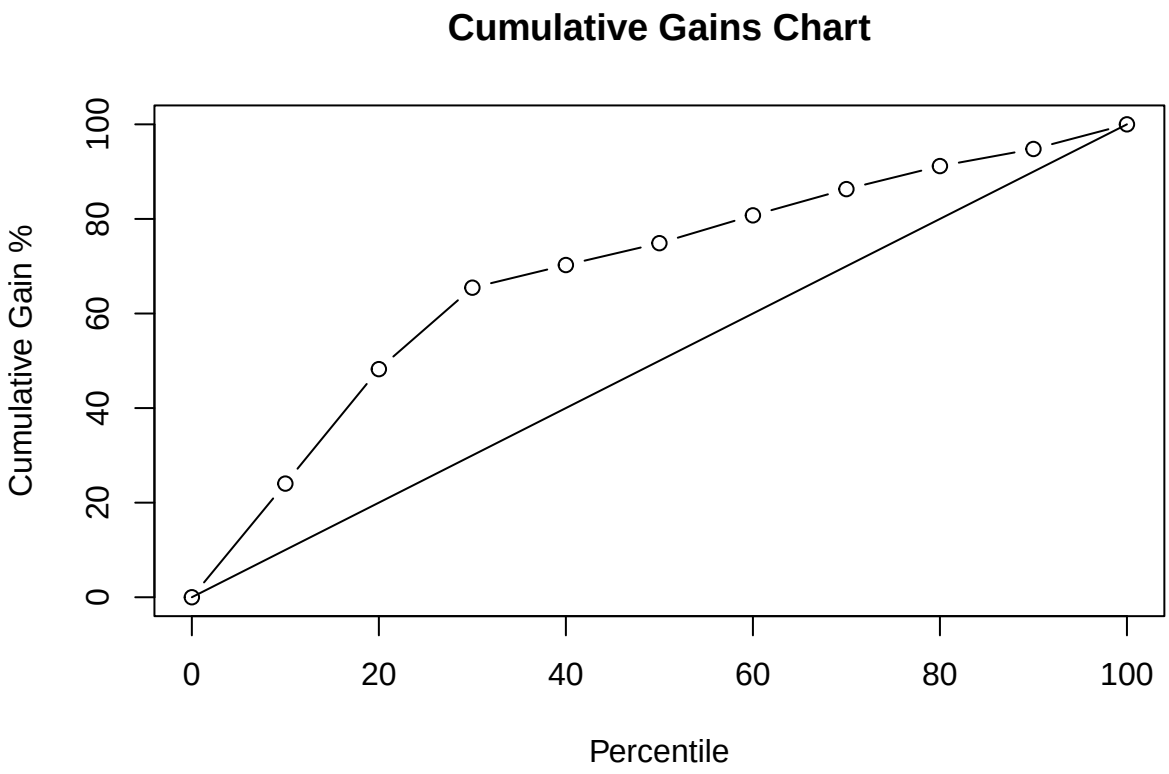

```
##trainingset 2

cumGainsChart(df_test_2$Churn, df_test_2$Forecast4, resolution = 1/10)
```




```
##testset 1

cumGainsChart(df_train_1$Churn, df_train_1$Forecast1, resolution = 1/10)
```

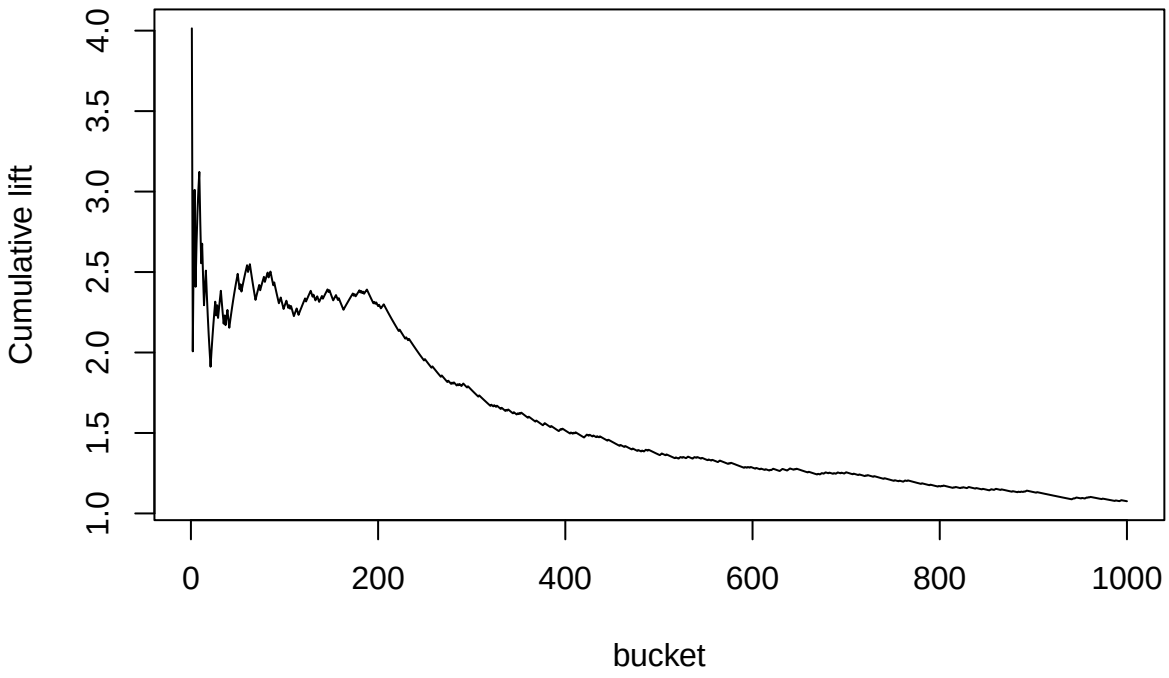



```
##testset 2


cumGainsChart(df_train_2$Churn, df_train_2$Forecast2, resolution = 1/10)
```

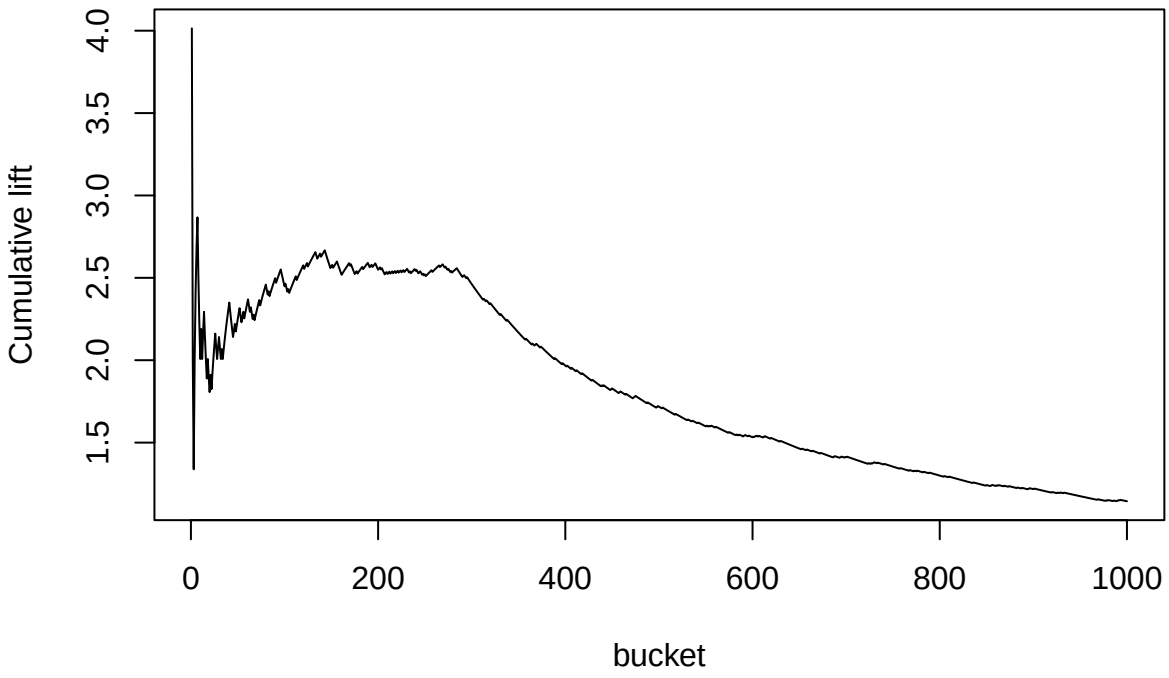


##Interpretation cummulative response curve In all four models, the response curve rises above the diagonal. The diagonal indicates an absence of reply. The cumulative response curve rises above the diagonal, indicating presence of a response.

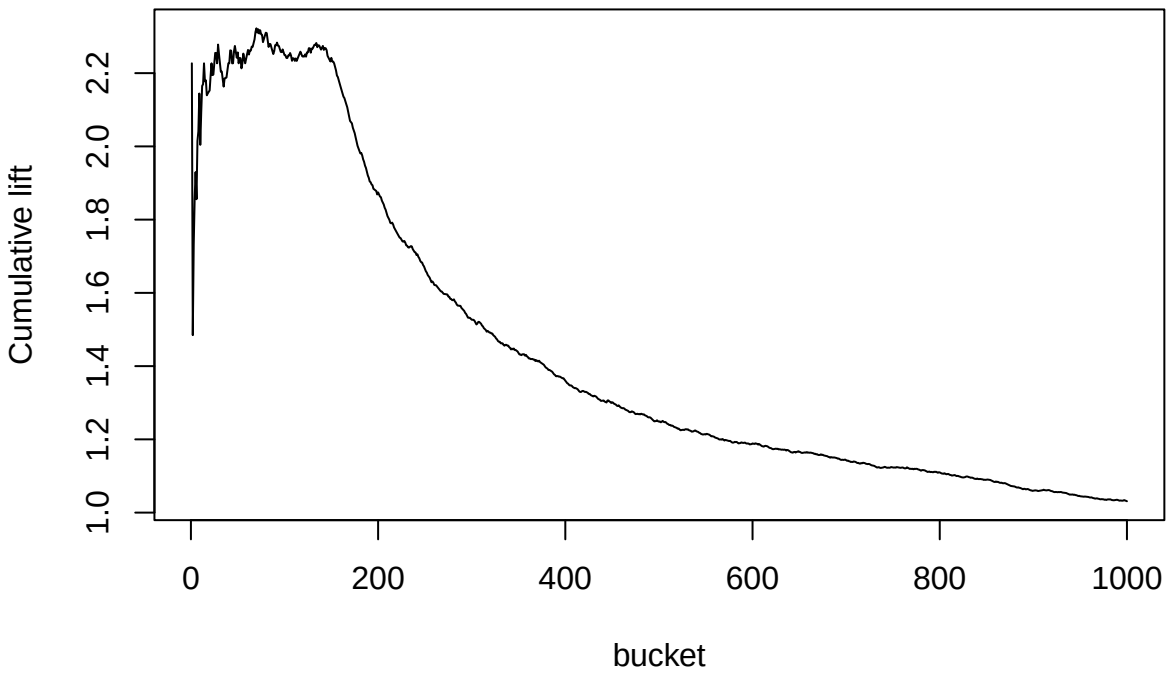
```
##plot lift curve

##testset 1

plotLift(df_test_1$Forecast3, df_test_1$Churn, cumulative = TRUE, n.buckets = 1000)
```



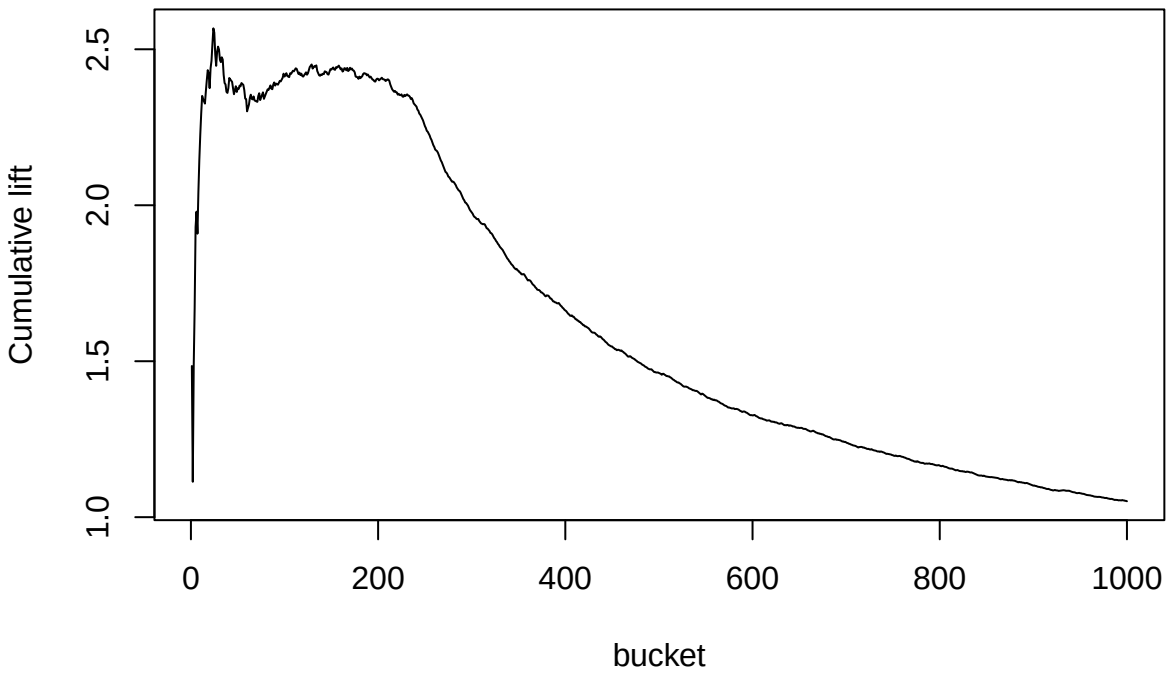

```
##testset 2

plotLift(df_test_2$Forecast4, df_test_2$Churn, cumulative = TRUE, n.buckets = 1000)
```




```
##trainset 1

plotLift(df_train_1$Forecast1, df_train_1$Churn, cumulative = TRUE, n.buckets = 1000)
```




```
##trainset 2

plotLift(df_train_2$Forecast2, df_train_2$Churn, cumulative = TRUE, n.buckets = 1000)
```



##Interpretion of lift curve The two models both give a 2x lift for the train- and dataset. This means that both the models' classifications are more than 2x better than random.