Name: Dhanal Patel

Title: Assignment 2 - Tree Based Models

```
# Clear objects if exists
rm(list = ls())
# Clear console
cat("\014")
# Clear plots
graphics.off()
# Clear packages
p_unload(all)
# Clear mem
gc()
# Clear console
rm(list=ls())
# Load packages
pacman::p_load(ggplot2, ggthemes, ggfortify, ggcorrplot, ggpairs, ggsurplot, ggcoxzph, ggcoxdiagnostics, ggcoxfunctional, ggforest, ggsurvplot, ggadjustedcurves, ggsurvevents, ggrmeanplot, ggrsurvfit, ggrmeanplot, ggrsurvevents, ggrsurvfit, ggrmeanplot, ggrsurvevents, ggrsurvfit, ggrmeanplot, ggrsurvevents
```

```
# Gradient Boosting

# Develop a Gradient Boosting model to predict the probability of loan default
gb = GradientBoostingClassifier(random_state=1)
gb.fit(X_train, y_train.ravel())
print("Accuracy on training set: {:.3f}".format(gb.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(gb.score(X_test, y_test)))
cm = confusion_matrix(y_test, gb.predict(X_test))
print("Confusion matrix\n", cm)
# Evaluate the accuracy of the model on the training and test sets
# Compute confusion matrix
cm = confusion_matrix(y_test, gb.predict(X_test))
print("Confusion matrix\n", cm)
# Compute and print the classification report
print("Classification report\n", classification_report(y_test,gb.predict(X_test)))
# Compute ROC curve and ROC AUC score
fpr, tpr, thresholds = roc_curve(y_test, gb.decision_function(X_test))
plot_roc_curve(fpr, tpr)

GradientBoostingClassifier(random_state=1)
Accuracy on training set: 0.873
Accuracy on test set: 0.866
Confusion matrix
 [[1166  161]
 [ 179 1194]]
Confusion matrix
 [[1166  161]
 [ 179 1194]]
Classification report
              precision    recall  f1-score   support

           0       0.87      0.88      0.87      1327
           1       0.88      0.87      0.87      1373

    accuracy                           0.87      2700
   macro avg       0.87      0.87      0.87      2700
weighted avg       0.87      0.87      0.87      2700

# Compute ROC curve and ROC AUC score
fpr, tpr, thresholds = roc_curve(y_test, gb.decision_function(X_test))
plot_roc_curve(fpr, tpr)
# Compute ROC AUC score
from sklearn.metrics import roc_auc_score
print("ROC AUC score: {:.3f}".format(roc_auc_score(y_test, gb.decision_function(X_test))))

ROC AUC score: 0.940

# Develop a Gradient Boosting model to predict the loss amount assuming that the loan defaults
# Create a copy of X_train and X_test
X_train_loss = X_train.copy()
# Add the target variable to the training set
X_train_loss['loss'] = y_train_loss
# Create a copy of X_test and X_test_loss
X_test_loss = X_test.copy()
# Add the target variable to the test set
X_test_loss['loss'] = y_test_loss
# Fit a Gradient Boosting regressor on the training set
gb = GradientBoostingRegressor(random_state=1)
gb.fit(X_train_loss, y_train_loss.ravel())
# Evaluate the model on the training and test sets
print("Accuracy on training set: {:.3f}".format(gb.score(X_train_loss, y_train_loss)))
print("Accuracy on test set: {:.3f}".format(gb.score(X_test_loss, y_test_loss)))

GradientBoostingRegressor(random_state=1)
Accuracy on training set: 0.000
Accuracy on test set: -0.000

# Compute and print the mean absolute error
from sklearn.metrics import mean_absolute_error
print("Mean absolute error: {:.3f}".format(mean_absolute_error(y_test_loss, gb.predict(X_test_loss))))

Mean absolute error: 1.000

# Compute and print the mean squared error
from sklearn.metrics import mean_squared_error
print("Mean squared error: {:.3f}".format(mean_squared_error(y_test_loss, gb.predict(X_test_loss))))

Mean squared error: 1.000

# Compute and print the root mean squared error
from sklearn.metrics import mean_squared_error
from math import sqrt
print("Root mean squared error: {:.3f}".format(sqrt(mean_squared_error(y_test_loss, gb.predict(X_test_loss)))))

Root mean squared error: 1.000

# Compute and print the mean squared log error
from sklearn.metrics import mean_squared_log_error
print("Mean squared log error: {:.3f}".format(mean_squared_log_error(y_test_loss, gb.predict(X_test_loss))))

Mean squared log error: 0.000

# Compute and print the median absolute error
from sklearn.metrics import median_absolute_error
print("Median absolute error: {:.3f}".format(median_absolute_error(y_test_loss, gb.predict(X_test_loss))))

Median absolute error: 0.000

# Compute and print the R-squared score
print("R-squared score: {:.3f}".format(gb.score(X_test_loss, y_test_loss)))

R-squared score: -0.000
```



```
# Compute and print the ROC AUC score
from sklearn.metrics import roc_auc_score
print("ROC AUC score: {:.3f}".format(roc_auc_score(y_test_loss, gb.predict(X_test_loss))))

ROC AUC score: 0.000
```

Develop a Gradient Boosting to predict the loss amount assuming that the loan defaults

```
# Create a copy of X_train and X_test
X_train_loss = X_train.copy()
# Add the target variable to the training set
X_train_loss['loss'] = y_train_loss
# Create a copy of X_test and X_test_loss
X_test_loss = X_test.copy()
# Add the target variable to the test set
X_test_loss['loss'] = y_test_loss
# Fit a Gradient Boosting regressor on the training set
gb = GradientBoostingRegressor(random_state=1)
gb.fit(X_train_loss, y_train_loss.ravel())
# Evaluate the model on the training and test sets
print("Accuracy on training set: {:.3f}".format(gb.score(X_train_loss, y_train_loss)))
print("Accuracy on test set: {:.3f}".format(gb.score(X_test_loss, y_test_loss)))

GradientBoostingRegressor(random_state=1)
Accuracy on training set: 0.000
Accuracy on test set: -0.000

# Compute and print the mean absolute error
from sklearn.metrics import mean_absolute_error
print("Mean absolute error: {:.3f}".format(mean_absolute_error(y_test_loss, gb.predict(X_test_loss))))

Mean absolute error: 1.000

# Compute and print the mean squared error
from sklearn.metrics import mean_squared_error
print("Mean squared error: {:.3f}".format(mean_squared_error(y_test_loss, gb.predict(X_test_loss))))

Mean squared error: 1.000

# Compute and print the root mean squared error
from sklearn.metrics import mean_squared_error
from math import sqrt
print("Root mean squared error: {:.3f}".format(sqrt(mean_squared_error(y_test_loss, gb.predict(X_test_loss)))))

Root mean squared error: 1.000

# Compute and print the mean squared log error
from sklearn.metrics import mean_squared_log_error
print("Mean squared log error: {:.3f}".format(mean_squared_log_error(y_test_loss, gb.predict(X_test_loss))))

Mean squared log error: 0.000

# Compute and print the median absolute error
from sklearn.metrics import median_absolute_error
print("Median absolute error: {:.3f}".format(median_absolute_error(y_test_loss, gb.predict(X_test_loss))))

Median absolute error: 0.000

# Compute and print the R-squared score
print("R-squared score: {:.3f}".format(gb.score(X_test_loss, y_test_loss)))

R-squared score: -0.000
```



```
# Compute and print the ROC AUC score
from sklearn.metrics import roc_auc_score
print("ROC AUC score: {:.3f}".format(roc_auc_score(y_test_loss, gb.predict(X_test_loss))))

ROC AUC score: 0.000
```

Develop a Gradient Boosting to predict the loss amount assuming that the loan defaults

```
# Create a copy of X_train and X_test
X_train_loss = X_train.copy()
# Add the target variable to the training set
X_train_loss['loss'] = y_train_loss
# Create a copy of X_test and X_test_loss
X_test_loss = X_test.copy()
# Add the target variable to the test set
X_test_loss['loss'] = y_test_loss
# Fit a Gradient Boosting regressor on the training set
gb = GradientBoostingRegressor(random_state=1)
gb.fit(X_train_loss, y_train_loss.ravel())
# Evaluate the model on the training and test sets
print("Accuracy on training set: {:.3f}".format(gb.score(X_train_loss, y_train_loss)))
print("Accuracy on test set: {:.3f}".format(gb.score(X_test_loss, y_test_loss)))

GradientBoostingRegressor(random_state=1)
Accuracy on training set: 0.000
Accuracy on test set: -0.000

# Compute and print the mean absolute error
from sklearn.metrics import mean_absolute_error
print("Mean absolute error: {:.3f}".format(mean_absolute_error(y_test_loss, gb.predict(X_test_loss))))

Mean absolute error: 1.000

# Compute and print the mean squared error
from sklearn.metrics import mean_squared_error
print("Mean squared error: {:.3f}".format(mean_squared_error(y_test_loss, gb.predict(X_test_loss))))

Mean squared error: 1.000

# Compute and print the root mean squared error
from sklearn.metrics import mean_squared_error
from math import sqrt
print("Root mean squared error: {:.3f}".format(sqrt(mean_squared_error(y_test_loss, gb.predict(X_test_loss)))))

Root mean squared error: 1.000

# Compute and print the mean squared log error
from sklearn.metrics import mean_squared_log_error
print("Mean squared log error: {:.3f}".format(mean_squared_log_error(y_test_loss, gb.predict(X_test_loss))))

Mean squared log error: 0.000

# Compute and print the median absolute error
from sklearn.metrics import median_absolute_error
print("Median absolute error: {:.3f}".format(median_absolute_error(y_test_loss, gb.predict(X_test_loss))))

Median absolute error: 0.000

# Compute and print the R-squared score
print("R-squared score: {:.3f}".format(gb.score(X_test_loss, y_test_loss)))

R-squared score: -0.000
```



```
# Compute and print the ROC AUC score
from sklearn.metrics import roc_auc_score
print("ROC AUC score: {:.3f}".format(roc_auc_score(y_test_loss, gb.predict(X_test_loss))))

ROC AUC score: 0.000
```

Include a discussion of the Decision Tree diagrams. Do they appear to make sense?

Decide Decision Tree Graphviz Display Hyperparameters

PLDEFINE <= 0.5 (True)

- The root node splits the sample for whether or not the loans defaulted is based on the PLDEFINE feature. Loans that did not default (PLDEFINE <= 0.5) are assigned to the left branch and loans that did default (PLDEFINE > 0.5) are assigned to the right branch. The root node has 10,800 samples.
- The branch of the tree that accepts the predicted loans that did not default is very deep. Many of the nodes in the left branch have a very small number of samples. The leaf nodes at the end of the left branch have a few samples which do not have their subtrees.
- The branch of the tree that accepts the predicted loans that did default (PLDEFINE > 0.5) is much shorter than the left branch. The leaf nodes at the end of the right branch have many samples which do have their subtrees.
- The root node splits the sample for whether or not the loans defaulted is based on the PLDEFINE feature. Loans that did not default (PLDEFINE <= 0.5) are assigned to the left branch and loans that did default (PLDEFINE > 0.5) are assigned to the right branch. The root node has 10,800 samples.
- The branch of the tree that accepts the predicted loans that did not default is very deep. Many of the nodes in the left branch have a very small number of samples. The leaf nodes at the end of the left branch have a few samples which do have their subtrees.
- The branch of the tree that accepts the predicted loans that did default (PLDEFINE > 0.5) is much shorter than the left branch. The leaf nodes at the end of the right branch have many samples which do have their subtrees.

PLDEFINE <= 0.5 (False)

- If loans income feature was missing in the data and binning or drop categories report where the loans income feature was missing, the loans income feature could be a good predictor for whether the loans defaulted or not.
- Similarly, if the loans loan amount feature was missing in the data and binning or drop categories report where the loans loan amount feature was missing, the loans loan amount feature could be a good predictor for whether the loans defaulted or not.
- Similarly, binning the debt income ratio feature in a report indicates these loan amount not repaid as well as the expression decision analysis shows.
- In the column file name which is the appraisal tree that achieved best data (AUC=0.940), which is the decision tree shown above.

LOAN <= 2.0(False)

- On the other hand, bigger the loans loan value the greater, ceteris paribus, the probability that the loans will default. Loans with loan amount greater than 2 (LOAN > 2) are more likely to default. Loans with loan amounts less than or equal to 2 (LOAN <= 2) are more likely to not default.
- The PLDEFINE ratio flag is most indicative for predicting of default loans. According to the Gini criterion, loans with ratio equal to one (PLDEFINE ratio flag=1) are more likely to default. Loans with ratio equal to zero (PLDEFINE ratio flag=0) are more likely to not default.
- Similarly, binning the debt income ratio feature in a report indicates these loan amount not repaid as well as the expression decision analysis shows.
- In the column file name which is the appraisal tree that achieved best data (AUC=0.940), which is the decision tree shown above.

LOAN <= 2.0(True)

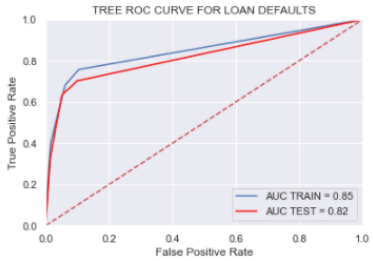
- On the other hand, bigger the loans loan value the greater, ceteris paribus, the probability that the loans will default. Loans with loan amount greater than 2 (LOAN > 2) are more likely to default. Loans with loan amounts less than or equal to 2 (LOAN <= 2) are more likely to not default.
- The PLDEFINE ratio flag is most indicative for predicting of default loans. According to the Gini criterion, loans with ratio equal to one (PLDEFINE ratio flag=1) are more likely to default. Loans with ratio equal to zero (PLDEFINE ratio flag=0) are more likely to not default.
- Similarly, binning the debt income ratio feature in a report indicates these loan amount not repaid as well as the expression decision analysis shows.
- In the column file name which is the appraisal tree that achieved best data (AUC=0.940), which is the decision tree shown above.

Which variables appear to be most predictive of loan default? Do they make sense?

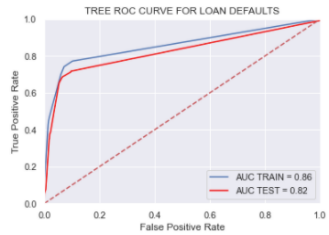
- The LOAN variable appears to be most predictive of loan default of the loan records. The LOAN variable is the loan amount. The LOAN variable is the loan amount that the borrower asks for. The loan amount that asks for the borrower to settle a debt or purchase an item bigger than usual. The LOAN variable is the loan amount that the borrower asks for. The LOAN variable is the number of credit lines the borrower has in the credit file. The LOAN variable is the number of credit lines the borrower has in the credit file. The LOAN variable is the number of credit lines the borrower has in the credit file

BINGO BONUS – Tree Model Analysis

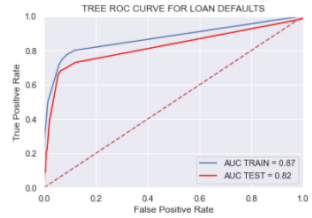
```
DECISION TREE SPLIT TRAINING DATA: 80% and TEST DATA: 20%
RANDOM STATE = 1, MAX_DEPTH = 3

Accuracy Metrics Train 0.8873741610738255
Accuracy Metrics Test 0.8825503355704698

TREE RMSE Train: 5302.823208185897
TREE RMSE Test: 6116.70195540116

M_DEROG
IMP_DELINQ
IMP_CLAGE
M_DEBTINC
IMP_DEBTINC
```



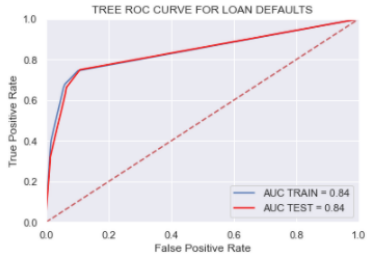

```
DECISION TREE SPLIT TRAINING DATA: 80% and TEST DATA: 20%
RANDOM STATE = 1, MAX_DEPTH = 4

Accuracy Metrics Train 0.8928271812080537
Accuracy Metrics Test 0.8859060402684564

TREE RMSE Train: 4491.106341702886
TREE RMSE Test: 5574.966442540175

M_VALUE
IMP_VALUE
IMP_YOJ
M_DEROG
IMP_DEROG
IMP_DELINQ
IMP_CLAGE
M_DEBTINC
IMP_DEBTINC
```



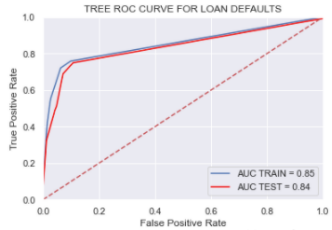

```
DECISION TREE SPLIT TRAINING DATA: 80% and TEST DATA: 20%
RANDOM STATE = 1, MAX_DEPTH = 5

Accuracy Metrics Train 0.8978607382550335
Accuracy Metrics Test 0.8859060402684564

TREE RMSE Train: 3637.8029771469874
TREE RMSE Test: 5320.505109287866

z_JOB_Mgr
z_JOB_Sales
M_VALUE
IMP_VALUE
M_YOJ
IMP_YOJ
M_DEROG
IMP_DEROG
IMP_DELINQ
IMP_CLAGE
M_DEBTINC
IMP_DEBTINC
```



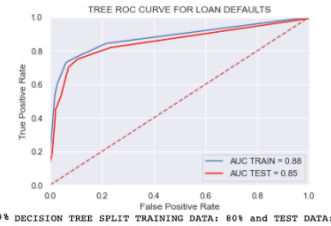

```
DECISION TREE SPLIT TRAINING DATA: 80% and TEST DATA: 20%
RANDOM STATE = 2, MAX_DEPTH = 3

Accuracy Metrics Train 0.8888422818791947
Accuracy Metrics Test 0.8766778523489933

TREE RMSE Train: 5302.823208185897
TREE RMSE Test: 6116.70195540116

M_VALUE
M_DEROG
IMP_DELINQ
IMP_CLAGE
M_DEBTINC
IMP_DEBTINC
```



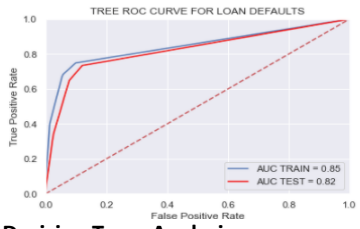

```
DECISION TREE SPLIT TRAINING DATA: 80% and TEST DATA: 20%
RANDOM STATE = 2, MAX_DEPTH = 4

Accuracy Metrics Train 0.8949244966442953
Accuracy Metrics Test 0.8791946308724832

TREE RMSE Train: 4491.106341702886
TREE RMSE Test: 5574.966442540175

M_VALUE
IMP_VALUE
IMP_MORTDUE
M_YOJ
IMP_YOJ
M_DEROG
IMP_DELINQ
IMP_CLAGE
M_DEBTINC
IMP_DEBTINC
```



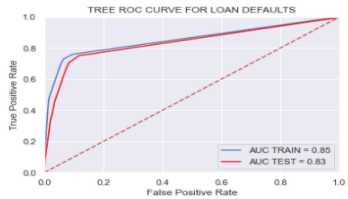

```
DECISION TREE SPLIT TRAINING DATA: 80% and TEST DATA: 20%
RANDOM STATE = 2, MAX_DEPTH = 5

Accuracy Metrics Train 0.8991191275167785
Accuracy Metrics Test 0.8691275167785235

TREE RMSE Train: 3637.8029771469874
TREE RMSE Test: 5315.064301473744

LOAN
M_VALUE
IMP_VALUE
IMP_MORTDUE
M_YOJ
IMP_YOJ
M_DEROG
IMP_DEROG
IMP_DELINQ
IMP_CLAGE
IMP_NINQ
M_DEBTINC
IMP_DEBTINC
```



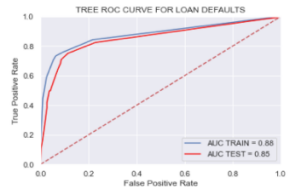

```
DECISION TREE SPLIT TRAINING DATA: 80% and TEST DATA: 20%
RANDOM STATE = 3, MAX_DEPTH = 3

Accuracy Metrics Train 0.8911493288590604
Accuracy Metrics Test 0.8674496644295302

TREE RMSE Train: 5302.823208185897
TREE RMSE Test: 6116.70195540116

M_VALUE
IMP_VALUE
M_DEROG
IMP_DELINQ
IMP_CLAGE
M_DEBTINC
IMP_DEBTINC
```




```
DECISION TREE SPLIT TRAINING DATA: 80% and TEST DATA: 20%
RANDOM STATE = 3, MAX_DEPTH = 4

Accuracy Metrics Train 0.8951342281879194
Accuracy Metrics Test 0.8758389261744967

TREE RMSE Train: 4491.106341702886
TREE RMSE Test: 5574.966442540175

M_VALUE
IMP_VALUE
M_YOJ
IMP_YOJ
M_DEROG
IMP_DEROG
IMP_DELINQ
IMP_CLAGE
IMP_NINQ
M_DEBTINC
IMP_DEBTINC
```




```
DECISION TREE SPLIT TRAINING DATA: 80% and TEST DATA: 20%
RANDOM STATE = 3, MAX_DEPTH = 5

Accuracy Metrics Train 0.8986996644295302
Accuracy Metrics Test 0.87248322147651

TREE RMSE Train: 3637.8029771469874
TREE RMSE Test: 5429.735308946132

z_JOB_Mgr
z_JOB_Office
M_VALUE
IMP_VALUE
IMP_MORTDUE
M_YOJ
IMP_YOJ
M_DEROG
IMP_DEROG
IMP_DELINQ
IMP_CLAGE
M_DEBTINC
IMP_DEBTINC
```

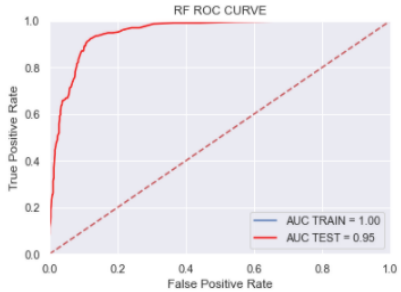


Decision Trees Analysis:

I tried using the following parameters random state: 1, 2, 3 and max_depth of 3,4,5 both Decision Tree Classifier and Regressor. Based on the analysis, I see most of the variables that appear to be predictive of loan default are repeating throughout the process when random_state is equal to 1,2,3. According to me, when parameter is random_state = 2, and max_depth = 3 is most accurate model for both train and test dataset shows exactly same 84% accuracy based on the ROC curve shows no overfitting in decision tree models. I did not see much difference in RMSE average for loss amount as well.

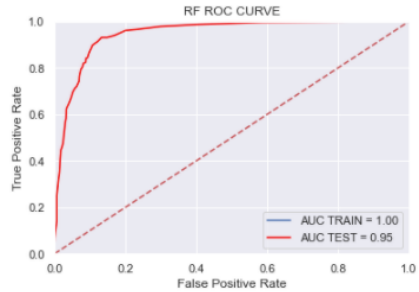
BINGO BONUS – Tree Model Analysis

```
RANDOM FOREST, n_estimators=100, random_state=1

Probability of loans default
Accuracy Train: 1.0
Accuracy Test: 0.8959731543624161
```



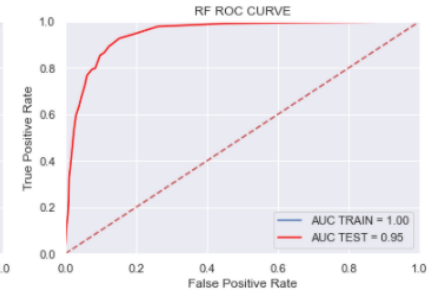

```
RF RMSE Train: 1197.9405112156378
RF RMSE Test: 2849.498972119216
('M_DEBTINC', 100)
('IMP_DEBTINC', 75)
('IMP_CLAGE', 44)
('IMP_DELINQ', 41)
('LOAN', 36)
('IMP_VALUE', 35)
('IMP_MORTDUE', 32)
('IMP_CLNO', 32)
('IMP_YOJ', 27)
('IMP_DEROG', 22)
```

```
RANDOM FOREST, n_estimators=50, random_state=1

Probability of loans default
Accuracy Train: 0.9997902684563759
Accuracy Test: 0.8984899328859061
```



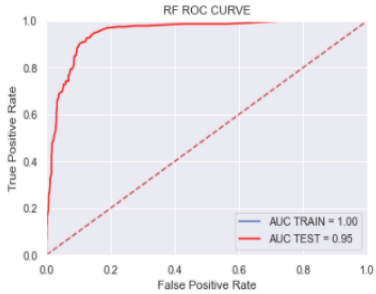

```
RF RMSE Train: 1241.1352252563606
RF RMSE Test: 2897.844181569543
('M_DEBTINC', 100)
('IMP_DEBTINC', 68)
('IMP_CLAGE', 43)
('IMP_DELINQ', 37)
('LOAN', 34)
('IMP_VALUE', 34)
('IMP_CLNO', 30)
('IMP_MORTDUE', 30)
('IMP_YOJ', 25)
('IMP_DEROG', 21)
```

```
RANDOM FOREST, n_estimators=25, random_state=1

Probability of loans default
Accuracy Train: 0.9991610738255033
Accuracy Test: 0.9001677852348994
```



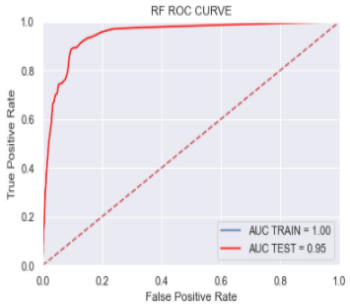

```
RF RMSE Train: 1357.9482266310504
RF RMSE Test: 2971.411326281644
('M_DEBTINC', 100)
('IMP_DEBTINC', 72)
('IMP_CLAGE', 43)
('IMP_DELINQ', 40)
('IMP_VALUE', 37)
('LOAN', 36)
('IMP_MORTDUE', 31)
('IMP_CLNO', 31)
('IMP_YOJ', 27)
('IMP_DEROG', 23)
```

```
RANDOM FOREST, n_estimators=100, random_state=2

Probability of loans default
Accuracy Train: 1.0
Accuracy Test: 0.9026845637583892
```



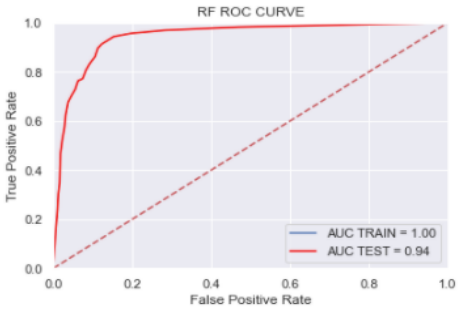

```
RF RMSE Train: 1216.1249383469587
RF RMSE Test: 2725.2868582707592
('M_DEBTINC', 100)
('IMP_DEBTINC', 74)
('IMP_CLAGE', 43)
('IMP_DELINQ', 42)
('IMP_VALUE', 37)
('LOAN', 36)
('IMP_MORTDUE', 33)
('IMP_CLNO', 32)
('IMP_YOJ', 27)
('IMP_DEROG', 23)
```

```
RANDOM FOREST, n_estimators=50, random_state=2

Probability of loans default
Accuracy Train: 0.9997902684563759
Accuracy Test: 0.9068791946308725
```



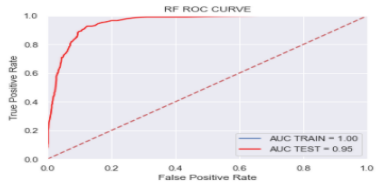

```
RF RMSE Train: 1219.0328614260393
RF RMSE Test: 2723.10478274755
('M_DEBTINC', 100)
('IMP_DEBTINC', 78)
('IMP_CLAGE', 45)
('IMP_DELINQ', 42)
('LOAN', 38)
('IMP_VALUE', 38)
('IMP_MORTDUE', 34)
('IMP_CLNO', 33)
('IMP_YOJ', 28)
('IMP_DEROG', 24)
```

```
RANDOM FOREST, n_estimators=25, random_state=2

Probability of loans default
Accuracy Train: 0.9995805369127517
Accuracy Test: 0.9052013422818792
```



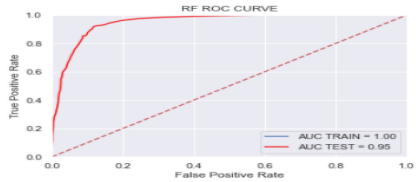

```
RF RMSE Train: 1199.5995899723578
RF RMSE Test: 2683.4073967144714
('M_DEBTINC', 100)
('IMP_DEBTINC', 77)
('IMP_CLAGE', 44)
('IMP_DELINQ', 43)
('LOAN', 39)
('IMP_VALUE', 38)
('IMP_CLNO', 35)
('IMP_MORTDUE', 32)
('IMP_YOJ', 30)
('IMP_DEROG', 24)
```

```
RANDOM FOREST, n_estimators=100, random_state=3

Probability of loans default
Accuracy Train: 1.0
Accuracy Test: 0.9052013422818792
```



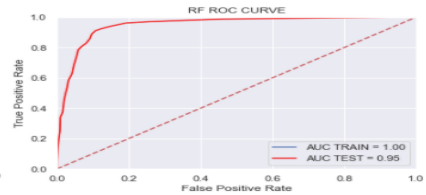

```
RF RMSE Train: 1236.9649689709129
RF RMSE Test: 2802.023399871712
('M_DEBTINC', 100)
('IMP_DEBTINC', 78)
('IMP_CLAGE', 46)
('IMP_DELINQ', 43)
('IMP_VALUE', 38)
('LOAN', 38)
('IMP_MORTDUE', 35)
('IMP_CLNO', 33)
('IMP_YOJ', 29)
('IMP_DEROG', 20)
```

```
RANDOM FOREST, n_estimators=50, random_state=3

Probability of loans default
Accuracy Train: 1.0
Accuracy Test: 0.8984899328859061
```




```
RF RMSE Train: 1305.1566605386315
RF RMSE Test: 2864.85052041424
('M_DEBTINC', 100)
('IMP_DEBTINC', 76)
('IMP_CLAGE', 46)
('IMP_DELINQ', 40)
('LOAN', 37)
('IMP_VALUE', 35)
('IMP_MORTDUE', 33)
('IMP_CLNO', 32)
('IMP_YOJ', 28)
('IMP_DEROG', 20)
```

```
RANDOM FOREST, n_estimators=25, random_state=3

Probability of loans default
Accuracy Train: 0.9985318791946308
Accuracy Test: 0.9001677852348994
```



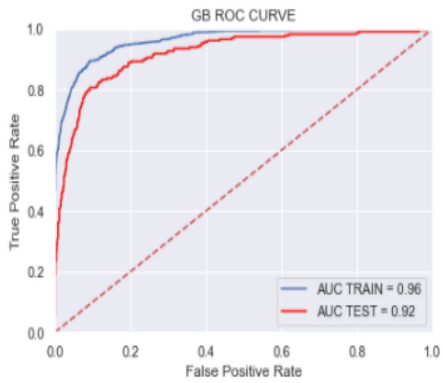

```
RF RMSE Train: 1336.0235532894253
RF RMSE Test: 2806.3880429914107
('M_DEBTINC', 100)
('IMP_DEBTINC', 84)
('IMP_CLAGE', 50)
('IMP_DELINQ', 44)
('LOAN', 41)
('IMP_VALUE', 37)
('IMP_MORTDUE', 35)
('IMP_CLNO', 34)
('IMP_YOJ', 29)
```

BINGO BONUS – Tree Model Analysis

Random Forest Analysis:

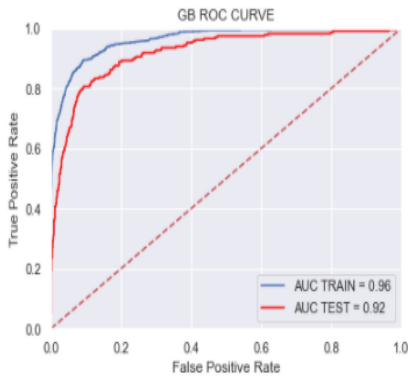
The following parameters random state: 1, 2, 3 and n_estimators (number of decision tree) we are creating is 100, 50, 25 for both Decision Tree Classifier and Regressor. Based on the analysis, I see most of the variables that appear to be predictive of loan default are repeating throughout the process when random_state is equal to 1,2,3. The ROC curve has remained constants through the testing with 95% accuracy for test data. The probability of loans defaults has accuracy around 90% for testing dataset it has 1-2% better accuracy than simple decision tree model for test data. Also, the RMSE (Root Mean Square Error) average for loss amount is around $2900 dollars for test dataset which is much better compare to RMSE error of simple decision tree.

```
GRADIENT BOOSTING, random_state=1

Probability of loan default
Accuracy Train: 0.9234479865771812
Accuracy Test: 0.8917785234899329
```



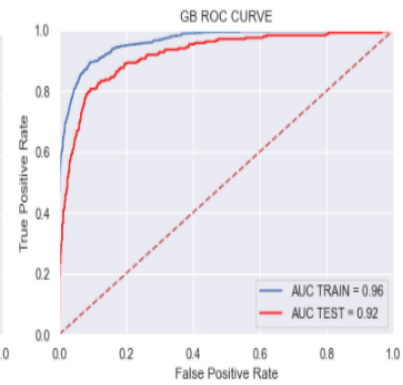

```
GB RMSE Train: 1236.1029566215122
GB RMSE Test: 2369.470492466002
('M_DEBTINC', 100)
('IMP_DEBTINC', 29)
('IMP_DELINQ', 19)
('IMP_CLAGE', 14)
```

```
GRADIENT BOOSTING, random_state=2

Probability of loan default
Accuracy Train: 0.9234479865771812
Accuracy Test: 0.8917785234899329
```




```
GB RMSE Train: 1236.1029566215127
GB RMSE Test: 2278.9733055947545
('M_DEBTINC', 100)
('IMP_DEBTINC', 29)
('IMP_DELINQ', 19)
('IMP_CLAGE', 14)
```

```
GRADIENT BOOSTING, random_state=3

Probability of loan default
Accuracy Train: 0.9234479865771812
Accuracy Test: 0.8917785234899329
```




```
GB RMSE Train: 1236.1029566215125
GB RMSE Test: 2314.878250832823
('M_DEBTINC', 100)
('IMP_DEBTINC', 29)
('IMP_DELINQ', 19)
('IMP_CLAGE', 14)
```

Gradient Boosting Analysis:

The following parameters random state: 1, 2, 3 for both Gradient Boosting Classifier and Regressor. Based on the analysis, I see all variables that appear to be predictive of loan default are repeating throughout the process when random_state is equal to 1,2,3. The ROC curve has remained constants through the testing with 92% accuracy for test data. The probability of loans defaults has accuracy around 92% for testing dataset it is 3% less accurate than random forest model for test data. Also, the RMSE (Root Mean Square Error) average for loss amount is around $2278 dollars for test dataset which is much better other models. I would use in production if its legal based on business criteria met for predicting loss amount.

In [67]:

```
package main
 import (
     "fmt"
     "log"
     "os"
 "github.com/go-gota/gota/dataframe"
    "base"
 )
```

```
error loading package "base" metadata: error executing "/usr/local/go/bin/go get base"
in directory "/Users/dhavalpatel/go/src/gomacro.imports/base": exit status 1
```

In [52]:

```
// Open CSV
     csvfile, err := os.Open("prepared_hmeqloss.csv")
     if err != nil {
         log.Fatal(err)
     }
    // Read CSV
    df := dataframe.ReadCSV(csvfile)
    fmt.Println(df)
```

```
[5960x29] DataFrame

    TARGET_BAD_FLAG TARGET_LOSS_AMT LOAN  z_REASON_DebtCon z_REASON_HomeImp ...
 0: 1               641.000000      1100  0                1                ...
 1: 1               1109.000000     1300  0                1                ...
 2: 1               767.000000      1500  0                1                ...
 3: 1               1425.000000     1500  0                0                ...
 4: 0               NaN             1700  0                1                ...
 5: 1               335.000000      1700  0                1                ...
 6: 1               1841.000000     1800  0                1                ...
 7: 1               373.000000      1800  0                1                ...
 8: 1               1217.000000     2000  0                1                ...
 9: 1               1523.000000     2000  0                1                ...
    ...             ...             ...   ...              ...              ...
    <int>           <float>         <int> <int>            <int>            ...

Not Showing: z_JOB_Mgr <int>, z_JOB_Office <int>, z_JOB_Other <int>, z_JOB_ProfExe <in
t>,
z_JOB_Sales <int>, z_JOB_Self <int>, M_VALUE <int>, IMP_VALUE <float>, M_MORTDUE <int
>,
IMP_MORTDUE <float>, M_YOJ <int>, IMP_YOJ <float>, M_DEROG <int>, IMP_DEROG <float>, M
_DELINQ <int>,
IMP_DELINQ <float>, M_CLAGE <int>, IMP_CLAGE <float>, M_NINQ <int>, IMP_NINQ <float>,
M_CLNO <int>,
IMP_CLNO <float>, M_DEBTINC <int>, IMP_DEBTINC <float>
```

Out[52]:

```
1497 <nil>
```

In [64]:

```
X := df.Copy()
X := X.Drop("TARGET_BAD_FLAG")
X := X.Drop("TARGET_LOSS_AMT")
X
```

Out[64]:

```
[5960x27] DataFrame

    LOAN  z_REASON_DebtCon z_REASON_HomeImp z_JOB_Mgr z_JOB_Office ...
 0: 1100  0                1                0         0            ...
 1: 1300  0                1                0         0            ...
 2: 1500  0                1                0         0            ...
 3: 1500  0                0                0         0            ...
 4: 1700  0                1                0         1            ...
 5: 1700  0                1                0         0            ...
 6: 1800  0                1                0         0            ...
 7: 1800  0                1                0         0            ...
 8: 2000  0                1                0         0            ...
 9: 2000  0                1                0         0            ...
    ...   ...              ...              ...       ...          ...
    <int> <int>            <int>            <int>     <int>        ...

Not Showing: z_JOB_Other <int>, z_JOB_ProfExe <int>, z_JOB_Sales <int>, z_JOB_Self <in
t>,
M_VALUE <int>, IMP_VALUE <float>, M_MORTDUE <int>, IMP_MORTDUE <float>, M_YOJ <int>,
IMP_YOJ <float>, M_DEROG <int>, IMP_DEROG <float>, M_DELINQ <int>, IMP_DELINQ <float>,
M_CLAGE <int>, IMP_CLAGE <float>, M_NINQ <int>, IMP_NINQ <float>, M_CLNO <int>,
IMP_CLNO <float>, M_DEBTINC <int>, IMP_DEBTINC <float>
```

In [60]:

```
Y := df.Select([]string{"TARGET_BAD_FLAG", "TARGET_LOSS_AMT"})
Y
```

Out[60]:

```
[5960x2] DataFrame

    TARGET_BAD_FLAG TARGET_LOSS_AMT
 0: 1               641.000000
 1: 1               1109.000000
 2: 1               767.000000
 3: 1               1425.000000
 4: 0               NaN
 5: 1               335.000000
 6: 1               1841.000000
 7: 1               373.000000
 8: 1               1217.000000
 9: 1               1523.000000
    ...             ...
    <int>           <float>
```

In [65]:

```
X_train, X_test, Y_train, Y_test := base.InstancesTrainTestSplit(csvfile, 0.80)
```

```
repl.go:1:37: undefined identifier: base
```

I ran into issues doing this assignment in Golang after spliting the data because of finding and researching about different packages in Golang, and install in Jupyter Notebook was time consuming. As a new programmer in Golang trying to find resources felt difficult, and to grasp the syntax, finding answer to different problems onlines was not user friendly compare to popular languages such as Python, R, Java, C++, SAS.

In [ ]:

In [86]:

```
using Pkg
Pkg.add("ML.Models")
Pkg.add("DecisionTree")
# Pkg.add("DataFrames")
# Pkg.add("CSV")
# Pkg.add("Lathe")
# Pkg.add("MLDataUtils")

import CSV
import DataFrames
import Lathe
import MLDataUtils
import MLModels
import DecisionTree
```

```
  Resolving package versions...
  Installed ScientificTypes ────────── v1.1.2
  Installed UnPack ─────────────────── v1.0.2
  Installed ProgressMeter ──────────── v1.7.1
  Installed ComputationalResources ─── v0.3.2
  Installed ML.ModelInterface ──────── v1.1.0
  Installed ML.ScientificTypes ─────── v0.4.5
  Installed ML.Models ──────────────── v0.14.7
  Installed LossFunctions ──────────── v0.6.0
  Installed RecipesBase ────────────── v1.1.2
  Installed PersistenceDiagramsBase ── v0.1.1
  Installed Distances ──────────────── v0.10.4
  Installed ColorTypes ─────────────── v0.10.12
  Installed StatisticalTraits ──────── v0.1.0
  Installed FixedPointNumbers ──────── v0.8.4
  Installed Parameters ─────────────── v0.12.3
  Installed ML.Base ────────────────── v0.18.1
    Updating `~/.julia/environments/v1.6/Project.toml`
  [d491faf4] + ML.Models v0.14.7
    Updating `~/.julia/environments/v1.6/Manifest.toml`
  [3da002f7] + ColorTypes v0.10.12
  [ed09ef8] + ComputationalResources v0.3.2
  [b4f34e82] + Distances v0.10.4
  [53c48c17] + FixedPointNumbers v0.8.4
  [30fc2ffe] + LossFunctions v0.6.0
  [a7f614a8] + ML.Base v0.18.1
  [e80e1ace] + ML.ModelInterface v1.1.0
  [d491faf4] + ML.Models v0.14.7
  [2e3236e0] + ML.ScientificTypes v0.4.5
  [d96e819e] + Parameters v0.12.3
  [b1ad91c1] + PersistenceDiagramsBase v0.1.1
  [92933f4c] + ProgressMeter v1.7.1
  [3cdcf5f2] + RecipesBase v1.1.2
  [321657f4] + ScientificTypes v1.1.2
  [64bff920] + StatisticalTraits v1.1.0
  [3a84ed6] + UnPack v1.0.2
Precompiling project...
  ✓ UnPack
  ✓ ComputationalResources
  ✓ ScientificTypes
  ✓ RecipesBase
  ✓ ProgressMeter
  ✓ Distances
  ✓ PersistenceDiagramsBase
  ✓ LearnBase
  ✓ Parameters
  ✓ StatisticalTraits
  ✓ FixedPointNumbers
  ✓ CategoricalArrays
  ✓ ML.ModelInterface
  ✓ ML.LabelUtils
  ✓ LossFunctions
  ✓ ML.DataPattern
  ✓ Distributions
  ✓ ColorTypes
  ✓ ML.ScientificTypes
  ✓ ML.Base
  ✓ ML.Models
  ✓ DataFrames
  ✓ ML.DataUtils
  ✓ Lathe
  24 dependencies successfully precompiled in 54 seconds (70 already precompiled, 1 sk
ipped during auto due to previous errors)
  ┌ Warning: Module ML.Models is precompiled but different versions are currently loaded. Restart juli
a to access the new versions
    Resolving package versions...
    Installed ScikitLearnBase ─ v0.5.0
    Installed DecisionTree ──── v0.10.11
      Updating `~/.julia/environments/v1.6/Project.toml`
  [7806a523] + DecisionTree v0.10.11
      Updating `~/.julia/environments/v1.6/Manifest.toml`
  [7806a523] + DecisionTree v0.10.11
  [6e75b9c4] + ScikitLearnBase v0.5.0
Precompiling project...
  ✓ ScikitLearnBase
  ✓ LearnBase
  ✓ CategoricalArrays
  ✓ DecisionTree
  ✓ ML.LabelUtils
  ✓ LossFunctions
  ✓ ML.ScientificTypes
  ✓ ML.DataPattern
  ✓ Distributions
  ✓ ML.Base
  ✓ ML.Models
  ✓ DataFrames
  ✓ ML.DataUtils
  ✓ Lathe
  14 dependencies successfully precompiled in 47 seconds (82 already precompiled, 1 sk
ipped during auto due to previous errors)
  ┌ Warning: Module ML.Models is precompiled but different versions are currently loaded. Restart juli
a to access the new versions
  ┌ Info: Precompiling ML.Models [d491faf4-2d78-11e9-2867-c94bc002c0b7]
  └ @ Base loading.jl:1342
  ┌ Warning: Module Missings with build ID 2237913648824247 is missing from the cache.
  │ This may mean Missings [ed2d9d7a-bbdc-5cf2-9ac0-f12de2c33e28] does not support preco
mpilation but is imported by a module that does.
  └ @ Base loading.jl:1030
  ┌ Warning: Module Missings with build ID 2237913648824247 is missing from the cache.
  │ This may mean Missings [ed2d9d7a-bbdc-5cf2-9ac0-f12de2c33e28] does not support preco
mpilation but is imported by a module that does.
  └ @ Base loading.jl:1030
  ┌ Warning: Module Missings with build ID 2237913648824247 is missing from the cache.
  │ This may mean Missings [ed2d9d7a-bbdc-5cf2-9ac0-f12de2c33e28] does not support preco
mpilation but is imported by a module that does.
  └ @ Base loading.jl:1030
  ┌ Info: Skipping precompilation since __precompile__(false). Importing MLJModels [d491
faf4-2d78-11e9-2867-c94bc002c0b7].
  └ @ Base loading.jl:1047
  ┌ Info: Precompiling ML.ScientificTypes [2e2323e0-db8b-457b-ae0d-bdfb3bc63afd]
  └ @ Base loading.jl:1342
  ┌ Warning: Module Missings with build ID 2237913648824247 is missing from the cache.
  │ This may mean Missings [ed2d9d7a-bbdc-5cf2-9ac0-f12de2c33e28] does not support preco
mpilation but is imported by a module that does.
  └ @ Base loading.jl:1030
  ┌ Warning: Module Missings with build ID 2237913648824247 is missing from the cache.
  │ This may mean Missings [ed2d9d7a-bbdc-5cf2-9ac0-f12de2c33e28] does not support preco
mpilation but is imported by a module that does.
  └ @ Base loading.jl:1030
  ┌ Info: Skipping precompilation since __precompile__(false). Importing ML.ScientificTy
pes [2e2323e0-db8b-457b-ae0d-bdfb3bc63afd].
  └ @ Base loading.jl:1047
  ┌ Info: Precompiling CategoricalArrays [324d7699-5711-5eae-9e2f-1d82baab6b597]
  └ @ Base loading.jl:1342
  ┌ Warning: Module Missings with build ID 2237913648824247 is missing from the cache.
  │ This may mean Missings [ed2d9d7a-bbdc-5cf2-9ac0-f12de2c33e28] does not support preco
mpilation but is imported by a module that does.
  └ @ Base loading.jl:1030
  ┌ Info: Skipping precompilation since __precompile__(false). Importing CategoricalArra
ys [324d7699-5711-5eae-9e2f-1d82baab6b597].
  └ @ Base loading.jl:1047
  ┌ Info: Precompiling MLJBase [a7f614a8-145f-11e9-1d2a-a57a1082229d]
  └ @ Base loading.jl:1342
  ┌ Warning: Module ML.ScientificTypes with build ID 2242357117685011 is missing from th
e cache.
  │ This may mean ML.ScientificTypes [2e2323e0-db8b-457b-ae0d-bdfb3bc63afd] does not su
pport precompilation but is imported by a module that does.
  └ @ Base loading.jl:1030
  ┌ Info: Skipping precompilation since __precompile__(false). Importing MLJBase [a7f614
a8-145f-11e9-1d2a-a57a1082229d].
  └ @ Base loading.jl:1047
  ┌ Info: Precompiling LossFunctions [30fc2ffe-d236-52d8-8643-a9d8f7c094a7]
  └ @ Base loading.jl:1342
  ┌ Warning: Module LearnBase with build ID 2242357117685011 is missing from the cache.
  │ This may mean LearnBase [7f8f8fb0-2700-5f03-b4bd-41f8cfc144b6] does not support prec
ompilation but is imported by a module that does.
  └ @ Base loading.jl:1030
  ┌ Info: Skipping precompilation since __precompile__(false). Importing LossFunctions
  [30fc2ffe-d236-52d8-8643-a9d8f7c094a7].
  └ @ Base loading.jl:1047
UndefVarError: DecisionTree_ not defined

Stacktrace:
 [1] top-level scope
   @ In[88]:1
 [2] eval
   @ ./boot.jl:360 [inlined]
 [3] include_string(mapexpr::typeof(REPL.softscope), mod::Module, code::String, filena
me::String)
   @ Base ./loading.jl:1116
```

In [64]:

```
# reading the file
csv_reader = CSV.File("prepared_hmeqloss.csv")
println(typeof(csv_reader))
```

```
CSV.File
```

In [65]:

```
df = DataFrame(CSV.File("prepared_hmeqloss.csv"))
first(df,5)
```

Out[65]: 5 rows × 29 columns (omitted printing of 23 columns)

	TARGET_BAD_FLAG	TARGET_LOSS_AMT	LOAN	z_REASON_DebtCon	z_REASON_HomeImp	z_JOB_M
	Int64	Float64?	Int64	Int64	Int64	In
1	1	641.0	1100	0	1	
2	1	1109.0	1300	0	1	
3	1	767.0	1500	0	1	
4	1	1425.0	1500	0	0	
5	0	*missing*	1700	0	1	

In [66]:

```
X = copy(df)
first(X, 5)
```

Out[66]: 5 rows × 29 columns (omitted printing of 23 columns)

	TARGET_BAD_FLAG	TARGET_LOSS_AMT	LOAN	z_REASON_DebtCon	z_REASON_HomeImp	z_JOB_M
	Int64	Float64?	Int64	Int64	Int64	In
1	1	641.0	1100	0	1	
2	1	1109.0	1300	0	1	
3	1	767.0	1500	0	1	
4	1	1425.0	1500	0	0	
5	0	*missing*	1700	0	1	

In [67]:

```
X = select!(X, Not(:TARGET_BAD_FLAG))
X = select!(X, Not(:TARGET_LOSS_AMT))
first(X,5)
```

Out[67]: 5 rows × 27 columns (omitted printing of 21 columns)

	LOAN	z_REASON_DebtCon	z_REASON_HomeImp	z_JOB_Mgr	z_JOB_Office	z_JOB_Other
	Int64	Int64	Int64	Int64	Int64	Int64
1	1100	0	1	0	0	1
2	1300	0	1	0	0	1
3	1500	0	1	0	0	1
4	1500	0	0	0	0	0
5	1700	0	1	0	1	0

In [68]:

```
Y = df[:,2, [:TARGET_BAD_FLAG, :TARGET_LOSS_AMT]]
first(Y,5)
```

Out[68]: 2 rows × 2 columns

	TARGET_BAD_FLAG	TARGET_LOSS_AMT
	Int64	Float64?
1	1	641.0
2	1	1109.0

In [81]:

```
X_train, X_test = splitobs(X, at = 0.80);
```

In [83]:

```
 println(size(X_train))
 println(size(X_test))
```

```
(4768, 27)
(1192, 27)
```

In [85]:

```
Y_train, Y_test = splitobs(Y, at=0.80)
 println(size(X_train))
 println(size(X_test))
```

```
(4768, 27)
(1192, 27)
```

In [88]:

```
fm01_Tree = DecisionTreeClassifier(max_depth=4)
fm01_Tree.fit(X, Y[TARGET_F])
```

```
UndefVarError: DecisionTreeClassifier not defined

Stacktrace:
 [1] top-level scope
   @ In[88]:1
 [2] eval
   @ ./boot.jl:360 [inlined]
 [3] include_string(mapexpr::typeof(REPL.softscope), mod::Module, code::String, filena
me::String)
   @ Base ./loading.jl:1116
```

I ran into issues doing this assignment in Julia after spliting the data because of finding and researching about different packages in Julia, and install in Jupyter Notebook was time consuming. As a new programmer in Julia trying to find resources felt difficult, and to grasp the syntax, finding answer to different problems onlines was not user friendly compare to popular languages such as Python, R, Java, C++, SAS.

BINGO BONUS (Tree Based Models in R) on HTML file

BINGO BONUS (I have attempted to do this Assignment in R). I was successful in putting an effort to build models and splitting the data in R. I have attached a html file for that since it could not be converted in pdf from RStudio.