Vivian Xia

Assignment 2: Statistical Output

Dataframe:

Original dataframe output:

```
df = pd.read_csv('HMEQ_Loss.csv')
print(df.head().T)
```

	0	1	2	3	4
TARGET_BAD_FLAG	1	1	1	1	0
TARGET_LOSS_AMT	641	1109	767	1425	NaN
LOAN	1100	1300	1500	1500	1700
MORTDUE	25860	70053	13500	NaN	97800
VALUE	39025	68400	16700	NaN	112000
REASON	HomeImp	HomeImp	HomeImp	NaN	HomeImp
JOB	Other	Other	Other	NaN	Office
YOJ	10.5	7	4	NaN	3
DEROG	0	0	0	NaN	0
DELINQ	0	2	0	NaN	0
CLAGE	94.3667	121.833	149.467	NaN	93.3333
NINQ	1	0	1	NaN	0
CLNO	9	14	10	NaN	14
DEBTINC	NaN	NaN	NaN	NaN	NaN

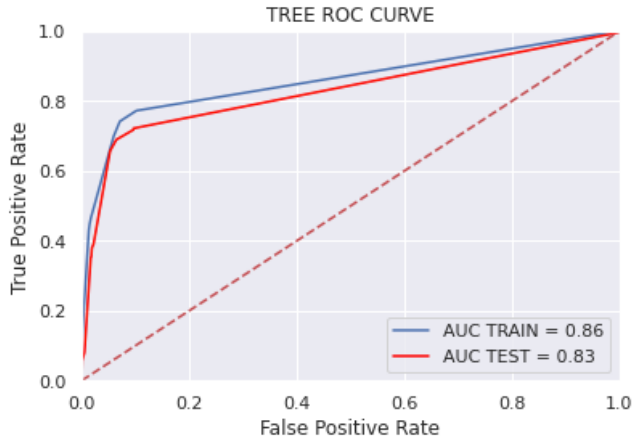
After handle missing values and categorical values output:

	0	1	2	3	4
TARGET_BAD_FLAG	1.000000	1.000000	1.000000	1.000000	0.000000
TARGET_LOSS_AMT	641.000000	1109.000000	767.000000	1425.000000	NaN
LOAN	1100.000000	1300.000000	1500.000000	1500.000000	1700.000000
z_IMP_REASON_HomeImp	1.000000	1.000000	1.000000	0.000000	1.000000
z_IMP_REASON_MISSING	0.000000	0.000000	0.000000	1.000000	0.000000
z_IMP_JOB_Mgr	0.000000	0.000000	0.000000	0.000000	0.000000
z_IMP_JOB_Office	0.000000	0.000000	0.000000	0.000000	1.000000
z_IMP_JOB_Other	1.000000	1.000000	1.000000	0.000000	0.000000
z_IMP_JOB_ProfExe	0.000000	0.000000	0.000000	0.000000	0.000000
z_IMP_JOB_Sales	0.000000	0.000000	0.000000	0.000000	0.000000
z_IMP_JOB_Self	0.000000	0.000000	0.000000	0.000000	0.000000
M_MORTDUE	0.000000	0.000000	0.000000	1.000000	0.000000
IMP_MORTDUE	25860.000000	70053.000000	13500.000000	65019.000000	97800.000000
M_VALUE	0.000000	0.000000	0.000000	1.000000	0.000000
IMP_VALUE	39025.000000	68400.000000	16700.000000	89235.500000	112000.000000
M_YOJ	0.000000	0.000000	0.000000	1.000000	0.000000
IMP_YOJ	10.500000	7.000000	4.000000	7.000000	3.000000
M_DEROG	0.000000	0.000000	0.000000	1.000000	0.000000
IMP_DEROG	0.000000	0.000000	0.000000	0.000000	0.000000
M_DELINQ	0.000000	0.000000	0.000000	1.000000	0.000000
IMP_DELINQ	0.000000	2.000000	0.000000	0.000000	0.000000
M_CLAGE	0.000000	0.000000	0.000000	1.000000	0.000000
IMP_CLAGE	94.366667	121.833333	149.466667	173.466667	93.333333
M_NINQ	0.000000	0.000000	0.000000	1.000000	0.000000
IMP_NINQ	1.000000	0.000000	1.000000	1.000000	0.000000
M_CLNO	0.000000	0.000000	0.000000	1.000000	0.000000
IMP_CLNO	9.000000	14.000000	10.000000	20.000000	14.000000
M_DEBTINC	1.000000	1.000000	1.000000	1.000000	1.000000
IMP_DEBTINC	34.818262	34.818262	34.818262	34.818262	34.818262

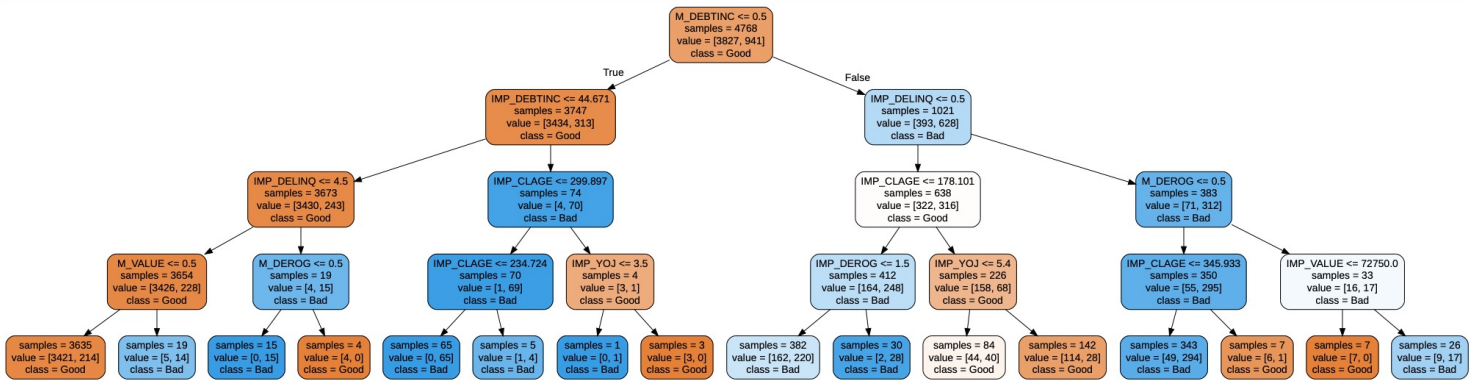
Decision Tree:

```
print("Probability of loan default")
print("Accuracy Train:",metrics.accuracy_score(Y_train[TARGET_FLAG], Y_Pred_train))
print("Accuracy Test:",metrics.accuracy_score(Y_test[TARGET_FLAG], Y_Pred_test))
```

```
Probability of loan default
Accuracy Train: 0.8928271812080537
Accuracy Test: 0.8875838926174496
```



The ROC curve shows the accuracy of the training and testing data on predicting the probability that the loan will default. The model looks reasonable. The training set did only a little better than the testing set, so the model is not overfit.



From first glance, the decision tree diagram for whether someone's loan defaults looks like it makes sense because the colors generally get more concentrated/darker to the bottom. At the top, the value [3827, 941] means that 3827 people did not have their loans default and 941 did so 20% that had their loan default.
The attribute of M_DEBTINC, is the first to be used to split the data depending on whether the debt-to-income ratio is missing or not. M_DEBTINC is encoded as 0 for not missing and 1 for missing. The users that do fill in the debt-to-income-ratio are encoded with 0 and are less than or equal to 0.5. The split puts those 3747 users

that do fill in their debt-to-income ratio in the class of Good, so these users have a higher chance of not having their loan default since 313 had their loan default while 3434 did not, so the chance of defaulting drop from 20% to 8%. On the other hand, if the debt-to-income ratio is missing, then the value is 1 and it goes to the right and is put into the class of Bad, so these users have a higher chance of their loan defaulting since 628 users had their loan default while only 393 did not. This split makes sense since those who do not fill in their debt-to-income ratio may tend to be those with bad ratios and would not want their ratios to be recorded and tied to their profile.

Branching off from the left side of the first node, the data is split on the attribute IMP_DEBTINC, which contains the actual values of the debt-to-income ratio. If people had a debt-to-income ratio less than or equal to 44.671 then the users have a higher chance of their loan not defaulting. If people had a debt-to-income ratio greater than 44.671 then the users have a higher chance of their loan defaulting. This split makes sense since a higher debt-to-income ratio means the user is riskier and may not be able to pay their bills.

This decision tree also looks at IMP_DELINQ, IMP_CLAGE, M_VALUE, IMP_VALUE, M_DEROG, IMP_DEROG, and IMP_YOJ. IMP_DELINQ refers to the delinquencies on a user's current credit report, so it makes sense that a smaller amount of delinquencies would lead to a higher chance that the loan will not default. IMP_CLAGE is the how long the user has had credit. Those who have had credit for longer are considered less risky, so it makes sense for the tree to split where if the user had less than a certain number of months, the user would be riskier and more likely to have their loan default. M_VALUE represents if the value of the user's house was missing or not where 0 is not missing and 1 is missing. Those with missing VALUE are more likely to have their loan default. The IMP_VALUE is also used to split the data. IMP_VALUE is the value of the house, so it is riskier if the user had more expensive house than not. The user is more likely to have their loan default if the value of the house is greater than a certain value. M_DEROG represents if the derogatory marks on credit record is missing or not. It makes sense that those with missing values for M_DEROG would be likelier to have their loan default. Similarly, the smaller the value for IMP_DEROG, the less risky the user and smaller chance that their loans will default. IMP_YOJ is years on job so users who have been at a job for a while are less risky those who have not been on a job for as long.

```
vars_tree_flag = getTreeVars( fm01_Tree, feature_cols )

for i in vars_tree_flag :
    print(i)
```

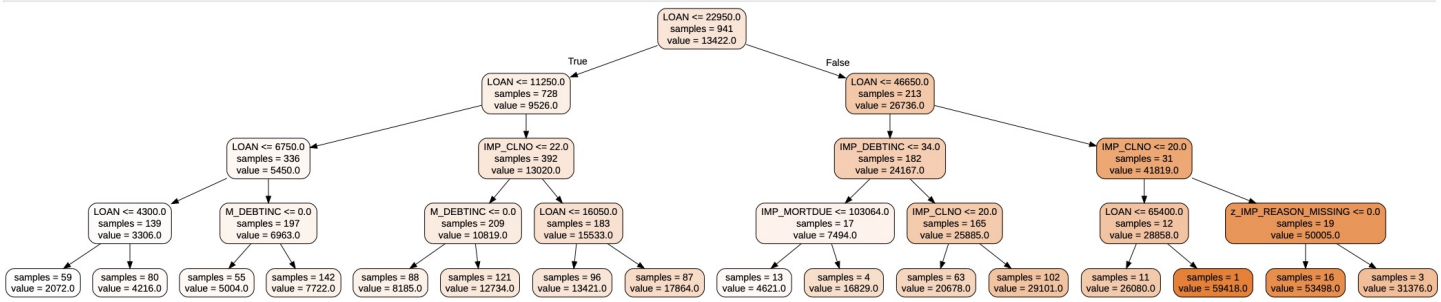
```
M_VALUE
IMP_VALUE
IMP_YOJ
M_DEROG
IMP_DEROG
IMP_DELINQ
IMP_CLAGE
M_DEBTINC
IMP_DEBTINC
```

The M_VALUE, IMP_VALUE, IMP_YOJ, M_DEROG, IMP_DEROG, IMP_DELINQ, IMP_CLAGE, M_DEBTINC, IMP_DEBTINC are the variables that were used in the tree, and therefore, the most predictive of loan default. These variables make sense in splitting the data into the classes as discussed above.

```
print("TREE RMSE Train:", RMSE_TRAIN )
print("TREE RMSE Test:", RMSE_TEST )
```

```
TREE RMSE Train: 4587.556685671267
TREE RMSE Test: 5763.9837632219205
```

The RMSE for the training and testing data set on predicting the loss amount are 4587.56 and 5763.98, respectively.



The average value of loss amount is $13422. LOAN refers to the credit line. The greater the value the larger the loss amount and vice versa, which makes sense as a larger loan will result in a larger loss amount when defaulted. LOAN is then used again to split the data and a few other times as well, so LOAN seems to be the most predictive in the loss amount. IMP_CLNO is the number of credit lines and used to split the data where if the user has more than 22 credit lines, the average loss amount will be higher. This makes sense because if the user has too many credit lines, in this case 22, the user could potentially have a lot of debt. IMP_DEBTINC refers to the debt-to-income ratio. If the debt-to-income ratio is high then the user may not be able to pay their bills and thus take out a larger loan and risk a larger loss amount. M_DEBTINC is if the debt-to-income ratio is missing. If it is missing, the user has a value of 1, resulting in a larger average loss amount. This makes sense since missing debt-to-income ratios may tend to be those with bad debt-to-income ratios that do not want to report it. IMP_MORTDUE is also used and is the current outstanding mortgage balance. The amount the user owes in their mortgage can reflect in the loan amount and thus the loss amount when the loan defaults. z_IMP_REASON_MISSING is the missing reason as to why the user wants a loan. It makes some sense since people who are having financial troubles may not want to record it. This attribute split was if the reason was missing then the loss amount would be less than if the reason was not missing.

```
feature_cols = list( X.columns.values )
vars_tree_amt = getTreeVars( amt_m01_Tree, feature_cols )
tree.export_graphviz(amt_m01_Tree,out_file='tree_a.txt',filled=True, rounded=True, feature_names = feature_cols, impurity=False, precision=0  )

#the variables that are determining the damage amount
for i in vars_tree_amt :
    print(i)
```

```
LOAN
z_IMP_REASON_MISSING
IMP_MORTDUE
IMP_CLNO
M_DEBTINC
IMP_DEBTINC
```

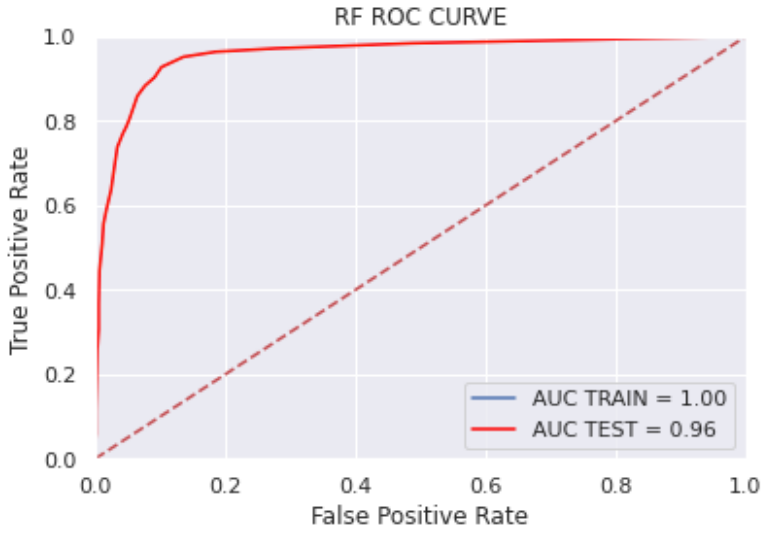
The variables LOAN, z_IMP_REASON_MISSING, IMP_MORTDUE, IMP_CLNO, M_DEBTINC, IMP_DEBTINC are the most predictive of loss amount.

Random Forest:

```
print("RANDOM FOREST\n")
print("Probability of loan default")
print("Accuracy Train:",metrics.accuracy_score(Y_train[TARGET_FLAG], Y_Pred_train))
print("Accuracy Test:",metrics.accuracy_score(Y_test[TARGET_FLAG], Y_Pred_test))
```

```
RANDOM FOREST

Probability of loan default
Accuracy Train: 0.9991610738255033
Accuracy Test: 0.912751677852349
```



The Random Forest area under the curve for the testing data set, 0.96, is greater than that of the decision tree, 0.83. The Random Forest test data had a higher accuracy than that of the Decision Tree test data. The Random Forest did better in predicting the loan default probability.

```
feature_cols = list( X.columns.values )
vars_RF_flag = getEnsembleTreeVars( fm01_RF, feature_cols )

for i in vars_RF_flag :
    print( i )
```

```
('M_DEBTINC', 100)
('IMP_DEBTINC', 80)
('IMP_CLAGE', 44)
('IMP_DELINQ', 43)
('IMP_VALUE', 42)
('LOAN', 36)
('IMP_CLNO', 33)
('IMP_MORTDUE', 32)
('IMP_YOJ', 29)
('IMP_DEROG', 26)
```

The variables that were included in the Random Forest to predict loan default are listed above. The same variables are used in the Decision Tree to predict loan default probability except for LOAN and IMP_CLNO. But these two variables were used in the Decision Tree to predict loss amount. A larger loan amount and number of credit lines is riskier so that user probably has a higher chance of having their loan default.

```
RMSE_TRAIN = math.sqrt( metrics.mean_squared_error(Z_train[TARGET_LOSS], Z_Pred_train))
RMSE_TEST = math.sqrt( metrics.mean_squared_error(Z_test[TARGET_LOSS], Z_Pred_test))

print("RF RMSE Train:", RMSE_TRAIN )
print("RF RMSE Test:", RMSE_TEST )
```

```
RF RMSE Train: 1224.2045361008875
RF RMSE Test: 3261.0178490237076
```

The RMSE for the training and testing data set on predicting the loss amount are 1224.20 and 3261.02, respectively. The training and testing data did well in accuracy. The testing data using the Random Forest model was more accurate than when using the Decision Tree.

```
feature_cols = list( X.columns.values )
vars_RF_amt = getEnsembleTreeVars( amt_m01_RF, feature_cols )

for i in vars_RF_amt :
    print( i )
```

```
('LOAN', 100)
('IMP_CLNO', 12)
('IMP_DEBTINC', 5)
```

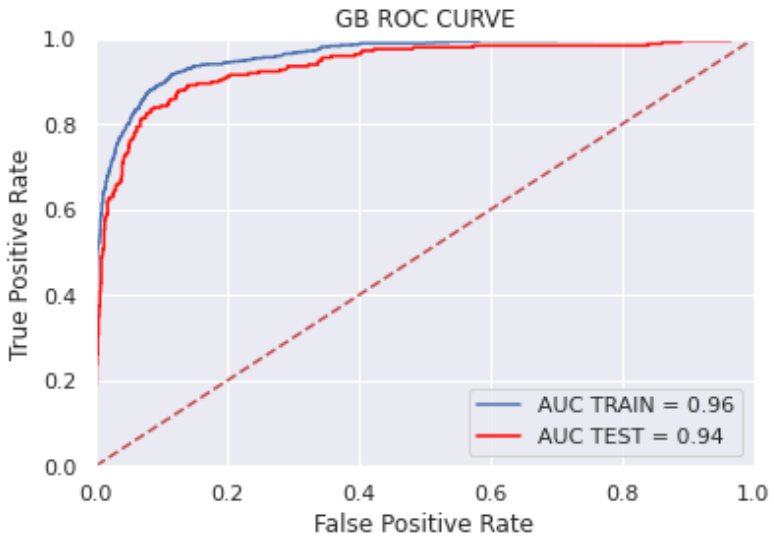
LOAN, IMP_CLNO, and IMP_DEBTINC were included in the Random Forest to predict loss amount. LOAN was the most used and predictive in the regressor. The use of LOAN and IMP_CLNO make sense in predicting loss amount since they reflect the size of the loan. IMP_DEBTINC also makes sense since a user with high debt-to-income ratio may not be able to pay their bills so it reflects that the user may be taking larger loans.

Gradient Boosting:

```
print("GRADIENT BOOSTING\n")
print("Probability of default")
print("Accuracy Train:",metrics.accuracy_score(Y_train[TARGET_FLAG], Y_Pred_train))
print("Accuracy Test:",metrics.accuracy_score(Y_test[TARGET_FLAG], Y_Pred_test))
```

```
GRADIENT BOOSTING

Probability of default
Accuracy Train: 0.9236577181208053
Accuracy Test: 0.9035234899328859
```



The test data set does well so it has high accuracy. Its area under the curve is 0.94, so it does better than the Decision Tree test data set. But the Random Forest test data does better than the Gradient Boosting test data.

```
feature_cols = list( X.columns.values )
vars_GB_flag = getEnsembleTreeVars( fm01_GB, feature_cols )

for i in vars_GB_flag :
    print(i)
```

```
('M_DEBTINC', 100)
('IMP_DEBTINC', 29)
('IMP_DELINQ', 19)
('IMP_CLAGE', 13)
```

The variables used in the Gradient Boosting model to predict the probability of a loan defaulting are the following: M_DEBTINC, IMP_DEBTINC, IMP_DELINQ, IMP_CLAGE. These variables were also used in the Decision Tree and Random Forest Classifier models to predict loan default probability.

```
RMSE_TRAIN = math.sqrt( metrics.mean_squared_error(Z_train[TARGET_LOSS], Z_Pred_train))
RMSE_TEST = math.sqrt( metrics.mean_squared_error(Z_test[TARGET_LOSS], Z_Pred_test))

print("GB RMSE Train:", RMSE_TRAIN )
print("GB RMSE Test:", RMSE_TEST )

RMSE_GB = RMSE_TEST
```

```
GB RMSE Train: 1218.1569933684325
GB RMSE Test: 2531.037788531138
```

The RMSE for the training and testing data using the Gradient Boosting model to predict loss amount are 1218.16 and 2531.04 respectively. The training and testing data did well in accuracy. The accuracy for the testing data for Gradient Boosting is better than the accuracy for the testing data for Random Forest and Decision Tree.

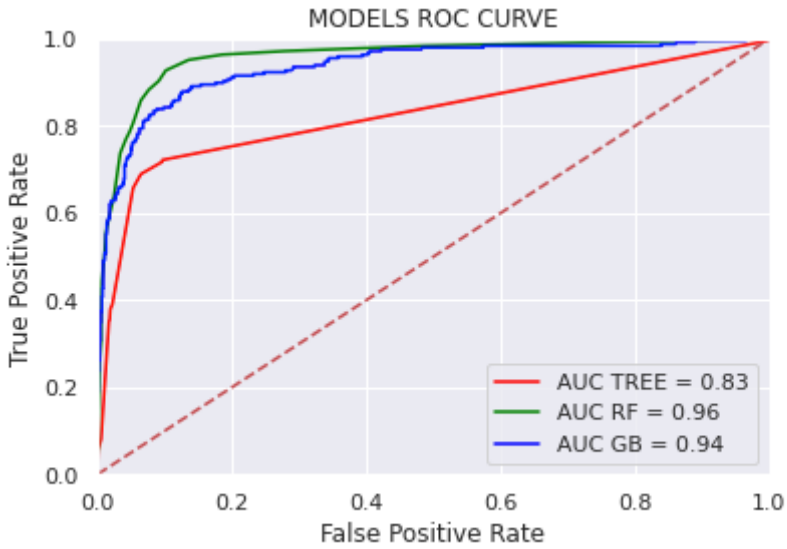
```
feature_cols = list( X.columns.values )
vars_GB_amt = getEnsembleTreeVars( amt_m01_GB, feature_cols )


for i in vars_GB_amt :
    print(i)
```

```
('LOAN', 100)
('IMP_CLNO', 14)
('IMP_DEBTINC', 5)
('M_DEBTINC', 5)
```

LOAN, IMP_CLNO, IMP_DEBTINC, M_DEBTINC are used in the Gradient Boosting model to predict loss amount. Similarly, to the other models, these variables make sense to be used to predict loss amount and reflects the loan size or assumptions of the loan size.

Comparison of the Three Models:



Random Forest is the most accurate, but Gradient Boosting is a close second, for predicting crash probability.

```
print("Root Mean Square Average For Loss Amounts")
print("TREE", RMSE_TREE)
print("RF", RMSE_RF)
print("GB", RMSE_GB)
```

```
Root Mean Square Average For Loss Amounts
TREE 5763.9837632219205
RF 3261.0178490237076
GB 2531.037788531138
```

Gradient Boosting has the lowest the Root Mean Square and thus, the most accurate in predicting the loss amount.

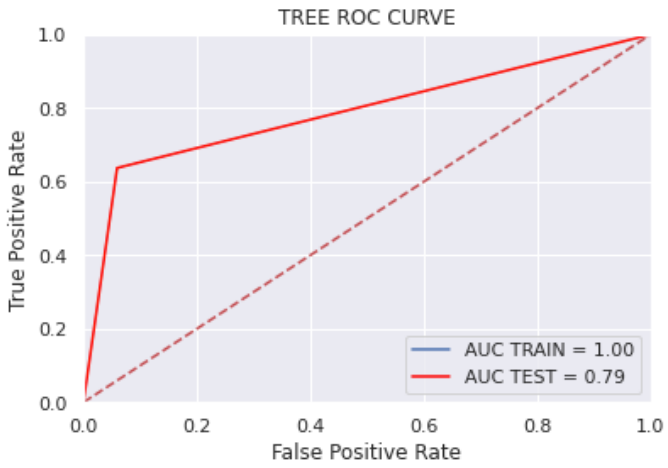
Bingo Bonus

Explore different parameters for the Decision Trees, Random Forests, and Gradient Boosting. Briefly discuss whether they had an effect on the results.

- Decision Tree Classifier – change max_depth

Let Python decide max_depth:

```
Probability of loan default
Accuracy Train: 1.0
Accuracy Test: 0.87751677852349
```




```
for i in vars_tree_flag :
    print(i)
```

```
LOAN
z_IMP_REASON_HomeImp
z_IMP_REASON_MISSING
z_IMP_JOB_Mgr
z_IMP_JOB_Office
z_IMP_JOB_Other
z_IMP_JOB_ProfExe
z_IMP_JOB_Sales
z_IMP_JOB_Self
M_MORTDUE
IMP_MORTDUE
M_VALUE
IMP_VALUE
M_YOJ
IMP_YOJ
M_DEROG
IMP_DEROG
IMP_DELINQ
M_CLAGE
IMP_CLAGE
M_NINQ
IMP_NINQ
M_CLNO
IMP_CLNO
M_DEBTINC
IMP_DEBTINC
```

The accuracy for the training data is perfect, so there is a potential it overfit the training data. The large number of variables used to split the data shows the likelihood of overfitting. The accuracy for the test data did not do better than the accuracy of the original max_depth = 4 version, 0.89. Similarly, the area under the curve for the test data set is less than that of the original version, so the original version is more accurate and is a better predictor for the probability of loan default.

Accuracy of different max_depth:

```
#max_depth = 2
fm01_Tree = tree.DecisionTreeClassifier( max_depth = 2 )
fm01_Tree = fm01_Tree.fit( X_train, Y_train[ TARGET_FLAG ] )

Y_Pred_train = fm01_Tree.predict(X_train)
Y_Pred_test = fm01_Tree.predict(X_test)

print("Probability of loan default")
print("Accuracy Train:",metrics.accuracy_score(Y_train[TARGET_FLAG], Y_Pred_train))
print("Accuracy Test:",metrics.accuracy_score(Y_test[TARGET_FLAG], Y_Pred_test))
```

```
Probability of loan default
Accuracy Train: 0.8670302013422819
Accuracy Test: 0.8481543624161074
```

```
#max_depth = 3
fm01_Tree = tree.DecisionTreeClassifier( max_depth = 3 )
fm01_Tree = fm01_Tree.fit( X_train, Y_train[ TARGET_FLAG ] )

Y_Pred_train = fm01_Tree.predict(X_train)
Y_Pred_test = fm01_Tree.predict(X_test)

print("Probability of loan default")
print("Accuracy Train:",metrics.accuracy_score(Y_train[TARGET_FLAG], Y_Pred_train))
print("Accuracy Test:",metrics.accuracy_score(Y_test[TARGET_FLAG], Y_Pred_test))
```

```
Probability of loan default
Accuracy Train: 0.8873741610738255
Accuracy Test: 0.8825503355704698
```

```
#max_depth = 4
fm01_Tree = tree.DecisionTreeClassifier( max_depth = 4 )
fm01_Tree = fm01_Tree.fit( X_train, Y_train[ TARGET_FLAG ] )

Y_Pred_train = fm01_Tree.predict(X_train)
Y_Pred_test = fm01_Tree.predict(X_test)

print("Probability of loan default")
print("Accuracy Train:",metrics.accuracy_score(Y_train[TARGET_FLAG], Y_Pred_train))
print("Accuracy Test:",metrics.accuracy_score(Y_test[TARGET_FLAG], Y_Pred_test))
```

```
Probability of loan default
Accuracy Train: 0.8928271812080537
Accuracy Test: 0.886744966442953
```

```
#max_depth = 5
fm01_Tree = tree.DecisionTreeClassifier( max_depth = 5 )
fm01_Tree = fm01_Tree.fit( X_train, Y_train[ TARGET_FLAG ] )

Y_Pred_train = fm01_Tree.predict(X_train)
Y_Pred_test = fm01_Tree.predict(X_test)

print("Probability of loan default")
print("Accuracy Train:",metrics.accuracy_score(Y_train[TARGET_FLAG], Y_Pred_train))
print("Accuracy Test:",metrics.accuracy_score(Y_test[TARGET_FLAG], Y_Pred_test))
```

```
Probability of loan default
Accuracy Train: 0.8974412751677853
Accuracy Test: 0.8875838926174496
```

```
#max_depth = 6
fm01_Tree = tree.DecisionTreeClassifier( max_depth = 6 )
fm01_Tree = fm01_Tree.fit( X_train, Y_train[ TARGET_FLAG ] )

Y_Pred_train = fm01_Tree.predict(X_train)
Y_Pred_test = fm01_Tree.predict(X_test)

print("Probability of loan default")
print("Accuracy Train:",metrics.accuracy_score(Y_train[TARGET_FLAG], Y_Pred_train))
print("Accuracy Test:",metrics.accuracy_score(Y_test[TARGET_FLAG], Y_Pred_test))
```

```
Probability of loan default
Accuracy Train: 0.902265100671141
Accuracy Test: 0.889261744966443
```

```
#max_depth = 7
fm01_Tree = tree.DecisionTreeClassifier( max_depth = 7 )
fm01_Tree = fm01_Tree.fit( X_train, Y_train[ TARGET_FLAG ] )

Y_Pred_train = fm01_Tree.predict(X_train)
Y_Pred_test = fm01_Tree.predict(X_test)

print("Probability of loan default")
print("Accuracy Train:",metrics.accuracy_score(Y_train[TARGET_FLAG], Y_Pred_train))
print("Accuracy Test:",metrics.accuracy_score(Y_test[TARGET_FLAG], Y_Pred_test))
```

```
Probability of loan default
Accuracy Train: 0.910234899328859
Accuracy Test: 0.8833892617449665
```

```
#max_depth = 10
fm01_Tree = tree.DecisionTreeClassifier( max_depth = 10 )
fm01_Tree = fm01_Tree.fit( X_train, Y_train[ TARGET_FLAG ] )

Y_Pred_train = fm01_Tree.predict(X_train)
Y_Pred_test = fm01_Tree.predict(X_test)

print("Probability of loan default")
print("Accuracy Train:",metrics.accuracy_score(Y_train[TARGET_FLAG], Y_Pred_train))
print("Accuracy Test:",metrics.accuracy_score(Y_test[TARGET_FLAG], Y_Pred_test))
```

```
Probability of loan default
Accuracy Train: 0.9305788590604027
Accuracy Test: 0.87751677852349
```

The test data set did the best with a max_depth = 6. Adding more depth than 6, the accuracy for the test set started decreasing. Compared to the original model where I used max_depth = 4, this version with max_depth = 6 is more accurate in predicting loan default of the test set.

- Decision Tree & Random Forest Classifier – changing criterion = “entropy”

The default for the criterion parameter is ‘Gini,’ which measured the quality of the split using Gini impurity to find the probability of misclassifying an observation. For this version, I ran the Decision Tree as well as the Random Forest Classifier with the criterion parameter “entropy,” which will measure the quality of the split using information gain.

```
fm01_Tree = tree.DecisionTreeClassifier( criterion='entropy', max_depth=4 )
fm01_Tree = fm01_Tree.fit( X_train, Y_train[ TARGET_FLAG ] )

Y_Pred_train = fm01_Tree.predict(X_train)
Y_Pred_test = fm01_Tree.predict(X_test)

print("Probability of loan default")
print("Accuracy Train:",metrics.accuracy_score(Y_train[TARGET_FLAG], Y_Pred_train))
print("Accuracy Test:",metrics.accuracy_score(Y_test[TARGET_FLAG], Y_Pred_test))
```

```
Probability of loan default
Accuracy Train: 0.8886325503355704
Accuracy Test: 0.8825503355704698
```

The above shows the accuracy for the training and testing data set using the Decision Tree model. The accuracy for the testing data decreased from the original model 0.8867 to the new model 0.8826. The Gini criterion gives a more accurate probability of loan default than the entropy criterion.

```
fm01_RF = RandomForestClassifier( criterion = 'entropy', n_estimators = 25, random_state=1 )
fm01_RF = fm01_RF.fit( X_train, Y_train[ TARGET_FLAG ] )

Y_Pred_train = fm01_RF.predict(X_train)
Y_Pred_test = fm01_RF.predict(X_test)

print("RANDOM FOREST\n")
print("Probability of loan default")
print("Accuracy Train:",metrics.accuracy_score(Y_train[TARGET_FLAG], Y_Pred_train))
print("Accuracy Test:",metrics.accuracy_score(Y_test[TARGET_FLAG], Y_Pred_test))
```

```
RANDOM FOREST

Probability of loan default
Accuracy Train: 0.9997902684563759
Accuracy Test: 0.912751677852349
```

There was no difference in the accuracy of the test sets with criterion = ‘entropy’ or ‘Gini.’

- Decision Tree Classifier – changing splitter = ‘random’

The default for the splitter parameter is “best” as the strategy to split at each node. This refers to the choosing the best split while “random” will choose the best random split.

```
fm01_Tree = tree.DecisionTreeClassifier( splitter = 'random', max_depth=4 )
fm01_Tree = fm01_Tree.fit( X_train, Y_train[ TARGET_FLAG ] )

Y_Pred_train = fm01_Tree.predict(X_train)
Y_Pred_test = fm01_Tree.predict(X_test)

print("Probability of loan default")
print("Accuracy Train:",metrics.accuracy_score(Y_train[TARGET_FLAG], Y_Pred_train))
print("Accuracy Test:",metrics.accuracy_score(Y_test[TARGET_FLAG], Y_Pred_test))
```

```
Probability of loan default
Accuracy Train: 0.8586409395973155
Accuracy Test: 0.8582214765100671
```

The splitter = ‘best’ results in a more accurate test data set on the probability of loan default than the splitter = ‘random.’