In [1]:
```
import math
import pandas as pd
import numpy as np
from operator import itemgetter
import matplotlib.pyplot as plt
import seaborn as sns
import os
from sklearn.model_selection import train_test_split
import sklearn.metrics as metrics
from sklearn import tree
from sklearn.tree import _tree
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matri
x
from mlxtend.feature_selection import SequentialFeatureSelector as
SFS
from mlxtend.plotting import plot_sequential_feature_selection as
plot_sfs
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
sns.set()
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
```

In [3]:
```
os.chdir("/Users/serrauzun/Desktop/MSDS_422_Practical")
df = pd.read_csv('HMEQ_Loss_clean.csv')
```

In [4]:
```
TARGET_F = "TARGET_BAD_FLAG"
TARGET_A = "TARGET_LOSS_AMT"
```

As our data has already gone through missing data imputation and removal of outliers we can directly start with splitting the dataset for our models

In [5]:
```
x = df.copy()
x = x.drop( TARGET_F, axis=1 )
x = x.drop( TARGET_A, axis=1 )
y = df[[TARGET_F, TARGET_A]]
```

Creating a train and test set from both x and y.

In [6]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, train_si
ze=0.8, test_size=0.2, random_state=2)
```

In [7]:
```
print("FLAG DATA")
print("TRAINING = ", x_train.shape)
print("TEST = ", x_test.shape)
```

```
FLAG DATA
TRAINING =  (4688, 20)
TEST =  (1173, 20)
```

Our training dataset has 4,688 rows and our test dataset has 1,173 rows, which is compliant with our 80/20 split above.

In [8]:
```
f = ~ y_train[TARGET_A].isna()
w_train = x_train[f].copy()
z_train = y_train[f].copy()

f = ~ y_test[TARGET_A].isna()
w_test = x_test[f].copy()
z_test = y_test[f].copy()
```

In [9]:
```
print(z_train.describe())
print(z_test.describe())
```

```
       TARGET_BAD_FLAG  TARGET_LOSS_AMT
count      4688.000000      4688.000000
mean          0.196886      2586.748933
std           0.397688      7038.433246
min           0.000000         0.000000
25%           0.000000         0.000000
50%           0.000000         0.000000
75%           0.000000         0.000000
max           1.000000     78987.000000
       TARGET_BAD_FLAG  TARGET_LOSS_AMT
count      1173.000000      1173.000000
mean          0.205456      2690.576300
std           0.404207      7049.842521
min           0.000000         0.000000
25%           0.000000         0.000000
50%           0.000000         0.000000
75%           0.000000         0.000000
max           1.000000     62463.000000
```

MODEL ACCURACY METRICS

In [10]:
```
def getProbAccuracyScores( NAME, MODEL, X, Y ) :
    pred = MODEL.predict( X )
    probs = MODEL.predict_proba( X )
    acc_score = metrics.accuracy_score(Y, pred)
    p1 = probs[:,1]
    fpr, tpr, threshold = metrics.roc_curve( Y, p1)
    auc = metrics.auc(fpr,tpr)
    return [NAME, acc_score, fpr, tpr, auc]
```

In [11]:
```
def print_ROC_Curve( TITLE, LIST ) :
    fig = plt.figure(figsize=(6,4))
    plt.title( TITLE )
    for theResults in LIST :
        NAME = theResults[0]
        fpr = theResults[2]
        tpr = theResults[3]
        auc = theResults[4]
        theLabel = "AUC " + NAME + ' %0.2f' % auc
        plt.plot(fpr, tpr, label = theLabel )
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
```

In [12]:
```
def print_Accuracy( TITLE, LIST ) :
    print( TITLE )
    print( "======" )
    for theResults in LIST :
        NAME = theResults[0]
        ACC = theResults[1]
        print( NAME, " = ", ACC )
    print( "------\n\n" )
```

In [13]:
```
def getAmtAccuracyScores( NAME, MODEL, X, Y ) :
    pred = MODEL.predict( X )
    MEAN = Y.mean()
    RMSE = math.sqrt( metrics.mean_squared_error( Y, pred))
    return [NAME, RMSE, MEAN]
```

** DECISION TREE **

In [14]:
```
def getTreeVars( TREE, varNames ) :
    tree_ = TREE.tree_
    varName = [ varNames[i] if i != _tree.TREE_UNDEFINED else "und
efined!" for i in tree_.feature ]

    nameSet = set()
    for i in tree_.feature :
        if i != _tree.TREE_UNDEFINED :
            nameSet.add( i )
    nameList = list( nameSet )
    parameter_list = list()
    for i in nameList :
        parameter_list.append( varNames[i] )
    return parameter_list
```

---- DEFAULT PROBABILITY ----

In [15]:
```
WHO = "TREE"
```

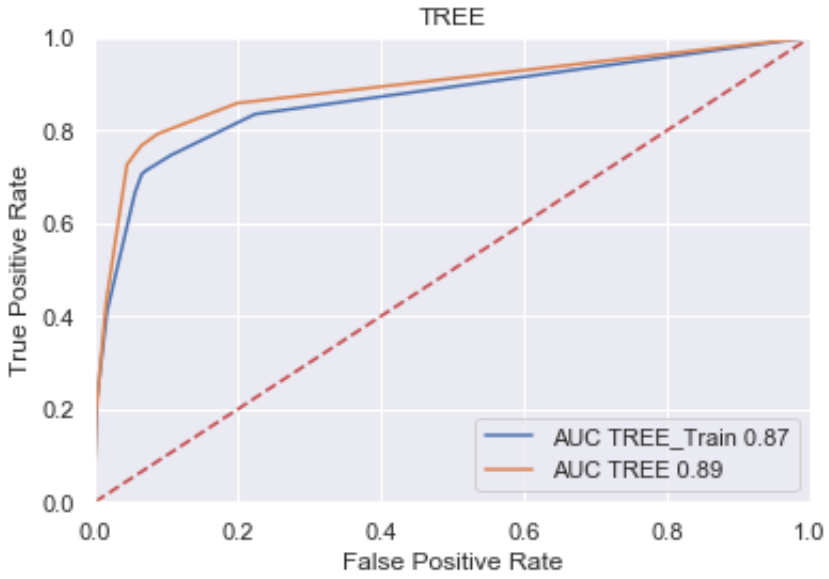
In [16]:
```
DFLT = tree.DecisionTreeClassifier(max_depth=4 )
DFLT = DFLT.fit(x_train, y_train[TARGET_F])
```

In [17]:
```
TRAIN_DFLT = getProbAccuracyScores(WHO + "_Train", DFLT, x_train,
y_train[TARGET_F])
TEST_DFLT = getProbAccuracyScores(WHO, DFLT, x_test, y_test[TARGET
_F])
```

In [18]:
```
print_Accuracy(WHO + " CLASSIFICATION ACCURACY", [TRAIN_DFLT, TEST
_DFLT])
```

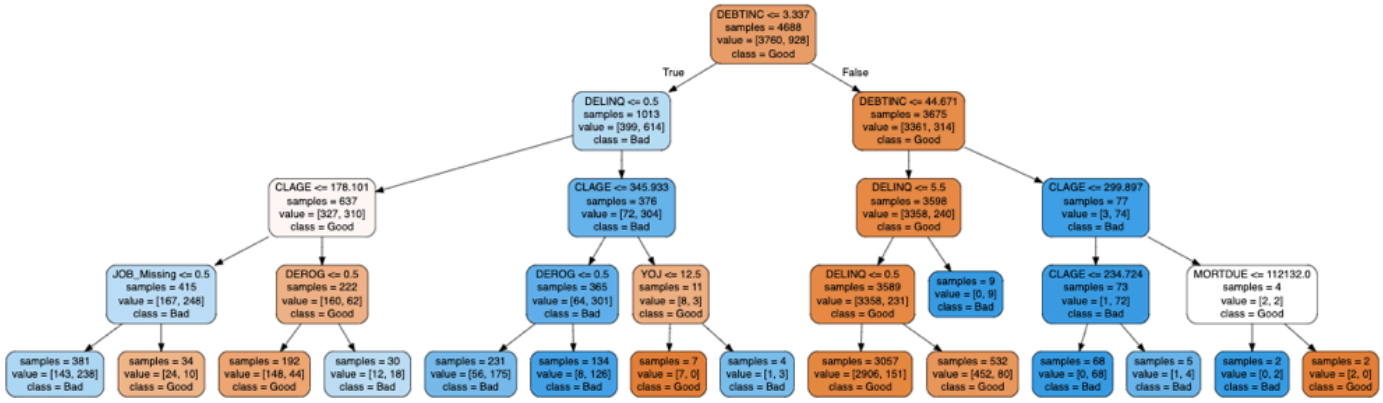
```
TREE CLASSIFICATION ACCURACY
======
TREE_Train  =  0.8892918088737202
TREE  =  0.9036658141517476
------
```

In [19]:
```
print_ROC_Curve(WHO, [TRAIN_DFLT, TEST_DFLT])
```



Both train and test ROC curves are curved towards the True Positive and indicates that our test data is slightly (2%) more accurate than our training data, 87% and 89% respectively.

In [20]:
```
feature_cols = list(x.columns.values)
tree.export_graphviz(DFLT,out_file='tree_f.txt',filled=True, round
ed=True, feature_names = feature_cols, impurity=False, class_names
=["Good","Bad"] )
vars_tree_flag = getTreeVars(DFLT, feature_cols)
```



The algorithm we have written have decided to split the data on DEBTINC (Debt to Income Ratio) and 3.337. The data then have incrementally been divided by DELINQ (Delinquencies on your current credit report) and CLAGE (Credit Line Age), then by DEROG (Derogatory Marks on Credit Record), YOJ (Year on Job), JOB (only records with Job as Missing information) and MORTDUE (Current Outstanding Mortgage Balance). Model's chosing of Debt to Income Ratio as the primary indicator does make sense as income is crucial in this context and DEBTINC is the main and only variable related to income. Thus, the obvious correlation between Debt to Income Ratio and the likelihood of credit default is a reasonable outcome of this model that we can accept.

Looking at the Decision Tree our algorithm has given us, we can see that person with DEBTINC less than 3.337, DELINQ less than 0.5, CLAGE less than 346, DEROG less than 0.5 are in the worst class whereas people with DEBTINC higher than 3.357 and less than 44.671, and DELINQ of less than 0.5 are the ones that are least likely to default.

In [21]:
```
print( "FLAG=",vars_tree_flag)
```

```
FLAG= ['YOJ', 'DEROG', 'DELINQ', 'CLAGE', 'DEBTINC', 'JOB_Missing
']
```

The variables listed above are the top indicators of a possible Home Equity Credit Default

---- LOSS PREDICTION ----

In [22]:
```
not_zero = y_train[TARGET_A] != 0
```

In [23]:
```
w_train = x_train[not_zero]
z_train = y_train[not_zero]
```

In [24]:
```
not_zero = y_test['TARGET_LOSS_AMT'] != 0
w_test = x_test[not_zero]
z_test = y_test[not_zero]
```

In [25]:
```
AMT = tree.DecisionTreeRegressor(max_depth= 3 )
AMT = AMT.fit(w_train, z_train[TARGET_A])
```

In [26]:
```
TRAIN_AMT = getAmtAccuracyScores(WHO + "_Train", AMT,w_train,z_tra
in[TARGET_A])
TEST_AMT = getAmtAccuracyScores(WHO, AMT, w_test, z_test[TARGET_A]
)
print_Accuracy(WHO + " RMSE ACCURACY", [TRAIN_AMT, TEST_AMT])
```

```
TREE RMSE ACCURACY
======
TREE_Train  =  5198.885686289661
TREE  =  6065.25844508915
------
```

It is a positive sign that train and test mean are close to each other. Three seems to be the ideal max_depth for Decision Tree Regressor. Having only 590 difference between test and training means, we can accept this result and continue with the model.

In [27]:
```
feature_cols = list(x.columns.values)
vars_tree_amt = getTreeVars(AMT, feature_cols)
tree.export_graphviz(AMT,out_file='tree_a.txt',filled=True, rounde
d=True, feature_names = feature_cols, impurity=False, precision=0)
```

In [28]:
```
TREE_DFLT = TEST_DFLT.copy()
TREE_AMT = TEST_AMT.copy()
```

RANDOM FOREST

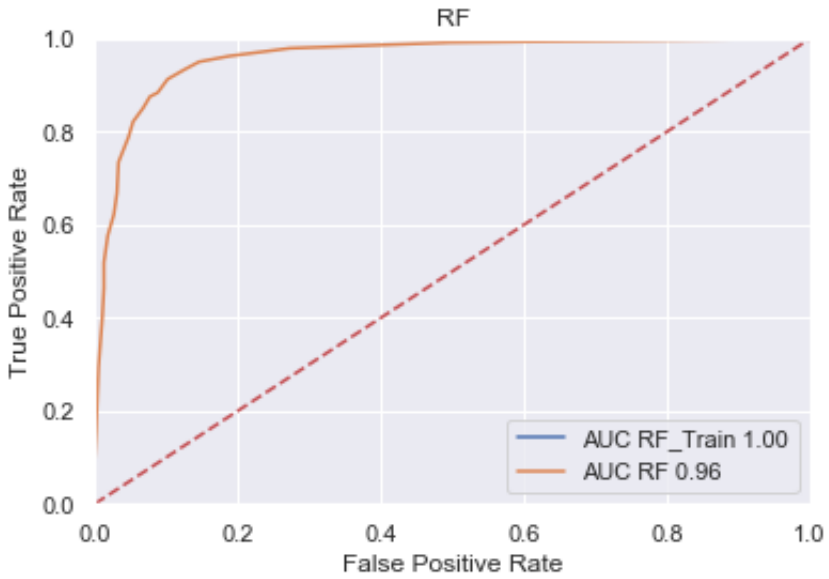
In [29]:
```
def getEnsembleTreeVars( ENSTREE, varNames ) :
    importance = ENSTREE.feature_importances_
    index = np.argsort(importance)
    theList = []
    for i in index :
        imp_val = importance[i]
        if imp_val > np.average( ENSTREE.feature_importances_ ) :
            v = int( imp_val / np.max( ENSTREE.feature_importances
_ ) * 100 )
            theList.append( ( varNames[i], v ) )
    theList = sorted(theList,key=itemgetter(1),reverse=True)
    return theList
```

In [30]:
```
WHO = "RF"
```

In [31]:
```
DFLT = RandomForestClassifier(n_estimators = 25, random_state=1 )
DFLT = DFLT.fit(x_train,y_train[TARGET_F])
```

In [32]:
```
TRAIN_DFLT = getProbAccuracyScores(WHO + "_Train", DFLT, x_train,
y_train[TARGET_F])
TEST_DFLT = getProbAccuracyScores(WHO, DFLT, x_test, y_test[TARGET
_F])
```

In [33]:
```
print_ROC_Curve(WHO, [TRAIN_DFLT, TEST_DFLT])
print_Accuracy(WHO + " CLASSIFICATION ACCURACY", [TRAIN_DFLT, TEST
_DFLT])
```




```
RF CLASSIFICATION ACCURACY
======
RF_Train  =  0.9982935153583617
RF  =  0.9190110826939472
------
```

As we had 100 decision trees through out Random Forest Classifier model, we got 100% accuracy on our training set. The training set is not the approperiate indicator of our models indicator, but our test set accuracy is also high, at 92%. Thus, we can say that the random forest classifier model is going to give use an result that is accurate over 90%.

In [34]:
```
feature_cols = list(x.columns.values )
vars_RF_flag = getEnsembleTreeVars(DFLT, feature_cols)
```

DAMAGES

In [35]:
```
AMT = RandomForestRegressor(n_estimators = 100, random_state=1)
AMT = AMT.fit(w_train, z_train[TARGET_A])
```

In [36]:
```
TRAIN_AMT = getAmtAccuracyScores(WHO + "_Train", AMT, w_train, z_t
rain[TARGET_A])
TEST_AMT = getAmtAccuracyScores(WHO, AMT, w_test, z_test[TARGET_A]
)
print_Accuracy(WHO + " RMSE ACCURACY", [TRAIN_AMT, TEST_AMT])
```

```
RF RMSE ACCURACY
======
RF_Train  =  1057.3831014358318
RF  =  4295.973809056954
------
```

In [37]:
```
feature_cols = list(x.columns.values )
vars_RF_amt = getEnsembleTreeVars(AMT, feature_cols)
```

In [38]:
```
for i in vars_RF_amt :
    print(i)
```

```
('LOAN', 100)
('CLNO', 12)
('DEBTINC', 8)
```

In [39]:
```
RF_DFLT = TEST_DFLT.copy()
RF_AMT = TEST_AMT.copy()
```

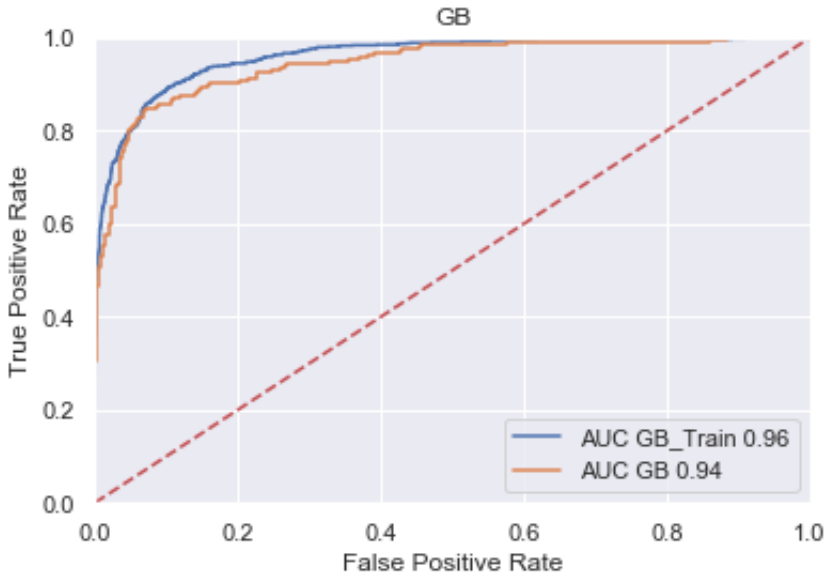
GRADIENT BOOSTING

In [40]:
```
WHO = "GB"
```

In [41]:
```
DFLT = GradientBoostingClassifier(random_state=1 )
DFLT = DFLT.fit(x_train, y_train[TARGET_F])
```

In [42]:
```
TRAIN_DFLT = getProbAccuracyScores(WHO + "_Train", DFLT, x_train,
y_train[TARGET_F])
TEST_DFLT = getProbAccuracyScores(WHO, DFLT, x_test, y_test[TARGET
_F])
```

In [43]:
```
print_ROC_Curve(WHO, [TRAIN_DFLT, TEST_DFLT])
print_Accuracy( WHO + " CLASSIFICATION ACCURACY", [TRAIN_DFLT, TES
T_DFLT])
```




```
GB CLASSIFICATION ACCURACY
======
GB_Train  =  0.925981228668942
GB  =  0.9121909633418585
------
```

In [44]:
```
feature_cols = list(x.columns.values )
vars_GB_flag = getEnsembleTreeVars(DFLT, feature_cols)
```

DAMAGES

In [45]:
```
AMT = GradientBoostingRegressor(random_state=1)
AMT = AMT.fit(w_train, z_train[TARGET_A])
```

In [46]:
```
TRAIN_AMT = getAmtAccuracyScores(WHO + "_Train", AMT, w_train, z_t
rain[TARGET_A])
TEST_AMT = getAmtAccuracyScores(WHO, AMT, w_test, z_test[TARGET_A]
)
print_Accuracy(WHO + " RMSE ACCURACY", [TRAIN_AMT, TEST_AMT])
```

```
GB RMSE ACCURACY
======
GB_Train  =  1142.0819565344898
GB  =  3149.092174701888
------
```

In [47]:
```
feature_cols = list(x.columns.values )
vars_GB_amt = getEnsembleTreeVars(AMT, feature_cols)
```

In [48]:
```
for i in vars_RF_amt :
    print(i)
```

```
('LOAN', 100)
('CLNO', 12)
('DEBTINC', 8)
```

In [49]:
```
GB_DFLT = TEST_DFLT.copy()
GB_AMT = TEST_AMT.copy()
```

Functions

In [50]:
```
def getCoefLogit( MODEL, TRAIN_DATA ) :
    varNames = list( TRAIN_DATA.columns.values )
    coef_dict = {}
    coef_dict["INTERCEPT"] = MODEL.intercept_[0]
    for coef, feat in zip(MODEL.coef_[0],varNames):
        coef_dict[feat] = coef
    print("\nDEFAULT")
    print("---------")
    print("Total Variables: ", len( coef_dict ) )
    for i in coef_dict :
        print( i, " = ", coef_dict[i]  )
```

In [51]:
```
def getCoefLinear( MODEL, TRAIN_DATA ) :
    varNames = list( TRAIN_DATA.columns.values )
    coef_dict = {}
    coef_dict["INTERCEPT"] = MODEL.intercept_
    for coef, feat in zip(MODEL.coef_,varNames):
        coef_dict[feat] = coef
    print("\nLOSS")
    print("---------")
    print("Total Variables: ", len( coef_dict ) )
    for i in coef_dict :
        print( i, " = ", coef_dict[i]  )
```

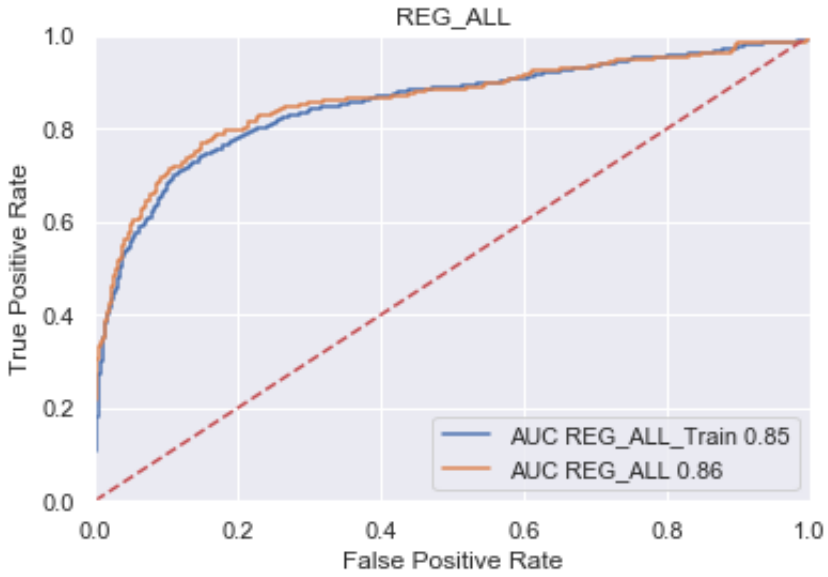
REGRESSION ALL VARIABLES

In [52]:
```
WHO = "REG_ALL"
```

In [53]:
```
DFLT = LogisticRegression(solver='newton-cg', max_iter=1000 )
DFLT = DFLT.fit(x_train, y_train[TARGET_F])
```

In [54]:
```
TRAIN_DFLT = getProbAccuracyScores(WHO + "_Train", DFLT, x_train,
y_train[TARGET_F])
TEST_DFLT = getProbAccuracyScores(WHO, DFLT, x_test, y_test[TARGET
_F])
```

In [55]:
```
print_ROC_Curve(WHO, [TRAIN_DFLT, TEST_DFLT])
print_Accuracy(WHO + " CLASSIFICATION ACCURACY", [TRAIN_DFLT, TEST
_DFLT])
```




```
REG_ALL CLASSIFICATION ACCURACY
======
REG_ALL_Train  =  0.8718003412969283
REG_ALL  =  0.8729752770673487
------
```

DAMAGES

In [56]:
```
AMT = LinearRegression()
AMT = AMT.fit(w_train, z_train[TARGET_A])
```

In [57]:
```
TRAIN_AMT = getAmtAccuracyScores(WHO + "_Train", AMT, w_train, z_t
rain[TARGET_A])
TEST_AMT = getAmtAccuracyScores(WHO, AMT, w_test, z_test[TARGET_A]
)
print_Accuracy(WHO + " RMSE ACCURACY", [TRAIN_AMT, TEST_AMT])
```

```
REG_ALL RMSE ACCURACY
======
REG_ALL_Train  =  3895.0943398137306
REG_ALL  =  4360.171040128947
------
```

In [58]:
```
varNames = list(x_train.columns.values )
```

In [59]:
```
REG_ALL_DFLT_COEF = getCoefLogit(DFLT, x_train )
REG_ALL_AMT_COEF = getCoefLinear(AMT, x_train )
```

```
DEFAULT
---------
Total Variables:  21
INTERCEPT  =  0.44644310718827385
LOAN  =  -1.32411146462894e-05
MORTDUE  =  3.5460872074646773e-06
VALUE  =  -2.94469895895604e-06
YOJ  =  -0.022035469735098064
DEROG  =  0.4929085937116025
DELINQ  =  0.6725895621850867
CLAGE  =  -0.005331700892235892
NINQ  =  0.17275356644435874
CLNO  =  0.0032879134383274233
DEBTINC  =  -0.057749704765397016
REASON_DebtCon  =  -0.10063484694262534
REASON_HomeImp  =  0.21104923357133334
REASON_Missing  =  0.1056311421333986
JOB_Mgr  =  0.1411894941141464
JOB_Missing  =  -1.1658940269519869
JOB_Office  =  -0.5002844798854287
JOB_Other  =  0.21486869213168544
JOB_ProfExe  =  -0.2497655793889872
JOB_Sales  =  1.1698359518233992
JOB_Self  =  0.6060954769192716

LOSS
---------
Total Variables:  21
INTERCEPT  =  -1568.859066279334
LOAN  =  0.7899015186066133
MORTDUE  =  -0.0003739843447146708
VALUE  =  0.005771707529919828
YOJ  =  -58.145405647435126
DEROG  =  308.1832794780266
DELINQ  =  681.9619614344859
CLAGE  =  -20.33753265959915
NINQ  =  -42.502502210186265
CLNO  =  227.91140683573275
DEBTINC  =  -72.05143453687562
REASON_DebtCon  =  1167.8544846387786
REASON_HomeImp  =  -399.3451280997374
REASON_Missing  =  -768.5093565390412
JOB_Mgr  =  -938.156509620311
JOB_Missing  =  -537.4017618870131
JOB_Office  =  -613.4736398424659
JOB_Other  =  -545.9223853098861
JOB_ProfExe  =  -1183.7929302482253
JOB_Sales  =  1709.8156903120532
JOB_Self  =  2108.9315365958487
```

In [60]:
```
REG_ALL_DFLT = TEST_DFLT.copy()
REG_ALL_AMT = TEST_AMT.copy()
```

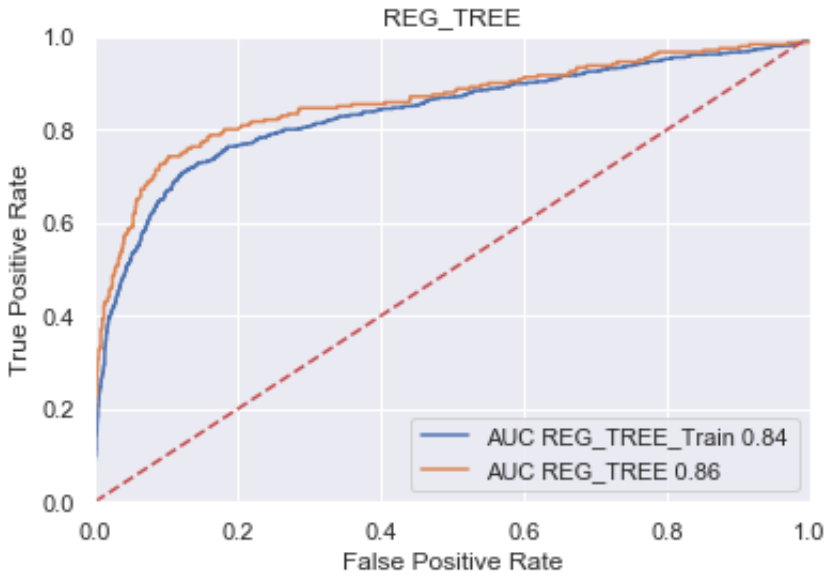
REGRESSION DECISION TREE

In [61]:
```
WHO = "REG_TREE"
```

In [62]:
```
DFLT = LogisticRegression(solver='newton-cg', max_iter=1000 )
DFLT = DFLT.fit(x_train[vars_tree_flag], y_train[TARGET_F])
```

In [63]:
```
TRAIN_DFLT = getProbAccuracyScores(WHO + "_Train", DFLT, x_train[v
ars_tree_flag], y_train[TARGET_F])
TEST_DFLT = getProbAccuracyScores(WHO, DFLT, x_test[vars_tree_flag
], y_test[TARGET_F])
```

In [64]:
```
print_ROC_Curve(WHO, [TRAIN_DFLT, TEST_DFLT])
print_Accuracy(WHO + " CLASSIFICATION ACCURACY", [TRAIN_DFLT, TEST
_DFLT])
```




```
REG_TREE CLASSIFICATION ACCURACY
======
REG_TREE_Train  =  0.8658276450511946
REG_TREE  =  0.875532821824382
------
```

DAMAGES

In [65]:
```
AMT = LinearRegression()
AMT = AMT.fit(w_train[vars_tree_amt], z_train[TARGET_A])
```

In [66]:
```
TRAIN_AMT = getAmtAccuracyScores(WHO + "_Train", AMT, w_train[vars
_tree_amt], z_train[TARGET_A])
TEST_AMT = getAmtAccuracyScores(WHO, AMT, w_test[vars_tree_amt], z
_test[TARGET_A])
print_Accuracy(WHO + " RMSE ACCURACY", [TRAIN_AMT, TEST_AMT])
```

```
REG_TREE RMSE ACCURACY
======
REG_TREE_Train  =  4542.151467044364
REG_TREE  =  5128.335320171946
------
```

In [67]:
```
varNames = list(x_train.columns.values )
```

In [68]:
```
REG_TREE_DFLT_COEF = getCoefLogit(DFLT, x_train[vars_tree_flag])
REG_TREE_AMT_COEF = getCoefLinear(AMT, x_train[vars_tree_amt])
```

```
DEFAULT
---------
Total Variables:  7
INTERCEPT  =  0.6724781203382588
YOJ  =  -0.02150472468821733
DEROG  =  0.53635073623535
DELINQ  =  0.6650022598622028
CLAGE  =  -0.006143139272392336
DEBTINC  =  -0.059728058070994725
JOB_Missing  =  -1.1504563865236825

LOSS
---------
Total Variables:  4
INTERCEPT  =  -4366.736865282563
LOAN  =  0.8138478478454072
CLNO  =  242.00912575271602
DEBTINC  =  -69.556039379462
```

The Decision Tree Regression Analysis we see that customers that have DEROG (Derogatory Marks on Credit Record) and DELINQ (Delinquencies on your current credit report) are more likely to default on their home equity credit. This result makes sense are both of the variables are things that will negatively impact a credit score, thus person's trustability. In addition, we can see that high DEBTINC (debt to income ratio) and CLAGE (credit line age) make a person less likely to default on their home equity credit.

Our model also shows us that people with higher CLNO (Number of credit lines one have) and LOAN (HMEQ Credit Line) are more likely to cause a higher loss amount on their defaulted credit. On the other hand, higher DEBTINC (debt to income ratio) are less likely to have a high loss amount, which makes sense.

In [69]:
```
REG_TREE_DFLT = TEST_DFLT.copy()
REG_TREE_AMT = TEST_AMT.copy()
```

REGRESSION RANDOM FOREST

In [70]:
```
WHO = "REG_RF"
```

In [71]:
```
RF_flag = []
for i in vars_RF_flag :
    print(i)
    theVar = i[0]
    RF_flag.append(theVar)
```

```
('DEBTINC', 100)
('CLAGE', 33)
('LOAN', 31)
('DELINQ', 31)
('VALUE', 30)
('CLNO', 26)
('MORTDUE', 24)
('YOJ', 19)
```

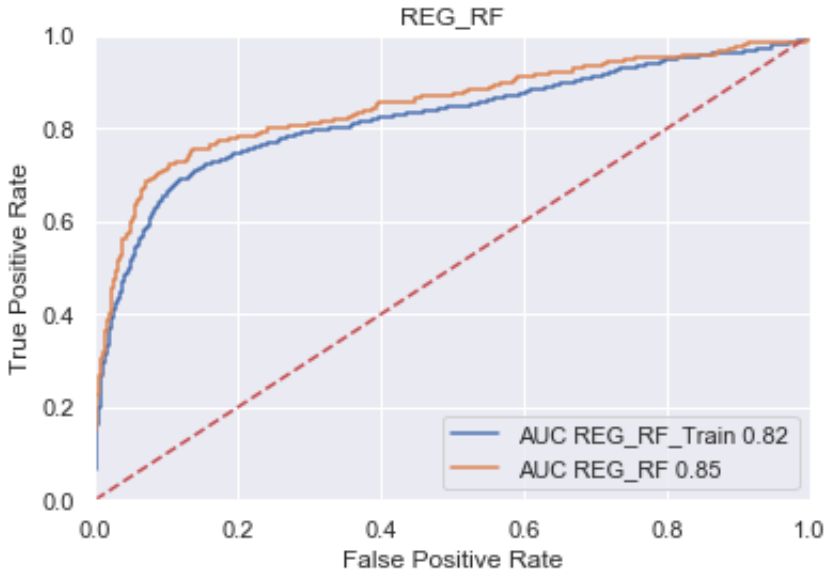
In [72]:
```
RF_amt = []
for i in vars_RF_amt :
    print(i)
    theVar = i[0]
    RF_amt.append(theVar)
```

```
('LOAN', 100)
('CLNO', 12)
('DEBTINC', 8)
```

In [73]:
```
DFLT = LogisticRegression(solver='newton-cg', max_iter=1000 )
DFLT = DFLT.fit(x_train[RF_flag],y_train[TARGET_F])
```

In [74]:
```
TRAIN_DFLT = getProbAccuracyScores(WHO + "_Train", DFLT, x_train[R
F_flag], y_train[TARGET_F] )
TEST_DFLT = getProbAccuracyScores(WHO, DFLT, x_test[RF_flag], y_te
st[TARGET_F])

print_ROC_Curve(WHO, [TRAIN_DFLT, TEST_DFLT])
print_Accuracy(WHO + " CLASSIFICATION ACCURACY", [TRAIN_DFLT, TEST
_DFLT])
```




```
REG_RF CLASSIFICATION ACCURACY
======
REG_RF_Train  =  0.8611348122866894
REG_RF  =  0.8729752770673487
------
```

DAMAGES

In [75]:
```
AMT = LinearRegression()
AMT = AMT.fit(w_train[RF_amt], z_train[TARGET_A])
```

In [76]:
```
TRAIN_AMT = getAmtAccuracyScores(WHO + "_Train", AMT, w_train[RF_a
mt], z_train[TARGET_A])
TEST_AMT = getAmtAccuracyScores(WHO, AMT, w_test[RF_amt], z_test[T
ARGET_A])
```

In [77]:
```
print_Accuracy(WHO + " RMSE ACCURACY", [TRAIN_AMT, TEST_AMT])
```

```
REG_RF RMSE ACCURACY
======
REG_RF_Train  =  4542.151467044364
REG_RF  =  5128.335320171946
------
```

In [78]:
```
REG_RF_DFLT_COEF = getCoefLogit(DFLT, x_train[RF_flag])
REG_RF_AMT_COEF = getCoefLinear(AMT, x_train[RF_amt])
```

```
DEFAULT
---------
Total Variables:  9
INTERCEPT  =  0.927518776374913
DEBTINC  =  -0.06016733448628527
CLAGE  =  -0.006559294022068178
LOAN  =  -7.590294793160134e-06
DELINQ  =  0.7130590056504256
VALUE  =  -3.4149333329619286e-06
CLNO  =  0.008910293673967038
MORTDUE  =  2.51054977905727e-06
YOJ  =  -0.021386157223213936

LOSS
---------
Total Variables:  4
INTERCEPT  =  -4366.736865282563
LOAN  =  0.8138478478454072
CLNO  =  242.00912575271602
DEBTINC  =  -69.556039379462
```

With our Random Forest Regression Model we see results that are similar and parallel with Decision Tree regression model. People with high DELINQ and who have MORT_DUE, and have multiple credit lines are the ones that are likely to get their credit default. Esspecially the higher people have a mortgage amount due, the more likely they are to default. On the other hand we see that people with high DEBTINC, CLAGE, House value (VALUE) and LOAN(credit line) are less likely to default. These results correspond with the previous results and makes sense.

Regarding the LOSS amount, we see that the results are accurate and consistent. People with high LOAN and CLNO are more likely to lose a higher amount on their defaulted credit while people with high DEBTINC are likely to lose a lesser amount on their loan when it gets default.

In [79]:
```
REG_RF_DFLT = TEST_DFLT.copy()
REG_RF_AMT = TEST_AMT.copy()
```

REGRESSION GRADIENT BOOSTING

In [80]:
```
WHO = "REG_GB"
```

In [81]:
```
GB_flag = []
for i in vars_GB_flag :
    print(i)
    theVar = i[0]
    GB_flag.append(theVar)
```

```
('DEBTINC', 100)
('DELINQ', 17)
('CLAGE', 14)
```

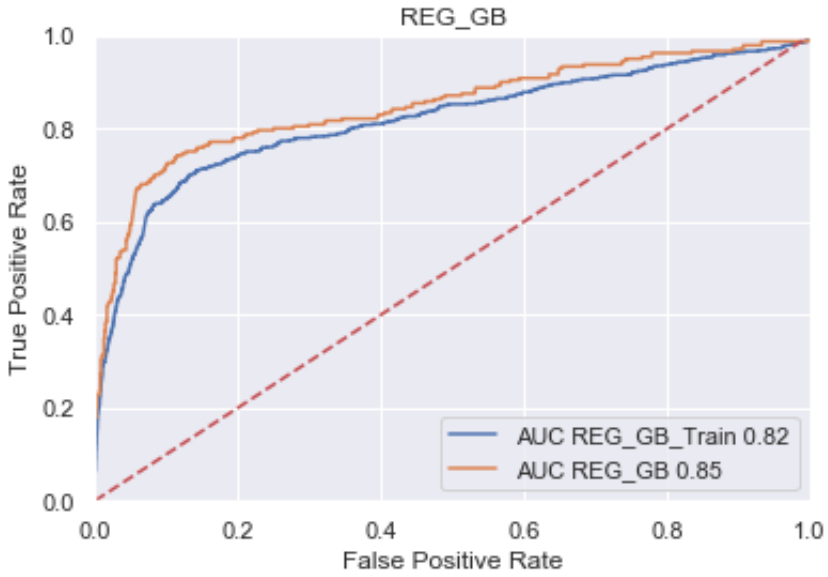
In [82]:
```
GB_amt = []
for i in vars_GB_amt :
    print(i)
    theVar = i[0]
    GB_amt.append(theVar)
```

```
('LOAN', 100)
('CLNO', 15)
('DEBTINC', 9)
```

In [83]:
```
DFLT = LogisticRegression(solver='newton-cg', max_iter=1000)
DFLT = DFLT.fit(x_train[GB_flag], y_train[TARGET_F])
```

In [84]:
```
TRAIN_DFLT = getProbAccuracyScores(WHO + "_Train", DFLT, x_train[G
B_flag], y_train[TARGET_F])
TEST_DFLT = getProbAccuracyScores(WHO, DFLT, x_test[GB_flag], y_te
st[TARGET_F])
```

In [85]:
```
print_ROC_Curve(WHO, [TRAIN_DFLT, TEST_DFLT])
print_Accuracy(WHO + " CLASSIFICATION ACCURACY", [TRAIN_DFLT, TEST
_DFLT])
```




```
REG_GB CLASSIFICATION ACCURACY
======
REG_GB_Train  =  0.8624146757679181
REG_GB  =  0.8678601875532822
------
```

DAMAGES

In [86]:
```
AMT = LinearRegression()
AMT = AMT.fit(w_train[GB_amt], z_train[TARGET_A])
```

In [87]:
```
TRAIN_AMT = getAmtAccuracyScores(WHO + "_Train", AMT, w_train[GB_a
mt], z_train[TARGET_A])
TEST_AMT = getAmtAccuracyScores(WHO, AMT, w_test[GB_amt], z_test[T
ARGET_A])
print_Accuracy(WHO + " RMSE ACCURACY", [TRAIN_AMT, TEST_AMT])
```

```
REG_GB RMSE ACCURACY
======
REG_GB_Train  =  4542.151467044364
REG_GB  =  5128.335320171946
------
```

In [88]:
```
REG_GB_DFLT_COEF = getCoefLogit(DFLT, x_train[GB_flag])
REG_GB_AMT_COEF = getCoefLinear(AMT, x_train[GB_amt])
```

```
DEFAULT
---------
Total Variables:  4
INTERCEPT  =  0.7033755666581065
DEBTINC  =  -0.060196123538218 22
DELINQ  =  0.7193263565314694
CLAGE  =  -0.006878205269428091

LOSS
---------
Total Variables:  4
INTERCEPT  =  -4366.736865282563
LOAN  =  0.8138478478454072
CLNO  =  242.00912575271602
DEBTINC  =  -69.556039379462
```

Our gradient boosting regression model have narrowed down the variables that are the most critical in predicting credit default down to three. The results are consistent with our previous models. We see that people with high Debt to income ratio and credit line age are less likely to default while people with high delinquencies on their credit lines are more likely.

The Loss Amount results remain consistent and reasonable.

In [89]:
```
REG_GB_DFLT = TEST_DFLT.copy()
REG_GB_AMT = TEST_AMT.copy()
```

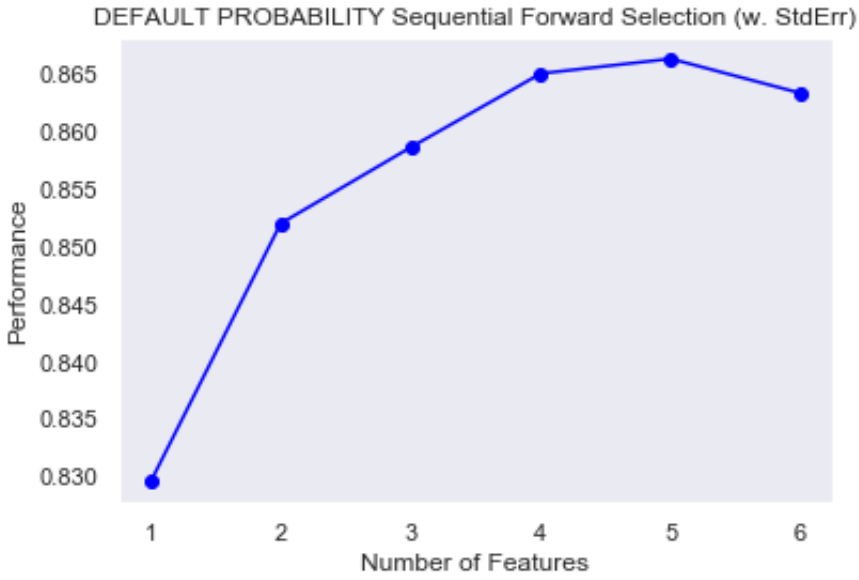
REGRESSION STEPWISE

In [90]:
```
u_train = x_train[vars_tree_flag]
stepVarNames = list(u_train.columns.values)
maxCols = u_train.shape[1]
```

In [91]:
```
sfs = SFS( LogisticRegression( solver='newton-cg', max_iter=100 ),
           k_features=( 1, maxCols ),
           forward=True,
           floating=False,
           cv=3
           )
sfs.fit(u_train.values, y_train[TARGET_F].values)
```

Out[91]:
```
SequentialFeatureSelector(clone_estimator=True, cv=3,
                          estimator=LogisticRegression(C=1.0, cla
ss_weight=None,
                                                       dual=False
,
                                                       fit_interc
ept=True,
                                                       intercept_
scaling=1,
                                                       l1_ratio=N
one,
                                                       max_iter=1
00,
                                                       multi_clas
s='warn',
                                                       n_jobs=Non
e,
                                                       penalty='l
2',
                                                       random_sta
te=None,
                                                       solver='ne
wton-cg',
                                                       tol=0.0001
, verbose=0,
                                                       warm_start
=False),
                          fixed_features=None, floating=False, fo
rward=True,
                          k_features=(1, 6), n_jobs=1, pre_dispat
ch='2*n_jobs',
                          scoring=None, verbose=0)
```

In [92]:
```
theFigure = plot_sfs(sfs.get_metric_dict(), kind=None)
plt.title('DEFAULT PROBABILITY Sequential Forward Selection (w. StdErr)')
plt.grid()
plt.show()
```



The Sequential Forward Selection graph we plotted for the default probability shows that the ideal number of variables to include for our analysis is 4 or 5.

In [93]:
```
dfm = pd.DataFrame.from_dict(sfs.get_metric_dict()).T
dfm = dfm[['feature_names', 'avg_score']]
dfm.avg_score = dfm.avg_score.astype(float)
```

In [94]:
```
print(" ..................... ")
maxIndex = dfm.avg_score.argmax()
print("argmax")
print( dfm.iloc[ maxIndex, ] )
print(" ..................... ")
```

```
 .....................
argmax
feature_names    (0, 1, 2, 3, 4, 5)
avg_score                  0.863269
Name: 6, dtype: object
 .....................
```

In [95]:
```
stepVars = dfm.iloc[maxIndex, ]
stepVars = stepVars.feature_names
print(stepVars)
```

```
('0', '1', '2', '3', '4', '5')
```

In [96]:
```
finalStepVars = []
for i in stepVars :
    index = int(i)
    try :
        theName = stepVarNames[ index ]
        finalStepVars.append( theName )
    except :
        pass

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

```
YOJ
DEROG
DELINQ
CLAGE
DEBTINC
JOB_Missing
```

As the sequential forward selection plot showed that the sweet spot is 4 to 5 variables, from the 6 variables listed above we would select the following:

DEROG , DELINQ , CLAGE , DEBTINC , YOJ

We would not choose JOB_Missing as it is a binary variables

In [97]:
```
u_train = x_train[finalStepVars]
u_test = x_test[finalStepVars]
```

In [98]:
```
v_train = w_train[GB_amt]
stepVarNames = list(v_train.columns.values)
maxCols = v_train.shape[1]
```

In [99]:
```
sfs = SFS( LinearRegression(),
           k_features=( 1, maxCols ),
           forward=True,
           floating=False,
           scoring = 'r2',
           cv=5
           )
sfs.fit(v_train.values, z_train[TARGET_A].values)
```

Out[99]:
```
SequentialFeatureSelector(clone_estimator=True, cv=5,
                          estimator=LinearRegression(copy_X=True,
                                                     fit_intercep
t=True,
                                                     n_jobs=None,
                                                     normalize=Fa
lse),
                          fixed_features=None, floating=False, fo
rward=True,
                          k_features=(1, 3), n_jobs=1, pre_dispat
ch='2*n_jobs',
                          scoring='r2', verbose=0)
```

In [100]:
```
theFigure = plot_sfs(sfs.get_metric_dict(), kind=None)
plt.title('LOSS Sequential Forward Selection (w. StdErr)')
plt.grid()
plt.show()
```

The Sequential Forward Selection plot for the Loss Amount shows us that 3 variables are ideal for a high performance

In [101]:
```
dfm = pd.DataFrame.from_dict(sfs.get_metric_dict()).T
dfm = dfm[['feature_names', 'avg_score']]
dfm.avg_score = dfm.avg_score.astype(float)
```

In [102]:
```
maxIndex = dfm.avg_score.idxmax()
maxIndex
```

Out[102]: 3

In [103]:
```
stepVars = dfm.iloc[maxIndex, ]
stepVars = stepVars.feature_names
print(stepVars)
```

```
---------------------------------------------------------------------
----------
IndexError                                Traceback (most recent
call last)
<ipython-input-103-21a3ac34e841> in <module>
----> 1 stepVars = dfm.iloc[maxIndex, ]
      2 stepVars = stepVars.feature_names
      3 print(stepVars)

~/opt/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.
py in __getitem__(self, key)
   1416                 except (KeyError, IndexError,
AttributeError):
   1417                     pass
-> 1418             return self._getitem_tuple(key)
   1419         else:
   1420             # we by definition only have the 0th axis

~/opt/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.
py in _getitem_tuple(self, tup)
   2090     def _getitem_tuple(self, tup):
   2091
-> 2092         self._has_valid_tuple(tup)
   2093         try:
   2094             return self._getitem_lowerdim(tup)

~/opt/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.
py in _has_valid_tuple(self, key)
    233                 raise IndexingError("Too many indexers")
    234             try:
--> 235                 self._validate_key(k, i)
    236             except ValueError:
    237                 raise ValueError(

~/opt/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.
py in _validate_key(self, key, axis)
   2012             return
```

```
   2013         elif is_integer(key):
-> 2014             self._validate_integer(key, axis)
   2015         elif isinstance(key, tuple):
   2016             # a tuple should already have been caught by
this point

~/opt/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.
py in _validate_integer(self, key, axis)
   2086         len_axis = len(self.obj._get_axis(axis))
   2087         if key >= len_axis or key < -len_axis:
-> 2088             raise IndexError("single positional indexer i
s out-of-bounds")
   2089
   2090     def _getitem_tuple(self, tup):

IndexError: single positional indexer is out-of-bounds
```

In [ ]:
```
finalStepVars = []
for i in stepVars :
    index = int(i)
    try :
        theName = stepVarNames[ index ]
        finalStepVars.append( theName )
    except :
        pass

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

We would need all three of these variables for our model to best predict the loss amount according to our regression models as well as the sequential forward selection plot.

In [104]:
```
v_train = w_train[finalStepVars]
v_test = w_test[finalStepVars]
```

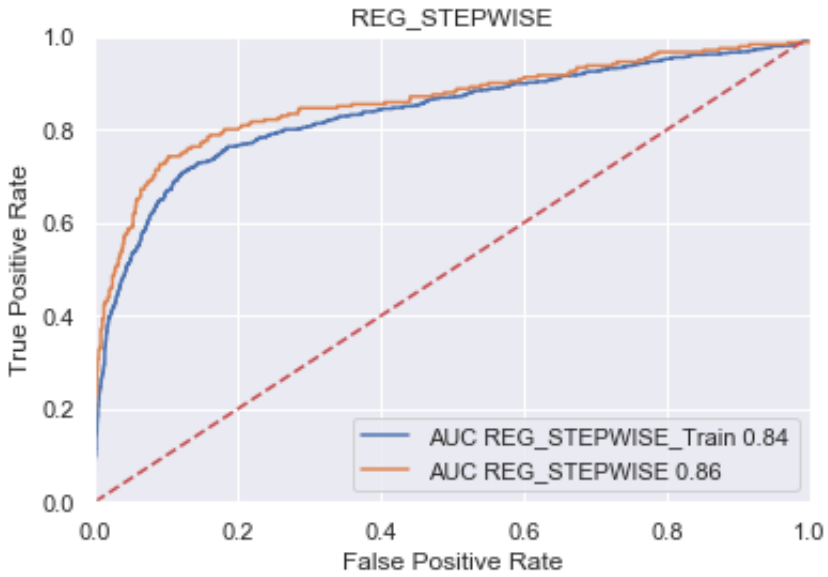
STEPWISE REGRESSION

In [105]:
```
WHO = "REG_STEPWISE"
```

In [106]:
```
DFLT = LogisticRegression(solver='newton-cg', max_iter=1000 )
DFLT = DFLT.fit(u_train, y_train[TARGET_F])
```

In [107]:
```
TRAIN_DFLT = getProbAccuracyScores(WHO + "_Train", DFLT, u_train,
y_train[TARGET_F])
TEST_DFLT = getProbAccuracyScores(WHO, DFLT, u_test, y_test[TARGET
_F])
```

In [108]:
```
print_ROC_Curve(WHO, [TRAIN_DFLT, TEST_DFLT])
print_Accuracy(WHO + " CLASSIFICATION ACCURACY", [TRAIN_DFLT, TEST
_DFLT])
```




```
REG_STEPWISE CLASSIFICATION ACCURACY
======
REG_STEPWISE_Train  =  0.8658276450511946
REG_STEPWISE  =  0.875532821824382
------
```

DAMAGES

In [109]:
```
AMT = LinearRegression()
AMT = AMT.fit(v_train, z_train[TARGET_A])
```

In [110]:
```
TRAIN_AMT = getAmtAccuracyScores(WHO + "_Train", AMT, v_train, z_t
rain[TARGET_A])
TEST_AMT = getAmtAccuracyScores(WHO, AMT, v_test, z_test[TARGET_A]
)
print_Accuracy(WHO + " RMSE ACCURACY", [TRAIN_AMT, TEST_AMT])
```

```
REG_STEPWISE RMSE ACCURACY
======
REG_STEPWISE_Train  =  10244.078370393285
REG_STEPWISE  =  9974.56604376007
------
```

In [111]:
```
REG_STEP_DFLT_COEF = getCoefLogit(DFLT, u_train)
REG_STEP_AMT_COEF = getCoefLinear(AMT, v_train)
```

```
DEFAULT
---------
Total Variables:  7
INTERCEPT  =  0.6724781203382588
YOJ  =  -0.02150472468821733
DEROG  =  0.53635073623535
DELINQ  =  0.6650022598622028
CLAGE  =  -0.006143139272392336
DEBTINC  =  -0.059728058070994725
JOB_Missing  =  -1.1504563865236825

LOSS
---------
Total Variables:  7
INTERCEPT  =  11958.100097649602
YOJ  =  10.921621452295586
DEROG  =  701.1870105333505
DELINQ  =  1305.3002198270146
CLAGE  =  -3.2897075405969645
DEBTINC  =  -28.97110628155889
JOB_Missing  =  382.45997700226786
```

The stepwise regression model gives similar and parallel results with our previous models. People with high DEROG and DELINQ are most likely to default whereas people with higher DEBTINC and YOJ are less likely. The results for Loss Amount are also consistent with previous results

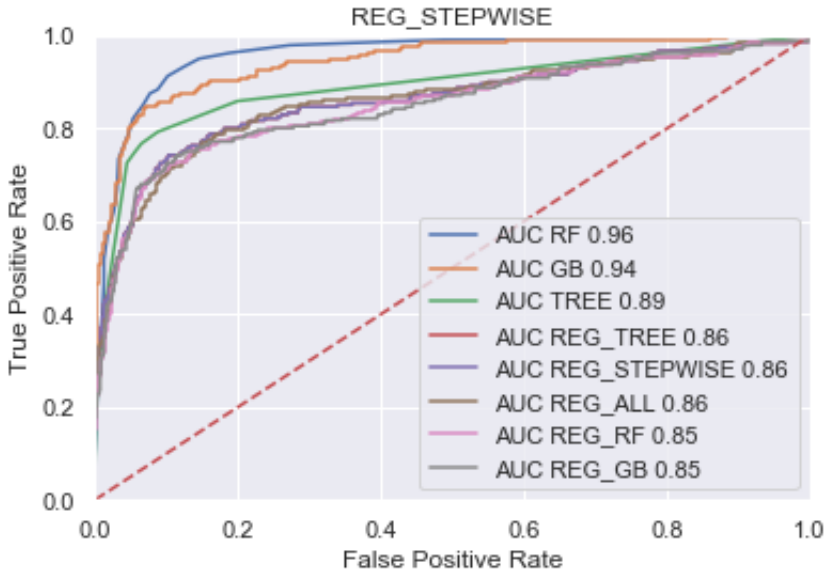
In [112]:
```
REG_STEP_DFLT = TEST_DFLT.copy()
REG_STEP_AMT = TEST_AMT.copy()
```

LOSS AMOUNT

In [113]:
```
ALL_DFLT = [TREE_DFLT, RF_DFLT, GB_DFLT, REG_ALL_DFLT, REG_TREE_DF
LT, REG_RF_DFLT, REG_GB_DFLT, REG_STEP_DFLT]
```

In [114]:
```
ALL_DFLT = sorted(ALL_DFLT, key = lambda x: x[4], reverse=True)
print_ROC_Curve(WHO, ALL_DFLT)

ALL_DFLT = sorted(ALL_DFLT, key = lambda x: x[1], reverse=True)
print_Accuracy("ALL CLASSIFICATION ACCURACY", ALL_DFLT)
```




```
ALL CLASSIFICATION ACCURACY
======
RF  =  0.9190110826939472
GB  =  0.9121909633418585
TREE  =  0.9036658141517476
REG_TREE  =  0.875532821824382
REG_STEPWISE  =  0.875532821824382
REG_ALL  =  0.8729752770673487
REG_RF  =  0.8729752770673487
REG_GB  =  0.8678601875532822
------
```

The ROC Curve graph with accuracy of all models we ran shows that our most accurate model is the Random Forest model, followed by the Gradient Boosting model. We see that Regression Model where we used only the Gradient Boosting variables is the least accurate.

While there isn't a drastically significant difference between accuracies of various in classification on the models we ran, we see that random forest and gradient boosting have the higher with 91% whereas Regression model for random forest variables and gradient boosting variables have the lowest accuracy of 87%.

In [115]:
```
ALL_AMT = [TREE_AMT, RF_AMT, GB_AMT, REG_ALL_AMT, REG_TREE_AMT, RE
G_RF_AMT, REG_GB_AMT, REG_STEP_AMT]
ALL_AMT = sorted(ALL_AMT, key = lambda x: x[1])
print_Accuracy("ALL DAMAGE MODEL ACCURACY", ALL_AMT)
```

```
ALL DAMAGE MODEL ACCURACY
======
GB  =  3149.092174701888
RF  =  4295.973809056954
REG_ALL  =  4360.171040128947
REG_TREE  =  5128.335320171946
REG_RF  =  5128.335320171946
REG_GB  =  5128.335320171946
TREE  =  6065.25844508915
REG_STEPWISE  =  9974.56604376007
------
```

Best models to predict the Loss Amount are Gradient Boosting, Random Forest and the regression model we ran with all variables.

Per the above results, I would choose Gradient Boosting Model as it is consistently highly accurate