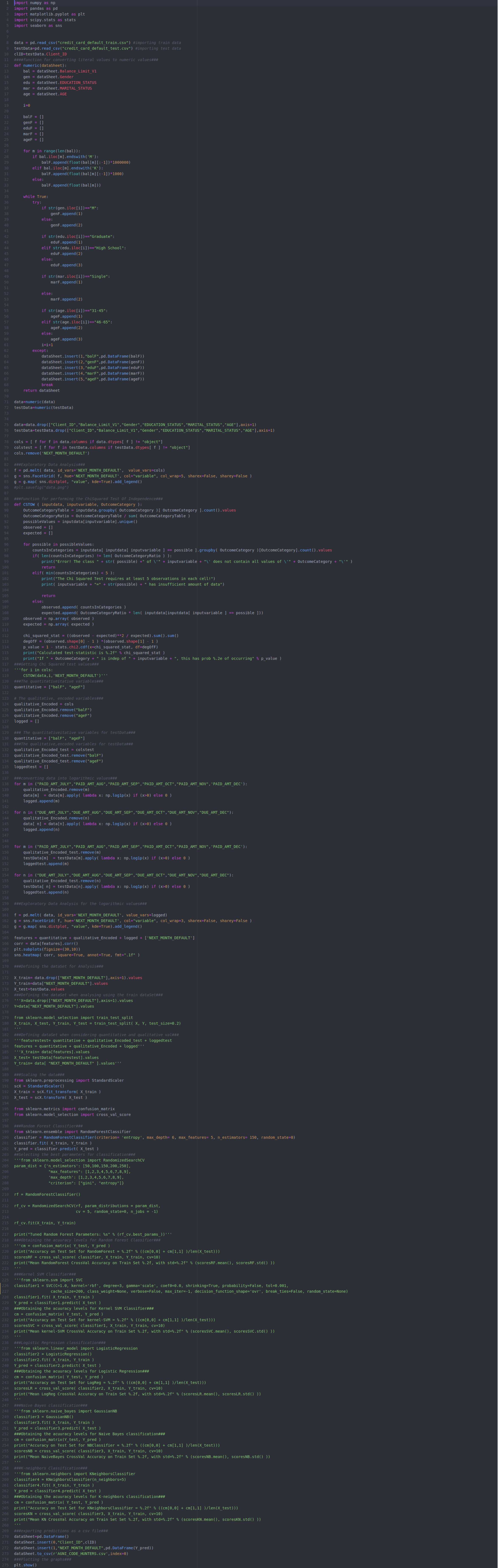

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
import seaborn as sns


data = pd.read_csv("credit_card_default_train.csv") #importing train data
testData=pd.read_csv("credit_card_default_test.csv") #importing test data
clID=testData.Client_ID
####function for converting literal values to numeric values###
def numeric(dataSheet):
    bal = dataSheet.Balance_Limit_V1
    gen = dataSheet.Gender
    edu = dataSheet.EDUCATION_STATUS
    mar = dataSheet.MARITAL_STATUS
    age = dataSheet.AGE

    i=0

    balF = []
    genF = []
    eduF = []
    marF = []
    ageF = []

    for m in range(len(bal)):
        if bal.iloc[m].endswith('M'):
            balF.append(float(bal[m][:-1])*1000000)
        elif bal.iloc[m].endswith('K'):
            balF.append(float(bal[m][:-1])*1000)
        else:
            balF.append(float(bal[m]))

    while True:
        try:
            if str(gen.iloc[i])=="M":
                genF.append(1)
            else:
                genF.append(2)

            if str(edu.iloc[i])=="Graduate":
                eduF.append(1)
            elif str(edu.iloc[i])=="High School":
                eduF.append(2)
            else:
                eduF.append(3)

            if str(mar.iloc[i])=="Single":
                marF.append(1)

            else:
                marF.append(2)

            if str(age.iloc[i])=="31-45":
                ageF.append(1)
            elif str(age.iloc[i])=="46-65":
                ageF.append(2)
            else:
                ageF.append(3)
            i=i+1
        except:
            dataSheet.insert(1,"balF",pd.DataFrame(balF))
            dataSheet.insert(2,"genF",pd.DataFrame(genF))
            dataSheet.insert(3,"eduF",pd.DataFrame(eduF))
            dataSheet.insert(4,"marF",pd.DataFrame(marF))
            dataSheet.insert(5,"ageF",pd.DataFrame(ageF))
            break
    return dataSheet

data=numeric(data)
testData=numeric(testData)


data=data.drop(["Client_ID","Balance_Limit_V1","Gender","EDUCATION_STATUS","MARITAL_STATUS","AGE"],axis=1)
testData=testData.drop(["Client_ID","Balance_Limit_V1","Gender","EDUCATION_STATUS","MARITAL_STATUS","AGE"],axis=1)

cols = [ f for f in data.columns if data.dtypes[ f ] != "object"]
colstest = [ f for f in testData.columns if testData.dtypes[ f ] != "object"]
cols.remove('NEXT_MONTH_DEFAULT')

###Exploratory Data Analysis###
f = pd.melt( data, id_vars='NEXT_MONTH_DEFAULT',  value_vars=cols)
g = sns.FacetGrid( f, hue='NEXT_MONTH_DEFAULT', col="variable", col_wrap=5, sharex=False, sharey=False )
g = g.map( sns.distplot, "value", kde=True).add_legend()
#plt.savefig("data.png")

###Function for performing the ChiSquared Test Of Independence###
def CSTOW ( inputdata, inputvariable, OutcomeCategory ):
    OutcomeCategoryTable = inputdata.groupby( OutcomeCategory )[ OutcomeCategory ].count().values
    OutcomeCategoryRatio = OutcomeCategoryTable / sum( OutcomeCategoryTable )
    possibleValues = inputdata[inputvariable].unique()
    observed = []
    expected = []

    for possible in possibleValues:
        countsInCategories = inputdata[ inputdata[ inputvariable ] == possible ].groupby( OutcomeCategory )[OutcomeCategory].count().values
        if( len(countsInCategories) != len( OutcomeCategoryRatio ) ):
            print("Error! The class " + str( possible) +" of \'" + inputvariable + "\' does not contain all values of \'" + OutcomeCategory + "\'" )
            return
        elif( min(countsInCategories) < 5 ):
            print("The Chi Squared Test requires at least 5 observations in each cell!")
            print( inputvariable + "=" + str(possible) + " has insufficient amount of data")

            return
        else:
            observed.append( countsInCategories )
            expected.append( OutcomeCategoryRatio * len( inputdata[inputdata[ inputvariable ] == possible ]))
    observed = np.array( observed )
    expected = np.array( expected )

    chi_squared_stat = ((observed - expected)**2 / expected).sum().sum()
    degOfF = (observed.shape[0] - 1 ) *(observed.shape[1] - 1 )
    p_value = 1 - stats.chi2.cdf(x=chi_squared_stat, df=degOfF)
    print("Calculated test-statistic is %.2f" % chi_squared_stat )
    print("If " + OutcomeCategory + " is indep of " + inputvariable + ", this has prob %.2e of occurring" % p_value )
###Getting Chi Squared test values###
'''for i in cols:
    CSTOW(data,i,'NEXT_MONTH_DEFAULT')'''
###The quantitativeitative variables###
quantitative = ["balF", "ageF"]

# The qualitative, encoded variables###
qualitative_Encoded = cols
qualitative_Encoded.remove("balF")
qualitative_Encoded.remove("ageF")
logged = []

### The quantitativeitative variables for testData###
quantitative = ["balF", "ageF"]
###The qualitative,encoded variables for testData###
qualitative_Encoded_test = colstest
qualitative_Encoded_test.remove("balF")
qualitative_Encoded_test.remove("ageF")
loggedtest = []

###converting data into logarithmic values###
for m in ("PAID_AMT_JULY","PAID_AMT_AUG","PAID_AMT_SEP","PAID_AMT_OCT","PAID_AMT_NOV",'PAID_AMT_DEC'):
    qualitative_Encoded.remove(m)
    data[m]  = data[m].apply( lambda x: np.log1p(x) if (x>0) else 0 )
    logged.append(m)

for n in ("DUE_AMT_JULY","DUE_AMT_AUG","DUE_AMT_SEP","DUE_AMT_OCT","DUE_AMT_NOV","DUE_AMT_DEC"):
    qualitative_Encoded.remove(n)
    data[ n] = data[n].apply( lambda x: np.log1p(x) if (x>0) else 0 )
    logged.append(n)


for m in ("PAID_AMT_JULY","PAID_AMT_AUG","PAID_AMT_SEP","PAID_AMT_OCT","PAID_AMT_NOV",'PAID_AMT_DEC'):
    qualitative_Encoded_test.remove(m)
    testData[m]  = testData[m].apply( lambda x: np.log1p(x) if (x>0) else 0 )
    loggedtest.append(m)

for n in ("DUE_AMT_JULY","DUE_AMT_AUG","DUE_AMT_SEP","DUE_AMT_OCT","DUE_AMT_NOV","DUE_AMT_DEC"):
    qualitative_Encoded_test.remove(n)
    testData[ n] = testData[n].apply( lambda x: np.log1p(x) if (x>0) else 0 )
    loggedtest.append(n)

###Exploratory Data Analysis for the logarithmic values###

f = pd.melt( data, id_vars='NEXT_MONTH_DEFAULT', value_vars=logged)
g = sns.FacetGrid( f, hue='NEXT_MONTH_DEFAULT', col="variable", col_wrap=3, sharex=False, sharey=False )
g = g.map( sns.distplot, "value", kde=True).add_legend()

features = quantitative + qualitative_Encoded + logged + ['NEXT_MONTH_DEFAULT']
corr = data[features].corr()
plt.subplots(figsize=(30,10))
sns.heatmap( corr, square=True, annot=True, fmt=".1f" )

###Defining the dataSet for Analysis###

X_train= data.drop(["NEXT_MONTH_DEFAULT"],axis=1).values
Y_train=data["NEXT_MONTH_DEFAULT"].values
X_test=testData.values
###Defining the dataSet when analysing using the train dataSet###
'''X=data.drop(["NEXT_MONTH_DEFAULT"],axis=1).values
Y=data["NEXT_MONTH_DEFAULT"].values

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size=0.2)
'''
###Defining dataSet when considering quantitative and qualitative val###
'''featurestest= quantitative + qualitative_Encoded_test + loggedtest
features = quantitative + qualitative_Encoded + logged'''
'''X_train= data[features].values
X_test= testData[featurestest].values
Y_train= data[ "NEXT_MONTH_DEFAULT" ].values'''

###Scaling the data###
from sklearn.preprocessing import StandardScaler
scX = StandardScaler()
X_train = scX.fit_transform( X_train )
X_test = scX.transform( X_test )

from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score

###Random Forest Classifier###
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(criterion= 'entropy', max_depth= 6, max_features= 5, n_estimators= 150, random_state=0)
classifier.fit( X_train, Y_train )
Y_pred = classifier.predict( X_test )
##Selecting the best parameters for classification###
'''from sklearn.model_selection import RandomizedSearchCV
param_dist = {'n_estimators': [50,100,150,200,250],
               "max_features": [1,2,3,4,5,6,7,8,9],
               'max_depth': [1,2,3,4,5,6,7,8,9],
               "criterion": ["gini", "entropy"]}

rf = RandomForestClassifier()

rf_cv = RandomizedSearchCV(rf, param_distributions = param_dist,
                           cv = 5, random_state=0, n_jobs = -1)

rf_cv.fit(X_train, Y_train)

print("Tuned Random Forest Parameters: %s" % (rf_cv.best_params_))'''
###Obtaining the acuuracy levels for Random Forest Classifier###
'''cm = confusion_matrix( Y_test, Y_pred )
print("Accuracy on Test Set for RandomForest = %.2f" % ((cm[0,0] + cm[1,1] )/len(X_test)))
scoresRF = cross_val_score( classifier, X_train, Y_train, cv=10)
print("Mean RandomForest CrossVal Accuracy on Train Set %.2f, with std=%.2f" % (scoresRF.mean(), scoresRF.std() ))
'''
###Kernel SVM Classifier###
'''from sklearn.svm import SVC
classifier1 = SVC(C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001,
                cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None)
classifier1.fit( X_train, Y_train )
Y_pred = classifier1.predict( X_test )
###Obtaining the acuuracy levels for Kernel SVM Classifier###
cm = confusion_matrix( Y_test, Y_pred )
print("Accuracy on Test Set for kernel-SVM = %.2f" % ((cm[0,0] + cm[1,1] )/len(X_test)))
scoresSVC = cross_val_score( classifier1, X_train, Y_train, cv=10)
print("Mean kernel-SVM CrossVal Accuracy on Train Set %.2f, with std=%.2f" % (scoresSVC.mean(), scoresSVC.std() ))
'''
###Logistic Regression classification###
'''from sklearn.linear_model import LogisticRegression
classifier2 = LogisticRegression()
classifier2.fit( X_train, Y_train )
Y_pred = classifier2.predict( X_test )
###Obtaining the acuuracy levels for Logistic Regression###
cm = confusion_matrix( Y_test, Y_pred )
print("Accuracy on Test Set for LogReg = %.2f" % ((cm[0,0] + cm[1,1] )/len(X_test)))
scoresLR = cross_val_score( classifier2, X_train, Y_train, cv=10)
print("Mean LogReg CrossVal Accuracy on Train Set %.2f, with std=%.2f" % (scoresLR.mean(), scoresLR.std() ))
'''
###Naive Bayes classification###
'''from sklearn.naive_bayes import GaussianNB
classifier3 = GaussianNB()
classifier3.fit( X_train, Y_train )
Y_pred = classifier3.predict( X_test )
###Obtaining the acuuracy levels for Naive Bayes classification###
cm = confusion_matrix(Y_test, Y_pred )
print("Accuracy on Test Set for NBClassifier = %.2f" % ((cm[0,0] + cm[1,1] )/len(X_test)))
scoresNB = cross_val_score( classifier3, X_train, Y_train, cv=10)
print("Mean NaiveBayes CrossVal Accuracy on Train Set %.2f, with std=%.2f" % (scoresNB.mean(), scoresNB.std() ))
'''
###K-neighbors Classification###
'''from sklearn.neighbors import KNeighborsClassifier
classifier4 = KNeighborsClassifier(n_neighbors=5)
classifier4.fit( X_train, Y_train )
Y_pred = classifier4.predict( X_test )
###Obtaining the acuuracy levels for K-neighbors classification###
cm = confusion_matrix( Y_test, Y_pred )
print("Accuracy on Test Set for KNeighborsClassifier = %.2f" % ((cm[0,0] + cm[1,1] )/len(X_test)))
scoresKN = cross_val_score( classifier3, X_train, Y_train, cv=10)
print("Mean KN CrossVal Accuracy on Train Set Set %.2f, with std=%.2f" % (scoresKN.mean(), scoresKN.std() ))
'''
###exporting predictions as a csv file###
dataSheet=pd.DataFrame()
dataSheet.insert(0,"Client_ID",clID)
dataSheet.insert(1,"NEXT_MONTH_DEFAULT",pd.DataFrame(Y_pred))
dataSheet.to_csv(r'AGNI_CODE_HUNTERS.csv',index=0)
###Plotting the graphs###
plt.show()
```