Instructions to run python code for Santander Customer transaction prediction with out sampling using Logistic regression, Decision trees, Random Forest, Naïve Bayes classification models

```
#Loading libraries
import os
import pandas as pd
import numpy as np
from fancyimpute import KNN
import matplotlib.pyplot as plt
import seaborn as sns
from ggplot import *
import gc
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
#setting the working directory
os.chdir("G:\Edwiser material\Project\Santandarcustomer problems\Edwiser project")
#check for the set directory
os.getcwd()
#Loading the required data
train=pd.read_csv('train.csv')
test=pd.read_csv('test.csv')
#Removing
#delete the ID_code variable which is nothing but string and no information
del train["ID_code"]
del test["ID_code"]
```

```
#plot a bar plot for count of target classes

ggplot(train, aes(x='target'))+\

   geom_bar(fill="Green")+\

   scale_color_brewer(type='diverging', palette=2)+\

   xlab('Target')+ylab('frequecy')+ggtitle("Distribution of target class values")+
theme_bw()

# make list a of columns of train data

cnames=list(train.columns)

#Remove target variable from the list

cnames.remove('target')


# plot for the distribution of all the values of the columns

print('Distributions columns')

plt.figure(figsize=(30, 185))

for i, col in enumerate(cnames):

   plt.subplot(50, 4, i + 1)

   plt.hist(train[col])

   plt.title(col)

gc.collect()

#plt.savefig('hist.png')

# distribution of all the values with respect to target class

print('Distributions columns')

plt.figure(figsize=(30, 185))

for i, col in enumerate(cnames):

   plt.subplot(50, 4, i + 1)

   plt.hist(train[train["target"] == 0][col], alpha=0.5, label='0', color='b')

   plt.hist(train[train["target"] == 1][col], alpha=0.5, label='1', color='r')

   plt.title(col)

gc.collect()

#plt.savefig('hist.png')
```

```
#plot for the frequency of mean
plt.figure(figsize=(10, 5))
train[cnames].mean().plot(kind='hist');
plt.title('Mean Frequency');
#plot for the frequency of Median
plt.figure(figsize=(10, 5))
train[cnames].median().plot(kind='hist');
plt.title('Median Frequency');
#Plot for frequency of standard deviation
plt.figure(figsize=(10, 5))
train[cnames].std().plot('hist');
plt.title('Standard Deviation Frequency');
#plot for the frequency of Skewness
plt.figure(figsize=(10, 5))
train[cnames].skew().plot('hist');
plt.title('Skewness Frequency');
#plot for the frequency of kurtosis
plt.figure(figsize=(10, 5))
train[cnames].kurt().plot('hist');
plt.title('Kurtosis Frequency');
#missing value Analysis
Missing_val_train=pd.DataFrame(train.isnull().sum()
Missing_val_test=pd.DataFrame(test.isnull().sum()
#Feature Selction
## correlation anlysis
#Correlation plot
df_corr=train.loc[:,cnames]

#set the width and height of the plot
f, ax=plt.subplots(figsize=(10,12))

#generate correlation matrix
```

```
corr=df_corr.corr()


#plot using seaborn library
sns.heatmap(corr,
mask=np.zeros_like(corr,dtype=np.bool),cmap=sns.diverging_palette(220,10,
as_cmap=True),
        square=True, ax=ax)
plt.savefig('cor.png')


#Feature scaling
#standardization
#for i in cnames:
#    #print(i)
#    train[i]=(train[i]-train[i].mean())/train[i].std()


# Creating a function to report confusion metrics
def confusion_metrics (conf_matrix):
  # save confusion matrix and slice into four pieces
  TP = conf_matrix[1][1]
  TN = conf_matrix[0][0]
  FP = conf_matrix[0][1]
  FN = conf_matrix[1][0]
  print('True Positives:', TP)
  print('True Negatives:', TN)
  print('False Positives:', FP)
  print('False Negatives:', FN)

  # calculate accuracy
  conf_accuracy = (float (TP+TN) / float(TP + TN + FP + FN))

  # calculate misclassification
  conf_misclassification = 1- conf_accuracy
```

```
   #calculate false negative rate
   conf_FNR = (FN/ float(FN + TN))

   #calculate false positive rate
   conf_FPR = (FP/ float(FP + TP))

   #calculating sensitivity
   conf_sensitivity = (TP / float(TP + FN))

   # calculate the specificity
   conf_specificity = (TN / float(TN + FP))

   # calculate precision
   conf_precision = (TN / float(TN + FP))

   # calculate f_1 score
   conf_f1 = 2 * ((conf_precision * conf_sensitivity) / (conf_precision + conf_sensitivity))

   print('-'*50)
   print(f'Accuracy: {round(conf_accuracy,2)}')
   print(f'Mis-Classification: {round(conf_misclassification,2)}')
   print(f'FNR: {round(conf_FNR,2)}')
   print(f'FPR: {round(conf_FPR,2)}')
   print(f'Sensitivity/TPR: {round(conf_sensitivity,2)}')
   print(f'Specificity/TNR: {round(conf_specificity,2)}')
   print(f'Precision: {round(conf_precision,2)}')
   print(f'f_1 Score: {round(conf_f1,2)}')

#divide data into train and test using simple random sampling
Sample_Index=np.random.rand(len(train))< 0.75
trainLR=train[Sample_Index]
```

```
testLR=train[~Sample_Index]
#select column indexes for independent variables
train_cols=trainLR.columns[1:201]
#build logistic regression model
import statsmodels.api as sm
logit=sm.Logit(trainLR['target'], trainLR[train_cols]).fit()
#summary of the model
logit.summary()
#predict on test data
testLR['Actual_prob']=logit.predict(testLR[train_cols])
#convert the probability into binary class since the classes binary class
testLR['ActualVal']=1
testLR.loc[testLR.Actual_prob<0.5, 'ActualVal']=0
# Evaluate the performance of trained model
#build confusion matrix
CML=pd.crosstab(testLR['target'], testLR['ActualVal'])
# Error metrics for confusion metrics
confusion_metrics(CML)
# AUC_ROC_SCORE
roc_auc_score(testLR['target'], testLR['ActualVal'])
# Splitting the data
#import library for train_test_split function sklearn library
from sklearn.model_selection import train_test_split
#divide data into train and test
x=train.values[:,1:201]
y=train.values[:,0]
x_train, x_test,y_train, y_test=train_test_split(x, y, test_size=0.3)
# Decision tree model
#import the library decision tree
from sklearn import tree
#decision tree
clf=tree.DecisionTreeClassifier(criterion='entropy').fit(x_train, y_train)
```

```
#predict new test cases
DT_Predictions=clf.predict(x_test)
#Build confusion matrix
CMD = pd.crosstab(y_test, DT_Predictions)
#Error matrices
confusion_metrics(CMD)
#AUROC SCORE
roc_auc_score(y_test, DT_Predictions)

# library for Naive Bayes
from sklearn.naive_bayes import GaussianNB

#Naive Bayes implementation
NB_model = GaussianNB().fit(x_train, y_train)

#predict test cases with trained model
NB_Predictions = NB_model.predict(x_test)
#Build confusion matrix for Naive Bayes predictions with actual test target class values
CMN = pd.crosstab(y_test, NB_Predictions)
#Error metrics for the model
confusion_metrics(CMN)
#AUROC SCORE
roc_auc_score(y_test, NB_Predictions)
#importing library required for Random forest model
from sklearn.ensemble import RandomForestClassifier
#Build random forest model on train data
RF_model = RandomForestClassifier(n_estimators = 50).fit(x_train, y_train)
# predict on new test cases
RF_Predictions = RF_model.predict(x_test)
#develop confusion matrix and calculate error
CMR=pd.crosstab(y_test, RF_Predictions)
#Error metrics for the model
```

```
confusion_metrics(CMR)
#AUROC SCORE of the model performance
roc_auc_score(y_test, RF_Predictions)

#importing library for the error metrics for ROC generation
from sklearn import metrics
#define the values to plot the roc curve
fpr_lr, tpr_lr, _ = roc_curve(testLR['target'], testLR['ActualVal'])
fpr_dt, tpr_dt, _ = roc_curve(y_test, DT_Predictions)
fpr_nb, tpr_nb, _ = roc_curve(y_test, NB_Predictions)
fpr_rf, tpr_rf, _ = roc_curve(y_test, RF_Predictions)

plt.figure(1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_lr, tpr_lr, label='LR')
plt.plot(fpr_dt, tpr_dt, label='DT')
plt.plot(fpr_nb, tpr_nb, label='NB')
plt.plot(fpr_rf, tpr_rf, label='RF')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```

Instructions to run python code for Santander Customer transaction prediction with under sampling using Logistic regression, Decision trees, Random Forest, Naïve Bayes classification models

```
#loading both train and test data
train=pd.read_csv('train.csv')
test=pd.read_csv('test.csv')
#loading train and test data
del train["ID_code"]
del test["ID_code"]
# target class counting
count_class_0, count_class_1 = train.target.value_counts()

# Divide by class
train_class_0 = train[train['target'] == 0]
train_class_1 = train[train['target'] == 1]
train_class_0.shape, train_class_1.shape
#Random under sampling
class_0_under = train_class_0.sample(count_class_1)
train_under = pd.concat([class_0_under, train_class_1], axis=0)
print('Random under-sampling:')
print(train_under.target.value_counts())
train_under.target.value_counts().plot(kind='bar', title='Count (target)');
#divide data into train and test using simple random sampling
Sample_Index=np.random.rand(len(train_under))< 0.75
trainLR=train_under[Sample_Index]
testLR=train_under[~Sample_Index]
#select coulmn indexes for independent variables
train_cols=trainLR.columns[1:201]
#importing the logistic regression model from stat models
import statsmodels.api as sm
```

```
#build logistic regression model
logit=sm.Logit(trainLR['target'], trainLR[train_cols]).fit()
#summary of the model
logit.summary()
#predict on test data
testLR['Actual_prob']=logit.predict(testLR[train_cols])
#converting predictions into probabilities
testLR['ActualVal']=1
testLR.loc[testLR.Actual_prob<0.5, 'ActualVal']=0
#build confusion matrix
CML=pd.crosstab(testLR['target'], testLR['ActualVal'])
#error metrics on confusion matrix
confusion_metrics (CML)
#AUROC SCORE calculation
roc_auc_score(testLR['target'], testLR['ActualVal'])
#import library for train_test_split function sklearn library
from sklearn.model_selection import train_test_split
#divide data into train and test
x=train_under.values[:,1:201]
y=train_under.values[:,0]
x_train, x_test,y_train, y_test=train_test_split(x, y, test_size=0.3)
#importing model
from sklearn import tree
#decision tree training
clf=tree.DecisionTreeClassifier(criterion='entropy').fit(x_train, y_train)
#predict new test cases
DT_Predictions=clf.predict(x_test)
#Build confusion matrix
CMD = pd.crosstab(y_test, DT_Predictions)
#Error metrics on confusion matrix
confusion_metrics (CMD)
#AUROC SCORE CALCULATION
```

```
roc_auc_score(y_test, DT_Predictions)
#importing gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
#Naive Bayes implementation on train data
NB_model = GaussianNB().fit(x_train, y_train)
#predicting on test cases
NB_Predictions = NB_model.predict(x_test)
#Build confusion matrix
CMN = pd.crosstab(y_test, NB_Predictions)
#Error metrics application
confusion_metrics (CMN)
#AUROC score calculation
roc_auc_score(y_test, NB_Predictions)
#importing Random Forest Classifier from sklearn library
from sklearn.ensemble import RandomForestClassifier
RF_model = RandomForestClassifier(n_estimators = 50).fit(x_train, y_train)
# predict on test cases
RF_Predictions = RF_model.predict(x_test)
#develop confusion matrix and calculate error
CMR=pd.crosstab(y_test, RF_Predictions)
#applying error metrics
confusion_metrics (CMR)
#AUROC score calculation
roc_auc_score(y_test, RF_Predictions)
```

Instructions to run python code for Santander Customer transaction prediction with Random Oversampling using Logistic regression, Decision trees, Random Forest, Naïve Bayes classification models

```
#loding both train test datasets
train=pd.read_csv('train.csv')
test=pd.read_csv('test.csv')
#deleting ID_code variable from both test and train data
del train["ID_code"]
del test["ID_code"]
# target class counting
count_class_0, count_class_1 = train.target.value_counts()

# Divide by class
train_class_0 = train[train['target'] == 0]
train_class_1 = train[train['target'] == 1]
# Application Random over sampling
class_1_over = train_class_1.sample(count_class_0, replace=True)
train_over = pd.concat([train_class_0, class_1_over], axis=0)

print('Random over-sampling:')
print(train_over.target.value_counts())

train_over.target.value_counts().plot(kind='bar', title='Count (target)');
cnames=list(train_over.columns)
cnames.remove('target')
#divide data into train and test using simple random sampling
Sample_Index=np.random.rand(len(train_over))< 0.75
trainLR=train_over[Sample_Index]
testLR=train_over[~Sample_Index]

#select column indexes for independent variables
```

```
train_cols=trainLR.columns[1:201]
#building logistic regression model
import statsmodels.api as sm
logit=sm.Logit(trainLR['target'], trainLR[train_cols]).fit()
#summary of logistic regression model
logit.summary()
#predict on test data
testLR['Actual_prob']=logit.predict(testLR[train_cols])
#converting probabilities into 0 and 1 classes
testLR['ActualVal']=1
testLR.loc[testLR.Actual_prob<0.5, 'ActualVal']=0
#building confusion matrix
CML=pd.crosstab(testLR['target'], testLR['ActualVal'])
confusion_metrics (CML)
#auroc score calculation
roc_auc_score(testLR['target'], testLR['ActualVal'])
#import library for train_test_split function sklearn library
from sklearn.model_selection import train_test_split
#divide data into train and test
x=train_over.values[:,1:201]
y=train_over.values[:,0]
x_train, x_test,y_train, y_test=train_test_split(x, y, test_size=0.3)
#importing decision tree from sklearn library
from sklearn import tree
#decision tree model building
clf=tree.DecisionTreeClassifier(criterion='entropy').fit(x_train, y_train)
#predict new test cases
DT_Predictions=clf.predict(x_test)
#Build confusion matrix
CMD = pd.crosstab(y_test, DT_Predictions)
#applying error metrics
confusion_metrics (CMD)
```

```
#AUROC SCORE
roc_auc_score(y_test, DT_Predictions)
#importing Gaussian Naive Bayes from sklearn library
from sklearn.naive_bayes import GaussianNB

#Naive Bayes implementation on train data
NB_model = GaussianNB().fit(x_train, y_train)
#predicting the test cases with the train model
NB_Predictions = NB_model.predict(x_test)
#Build confusion matrix
CMN = pd.crosstab(y_test, NB_Predictions)
#Applying error metrics on Confusion matrix
confusion_metrics (CMN)
#AUROC score calculation
roc_auc_score(y_test, NB_Predictions)
#importing random forest model from sklearn library
from sklearn.ensemble import RandomForestClassifier
#training on train  data
RF_model = RandomForestClassifier(n_estimators =50).fit(x_train, y_train)
# predicting test cases with trained model
RF_Predictions = RF_model.predict(x_test)
#develop confusion matrix and calculate error
CMR=pd.crosstab(y_test, RF_Predictions)
#Application of error metrics on Confusion matrix
confusion_metrics (CMR)
#AUROC score calculation
roc_auc_score(y_test, RF_Predictions)
```

Instructions to run python code for Santander Customer transaction prediction with SMOTE sampling using Logistic regression, Decision trees, Random Forest, Naïve Bayes classification models

#Load the both test and train data

```
train=pd.read_csv('train.csv')

test=pd.read_csv('test.csv')
```

#deleting the ID_code variable from both test and train data as it is nothing but code and no #information

```
del train["ID_code"]

del test["ID_code"]
```

#plotting a barplot for count of target classes

```
ggplot(train, aes(x='target'))+\
  geom_bar(fill="Green")+\
  scale_color_brewer(type='diverging', palette=2)+\
  xlab('Target')+ylab('frequecy')+ggtitle("Distribution of target class values")+
theme_bw()
```

 # defining a X and y values from train data and store

```
X=train.values[:,1:201]

y=train.values[:,0]
```

 # Splitting the X and y values into X_train, y_train , X_test and y_test values

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

#Checking the no of target classes

```
sum(y_train==1), sum(y_train==0)
```

#Applying SMOTE oversampling method to increase the no of minority class

```
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=2)

X_train_res, y_train_res = sm.fit_sample(X_train, y_train.ravel())
```

#Check the no of target class after oversampling

```
sum(y_train_res==1), sum(y_train_res==0)
```

```
# Loading Decision tree classifier from sklearn library
from sklearn import tree
#Building decision tree model
clf=tree.DecisionTreeClassifier(criterion='entropy').fit(X_train_res, y_train_res)
#predicting test cases with trained model
DT_Predictions=clf.predict(X_test)
#Building confusion matrix
CMD = pd.crosstab(y_test, DT_Predictions)
#Applying the error metrics on Confusion metrics
confusion_metrics (CMD)
#calculating AUROC score with roc_auc_score function from sklearn.metrics library
roc_auc_score(y_test, DT_Predictions)
# Importing Gaussian Naive Bayes from from sklearn library
from sklearn.naive_bayes import GaussianNB
#Naive Bayes implementation on the train data
NB_model = GaussianNB().fit(X_train_res, y_train_res)
#predicting the test cases with the trained model
NB_Predictions = NB_model.predict(X_test)
#Build confusion matrix between NB predictions and test case values
CMN = pd.crosstab(y_test, NB_Predictions)
#Applying error metrics on confusion matrix
confusion_metrics (CMN)
#calculating AUROC score with roc_auc_score function from sklearn.metrics library
roc_auc_score(y_test, NB_Predictions)
#importing random forest model from sklearn library
from sklearn.ensemble import RandomForestClassifier
#training on train  data
RF_model = RandomForestClassifier(n_estimators =50).fit(X_train, y_train)
# predicting test cases with trained model
RF_Predictions = RF_model.predict(X_test)
#develop confusion matrix and calculate error
CMR=pd.crosstab(y_test, RF_Predictions)
```

#Application of error metrics on Confusion matrix

```
confusion_metrics (CMR)
```

#calculating AUROC score with roc_auc_score function from sklearn.metrics library

```
roc_auc_score(y_test, RF_Predictions)
```

Finalizing the model for test data Prediction

```
#loding both train test datasets
train=pd.read_csv('train.csv')
test=pd.read_csv('test.csv')
#deleting ID_code variable from both test and train data
del train["ID_code"]
del test["ID_code"]
# target class counting
count_class_0, count_class_1 = train.target.value_counts()
# Divide by class
train_class_0 = train[train['target'] == 0]
train_class_1 = train[train['target'] == 1]
# Application of Random over sampling
class_1_over = train_class_1.sample(count_class_0, replace=True)
train_over = pd.concat([train_class_0, class_1_over], axis=0)
print('Random over-sampling:')
print(train_over.target.value_counts())
train_over.target.value_counts().plot(kind='bar', title='Count (target)');
#importing Random Forest Classifier from sklearn library
from sklearn.ensemble import RandomForestClassifier
RF_model = RandomForestClassifier(n_estimators =
50).fit(train_over.iloc[:,1:201], train_over.iloc[:,0])
import pickle
pickle.dump(RF_model, open('model.pkl','wb'))
# predict on test cases
RF_Predictions = RF_model.predict(test)
#making the data frame with random forest predictions
pred=pd.DataFrame(RF_Predictions)
test=pd.read_csv("test.csv")
```

```
# concatenating the both predictions and test data with pd.concat function

test_data_pred_with_RF = pd.concat([test, pred], axis=1)

# renaming the predicted transaction column name

test_data_pred_with_RF=test_data_pred_with_RF.rename(columns = { 0:
'precited_transaction'})

#saving the predicted values in test data into disc

test_data_pred_with_RF.to_csv("predictions_RF.csv",index=False)
```