Bank Loan Default Risk Analysis

Problem Statement

For banks/financial institutions to be able to weight the risk of their prospective borrower being able to fulfill their repayments. It is classification task.

Classification

1. It is a classification task, Binary-classification
2. Performance Matrix: log-loss,Accuracy,Confusion Matrix(Precision, Recall),F1-Score,ROC & AUC.
3. Apply -machine learning methodologies including
    1. Logistic Regression, Classification
    2. Decision Trees,
    3. Bagging - Random Forest (ensemble approach)
4. Synthetic Minority Oversampling Technique (SMOTE) (There are different methods to balance the data such as oversampling,under-sampling)

Desire Outcome: The importance to manage risk has become more and more important recently

1. Import python modules

```
# Load necessary python modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from imblearn.over_sampling import SMOTE

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve,auc
from sklearn.metrics import log_loss
```

2. Load Dataset

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

loan =  pd.read_csv("/kaggle/input/freddie-mac-singlefamily-loanlevel-dataset/loan_level_500k.csv")
loan=loan.sample(100000)
loan.reset_index(drop=True)
```

```
/kaggle/input/freddie-mac-singlefamily-loanlevel-dataset/loan_level_500k.csv
```

	CREDIT_SCORE	FIRST_PAYMENT_DATE	FIRST_TIME_HOMEBUYER_FLAG	MATURITY_DATE	METROPOLITAN_STATIS
0	655.0	200105	N	203104	
1	718.0	200104	N	203103	
2	697.0	199905	N	202904	
3	668.0	200203	NaN	203202	

Note:

1. Sample size reduced to 100000 , for computing speed

3. Basic Inspection on dataset

Find the Answers to the following Questions

1. Dataset - features , shape
2. Null / Nan Values
3. Features - data types
4. Statistics - mean,median,std,25%,50%,75%,count , min,max on coloum wise
5. Balanced/Imbalanced Dataset

```
print(loan.shape)
print("\n")
print(loan.columns)
print("\n")
print(loan.dtypes)
print("\n")
print(loan.info())
print("\n")
print(loan.isnull().sum().sort_values(ascending=False))
print("\n")
print(loan.DELINQUENT.value_counts(normalize=True))
print("\n")
print(loan.describe().T)
```

```
                                    max
CREDIT_SCORE                     839.00
FIRST_PAYMENT_DATE            200701.00
MATURITY_DATE                 203612.00
METROPOLITAN_STATISTICAL_AREA  49740.00
MORTGAGE_INSURANCE_PERCENTAGE     53.00
NUMBER_OF_UNITS                    4.00
ORIGINAL_COMBINED_LOAN_TO_VALUE  160.00
ORIGINAL_DEBT_TO_INCOME_RATIO     65.00
ORIGINAL_UPB                  544000.00
ORIGINAL_LOAN_TO_VALUE           100.00
ORIGINAL_INTEREST_RATE            10.75
POSTAL_CODE                    99900.00
ORIGINAL_LOAN_TERM               362.00
NUMBER_OF_BORROWERS                2.00
```

Observations:

1. Dataset rows/samples = 100000 with columns/features=27
2. Datsset is higly imbalanced (96:4)
3. Dataset is having null/missing values in the following columns (with missing value count)
    1. FIRST_TIME_HOMEBUYER_FLAG : 25992
    2. METROPOLITAN_STATISTICAL_AREA : 14065
    3. MORTGAGE_INSURANCE_PERCENTAGE : 10140
    4. ORIGINAL_DEBT_TO_INCOME_RATIO : 2915
    5. PREPAYMENT_PENALTY_MORTGAGE_FLAG : 1086
    6. CREDIT_SCORE : 547
    7. NUMBER_OF_BORROWERS : 50
    8. PROPERTY_TYPE : 17
4. DELINQUENT is output dependent variable(Binary)

```
loan.DELINQUENT.value_counts()
```

```
False    96409
True      3591
Name: DELINQUENT, dtype: int64
```

4. Data Cleaning & Manipulation

5. Data Analysis, Visualization and Interpretation

Drop irrelevant columns

The dataset used contains information that is unavailable at the time of loan application. We will drop these columns before starting with our analysis. The columns we will drop are:

```
LOAN_SEQUENCE_NUMBER
FIRST_PAYMENT_DATE
MATURITY_DATE
MORTGAGE_INSURANCE_PERCENTAGE
ORIGINAL_UPB
ORIGINAL_INTEREST_RATE
PREPAYMENT_PENALTY_MORTGAGE_FLAG
```

Other columns we will drop are:

```
We will drop PROPERTY_STATE as this information is encoded in the MSA column.

LOAN_SEQUENCE_NUMBER is a unique id assigned to each loan. As it provides no information we will drop this column.

SELLER_NAME and SERVICER_NAME are dependent loan activity and since this information is not available at the time of loan request we will drop the
```

```
drop_features=['FIRST_PAYMENT_DATE', 'MATURITY_DATE', 'MORTGAGE_INSURANCE_PERCENTAGE','ORIGINAL_UPB', 'ORIGINAL_INTEREST_RATE','PREPAYMEN
loan.drop(columns=drop_features, inplace=True)
```

```
loan.isnull().sum().sort_values(ascending=False)
```

```
FIRST_TIME_HOMEBUYER_FLAG         26049
METROPOLITAN_STATISTICAL_AREA     14128
ORIGINAL_DEBT_TO_INCOME_RATIO      3024
CREDIT_SCORE                        528
NUMBER_OF_BORROWERS                  43
PROPERTY_TYPE                        18
POSTAL_CODE                           8
ORIGINAL_COMBINED_LOAN_TO_VALUE       4
ORIGINAL_LOAN_TO_VALUE                3
NUMBER_OF_UNITS                       1
ORIGINAL_LOAN_TERM                    0
PREPAID                               0
CHANNEL                               0
LOAN_PURPOSE                          0
PRODUCT_TYPE                          0
OCCUPANCY_STATUS                      0
DELINQUENT                            0
dtype: int64
```

Categorical variables

```
categorical_variables = [i for i in loan.columns if loan[i].dtype == "object"]
print(categorical_variables)
```

```
['FIRST_TIME_HOMEBUYER_FLAG', 'OCCUPANCY_STATUS', 'CHANNEL', 'PRODUCT_TYPE', 'PROPERTY_TYPE', 'LOAN_PURPOSE']
```

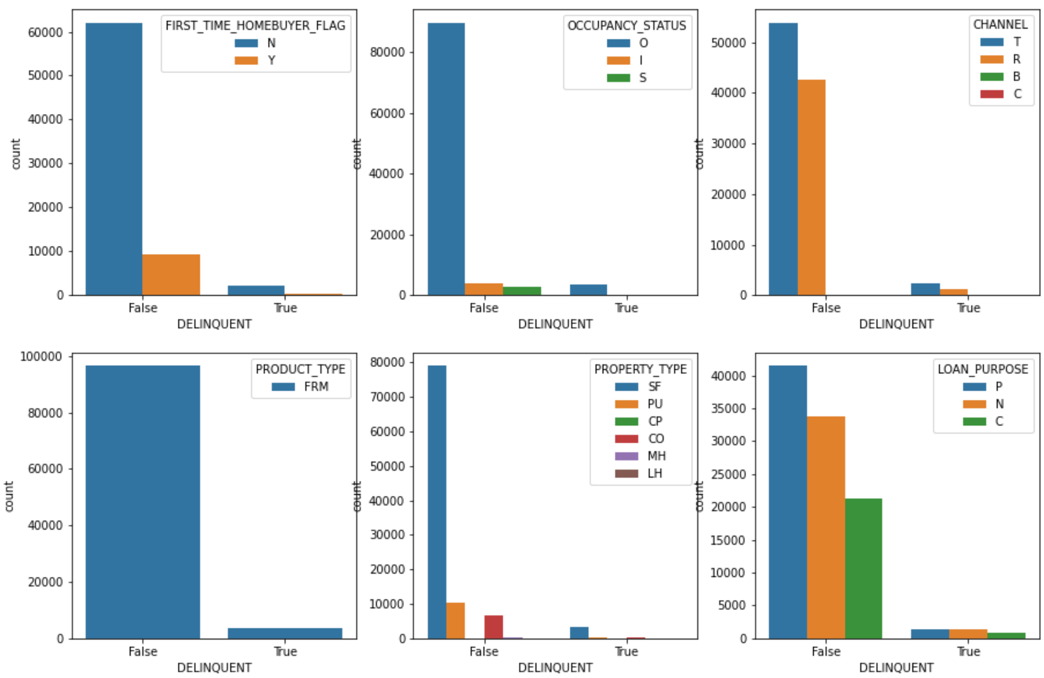
```
for features in loan:
    if (loan[features].dtype=="object"):
        print(features,":",loan[features].unique())
```

```
FIRST_TIME_HOMEBUYER_FLAG : ['N' nan 'Y']
OCCUPANCY_STATUS : ['O' 'I' 'S']
CHANNEL : ['T' 'R' 'B' 'C']
PRODUCT_TYPE : ['FRM']
PROPERTY_TYPE : ['SF' 'PU' 'CP' 'CO' 'MH' 'LH' nan]
LOAN_PURPOSE : ['P' 'N' 'C']
```

Countplot

Show the counts of observations in each categorical bin using bars.

```
fig,ax = plt.subplots(2,3,figsize=(15,10))
for axi,x in zip(ax.flat,categorical_variables):
    sns.countplot(x=loan.DELINQUENT,hue=loan[x],ax=axi)
```



Observations

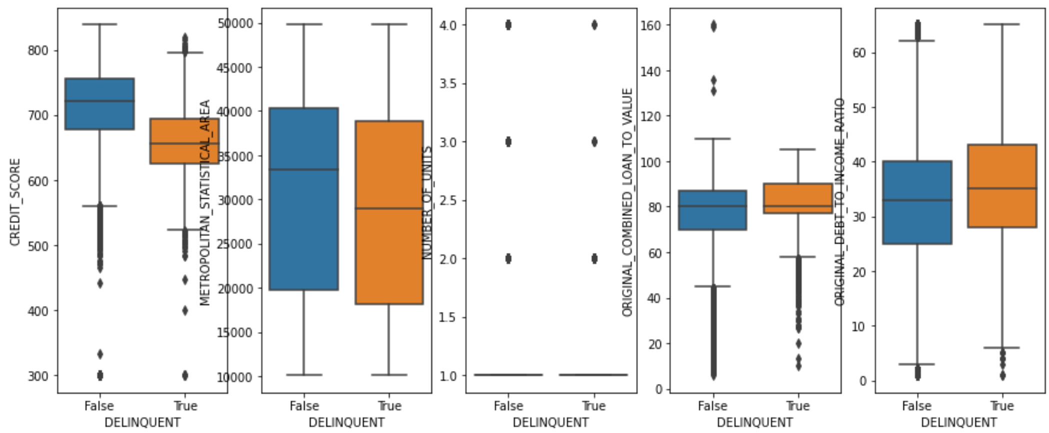
1. PRODUCT_TYPE (FRM – Fixed Rate Mortgage) "FRM" , has impact on outcome(default or non-default)
2. LOAN PURPOSE (Indicates whether the mortgage loan is a (C) Cash- out Refinance mortgage, (N)No Cash-out Refinance mortgage, or a (P)Purchase mortgage) , All(N,C,P) have same amount/count impact on outcome(default)
3. PROPERTY TYPE (Denotes whether the property type secured by the mortgage is a condominium, leasehold, planned unit development (PUD), cooperative share, manufactured home, or Single-Family home.). Single-Family home has impact on outcome default , other have null(no impact)
4. CHANNEL, (R = Retail,B = Broker,C = Correspondent,T = TPO Not Specified),T,R have impact on outcome (default)
5. OCCUPANCY_STATUS , Primary Residence has impact on outcome(default), (Investment Property,Second Home) are no impact on outcome.
6. FIRST_TIME_HOMEBUYER_FLAG , (will reside in the mortgaged property as a primary residence) are not defaulter, (individual who is purchasing the mortgaged property) are found as default

Continuous variables

```
continuous_variables = [i for i in loan.columns if loan[i].dtype != "object"]
continuous_variables.pop()
print(continuous_variables)
```

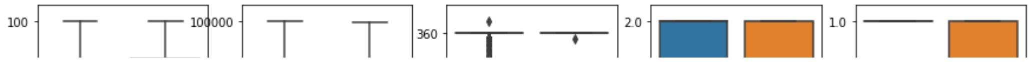
```
['CREDIT_SCORE', 'METROPOLITAN_STATISTICAL_AREA', 'NUMBER_OF_UNITS', 'ORIGINAL_COMBINED_LOAN_TO_VALUE', 'ORIGINAL_DEBT_TO_INCOME_RA
```

```
continuous_variables_1=['CREDIT_SCORE', 'METROPOLITAN_STATISTICAL_AREA', 'NUMBER_OF_UNITS', 'ORIGINAL_COMBINED_LOAN_TO_VALUE', 'ORIGINAL_
fig,ax = plt.subplots(1,5,figsize=(15,6))
for i,x in enumerate(continuous_variables_1):
    sns.boxplot(x=loan.DELINQUENT, y=loan[x] ,ax=ax[i])
```




```
replace_dict = {True: 1, False: 0}
loan.PREPAID.replace(replace_dict, inplace=True)
```

```
continuous_variables_2=['ORIGINAL_LOAN_TO_VALUE', 'POSTAL_CODE', 'ORIGINAL_LOAN_TERM', 'NUMBER_OF_BORROWERS', 'PREPAID']
fig,ax = plt.subplots(1,5,figsize=(15,6))
for i,x in enumerate(continuous_variables_2):
    sns.boxplot(x=loan.DELINQUENT, y=loan[x] ,ax=ax[i])
```

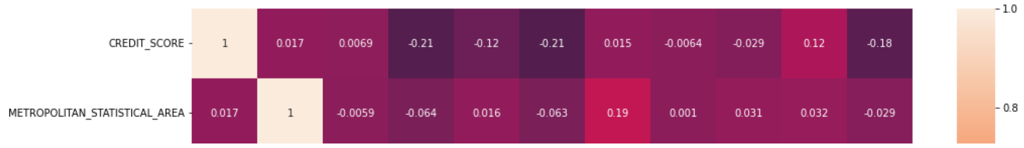


Observations

1. CREDIT_SCORE (A number, prepared by third parties, summarizing the borrower’s creditworthiness) has outliers in both cases(default and non-default)
2. METROPOLITAN_STATISTICAL_AREA(This disclosure will be based on the designation of the Metropolitan Statistical Area) does not have outliers . Both looks same.
3. NUMBER_OF_UNITS(Denotes whether the mortgage is a one, two, three, or four-unit property.) is categorical variable - finite - discrete - units
4. ORIGINAL_COMBINED_LOAN_TO_VALUE(In the case of a purchase mortgage loan, the ratio is obtained by dividing the original mortgage loan amount) - has outliers in both cases
5. ORIGINAL_DEBT_TO_INCOME_RATIO (Disclosure of the debt to income ratio): Default is high compared to rest
6. ORIGINAL_LOAN_TO_VALUE: Default is high compared to rest
7. ORIGINAL_LOAN_TERM:(A calculation of the number of scheduled monthly payments of the mortgage based on the First Payment Date and Maturity Date.)
8. NUMBER_OF_BORROWERS:(The number of Borrower(s) who are obligated to repay the mortgage note secured by the mortgaged property) Has no impact. Both are same.

Correlation-heatmap

```
f, ax = plt.subplots(figsize=(16, 14))
sns.heatmap(loan.corr(), annot=True)
plt.show()
```



Observations

Correlation:

1. ORIGINAL_COMBINED_LOAN_TO_VALUE
2. ORIGINAL_DEBT_TO_INCOME_RATIO
3. ORIGINAL_LOAN_TO_VALUE
4. METROPOLITAN_STATISTICAL_AREA
5. POSTAL_CODE

Missing Values -filling

```
# For rows with missing values, how many loans are delinquent
loan.loc[loan.CREDIT_SCORE.isnull(), 'DELINQUENT'].value_counts()
```

```
False    499
True      29
Name: DELINQUENT, dtype: int64
```

```
loan.CREDIT_SCORE.fillna(value=0, inplace=True)
```

```
#replace with mode (most frequent)
loan.FIRST_TIME_HOMEBUYER_FLAG.fillna(value="N", inplace=True)
```

```
#replace with median
loan.METROPOLITAN_STATISTICAL_AREA.fillna(value=loan.METROPOLITAN_STATISTICAL_AREA.median(), inplace=True)
```

```
#replace with median
loan.ORIGINAL_COMBINED_LOAN_TO_VALUE.fillna(loan.ORIGINAL_COMBINED_LOAN_TO_VALUE.median(),inplace=True)
```

```
#replace with median
loan.ORIGINAL_LOAN_TO_VALUE.fillna(loan.ORIGINAL_LOAN_TO_VALUE.median(),inplace=True)
```

```
#replace with median
loan.NUMBER_OF_UNITS.fillna(value=1,inplace=True)
```

```
loan = loan[~loan.POSTAL_CODE.isnull()].reset_index(drop=True, inplace=False)
```

```
#replace with median
loan.NUMBER_OF_BORROWERS.fillna(value=2, inplace=True)
```

```
#replace with mode
loan.PROPERTY_TYPE.fillna(value='SF', inplace=True)
```

```
#replace with median
loan.ORIGINAL_DEBT_TO_INCOME_RATIO.fillna(value=loan.ORIGINAL_DEBT_TO_INCOME_RATIO.median(), inplace=True)
```

```
loan.isnull().sum().sort_values(ascending=False)
```

```
CREDIT_SCORE                     0
PRODUCT_TYPE                     0
PREPAID                          0
NUMBER_OF_BORROWERS              0
ORIGINAL_LOAN_TERM               0
LOAN_PURPOSE                     0
POSTAL_CODE                      0
PROPERTY_TYPE                    0
CHANNEL                          0
FIRST_TIME_HOMEBUYER_FLAG        0
ORIGINAL_LOAN_TO_VALUE           0
ORIGINAL_DEBT_TO_INCOME_RATIO    0
ORIGINAL_COMBINED_LOAN_TO_VALUE  0
OCCUPANCY_STATUS                 0
NUMBER_OF_UNITS                  0
METROPOLITAN_STATISTICAL_AREA    0
```

```
DELINQUENT                       0
dtype: int64
```

6. Categorical Data Encoding

```
le = LabelEncoder()
for feature in loan:
    if (loan[feature].dtype=="object") and (loan[feature].nunique()==2):
        loan[feature] = le.fit_transform(loan[feature])
```

```
loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99992 entries, 0 to 99991
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   CREDIT_SCORE                    99992 non-null  float64
 1   FIRST_TIME_HOMEBUYER_FLAG       99992 non-null  int64  
 2   METROPOLITAN_STATISTICAL_AREA   99992 non-null  float64
 3   NUMBER_OF_UNITS                 99992 non-null  float64
 4   OCCUPANCY_STATUS                99992 non-null  object 
 5   ORIGINAL_COMBINED_LOAN_TO_VALUE 99992 non-null  float64
 6   ORIGINAL_DEBT_TO_INCOME_RATIO   99992 non-null  float64
 7   ORIGINAL_LOAN_TO_VALUE          99992 non-null  float64
 8   CHANNEL                         99992 non-null  object 
 9   PRODUCT_TYPE                    99992 non-null  object 
 10  PROPERTY_TYPE                   99992 non-null  object 
 11  POSTAL_CODE                     99992 non-null  float64
 12  LOAN_PURPOSE                    99992 non-null  object 
 13  ORIGINAL_LOAN_TERM              99992 non-null  int64  
 14  NUMBER_OF_BORROWERS             99992 non-null  float64
 15  PREPAID                         99992 non-null  int64  
 16  DELINQUENT                      99992 non-null  bool   
dtypes: bool(1), float64(8), int64(3), object(5)
memory usage: 12.3+ MB
```

```
loan = pd.get_dummies(loan,columns=[i for i in loan.columns if loan[i].dtypes=='object'],drop_first=True)
```

```
replace_dict = {True: 1, False: 0}
loan.DELINQUENT.replace(replace_dict, inplace=True)
```

```
loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99992 entries, 0 to 99991
Data columns (total 24 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   CREDIT_SCORE                    99992 non-null  float64
 1   FIRST_TIME_HOMEBUYER_FLAG       99992 non-null  int64  
 2   METROPOLITAN_STATISTICAL_AREA   99992 non-null  float64
 3   NUMBER_OF_UNITS                 99992 non-null  float64
 4   ORIGINAL_COMBINED_LOAN_TO_VALUE 99992 non-null  float64
 5   ORIGINAL_DEBT_TO_INCOME_RATIO   99992 non-null  float64
 6   ORIGINAL_LOAN_TO_VALUE          99992 non-null  float64
 7   POSTAL_CODE                     99992 non-null  float64
 8   ORIGINAL_LOAN_TERM              99992 non-null  int64  
 9   NUMBER_OF_BORROWERS             99992 non-null  float64
 10  PREPAID                         99992 non-null  int64  
 11  DELINQUENT                      99992 non-null  int64  
 12  OCCUPANCY_STATUS_O              99992 non-null  uint8  
 13  OCCUPANCY_STATUS_S              99992 non-null  uint8  
 14  CHANNEL_C                       99992 non-null  uint8  
 15  CHANNEL_R                       99992 non-null  uint8  
 16  CHANNEL_T                       99992 non-null  uint8  
 17  PROPERTY_TYPE_CP                99992 non-null  uint8  
 18  PROPERTY_TYPE_LH                99992 non-null  uint8  
 19  PROPERTY_TYPE_MH                99992 non-null  uint8  
 20  PROPERTY_TYPE_PU                99992 non-null  uint8  
 21  PROPERTY_TYPE_SF                99992 non-null  uint8  
 22  LOAN_PURPOSE_N                  99992 non-null  uint8  
 23  LOAN_PURPOSE_P                  99992 non-null  uint8  
dtypes: float64(8), int64(4), uint8(12)
memory usage: 10.3 MB
```

```
loan.DELINQUENT.value_counts()
```

```
0    96401
1     3591
```

```
Name: DELINQUENT, dtype: int64
```

7. Scalling and Spliting

Standardize features by removing the mean and scaling to unit variance

```
scaler = StandardScaler()
scaler.fit_transform(loan)
```

```
array([[-0.7091103 , -0.32686515,  0.4082928 , ...,  0.46576415,
        -0.7351999 ,  1.15610953],
       [ 0.12679463, -0.32686515, -1.33626423, ...,  0.46576415,
        -0.7351999 ,  1.15610953],
       [-0.15184035, -0.32686515,  0.43331525, ...,  0.46576415,
         1.36017429, -0.86496995],
       ...,
       [-0.24471867,  3.05936565, -0.44095369, ..., -2.1470094 ,
        -0.7351999 ,  1.15610953],
       [-1.10716027, -0.32686515,  0.21493758, ...,  0.46576415,
        -0.7351999 ,  1.15610953],
       [-0.12530368, -0.32686515,  0.93148931, ...,  0.46576415,
        -0.7351999 , -0.86496995]])
```

Split arrays or matrices into random train and test subsets.

SMOTE is an oversampling algorithm that relies on the concept of nearest neighbors to create its synthetic data

```
from collections import Counter
X=loan.drop(['DELINQUENT'],axis='columns')
Y=loan['DELINQUENT']
sm = SMOTE()
X, Y = sm.fit_resample(X, Y)
print('Resampled dataset shape %s' % Counter(Y))
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.30)
print("train data length:",len(X_train))
print("test data length:",len(X_test))
```

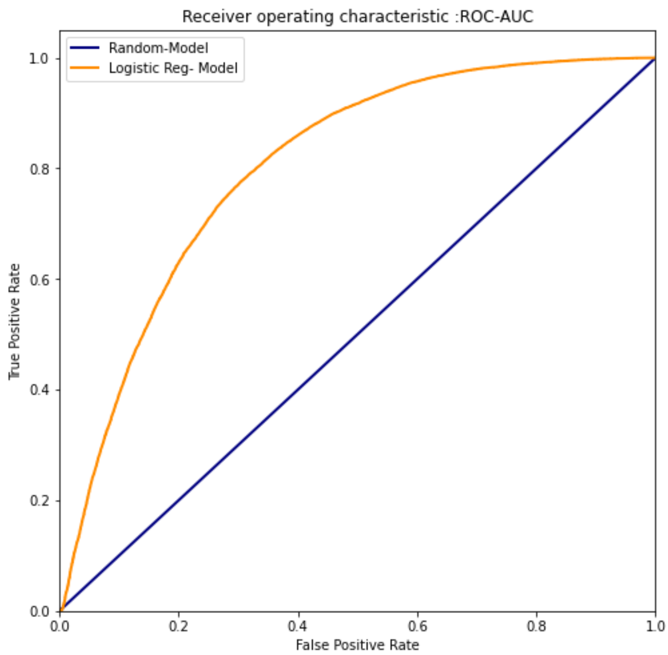
```
Resampled dataset shape Counter({0: 96401, 1: 96401})
train data length: 134961
test data length: 57841
```

8. ML - LogisticRegression -Model

```
lg_model = LogisticRegression(C=1.0)
lg_model.fit(X_train,Y_train)
print("LogisticRegression")
print("train score",lg_model.score(X_train,Y_train))
print("test score",lg_model.score(X_test,Y_test))
print("log-loss:",log_loss(Y_test,lg_model.predict_proba(X_test)))

print(confusion_matrix(Y_test,lg_model.predict(X_test)))
print(lg_model.get_params())
sns.heatmap(confusion_matrix(Y_test,lg_model.predict(X_test)), annot=True)
plt.show()
print(classification_report(Y_test, lg_model.predict(X_test)))
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:818: Converg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
LogisticRegression
train score 0.7315668971036077
test score 0.734928510917861
log-loss: 0.5637323845474461
[[20608  8316]
 [ 7016 21901]]
{'C': 1.0, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_sca
```


```
from sklearn.metrics import roc_curve,auc
fpr,tpr, thresholds = roc_curve(Y_test,lg_model.predict_proba(X_test)[:,1])
plt.figure(figsize=(8,8))
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.plot([0,1],[0,1],color="navy",lw=2,label="Random-Model")
plt.plot(fpr,tpr,color="darkorange",lw=2, label="Logistic Reg- Model")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic :ROC-AUC")
plt.legend()
plt.show()
print("Computed Area Under the Curve (AUC)",auc(fpr, tpr))
```

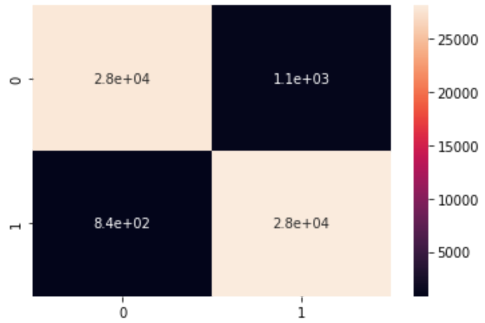



```
Computed Area Under the Curve (AUC) 0.8025414520857164
```

9. ML - DecisionTree Classifier Model

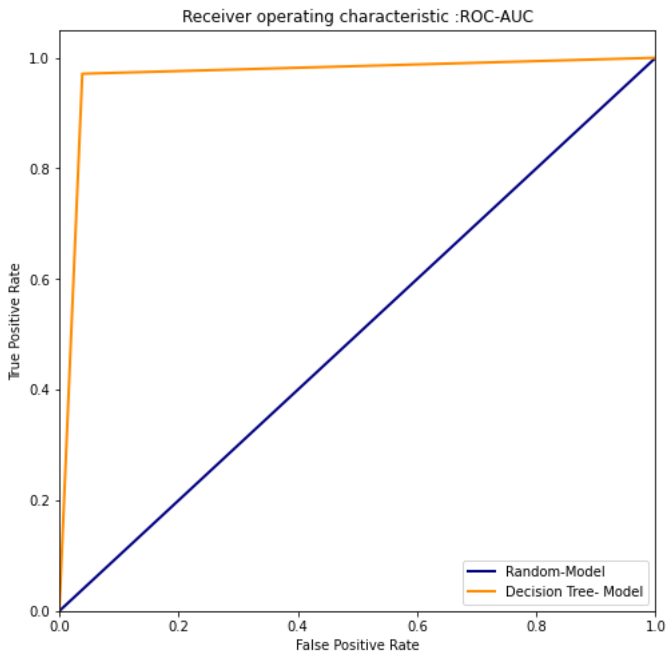
```
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train,Y_train)
print("DecisionTree")
print("Train Score:",dt_model.score(X_train,Y_train))
print("Test Score:",dt_model.score(X_test,Y_test))

print(confusion_matrix(Y_test,dt_model.predict(X_test)))
sns.heatmap(confusion_matrix(Y_test,dt_model.predict(X_test)), annot=True)
plt.show()
print(classification_report(Y_test, dt_model.predict(X_test)))
```

```
DecisionTree
Train Score: 1.0
Test Score: 0.966321467471171
[[27811  1113]
 [  835 28082]]
```




```
              precision    recall  f1-score   support

           0       0.97      0.96      0.97     28924
           1       0.96      0.97      0.97     28917
```

```
from sklearn.metrics import roc_curve,auc
fpr,tpr, thresholds = roc_curve(Y_test,dt_model.predict_proba(X_test)[:,1])
plt.figure(figsize=(8,8))
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.plot([0,1],[0,1],color="navy",lw=2,label="Random-Model")
plt.plot(fpr,tpr,color="darkorange",lw=2, label="Decision Tree- Model")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic :ROC-AUC")
plt.legend()
plt.show()
print("Computed Area Under the Curve (AUC)",auc(fpr, tpr))
```




```
Computed Area Under the Curve (AUC) 0.966322048640665
```

10. ML - Random Forest Model

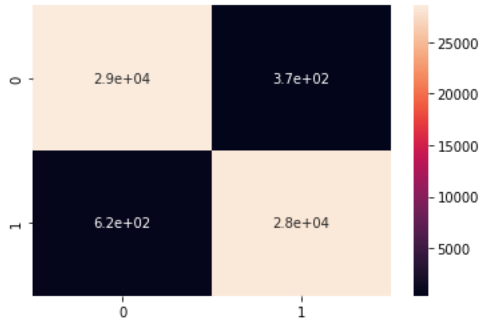
```
rf_model = RandomForestClassifier()
rf_model.fit(X_train,Y_train)
print("Random Forest")
print("train score:",rf_model.score(X_train,Y_train))
print("test score:",rf_model.score(X_test,Y_test))
```

```
Random Forest
train score: 0.999985180904113
test score: 0.9828149582476098
```

```
print(classification_report(Y_test, rf_model.predict(X_test)))
print(confusion_matrix(Y_test,rf_model.predict(X_test)))
```

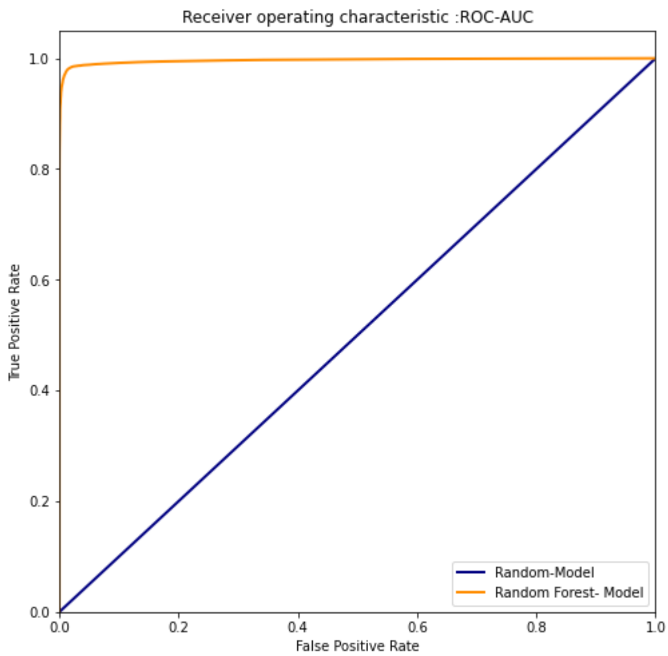
```
sns.heatmap(confusion_matrix(Y_test,rf_model.predict(X_test)), annot=True)
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98     28924
           1       0.99      0.98      0.98     28917

    accuracy                           0.98     57841
   macro avg       0.98      0.98      0.98     57841
weighted avg       0.98      0.98      0.98     57841

[[28555   369]
 [  625 28292]]
```



Compute Receiver operating characteristic (ROC).

```
from sklearn.metrics import roc_curve,auc
fpr,tpr, thresholds = roc_curve(Y_test,rf_model.predict_proba(X_test)[:,1])
plt.figure(figsize=(8,8))
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.plot([0,1],[0,1],color="navy",lw=2,label="Random-Model")
plt.plot(fpr,tpr,color="darkorange",lw=2, label="Random Forest- Model")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic :ROC-AUC")
plt.legend()
plt.show()
print("Computed Area Under the Curve (AUC)",auc(fpr, tpr))
```




```
Computed Area Under the Curve (AUC) 0.9964060881604084
```

```
print(rf_model.feature_importances_)
```

```
[1.09914981e-01 6.44913242e-03 2.52640921e-02 2.24606707e-03
 5.26127810e-02 2.59401396e-02 6.64978243e-02 3.25777024e-02
 7.69703350e-04 1.29988788e-01 3.89245978e-01 3.66132413e-03
 1.23465832e-03 2.15297157e-04 4.17023600e-02 4.45076683e-02
 4.08675145e-05 1.56676409e-05 4.06510723e-04 7.76287034e-03
 4.74445161e-03 2.35666954e-02 3.06344400e-02]
```

Summary

Logistic Regression:

1. Accuracy: 0.73
2. AUC:0.79

Decision Tree:

1. Accuracy : 0.97
2. AUC:0.96

Random Forest :

1. Accuracy: 0.98
2. AUC :0.99