Install Package

In [1]:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
from sklearn import ensemble
from sklearn.feature_selection import VarianceThreshold
from sklearn.manifold import TSNE
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
from matplotlib import pyplot
```

In [2]:

```python
import xgboost as xgb
```

```
C:\Users\ADMIN\Anaconda3\lib\site-packages\dask\config.py:168: YAMLLoadWarnin
g: calling yaml.load() without Loader=... is deprecated, as the default Loade
r is unsafe. Please read https://msg.pyyaml.org/load for full details.
  data = yaml.load(f.read()) or {}
C:\Users\ADMIN\Anaconda3\lib\site-packages\dask\dataframe\utils.py:13: Future
Warning: pandas.util.testing is deprecated. Use the functions in the public A
PI at pandas.testing instead.
  import pandas.util.testing as tm
C:\Users\ADMIN\Anaconda3\lib\site-packages\distributed\config.py:20: YAMLLoad
Warning: calling yaml.load() without Loader=... is deprecated, as the default
Loader is unsafe. Please read https://msg.pyyaml.org/load for full details.
  defaults = yaml.load(f)
```

In [3]:

```python
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
```

Load Data

all missing values were removed. Other missing values are imputed using missRanger() in R.

In [4]:

```python
traindf = pd.read_csv("D:/Github/Risk Modelling/Home_Credit/imputed_applicatio
n_Train3.csv", index_col=0)
```

In [84]:

```python
testdf =  pd.read_csv("D:/Github/Risk Modelling/Home_Credit/imputed_applicatio
n_Test3.csv",  index_col=0)
```

In [6]:
```
traindf.drop('SK_ID_CURR', axis=1, inplace=True)
#testdf.drop('SK_ID_CURR', axis=1, inplace=True)
```

In [7]:
```
traindf.reset_index(drop=True, inplace=True)
```

In [63]:
```
testdf.reset_index(drop=True, inplace=True)
```

Check the shape of Train and Test

In [9]:
```
traindf.shape
```

Out[9]:
```
(307511, 121)
```

In [10]:
```
testdf.shape
```

Out[10]:
```
(48744, 121)
```

Check If Train and Test still have any Missing Values.

In [11]:
```
traindf.isnull().sum().sum()
```

Out[11]:
```
0
```

In [12]:
```
testdf.isnull().sum().sum()
```

Out[12]:
```
0
```

In [13]:
```
Y_train = traindf['TARGET']
traindf.drop('TARGET', axis=1, inplace=True)
```

PreProcessing

Dimensionality Reduction / Feature Selection

In [14]:
```
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
train_numerical = traindf.select_dtypes(include=numerics)
```

Correlation

In [15]:

```
# Calculate the correlation matrix and take the absolute value
corr_matrix = train_numerical.corr().abs()
# Create a True/False mask and apply it
mask_highcor = np.triu(np.ones_like(corr_matrix, dtype=bool))
tri_df = corr_matrix.mask(mask_highcor)
# List column names of highly correlated features (r > 0.95)
to_drop_corr = [c for c in tri_df.columns if any(tri_df[c] >  0.95)]
to_drop_corr
```

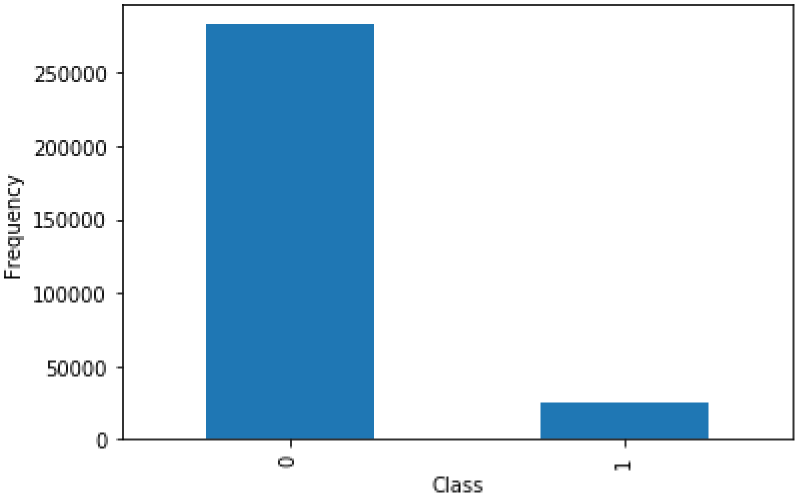
Out[15]:

```
['AMT_CREDIT',
 'DAYS_EMPLOYED',
 'REGION_RATING_CLIENT',
 'APARTMENTS_AVG',
 'BASEMENTAREA_AVG',
 'YEARS_BEGINEXPLUATATION_AVG',
 'YEARS_BUILD_AVG',
 'COMMONAREA_AVG',
 'ELEVATORS_AVG',
 'ENTRANCES_AVG',
 'FLOORSMAX_AVG',
 'FLOORSMIN_AVG',
 'LANDAREA_AVG',
 'LIVINGAPARTMENTS_AVG',
 'LIVINGAREA_AVG',
 'NONLIVINGAPARTMENTS_AVG',
 'NONLIVINGAREA_AVG',
 'APARTMENTS_MODE',
 'BASEMENTAREA_MODE',
 'YEARS_BEGINEXPLUATATION_MODE',
 'YEARS_BUILD_MODE',
 'COMMONAREA_MODE',
 'ELEVATORS_MODE',
 'ENTRANCES_MODE',
 'FLOORSMAX_MODE',
 'FLOORSMIN_MODE',
 'LANDAREA_MODE',
 'LIVINGAPARTMENTS_MODE',
 'LIVINGAREA_MODE',
 'NONLIVINGAPARTMENTS_MODE',
 'NONLIVINGAREA_MODE',
 'APARTMENTS_MEDI',
 'OBS_30_CNT_SOCIAL_CIRCLE']
```

Check If Dataset is Imbalance

In [16]:
```
Y_train.value_counts().plot.bar()
plt.xlabel('Class')
plt.ylabel('Frequency')
Y_train.value_counts()
```

Out[16]:
```
0    282686
1     24825
Name: TARGET, dtype: int64
```



Random Forest for Feature Selection (aka. Variable Importance)

In [17]:
```
# Fit the random forest model to the training data
rf = ensemble.RandomForestClassifier(random_state=123, n_jobs = -1)
rf.fit(train_numerical, Y_train)
```

Out[17]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=-1, oob_score=False, random_state=123, verbose=
0,
                       warm_start=False)
```

In [18]:
```
mask_rf = rf.feature_importances_ > 0.1
mask_rf
```

Out[18]:
```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False])
```

Variable Importance could not provide useful info for this Dataset

Low Variance Features

In [19]:
```
train_numerical_normalized = normalize(train_numerical)
```

In [20]:
```
train_numerical_normalized = pd.DataFrame(train_numerical_normalized, columns=
train_numerical.columns)
```

In [21]:
```
train_numerical_normalized.describe()
```

Out[21]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRIC
count	3.075110e+05	307511.000000	307511.000000	307511.000000	307511.00000
mean	7.582182e-07	0.250196	0.680537	0.035601	0.60969
std	1.718293e-06	0.154356	0.097865	0.013674	0.08919
min	0.000000e+00	0.008285	0.004808	0.000224	0.00388
25%	0.000000e+00	0.137588	0.662149	0.025049	0.58066
50%	0.000000e+00	0.209651	0.701143	0.032915	0.63306
75%	9.049948e-07	0.321184	0.740990	0.041307	0.66552
max	8.554940e-05	0.999981	0.945914	0.088115	0.98438

8 rows × 104 columns

In [22]:

```
# Create a VarianceThreshold feature selector
sel =VarianceThreshold(threshold=10**-3)
# Fit the selector to normalized head_df
sel.fit(train_numerical_normalized / train_numerical_normalized.mean())
# Create a boolean mask
mask_lowvar = sel.get_support()
```

In [23]:

```
mask_lowvar
```

Out[23]:

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True])
```

PreProcessing

Convert Categorical Variables into Numerical using OneHotEncoding(OHE) and LabelEncoder

In [24]:

```
categorical_mask = (traindf.dtypes == object)
```

In [25]:

```
categorical_columns = traindf.columns[categorical_mask].tolist()
categorical_columns
```

Out[25]:

```
['NAME_CONTRACT_TYPE',
 'CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'NAME_TYPE_SUITE',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'OCCUPATION_TYPE',
 'WEEKDAY_APPR_PROCESS_START',
 'ORGANIZATION_TYPE',
 'FONDKAPREMONT_MODE',
 'HOUSETYPE_MODE',
 'WALLSMATERIAL_MODE',
 'EMERGENCYSTATE_MODE']
```

In [26]:

```
Categorical_Level  =traindf[categorical_columns].nunique().sort_values(ascendi
ng=False)
Categorical_Level
```

Out[26]:

```
ORGANIZATION_TYPE             57
OCCUPATION_TYPE               18
NAME_INCOME_TYPE               8
WALLSMATERIAL_MODE             7
WEEKDAY_APPR_PROCESS_START     7
NAME_TYPE_SUITE                7
NAME_HOUSING_TYPE              6
NAME_FAMILY_STATUS             6
NAME_EDUCATION_TYPE            5
FONDKAPREMONT_MODE             4
HOUSETYPE_MODE                 3
CODE_GENDER                    3
EMERGENCYSTATE_MODE            2
FLAG_OWN_REALTY                2
FLAG_OWN_CAR                   2
NAME_CONTRACT_TYPE             2
dtype: int64
```

If the Variable has more than 5 levels then It would be applied LabelEncoder, otherwise applied OHE

In [27]:

```
OHE_List = Categorical_Level[Categorical_Level<=5].index.tolist()
LE_List = Categorical_Level[Categorical_Level>5].index.tolist()
```

In [28]:

```
le = LabelEncoder()
# Apply LabelEncoder to categorical columns
df_le = traindf[LE_List].apply(lambda x: le.fit_transform(x))
```

In [29]:

```
df_ohe = pd.get_dummies(traindf[OHE_List])
```

Train Data after Converting

In [33]:

```
traindf.drop(categorical_columns, axis=1, inplace=True)
```

In [34]:

```
traindf = pd.concat([traindf,df_ohe, df_le], axis=1)
```

In [35]:

```
traindf.head()
```

Out[35]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	RE
0	0	202500.0	406597.5	24700.5	351000.0	
1	0	270000.0	1293502.5	35698.5	1129500.0	
2	0	67500.0	135000.0	6750.0	135000.0	
3	0	135000.0	312682.5	29686.5	297000.0	
4	0	121500.0	513000.0	21865.5	513000.0	

5 rows × 135 columns

In [36]:

```
traindf.columns
```

Out[36]:

```
Index(['CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY',
       'AMT_GOODS_PRICE', 'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH',
       'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH',
       ...
       'NAME_CONTRACT_TYPE_Cash loans', 'NAME_CONTRACT_TYPE_Revolving loans',
       'ORGANIZATION_TYPE', 'OCCUPATION_TYPE', 'NAME_INCOME_TYPE',
       'WALLSMATERIAL_MODE', 'WEEKDAY_APPR_PROCESS_START', 'NAME_TYPE_SUITE',
       'NAME_HOUSING_TYPE', 'NAME_FAMILY_STATUS'],
      dtype='object', length=135)
```

Model Building

1. Xgboost (with Tuning Hyperparameter)

In [37]:

```
xgbpipeline = Pipeline([
        ('scale', StandardScaler()),
        ('clf', xgb.XGBClassifier())])
```

In [39]:

```
# Create the parameter grid
gbm_param_grid = {
    'clf__learning_rate': np.arange(0.05, 1, 0.05),
    'clf__max_depth': np.arange(3, 15, 1),
    'clf__n_estimators': np.arange(50, 300, 50),
    'clf__clf_colsample_bytree' : [0.2,0.4,0.6,0.8,1.0]
}
```

In [40]:
```
# Perform RandomizedSearchCV
randomized_roc_auc = RandomizedSearchCV(estimator=xgbpipeline, param_distribut
ions=gbm_param_grid,
                                        n_iter=10, scoring='roc_auc', cv=5,
                                        random_state=123, n_jobs = -2)
```

In [41]:
```
# Fit the estimator
randomized_roc_auc.fit(traindf,Y_train)
```

Out[41]:
```
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=Pipeline(memory=None,
                                      steps=[('scale',
                                              StandardScaler(copy=True,
                                                             with_mean=True,
                                                             with_std=True)),
                                             ('clf',
                                              XGBClassifier(base_score=None,
                                                            booster=None,
                                                            colsample_bylevel
=None,
                                                            colsample_bynode=
None,
                                                            colsample_bytree=
None,
                                                            gamma=None,
                                                            gpu_id=None,
                                                            importance_type
='gain',
                                                            interaction_const
raints=None,
                                                            learning_rate=
N...
                   param_distributions={'clf__clf_colsample_bytree': [0.2, 0.
4,
                                                                      0.6, 0.
8,
                                                                      1.0],
                                        'clf__learning_rate': array([0.05, 0.
1 , 0.15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 , 0.55,
       0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95]),
                                        'clf__max_depth': array([ 3,  4,  5,
6,  7,  8,  9, 10, 11, 12, 13, 14]),
                                        'clf__n_estimators': array([ 50, 100,
150, 200, 250])},
                   pre_dispatch='2*n_jobs', random_state=123, refit=True,
                   return_train_score=False, scoring='roc_auc', verbose=0)
```

Best Estimator of XGB Model

In [42]:

```
# Compute metrics
print(randomized_roc_auc.best_estimator_)
```

```
Pipeline(memory=None,
         steps=[('scale',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('clf',
                 XGBClassifier(base_score=0.5, booster=None,
                               clf_colsample_bytree=0.8, colsample_bylevel=1,
                               colsample_bynode=1, colsample_bytree=1, gamma=
0,
                               gpu_id=-1, importance_type='gain',
                               interaction_constraints=None, learning_rate=0.
05,
                               max_delta_step=0, max_depth=9,
                               min_child_weight=1, missing=nan,
                               monotone_constraints=None, n_estimators=250,
                               n_jobs=0, num_parallel_tree=1,
                               objective='binary:logistic', random_state=0,
                               reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
                               subsample=1, tree_method=None,
                               validate_parameters=False, verbosity=None))],
         verbose=False)
```

In [43]:

```
print(randomized_roc_auc.best_score_)
```

```
0.8447346765271921
```

In [44]:

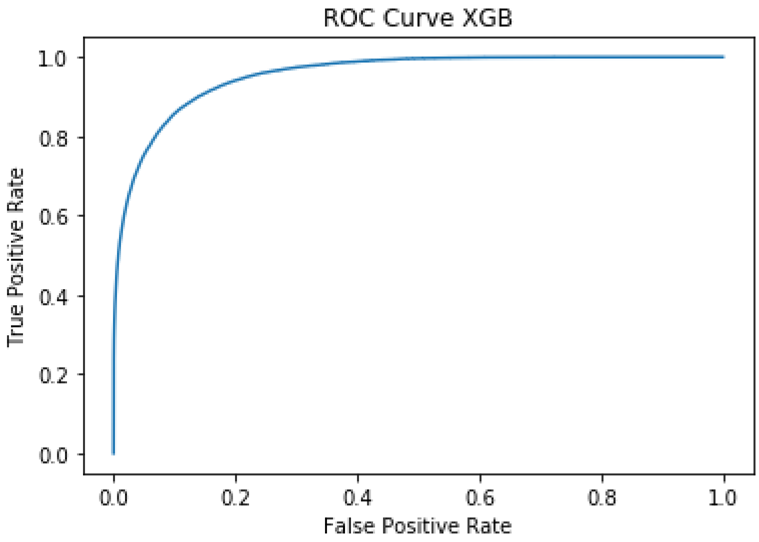
```
model_xgb_probs = randomized_roc_auc.predict_proba(traindf)
```

In [45]:

```
scores = randomized_roc_auc.predict_proba(traindf)[:,1]

fpr, tpr, thresholds = roc_curve(Y_train, scores)

roc_auc = roc_auc_score(Y_train, scores)

print("AUC of ROC Curve:", roc_auc)
```

```
AUC of ROC Curve: 0.9551377164186577
```

In [46]:
```
plt.plot(fpr, tpr)
plt.title("ROC Curve XGB")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```



In [47]:
```
predictions = [round(value) for value in scores]
accuracy = accuracy_score(Y_train, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 94.24%
```

Predict for Test Data

In [85]:
```
testdf.shape
```

Out[85]:
```
(48744, 121)
```

In [86]:
```
SK_ID_CURR_Col = testdf['SK_ID_CURR']
```

In [87]:
```
testdf.drop('SK_ID_CURR', axis=1, inplace=True)
```

Transform

In [88]:
```
df_ohe_test = pd.get_dummies(testdf[OHE_List])
```

In [89]:
```
df_le_test = testdf[LE_List].apply(lambda x: le.fit_transform(x))
```

In [90]:
```
testdf.drop(categorical_columns, axis=1, inplace=True)
```

In [91]:
```
testdf = pd.concat([testdf, df_ohe_test, df_le_test], axis=1)
```

```
In [92]: testdf.shape

Out[92]: (48744, 135)
```

```
In [93]: test_cols = testdf.columns.tolist()
         train_cols = traindf.columns.tolist()
```

Check if Train and Test have same set of Variables

```
In [94]: list(set(train_cols) - set(test_cols))

Out[94]: []
```

Fit XGB to Test Data

```
In [95]: score_test = randomized_roc_auc.predict_proba(testdf)[:,1]
```

```
In [ ]: submit_df1 = pd.DataFrame({'SK_ID_CURR':SK_ID_CURR_Col, 'TARGET': score_test})
```

```
In [ ]: submit_df1.to_csv("submit_df1.csv",index=False)
```

The Score is 73.2