1. Import Library

In [9]:
```
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 [10]:
```
import xgboost as xgb
```

In [11]:
```
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
```

2. Load Data

Variables have more than 60% of missing were removed (17 Variables). Other missing values are imputed using missRanger() in R.

In [12]:
```
traindf = pd.read_csv("D:/Github/Risk Modelling/Home_Credit/imputed_applicatio
n_train.csv", index_col=0)
```

In [71]:
```
testdf =  pd.read_csv("D:/Github/Risk Modelling/Home_Credit/imputed_applicatio
n_test.csv",  index_col=0)
```

In [14]:
```
traindf.reset_index(drop=True, inplace=True)
```

In [72]:
```
testdf.reset_index(drop=True, inplace=True)
```

In [16]:
```
traindf.drop('SK_ID_CURR', axis=1, inplace=True)
#testdf.drop('SK_ID_CURR', axis=1, inplace=True)
```

Check the shape of Train and Test

In [23]:
```
traindf.shape
```
Out[23]:
```
(307511, 104)
```

In [24]:
```
testdf.shape
```
Out[24]:
```
(48744, 104)
```

Check If Train and Test still have any Missing Values.

In [25]:
```
traindf.isnull().sum().sum()
```
Out[25]:
```
0
```

In [26]:
```
testdf.isnull().sum().sum()
```
Out[26]:
```
0
```

In [27]:
```
Y_train = traindf['TARGET']
traindf.drop('TARGET', axis=1, inplace=True)
```

3. PreProcessing

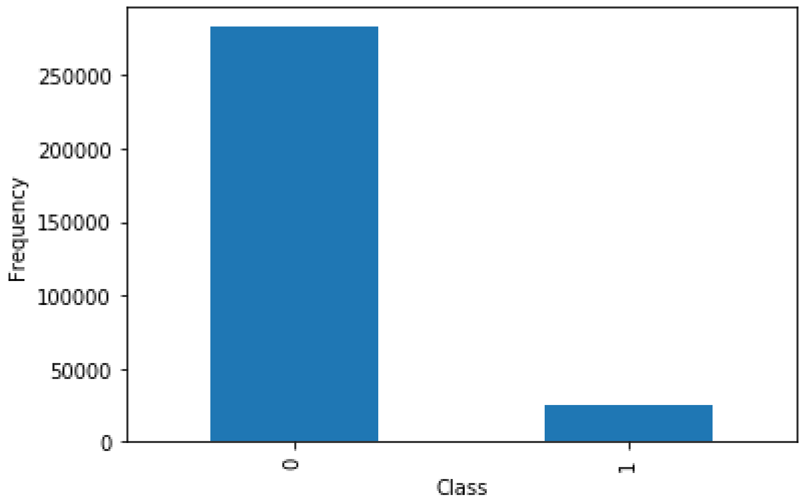
Dimensionality Reduction / Feature Selection

In [28]:
```
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
train_numerical = traindf.select_dtypes(include=numerics)
```

Check If Dataset is Imbalance

In [30]:
```
Y_train.value_counts().plot.bar()
plt.xlabel('Class')
plt.ylabel('Frequency')
Y_train.value_counts()
```

Out[30]:
```
0    282686
1     24825
Name: TARGET, dtype: int64
```



Correlation

In [29]:
```
# 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[29]:
```
['AMT_CREDIT',
 'DAYS_EMPLOYED',
 'REGION_RATING_CLIENT',
 'APARTMENTS_AVG',
 'BASEMENTAREA_AVG',
 'YEARS_BEGINEXPLUATATION_AVG',
 'ELEVATORS_AVG',
 'ENTRANCES_AVG',
 'FLOORSMAX_AVG',
 'LANDAREA_AVG',
 'LIVINGAREA_AVG',
 'NONLIVINGAREA_AVG',
 'APARTMENTS_MODE',
 'BASEMENTAREA_MODE',
 'YEARS_BEGINEXPLUATATION_MODE',
 'ELEVATORS_MODE',
 'ENTRANCES_MODE',
 'FLOORSMAX_MODE',
 'LANDAREA_MODE',
 'LIVINGAREA_MODE',
 'NONLIVINGAREA_MODE',
 'OBS_30_CNT_SOCIAL_CIRCLE']
```

Random Forest for Feature Selection (aka. Variable Importance)

In [31]:
```
# 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[31]:
```
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 [32]:
```
mask_rf = rf.feature_importances_ > 0.1
mask_rf
```

Out[32]:
```
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])
```

Variable Importance could not provide useful info for this Dataset

Low Variance Features

In [33]:
```
train_numerical_normalized = normalize(train_numerical)
```

In [34]:
```
train_numerical_normalized = pd.DataFrame(train_numerical_normalized, columns=
train_numerical.columns)
```

In [35]:
```
train_numerical_normalized.describe()
```

Out[35]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICI
count	3.075110e+05	307511.000000	307511.000000	307511.000000	307511.00000
mean	7.582951e-07	0.250205	0.680554	0.035602	0.60966
std	1.718599e-06	0.154365	0.097869	0.013674	0.08920
min	0.000000e+00	0.008285	0.004808	0.000224	0.00388
25%	0.000000e+00	0.137588	0.662169	0.025049	0.58062
50%	0.000000e+00	0.209656	0.701153	0.032916	0.63304
75%	9.049948e-07	0.321186	0.741016	0.041307	0.66551
max	8.554940e-05	0.999981	0.945914	0.088115	0.98438

8 rows × 88 columns

In [36]:
```
# 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 [37]:
```
mask_lowvar
```

Out[37]:
```
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])
```

Transform Variables

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

In [38]:
```
categorical_mask = (traindf.dtypes == object)
```

In [39]:
```
categorical_columns = traindf.columns[categorical_mask].tolist()
categorical_columns
```

Out[39]:
```
['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',
 'HOUSETYPE_MODE',
 'WALLSMATERIAL_MODE',
 'EMERGENCYSTATE_MODE']
```

In [40]:
```
Categorical_Level  =traindf[categorical_columns].nunique().sort_values(ascendi
ng=False)
Categorical_Level
```

Out[40]:
```
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
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 [41]:
```
OHE_List = Categorical_Level[Categorical_Level<=5].index.tolist()
LE_List = Categorical_Level[Categorical_Level>5].index.tolist()
```

In [42]:
```
le = LabelEncoder()
# Apply LabelEncoder to categorical columns
df_le = traindf[LE_List].apply(lambda x: le.fit_transform(x))
```

In [43]:
```
df_ohe = pd.get_dummies(traindf[OHE_List])
```

Train Data after Converting

In [44]:
```
traindf.drop(categorical_columns, axis=1, inplace=True)
```

In [45]:
```
traindf = pd.concat([traindf,df_ohe, df_le], axis=1)
```

In [46]:
```
traindf.drop(to_drop_corr, axis=1, inplace=True)
```

In [47]:
```
traindf.head()
```

Out[47]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULA
0	0	202500.0	24700.5	351000.0	
1	0	270000.0	35698.5	1129500.0	
2	0	67500.0	6750.0	135000.0	
3	0	135000.0	29686.5	297000.0	
4	0	121500.0	21865.5	513000.0	

5 rows × 93 columns

In [48]:
```
traindf.columns
```

Out[48]:
```
Index(['CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
       'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_REGISTRATION',
       'DAYS_ID_PUBLISH', 'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE',
       'FLAG_CONT_MOBILE', 'FLAG_PHONE', 'FLAG_EMAIL', 'CNT_FAM_MEMBERS',
       'REGION_RATING_CLIENT_W_CITY', 'HOUR_APPR_PROCESS_START',
       'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION',
       'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY',
       'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'EXT_SOURCE_1',
       'EXT_SOURCE_2', 'EXT_SOURCE_3', 'APARTMENTS_MEDI', 'BASEMENTAREA_MED
I',
       'YEARS_BEGINEXPLUATATION_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_MEDI',
       'FLOORSMAX_MEDI', 'LANDAREA_MEDI', 'LIVINGAREA_MEDI',
       'NONLIVINGAREA_MEDI', 'TOTALAREA_MODE', 'DEF_30_CNT_SOCIAL_CIRCLE',
       'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE',
       'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3',
       'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6',
       'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9',
       'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12',
       'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15',
       'FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17', 'FLAG_DOCUMENT_18',
       'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21',
       'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY',
       'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON',
       'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR',
       'NAME_EDUCATION_TYPE_Academic degree',
       'NAME_EDUCATION_TYPE_Higher education',
       'NAME_EDUCATION_TYPE_Incomplete higher',
       'NAME_EDUCATION_TYPE_Lower secondary',
       'NAME_EDUCATION_TYPE_Secondary / secondary special',
       'HOUSETYPE_MODE_block of flats', 'HOUSETYPE_MODE_specific housing',
       'HOUSETYPE_MODE_terraced house', 'CODE_GENDER_F', 'CODE_GENDER_M',
       'CODE_GENDER_Other', 'EMERGENCYSTATE_MODE_No',
       'EMERGENCYSTATE_MODE_Yes', 'FLAG_OWN_REALTY_N', 'FLAG_OWN_REALTY_Y',
       'FLAG_OWN_CAR_N', 'FLAG_OWN_CAR_Y', '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')
```

Model Building

Xgboost (with Tuning Hyperparameter and remove highly correlated features)

In [49]:
```
xgbpipeline = Pipeline([
        ('scale', StandardScaler()),
        ('clf', xgb.XGBClassifier())])
```

In [51]:
```
# Create the parameter grid
gbm_param_grid = {
    'clf__learning_rate': np.arange(0.05, 1, 0.05),
    'clf__max_depth': np.arange(3, 10, 1),
    'clf__n_estimators': np.arange(50, 300, 50),
    'clf__clf_colsample_bytree' : [0.6,0.8,1.0]
}
```

In [52]:
```
# 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 [53]:
```
# Fit the estimator
randomized_roc_auc.fit(traindf,Y_train)
```

Out[53]:
```
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': [0.6, 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]),
                                        '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 [54]:
```
# 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=1.0, colsample_bylevel=1,
                               colsample_bynode=1, colsample_bytree=1, gamma=
0,
                               gpu_id=-1, importance_type='gain',
                               interaction_constraints=None,
                               learning_rate=0.15000000000000002,
                               max_delta_step=0, max_depth=4,
                               min_child_weight=1, missing=nan,
                               monotone_constraints=None, n_estimators=100,
                               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 [55]:
```
print(randomized_roc_auc.best_score_)
```

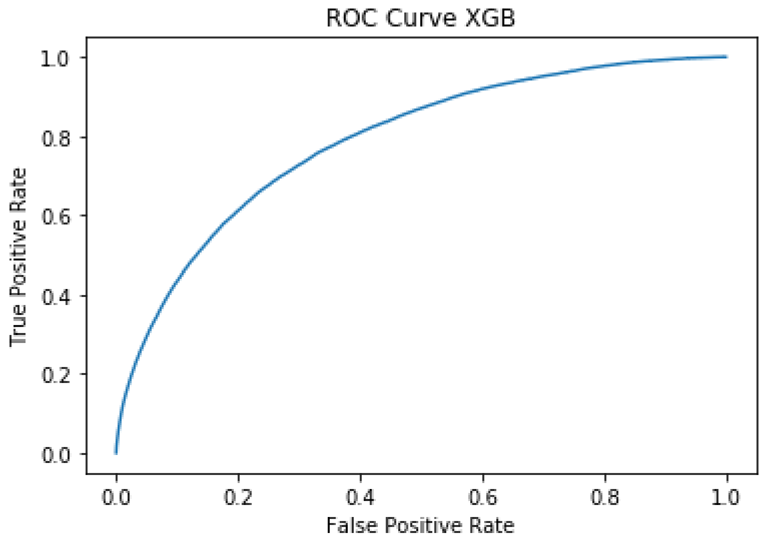
```
0.7708415079877616
```

In [56]:
```
model_xgb_probs = randomized_roc_auc.predict_proba(traindf)
```

In [57]:
```
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: 0.7870927972427697
```

In [58]:
```
plt.plot(fpr, tpr)
plt.title("ROC Curve XGB")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```



In [59]:
```
predictions = [round(value) for value in scores]
accuracy = accuracy_score(Y_train, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 92.02%
```

Predict for Test Data

In [73]:
```
testdf.shape
```

Out[73]:
```
(48744, 104)
```

In [74]:
```
SK_ID_CURR_Col = testdf['SK_ID_CURR']
```

In [75]:
```
testdf.drop('SK_ID_CURR', axis=1, inplace=True)
```

Transform Test Data

In [76]:
```
df_ohe_test = pd.get_dummies(testdf[OHE_List])
```

In [77]:
```
df_le_test = testdf[LE_List].apply(lambda x: le.fit_transform(x))
```

In [78]:
```
testdf.drop(categorical_columns, axis=1, inplace=True)
```

In [79]:
```
testdf = pd.concat([testdf, df_ohe_test, df_le_test], axis=1)
```

In [80]:
```
testdf.drop(to_drop_corr, axis=1, inplace=True)
```

In [81]:
```
testdf.shape
```

Out[81]:
```
(48744, 93)
```

In [82]:
```
test_cols = testdf.columns.tolist()
train_cols = traindf.columns.tolist()
```

Check if Train and Test have same set of Variables

In [83]:
```
list(set(train_cols) - set(test_cols))
```

Out[83]:
```
[]
```

Fit XGB to Test Data

In [84]:
```
score_test = randomized_roc_auc.predict_proba(testdf)[:,1]
```

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

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

In [86]:
```
SK_ID_CURR_Col.shape
```

Out[86]:
```
(48744,)
```

In [87]:
```
submit_df1 = pd.DataFrame({'SK_ID_CURR':SK_ID_CURR_Col, 'TARGET': score_test})
```

In [88]:
```
submit_df1.shape
```

Out[88]:
```
(48744, 2)
```

In [89]:
```
submit_df1.to_csv("submit_df_3004.csv",index=False)
```

The score is 0.74392