Steps:

1. Remove Variables have more than 60% of missing values.
2. Variables have <= 60% are imputed using missRanger in R.
3. Dimenstionality Reduction (VarImp, Low Variance)
4. Categorical variables have more than 5 variables will be transformed into Numerical using LabelEncoder()
5. Categorical variables have less than 5 variables will be transformed into Numerical using OneHotEncoding()
6. Model 1: XGBoost with Tuning Hyperparamter, Random Search
7. Model 2: XGBoost with PreProcessing (T-SNE, Correlation), Random Search, Tuning Hyperparamter

Import Library

In [1]:
```
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]:
```
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]:
```
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

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

In [4]:
```
traindf = pd.read_csv("D:/Github/Risk Modelling/Home_Credit/imputed_application_train.csv", index_col=0)
```

In [5]:
```
testdf =  pd.read_csv("D:/Github/Risk Modelling/Home_Credit/imputed_application_Test.csv",  index_col=0)
```

In [6]:
```
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 [7]:
```
traindf.shape
```

Out[7]: (307511, 104)

In [8]:
```
testdf.shape
```

Out[8]: (48744, 104)

Check If Train and Test still have any Missing Values.

In [9]:
```
traindf.isnull().sum().sum()
```

Out[9]: 0

In [10]:
```
testdf.isnull().sum().sum()
```

Out[10]: 0

In [11]:
```
Y_train = traindf['TARGET']
traindf.drop('TARGET', axis=1, inplace=True)
```

PreProcessing

Dimensionality Reduction / Feature Selection

In [12]:
```
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
train_numerical = traindf.select_dtypes(include=numerics)
```

Correlation

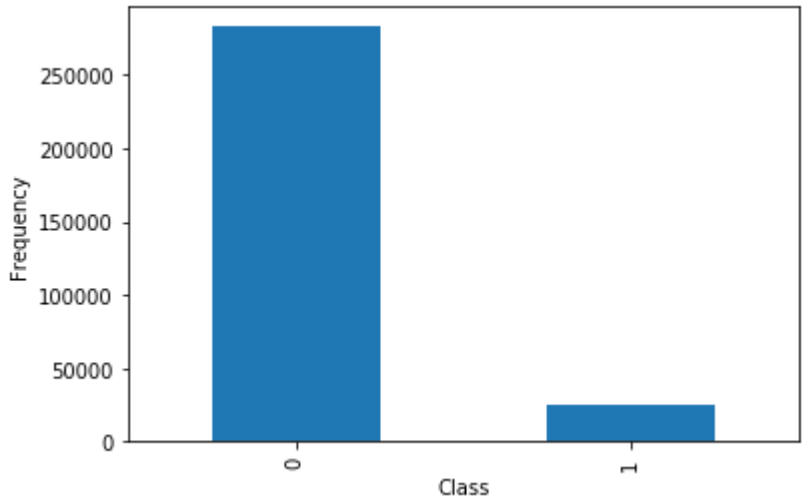
In [13]:
```
# Calculate the correlation matrix and take the absolute value
corr_matrix = train_numerical.corr().abs()
# Create a True/False mask and apply it
mask_highcor = np.triu(np.ones_like(corr_matrix, dtype=bool))
tri_df = corr_matrix.mask(mask_highcor)
# List column names of highly correlated 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[13]:
```
['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']
```

Check If Dataset is Imbalance

In [14]:
```
Y_train.value_counts().plot.bar()
plt.xlabel('Class')
plt.ylabel('Frequency')
Y_train.value_counts()
```

Out[14]:
```
0    282686
1     24825
Name: TARGET, dtype: int64
```



Random Forest for Feature Selection (aka. Variable Importance)

In [15]:
```
# 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[15]:
```
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 [16]:
```
mask_rf = rf.feature_importances_ > 0.1
mask_rf
```

Out[16]:
```
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])
```

Variable Importance could not provide useful info for this Dataset

Low Variance Features

In [17]:
```
train_numerical_normalized = normalize(train_numerical)
```

In [18]:
```
train_numerical_normalized = pd.DataFrame(train_numerical_normalized, columns=
train_numerical.columns)
```

In [19]:
```
train_numerical_normalized.describe()
```

Out[19]:

	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	7.582948e-07	0.250207	0.680557	0.035602	0.60966
std	1.718617e-06	0.154365	0.097870	0.013674	0.08920
min	0.000000e+00	0.008285	0.004808	0.000224	0.00388
25%	0.000000e+00	0.137587	0.662174	0.025049	0.58061
50%	0.000000e+00	0.209654	0.701155	0.032917	0.63304
75%	9.049948e-07	0.321186	0.741016	0.041308	0.66551
max	8.554940e-05	0.999981	0.945914	0.088115	0.98438

8 rows × 88 columns

In [20]:
```
# 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 [21]:
```
mask_lowvar
```

Out[21]:
```
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])
```

PreProcessing

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

In [21]:
```
categorical_mask = (traindf.dtypes == object)
```

In [22]:
```
categorical_columns = traindf.columns[categorical_mask].tolist()
categorical_columns
```

Out[22]:
```
['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 [23]:
```
Categorical_Level  =traindf[categorical_columns].nunique().sort_values(ascending=False)
Categorical_Level
```

Out[23]:
```
ORGANIZATION_TYPE             58
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
EMERGENCYSTATE_MODE            2
FLAG_OWN_REALTY                2
FLAG_OWN_CAR                   2
CODE_GENDER                    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 [24]:
```
OHE_List = Categorical_Level[Categorical_Level<=5].index.tolist()
LE_List = Categorical_Level[Categorical_Level>5].index.tolist()
```

In [25]:
```
le = LabelEncoder()
# Apply LabelEncoder to categorical columns
df_le = traindf[LE_List].apply(lambda x: le.fit_transform(x))
```

In [26]:
```
df_ohe = pd.get_dummies(traindf[OHE_List])
```

Train Data after Converting

In [27]:
```
#traindf.drop(to_drop_corr, axis=1, inplace=True)
```

In [28]:
```
traindf.drop(categorical_columns, axis=1, inplace=True)
```

In [29]:
```
traindf = pd.concat([traindf,df_ohe, df_le], axis=1)
```

In [30]:
```
traindf.head()
```

Out[30]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	RE
1	0	202500.0	406597.5	24700.5	351000.0	
2	0	270000.0	1293502.5	35698.5	1129500.0	
3	0	67500.0	135000.0	6750.0	135000.0	
4	0	135000.0	312682.5	29686.5	297000.0	
5	0	121500.0	513000.0	21865.5	513000.0	

5 rows × 114 columns

In [31]:
```
traindf.columns
```

Out[31]:
```
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=114)
```

Model Building

1. Xgboost (with Tuning Hyperparameter)

In [32]:
```
xgbpipeline = Pipeline([
        ('scale', StandardScaler()),
        ('clf', xgb.XGBClassifier())])
```

In [33]:
```
# 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)
}
```

In [34]:
```
# Perform RandomizedSearchCV
randomized_roc_auc = RandomizedSearchCV(estimator=xgbpipeline, param_distributions=gbm_param_grid,
                                        n_iter=10, scoring='roc_auc', cv=5,
                                        random_state=123, n_jobs = -2)
```

In [35]:
```
# Fit the estimator
randomized_roc_auc.fit(traindf,Y_train)
```

Out[35]:
```
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...
                   iid='deprecated', n_iter=10, n_jobs=-2,
                   param_distributions={'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 [36]:
```
# 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,
                               colsample_bylevel=1, colsample_bynode=1,
                               colsample_bytree=1, gamma=0, gpu_id=-1,
                               importance_type='gain',
                               interaction_constraints=None, learning_rate=0.
1,
                               max_delta_step=0, max_depth=5,
                               min_child_weight=1, missing=nan,
                               monotone_constraints=None, n_estimators=200,
                               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 [37]:
```
print(randomized_roc_auc.best_score_)
```

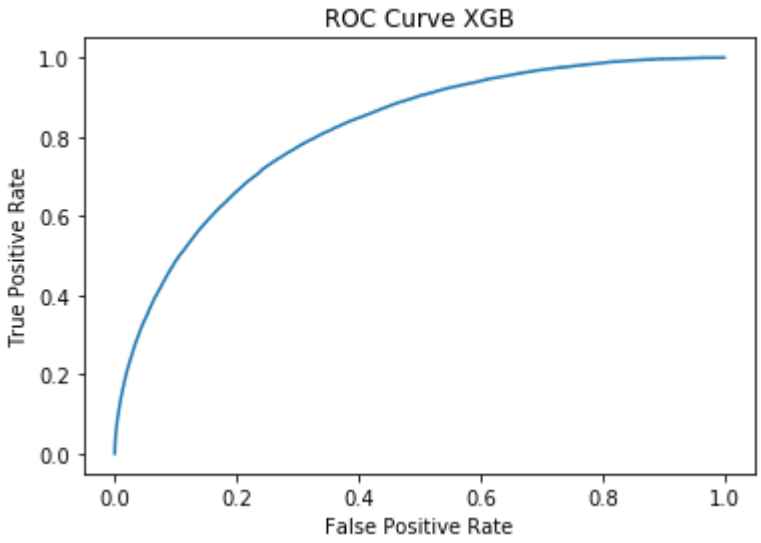
```
0.7782080284066577
```

In [38]:
```
model_xgb_probs = randomized_roc_auc.predict_proba(traindf)
```

In [39]:
```
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: 0.8161555842967732
```

In [40]:
```
plt.plot(fpr, tpr)
plt.title("ROC Curve XGB")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```



In [41]:
```
predictions = [round(value) for value in scores]
accuracy = accuracy_score(Y_train, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 92.18%
```

Predict for Test Data

In [42]:
```
testdf.shape
```

Out[42]:
```
(48744, 104)
```

In [43]:
```
SK_ID_CURR_Col = testdf['SK_ID_CURR']
```

In [44]:
```
testdf.drop('SK_ID_CURR', axis=1, inplace=True)
```

Transform

In [45]:
```
df_ohe_test = pd.get_dummies(testdf[OHE_List])
```

In [46]:
```
df_le_test = testdf[LE_List].apply(lambda x: le.fit_transform(x))
```

In [47]:
```
testdf.drop(categorical_columns, axis=1, inplace=True)
```

In [48]:
```
testdf = pd.concat([testdf, df_ohe_test, df_le_test], axis=1)
```

In [49]:
```
testdf.shape
```

Out[49]: (48744, 114)

In [50]:
```
test_cols = testdf.columns.tolist()
train_cols = traindf.columns.tolist()
```

Check if Train and Test have same set of Variables

In [51]:
```
list(set(train_cols) - set(test_cols))
```

Out[51]: []

Fit XGB to Test Data

In [52]:
```
score_test = randomized_roc_auc.predict_proba(testdf)[:,1]
```

In [53]:
```
score_test.shape
```

Out[53]: (48744,)

In [54]:
```
SK_ID_CURR_Col.shape
```

Out[54]: (48744,)

In [55]:
```
submit_df1 = pd.DataFrame({'SK_ID_CURR':SK_ID_CURR_Col, 'TARGET': score_test})
```

In [56]:
```
submit_df1.shape
```

Out[56]: (48744, 2)

In [57]:
```
submit_df1.to_csv("D:/Github/Risk Modelling/Home_Credit/submit_df1.csv",index=
False)
```

The score is 0.72000

2. XGB with T-SNE and Remove highly correlated

In [118]:
```
traindf2 = traindf.copy()
```

In [119]:
```
testdf2 = testdf.copy()
```

Mapping Train Data to 2-D with T-SNE

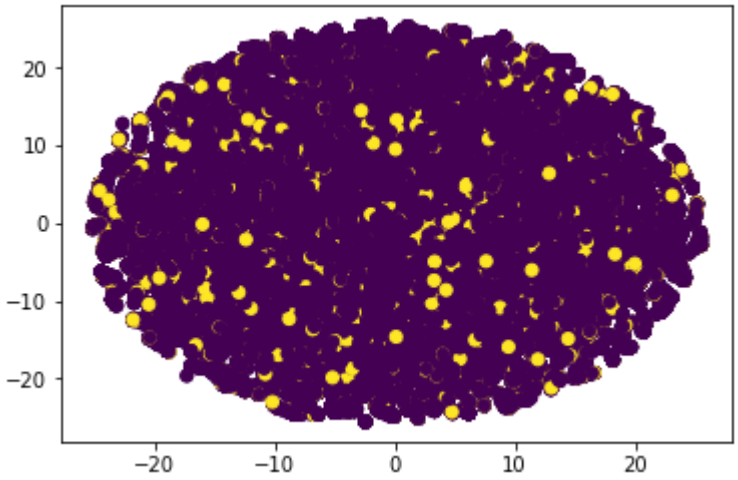
In [80]:
```
model = TSNE(n_components=2,learning_rate=100, random_state=123, n_jobs = -1)
```

In [81]:
```
tsne_features = model.fit_transform(traindf2)
print(tsne_features.shape)
```

```
(307511, 2)
```

In [61]:
```
plt.scatter(tsne_features[:,0], tsne_features[:,1],c=Y_train)
```

Out[61]:
```
<matplotlib.collections.PathCollection at 0x2ab030ed240>
```



In [83]:
```
tsne_features_test = model.fit_transform(testdf2)
print(tsne_features_test.shape)
```

```
(48744, 2)
```

In [84]:
```
tsne_train = pd.DataFrame({'X1':tsne_features[:,0], 'X2': tsne_features[:,1]})
```

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

Out[85]:
```
(307511, 2)
```

In [86]:
```
tsne_test = pd.DataFrame({'X1':tsne_features_test[:,0], 'X2': tsne_features_te
st[:,1]})
```

In [87]:
```
tsne_test.shape
```

Out[87]:
```
(48744, 2)
```

In [120]:
```
traindf2.reset_index(drop=True, inplace=True)
```

In [121]:
```
testdf2.reset_index(drop=True, inplace=True)
```

Check Highly Correlated Features again

In [122]:
```
to_drop_corr
```

Out[122]:
```
['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']
```

Add T-SNE features and remove High Correlation features

In [123]:
```
traindf2.drop(to_drop_corr, axis=1, inplace=True)
```

In [124]:
```
traindf2 = pd.concat([traindf2, tsne_train], axis=1)
```

In [125]:
```
testdf2.drop(to_drop_corr, axis=1, inplace=True)
```

In [126]:
```
testdf2 = pd.concat([testdf2, tsne_test], axis=1)
```

In [127]:
```
traindf2.shape
```

Out[127]:
```
(307511, 94)
```

In [128]:
```
testdf2.shape
```

Out[128]:
```
(48744, 94)
```

Train Model after T-SNE and remove highly correlated

In [131]:
```
xgbpipeline = Pipeline([
        ('scale', StandardScaler()),
        ('clf', xgb.XGBClassifier())])
```

In [132]:
```
# 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)
}
```

In [133]:
```
# Perform RandomizedSearchCV
randomized_roc_auc = RandomizedSearchCV(estimator=xgbpipeline, param_distributions=gbm_param_grid,
                                        n_iter=10, scoring='roc_auc', cv=5,
                                        random_state=123, n_jobs = -2)
```

In [134]:
```
# Fit the estimator
randomized_roc_auc.fit(traindf2,Y_train)
```

Out[134]:
```
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...
                   iid='deprecated', n_iter=10, n_jobs=-2,
                   param_distributions={'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)
```

In [135]:
```
print(randomized_roc_auc.best_score_)
```

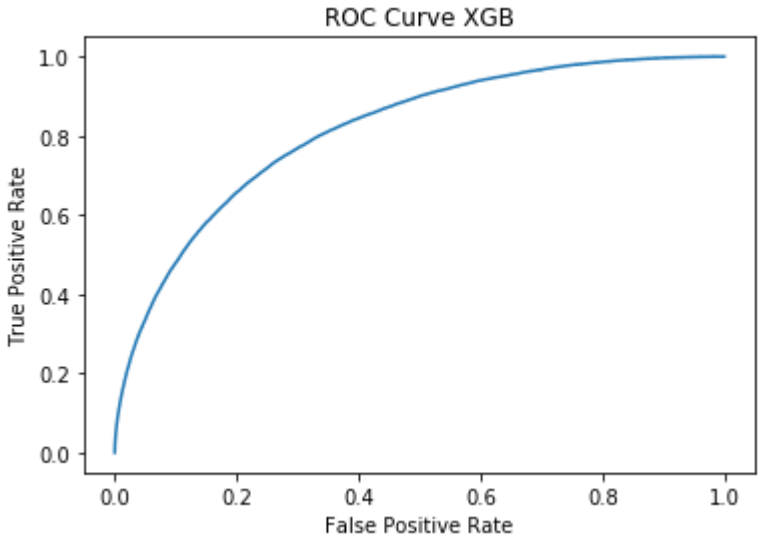
```
0.7759677148098278
```

In [137]:
```
model_xgb_probs2 = randomized_roc_auc.predict_proba(traindf2)
```

In [139]:
```
scores = randomized_roc_auc.predict_proba(traindf2)[:,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: 0.813443665523675
```

In [140]:
```
plt.plot(fpr, tpr)
plt.title("ROC Curve XGB")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```



In [141]:
```
predictions = [round(value) for value in scores]
accuracy = accuracy_score(Y_train, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 92.16%
```

In [142]:
```
score_test = randomized_roc_auc.predict_proba(testdf2)[:,1]
```

In [143]:
```
submit_df2 = pd.DataFrame({'SK_ID_CURR':SK_ID_CURR_Col, 'TARGET': score_test})
```

In [144]:
```
submit_df2.to_csv("D:/Github/Risk Modelling/Home_Credit/submit_df2.csv",index=False)
```

Score for Test Set: 0.71860