Section-IV- Self Case Study-1_Pump it up-Data Mining the Water Table

6. Final model Pipeline

In [203]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import re
import time
import warnings
warnings.filterwarnings("ignore")
import numpy as np
from nltk.corpus import stopwords
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
import sys
from category_encoders import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from category_encoders import TargetEncoder, LeaveOneOutEncoder, WOEEncoder
from sklearn.preprocessing import RobustScaler
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from prettytable import PrettyTable
import category_encoders as ce
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
import joblib
```

In [219]:
```
df_train = pd.read_csv("train.csv")
df_train_labels = pd.read_csv("train_lables.csv")
df = df_train.merge(df_train_labels, how='left', on='id')
```

In [220]:
```
def preprocessing1(df):
    df['funder'].fillna(value='Undefined',inplace=True)
    df['funder'].replace(to_replace = '0', value ='Undefined' , inplace=True) #replacing '0' & missing values with 'Undefined'
    top_30_funders = ['Government Of Tanzania','Undefined','Danida','Hesawa','Rwssp','World Bank','Kkkt','World Vision',
                      'Unicef','Tasaf','District Council','Dhv','Private Individual','Dwsp','Norad','Germany Republi',
                      'Tcrs','Ministry Of Water','Water','Dwe','Netherlands','Hifab','Adb','Lga','Amref','Fini Water',
                      'Oxfam','Wateraid','Rc Church','Isf']
    df.loc[~df["funder"].isin(top_30_funders), "funder"] = "other"

    df['installer'].fillna(value='Undefined',inplace=True)
    df['installer'].replace(to_replace = '0', value ='Undefined' , inplace=True) #replacing '0' category with 'Undefined'


    df['installer'].replace(to_replace = ("Gove","Gover","GOVERM", "GOVERN", "GOVERNME", "Governmen","Government",
                                          "GOVER"), value ="Government", inplace=True)
    df['installer'].replace(to_replace = ("RW","RWE","RWE /Community","RWE Community","RWE/ Community","RWE/Community",
                                          "RWE/DWE","RWE/TCRS","RWEDWE","RWET/WESA"), value ="RWE", inplace=True)
    df['installer'].replace(to_replace = ("Commu", "Communit", "Community", "COMMUNITY BANK",
                                          "Comunity"), value ="Community", inplace=True)
    df['installer'].replace(to_replace = ("Danda","DANIAD","Danid","DANIDA","DANIDA CO","DANIDS","DANNIDA",
                                          "DANID"), value ="DANIDA", inplace=True)
    df['installer'].replace(to_replace = ("Cebtral Government","Cental Government","Centr","Centra Government","Centra govt",
                                          "Central government","Central govt","Cetral government /RC","Tanzania Government",
                                          "TANZANIAN GOVERNMENT"), value ="Central Government", inplace=True)
    df['installer'].replace(to_replace = ("COUN","Counc","Council","Distri","District  Council",
                                          "District Community j","District Counci", "District council",
                                          "District Council"), value ="District Council", inplace=True)
    df['installer'].replace(to_replace = ("Hesawa","HESAW","Hesewa","HESAWA"),value ='HESAWA' , inplace=True)
    df['installer'].replace(to_replace = ("World Division","World Visiin","World vision","World Vission",
                                          "World Vision"),value ='World Vision' , inplace=True)
    df['installer'].replace(to_replace = ("Distric Water Department","District Water Department",
                                          "District water depar","District water department",
                                          "Water Department"),value ='District water department' , inplace=True)
    df['installer'].replace(to_replace = ("FINN WATER","FinW","FinWate","FinWater","Fini water",
                                          "Fini Water" ),value ='Fini Water' , inplace=True)
    df['installer'].replace(to_replace = ("RC","RC .Church","RC C","RC Ch","RC Churc","RC Church",
                                          "RC CHURCH BROTHER","RC church/CEFA","RC church/Central Gover",
                                          "RCchurch/CEFA","RC CHURCH"),value ="RC Church" , inplace=True)
    df['installer'].replace(to_replace = ("Villa","VILLAGER","Villagerd","Villagers","Villages","Villege Council",
                                          "Villi","villigers"),value ="Villagers" , inplace=True)


    top_30_installer = ["DWE","Undefined","Government","DANIDA","Community","HESAWA","RWE","District Council",
                        "Central Government","KKKT","TCRS","World Vision","CES","Fini Water","RC Church","LGA",
                        "WEDECO","TASAF","AMREF","TWESA","WU","Dmdd","ACRA","Villagers","SEMA","DW","OXFAM","Da",
                        "Idara ya maji","UNICEF"]
```

```
    df.loc[~df["installer"].isin(top_30_installer), "installer"] = "other"

    df['longitude'].replace(to_replace = 0 , value =34.07742669202832 , inplace=True)
    df['population'].replace(to_replace = 0, value = 281.087167 , inplace=True)
    df['construction_year'].replace(to_replace = 0, value = 1996, inplace=True)
    df['date_recorded'] = pd.to_datetime(df['date_recorded']) #converting dates to 'datetime' datatype
    df['operational_year'] = df.date_recorded.dt.year - df.construction_year
    df.operational_year.head(5)
    df.loc[df['operational_year']<0, 'operational_year'] = 0

    df.drop(columns=["construction_year", "date_recorded","extraction_type", "extraction_type_class",'payment_type',
                "quality_group", "quantity_group","source_type", "source_class","waterpoint_type_group",'permit',
                "scheme_management", 'id', 'amount_tsh','ward','wpt_name', 'num_private','subvillage','region_code',
                'public_meeting','recorded_by','management_group','scheme_name'], inplace=True)
    return df
```

In [221]:

```
def vectorizing1(df):
    df = preprocessing1(df)
    numeric_target_values = {'functional':1, 'non functional':0, 'functional needs repair':2}
    df['status_group'] = df['status_group'].replace(numeric_target_values)

    #encoding target variables manualy

    numerical_features = ['gps_height','longitude', 'latitude', 'district_code','population', 'operational_year']
    categorical_features = ['funder','installer','basin', 'region', 'lga', 'extraction_type_group','management',
                            'payment', 'water_quality', 'quantity', 'source','waterpoint_type']

    y=df['status_group']
    X = df.drop(columns = ['status_group'])
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=40)

    scaler1 = RobustScaler()
    encoder1 = ce.TargetEncoder()

    scaler = scaler1.fit(X_train[numerical_features])
    scaler_pkl = joblib.dump(scaler, 'scaler.pkl')
    X_train[numerical_features] = scaler.transform(X_train[numerical_features])
    X_test[numerical_features] = scaler.transform(X_test[numerical_features])

    encoder = encoder1.fit(X_train[categorical_features], y_train)
    encoder_pkl = joblib.dump(encoder, 'encoder.pkl')
    X_train[categorical_features] = encoder.transform(X_train[categorical_features])
    X_test[categorical_features] = encoder.transform(X_test[categorical_features])

    smote1 = SMOTE(sampling_strategy = 'auto', n_jobs = -1)
    X_smote_train, y_smote_train = smote1.fit_resample(X_train, y_train)
    y_smote_train = pd.Series(y_smote_train)

    smote2 = SMOTE(sampling_strategy = 'auto', n_jobs = -1)
    X_smote_test, y_smote_test = smote2.fit_resample(X_test, y_test)
    y_smote_test = pd.Series(y_smote_test)

    return X_smote_train, y_smote_train, X_smote_test, y_smote_test,  scaler_pkl, encoder_pkl
```

In [222]:
```
def predict1(df):
    X_smote_train, y_smote_train, X_smote_test, y_smote_test, scaler_pkl, encoder_pkl  = vectorizing1(df)

    from xgboost import XGBClassifier
    clf_xgb =XGBClassifier()
    clf_xgb.fit(X_smote_train, y_smote_train)
    model = joblib.dump(clf_xgb, 'model.pkl')
    # making predictions on test set
    y_pred_train = clf_xgb.predict(X_smote_train)
    y_pred_train_proba = clf_xgb.predict_proba(X_smote_train)
    # making predictions on test set
    y_pred_test = clf_xgb.predict(X_smote_test)
    y_pred_test_proba = clf_xgb.predict_proba(X_smote_test)

    Bal_Acc_train = balanced_accuracy_score(y_smote_train, y_pred_train)
    Bal_Acc_test = balanced_accuracy_score(y_smote_test, y_pred_test)

    F1_train = f1_score(y_smote_train, y_pred_train, average="weighted")
    F1_test = f1_score(y_smote_test, y_pred_test, average="weighted")

    roc_auc_score_train = roc_auc_score(y_smote_train, y_pred_train_proba, multi_class='ovr')
    roc_auc_score_test = roc_auc_score(y_smote_test, y_pred_test_proba, multi_class='ovr')

    return Bal_Acc_train, Bal_Acc_test, F1_train, F1_test, roc_auc_score_train, roc_auc_score_test, model
```

In [223]:
```
predict1(df)
```

```
[23:39:13] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0,
the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly set e
val_metric if you'd like to restore the old behavior.
```

Out[223]:
```
(0.859932790487269,
 0.8149121451022827,
 0.8595785120225174,
 0.8141919402424288,
 0.965501298178791,
 0.9401492345110912,
 ['model.pkl'])
```

In [ ]: