# Assessment of the Conditions of Wells in Tanzania

## Business Understanding

Tanzania, a developing country, has a problem of providing water to its fast growing population of 57 million. The country has already established wells that are expecetd to provide the much needed water, however, some of the wells require repairing for the goal of enough water supply to be met. Enthusiastic Environmentalists (EE), a renown NGO is interested in locating the wells that require repairing and repair them, and enable the country curb the water problem. I am the data scientist tasked with developing a predictive model to know the wells that require repairing. This will enable them cut the cost of survying, to pinpoint the exact wells that will require any repairing as using the model is easier to identify the wells without having to spend on nation-wide physical assessement.

## Data Understanding

In [1]:

```
# importing libraries and functions
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.metrics import accuracy_score,
classification_report,confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE
```

In [2]:

```
# Setting the display of columns to a maximum of 500
pd.set_option('display.max_columns',500)
```

In [3]:

```
# reading the two data files
X = pd.read_csv('Independent_Variables.csv')
y = pd.read_csv('Dependent_Variable.csv')
```

X dataframe is the dataset containing all the factors that relate to the target variable. The y dataframe contains the target variable which is the condition of the well.

In [4]:

```
# viewing the first 5 rows of the independent variables to get a preview
X.head()
```

Out[4]:

| | id | amount_tsh | date_recorded | funder | gps_height | installer | longitude | latitude | wpt_name |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 69572 | 6000.0 | 2011-03-14 | Roman | 1390 | Roman | 34.938093 | -9.856322 | none |

| | id | amount_tsh | date_recorded | funder | gps_height | installer | longitude | latitude | wpt_name |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 8776 | 0.0 | 2013-03-06 | Grumeti | 1399 | GRUMETI | 34.698766 | -2.147466 | Zahanati |
| 2 | 34310 | 25.0 | 2013-02-25 | Lottery Club | 686 | World vision | 37.460664 | -3.821329 | Kwa Mahundi |
| 3 | 67743 | 0.0 | 2013-01-28 | Unicef | 263 | UNICEF | 38.486161 | -11.155298 | Zahanati Ya Nanyumbu |
| 4 | 19728 | 0.0 | 2011-07-13 | Action In A | 0 | Artisan | 31.130847 | -1.825359 | Shuleni |

In [5]:

```
# Viewing the first 5 rows of the dependent variable to get a preview
y.head()
```

Out[5]:

| | id | status_group |
|---|---|---|
| 0 | 69572 | functional |
| 1 | 8776 | functional |
| 2 | 34310 | functional |
| 3 | 67743 | non functional |
| 4 | 19728 | functional |

In [6]:

```
# Merging the two data sets for easier cleaning and manipulation
df = pd.concat([y['status_group'],X],axis = 1)
df.head()
```

Out[6]:

| | status_group | id | amount_tsh | date_recorded | funder | gps_height | installer | longitude | latitud |
|---|---|---|---|---|---|---|---|---|---|
| 0 | functional | 69572 | 6000.0 | 2011-03-14 | Roman | 1390 | Roman | 34.938093 | -9.85632 |
| 1 | functional | 8776 | 0.0 | 2013-03-06 | Grumeti | 1399 | GRUMETI | 34.698766 | -2.14746 |
| 2 | functional | 34310 | 25.0 | 2013-02-25 | Lottery Club | 686 | World vision | 37.460664 | -3.82132 |
| 3 | non functional | 67743 | 0.0 | 2013-01-28 | Unicef | 263 | UNICEF | 38.486161 | -11.15529 |

| | status_group | id | amount_tsh | date_recorded | funder | gps_height | installer | longitude | latitud |
|---|---|---|---|---|---|---|---|---|---|
| **4** | functional | 19728 | 0.0 | 2011-07-13 | Action In A | 0 | Artisan | 31.130847 | -1.82535 |

In [7]:
```
# to get a general understanding of our data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59400 entries, 0 to 59399
Data columns (total 41 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   status_group           59400 non-null  object 
 1   id                     59400 non-null  int64  
 2   amount_tsh             59400 non-null  float64
 3   date_recorded          59400 non-null  object 
 4   funder                 55765 non-null  object 
 5   gps_height             59400 non-null  int64  
 6   installer              55745 non-null  object 
 7   longitude              59400 non-null  float64
 8   latitude               59400 non-null  float64
 9   wpt_name               59400 non-null  object 
 10  num_private            59400 non-null  int64  
 11  basin                  59400 non-null  object 
 12  subvillage             59029 non-null  object 
 13  region                 59400 non-null  object 
 14  region_code            59400 non-null  int64  
 15  district_code          59400 non-null  int64  
 16  lga                    59400 non-null  object 
 17  ward                   59400 non-null  object 
 18  population             59400 non-null  int64  
 19  public_meeting         56066 non-null  object 
 20  recorded_by            59400 non-null  object 
 21  scheme_management      55523 non-null  object 
 22  scheme_name            31234 non-null  object 
 23  permit                 56344 non-null  object 
 24  construction_year      59400 non-null  int64  
 25  extraction_type        59400 non-null  object 
 26  extraction_type_group  59400 non-null  object 
 27  extraction_type_class  59400 non-null  object 
 28  management             59400 non-null  object 
 29  management_group       59400 non-null  object 
 30  payment                59400 non-null  object 
 31  payment_type           59400 non-null  object 
 32  water_quality          59400 non-null  object 
 33  quality_group          59400 non-null  object 
 34  quantity               59400 non-null  object 
 35  quantity_group         59400 non-null  object 
 36  source                 59400 non-null  object 
 37  source_type            59400 non-null  object 
 38  source_class           59400 non-null  object 
 39  waterpoint_type        59400 non-null  object 
 40  waterpoint_type_group  59400 non-null  object 
dtypes: float64(3), int64(7), object(31)
memory usage: 18.6+ MB
```

In [8]:
```
# to get descriptive statistics of our data
df.describe()
```

Out[8]:

| | id | amount_tsh | gps_height | longitude | latitude | num_private | region_cc |
|---|---|---|---|---|---|---|---|
| **count** | 59400.000000 | 59400.000000 | 59400.000000 | 59400.000000 | 5.940000e+04 | 59400.000000 | 59400.000( |
| **mean** | 37115.131768 | 317.650385 | 668.297239 | 34.077427 | -5.706033e+00 | 0.474141 | 15.297( |
| **std** | 21453.128371 | 2997.574558 | 693.116350 | 6.567432 | 2.946019e+00 | 12.236230 | 17.587 |
| **min** | 0.000000 | 0.000000 | -90.000000 | 0.000000 | -1.164944e+01 | 0.000000 | 1.000( |
| **25%** | 18519.750000 | 0.000000 | 0.000000 | 33.090347 | -8.540621e+00 | 0.000000 | 5.000( |
| **50%** | 37061.500000 | 0.000000 | 369.000000 | 34.908743 | -5.021597e+00 | 0.000000 | 12.000( |
| **75%** | 55656.500000 | 20.000000 | 1319.250000 | 37.178387 | -3.326156e+00 | 0.000000 | 17.000( |
| **max** | 74247.000000 | 350000.000000 | 2770.000000 | 40.345193 | -2.000000e-08 | 1776.000000 | 99.000( |

In [9]:

```
# Trying to showcase the null values in the dataset
df.isna().sum()
```

Out[9]:
```
status_group                 0
id                           0
amount_tsh                   0
date_recorded                0
funder                    3635
gps_height                   0
installer                 3655
longitude                    0
latitude                     0
wpt_name                     0
num_private                  0
basin                        0
subvillage                 371
region                       0
region_code                  0
district_code                0
lga                          0
ward                         0
population                   0
public_meeting            3334
recorded_by                  0
scheme_management         3877
scheme_name              28166
permit                    3056
construction_year            0
extraction_type              0
extraction_type_group        0
extraction_type_class        0
management                   0
management_group             0
payment                      0
payment_type                 0
water_quality                0
quality_group                0
quantity                     0
quantity_group               0
source                       0
source_type                  0
source_class                 0
waterpoint_type              0
waterpoint_type_group        0
dtype: int64
```

In [10]:
```
# Investigate the  ternary target variable
df['status_group'].value_counts()
```

Out[10]:
```
functional                 32259
non functional             22824
functional needs repair     4317
Name: status_group, dtype: int64
```

In [11]:
```
# Turn the target variable into binary
df['status_group'] =
df['status_group'].replace({"functional":"No_repair",
                            "non functional":"Repair",
                            "functional needs repair":"Repair"})
df['status_group'].value_counts()
```

Out[11]:
```
No_repair    32259
Repair       27141
Name: status_group, dtype: int64
```

From the data provided we see that 32,259 wells do not need any repair while 27,141 wells need to be repaired. We therefore use the various factors that predict whether a well would require repairing or not.

In [12]:
```
# I chose to drop all the columns that had null values
df = df.dropna(axis=1)
df.isna().sum()
```

Out[12]:
```
status_group             0
id                       0
amount_tsh               0
date_recorded            0
gps_height               0
longitude                0
latitude                 0
wpt_name                 0
num_private              0
basin                    0
region                   0
region_code              0
district_code            0
lga                      0
ward                     0
population               0
recorded_by              0
construction_year        0
extraction_type          0
extraction_type_group    0
extraction_type_class    0
management               0
management_group         0
payment                  0
payment_type             0
water_quality            0
quality_group            0
quantity                 0
quantity_group           0
source                   0
source_type              0
source_class             0
waterpoint_type          0
waterpoint_type_group    0
dtype: int64
```

In [13]:
```
# Understanding the shape of the dataframe
print(f''' The data has {df.shape[0]} rows and {df.shape[1]} columns''')
```

```
 The data has 59400 rows and 34 columns
```

In [14]:
```
# showing the columns in the dataframe
df.columns
```

Out[14]:
```
Index(['status_group', 'id', 'amount_tsh', 'date_recorded', 'gps_height',
       'longitude', 'latitude', 'wpt_name', 'num_private', 'basin', 'region',
       'region_code', 'district_code', 'lga', 'ward', 'population',
       'recorded_by', 'construction_year', 'extraction_type',
       'extraction_type_group', 'extraction_type_class', 'management',
       'management_group', 'payment', 'payment_type', 'water_quality',
       'quality_group', 'quantity', 'quantity_group', 'source', 'source_type',
       'source_class', 'waterpoint_type', 'waterpoint_type_group'],
      dtype='object')
```

In [15]:
```
# Creating the dependent and independent variables
X_ = df.drop(columns=['status_group','id'])
y_ = df[['status_group']]
```

# Data Preparation

After reading the two datasets, getting to understand the content in both and cleaning the data to make it usable for modelling, we now head into preparing the data through transformations for the purposes of modelling.

In [16]:
```
# splitting the dataset into training and testing datasets
X_train,X_test,y_train,y_test = train_test_split(X_,y_,random_state=42)
```

In [17]:
```
# One hot encoding the y variable
ohe = OneHotEncoder(handle_unknown='ignore')
ohe.fit(y_train)
y_train_ = ohe.transform(y_train).toarray()
y_train_enc =
pd.DataFrame(y_train_,columns=ohe.get_feature_names(y_train.columns))
y_train_enc = y_train_enc[['status_group_Repair']]
y_train_enc
```

Out[17]:

| | status_group_Repair |
|---|---|
| 0 | 1.0 |
| 1 | 0.0 |
| 2 | 0.0 |
| 3 | 0.0 |
| 4 | 1.0 |
| ... | ... |
| 44545 | 0.0 |
| 44546 | 0.0 |
| 44547 | 1.0 |
| 44548 | 0.0 |
| 44549 | 1.0 |

44550 rows × 1 columns

In [18]:

```
#Onehotencoding the categorical variables in the independent variables dataset
X_train_categorical = X_train.drop(columns=
['date_recorded','wpt_name']).select_dtypes(include=['object'])
ohe.fit(X_train_categorical)
X_train_ = ohe.transform(X_train_categorical).toarray()
X_train_enc =
pd.DataFrame(X_train_,columns=ohe.get_feature_names(X_train_categorical.co
X_train_enc
```

Out[18]:

| | basin_Internal | basin_Lake Nyasa | basin_Lake Rukwa | basin_Lake Tanganyika | basin_Lake Victoria | basin_Pangani | basin_Rufiji | basin_ / |
|---|---|---|---|---|---|---|---|---|
| 0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 1 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | |
| 2 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | |
| 3 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 4 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| ... | ... | ... | ... | ... | ... | ... | ... | |
| 44545 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | |
| 44546 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| 44547 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | |
| 44548 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | |
| 44549 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | |

44550 rows × 2353 columns

In [19]:
```
# Creating a variable with numeric variables
X_train_num = X_train.select_dtypes(exclude=['object'])
X_train_num = X_train_num.reset_index()
```

In [20]:
```
# Scaling the numeric variables
scaler = MinMaxScaler()
scaler.fit(X_train_num.drop(columns='index'))
X_train_scaled = scaler.transform(X_train_num.drop(columns='index'))
X_train_scaled_df =
pd.DataFrame(X_train_scaled,columns=X_train_num.drop(columns='index').colu
X_train_scaled_df
```

Out[20]:

| | amount_tsh | gps_height | longitude | latitude | num_private | region_code | district_code | populatior |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.000057 | 0.138722 | 0.944941 | 0.477474 | 0.0 | 0.051020 | 0.0125 | 0.00262: |
| **1** | 0.000000 | 0.022238 | 0.000000 | 1.000000 | 0.0 | 0.163265 | 0.0125 | 0.00000( |
| **2** | 0.000000 | 0.022238 | 0.825683 | 0.758435 | 0.0 | 0.183673 | 0.0500 | 0.00000( |
| **3** | 0.000000 | 0.566537 | 0.862136 | 0.584350 | 0.0 | 0.122449 | 0.0500 | 0.00075‹ |
| **4** | 0.000000 | 0.206848 | 0.859110 | 0.080872 | 0.0 | 0.091837 | 0.0375 | 0.00003: |
| **...** | ... | ... | ... | ... | ... | ... | ... | .. |
| **44545** | 0.002857 | 0.137663 | 0.901399 | 0.246765 | 0.0 | 0.040816 | 0.0500 | 0.00836 |
| **44546** | 0.002857 | 0.637487 | 0.855902 | 0.161367 | 0.0 | 0.102041 | 0.0625 | 0.00114{ |
| **44547** | 0.000000 | 0.017649 | 0.966024 | 0.534671 | 0.0 | 0.030612 | 0.0625 | 0.03278 |
| **44548** | 0.000000 | 0.022238 | 0.850574 | 0.733278 | 0.0 | 0.163265 | 0.0750 | 0.00000( |
| **44549** | 0.000000 | 0.477586 | 0.932612 | 0.724325 | 0.0 | 0.020408 | 0.0125 | 0.00003: |

44550 rows × 9 columns

In [21]:
```
# Creating one dataset with all independent variables ready for
modelling
X_values_train = pd.concat([X_train_scaled_df,X_train_enc],axis=1)
X_values_train
```

Out[21]:

| | amount_tsh | gps_height | longitude | latitude | num_private | region_code | district_code | populatior |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.000057 | 0.138722 | 0.944941 | 0.477474 | 0.0 | 0.051020 | 0.0125 | 0.00262: |
| **1** | 0.000000 | 0.022238 | 0.000000 | 1.000000 | 0.0 | 0.163265 | 0.0125 | 0.00000( |
| **2** | 0.000000 | 0.022238 | 0.825683 | 0.758435 | 0.0 | 0.183673 | 0.0500 | 0.00000( |
| **3** | 0.000000 | 0.566537 | 0.862136 | 0.584350 | 0.0 | 0.122449 | 0.0500 | 0.00075‹ |
| **4** | 0.000000 | 0.206848 | 0.859110 | 0.080872 | 0.0 | 0.091837 | 0.0375 | 0.00003: |

| | amount_tsh | gps_height | longitude | latitude | num_private | region_code | district_code | populatior |
|---|---|---|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... | ... | ... | .. |
| **44545** | 0.002857 | 0.137663 | 0.901399 | 0.246765 | 0.0 | 0.040816 | 0.0500 | 0.00836 |
| **44546** | 0.002857 | 0.637487 | 0.855902 | 0.161367 | 0.0 | 0.102041 | 0.0625 | 0.00114 |
| **44547** | 0.000000 | 0.017649 | 0.966024 | 0.534671 | 0.0 | 0.030612 | 0.0625 | 0.03278 |
| **44548** | 0.000000 | 0.022238 | 0.850574 | 0.733278 | 0.0 | 0.163265 | 0.0750 | 0.00000 |
| **44549** | 0.000000 | 0.477586 | 0.932612 | 0.724325 | 0.0 | 0.020408 | 0.0125 | 0.00003 |

44550 rows × 2362 columns

# Modeling and Evaluation

After processing the data and getting it ready for modelling, we head into creating predictive models from which the model with the best performance will be selected and used by the NGO in making predictions.

In [22]:
```
# Creating the baseline model which is a logistic regression
baseline_model = LogisticRegression(max_iter=1000)
baseline_model.fit(X=X_values_train,y=np.ravel(y_train_enc))
```

Out[22]:
```
LogisticRegression(max_iter=1000)
```

In [23]:
```
# Predicting the y values with train x values
y_pred = baseline_model.predict(X_values_train)
```

In [24]:
```
# Seeing the performance of the model with the train dataset
train_report = classification_report(y_true=y_train_enc,y_pred=y_pred)
print(train_report)
print(f'''{accuracy_score(y_true=y_train_enc,y_pred=y_pred)}''')
```

```
              precision    recall  f1-score   support

         0.0       0.79      0.87      0.83     24161
         1.0       0.82      0.73      0.77     20389

    accuracy                           0.80     44550
   macro avg       0.81      0.80      0.80     44550
weighted avg       0.81      0.80      0.80     44550

0.804287317620651
```

In [25]:
```
# Preprocessing the test dataset
ohe.fit(y_train)
y_test_ = ohe.transform(y_test).toarray()
y_test_enc = 
pd.DataFrame(y_test_,columns=ohe.get_feature_names(y_test.columns))
```

```
X_test_categorical = X_test.drop(columns=
['date_recorded','wpt_name']).select_dtypes(include=['object'])
ohe.fit(X_train_categorical)
X_test_ = ohe.transform(X_test_categorical).toarray()
X_test_enc =
pd.DataFrame(X_test_,columns=ohe.get_feature_names(X_test_categorical.colu
X_test_num = X_test.select_dtypes(exclude=['object'])
X_test_num = X_test_num.reset_index()
X_test_scaled = scaler.transform(X_test_num.drop(columns='index'))
X_test_scaled_df =
pd.DataFrame(X_test_scaled,columns=X_test_num.drop(columns='index').column
X_test_scaled_df
X_values_test = pd.concat([X_test_scaled_df,X_test_enc],axis=1)
```

In [26]:

```
# predicting the y values using the test x values
y_pred2 = baseline_model.predict(X_values_test)
```

In [27]:

```
# creating the y variable
y_test_enc = y_test_enc[['status_group_Repair']]
```

In [28]:

```
# Seeing the performance of the model using the test dataset
test_report = classification_report(y_true=y_test_enc,y_pred=y_pred2)
print(test_report)
```

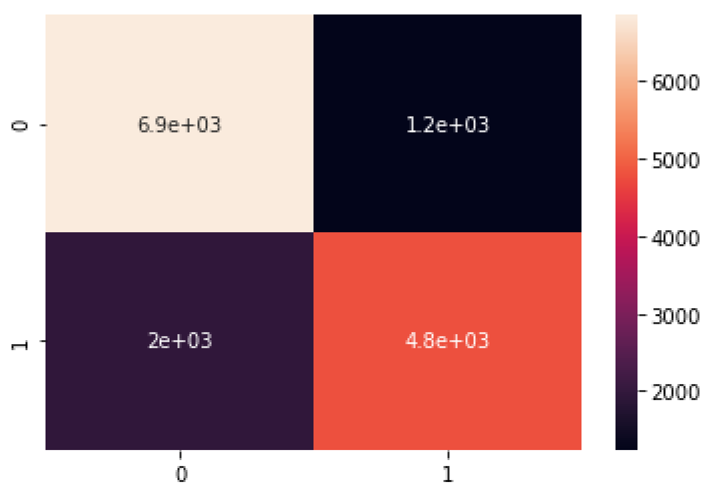
```
              precision    recall  f1-score   support

         0.0       0.78      0.85      0.81      8098
         1.0       0.79      0.71      0.75      6752

    accuracy                           0.78     14850
   macro avg       0.79      0.78      0.78     14850
weighted avg       0.79      0.78      0.78     14850
```

In [29]:

```
# Visualizing the confusion matrix
sns.heatmap(confusion_matrix(y_true = y_test_enc,y_pred=y_pred2),annot =
True);
```




In [30]:

```
# Using the AUC score to test the performance of the model
```

```
sklearn.metrics.roc_auc_score(y_true=y_test_enc,y_score=y_pred2)
```

Out[30]: 0.7784189919048529

In [31]:
```
# Creating a decision tree model
improved_model = sklearn.tree.DecisionTreeClassifier()
improved_model.fit(X=X_values_train,y=y_train_enc)
```

Out[31]: DecisionTreeClassifier()

In [32]:
```
# predicting y values using the decision tree model
y_pred3 = improved_model.predict(X_values_test)
```

In [33]:
```
# testing the performance of the model using the AUC score
sklearn.metrics.roc_auc_score(y_true=y_test_enc,y_score=y_pred3)
```

Out[33]: 0.7842471453076589

In [34]:
```
# classification report
test_report_improved =
classification_report(y_true=y_test_enc,y_pred=y_pred3)
print(test_report_improved)
```

```
              precision    recall  f1-score   support

         0.0       0.80      0.81      0.81      8098
         1.0       0.77      0.76      0.76      6752

    accuracy                           0.79     14850
   macro avg       0.79      0.78      0.78     14850
weighted avg       0.79      0.79      0.79     14850
```

# Conclusion

For this model the selected metric used to determine the most appropriate model was AUC score since it is best for binary target variables. The decision tree classifiaction model (improved_model) had an AUC score of 78%, indicating that it is the best model to use. The NGO should then use the improved model to identify wells that need to be repaired.