Exploratory Data Analysis

Importing the [Tanzanian Water Well Data (https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/page/23/)](https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/page/23/)

In [1]:

```
import pandas as pd

#Training_set_labels = The dependent variable ['status_group'] for each of the rows in Training_set_values
original_y = pd.read_csv('Training_set_labels.csv')

#Training_set_values = The independent variables for the training set
original_X = pd.read_csv('Training_set_values.csv')
```

executed in 711ms, finished 18:29:43 2021-03-28

Exploring the Dependent Variables

In [2]:

```
original_y.nunique()
```

executed in 26ms, finished 18:29:45 2021-03-28

Out[2]:

```
id              59400
status_group        3
dtype: int64
```

In [3]:

```
original_y['status_group'].value_counts()
```

executed in 11ms, finished 18:29:45 2021-03-28

Out[3]:

```
functional                 32259
non functional             22824
functional needs repair     4317
Name: status_group, dtype: int64
```

Exploring the Independent Variables

In [4]:
```
original_X.info()
```
executed in 71ms, finished 18:29:47 2021-03-28

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59400 entries, 0 to 59399
Data columns (total 40 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     59400 non-null  int64  
 1   amount_tsh             59400 non-null  float64
 2   date_recorded          59400 non-null  object 
 3   funder                 55765 non-null  object 
 4   gps_height             59400 non-null  int64  
 5   installer              55745 non-null  object 
 6   longitude              59400 non-null  float64
 7   latitude               59400 non-null  float64
 8   wpt_name               59400 non-null  object 
 9   num_private            59400 non-null  int64  
 10  basin                  59400 non-null  object 
 11  subvillage             59029 non-null  object 
 12  region                 59400 non-null  object 
 13  region_code            59400 non-null  int64  
 14  district_code          59400 non-null  int64  
 15  lga                    59400 non-null  object 
 16  ward                   59400 non-null  object 
 17  population             59400 non-null  int64  
 18  public_meeting         56066 non-null  object 
 19  recorded_by            59400 non-null  object 
 20  scheme_management      55523 non-null  object 
 21  scheme_name            31234 non-null  object 
 22  permit                 56344 non-null  object 
 23  construction_year      59400 non-null  int64  
 24  extraction_type        59400 non-null  object 
 25  extraction_type_group  59400 non-null  object 
 26  extraction_type_class  59400 non-null  object 
 27  management             59400 non-null  object 
 28  management_group       59400 non-null  object 
 29  payment                59400 non-null  object 
 30  payment_type           59400 non-null  object 
 31  water_quality          59400 non-null  object 
 32  quality_group          59400 non-null  object 
 33  quantity               59400 non-null  object 
 34  quantity_group         59400 non-null  object 
 35  source                 59400 non-null  object 
 36  source_type            59400 non-null  object 
 37  source_class           59400 non-null  object 
 38  waterpoint_type        59400 non-null  object 
 39  waterpoint_type_group  59400 non-null  object 
dtypes: float64(3), int64(7), object(30)
memory usage: 18.1+ MB
```

Exploring Smaller DataFrames by Category

In [5]:
```
df = original_X.merge(original_y, on='id', how='outer')
len(df.columns)
# Merging independent and dependent dataframes should yield 41 columns.
```
executed in 190ms, finished 18:29:52 2021-03-28

Out[5]:
```
41
```

In [6]:

```
import matplotlib.pyplot as plt
import numpy as np

def plot_target_bar_graph(category, target, plotname):

    # Function created to plot functionality dependent on any variable in the DataFrame

    # category = df_independent variable
    # target = df_dependent variable (status)
    # plotname = df_independent plot title

    set_labels = category.unique()
    bar_labels = target.unique()
    bar = {}
    for i in range(len(bar_labels)):
        bar[i] = {}
        for j in set_labels:
            bar[i][j] = 0

    count = 0
    for k in category:
        if target[count] == bar_labels[0]:
            bar[0][k] = bar[0][k] + 1
        elif target[count] == bar_labels[1]:
            bar[1][k] = bar[1][k] + 1
        else:
            bar[2][k] = bar[2][k] + 1
        count += 1

    x = np.arange(len(set_labels))  # the label locations
    width = 0.25  # the width of the bars

    fig, ax = plt.subplots()
    ax.bar(x - width/2, list(bar[0].values()), width, label=bar_labels[0], color='royalblue') #bar1
    ax.bar(x,           list(bar[1].values()), width, label=bar_labels[1], color='black') #bar2
    ax.bar(x + width/2, list(bar[2].values()), width, label=bar_labels[2], color='lightblue') #bar3

    # Add some text for labels, title and custom x-axis tick labels, etc.
    ax.set_ylabel('Number of Waterpoints')
    ax.set_title(f'Amount of Waterpoints by {plotname}')
    ax.set_xticks(x)
    ax.set_xticklabels(set_labels)
    ax.legend()

    fig.tight_layout()

    plt.show()
```

executed in 232ms, finished 18:30:11 2021-03-28

Time Related: date_recorded, construction_year

In [7]:

```
df_time = df.iloc[:, [2, 23, 40]].copy()
print(df_time.info())
df_time.nunique()
```

executed in 59ms, finished 18:30:13 2021-03-28

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59400 entries, 0 to 59399
Data columns (total 3 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   date_recorded      59400 non-null  object
 1   construction_year  59400 non-null  int64 
 2   status_group       59400 non-null  object
dtypes: int64(1), object(2)
memory usage: 1.8+ MB
None
```

Out[7]:

```
date_recorded        356
construction_year     55
status_group           3
dtype: int64
```

In [8]:

```
df_time['date_recorded'] = pd.to_datetime(df_time['date_recorded'], format='%Y-%m-%d')
df_time['year_recorded'] = pd.DatetimeIndex(df_time['date_recorded']).year
df_time.year_recorded.value_counts()
```

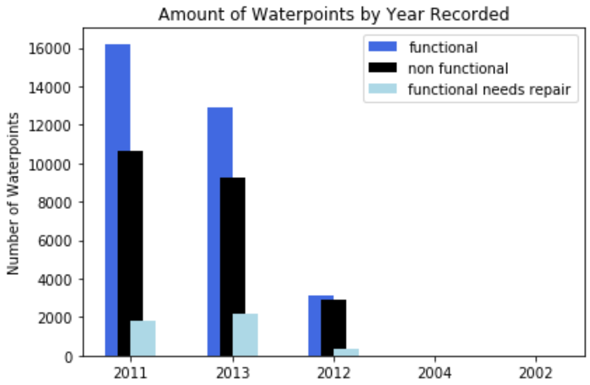
executed in 20ms, finished 18:30:14 2021-03-28

Out[8]:

```
2011    28674
2013    24271
2012     6424
2004       30
2002        1
Name: year_recorded, dtype: int64
```

```
In [9]: plot_target_bar_graph(df_time.year_recorded, df_time.status_group, 'Year Recorded')
```

executed in 991ms, finished 18:30:21 2021-03-28



```
In [10]: df_time.construction_year.value_counts()
```

executed in 6ms, finished 18:30:23 2021-03-28

```
Out[10]: 0       20709
2010     2645
2008     2613
2009     2533
2000     2091
2007     1587
2006     1471
2003     1286
2011     1256
2004     1123
2012     1084
2002     1075
1978     1037
1995     1014
2005     1011
1999      979
1998      966
1990      954
1985      945
1980      811
1996      811
1984      779
1982      744
1994      738
1972      708
1974      676
1997      644
1992      640
1993      608
2001      540
1988      521
1983      488
1975      437
1986      434
1976      414
1970      411
1991      324
1989      316
1987      302
1981      238
1977      202
1979      192
1973      184
2013      176
1971      145
1960      102
1967       88
1963       85
1968       77
1969       59
1964       40
1962       30
1961       21
1965       19
1966       17
Name: construction_year, dtype: int64
```

In [ ]:

```
counter = 0
for year in df_time['construction_year']:
    if year >= 2010:
        df_time.loc[count, 'construction_decade'] = 'After_2010'
    if year >= 2000:
        df_time.loc[count, 'construction_decade'] = '2000-2010'
    if year >= 1990:
        df_time.loc[count, 'construction_decade'] = '1990-2000'
    if year >= 1980:
        df_time.loc[count, 'construction_decade'] = '1980-1990'
    if year >= 1970:
        df_time.loc[count, 'construction_decade'] = '1970-1980'
    if year >= 1960:
        df_time.loc[count, 'construction_decade'] = '1960-1970'
    else:
        df_time.loc[count, 'construction_decade'] = 'Before_1950'
    counter += 1
```

executed in 26.4s, finished 17:48:07 2021-03-28

In [ ]:

```
df_time.construction_decade.value_counts()
```

executed in 24.3s, finished 17:48:07 2021-03-28

In [ ]:

```
plot_target_bar_graph(df_time.year_recorded, df_time.status_group, 'Year Recorded')
```

executed in 10ms, finished 18:22:42 2021-03-28

The time related information provided will not be further explored in the dataset used for modelling. There is not distinct relationship found in the plots that compare the functionality of the waterpoints versus the time the information was recorded or when the waterpoint was created.

Geographical Descriptions: gps_height, longitude, latitude

In [ ]:

```
df_geographical = df.iloc[:, [4, 6, 7, 40]].copy()
df_geographical.head()
```

executed in 20ms, finished 16:46:07 2021-03-28

In [ ]:

```
# Intial plot showed an outlier where long/lat = 0.
df.loc[df['longitude']==0].head()
```

executed in 92ms, finished 16:57:35 2021-03-28

In [ ]:

```
# Dropping outlier for a better plot.
df_geographical = df_geographical[df_geographical.longitude != 0]
```

executed in 11ms, finished 16:57:48 2021-03-28

In [ ]:

```
import seaborn as sns

plt.figure(figsize=(10,10))
sns.scatterplot(data=df_geographical,x='longitude', y='latitude', hue='status_group', palette=['green', 'red', 'yellow'])
plt.title('Status of Waterpoints by Location in Tanzania')
plt.show()
```

executed in 6.43s, finished 16:57:57 2021-03-28

In [ ]:

```
import seaborn as sns

plt.figure(figsize=(10,10))
sns.scatterplot(data=df_geographical,x='longitude', y='latitude', hue='gps_height')
plt.title('Waterpoints by GPS Height in Tanzania')
plt.show()
```

executed in 6.28s, finished 16:58:03 2021-03-28

The geographical information provided will not be further explored in the dataset used for modelling. The status of the waterpoints seem to be evenly distributed throughout Tanzania. Other location and population data may show more correlation.

Location Information: subvillage, region, region_code, district_code, lga, ward

In [ ]:

```
df_location = df.iloc[:, [11, 12, 13, 14, 15, 16, 40]].copy()
df_location.head()
```

executed in 41ms, finished 16:58:03 2021-03-28

In [ ]:

```
df_location.nunique()
```

executed in 70ms, finished 16:58:04 2021-03-28

In [ ]:

Community/Interaction Information: population, public_meeting

In [ ]:

```
df_community = df.iloc[:, [17, 18, 40]].copy()
df_community.head()
```

executed in 20ms, finished 03:44:37 2021-03-28

In [ ]:

```
df_community['public_meeting'].describe()
```

executed in 18ms, finished 03:48:14 2021-03-28

Management (Creation -> Current):

In [ ]:

```
df_management = df.iloc[:, [3, 5, 20, 21, 27, 28, 40]].copy()
df_management.head()
```

executed in 30ms, finished 20:42:28 2021-03-27

Extraction Type:

In [ ]:

```
df_extraction = df.iloc[:, [24, 25, 26, 40]].copy()
df_extraction.head()
```

executed in 21ms, finished 20:46:03 2021-03-27

Source Type:

In [ ]:

```
df_source = df.iloc[:, [35, 36, 37, 40]].copy()
df_source.head()
```

executed in 16ms, finished 20:46:55 2021-03-27

Quality of Water:

In [ ]:

```
df_quality = df.iloc[:, [31, 32, 40]].copy()
df_quality.head()
```

executed in 20ms, finished 20:47:41 2021-03-27

Quantity of Water:

In [ ]:

```
df_quantity = df.iloc[:, [1, 33, 34, 40]].copy()
df_quantity.head()
```

executed in 22ms, finished 20:48:20 2021-03-27

Waterpoint Type:

In [ ]:

```
df_waterpoint = df.iloc[:, [38, 39, 40]].copy()
df_waterpoint.head()
```

executed in 19ms, finished 20:49:22 2021-03-27

Misc. Information:

In [ ]:

```
df_misc = df.iloc[:, [0, 8, 9, 10, 22, 29, 30, 40]].copy()
df_misc.head()
```

executed in 36ms, finished 20:51:33 2021-03-27

In [ ]:

In [ ]:

In [ ]:

```
#Transforming data for analysis purpose

count = 0
original_y['status_group_binary'] = ""
original_y['status_group_tertiary'] = ""

for f in original_y['status_group']:
    if f == 'functional':
        original_y.loc[count, 'status_group_binary'] = 1
        original_y.loc[count, 'status_group_tertiary'] = 1
    elif f == 'non functional':
        original_y.loc[count, 'status_group_binary'] = 2
        original_y.loc[count, 'status_group_tertiary'] = 2
    else:
        original_y.loc[count, 'status_group_binary'] = 1
        original_y.loc[count, 'status_group_tertiary'] = 3
    count += 1
```

executed in 1m 41.0s, finished 00:07:14 2021-03-26

In [ ]:

```
from pandas.api.types import is_string_dtype
from pandas.api.types import is_numeric_dtype

for column in original_X:
    if is_numeric_dtype(column):
        column = column.fillna(0)
    else:
        original_X[column] = original_X[column].fillna('unknown')
```

executed in 108ms, finished 00:07:14 2021-03-26

In [ ]:

```
import matplotlib.pyplot as plt

tracker = []

for x in original_X:
    s2 = original_X[x]
    if len(s2.value_counts()) < 10:
        tracker.append(x)
        prob = s2.value_counts()
        prob.plot(kind='bar')
        plt.xticks(rotation=90)
        plt.title(x)
        plt.show()
    else:
        pass
```

executed in 2.23s, finished 00:07:16 2021-03-26

In [ ]:

```
df = original_X[tracker]
```

executed in 13ms, finished 00:07:16 2021-03-26

In [ ]:

```
df['functions'] = original_y['status_group_binary']
```

executed in 23ms, finished 00:07:16 2021-03-26

In [ ]:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

def preprocess(X, y):
    '''Takes in features and target and implements all preprocessing steps for categorical and continuous features returning
    train and test DataFrames with targets'''

    # Train-test split (75-25), set seed to 10
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=10)

    # OneHotEncode Categorical variables
    ohe = OneHotEncoder(handle_unknown='ignore')

    X_train_ohe = ohe.fit_transform(X_train)
    X_test_ohe = ohe.transform(X_test)

    columns = ohe.get_feature_names(input_features=X_train.columns)
    cat_train_df = pd.DataFrame(X_train_ohe.todense(), columns=columns)
    cat_test_df = pd.DataFrame(X_test_ohe.todense(), columns=columns)

    # Combine categorical and continuous features into the final dataframe
    X_train_all = pd.concat([cat_train_df], axis=1)
    X_test_all = pd.concat([cat_test_df], axis=1)

    return X_train_all, X_test_all, y_train, y_test
```

executed in 819ms, finished 00:07:17 2021-03-26

In [ ]:

```
y = df['functions']
X = df.drop(['functions'], axis=1)

X_train_all, X_test_all, y_train, y_test = preprocess(X, y)
```

executed in 189ms, finished 00:07:17 2021-03-26

In [ ]:

In [ ]:

In [ ]: