Phase 3: Project Submission

Student name: James Kimani

Student pace: Partime

Instructor name: Noah Kandie

In [ ]:

Business Problem

The goal of this analysis and model determine with water pumps are faulty and hence enable access to clean water accross Tanzania reliably.

Objective

The purpose of this analysis is to use classification modeling tecniques to accurately predict which water pumps are faulty and allow for stakeholders to repair/replace such and ensure consistent supply of clean water accross Tanzania.

Data Understanding

The dataset used in this modelling is derived from https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/page/25/.

The data contain information on water pumps recorded across the country of Tanzania. The Target variable is the status group of a pump given various characteristics of the pump including but not limited to Geographic location, Water quantity, GPS height location and Construction year among other features

Success Criteria

The objective is to obtain a prediction model that can correctly predict the condition of a water pump by 80% accuracy.

1. Importing Relevant Libraries

In [1]:
```
import pandas as pd
pd.set_option('display.max_columns', 40)
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import statsmodels.api as sm
sns.set()

from IPython.display import display
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.ensemble import GradientBoostingClassifier
```

```
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
```

2. Loading our data and Data Description

In [2]:
```
#Loading our data set into a pandas dataframes

df_1 = pd.read_csv('train_data.csv', header = 0)
df_2 = pd.read_csv('test_data.csv', header = 0)
df_3 = pd.read_csv('labels.csv')

# Combining both the train and test dataframes to one dataset, split to be done at the p
df = pd.concat([df_1, df_2])
# merging our labels into the dataset

df = pd.merge(df, df_3, on='id')
```

In [3]:
```
# Displaying the 1st 5 rows of our data
df.head()
```

Out[3]:

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wpt_name
0	69572	6000.0	14/03/2011	Roman	1390	Roman	34.938093	-9.856322	none
1	8776	0.0	06/03/2013	Grumeti	1399	GRUMETI	34.698766	-2.147466	Zahanati
2	34310	25.0	25/02/2013	Lottery Club	686	World vision	37.460664	-3.821329	Kwa Mahundi
3	67743	0.0	28/01/2013	Unicef	263	UNICEF	38.486161	-11.155298	Zahanati Ya Nanyumbu
4	19728	0.0	13/07/2011	Action In A	0	Artisan	31.130847	-1.825359	Shuleni

5 rows × 41 columns

In [4]:
```
#Checking the datatypes of our columns
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59400 entries, 0 to 59399
Data columns (total 41 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
 40  status_group           59400 non-null  object
dtypes: float64(3), int64(7), object(31)
memory usage: 19.0+ MB
```

In [5]:
```
# Descriptive Statistics, including for non-numerical columns in our train data
df.describe(include = "all")
```

Out[5]:

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude
count	59400.000000	59400.000000	59400	55765	59400.000000	55745	59400.000000
unique	NaN	NaN	356	1897	NaN	2145	NaN
top	NaN	NaN	15/03/2011	Government Of Tanzania	NaN	DWE	NaN
freq	NaN	NaN	572	9084	NaN	17402	NaN
mean	37115.131768	317.650385	NaN	NaN	668.297239	NaN	34.077427
std	21453.128371	2997.574558	NaN	NaN	693.116350	NaN	6.567432
min	0.000000	0.000000	NaN	NaN	-90.000000	NaN	0.000000
25%	18519.750000	0.000000	NaN	NaN	0.000000	NaN	33.090347
50%	37061.500000	0.000000	NaN	NaN	369.000000	NaN	34.908743
75%	55656.500000	20.000000	NaN	NaN	1319.250000	NaN	37.178387

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude
max	74247.000000	350000.000000	NaN	NaN	2770.000000	NaN	40.345193

11 rows × 41 columns

In [6]:
```
df.isnull().sum().sort_index()/len(df)
```

Out[6]:
```
amount_tsh               0.000000
basin                    0.000000
construction_year        0.000000
date_recorded            0.000000
district_code            0.000000
extraction_type          0.000000
extraction_type_class    0.000000
extraction_type_group    0.000000
funder                   0.061195
gps_height               0.000000
id                       0.000000
installer                0.061532
latitude                 0.000000
lga                      0.000000
longitude                0.000000
management               0.000000
management_group         0.000000
num_private              0.000000
payment                  0.000000
payment_type             0.000000
permit                   0.051448
population               0.000000
public_meeting           0.056128
quality_group            0.000000
quantity                 0.000000
quantity_group           0.000000
recorded_by              0.000000
region                   0.000000
region_code              0.000000
scheme_management        0.065269
scheme_name              0.474175
source                   0.000000
source_class             0.000000
source_type              0.000000
status_group             0.000000
subvillage               0.006246
ward                     0.000000
water_quality            0.000000
waterpoint_type          0.000000
waterpoint_type_group    0.000000
wpt_name                 0.000000
dtype: float64
```

4. Data Preperation

4.1 Droping uneccessary columns

In [7]:
```
df['recorded_by'].unique()
```

Out[7]:
```
array(['GeoData Consultants Ltd'], dtype=object)
```

In [8]:
```
# Drop the recorded_by column since the entry is consistent all through
df.drop(['recorded_by'], axis=1, inplace=True)
```

In [9]:
```
# Drop columns with a very high distinct count of unique values, to avoid high cadinali
df.drop(['funder', 'installer', 'lga', 'scheme_name', 'subvillage', 'ward', 'wpt_name']
```

4.2. Working on columns with Missing Values

In [10]:
```
### Checking for missing values
df.isnull().sum()
```

Out[10]:
```
id                         0
amount_tsh                 0
date_recorded              0
gps_height                 0
longitude                  0
latitude                   0
num_private                0
basin                      0
region                     0
region_code                0
district_code              0
population                 0
public_meeting          3334
scheme_management       3877
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
status_group               0
dtype: int64
```

Working on public_meeting column

In [11]:
```
df['public_meeting'].value_counts()
```

Out[11]:
```
True     51011
False     5055
Name: public_meeting, dtype: int64
```

In [12]:
```
# We replace all the entries with nan with the mode, which is True
df['public_meeting'] = df['public_meeting'].fillna(False)
```

Working on scheme_management column

In [13]:
```
#Replacing the scheme_management column with the mode
df['scheme_management'].fillna(df['scheme_management'].mode()[0], inplace=True)
```

Working on Permit column

In [14]:
```
# Filling missing values in the permit column with the mode of the column
df['permit'].fillna(df['permit'].mode()[0], inplace=True)
df['permit'].unique()
```

Out[14]:
```
array([False,  True])
```

In [15]:
```
#checking if there are missing values
df.isnull().sum()
```

Out[15]:
```
id                     0
amount_tsh             0
date_recorded          0
gps_height             0
longitude              0
latitude               0
num_private            0
basin                  0
region                 0
region_code            0
district_code          0
population             0
public_meeting         0
scheme_management      0
permit                 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
status_group           0
dtype: int64
```

4.3 Feature Engineering

Working on data_recorded column

In [16]:
```
# Convert to date time data type
df['date_recorded'] = pd.to_datetime(df['date_recorded'], errors = 'coerce')

#Creating a column to of the month and year the record was taken

df['month_recorded'] =  df['date_recorded'].dt.month #extracting the month of recording
df['year_recorded'] =  df['date_recorded'].dt.year # extracting the year of recording
df['date_recorded'] =  df['date_recorded'].dt.day # extracting the day of recording
```

```
# Create a column for the season when the record was made, either dry or wet based on t
# Wet Months = March,April,May,October,November December
# Dry Months = January, February, June, July, August, September
wet_months = [3, 4, 5, 10, 11, 12]
dry_months = [1, 2, 6, 7, 8, 9]

# Create a new column for the season
df['season_recorded'] = df['month_recorded'].apply(lambda x: 'Wet' if x in wet_months e
df['season_recorded'].unique()

# Drop the date_recorded column as it is nolonger as useful
df.drop(['date_recorded'], axis=1, inplace=True)
```

In [17]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59400 entries, 0 to 59399
Data columns (total 35 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   id                     59400 non-null  int64
 1   amount_tsh             59400 non-null  float64
 2   gps_height             59400 non-null  int64
 3   longitude              59400 non-null  float64
 4   latitude               59400 non-null  float64
 5   num_private            59400 non-null  int64
 6   basin                  59400 non-null  object
 7   region                 59400 non-null  object
 8   region_code            59400 non-null  int64
 9   district_code          59400 non-null  int64
 10  population             59400 non-null  int64
 11  public_meeting         59400 non-null  bool
 12  scheme_management      59400 non-null  object
 13  permit                 59400 non-null  bool
 14  construction_year      59400 non-null  int64
 15  extraction_type        59400 non-null  object
 16  extraction_type_group  59400 non-null  object
 17  extraction_type_class  59400 non-null  object
 18  management             59400 non-null  object
 19  management_group       59400 non-null  object
 20  payment                59400 non-null  object
 21  payment_type           59400 non-null  object
 22  water_quality          59400 non-null  object
 23  quality_group          59400 non-null  object
 24  quantity               59400 non-null  object
 25  quantity_group         59400 non-null  object
 26  source                 59400 non-null  object
 27  source_type            59400 non-null  object
 28  source_class           59400 non-null  object
 29  waterpoint_type        59400 non-null  object
 30  waterpoint_type_group  59400 non-null  object
 31  status_group           59400 non-null  object
 32  month_recorded         59400 non-null  int64
 33  year_recorded          59400 non-null  int64
 34  season_recorded        59400 non-null  object
dtypes: bool(2), float64(3), int64(9), object(21)
memory usage: 15.5+ MB
```

Working on the district_code column

In [18]:
```
df['district_code'].unique()
```

Out[18]:
```
array([ 5,  2,  4, 63,  1,  8,  3,  6, 43,  7, 23, 33, 53, 62, 60, 30, 13,
        0, 80, 67], dtype=int64)
```

In [19]:
```
# Checking the number of rows where the district is  0
count_0 = (df['district_code'] == 0).sum()
count_0
```

Out[19]:
```
23
```

In [20]:
```
df['district_code'].value_counts()
```

Out[20]:
```
1     12203
2     11173
3      9998
4      8999
5      4356
6      4074
7      3343
8      1043
30      995
33      874
53      745
43      505
13      391
23      293
63      195
62      109
60       63
0        23
80       12
67        6
Name: district_code, dtype: int64
```

In [21]:
```
#Replacing the rows where the district code with 1 where the entry is 0
df['district_code'] = df['district_code'].replace(0,1)
df['district_code'].unique()
```

Out[21]:
```
array([ 5,  2,  4, 63,  1,  8,  3,  6, 43,  7, 23, 33, 53, 62, 60, 30, 13,
       80, 67], dtype=int64)
```

Status_group column

In [22]:
```
df['status_group'].unique()
```

Out[22]:
```
array(['functional', 'non functional', 'functional needs repair'],
      dtype=object)
```

In [23]:
```
### Replacing the categories functional', 'non functional', 'functional needs repair' w

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

Out[23]:
```
array([0, 1, 2], dtype=int64)
```

Working on the contruction_year column

In [24]:
```
## we impute the longitude where it has been recorded as 0 with the median construction
df['construction_year'] = df.groupby('district_code')['construction_year'].apply(lambda
df['construction_year'] = df.groupby('region')['construction_year'].apply(lambda x: x.r
df['construction_year'].value_counts()
```

Out[24]:
```
1990    7931
1985    5500
1984    3849
1982    3659
1988    3445
2010    2645
2008    2616
2009    2533
2000    2106
2007    1587
2006    1586
2003    1298
2011    1256
2004    1123
2012    1084
2002    1075
1978    1037
1995    1029
2005    1011
1999     981
1998     967
1996     811
1980     811
1994     738
1972     708
1974     676
1997     658
1992     640
1993     608
2001     540
1983     488
1986     477
1975     437
1976     414
1970     411
1989     364
1991     324
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
```

Working on the gps_height column

In [25]:
```
## we impute the gps_height where it has been recorded as 0 with the mean construction y
df['gps_height'] = df.groupby('district_code')['construction_year'].apply(lambda x: x.r
df['gps_height'] = df.groupby('region')['construction_year'].apply(lambda x: x.replace(
df['gps_height'].value_counts()
```

Out[25]:
```
1990    7931
1985    5500
1984    3849
1982    3659
1988    3445
2010    2645
2008    2616
2009    2533
2000    2106
2007    1587
2006    1586
2003    1298
2011    1256
2004    1123
2012    1084
2002    1075
1978    1037
1995    1029
2005    1011
1999     981
1998     967
1996     811
1980     811
1994     738
1972     708
1974     676
1997     658
1992     640
1993     608
2001     540
1983     488
1986     477
1975     437
1976     414
1970     411
1989     364
1991     324
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: gps_height, dtype: int64
```

Working on the longitude column

In [26]:
```
## we impute the longitude where it has been recorded as 0 with the median construction
df['longitude'] = df.groupby('district_code')['construction_year'].apply(lambda x: x.re
df['longitude'] = df.groupby('region')['construction_year'].apply(lambda x: x.replace(0
df['longitude'].value_counts()
```

Out[26]:
```
1990    7931
1985    5500
1984    3849
1982    3659
1988    3445
2010    2645
2008    2616
2009    2533
2000    2106
2007    1587
2006    1586
2003    1298
2011    1256
2004    1123
2012    1084
2002    1075
1978    1037
1995    1029
2005    1011
1999     981
1998     967
1996     811
1980     811
1994     738
1972     708
1974     676
1997     658
1992     640
1993     608
2001     540
1983     488
1986     477
1975     437
1976     414
1970     411
1989     364
1991     324
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: longitude, dtype: int64
```

In [27]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59400 entries, 0 to 59399
Data columns (total 35 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     59400 non-null  int64  
 1   amount_tsh             59400 non-null  float64
 2   gps_height             59400 non-null  int64  
 3   longitude              59400 non-null  int64  
 4   latitude               59400 non-null  float64
 5   num_private            59400 non-null  int64  
 6   basin                  59400 non-null  object 
 7   region                 59400 non-null  object 
 8   region_code            59400 non-null  int64  
 9   district_code          59400 non-null  int64  
 10  population             59400 non-null  int64  
 11  public_meeting         59400 non-null  bool   
 12  scheme_management      59400 non-null  object 
 13  permit                 59400 non-null  bool   
 14  construction_year      59400 non-null  int64  
 15  extraction_type        59400 non-null  object 
 16  extraction_type_group  59400 non-null  object 
 17  extraction_type_class  59400 non-null  object 
 18  management             59400 non-null  object 
 19  management_group       59400 non-null  object 
 20  payment                59400 non-null  object 
 21  payment_type           59400 non-null  object 
 22  water_quality          59400 non-null  object 
 23  quality_group          59400 non-null  object 
 24  quantity               59400 non-null  object 
 25  quantity_group         59400 non-null  object 
 26  source                 59400 non-null  object 
 27  source_type            59400 non-null  object 
 28  source_class           59400 non-null  object 
 29  waterpoint_type        59400 non-null  object 
 30  waterpoint_type_group  59400 non-null  object 
 31  status_group           59400 non-null  int64  
 32  month_recorded         59400 non-null  int64  
 33  year_recorded          59400 non-null  int64  
 34  season_recorded        59400 non-null  object 
dtypes: bool(2), float64(2), int64(11), object(20)
memory usage: 18.0+ MB
```

5. Data Visualization

In [28]:
```
df.columns
```

Out[28]:
```
Index(['id', 'amount_tsh', 'gps_height', 'longitude', 'latitude',
       'num_private', 'basin', 'region', 'region_code', 'district_code',
       'population', 'public_meeting', 'scheme_management', 'permit',
       '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', 'status_group',
       'month_recorded', 'year_recorded', 'season_recorded'],
      dtype='object')
```

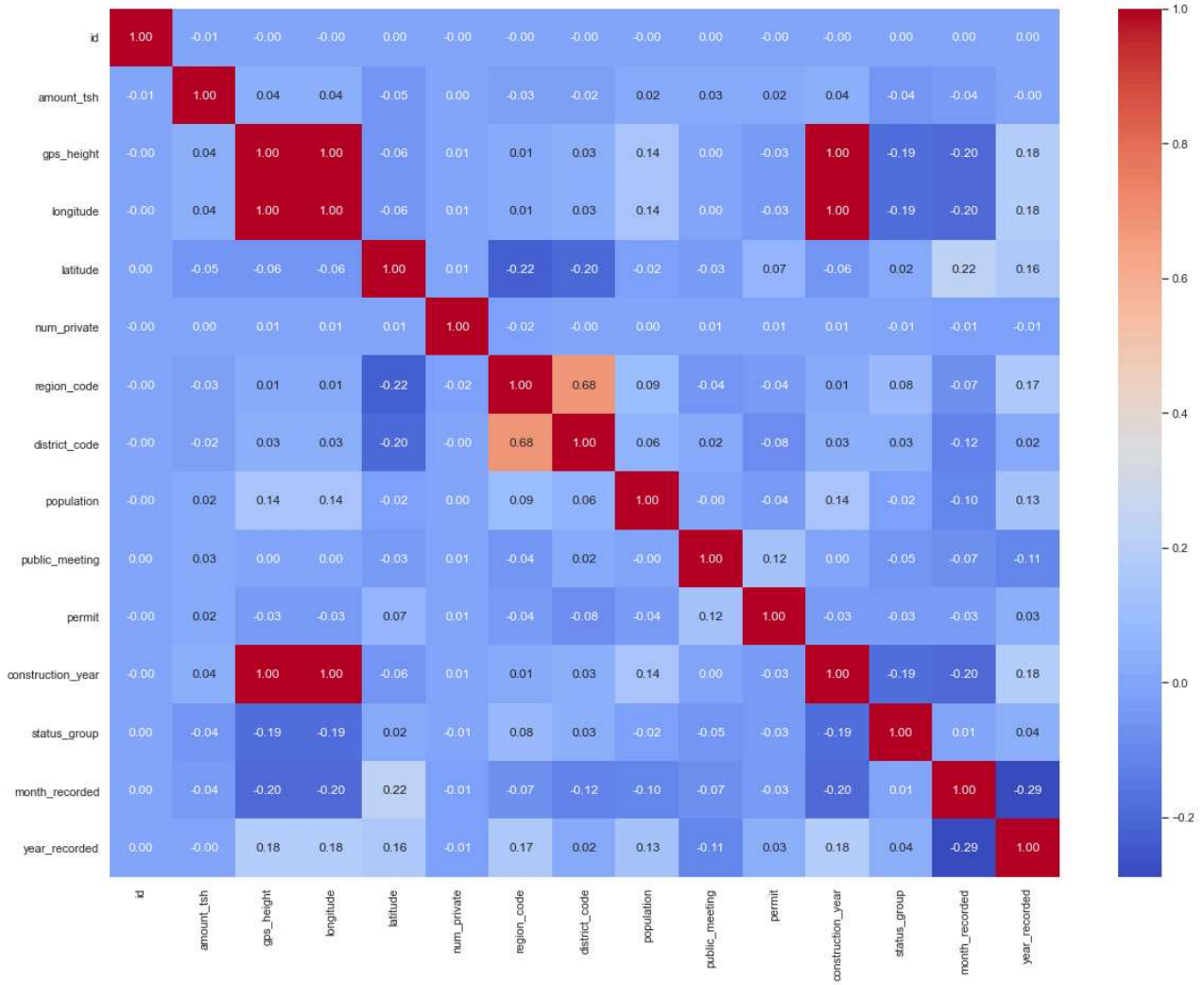
In [29]:
```
# Correlation of the target and features of our dataset
df.corr()['status_group'].sort_values(ascending = False)
```

Out[29]:
```
status_group         1.000000
region_code          0.083590
```

```
year_recorded        0.043088
district_code        0.034005
latitude             0.021020
month_recorded       0.009637
id                   0.003354
num_private         -0.006159
population          -0.015198
permit              -0.029597
amount_tsh          -0.043533
public_meeting      -0.053355
construction_year   -0.187401
longitude           -0.187401
gps_height          -0.187401
Name: status_group, dtype: float64
```

Correlation Matrix (Heatmap)

In [30]:
```
#correlation heatmap of numerical features
plt.figure(figsize = (20,15))
sns.heatmap(df.corr(), annot = True,cmap='coolwarm', fmt=".2f");
```



Despite the low correlation among the features and target, the gps_height and region have the highest correlation, We also observe a strong correlation between region_code and district_code.

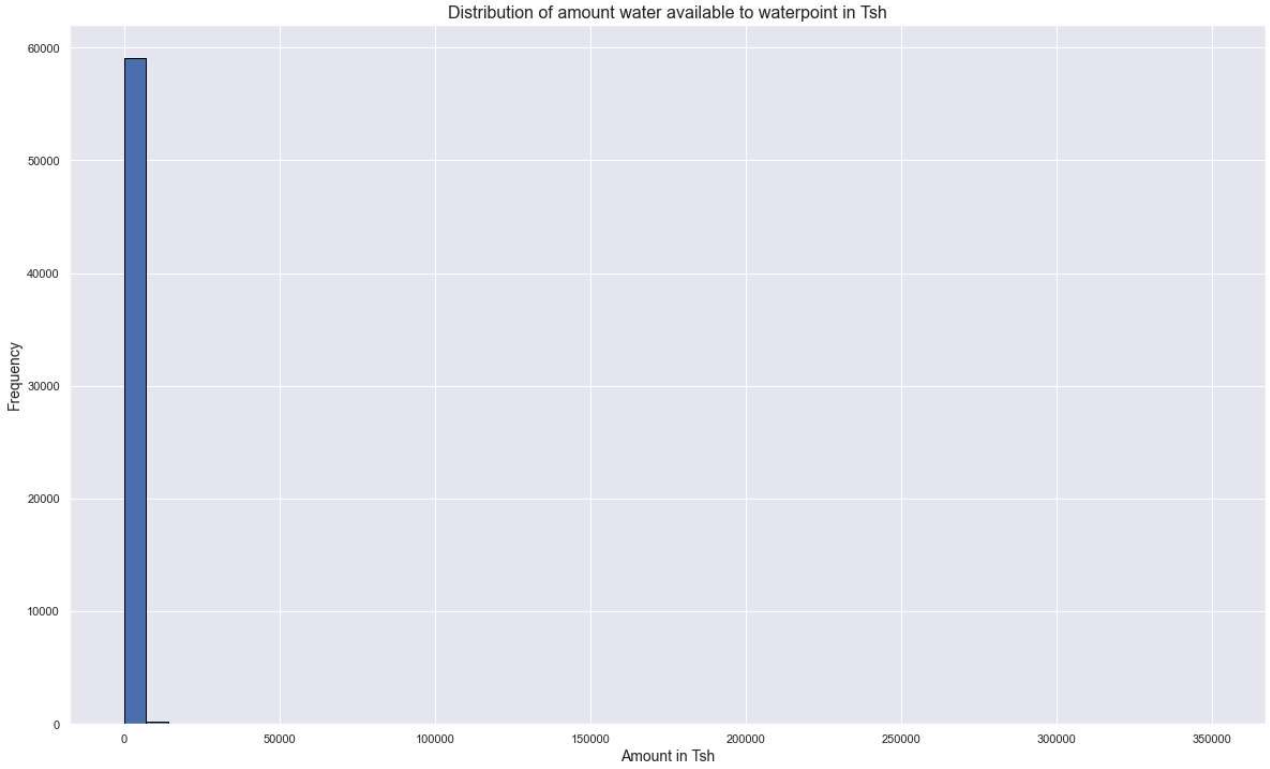
In [ ]:

In [31]:
```
#we drop the region_code column due to the possiblity of high multicoliniariy with the
df.drop(['region_code'], axis=1, inplace=True)
```

In [ ]:
```
```

Distribution of amount_tsh

In [32]:
```
plt.figure(figsize=(20, 12))
plt.hist(df['amount_tsh'], bins = 50, edgecolor =  'black')
plt.title('Distribution of amount water available to waterpoint in Tsh', fontsize = 16)
plt.xlabel('Amount in Tsh', fontsize = 14)
plt.ylabel('Frequency', fontsize = 14)
plt.show;
```



The Distribution of amount water available to waterpoint in Tsh is negatively skewed.

In [33]:
```
df['status_group'].value_counts()
```

Out[33]:
```
0    32259
1    22824
2     4317
Name: status_group, dtype: int64
```

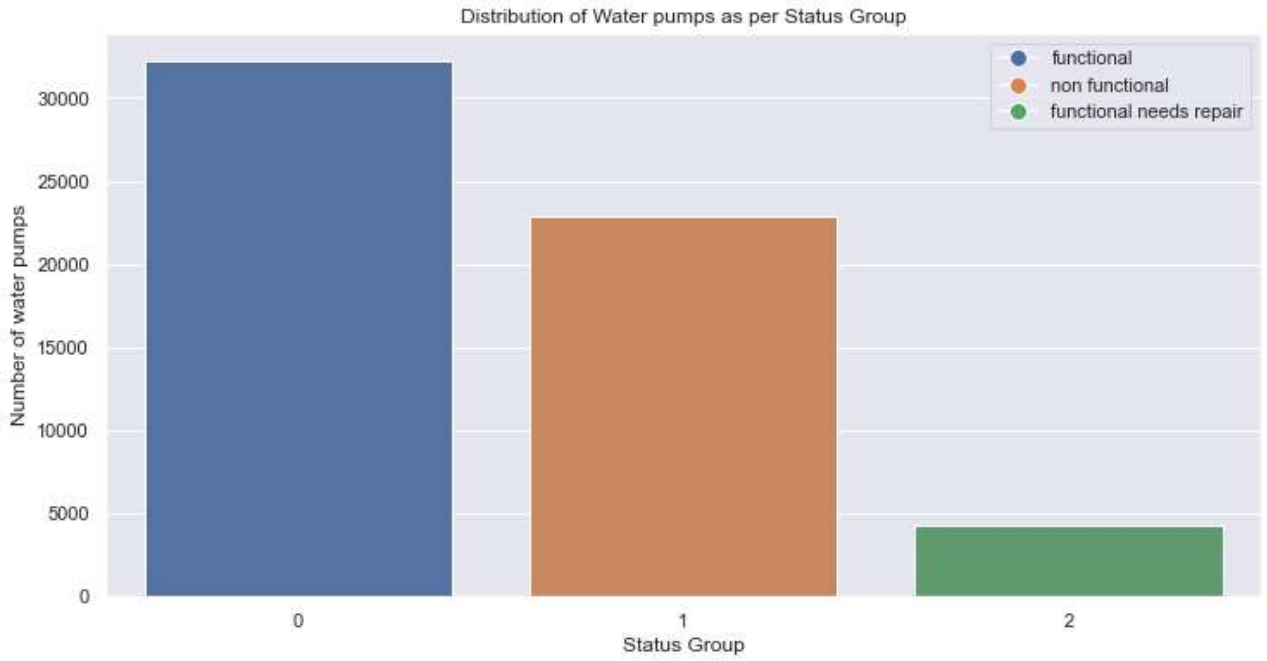
Distribution of status group

In [34]:
```
key = {
    'functional': 0,
    'non functional': 1,
    'functional needs repair': 2
}
plt.figure(figsize=(12, 6))
ax = sns.countplot(data=df, x='status_group')
```

```
plt.xlabel('Status Group')
plt.ylabel('Number of water pumps')

plt.title('Distribution of Water pumps as per Status Group')
legend_labels = list(key.keys())
legend_handles = [plt.Line2D([0], [0], marker='o', color='w', label = label,
                             markerfacecolor=sns.color_palette()[key[label]], markersiz
plt.legend(handles=legend_handles, labels=legend_labels, loc='upper right')

plt.show();
```

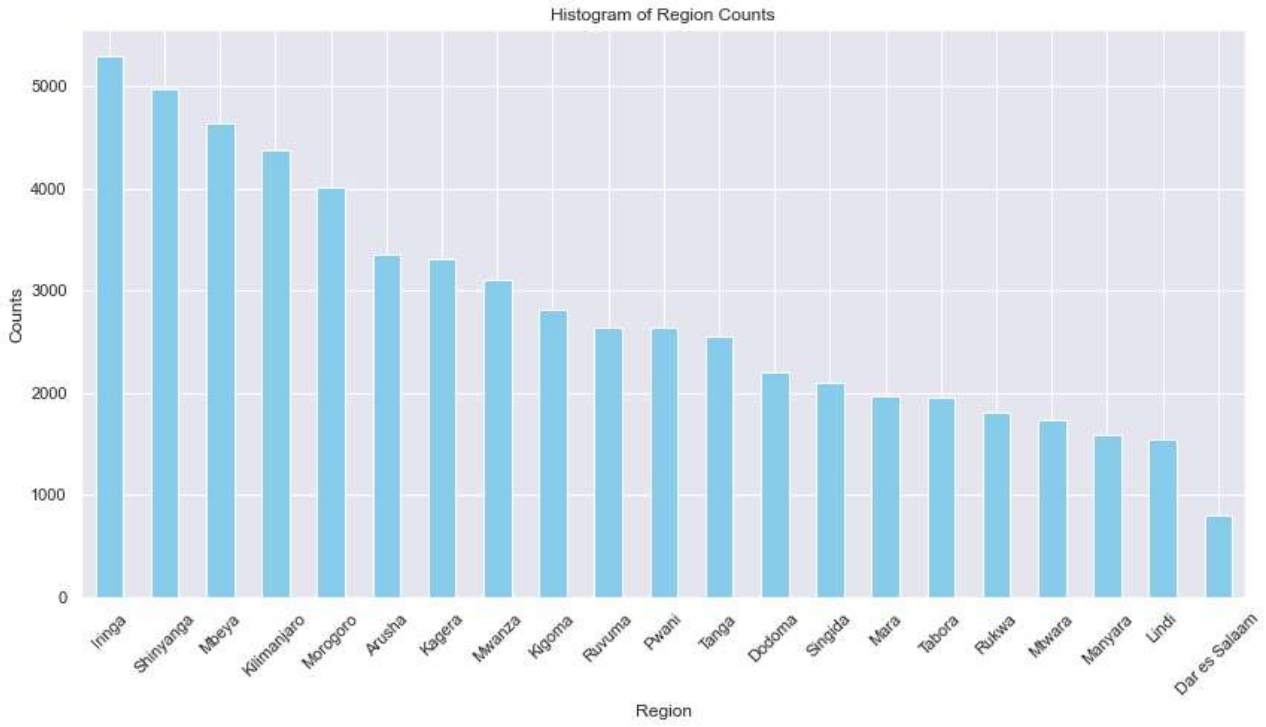


Distribution of water pumps among districts

In [35]:

```
df_region = df['region'].value_counts()

plt.figure(figsize=(14, 7))
df_region.plot(kind='bar', color='skyblue')
plt.title('Histogram of Region Counts')
plt.xlabel('Region')
plt.ylabel('Counts')
plt.xticks(rotation=45)
plt.show();
```

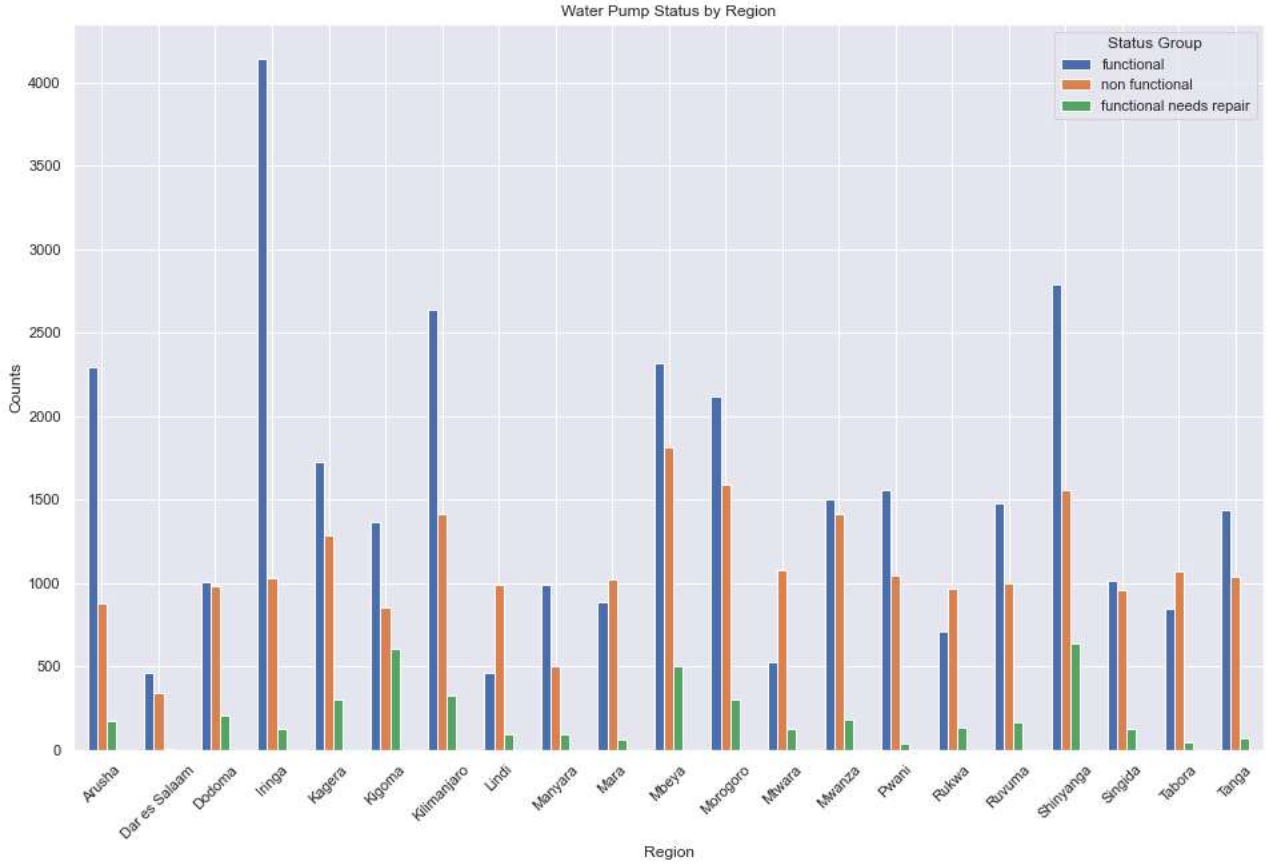


Distribution of water pump status based on the region

In [36]:

```
# Group by 'region' and 'status_group', then count occurrences
df_grouped = df.groupby(['region', 'status_group']).size().unstack(fill_value=0)

# Renaming the columns of our grouped df
df_grouped = df_grouped.rename(columns={
    0: 'functional',
    1: 'non functional',
    2:'functional needs repair'
})
# Plotting the grouped barchart
df_grouped.plot(kind='bar', figsize=(16, 10), stacked=False)
plt.title('Water Pump Status by Region')
plt.xlabel('Region')
plt.ylabel('Counts')
plt.xticks(rotation=45)
plt.legend(title='Status Group')
plt.show()
```

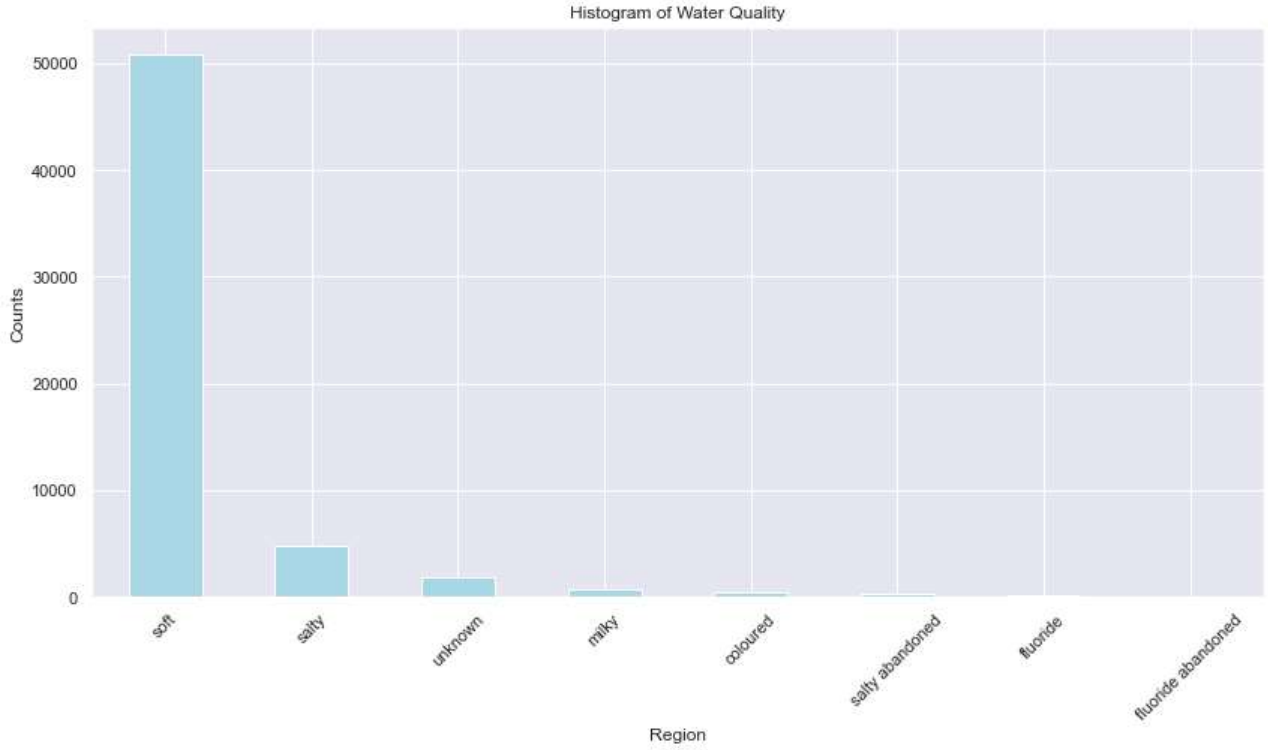


Distribution of water quality

In [37]:

```
df_water_quality = df['water_quality'].value_counts()

plt.figure(figsize=(14, 7))
df_water_quality.plot(kind='bar', color='lightblue')
plt.title('Histogram of Water Quality')
plt.xlabel('Region')
plt.ylabel('Counts')
plt.xticks(rotation=45)
plt.show();
```

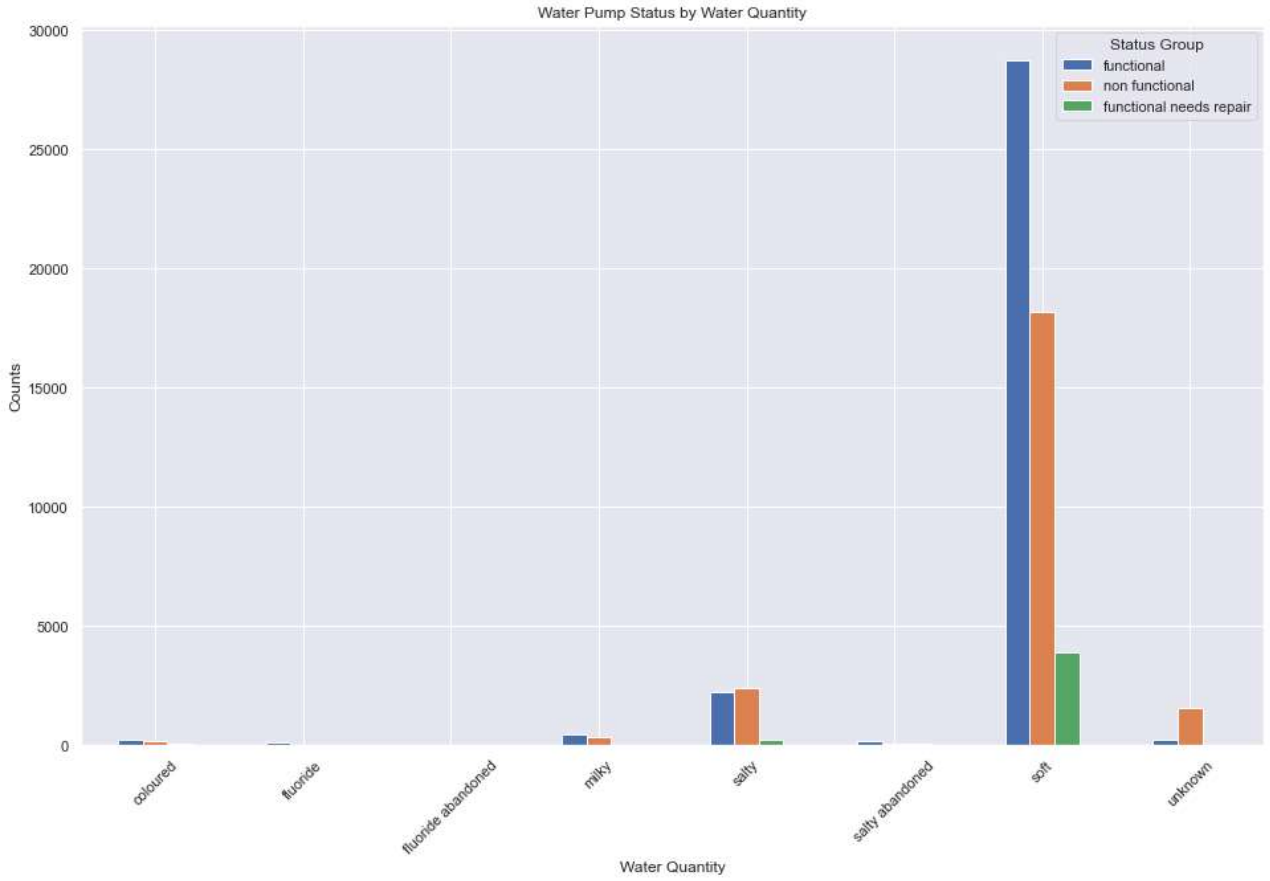


Distribution of Pump status and Water Quality Group

In [38]:

```
# Group by 'water quantity' and 'status_group', then count occurrences
df_grouped = df.groupby(['water_quality', 'status_group']).size().unstack(fill_value=0)

# Renaming the columns of our grouped df
df_grouped = df_grouped.rename(columns={
    0: 'functional',
    1: 'non functional',
    2:'functional needs repair'
})
# Plotting the grouped barchart
df_grouped.plot(kind='bar', figsize=(16, 10), stacked=False)
plt.title('Water Pump Status by Water Quantity')
plt.xlabel('Water Quantity')
plt.ylabel('Counts')
plt.xticks(rotation=45)
plt.legend(title='Status Group')
plt.show()
```



In [ ]:

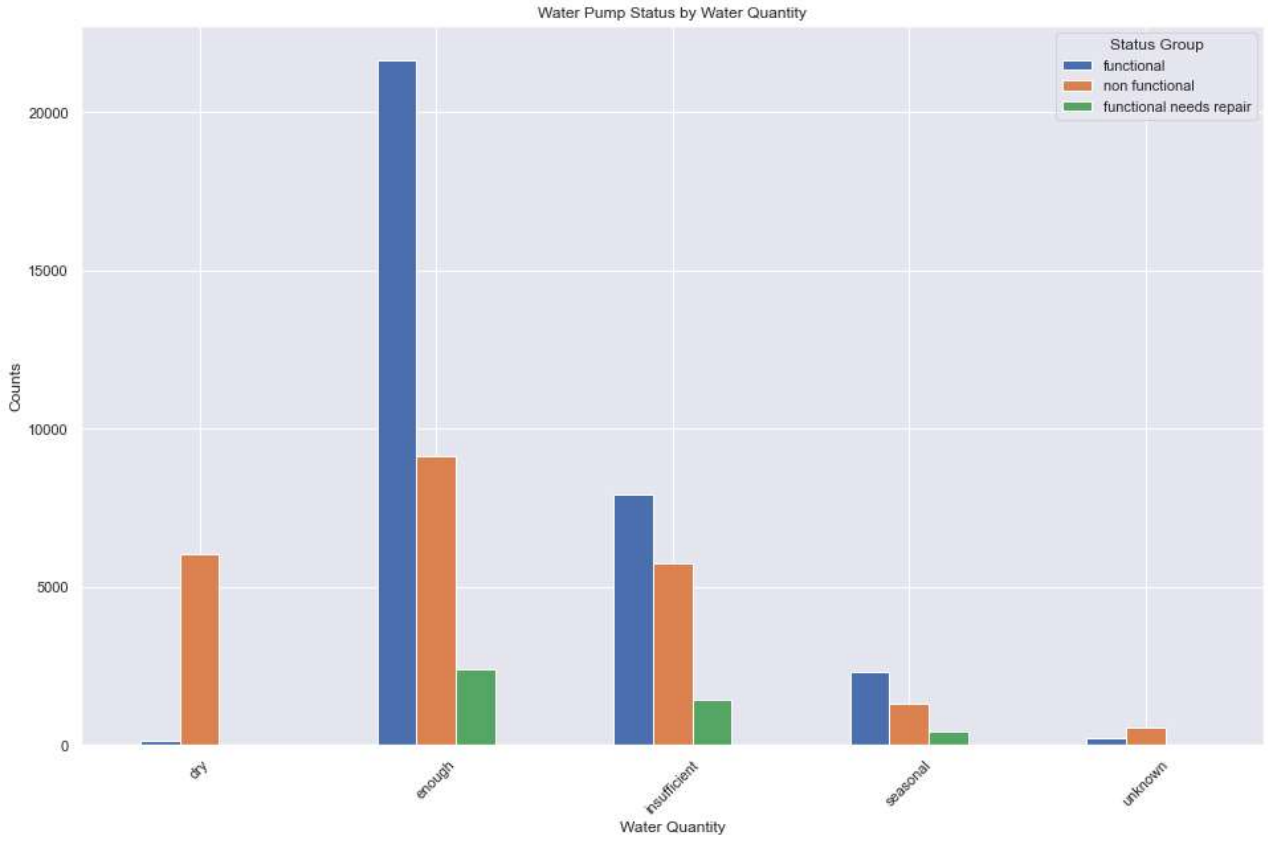
Distribution of Pump status and Water Quantity Group

In [39]:
```
df['quantity_group'].unique()
```

Out[39]:
```
array(['enough', 'insufficient', 'dry', 'seasonal', 'unknown'],
      dtype=object)
```

In [40]:
```
# Group by 'water quantity' and 'status_group', then count occurrences
df_grouped = df.groupby(['quantity_group', 'status_group']).size().unstack(fill_value=0

# Renaming the columns of our grouped df
df_grouped = df_grouped.rename(columns={
    0: 'functional',
    1: 'non functional',
    2:'functional needs repair'
})
# Plotting the grouped barchart
df_grouped.plot(kind='bar', figsize=(16, 10), stacked=False)
plt.title('Water Pump Status by Water Quantity')
plt.xlabel('Water Quantity')
plt.ylabel('Counts')
plt.xticks(rotation=45)
plt.legend(title='Status Group')
plt.show()
```



6. Data Modeling

In [41]:
```
# Creating dummy variables for our categorical variables

cols = df.select_dtypes(exclude=[np.number])
list(cols)
```

Out[41]:
```
['basin',
 'region',
 'public_meeting',
 'scheme_management',
 'permit',
 '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',
 'season_recorded']
```

In [42]:
```
#Creating Dummy variables for our categorical data

columns_create_dummies = ['basin',
```

```
 'region',
 'public_meeting',
 'scheme_management',
 'permit',
 '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',
 'season_recorded']

for column in columns_create_dummies:
    df = pd.get_dummies(df, columns=[column], drop_first=True)

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59400 entries, 0 to 59399
Columns: 164 entries, id to season_recorded_Wet
dtypes: float64(2), int64(10), uint8(152)
memory usage: 17.0 MB
```

Simple Linear Regression

In [43]:
```
#Defining our target and feature variables
y = df['status_group']
X = df['gps_height']

# Create an OLS model
lin_model = sm.OLS(endog = y, exog = sm.add_constant(X))
results = lin_model.fit()
```

In [44]:
```
results.summary()
```

Out[44]:

OLS Regression Results

Dep. Variable:	status_group	**R-squared:**	0.035
Model:	OLS	**Adj. R-squared:**	0.035
Method:	Least Squares	**F-statistic:**	2162.
Date:	Thu, 23 May 2024	**Prob (F-statistic):**	0.00
Time:	00:12:00	**Log-Likelihood:**	-55596.
No. Observations:	59400	**AIC:**	1.112e+05
Df Residuals:	59398	**BIC:**	1.112e+05
Df Model:	1		

Covariance Type: nonrobust

	coef	std err	t	P>|t|	[0.025	0.975]
const	21.2297	0.445	47.685	0.000	20.357	22.102
gps_height	-0.0104	0.000	-46.497	0.000	-0.011	-0.010

Omnibus:	5033.785	**Durbin-Watson:**	2.001
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	6364.312
Skew:	0.796	**Prob(JB):**	0.00
Kurtosis:	2.810	**Cond. No.**	3.51e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.51e+05. This might indicate that there are strong multicollinearity or other numerical problems.

In [45]:
```
df.columns
```

Out[45]:
```
Index(['id', 'amount_tsh', 'gps_height', 'longitude', 'latitude',
       'num_private', 'district_code', 'population', 'construction_year',
       'status_group',
       ...
       'waterpoint_type_dam', 'waterpoint_type_hand pump',
       'waterpoint_type_improved spring', 'waterpoint_type_other',
       'waterpoint_type_group_communal standpipe', 'waterpoint_type_group_dam',
       'waterpoint_type_group_hand pump',
       'waterpoint_type_group_improved spring', 'waterpoint_type_group_other',
       'season_recorded_Wet'],
      dtype='object', length=164)
```

Multiple Linear Regression

Defining our target and feature variables

In [46]:
```
X = df.drop(columns = ['id', 'status_group'])
y = df['status_group']
```

Creating our train and test dataframes

In [48]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=
```

Scaling the numerical features

In [49]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [ ]:
```

```

In [51]:
```
# Create an OLS model
lin_model = sm.OLS(endog = y, exog = sm.add_constant(X))
results = lin_model.fit()

#Printing a summary of our results
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           status_group   R-squared:                       0.205
Model:                            OLS   Adj. R-squared:                  0.204
Method:                 Least Squares   F-statistic:                     139.2
Date:                Thu, 23 May 2024   Prob (F-statistic):               0.00
Time:                        00:16:24   Log-Likelihood:                -49837.
No. Observations:               59400   AIC:                         9.990e+04
Df Residuals:                   59289   BIC:                         1.009e+05
Df Model:                         110
Covariance Type:            nonrobust
================================================================================================
                                                   coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------------------------
const                                           -2.7991      9.311     -0.301      0.764
-21.048      15.450
amount_tsh                                   -2.95e-06   7.91e-07     -3.730      0.000
-4.5e-06    -1.4e-06
gps_height                                      -0.0016   8.22e-05    -19.748      0.000
-0.002      -0.001
longitude                                       -0.0016   8.22e-05    -19.748      0.000
-0.002      -0.001
latitude                                         0.0467      0.004     11.747      0.000
0.039       0.055
num_private                                      0.0001      0.000      0.655      0.513
-0.000       0.000
district_code                                -5.227e-05      0.000     -0.158      0.874
-0.001       0.001
population                                   -2.312e-05   5.29e-06     -4.370      0.000
-3.35e-05   -1.28e-05
construction_year                               -0.0016   8.22e-05    -19.748      0.000
-0.002      -0.001
month_recorded                                  -0.0048      0.001     -5.117      0.000
-0.007      -0.003
year_recorded                                    0.0077      0.008      0.951      0.342
-0.008       0.024
basin_Lake Nyasa                                -0.1036      0.024     -4.384      0.000
-0.150      -0.057
basin_Lake Rukwa                                 0.1533      0.023      6.724      0.000
0.109       0.198
basin_Lake Tanganyika                            0.0397      0.016      2.462      0.014
0.008       0.071
basin_Lake Victoria                              0.0984      0.017      5.677      0.000
0.064       0.132
basin_Pangani                                    0.0485      0.016      2.974      0.003
0.017       0.080
basin_Rufiji                                    -0.0032      0.020     -0.157      0.875
-0.043       0.037
basin_Ruvuma / Southern Coast                    0.0548      0.030      1.798      0.072
-0.005       0.115
basin_Wami / Ruvu                               -0.0315      0.019     -1.622      0.105
-0.070       0.007
region_Dar es Salaam                             0.2300      0.037      6.222      0.000
0.158       0.302
```

```
region_Dodoma                          0.2731      0.029      9.532      0.000
0.217       0.329
region_Iringa                          0.2538      0.033      7.688      0.000
0.189       0.319
region_Kagera                          0.0403      0.031      1.314      0.189
-0.020       0.100
region_Kigoma                          0.3903      0.026     14.735      0.000
0.338       0.442
region_Kilimanjaro                     0.1759      0.016     10.857      0.000
0.144       0.208
region_Lindi                           0.4971      0.039     12.753      0.000
0.421       0.573
region_Manyara                         0.1662      0.020      8.223      0.000
0.127       0.206
region_Mara                            0.1198      0.028      4.209      0.000
0.064       0.176
region_Mbeya                           0.4856      0.033     14.705      0.000
0.421       0.550
region_Morogoro                        0.4197      0.031     13.546      0.000
0.359       0.480
region_Mtwara                          0.5376      0.041     13.213      0.000
0.458       0.617
region_Mwanza                         -0.0248      0.031     -0.800      0.423
-0.085       0.036
region_Pwani                           0.3895      0.032     11.989      0.000
0.326       0.453
region_Rukwa                           0.3824      0.031     12.219      0.000
0.321       0.444
region_Ruvuma                          0.4778      0.037     12.945      0.000
0.405       0.550
region_Shinyanga                       0.1448      0.023      6.270      0.000
0.100       0.190
region_Singida                         0.2140      0.022      9.727      0.000
0.171       0.257
region_Tabora                          0.1773      0.025      7.036      0.000
0.128       0.227
region_Tanga                           0.1050      0.024      4.458      0.000
0.059       0.151
public_meeting_True                   -0.0897      0.008    -11.440      0.000
-0.105      -0.074
scheme_management_None                -0.2759      0.562     -0.491      0.623
-1.377       0.826
scheme_management_Other               -0.1527      0.043     -3.553      0.000
-0.237      -0.068
scheme_management_Parastatal           0.0653      0.044      1.473      0.141
-0.022       0.152
scheme_management_Private operator     0.0804      0.037      2.168      0.030
0.008       0.153
scheme_management_SWC                 -0.0277      0.129     -0.215      0.829
-0.280       0.224
scheme_management_Trust               -0.2344      0.118     -1.983      0.047
-0.466      -0.003
scheme_management_VWC                 -0.0428      0.032     -1.317      0.188
-0.106       0.021
scheme_management_WUA                 -0.0742      0.036     -2.040      0.041
-0.146      -0.003
scheme_management_WUG                  0.0520      0.036      1.450      0.147
-0.018       0.122
scheme_management_Water Board         -0.2030      0.037     -5.505      0.000
-0.275      -0.131
scheme_management_Water authority      0.0409      0.035      1.176      0.239
-0.027       0.109
permit_True                           -0.0396      0.006     -6.413      0.000
-0.052      -0.027
extraction_type_cemo                  -0.1528      0.420     -0.364      0.716
```

```
-0.975       0.670
extraction_type_climax                          -0.0293      0.422     -0.069      0.945
-0.856       0.798
extraction_type_gravity                         -0.3656      1.354     -0.270      0.787
-3.020       2.289
extraction_type_india mark ii                    0.0147      0.009      1.606      0.108
-0.003       0.033
extraction_type_india mark iii                   0.1101      0.029      3.745      0.000
0.052       0.168
extraction_type_ksb                             -0.1051      0.542     -0.194      0.846
-1.167       0.957
extraction_type_mono                            -0.1883      0.625     -0.301      0.763
-1.414       1.037
extraction_type_nira/tanira                      0.0011      0.008      0.131      0.896
-0.015       0.017
extraction_type_other                           -0.2401      0.903     -0.266      0.790
-2.010       1.529
extraction_type_other - mkulima/shinyanga        0.3121      0.318      0.982      0.326
-0.311       0.935
extraction_type_other - play pump                0.0504      0.095      0.528      0.597
-0.137       0.238
extraction_type_other - rope pump               -0.2892      0.903     -0.320      0.749
-2.059       1.480
extraction_type_other - swn 81                  -0.0992      0.087     -1.135      0.256
-0.270       0.072
extraction_type_submersible                     -0.1888      0.542     -0.348      0.727
-1.250       0.873
extraction_type_swn 80                           0.0550      0.009      6.229      0.000
0.038       0.072
extraction_type_walimi                          -0.0279      0.104     -0.269      0.788
-0.231       0.176
extraction_type_windmill                        -0.2339      0.903     -0.259      0.796
-2.004       1.536
extraction_type_group_gravity                   -0.3656      1.354     -0.270      0.787
-3.020       2.289
extraction_type_group_india mark ii              0.0147      0.009      1.606      0.108
-0.003       0.033
extraction_type_group_india mark iii             0.1101      0.029      3.745      0.000
0.052       0.168
extraction_type_group_mono                      -0.1883      0.625     -0.301      0.763
-1.414       1.037
extraction_type_group_nira/tanira                0.0011      0.008      0.131      0.896
-0.015       0.017
extraction_type_group_other                     -0.2401      0.903     -0.266      0.790
-2.010       1.529
extraction_type_group_other handpump             0.2354      0.083      2.835      0.005
0.073       0.398
extraction_type_group_other motorpump           -0.1821      0.833     -0.218      0.827
-1.816       1.451
extraction_type_group_rope pump                 -0.2892      0.903     -0.320      0.749
-2.059       1.480
extraction_type_group_submersible               -0.2938      1.083     -0.271      0.786
-2.417       1.829
extraction_type_group_swn 80                     0.0550      0.009      6.229      0.000
0.038       0.072
extraction_type_group_wind-powered              -0.2339      0.903     -0.259      0.796
-2.004       1.536
extraction_type_class_handpump                  -1.0061      2.707     -0.372      0.710
-6.312       4.300
extraction_type_class_motorpump                 -0.3704      1.458     -0.254      0.799
-3.229       2.488
extraction_type_class_other                     -0.2401      0.903     -0.266      0.790
-2.010       1.529
extraction_type_class_rope pump                 -0.2892      0.903     -0.320      0.749
-2.059       1.480
```

```
extraction_type_class_submersible             -0.2938      1.083     -0.271      0.786
-2.417       1.829
extraction_type_class_wind-powered            -0.2339      0.903     -0.259      0.796
-2.004       1.536
management_other                              -0.1427      0.046     -3.070      0.002
-0.234      -0.052
management_other - school                      0.0880      0.084      1.052      0.293
-0.076       0.252
management_parastatal                         -0.0780      0.025     -3.161      0.002
-0.126      -0.030
management_private operator                   -0.3590      0.041     -8.820      0.000
-0.439      -0.279
management_trust                              -0.0146      0.117     -0.125      0.900
-0.243       0.214
management_unknown                            -0.0715      0.024     -3.009      0.003
-0.118      -0.025
management_vwc                                 0.0303      0.011      2.651      0.008
0.008       0.053
management_water authority                    -0.1651      0.046     -3.603      0.000
-0.255      -0.075
management_water board                        -0.0329      0.018     -1.863      0.062
-0.068       0.002
management_wua                                 0.0260      0.021      1.255      0.209
-0.015       0.067
management_wug                                -0.0976      0.015     -6.357      0.000
-0.128      -0.067
management_group_other                        -0.0547      0.049     -1.105      0.269
-0.152       0.042
management_group_parastatal                   -0.0780      0.025     -3.161      0.002
-0.126      -0.030
management_group_unknown                      -0.0715      0.024     -3.009      0.003
-0.118      -0.025
management_group_user-group                   -0.0742      0.032     -2.340      0.019
-0.136      -0.012
payment_other                                 -0.2894      1.035     -0.280      0.780
-2.317       1.739
payment_pay annually                          -0.6889      2.069     -0.333      0.739
-4.744       3.366
payment_pay monthly                           -0.3149      1.035     -0.304      0.761
-2.343       1.713
payment_pay per bucket                        -0.3624      1.035     -0.350      0.726
-2.390       1.665
payment_pay when scheme fails                 -0.3151      1.035     -0.305      0.761
-2.343       1.713
payment_unknown                               -0.3078      1.034     -0.298      0.766
-2.335       1.720
payment_type_monthly                          -0.3149      1.035     -0.304      0.761
-2.343       1.713
payment_type_never pay                        -0.5206      2.069     -0.252      0.801
-4.576       3.534
payment_type_on failure                       -0.3151      1.035     -0.305      0.761
-2.343       1.713
payment_type_other                            -0.2894      1.035     -0.280      0.780
-2.317       1.739
payment_type_per bucket                       -0.3624      1.035     -0.350      0.726
-2.390       1.665
payment_type_unknown                          -0.3078      1.034     -0.298      0.766
-2.335       1.720
water_quality_fluoride                        -0.1303      0.053     -2.439      0.015
-0.235      -0.026
water_quality_fluoride abandoned               0.1141      0.092      1.238      0.216
-0.067       0.295
water_quality_milky                           -0.0682      0.017     -4.026      0.000
-0.101      -0.035
water_quality_salty                           -0.0650      0.015     -4.409      0.000
```

```
-0.094      -0.036
water_quality_salty abandoned                 0.1297      0.023      5.658      0.000
0.085       0.175
water_quality_soft                           -0.0131      0.013     -1.007      0.314
-0.039       0.012
water_quality_unknown                        -0.0327      0.015     -2.119      0.034
-0.063      -0.002
quality_group_fluoride                       -0.0163      0.050     -0.323      0.747
-0.115       0.082
quality_group_good                           -0.0131      0.013     -1.007      0.314
-0.039       0.012
quality_group_milky                          -0.0682      0.017     -4.026      0.000
-0.101      -0.035
quality_group_salty                           0.0646      0.020      3.159      0.002
0.025       0.105
quality_group_unknown                        -0.0327      0.015     -2.119      0.034
-0.063      -0.002
quantity_enough                              -0.2299      0.004    -52.610      0.000
-0.239      -0.221
quantity_insufficient                        -0.1711      0.005    -36.169      0.000
-0.180      -0.162
quantity_seasonal                            -0.1944      0.006    -30.629      0.000
-0.207      -0.182
quantity_unknown                             -0.1098      0.012     -8.945      0.000
-0.134      -0.086
quantity_group_enough                        -0.2299      0.004    -52.610      0.000
-0.239      -0.221
quantity_group_insufficient                  -0.1711      0.005    -36.169      0.000
-0.180      -0.162
quantity_group_seasonal                      -0.1944      0.006    -30.629      0.000
-0.207      -0.182
quantity_group_unknown                       -0.1098      0.012     -8.945      0.000
-0.134      -0.086
source_hand dtw                              -0.6080      2.280     -0.267      0.790
-5.076       3.860
source_lake                                  -0.0851      0.241     -0.354      0.724
-0.557       0.386
source_machine dbh                           -0.6646      2.280     -0.291      0.771
-5.133       3.804
source_other                                 -0.2496      0.457     -0.546      0.585
-1.146       0.647
source_rainwater harvesting                  -0.1494      0.360     -0.415      0.678
-0.855       0.556
source_river                                 -0.0455      0.240     -0.189      0.850
-0.516       0.425
source_shallow well                          -0.3391      1.140     -0.297      0.766
-2.573       1.895
source_spring                                -0.3734      1.140     -0.328      0.743
-2.607       1.861
source_unknown                               -0.0907      0.458     -0.198      0.843
-0.988       0.807
source_type_dam                              -0.1939      0.720     -0.269      0.788
-1.605       1.218
source_type_other                            -0.3403      0.912     -0.373      0.709
-2.128       1.447
source_type_rainwater harvesting             -0.1494      0.360     -0.415      0.678
-0.855       0.556
source_type_river/lake                       -0.1305      0.480     -0.272      0.786
-1.071       0.810
source_type_shallow well                     -0.3391      1.140     -0.297      0.766
-2.573       1.895
source_type_spring                           -0.3734      1.140     -0.328      0.743
-2.607       1.861
source_class_surface                         -0.4737      1.560     -0.304      0.761
-3.531       2.584
```

```
source_class_unknown                            -0.3403      0.912     -0.373      0.709
-2.128       1.447
waterpoint_type_communal standpipe              -0.0135      0.018     -0.743      0.458
-0.049       0.022
waterpoint_type_communal standpipe multiple      0.1699      0.019      9.157      0.000
0.134        0.206
waterpoint_type_dam                             -0.1106      0.110     -1.007      0.314
-0.326       0.105
waterpoint_type_hand pump                        0.1027      0.028      3.616      0.000
0.047        0.158
waterpoint_type_improved spring                 -0.0334      0.029     -1.161      0.246
-0.090       0.023
waterpoint_type_other                            0.1868      0.027      6.855      0.000
0.133        0.240
waterpoint_type_group_communal standpipe         0.1564      0.036      4.401      0.000
0.087        0.226
waterpoint_type_group_dam                       -0.1106      0.110     -1.007      0.314
-0.326       0.105
waterpoint_type_group_hand pump                  0.1027      0.028      3.616      0.000
0.047        0.158
waterpoint_type_group_improved spring           -0.0334      0.029     -1.161      0.246
-0.090       0.023
waterpoint_type_group_other                      0.1868      0.027      6.855      0.000
0.133        0.240
season_recorded_Wet                              0.0215      0.006      3.617      0.000
0.010        0.033
==============================================================================
Omnibus:                     8779.934   Durbin-Watson:                   1.999
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            13345.312
Skew:                           1.076   Prob(JB):                         0.00
Kurtosis:                       3.872   Cond. No.                     1.00e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[2] The smallest eigenvalue is 9.55e-21. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Observations.

The Feature variables only explain 20% in the variability of our target variable Further analysis is required to enable us increase the predictive power of our model, using classification models

7. Classification Analysis

Logistic Regression

In [52]:

```
# Model training
model = LogisticRegression(max_iter=100, tol=1e-4, solver='sag', random_state=42)
model.fit(X_train, y_train)

# Model evaluation
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.72      0.90      0.80      6457
           1       0.79      0.64      0.71      4572
```

```
           2       0.49      0.07      0.12       851

    accuracy                           0.74     11880
   macro avg       0.66      0.53      0.54     11880
weighted avg       0.73      0.74      0.71     11880

[[5781  642   34]
 [1625 2921   26]
 [ 645  149   57]]
C:\Users\USER\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_sag.py:329: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn("The max_iter was reached which means "
```

Observations

Our Logistic Regression perfoms with weighted precision of only 0.73 and an accuracy of 74%

In [ ]:

Gradient Boosting Model

In [53]:
```
from sklearn.ensemble import GradientBoostingClassifier

# Model training and fitting
model = GradientBoostingClassifier(max_depth=8,random_state=42)
model.fit(X_train, y_train)

# Model evaluation
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.91      0.84      6457
           1       0.85      0.74      0.79      4572
           2       0.60      0.30      0.40       851

    accuracy                           0.80     11880
   macro avg       0.74      0.65      0.68     11880
weighted avg       0.80      0.80      0.79     11880

[[5864  476  117]
 [1121 3396   55]
 [ 465  130  256]]
```

Observations

Our Gradient boosting model gives us a weighted precision 0.8 and an accuracy of 80% The model performs well

In [ ]:

Decision Tree Model

In [54]:
```
from sklearn.tree import DecisionTreeClassifier

# Model training
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
```

```
# Model evaluation
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.79      0.80      6457
           1       0.76      0.76      0.76      4572
           2       0.36      0.38      0.37       851

    accuracy                           0.75     11880
   macro avg       0.64      0.64      0.64     11880
weighted avg       0.75      0.75      0.75     11880

[[5111  963  383]
 [ 904 3482  186]
 [ 372  155  324]]
```

Observations

Our Decision Tree model gives us a weighted precision 0.75 and an accuracy of 75%

In [ ]:

Random Forest Model

In [55]:
```
from sklearn.ensemble import RandomForestClassifier

# Model training
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Model evaluation
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.84      0.82      6457
           1       0.80      0.78      0.79      4572
           2       0.44      0.36      0.39       851

    accuracy                           0.78     11880
   macro avg       0.68      0.66      0.67     11880
weighted avg       0.78      0.78      0.78     11880

[[5439  748  270]
 [ 889 3559  124]
 [ 401  144  306]]
```

Observations

Our Random Forest model gives us a weighted precision 0.78 and an accuracy of 78% The model performs well

In [ ]:

K-Nearest Neighbors

In [56]:
```
from sklearn.neighbors import KNeighborsClassifier

# Model training
model = KNeighborsClassifier(n_neighbors=5)
model.fit(X_train, y_train)

# Model evaluation
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.77      0.86      0.81      6457
           1       0.79      0.73      0.76      4572
           2       0.50      0.29      0.37       851

    accuracy                           0.77     11880
   macro avg       0.69      0.63      0.65     11880
weighted avg       0.76      0.77      0.76     11880

[[5533  758  166]
 [1160 3335   77]
 [ 455  150  246]]
```

Observations

Our K-nearest Neighbors model gives us a weighted precision 0.71 and an accuracy of 72%

In [ ]:
```
```

AdaBoost

In [57]:
```
from sklearn.ensemble import GradientBoostingClassifier

# Create and fit the Gradient Boosting model
gb = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb.fit(X_train, y_train)

# Model evaluation
y_pred = gb.predict(X_test)
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.72      0.92      0.81      6457
           1       0.83      0.62      0.71      4572
           2       0.54      0.13      0.21       851

    accuracy                           0.75     11880
   macro avg       0.70      0.56      0.58     11880
weighted avg       0.75      0.75      0.73     11880

[[5968  439   50]
 [1688 2842   42]
 [ 605  138  108]]
```

Observations

Our Adaboost model gives us a weighted precision 0.75 and an accuracy of 75%

In [ ]:

8. Model Evaluation

Overall, The best perfoming classsication model is the Gradient Boosting Model with an accurancy of 80%. Further, the random forest model performed well with an accuracy of 78%

In [ ]:

9. Conclusion

Using the Gradient Bosting Model, We can accuraly predict 80% if a water pump needs repair

In [ ]: