samuelhellen /
Tanzania-project

Code Issues Pull requests Actions Projects Security Insights

Tanzania-project / Untitled-1.ipynb

samuelhellen changes d99a545 · 5 hours ago

3146 lines (3146 loc) · 602 KB

Final Project Submission

Please fill out:

- Student name: HELLEN SAMUEL
- Student pace: full time
- Scheduled project review date/time:
- Instructor name: NIKITA
- Blog post URL:

TANZANIA'S OPERATIONAL WATER WELLS PREDICTION

BUSINESS UNDERSTANDING

OVERVIEW

Water bodies are not just beautiful; they are essential for our survival. Water is vital for household use, economic activities like producing electricity, and maintaining health and hygiene. Recent data from the World Bank show that Tanzania has about 60 million people. Many Tanzanians still face challenges in getting clean and safe water (Nsemwa, 2022). Only 30.6% of Tanzanian households use recommended methods to treat water, and just 22.8% have proper hand-washing facilities (Ministry of Health, 2019). Poor sanitation leads to around 432,000 deaths from diarrhea each year and contributes to several neglected tropical diseases like intestinal worms, schistosomiasis, and trachoma. It also worsens malnutrition (WHO, 2019).

CHALLENGES

Determine the water point type that occurs most frequently in the dataset.

Investigate whether the functionality of water points varies based on the payment type.

Understand if there's a correlation between altitude and the class of extraction type used for water points.

SOLUTIONS

By analyzing the distribution of water point types and determining the one with the highest occurrence, we can identify the most popular water point type.

Grouping the data by payment type allows us to analyze the distribution of functionality statuses within each group. This comparison helps us understand if there's any variation in functionality based on different payment types.

Understanding any correlation or patterns, we can determine if altitude affects the class of extraction type used for water points.

PROBLEM STATEMENT

Our NGO, Danida, is focused on finding and replacing water wells that need repair. Our classification model is being used to better predict which wells in the area are operational, need repairs or are non-functional using various information such as when each well was installed, who funded the project, and population around each well. Any improvement in determining the best wells to install or in the predictability of which wells need repairs could have an enormous impact on the people of Tanzania.

OBJECTIVES main objective

The primary goal of this study is to develop a predictive model that can classify the functional status of water wells in Tanzania. This model will provide valuable insights to the ministry by identifying both functional and non-functional water points, offering essential information for future well planning, and pinpointing regions vulnerable to water scarcity.

specific objective

To aid in improving maintenance operations by focusing inspections on the water points that have a high likelihood of requiring repair or having failed altogether

To provide 70%-75% accurate predictions on the functionality of wells

To determine the functionality status concerning payment type

DATA UNDERSTANDING

The data is sourced from Taarifa and the Tanzanian Ministry of Water. Data utilized can be found here: https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/data/

For the purposes of our evaluation, we are utilizing the Training Set Labels and Training Set Values, which include data from 59,400 pumps. Our cleaned data contains information from 59,028 pumps.

The target variable is status_group with the labels:

functional - the waterpoint is operational and there are no repairs needed functional needs repair - the waterpoint is operational, but needs repairs non functional - the waterpoint is not operational The predictor variables in this data include:

amount_tsh - Total static head (amount water available to waterpoint) date_recorded - The date the row was entered funder - Who funded the well gps_height - Altitude of the well installer - Organization that installed the well longitude - GPS coordinate latitude - GPS coordinate wpt_name - Name of the waterpoint if there is one num_private - basin - Geographic water basin subvillage - Geographic location region - Geographic location region_code - Geographic location (coded) district_code - Geographic location (coded) lga - Geographic location ward - Geographic location population - Population around the well public_meeting - True/False recorded_by - Group entering this row of data scheme_management - Who operates the waterpoint scheme_name - Who operates the

waterpoint permit - If the waterpoint is permitted construction_year - Year the waterpoint was constructed extraction_type - The kind of extraction the waterpoint uses extraction_type_group - The kind of extraction the waterpoint uses extraction_type_class - The kind of extraction the waterpoint uses management - How the waterpoint is managed management_group - How the waterpoint is managed payment - What the water costs payment_type - What the water costs water_quality - The quality of the water quality_group - The quality of the water quantity - The quantity of water quantity_group - The quantity of water source - The source of the water source_type - The source of the water source_class - The source of the water waterpoint_type - The kind of waterpoint waterpoint_type_group - The kind of waterpoint

DATA PREPARATION

In [24]:
```
#import necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
import io
```

In [25]:
```
class DataLoader:
    def __init__(self, training_set_path, test_set_path):
        self.training_set_path = training_set_path
        self.test_set_path = test_set_path
        self.training_set_label = None
        self.test_set_values = None

    def load_data(self):
        self.training_set_label = pd.read_csv(self.training_set_path)
        self.test_set_values = pd.read_csv(self.test_set_path)

    def preview_data(self):
        print("Training Set Label Head:\n", self.training_set_label.head())
        print("Test Set Values Head:\n", self.test_set_values.head())

    def get_shapes(self):
        print("Training Set Label Shape:", self.training_set_label.shape)
        print("Test Set Values Shape:", self.test_set_values.shape)
if __name__ == "__main__":
    # Paths to the datasets
    training_set_path = "training set labels.csv"
    test_set_path = "test set values.csv"

    # Creating an instance of DataLoader
    data_loader = DataLoader(training_set_path, test_set_path)

    # Loading the data
    data_loader.load_data()

    # Previewing the data
    data_loader.preview_data()
```

```
    # Getting the shapes of the datasets
    data_loader.get_shapes()
```

```
Training Set Label Head:
       id    status_group
0  69572      functional
1   8776      functional
2  34310      functional
3  67743  non functional
4  19728      functional
Test Set Values Head:
       id  amount_tsh date_recorded        funder  gps_height     installer  \
0  69572      6000.0    2011-03-14         Roman        1390         Roman
1   8776         0.0    2013-03-06       Grumeti        1399       GRUMETI
2  34310        25.0    2013-02-25  Lottery Club         686  World vision
3  67743         0.0    2013-01-28        Unicef         263        UNICEF
4  19728         0.0    2011-07-13   Action In A           0       Artisan

   longitude   latitude              wpt_name  num_private  ... payment_type  \
0  34.938093  -9.856322                  none            0  ...     annually
1  34.698766  -2.147466              Zahanati            0  ...    never pay
2  37.460664  -3.821329           Kwa Mahundi            0  ...   per bucket
3  38.486161 -11.155298  Zahanati Ya Nanyumbu            0  ...    never pay
4  31.130847  -1.825359               Shuleni            0  ...    never pay

  water_quality quality_group      quantity  quantity_group  \
0          soft          good        enough          enough
1          soft          good  insufficient    insufficient
2          soft          good        enough          enough
3          soft          good           dry             dry
4          soft          good      seasonal        seasonal

                source           source_type  source_class  \
0               spring                spring   groundwater
1  rainwater harvesting  rainwater harvesting       surface
2                  dam                   dam       surface
3          machine dbh              borehole   groundwater
4  rainwater harvesting  rainwater harvesting       surface

             waterpoint_type waterpoint_type_group
0          communal standpipe    communal standpipe
1          communal standpipe    communal standpipe
2  communal standpipe multiple    communal standpipe
3  communal standpipe multiple    communal standpipe
4          communal standpipe    communal standpipe

[5 rows x 40 columns]
Training Set Label Shape: (59400, 2)
Test Set Values Shape: (59400, 40)
```

In [26]:

```
class DataProcessor:
    def __init__(self, training_set_label_path, test_set_values_path):
        self.training_set_label_path = training_set_label_path
        self.test_set_values_path = test_set_values_path
        self.df = None

    def load_data(self):
        self.training_set_label = pd.read_csv(self.training_set_label_path)
        self.test_set_values = pd.read_csv(self.test_set_values_path)
        print("Training and Test data loaded successfully.")
```

```
    def merge_data(self):
        self.df = self.test_set_values.merge(self.training_set_label, how='left'
        print("Data merged successfully.")

    def data_info(self):
        print("\nData Head:\n", self.df.head())
        print("\nData Info:\n")
        self.df.info()
        print("\nData Description:\n", self.df.describe())
        print("\nData Types:\n", self.df.dtypes.value_counts())

    def value_counts(self, column_name):
        if column_name in self.df.columns:
            print(f"\nValue counts for {column_name}:\n", self.df[column_name].v
        else:
            print(f"{column_name} not found in dataframe.")

# Example usage
if __name__ == "__main__":
    # Paths to the datasets
    training_set_label_path = "training set labels.csv"
    test_set_values_path = "test set values.csv"

    # Creating an instance of DataProcessor
    data_processor = DataProcessor(training_set_label_path, test_set_values_path

    # Loading the data
    data_processor.load_data()

    # Merging the data
    data_processor.merge_data()

    # Displaying information about the merged data
    data_processor.data_info()

    # Displaying value counts for specific columns
    columns_to_check = [
        "num_private", "management", "management_group",
        "scheme_management", "scheme_name", "payment_type",
```



```
Training and Test data loaded successfully.
Data merged successfully.

Data Head:
       id  amount_tsh date_recorded        funder  gps_height     installer  \
0  69572      6000.0    2011-03-14         Roman        1390         Roman
1   8776         0.0    2013-03-06       Grumeti        1399       GRUMETI
2  34310        25.0    2013-02-25  Lottery Club         686  World vision
3  67743         0.0    2013-01-28        Unicef         263        UNICEF
4  19728         0.0    2011-07-13   Action In A           0       Artisan

   longitude   latitude             wpt_name  num_private  ... water_quality  \
0  34.938093  -9.856322                 none            0  ...          soft
1  34.698766  -2.147466             Zahanati            0  ...          soft
```

```
2  37.460664  -3.821329           Kwa Mahundi            0  ...          soft
3  38.486161 -11.155298  Zahanati Ya Nanyumbu            0  ...          soft
4  31.130847  -1.825359               Shuleni            0  ...          soft

  quality_group      quantity  quantity_group                source  \
0          good        enough          enough                spring
1          good  insufficient    insufficient  rainwater harvesting
2          good        enough          enough                   dam
3          good           dry             dry           machine dbh
4          good      seasonal        seasonal  rainwater harvesting

            source_type source_class              waterpoint_type  \
0                spring  groundwater           communal standpipe
1  rainwater harvesting      surface           communal standpipe
2                   dam      surface  communal standpipe multiple
3              borehole  groundwater  communal standpipe multiple
4  rainwater harvesting      surface           communal standpipe

  waterpoint_type_group    status_group
0    communal standpipe      functional
1    communal standpipe      functional
2    communal standpipe      functional
3    communal standpipe  non functional
4    communal standpipe      functional

[5 rows x 41 columns]

Data 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

Data Description:
                  id     amount_tsh    gps_height     longitude      latitude  \
count  59400.000000   59400.000000  59400.000000  59400.000000  5.940000e+04
mean   37115.131768     317.650385    668.297239     34.077427 -5.706033e+00
std    21453.128371    2997.574558    693.116350      6.567432  2.946019e+00
min        0.000000       0.000000    -90.000000      0.000000 -1.164944e+01
25%    18519.750000       0.000000      0.000000     33.090347 -8.540621e+00
50%    37061.500000       0.000000    369.000000     34.908743 -5.021597e+00
75%    55656.500000      20.000000   1319.250000     37.178387 -3.326156e+00
max    74247.000000  350000.000000   2770.000000     40.345193 -2.000000e-08

        num_private   region_code  district_code    population  \
count  59400.000000  59400.000000   59400.000000  59400.000000
mean       0.474141     15.297003       5.629747    179.909983
std       12.236230     17.587406       9.633649    471.482176
min        0.000000      1.000000       0.000000      0.000000
25%        0.000000      5.000000       2.000000      0.000000
50%        0.000000     12.000000       3.000000     25.000000
75%        0.000000     17.000000       5.000000    215.000000
max     1776.000000     99.000000      80.000000  30500.000000

       construction_year
count       59400.000000
mean         1300.652475
std           951.620547
min             0.000000
25%             0.000000
50%          1986.000000
75%          2004.000000
max          2013.000000

Data Types:
 object     31
int64       7
float64     3
dtype: int64

Value counts for num_private:
 0      58643
6         81
1         73
5         46
8         46
       ...
```

```
180        1
213        1
23         1
55         1
94         1
Name: num_private, Length: 65, dtype: int64

Value counts for management:
 vwc                40507
wug                 6515
water board         2933
wua                 2535
private operator    1971
parastatal          1768
water authority      904
other                844
company              685
unknown              561
other - school        99
trust                 78
Name: management, dtype: int64

Value counts for management_group:
 user-group    52490
commercial     3638
parastatal     1768
other           943
unknown         561
Name: management_group, dtype: int64

Value counts for scheme_management:
 VWC                36793
WUG                 5206
Water authority     3153
WUA                 2883
Water Board         2748
Parastatal          1680
Private operator    1063
Company             1061
Other                766
SWC                   97
Trust                 72
None                   1
Name: scheme_management, dtype: int64

Value counts for scheme_name:
 K                               682
None                            644
Borehole                        546
Chalinze wate                   405
M                               400
                               ...
Namahimba water gravity scheme    1
Kilimb                            1
Ngusero water supply              1
Mzizima Water                     1
Loronu water supply               1
Name: scheme_name, Length: 2696, dtype: int64

Value counts for payment_type:
 never pay     25348
```

```
per bucket     8985
monthly        8300
unknown        8157
on failure     3914
annually       3642
other          1054
Name: payment_type, dtype: int64

Value counts for payment:
 never pay                25348
pay per bucket            8985
pay monthly               8300
unknown                   8157
pay when scheme fails     3914
pay annually              3642
other                     1054
Name: payment, dtype: int64

Value counts for source:
 spring                  17021
shallow well             16824
machine dbh              11075
river                     9612
rainwater harvesting      2295
hand dtw                   874
lake                       765
dam                        656
other                      212
unknown                     66
Name: source, dtype: int64

Value counts for source_class:
 groundwater    45794
surface         13328
unknown           278
Name: source_class, dtype: int64
```

Observations

We can see all our columns and data types, we can also see that we have sum null or missing values

region_code and district_code seem like they should be categorical features.

construction_year should be casted as a datetime object.

id should be casted to object.

longitude and latitude look fine.

gps_height, amount_tsh and population also look fine.

There are 10 numeric features in the data and 31 string features in the data.

In [27]:

```
repetitive_and_unuseful = ['date_recorded', 'num_private', 'wpt_name', 'construc
                    'subvillage', 'region_code', 'district_code', 'lga',
                    'ward', 'public_meeting', 'recorded_by', 'scheme_management
                    'scheme_name', 'extraction_type', 'extraction_type_group',
```

```
                    'water_quality', 'source','source_type', 'waterpoint_type_
                   'payment_type', 'management', 'id', 'quantity_group']
```

In [28]:

```
print(f" There are {repetitive_and_unuseful.__len__()} columns that we do no need
```

```
There are 23 columns that we do no need in the data
```

After going through the variable description of the data and performing the preliminary data inspection, the study has proposed that the columns categorized as repetitive_and_unuseful be dropped on the basis that some provide similar information and some do not provide any relevant information, such as public_meeting and num_private.

DATA PREPARATION AND CLEANING

It is vital for data to be prepared before being staged for modelling to enhance the model's efficiency and prevent the generation of misleading knowledge. In this phase of the investigation, the study will look at missing values, duplicated entries, inconsistencies and invalid data.

In [33]:

```
class DataProcessor:
    def __init__(self, df):
        self.data = df.copy()
        self.repetitive_and_unuseful = ['date_recorded', 'num_private', 'wpt_nam
                                        'subvillage', 'region_code', 'district_
                                        'ward', 'public_meeting', 'recorded_by'
                                        'scheme_name', 'extraction_type', 'extra
                                        'water_quality', 'source','source_type'
                                        'payment_type', 'management', 'id', 'qu

    def drop_irrelevant_columns(self):
        self.data = self.data.drop(self.repetitive_and_unuseful, axis=1)

    def remove_duplicates(self, subset_columns):
        self.data = self.data.drop_duplicates(subset=subset_columns)

    def check_missing_values(self):
        return self.data.isna().sum()

    def percentage_missing_values(self):
        percentages = (self.data.isna().sum() / len(self.data)) * 100
        return percentages.sort_values(ascending=False)

    def process_data(self, subset_columns):
        self.drop_irrelevant_columns()
        self.remove_duplicates(subset_columns)

# Example usage
if __name__ == "__main__":
    # Assuming 'df' is your DataFrame
    processor = DataProcessor(df)
    subset_columns = ["latitude", "longitude"]
```

```
    # Process data
    processor.process_data(subset_columns)

    # Check missing values
    missing_values = processor.check_missing_values()
    print("Missing values:")
    print(missing_values)

    # Check percentage of missing values
    percentage_missing = processor.percentage_missing_values()
    print("\nPercentage of missing values:")
    print(percentage_missing)
```

```
Missing values:
amount_tsh                0
funder                 3609
gps_height                0
installer              3623
longitude                 0
latitude                  0
basin                     0
region                    0
population                0
permit                 3043
extraction_type_class     0
management_group          0
payment                   0
quality_group             0
quantity                  0
source_class              0
waterpoint_type           0
status_group              0
dtype: int64

Percentage of missing values:
installer              6.298679
funder                 6.274339
permit                 5.290334
status_group           0.000000
region                 0.000000
gps_height             0.000000
longitude              0.000000
latitude               0.000000
basin                  0.000000
population             0.000000
waterpoint_type        0.000000
extraction_type_class  0.000000
management_group       0.000000
payment                0.000000
quality_group          0.000000
quantity               0.000000
source_class           0.000000
amount_tsh             0.000000
dtype: float64
```

The list of irrelevant columns is encapsulated within the class, so we don't need to pass it as an argument every time we create an instance of DataProcessor

We dropped the irrelevant columns checked for duplicated and dropped the duplicates

We dropped the irrelevant columns,checked for duplicated and dropped the duplicates too

We also checked for missing values and the percentage of those missing values

The columns funder, installer and permit are the only features with missing values. In addition, the percentage of entries classified as missing values in funder and installer are small enough for us to drop those entries without sacrificing a big chunk of our data.

In [503…
```
#checking for unique values in installer, funder and permit
data['installer'].value_counts()
```

Out[503…
```
DWE                     16251
Government               1670
RWE                      1167
Commu                    1060
DANIDA                   1050
                        ...
Kwasenenge Group            1
Eastmeru medium School      1
St Gasper                   1
KKKT Katiti juu             1
kw                          1
Name: installer, Length: 2113, dtype: int64
```

In [504…
```
data['funder'].value_counts()
```

Out[504…
```
Government Of Tanzania    8834
Danida                    3114
Hesawa                    1914
World Bank                1345
Kkkt                      1287
                          ...
Meco                         1
Wafidhi Wa Ziwa T            1
Kalitesi                     1
Noshadi                      1
Juma                         1
Name: funder, Length: 1858, dtype: int64
```

In [505…
```
data['permit'].value_counts()
```

Out[505…
```
True     38055
False    16422
Name: permit, dtype: int64
```

this means that installer has 2113 unique values,funder has 1858 and permit has 2 unique values

In [506…
```
# creating an array of densities to impute missing values for permit
densities = list(data.permit.value_counts(normalize=True))

# previewing densities
densities
```

Out[506… [0.6985516823613636, 0.3014483176386365]

In [507…
```
#setting seed for reproducibility
np.random.seed(0)

# Defining a function to fill missing values for permit
def impute_missing(value):
    '''A function that fills missing values for permit'''
    if value not in [0,1]:
        return np.random.choice([0,1], p=densities)
    else:
        return value

# Applying the function to the column permit
data['permit'] = data['permit'].map(lambda x: impute_missing(x))

# #previewing the columns to check for missing values
for col in ['funder', 'installer', 'permit']:
    print('For', col,':')
    print('The densities are:', data[col].value_counts(normalize=True).\
        sort_values(ascending=False).head())
    print("Number of missing values are : {}".format(data[col].isna().sum()))
    print("-----------------------------")
```

```
For funder :
The densities are: Government Of Tanzania    0.163863
Danida                     0.057762
Hesawa                     0.035503
World Bank                 0.024949
Kkkt                       0.023873
Name: funder, dtype: float64
Number of missing values are : 3609
-----------------------------
For installer :
The densities are: DWE           0.301520
Government    0.030985
RWE           0.021652
Commu         0.019667
DANIDA        0.019482
Name: installer, dtype: float64
Number of missing values are : 3623
-----------------------------
For permit :
The densities are: True     0.678025
False    0.321975
Name: permit, dtype: float64
Number of missing values are : 0
-----------------------------
```

The missing values from permit have been imputed successfully using a user-defined function based on the probability densities of the feature. But the columns installer and funder still have missing values due to the fact that they are string values and the study thought it wise to remove those entries.

In [508…
```
# Dropping missing values from  installer and funder along the rows
data.dropna(axis=0, inplace=True)
```

```
# Checking once more for missing values
data.isna().sum()
```

Out[508…

```
amount_tsh               0
funder                   0
gps_height               0
installer                0
longitude                0
latitude                 0
basin                    0
region                   0
population               0
permit                   0
extraction_type_class    0
management_group         0
payment                  0
quality_group            0
quantity                 0
source_class             0
waterpoint_type          0
status_group             0
dtype: int64
```

Our data is clean there are no missing values.

outliers were not considered in the project as all the values in the numeric columns were thought of as important.

In [509…

```
# converting some string objects to categorical

for col in ['basin', 'region', 'extraction_type_class', 'management_group',
            'payment', 'quality_group', 'quantity', 'source_class',
            'waterpoint_type', 'status_group']:
    data[col] = data[col].astype('category')
data.dtypes
```

Out[509…

```
amount_tsh               float64
funder                    object
gps_height                 int64
installer                 object
longitude                float64
latitude                 float64
basin                   category
region                  category
population                 int64
permit                    object
extraction_type_class   category
management_group        category
payment                 category
quality_group           category
quantity                category
source_class            category
waterpoint_type         category
status_group            category
dtype: object
```

In [510…

```
# checking the data types of the columns
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53844 entries, 0 to 59399
Data columns (total 18 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   amount_tsh             53844 non-null  float64
 1   funder                 53844 non-null  object
 2   gps_height             53844 non-null  int64
 3   installer              53844 non-null  object
 4   longitude              53844 non-null  float64
 5   latitude               53844 non-null  float64
 6   basin                  53844 non-null  category
 7   region                 53844 non-null  category
 8   population             53844 non-null  int64
 9   permit                 53844 non-null  object
 10  extraction_type_class  53844 non-null  category
 11  management_group       53844 non-null  category
 12  payment                53844 non-null  category
 13  quality_group          53844 non-null  category
 14  quantity               53844 non-null  category
 15  source_class           53844 non-null  category
 16  waterpoint_type        53844 non-null  category
 17  status_group           53844 non-null  category
dtypes: category(10), float64(3), int64(2), object(3)
memory usage: 4.2+ MB
```

In [511…

```
# Saving cleaned data set for later use
data.to_csv('cleaned_data.csv')
```

Exploratory Data Analysis

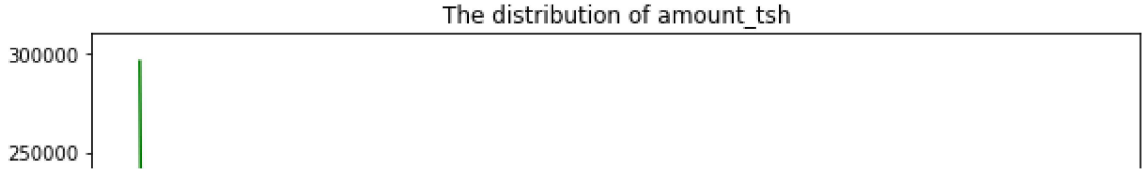
In this phase of the investigation, the study will look at the trends, patterns using visualizations and statistics to show the relationships between the variables within the data.

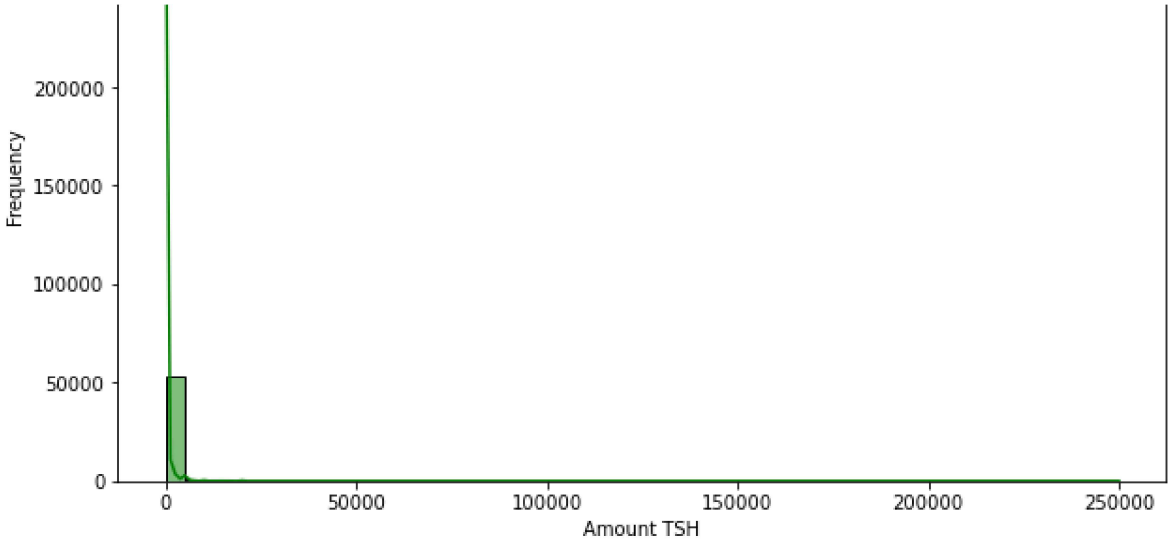
Univariate Analysis

Feature amount_tsh

In [512…

```
# Summary statistics of amount_tsh
data.amount_tsh.describe().T

# Visualizing the distribution of 'price' using a histogram
plt.figure(figsize=(10, 6))
sns.histplot(data.amount_tsh.T, bins=50, kde=True, color='green')
plt.title('The distribution of amount_tsh')
plt.xlabel('Amount TSH')
plt.ylabel('Frequency')
plt.show()
```





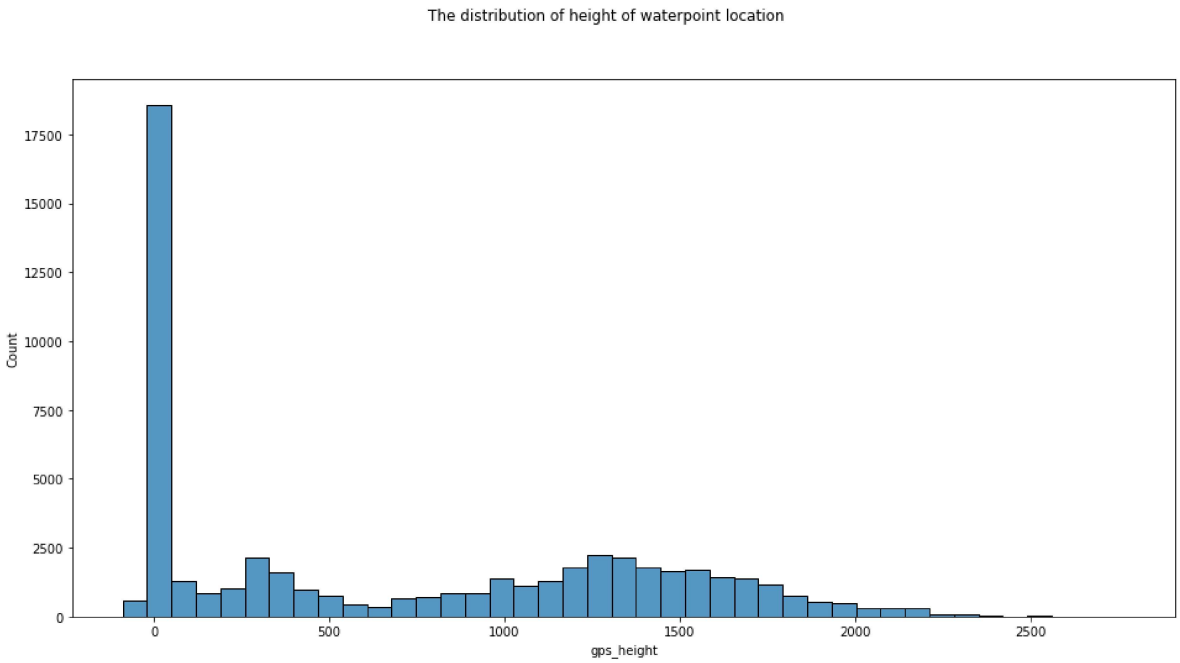
Observations

the data does not seem to be normally distributed

Feature gps_height

In [513…

```
# summary statistics of height in metres
data.gps_height.describe().T

# plotting the distribution of gps_height
plt.figure(figsize=(16,8))
sns.histplot(data.gps_height)
plt.suptitle("The distribution of height of waterpoint location")
plt.show()
```



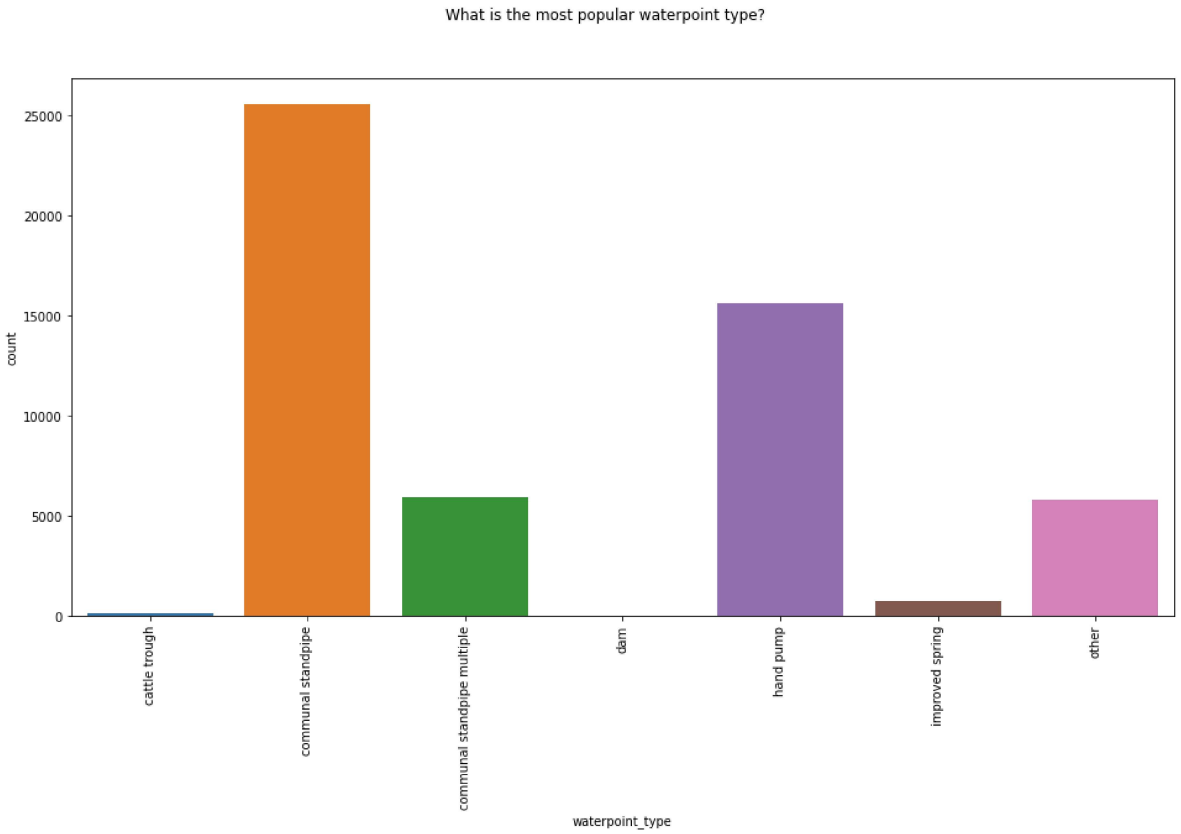
Observations:

The distribution of gps_height does not seem to be normally distributed

Feature waterpoint_type

In [514…

```
# These are the types of waterpoint:
data['waterpoint_type'].value_counts()

# plotting the most popular waterpoint type
plt.figure(figsize=(16,8))
sns.countplot(x='waterpoint_type', data=data)
plt.suptitle("What is the most popular waterpoint type?")
plt.xticks(rotation=90)
plt.show()
```



Observations:

The most featured waterpoint type is communal standpipe followed by handpump. The least featured is dam.

Feature population

In [515…

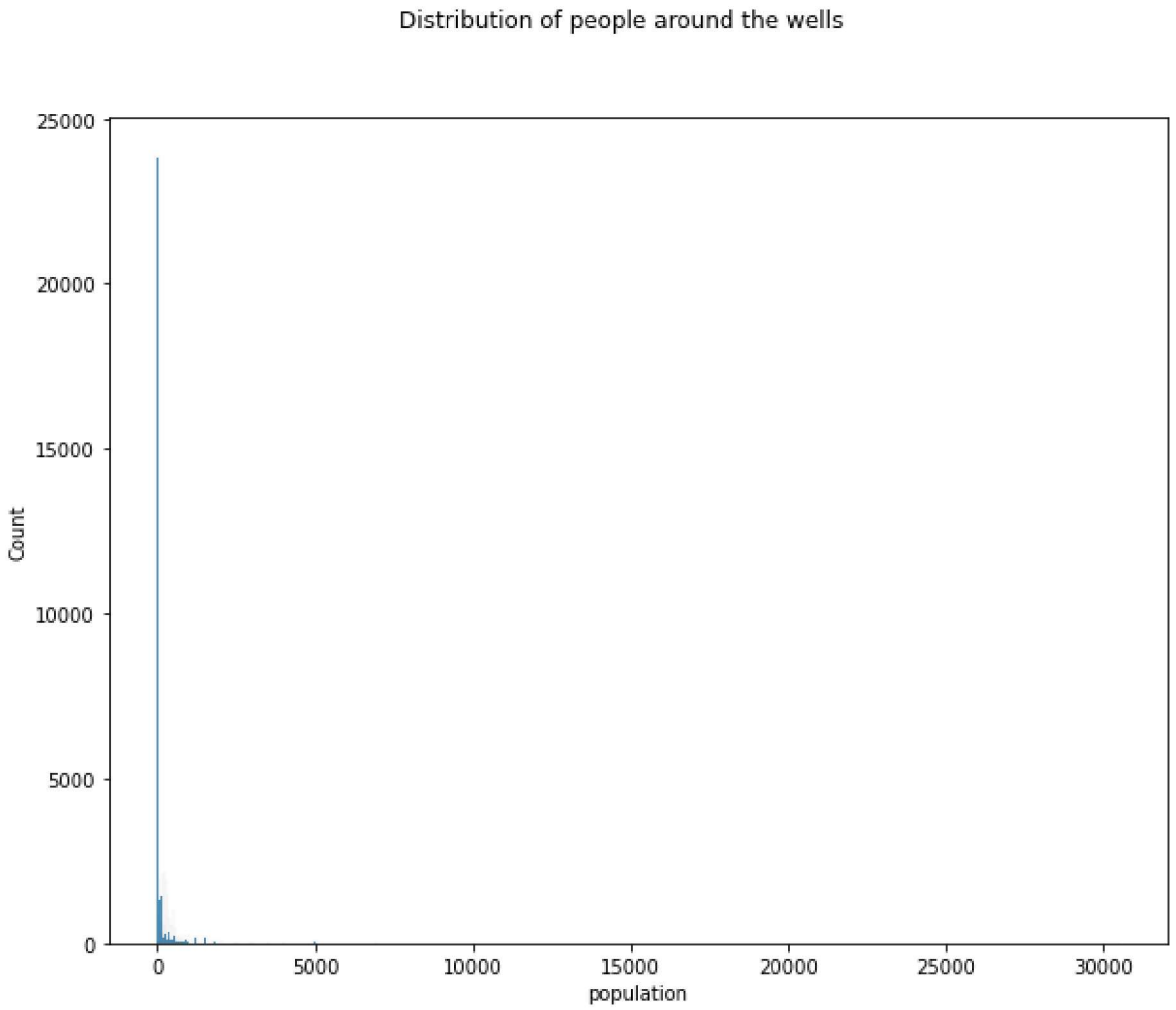
```
summary_statistics = data['population'].describe()
print("Total number of people around the wells is",summary_statistics[0])
print("\n")
print("The average number of people living around the wells is",summary_statistic
print("\n")
print("The minimum population value is",summary_statistics[3])
print("\n")
print("The maximum population value is",summary_statistics[7])
```

```
Total number of people around the wells is 53844.0
```

```
The average number of people living around the wells is 192.85413416536662

The minimum population value is 0.0

The maximum population value is 30500.0
```

In [516…

```
# plotting the distribution of population
plt.figure(figsize=(10,8))
sns.histplot(data["population"])
plt.suptitle("Distribution of people around the wells");
```



Observations

The number of people around the wells are not normally distributed.

Feature extraction_type_class

In [517…

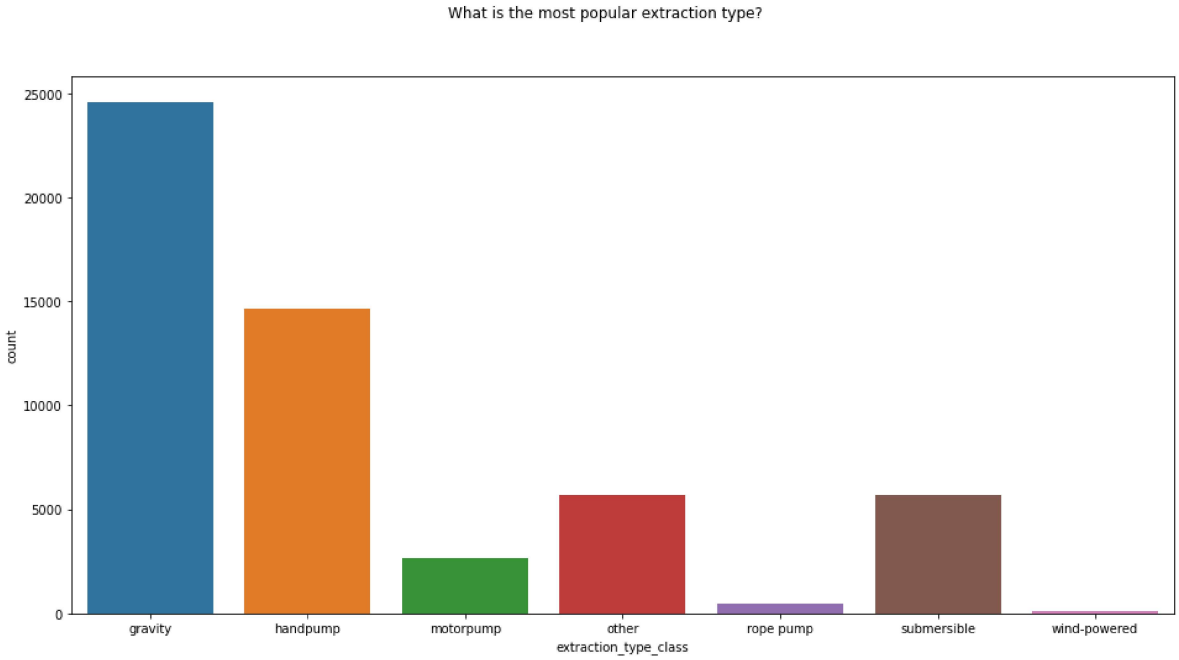
```
#getting the value counts
data['extraction_type_class'].value_counts()
```

Out[517… gravity 24589

```
handpump        14646
other            5720
submersible      5679
motorpump        2650
rope pump         448
wind-powered      112
Name: extraction_type_class, dtype: int64
```

In [518…

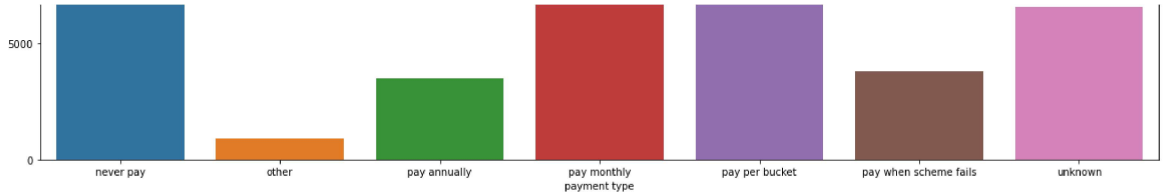
```
#analysing extraction type class  column
plt.figure(figsize=(16,8))
sns.countplot(x='extraction_type_class', data=data)
plt.suptitle("What is the most popular extraction type?")
plt.show()
```



Feature payment

In [519…

```
#plotting the variable payment
plt.figure(figsize=(20,10))
sns.countplot(x='payment', data=data)
plt.suptitle("The distribution of payment")
plt.xlabel('payment type')
plt.show()
```



Observations

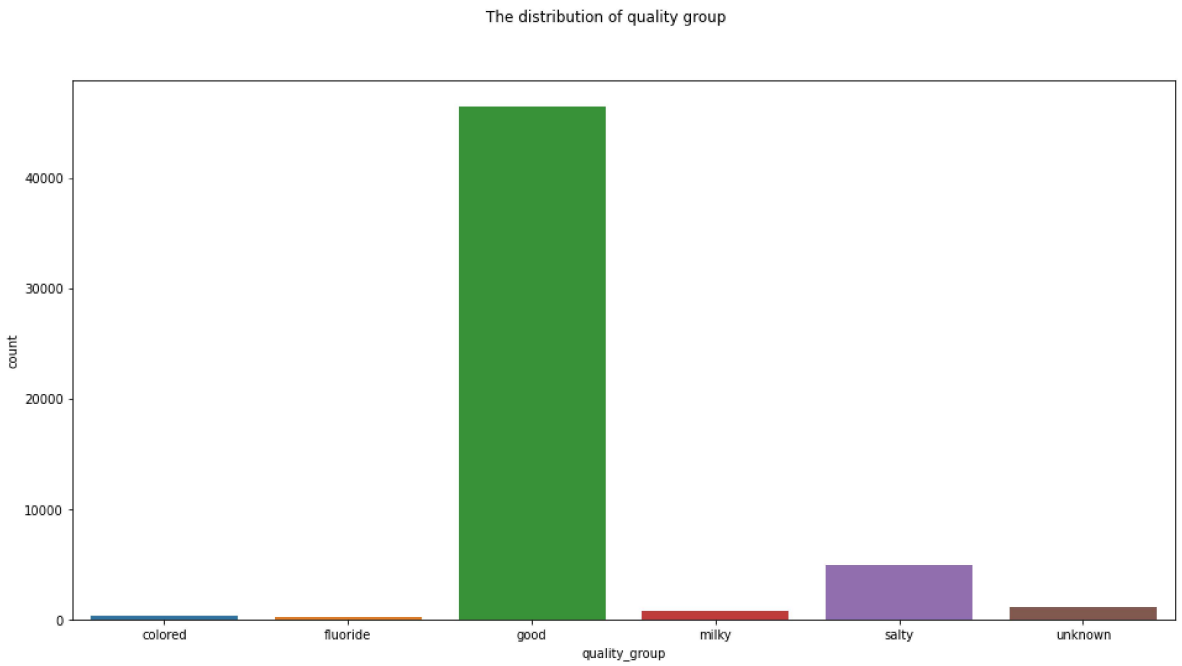
Wells that are not payed for feature the most in this data, followed by pay per bucket, unknown and pay monthly.

The least occuring are pay annually, pay when scheme fails and other.

Feature quality_group

In [520…

```
# plotting the distribution of quality group
plt.figure(figsize=(16,8))
sns.countplot(x='quality_group', data=data)
plt.suptitle("The distribution of quality group")
plt.show()
```



Observations:

The most featured quality group is good followed by salty and the least is fluoride.

Feature region

In [521…

```
# Count of number of wells per region
region_dict = dict(zip(list(data.region.value_counts().index),
                       list(data.region.value_counts().values)))
region_dict
```

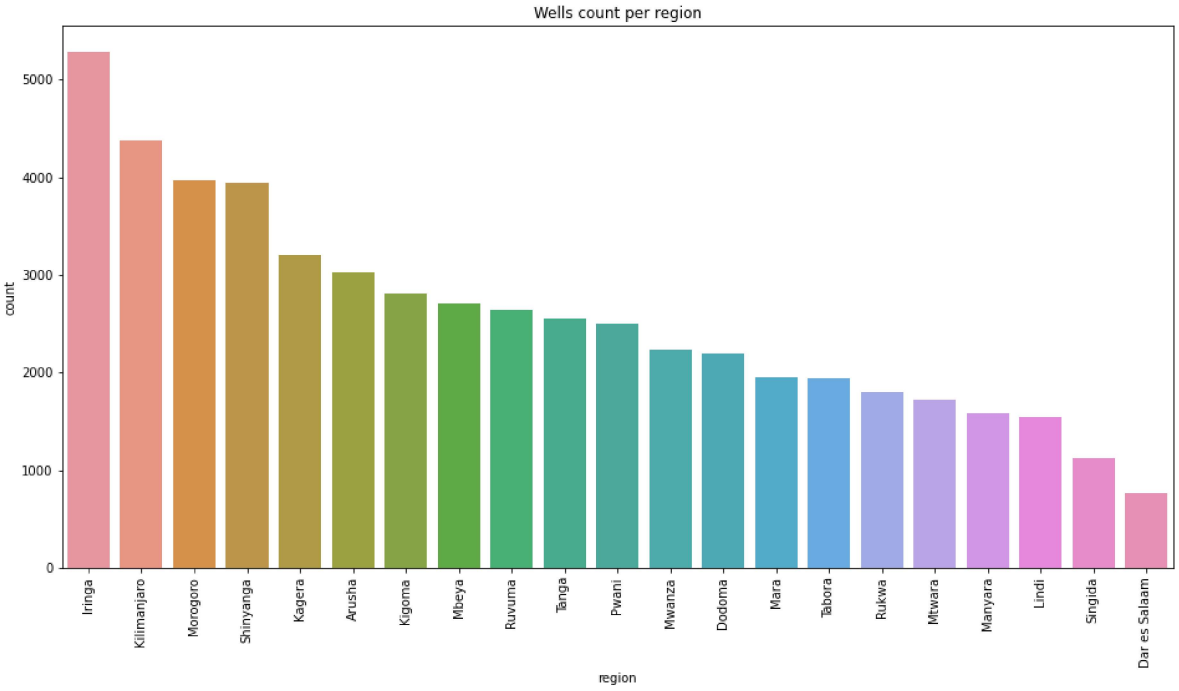
Out[521…

```
{'Iringa': 5284,
```

```
'Kilimanjaro': 4373,
'Morogoro': 3972,
'Shinyanga': 3940,
'Kagera': 3205,
'Arusha': 3024,
'Kigoma': 2805,
'Mbeya': 2703,
'Ruvuma': 2638,
'Tanga': 2546,
'Pwani': 2497,
'Mwanza': 2228,
'Dodoma': 2196,
'Mara': 1953,
'Tabora': 1940,
'Rukwa': 1803,
'Mtwara': 1725,
'Manyara': 1580,
'Lindi': 1542,
'Singida': 1124,
'Dar es Salaam': 766}
```

In [522…

```
# Analysing region column
plt.figure(figsize=(16,8))
sns.countplot(x = data.region,
              order = data['region'].value_counts().index).set(title="Wells coun
plt.xticks(rotation=90);
```



Observations:

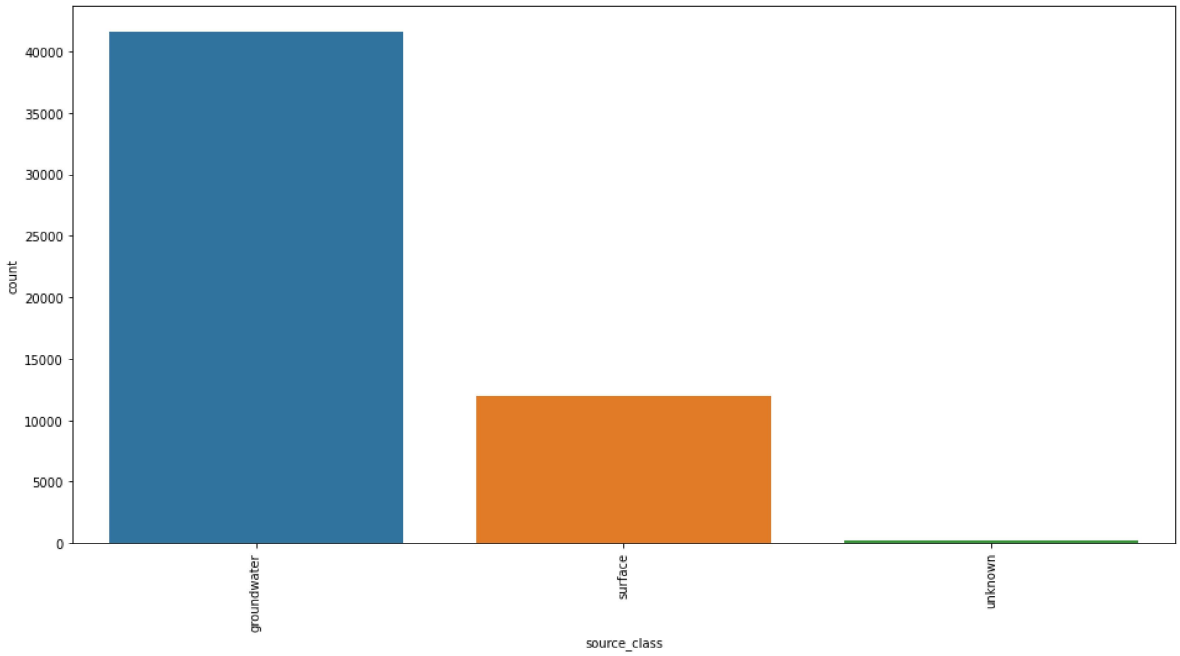
From the analysis, Iringa region has the greatest number of wells, while Dar es Salaam has the least number.

Feature source_class

In [523…

```
# Look at the unique values and their count
```

```
data.source_class.value_counts()

# Visualize the distribution using a count plot
plt.figure(figsize=(16,8))
sns.countplot(x= 'source_class', data= data, order=data.source_class.value_counts
              .index).set_xticklabels(data.source_class.value_counts()
              .index, rotation=90);
```



Observation

Most waterpoints get their water from ground water sources according to the data

BIVARIATE ANALYSIS

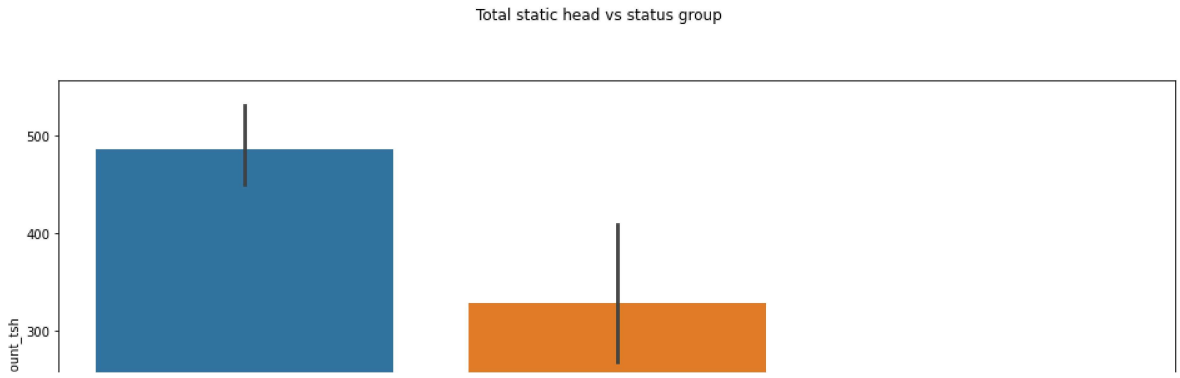
What is the relationship between amount_tsh and status_group

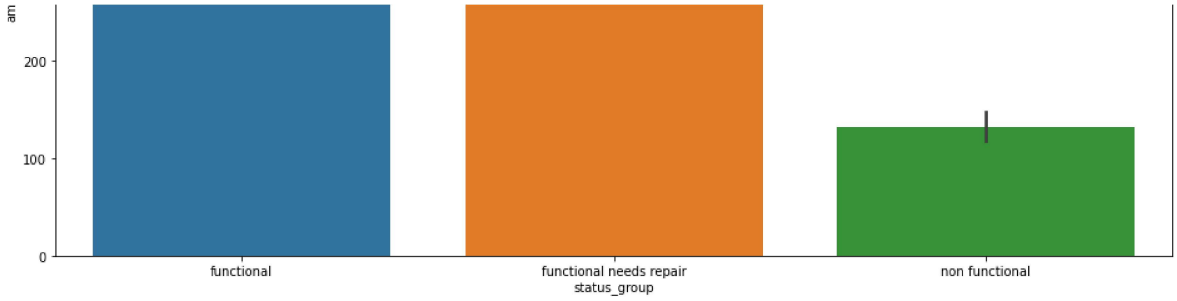
In [524…

```
# plotting the distribution of amount by status group
plt.figure(figsize=(16,8))
sns.barplot(x='status_group', y ='amount_tsh',
                data=data)
plt.suptitle("Total static head vs status group")
```

Out[524…

```
Text(0.5, 0.98, 'Total static head vs status group')
```





Observations:

Functional water points have the highest amount of water available followed by those that are functional but need repair.

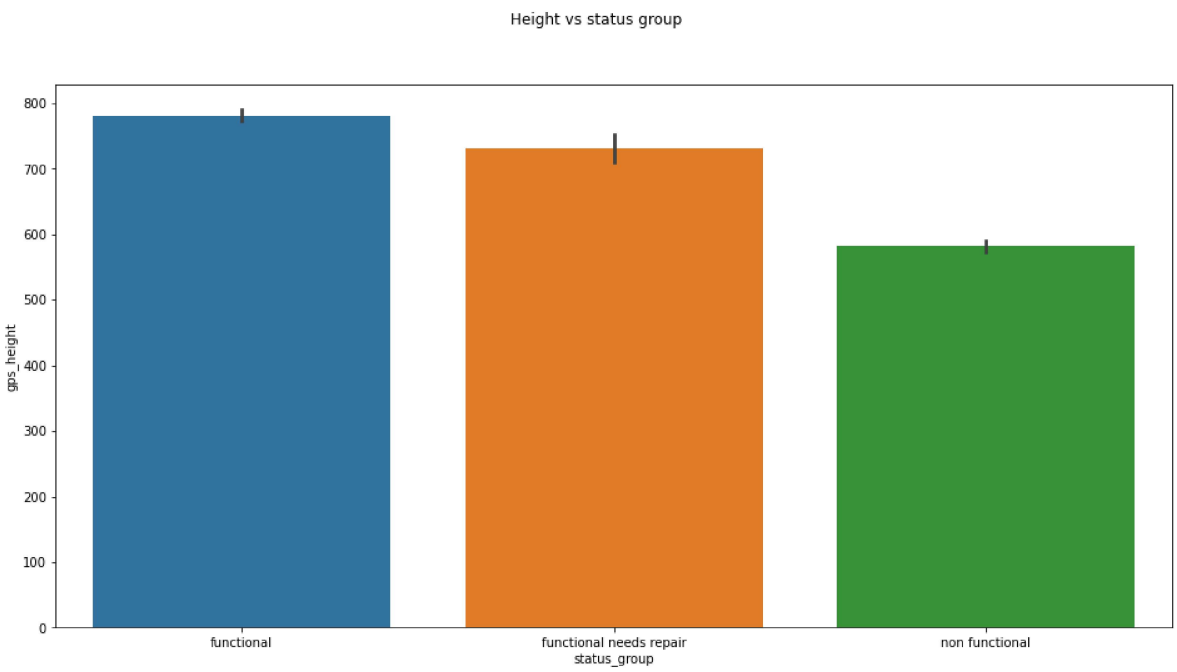
Non functional water points had the least amount of water available

What is the relationship between gps_height and status_group?

In [525…

```
# Plotting the relationship between gps_height and status_group
plt.figure(figsize=(16,8))
sns.barplot(x='status_group', y ='gps_height',
                data=data)
plt.suptitle("Height vs status group")
```

Out[525… `Text(0.5, 0.98, 'Height vs status group')`



Observations:

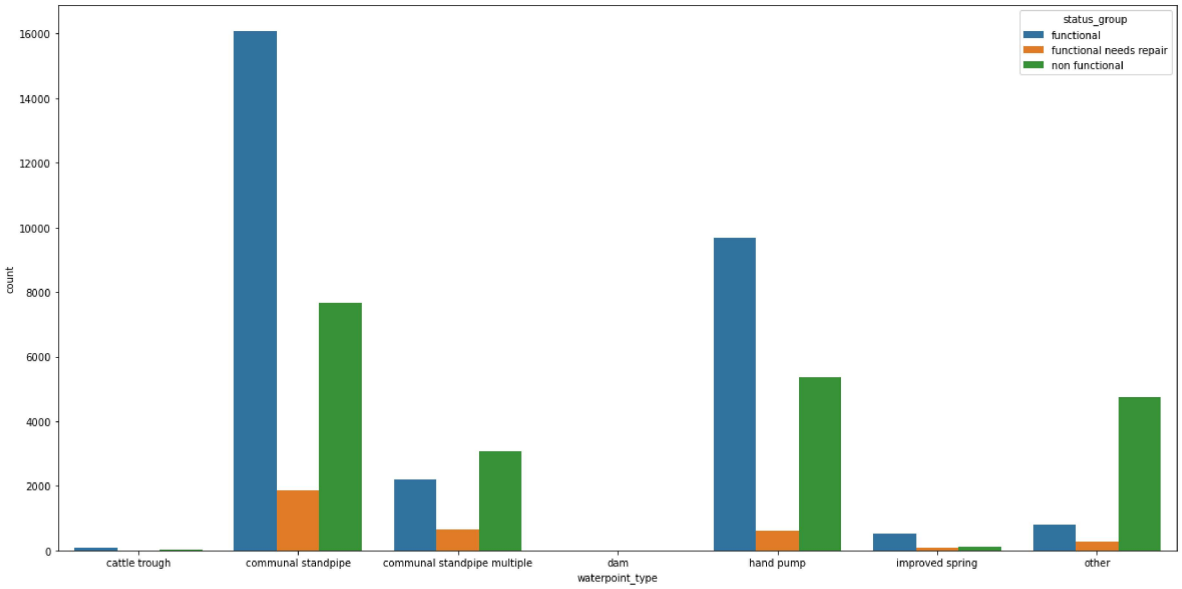
Functional waterpoints are generally located at higher altitudes than the rest.

What is the relationship between waterpoint_type and status_group?

In [526…

```
#comparing the water point type with the target varible.
plt.figure(figsize=(20,10))
```

```
plt.figure(figsize=(20,10))
sns.countplot(x='waterpoint_type', hue="status_group", data=data)
plt.show()
```



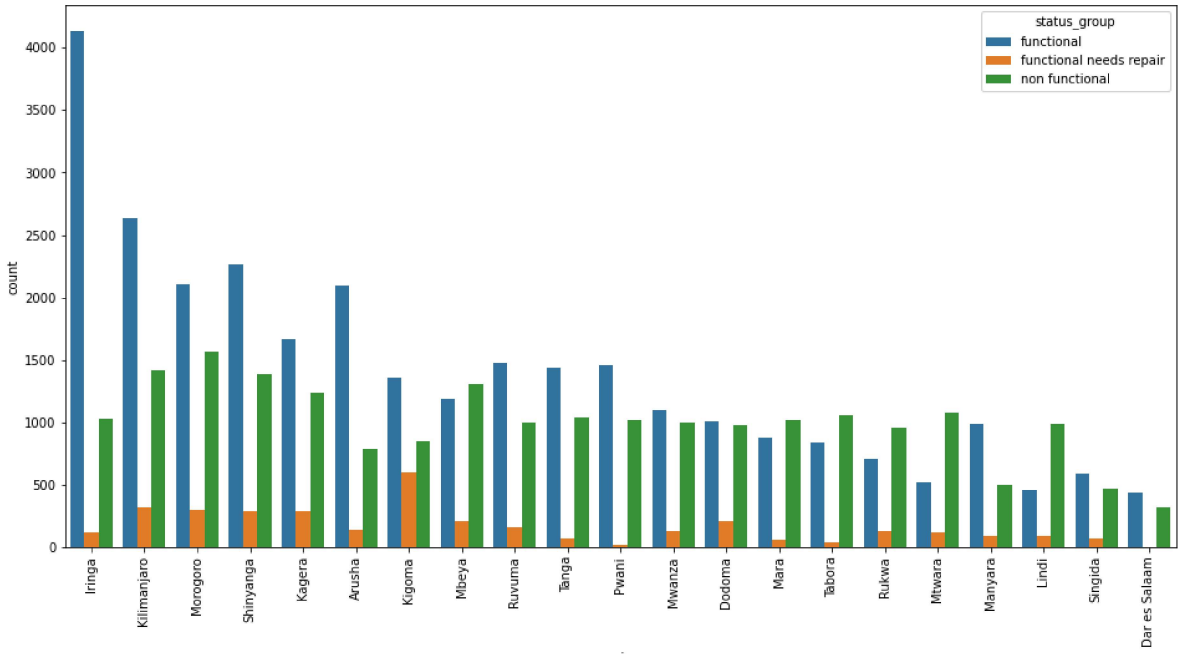
Observations

Communal standpipe has more count and functional water points are generally more than the others in except communal standpipe and other. This may be due to the fact that is is the most popular water point.

What is the relationship between region and status_group?

In [527…

```
# count of functional and non-functional wells based on the basin
fig = plt.subplots(figsize = (16,8))
sns.countplot(x= 'region', hue= 'status_group', data= data,
              order=data.region.value_counts()\
              .index).set_xticklabels(data.region.value_counts()
               .index, rotation=90);
```



region

Data preprocessing

we are going to prepare the data for machine learning. The data have to be normalized to change the values of numerical columns ensuring they are of a common scale to prevent features with large values from dominating those with smaller values giving the features have the same influence on the measurement metric.

In [528…

```
# Loading the cleaned data
data = pd.read_csv("cleaned_data.csv",index_col = 0)
```

In [529…

```
# coping the data
data_2 = data.copy()
# Dropping columns not necessary for modelling
data_2.drop(['funder',"installer","longitude","latitude"],
            axis = 1,inplace = True)
data_2.head()
```

Out[529…

	amount_tsh	gps_height	basin	region	population	permit	extraction_type_class
0	6000.0	1390	Lake Nyasa	Iringa	109	False	gravity
1	0.0	1399	Lake Victoria	Mara	280	True	gravity
2	25.0	686	Pangani	Manyara	250	True	gravity
3	0.0	263	Ruvuma / Southern Coast	Mtwara	58	True	submersible
4	0.0	0	Lake Victoria	Kagera	0	True	gravity

In [530…

```
# Importing libraries needed
# For Scaling
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

We decided to split the data into testing and training set so that the machine learning algorithms can learn from the training set and evaluate its performance using the testing set.

In [531…

```
# Determining the predictor variables
X = data_2.drop(["status_group"],axis = 1)
```

```
# Determining the target variables

y = data_2.status_group

# Splitting the data to training set and testing set
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size =0.3,random_st
```

In [532…

```
# Checking the shape of the split data
print(f'X_train has {X_train.shape[0]} rows and {X_train.shape[1]} columns')
print(f'X_test has {X_test.shape[0]} rows and {X_test.shape[1]} columns')
print(f'y_train has {y_train.shape[0]} rows')
print(f'y_test has {y_test.shape[0]} rows')
```

```
X_train has 37690 rows and 13 columns
X_test has 16154 rows and 13 columns
y_train has 37690 rows
y_test has 16154 rows
```

We've chosen to transform the categorical data into numerical format using the pandas get_dummies method because sklearn's preprocessing modules, like OneHotEncoder, expect numeric input data.

In [533…

```
# Converting categorical variable into dummy variables for X_train
X_train_cat = pd.get_dummies(X_train.select_dtypes(include = ["object"]))

# Converting categorical variable into dummy variables for X_test

X_test_cat = pd.get_dummies( X_test.select_dtypes(include = ["object"]))
```

In [534…

```
# Concatenating Numerical values with dummy variables for X_train
X_train_2 = pd.concat([X_train.select_dtypes(exclude = ["object"]),X_train_cat,]

# Concatenating Numerical values with dummy variables for X_test
X_test_2 = pd.concat([X_test.select_dtypes(exclude = ["object"]),X_test_cat],axi
```

Modeling

We want to determine the most suitable machine learning model that will effectively recognize patterns of the wells and give predictions.

Models used:

Decision trees

Random forest

XG boost

DECISION TREE MODEL

Why use decision trees?

Because decision trees are versatile models that offer a balance between interpretability, flexibility, and predictive performance.

In [535…

```
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import accuracy_score
```

In [536…

```
# Define preprocessing steps
numeric_features = X_train.select_dtypes(include=['number']).columns
categorical_features = X_train.select_dtypes(include=['object']).columns

numeric_transformer = SimpleImputer(strategy='mean')
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

In [537…

```
# Create pipeline with preprocessing and the decision tree classifier
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', DecisionTreeClassifier(random_state=42))
])

# Fit the pipeline on the training data
pipeline.fit(X_train, y_train)

# Make predictions on training and testing data
y_train_pred = pipeline.predict(X_train)
y_test_pred = pipeline.predict(X_test)

# Calculate training and testing accuracies
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

print('Training Accuracy: {:.4}%'.format(train_accuracy * 100))
print('Testing Accuracy: {:.4}%'.format(test_accuracy * 100))
```

```
Training Accuracy: 94.08%
Testing Accuracy: 74.01%
```

In [538…

```
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Model Evaluation
print("Training Classification Report:")
```

```
print(classification_report(y_train, y_train_pred))

print("Testing Classification Report:")
print(classification_report(y_test, y_test_pred))
```

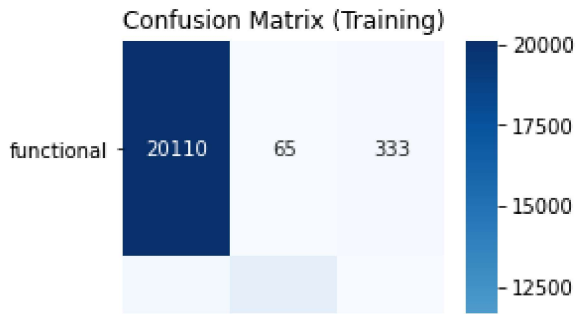
```
Training Classification Report:
                         precision    recall  f1-score   support

             functional       0.92      0.98      0.95     20508
functional needs repair       0.93      0.78      0.85      2435
         non functional       0.97      0.91      0.94     14747

               accuracy                           0.94     37690
              macro avg       0.94      0.89      0.91     37690
           weighted avg       0.94      0.94      0.94     37690

Testing Classification Report:
                         precision    recall  f1-score   support

             functional       0.77      0.82      0.79      8839
functional needs repair       0.31      0.27      0.29      1056
         non functional       0.76      0.71      0.73      6259

               accuracy                           0.74     16154
              macro avg       0.61      0.60      0.61     16154
           weighted avg       0.74      0.74      0.74     16154
```

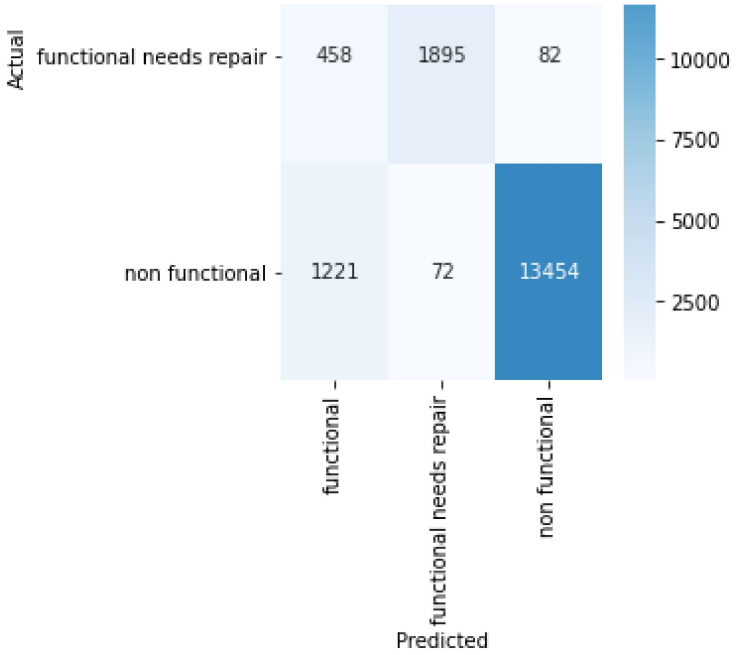
The model achieves high precision, recall, and F1-score for each class, indicating good performance on the training data with 94% suggesting that the model performs well overall. The model's performance decreases slightly on the testing data compared to the training data. The accuracy on the testing data is 73%, which is lower than the training accuracy indicating some degree of overfitting.

In [539…

```
# Confusion Matrix
plt.figure(figsize=(8, 6))
cm_train = confusion_matrix(y_train, y_train_pred)
cm_test = confusion_matrix(y_test, y_test_pred)

plt.subplot(1, 2, 1)
sns.heatmap(cm_train, annot=True, cmap='Blues', fmt='d', xticklabels=pipeline.cla
plt.title('Confusion Matrix (Training)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

Out[539… Text(50.99999999999999, 0.5, 'Actual')



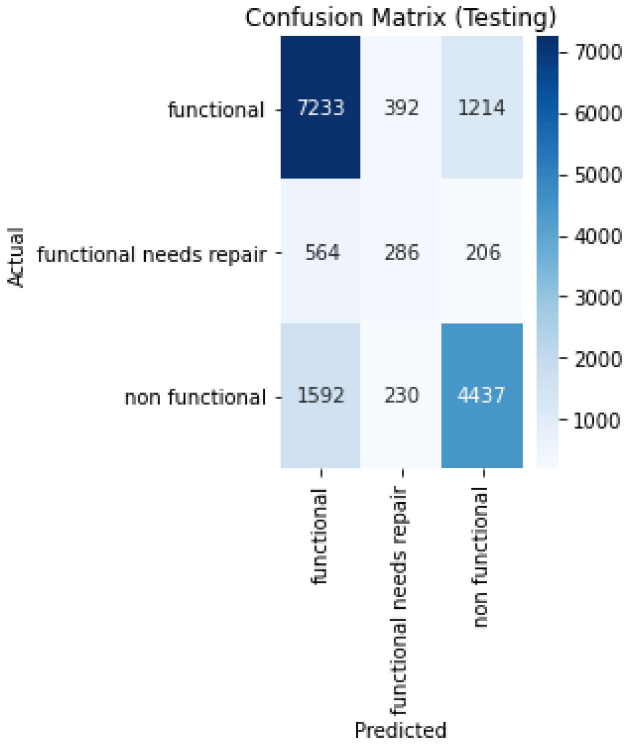


In [540…

```
plt.subplot(1, 2, 2)
sns.heatmap(cm_test, annot=True, cmap='Blues', fmt='d', xticklabels=pipeline.cla
plt.title('Confusion Matrix (Testing)')
plt.xlabel('Predicted')
plt.ylabel('Actual')
```

Out[540…

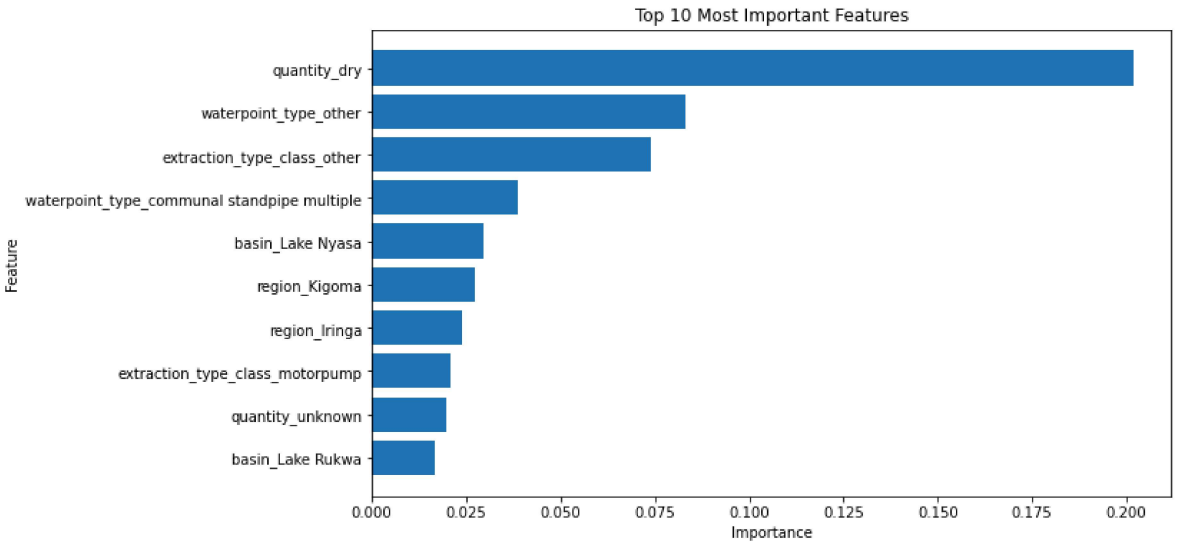
```
Text(215.61818181818177, 0.5, 'Actual')
```



In [541…

```
# Sort feature importance scores
sorted_idx = np.argsort(feature_importance)[::-1]
top_n = 10  # Select top N features to display
```

```
# Plot top N features
plt.figure(figsize=(10, 6))
plt.barh(np.arange(top_n), feature_importance[sorted_idx][:top_n], align='center
plt.yticks(np.arange(top_n), np.array(feature_names)[sorted_idx][:top_n])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Top {} Most Important Features'.format(top_n))
plt.gca().invert_yaxis()  # Invert y-axis to display highest importance at the to
plt.show()
```



In feature importance we sorted the indices in descending order from highest to lowest the first top 10.The aim of feature importance is to identify and quantify the relative importance of each feature in a predictive model. Feature importance provides insights into which features have the most significant influence on the model's predictions. Understanding feature importance is crucial for several reasons:

RANDOM FOREST CLASSIFIER

In [542…

```
# Importing RandomForestClassifier library
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matr
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import cross_val_score
```

In [543…

```
# instantiating RandomForestClassifier
RF = RandomForestClassifier()
# Fitting RandomForestClassifier
RF.fit(X_train_resampled, y_train_resampled)

# Predict on training and test sets
RF_training_preds = RF.predict(X_train_resampled)
RF_test_preds = RF.predict(X_test_scaled)
```

```
# Accuracy of training and test sets
RF_training_accuracy = accuracy_score(y_train_resampled, RF_training_preds)
RF_test_accuracy = accuracy_score(y_test_2, RF_test_preds)

print('Training Accuracy: {:.4}%'.format(RF_training_accuracy * 100))
print('Validation accuracy: {:.4}%'.format(RF_test_accuracy * 100))
```

```
Training Accuracy: 93.95%
Validation accuracy: 73.94%
```

This Random forest baseline model is overfitting the training data so we are going to do a grid search to get the best hyperparameters that would reduce overfitting.

In [544…

```
# Creating a parameter grid
param_grid = {
    'n_estimators': [100, 200],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6],
    'criterion' :['gini', 'entropy']}
```

In [545…

```
# Check if RF is properly defined
print(type(RF))

# Check the param_grid dictionary
print(param_grid)

# Check dimensions and data types of X_train_resampled and y_train_resampled
print(X_train_resampled.shape, y_train_resampled.shape)
print(type(X_train_resampled), type(y_train_resampled))
```

```
<class 'sklearn.ensemble._forest.RandomForestClassifier'>
{'n_estimators': [100, 200], 'max_features': ['auto', 'sqrt', 'log2'], 'max_dept
h': [4, 5, 6], 'criterion': ['gini', 'entropy']}
(61524, 77) (61524,)
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

In [552…

```
# Create GridSearchCV object
RF_CV = GridSearchCV(estimator=RF, param_grid=param_grid, cv=5)
# Fit the model to the resampled training data
RF_CV.fit(X_train_resampled, y_train_resampled)
```

Out[552…

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [4, 5, 6],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_estimators': [100, 200]})
```

In [553…

```
# Getting the best hyperparameters for our RandomForestClassifier
RF_CV.best_params_
```

Out[553…

```
{'criterion': 'entropy',
 'max_depth': 6,
 'max_features': 'sqrt',
 'n_estimators': 100}
```

Iterated random forest

In [ ]:
```
# Instantiating RandomForestClassifier model using the best parameters from the g
best_forest = RandomForestClassifier(criterion = 'entropy',max_depth = 6, max_fea
                                     n_estimators = 200)
```

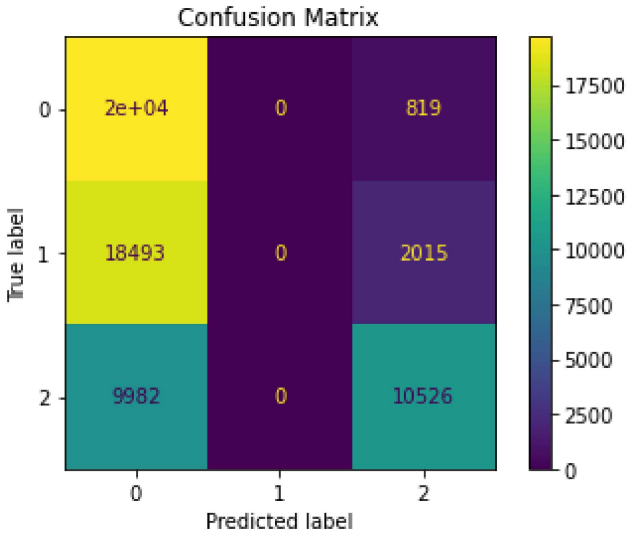
In [ ]:
```
# Fitting the model
best_forest.fit(X_train_scaled,y_train_2)
```

Out[ ]:
```
RandomForestClassifier(criterion='entropy', max_depth=6, max_features='sqrt',
                       n_estimators=200)
```

Evaluating random forest model

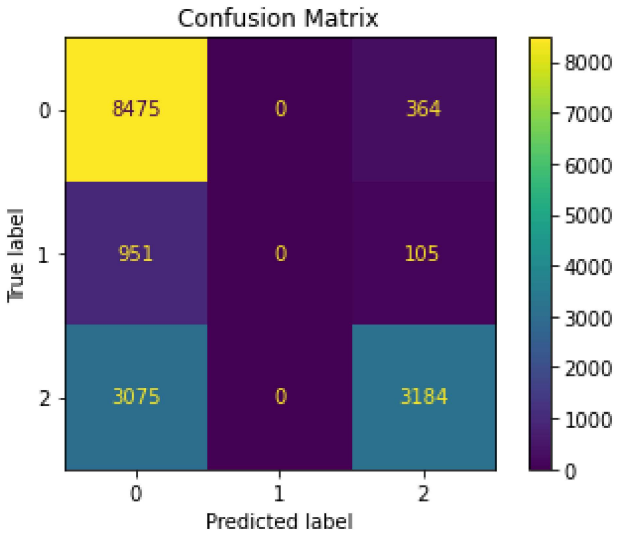
In [ ]:
```
# Confusion matrix for trainig data
plot_confusion_matrix(best_forest, X_train_resampled, y_train_resampled)
plt.title('Confusion Matrix')
plt.show()
```

```
c:\Users\user\anaconda3\envs\learn-env\lib\site-packages\sklearn\utils\deprecatio
n.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `pl
ot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of t
he class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDispla
y.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



In [ ]:
```
# Confusion matrix for testing data
plot_confusion_matrix(best_forest, X_test_scaled, y_test_2)
plt.title('Confusion Matrix')
plt.show()
```

```
c:\Users\user\anaconda3\envs\learn-env\lib\site-packages\sklearn\utils\deprecatio
n.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `pl
ot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of t
he class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDispla
y.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



In [ ]:

```
# Predict on training and test sets
best_forest_training_preds = best_forest.predict(X_train_resampled)
best_forest_preds = best_forest.predict(X_test_scaled)


# Accuracy of training and test sets
best_forest_training_accuracy = accuracy_score(y_train_resampled, best_forest_tra
best_forest_test_accuracy = accuracy_score(y_test_2, best_forest_preds)

print('Training Accuracy: {:.4}%'.format(best_forest_training_accuracy * 100))
print('Validation accuracy: {:.4}%'.format(best_forest_test_accuracy * 100))
```

```
Training Accuracy: 49.11%
Validation accuracy: 72.17%
```

In [ ]:

```
# validate RandomForest regressor

#train cross validation
best_forest_train_cv_score = cross_val_score(best_forest, X_train_resampled,
                                             y_train_resampled, cv = 5, n_jobs = -1).
print('Train cross validation:', best_forest_train_cv_score)

#test cross validation
best_forest_test_cv_score = cross_val_score(best_forest, X_test_scaled, y_test_2
                                            cv = 5, n_jobs = -1).mean()
print('Test cross validation:', best_forest_test_cv_score)
```

```
Train cross validation: 0.649551348236653
Test cross validation: 0.7211833696973878
```

The hyperparameter tuned random forest model seems to underfit the training data therefore we will use another machine learning algorithm.

EXTREME GRADIENT BOOSTING (XG BOOST)

We used xgboost because of it's versatile nature and powerful algorith that offers superior performance, scalability, and flexibility.

In [ ]:
```
#import the necessary libraries
import xgboost as xgb
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, r
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

In [ ]:
```
# Define and train XGBoost model
xgb_model = xgb.XGBClassifier(random_state=42)
xgb_model.fit(X_train_2, y_train)

# Make predictions on training and testing data
y_train_pred = xgb_model.predict(X_train_2)
y_test_pred = xgb_model.predict(X_test_2)

# Calculate accuracy on training and testing data
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

print('Training Accuracy: {:.4}%'.format(train_accuracy * 100))
print('Testing Accuracy: {:.4}%'.format(test_accuracy * 100))
```

```
Training Accuracy: 81.82%
Testing Accuracy: 76.91%
```

The training accuracy is slightly higher than the testing accuracy, indicating a small degree of overfitting. However, the difference between the training and testing accuracies is not substantial, suggesting that the overfitting is relatively minor.

Thus we will try to tune our hyperparameter to see if we can meet our target range.

In [ ]:
```
#  Hyperparameter Tuning
param_grid = {
    'max_depth': [3, 5],
    'learning_rate': [0.1, 0.01],
    'n_estimators': [100, 200],
    'min_child_weight': [1, 3, ]
}
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=5, n_j
grid_search.fit(X_train_2, y_train)
best_params = grid_search.best_params_
```

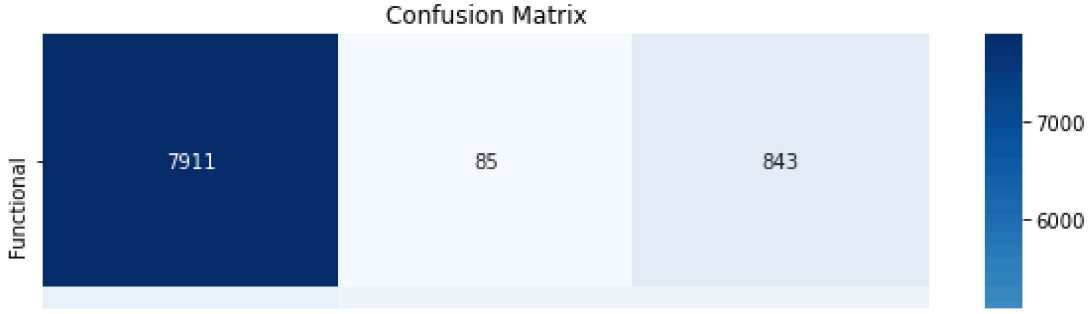
In [ ]:
```
#  Cross-Validation
cv_scores = cross_val_score(xgb_model, X_train_2, y_train, cv=5)
mean_cv_score = cv_scores.mean()

#Get feature importance scores from the trained XGBoost model
feature_importance = grid_search.best_estimator_.feature_importances_
```

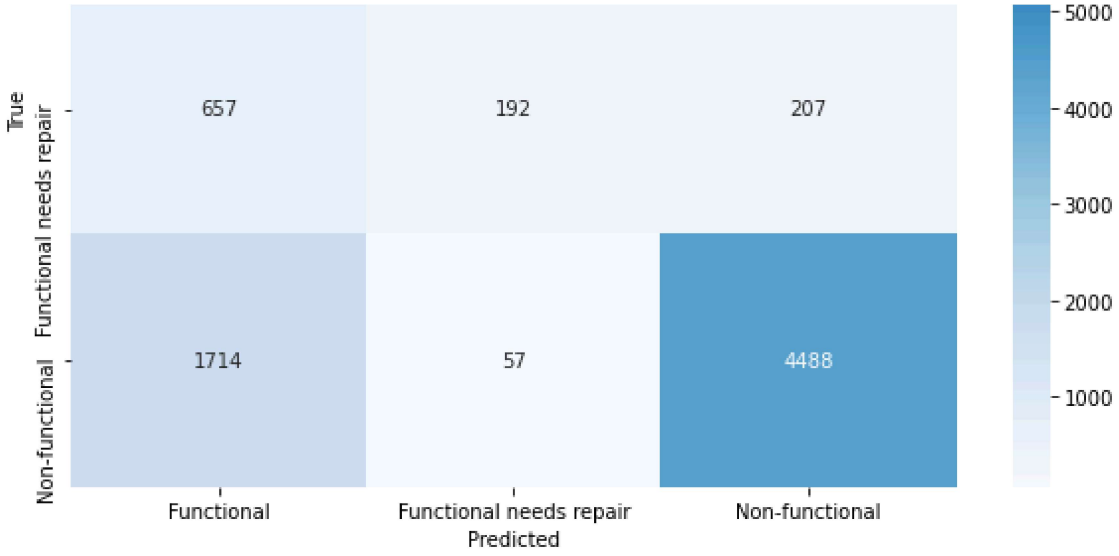
In [ ]:
```
# Create a VotingClassifier with XGBoost and RandomForestClassifier
voting_clf = VotingClassifier(estimators=[('xgb', grid_search.best_estimator_),
voting_clf.fit(X_train_2, y_train)

#  Error Analysis
y_pred = voting_clf.predict(X_test_2)
error_report = classification_report(y_test, y_pred)
```

In [ ]:
```
# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Calculate accuracy score
accuracy = accuracy_score(y_test, y_pred)

# Calculate precision, recall, and F1 score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print("Confusion Matrix:\n", conf_matrix)
print("Accuracy Score:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Confusion Matrix:
 [[7911   85  843]
 [ 657  192  207]
 [1714   57 4488]]
Accuracy Score: 0.7794354339482481
Precision: 0.7725698058121772
Recall: 0.7794354339482481
F1 Score: 0.7656311790688866
```

In [ ]:
```
# Define the class labels
class_names = ['Functional', 'Functional needs repair', 'Non-functional']

# Plot the confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=class_nar
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```





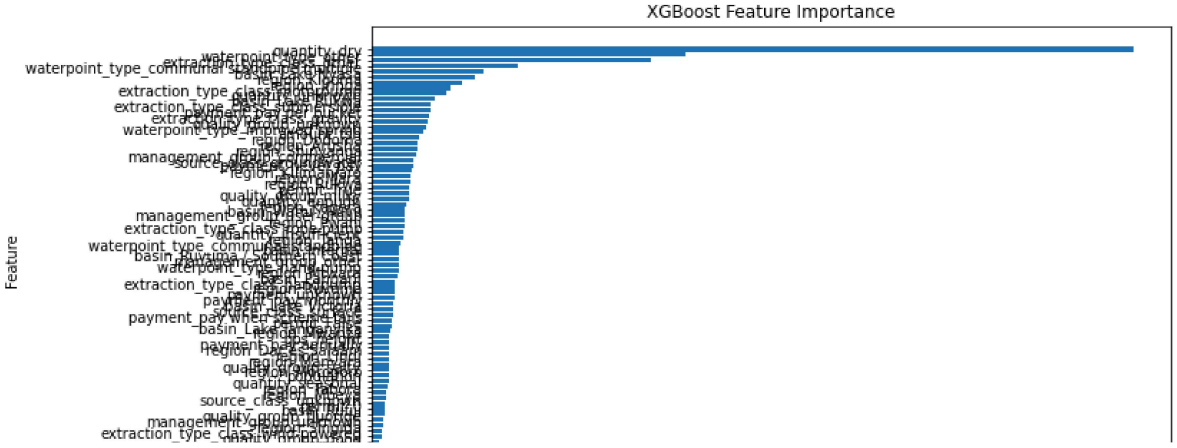
Based on the evaluation metrics provided, the model demonstrates reasonable performance in classifying the water pump functionality.

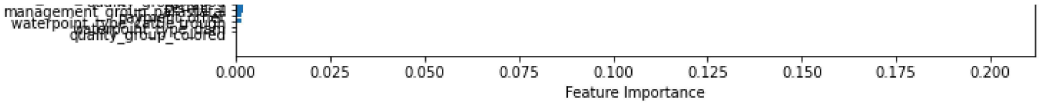
The model achieves an accuracy score of approximately 77.94%, indicating that it correctly predicts the class for about 77.94% of the instances in the test dataset.

PLOTTING THE FEATURE IMPORTANCE GRAPH

In [ ]:

```
import matplotlib.pyplot as plt
#Get feature names
feature_names = X_train_2.columns

# Sort feature importance scores and corresponding feature names
sorted_idx = feature_importance.argsort()
sorted_feature_importance = feature_importance[sorted_idx]
sorted_feature_names = feature_names[sorted_idx]

# Create a bar plot of feature importances
plt.figure(figsize=(10, 6))
plt.barh(range(len(sorted_idx)), sorted_feature_importance, align='center')
plt.yticks(range(len(sorted_idx)), sorted_feature_names)
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('XGBoost Feature Importance')
plt.show()
```

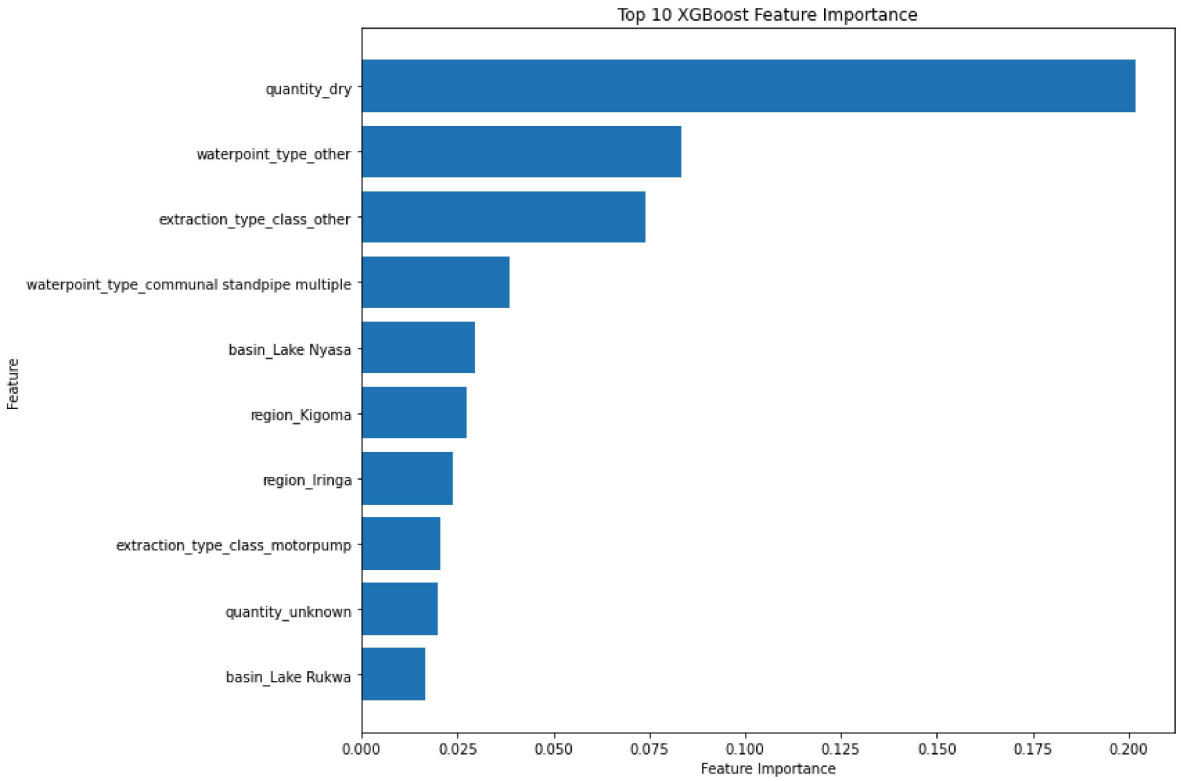




We created another graph containing the top 10 as this graph was unreadable

In [ ]:

```
# Select top N features
N = 10
top_features = sorted_feature_names[-N:]
top_importance = sorted_feature_importance[-N:]

# Create a bar plot of top N feature importances
plt.figure(figsize=(12, 8))
plt.barh(range(N), top_importance, align='center')
plt.yticks(range(N), top_features)
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Top {} XGBoost Feature Importance'.format(N))
plt.tight_layout()  # Adjust layout to prevent overlapping
plt.show()
```



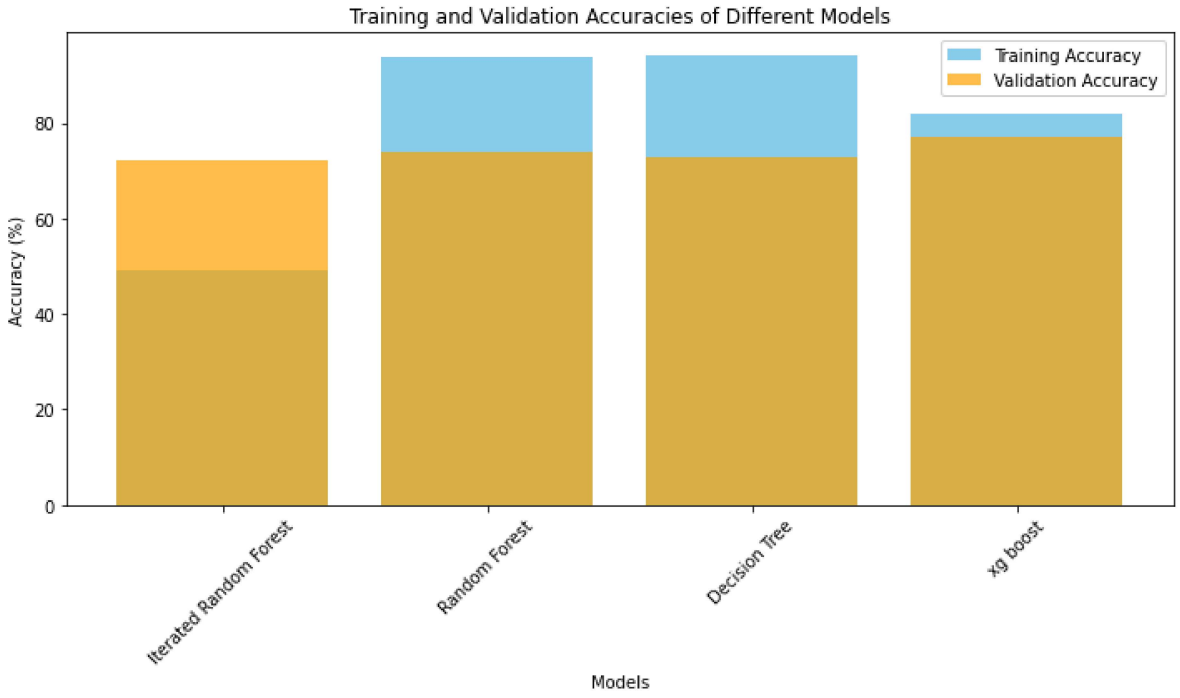
The plotted graph represents the feature importance scores for the top N features obtained from an XGBoost model.

PLOTTING A GRAPH OF THE MODELS ABOVE

In [ ]:

```
# Model names
models = ['Iterated Random Forest', 'Random Forest', 'Decision Tree','xg boost']

# Training and validation accuracies
training_accuracies = [49.11, 93.95, 94.19,81.82]
validation_accuracies = [72.17, 73.96, 72.86,76.91]
```

```
# Plotting
plt.figure(figsize=(10, 6))
plt.bar(models, training_accuracies, color='skyblue', label='Training Accuracy')
plt.bar(models, validation_accuracies, color='orange', alpha=0.7, label='Validat

# Adding labels and title
plt.xlabel('Models')
plt.ylabel('Accuracy (%)')
plt.title('Training and Validation Accuracies of Different Models')
plt.legend()

# Showing plot
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



EVALUATION AND INTERPRETATION OF THE BEST MODEL

XGBoost appears to be the best model overall, as it shows a good balance between training and validation accuracies, indicating that it generalizes well to unseen data.

Random Forest and Decision Tree models show signs of overfitting, performing exceptionally well on training data but with a drop in validation accuracy.

Iterated Random Forest shows signs of underfitting, with lower accuracies on both training and validation data, indicating it may not have captured the underlying patterns in the data effectively.

This suggests that the model correctly predicts the "functional" class. It was chosen not for having the highest training or validation accuracy scores, but for exhibiting the smallest difference between the two. This indicates that the model is less likely to perform best when applied to unseen data. It's the least prone to overfitting among the models that

which applied to unseen data. It's the least prone to overfitting among the models that meet our success criteria.

CONCLUSION

The xg boost model performed the best with an accuracy of 77.94% with decision trees and random overfitting and iterated random forest underfitting.

Communal standpipe seems to be the most popular waterpoint type.

Most water points that are not paid for are non functional.

Most water points that are paid for are functional.

Dry water points have more non functional wells than functional wells.

RECOMMENDATION

Investments should be directed towards communal standpipes, rather than communal standpipe multiples, as the majority of the latter are non-functional.

Priority should be given to non-functional wells and functional wells that need repair, provided they have sufficient water.

Providing payment creates an incentive and means to maintain wells in a functional state.

The central regions of the country may have fewer wells, likely due to lower population density. However, it's crucial for the government to ensure adequate water supply for the residents in those areas.

NEXT STEPS

Monitor the wells and update the model regularly to continuously improve our startegy

Better data trained in our model will improve the predictions

Learn cost of repairs, construction, and preventive maintenance

Create a cost-benefit function to prioritize actions