Tanzania Water Wells.

Overview

Business Understanding

Tanzania is known to be a well developing nation with a population of approximaltely 60 milion people.Millions of people in Tanzania, struggle to get clean water and are forced to cover very long distances in search of clean water. Wells are the main source of water for most Tanzanians,however many are broken or in bad shape. This shortage of water is a major issue as water is an essential in various sectors,leading to poor health,slow economic growth and hindered productivity. This projects with the use of machine learning tools aims at identifying why some wells fail,predict whether the new wells will work .

. Challenges .

1. The population of some areas is relatively high as compared to access of water.
2. Poor water quality resulting to diseases.
3. Insufficient funding for the maintainance and also development.
4. Climate change

Problem Statement

Water shortage is a crucial problem in Tanzania and this arises majorly from the maintenance of wells.Maintaning these water sources is also a challenge maybe because of water quality,geographical location,infrastracture upkeep and many other factors. Some are non-functonal while others are partially functional.This may limit the local citizens from accessing clean water. Challenges faced include: Health and sanitation issues,Poor economic growth and hindered productivity.

Our project here aims to help those who are in charge of water be it governmeantal ,non-governmental organisations and maybe policymakers to make improved or better decisions that would help in improvement of the functionality and maintenance of water wells in the country.

1.3 Objective

The objective of our project is come up with a prdeictive model that will determine functionality of water wells accurately.Using the data provided and making use of our knowledge in data science, our project focuses on addressing the challenge of the non-funcionality of the wells,improving access to clean water for the people. Focus of our project is:

1. Identifying key factors that may contribute to functionality of waterwells in Tanzania.
2. Developing a predictive model that can predict functionality of water wells based on attributes such as geographical location,infrastracture and water quality.
3. Enable data-driven decision making.
4. provide recommendations that can boost on the efforts for maintenance

Specific Objectives

1. determine if functionality of the wells varies on quantity.
2. Identify the most popular water point type.
3. Determine whether the status of functionality is related to the payment type.

Main objective

Build a classifier model that will determine functionality of waterwells accurately.

Success Metrics

Provide 60 % and above accuracy on wells functionality.

Data Understanding

We start by describing our data.

Data Sources

Got this data provided from Taarifa and the Tanzanian Ministry of Water.It is organized into three separate CSV files: [https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/data/ (https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/data/)](https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/data/)

1. Test Set Values
2. Training Set Labels
3. Training Set Values

The labels in this dataset The labels in this dataset are simple. There are three possible values:

1. functional - the waterpoint is operational and there are no repairs needed
2. functional needs repair - the waterpoint is operational, but needs repairs
3. non functional - the waterpoint is not operational

The predictor variables in this data include:

1. amount_tsh - Total static head (amount water available to wterpoint)
2. date_recorded - The date the row was entered
3. funder - Who funded the well
4. gps_height - Altitude of the well
5. installer - Organization that installed the well
6. longitude - GPS coordinate
7. latitude - GPS coordinate
8. wpt_name - Name of the waterpoint if there is one
9. num_private - Number of households with private access to the well
10. basin - Geographic water basin
11. subvillage - Geographic location
12. region - Geographic location
13. region_code - Geographic location
14. district_code - Geographic location
15. lga - Geographic location
16. ward - Geographic location
17. population - Population around the well
18. public_meeting - True/False
19. recorded_by - Group entering this row of data
20. scheme_management - Who operates the waterpoint
21. scheme_name - Who operates the waterpoint
22. permit - If the waterpoint is permitted
23. construction_year - Year the waterpoint was constructed
24. extraction_type - The kind of extraction the waterpoint uses
25. extraction_type_group - The kind of extraction the waterpoint uses
26. extraction_type_class - The kind of extraction the waterpoint uses
27. management - How the waterpoint is managed
28. management_group - How the waterpoint is managed
29. payment - What the water costs
30. payment_type - What the water costs
31. water_quality - The quality of the water
32. quality_group - The quality of the water
33. quantity - The quantity of water
34. quantity_group - The quantity of water
35. source - The source of the water
36. source_type - The source of the water
37. source_class - The source of the water
38. waterpoint_type - The kind of waterpoint
39. waterpoint_type_group - The kind of waterpoint

Since we have described our data already,we will go ahead and import the relevant libraries and load our data.

In [9]:
```
#import the relevant libraries
import pandas as pd
import numpy as np
import seaborn as sns
import statsmodels.api as sm
import matplotlib.pyplot as plt
%matplotlib inline
from scipy.stats import norm
from scipy import stats
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

We are going to go ahead and load our datasets

In [10]:
```
# Loading the datasets
training_data = pd.read_csv("Training_Set_values.csv")
testing_data = pd.read_csv("Test_set.csv")
training_labels = pd.read_csv("Training_Set_labels.csv")
```

In [11]:
```
# concatenating train_labels and data
train_df = pd.concat([training_data, training_labels],axis=1)

# merging the data sets
train_df = training_data.merge(training_labels, how = "inner")

print(train_df)
```

```
          id  amount_tsh date_recorded          funder  gps_height  \
0      69572      6000.0    2011-03-14           Roman        1390   
1       8776         0.0    2013-03-06         Grumeti        1399   
2      34310        25.0    2013-02-25    Lottery Club         686   
3      67743         0.0    2013-01-28          Unicef         263   
4      19728         0.0    2011-07-13     Action In A           0   
...      ...         ...           ...             ...         ...   
59395  60739        10.0    2013-05-03  Germany Republi        1210   
59396  27263      4700.0    2011-05-07     Cefa-njombe        1212   
59397  37057         0.0    2011-04-11             NaN           0   
59398  31282         0.0    2011-03-08           Malec           0   
59399  26348         0.0    2011-03-23      World Bank         191   

          installer  longitude   latitude              wpt_name  num_pr
ivate  \
0             Roman  34.938093  -9.856322                  none       
0  
1           GRUMETI  34.698766  -2.147466              Zahanati       
0  
2      World vision  37.460664  -3.821329           Kwa Mahundi       
0  
3            UNICEF  38.486161 -11.155298  Zahanati Ya Nanyumbu       
0  
4           Artisan  31.130847  -1.825359               Shuleni       
0  
...             ...        ...        ...                   ...       
...  
59395           CES  37.169807  -3.253847   Area Three Namba 27       
0  
59396          Cefa  35.249991  -9.070629     Kwa Yahona Kuvala       
0  
59397           NaN  34.017087  -8.750434               Mashine       
0  
59398          Musa  35.861315  -6.378573                Mshoro       
0  
59399         World  38.104048  -6.747464       Kwa Mzee Lugawa       
0  

       ... 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   
...    ...           ...           ...           ...             ...   
59395  ...          soft          good        enough          enough   
59396  ...          soft          good        enough          enough   
59397  ...      fluoride      fluoride        enough          enough   
59398  ...          soft          good  insufficient    insufficient   
59399  ...         salty         salty        enough          enough   

                     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   
```

```
...                    ...                  ...          ...
59395               spring               spring  groundwater
59396                river           river/lake      surface
59397          machine dbh             borehole  groundwater
59398         shallow well         shallow well  groundwater
59399         shallow well         shallow well  groundwater

                  waterpoint_type waterpoint_type_group    status_grou
p
0              communal standpipe    communal standpipe      functiona
l
1              communal standpipe    communal standpipe      functiona
l
2     communal standpipe multiple    communal standpipe      functiona
l
3     communal standpipe multiple    communal standpipe  non functiona
l
4              communal standpipe    communal standpipe      functiona
l
...                           ...                   ...
...
59395          communal standpipe    communal standpipe      functiona
l
59396          communal standpipe    communal standpipe      functiona
l
59397                   hand pump             hand pump      functiona
l
59398                   hand pump             hand pump      functiona
l
59399                   hand pump             hand pump      functiona
l

[59400 rows x 41 columns]
```

In [12]:
```
train_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 [13]:
```
# Getting the data types of the data
train_df.dtypes.value_counts()
```

Out[13]:
```
object     31
int64       7
float64     3
dtype: int64
```

In [14]:
```
#function to check on the dataset shape, column names
def check_dataset(train_df):

    # Output the shape of the dataset
    print("Shape of dataset:", train_df.shape)

    # Output the column names of the dataset
    print("Column names:", list(train_df.columns))

check_dataset(train_df)
```

```
Shape of dataset: (59400, 41)
Column names: ['id', 'amount_tsh', 'date_recorded', 'funder', 'gps_heig
ht', 'installer', 'longitude', 'latitude', 'wpt_name', 'num_private',
'basin', 'subvillage', 'region', 'region_code', 'district_code', 'lga',
'ward', 'population', 'public_meeting', 'recorded_by', 'scheme_manageme
nt', 'scheme_name', 'permit', 'construction_year', 'extraction_type',
'extraction_type_group', 'extraction_type_class', 'management', 'manage
ment_group', 'payment', 'payment_type', 'water_quality', 'quality_grou
p', 'quantity', 'quantity_group', 'source', 'source_type', 'source_clas
s', 'waterpoint_type', 'waterpoint_type_group', 'status_group']
```

Data Preparation

In [15]:
```
class DataCleaner:
    def __init__(self, dataframe):
        self.dataframe = dataframe

    def check_duplicates(self):
        # Returns a boolean Series.
        duplicated = self.dataframe[self.dataframe.duplicated()]
        if not duplicated.empty:
            print("Found duplicated rows:")
            print(duplicated)
        else:
            print("No duplicated rows found.")
        return duplicated

    def drop_duplicates(self):
        self.dataframe = self.dataframe.drop_duplicates().reset_index(dr
        print("Duplicated rows dropped.")
        return self.dataframe


df = pd.DataFrame(train_df)
cleaner = DataCleaner(df)

# Check for duplicates
duplicates = cleaner.check_duplicates()

# Drop duplicates
cleaned_df = cleaner.drop_duplicates()

print("Cleaned DataFrame:")
print(cleaned_df)

# Loading the large dataset (same as before)
```

```
No duplicated rows found.
Duplicated rows dropped.
Cleaned DataFrame:
          id  amount_tsh date_recorded          funder  gps_height  \
0      69572      6000.0    2011-03-14           Roman        1390   
1       8776         0.0    2013-03-06         Grumeti        1399   
2      34310        25.0    2013-02-25    Lottery Club         686   
3      67743         0.0    2013-01-28          Unicef         263   
4      19728         0.0    2011-07-13    Action In A           0   
...      ...         ...           ...             ...         ...   
59395  60739        10.0    2013-05-03  Germany Republi        1210   
59396  27263      4700.0    2011-05-07      Cefa-njombe        1212   
59397  37057         0.0    2011-04-11             NaN           0   
59398  31282         0.0    2011-03-08           Malec           0   
59399  26348         0.0    2011-03-23      World Bank         191   

          installer  longitude   latitude              wpt_name  num_pr
ivate  \
0             Roman  34.938093  -9.856322                  none        
0   
1           GRUMETI  34.698766  -2.147466              Zahanati        
0   
2      World vision  37.460664  -3.821329           Kwa Mahundi        
0   
3            UNICEF  38.486161 -11.155298  Zahanati Ya Nanyumbu        
0   
4           Artisan  31.130847  -1.825359               Shuleni        
0   
...             ...        ...        ...                   ...        
...   
59395           CES  37.169807  -3.253847   Area Three Namba 27        
0   
59396          Cefa  35.249991  -9.070629     Kwa Yahona Kuvala        
0   
59397           NaN  34.017087  -8.750434               Mashine        
0   
59398          Musa  35.861315  -6.378573                Mshoro        
0   
59399         World  38.104048  -6.747464       Kwa Mzee Lugawa        
0   

       ... 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   
...    ...           ...           ...           ...             ...   
59395  ...          soft          good        enough          enough   
59396  ...          soft          good        enough          enough   
59397  ...      fluoride      fluoride        enough          enough   
59398  ...          soft          good  insufficient    insufficient   
59399  ...         salty         salty        enough          enough   

                     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
...                      ...                      ...          ...
59395                 spring                   spring  groundwater
59396                  river               river/lake      surface
59397            machine dbh                 borehole  groundwater
59398           shallow well             shallow well  groundwater
59399           shallow well             shallow well  groundwater

                   waterpoint_type waterpoint_type_group    status_grou
p
0               communal standpipe    communal standpipe      functiona
l
1               communal standpipe    communal standpipe      functiona
l
2      communal standpipe multiple    communal standpipe      functiona
l
3      communal standpipe multiple    communal standpipe  non functiona
l
4               communal standpipe    communal standpipe      functiona
l
...                            ...                   ...
...
59395           communal standpipe    communal standpipe      functiona
l
59396           communal standpipe    communal standpipe      functiona
l
59397                    hand pump             hand pump      functiona
l
59398                    hand pump             hand pump      functiona
l
59399                    hand pump             hand pump      functiona
l

[59400 rows x 41 columns]
```

The DataCleaner class is defined and it contains methods to check for and remove duplicate rows from dataframe.

We create a large dataset with multiple columns. From the output we see that we had duplicated rows and we ares shown how our cleaned dataframe looked like after we dropped duplicates. and then checked for the duplicates again and none were found.

Checking for missing values and handling them

In [16]:
```
class DataCleaner:
    def __init__(self, dataframe):
        self.dataframe = dataframe

    def replace_missing_with_mode(self):
        """
        Replaces missing values in the DataFrame with the mode of each co

        Returns:
            DataFrame: A DataFrame with missing values replaced by the mo
        """
        for column in self.dataframe.columns:
            if self.dataframe[column].isnull().any():
                mode_value = self.dataframe[column].mode()[0]
                self.dataframe[column].fillna(mode_value, inplace=True)
                print(f"Missing values in column '{column}' replaced with
        return self.dataframe

df = pd.DataFrame(train_df)
handler = DataCleaner(train_df)

# Replace missing values with mode (after method definition)
df_mode = handler.replace_missing_with_mode()
print("DataFrame after replacing missing values with mode:")
print(df_mode)
```

```
Missing values in column 'funder' replaced with mode: Government Of Tan
zania.
Missing values in column 'installer' replaced with mode: DWE.
Missing values in column 'subvillage' replaced with mode: Madukani.
Missing values in column 'public_meeting' replaced with mode: True.
Missing values in column 'scheme_management' replaced with mode: VWC.
Missing values in column 'scheme_name' replaced with mode: K.
Missing values in column 'permit' replaced with mode: True.
DataFrame after replacing missing values with mode:
          id  amount_tsh date_recorded                  funder  gps_hei
ght  \
0      69572      6000.0    2011-03-14                   Roman        1
390
1       8776         0.0    2013-03-06                 Grumeti        1
399
2      34310        25.0    2013-02-25            Lottery Club
686
3      67743         0.0    2013-01-28                  Unicef
263
4      19728         0.0    2011-07-13            Action In A
```

We checked for missing values and found some that were shown in the columns above which were replaced by the mode using fillna method.

Some columns here need to be converted to categorical

In [17]:
```
df_mode.info
```

Out[17]:
```
<bound method DataFrame.info of           id  amount_tsh date_recorded  
funder  gps_height  \
0      69572      6000.0    2011-03-14                   Roman        1
390
1       8776         0.0    2013-03-06                 Grumeti        1
399
2      34310        25.0    2013-02-25            Lottery Club         
686
3      67743         0.0    2013-01-28                  Unicef         
263
4      19728         0.0    2011-07-13             Action In A         
0
...      ...         ...           ...                     ...         
...
59395  60739        10.0    2013-05-03         Germany Republi        1
210
59396  27263      4700.0    2011-05-07             Cefa-njombe        1
212
59397  37057         0.0    2011-04-11  Government Of Tanzania         
0
59398  31282         0.0    2011-03-08                   Malec         
0
59399  26348         0.0    2011-03-23              World Bank         
191

         installer  longitude   latitude              wpt_name  num_pr
ivate  \
0            Roman  34.938093  -9.856322                  none         
0
1          GRUMETI  34.698766  -2.147466              Zahanati         
0
2     World vision  37.460664  -3.821329           Kwa Mahundi         
0
3           UNICEF  38.486161 -11.155298  Zahanati Ya Nanyumbu         
0
4          Artisan  31.130847  -1.825359               Shuleni         
0
...            ...        ...        ...                   ...         
...
59395          CES  37.169807  -3.253847   Area Three Namba 27         
0
59396         Cefa  35.249991  -9.070629     Kwa Yahona Kuvala         
0
59397          DWE  34.017087  -8.750434               Mashine         
0
59398         Musa  35.861315  -6.378573                Mshoro         
0
59399        World  38.104048  -6.747464       Kwa Mzee Lugawa         
0

       ... 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   
...    ...           ...           ...           ...             ...   
```

```
59395  ...          soft          good       enough          enough
59396  ...          soft          good       enough          enough
59397  ...      fluoride      fluoride       enough          enough
59398  ...          soft          good  insufficient    insufficient
59399  ...         salty         salty       enough          enough

                      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
...                      ...                   ...          ...
59395                 spring                spring  groundwater
59396                  river            river/lake      surface
59397            machine dbh              borehole  groundwater
59398           shallow well          shallow well  groundwater
59399           shallow well          shallow well  groundwater

                   waterpoint_type  waterpoint_type_group    status_gro
up
0               communal standpipe     communal standpipe      function
al
1               communal standpipe     communal standpipe      function
al
2      communal standpipe multiple     communal standpipe      function
al
3      communal standpipe multiple     communal standpipe  non function
al
4               communal standpipe     communal standpipe      function
al
...                            ...                    ...
...
59395           communal standpipe     communal standpipe      function
al
59396           communal standpipe     communal standpipe      function
al
59397                    hand pump              hand pump      function
al
59398                    hand pump              hand pump      function
al
59399                    hand pump              hand pump      function
al

[59400 rows x 41 columns]>
```

In [18]:
```
#Converting some columns into categorical
for col in ['basin','region','region_code','district_code','public_meeti
    df_mode[col] = df_mode[col].astype('category')

print(df_mode.dtypes)
```

```
id                          int64
amount_tsh                float64
date_recorded              object
funder                     object
gps_height                  int64
installer                  object
longitude                 float64
latitude                  float64
wpt_name                   object
num_private                 int64
basin                    category
subvillage                 object
region                   category
region_code              category
district_code            category
lga                        object
ward                       object
population                  int64
public_meeting           category
recorded_by              category
scheme_management        category
scheme_name                object
permit                       bool
construction_year           int64
extraction_type            object
extraction_type_group      object
extraction_type_class      object
management                 object
management_group           object
payment                    object
payment_type             category
water_quality            category
quality_group              object
quantity                 category
quantity_group             object
source                   category
source_type                object
source_class               object
waterpoint_type          category
waterpoint_type_group      object
status_group               object
dtype: object
```

We chose to convert specific columns into categorical due to nature of the data and the modelling taks we plan to perform.

In [19]:
```
columns_to_drop = ['id', 'num_private', 'recorded_by']  # List columns yo

df_mode.drop(columns=columns_to_drop, inplace=True)
```

Explanatory Data Analysis

In EDA, we delve into a deeper understanding of the dataset gining insights from the content and also its structures through visualizations,and exploration techniques.

Univariate Analysis

We will start by finding the number of unique funders for the project and the call a function to display the top 12 funders.

In [20]:
```
U_funders = df_mode['funder'].nunique()
print("There are {} unique values of funders".format(U_funders))
```

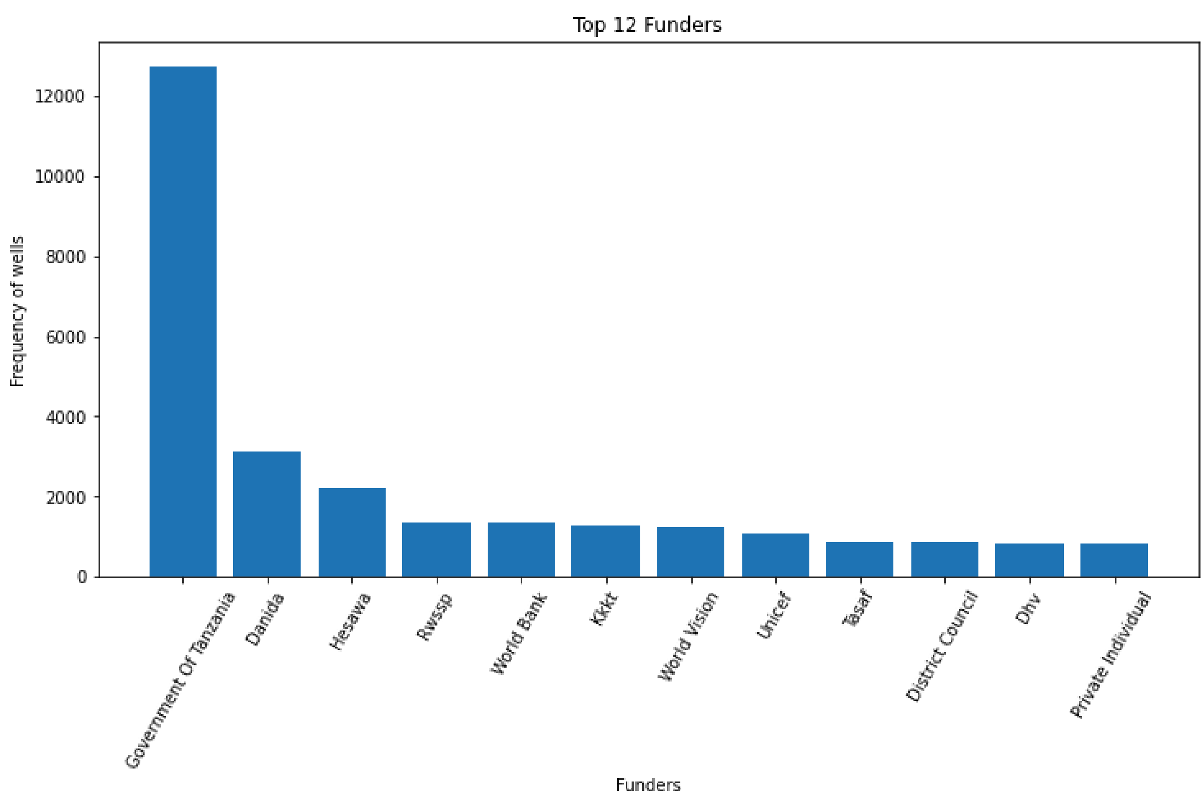
```
There are 1897 unique values of funders
```

In [21]:
```
def display_top_funders(data, n=15):
    top_funders = df_mode['funder'].value_counts().head(n)
    print(f"Top {n} Funders:")
    print(top_funders)


display_top_funders(df_mode, n=12)
```

```
Top 12 Funders:
Government Of Tanzania    12719
Danida                     3114
Hesawa                     2202
Rwssp                      1374
World Bank                 1349
Kkkt                       1287
World Vision               1246
Unicef                     1057
Tasaf                       877
District Council            843
Dhv                         829
Private Individual          826
Name: funder, dtype: int64
```

Bivariate Analysis

In [22]:
```
# Now plot the top twelve funders for thewells.

def display_funders(data, n=12):
    funders = df_mode['funder'].value_counts().head(n)

    # Plot the  funders
    plt.figure(figsize=(12, 6))
    plt.bar(funders.index, funders.values)
    plt.title('Top 12 Funders')
    plt.xlabel('Funders')
    plt.ylabel('Frequency of wells')
    plt.xticks(rotation=60)
    plt.show()

    return funders


funders = display_funders(df_mode, n=12)
```



It is evident that the government of Tanzania is the biggest funder,seconded by Danida and rthen the rest follow.

We will go ahead and for the number of unique well installers.

In [23]:
```
# Assuming df_mode is already defined
unique_installers = df_mode['installer'].nunique()
print("There are {} unique values of installers".format(unique_installer
```

```
There are 2145 unique values of installers
```

In [24]:
```

data = {'installer': ['Roman', 'Grumeti', 'World vision', 'UNICEF', 'Art
df = pd.DataFrame(data)

# Get the value counts for the 'installer' column
installer_counts = df['installer'].value_counts()

# Print the sum of the value counts
print(installer_counts.sum())
```

```
18
```

Given 18 then that means the None was nit counted so will introduce a code that will cater for None.

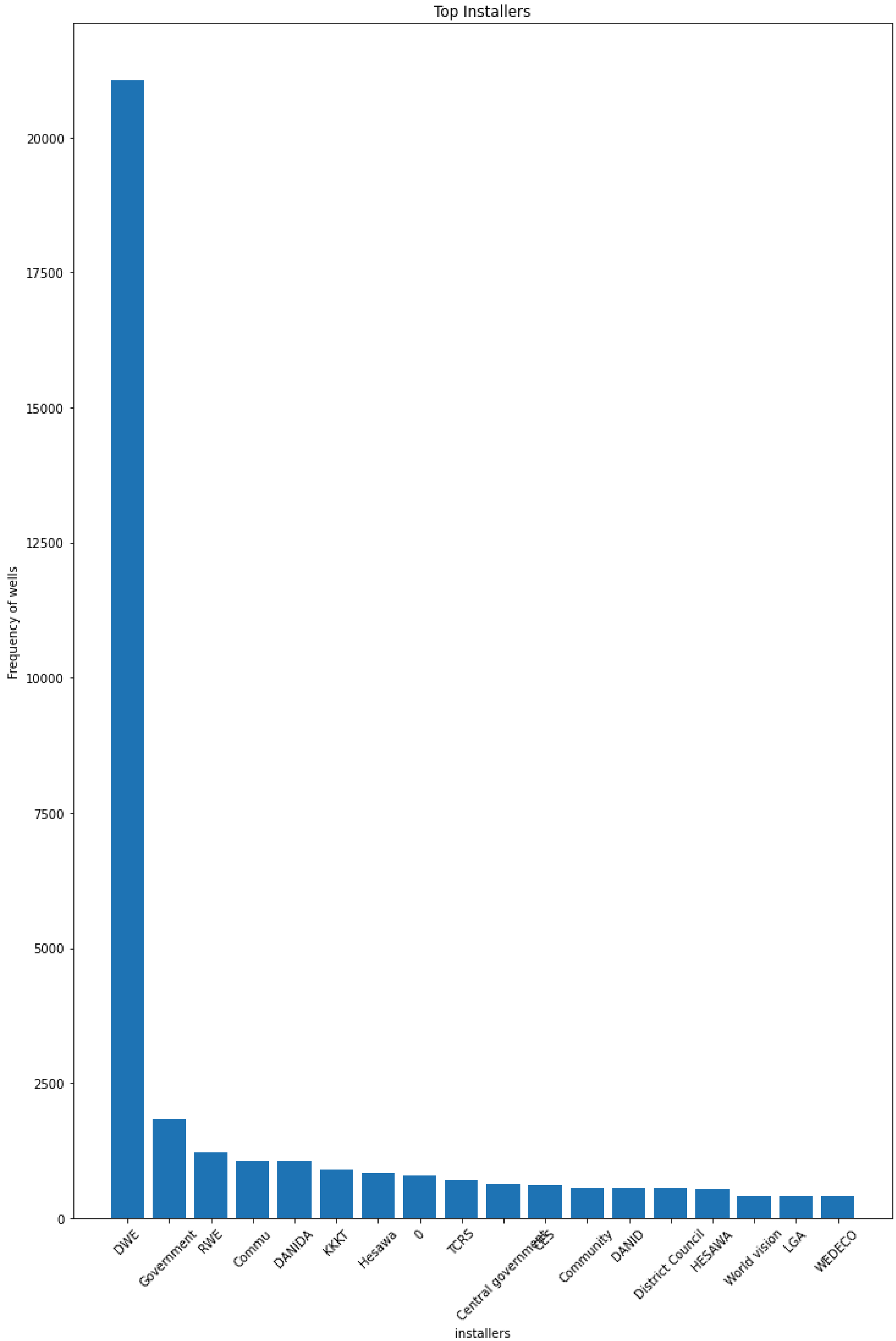
In [25]:
```
# Check for missing values (None) in the 'installer' column
missing_values = df['installer'].isnull().sum()
print(f"Number of missing values (None): {missing_values}")
#instead of excluding 'None' from the count, we will then replace ut wit


  # Option 2: Replace None with a different value
df['installer'] = df['installer'].fillna('Unknown')
installer_counts = df['installer'].value_counts()
print(f"Total installers (including replaced None): {installer_counts.su
```

```
Number of missing values (None): 2
Total installers (including replaced None): 20
```

In [26]:
```
#Display the ten top installers
top_installers = df_mode ['installer'].value_counts().head(10)
print(top_installers)
```

```
DWE                   21057
Government             1825
RWE                    1206
Commu                  1060
DANIDA                 1050
KKKT                    898
Hesawa                  840
0                       777
TCRS                    707
Central government      622
Name: installer, dtype: int64
```

In [27]:
```python
# Now plot the top installers for the wells.

def display_installers(data, n=18):
    top_installers = df_mode['installer'].value_counts().head(n)

    # Plot the  funders
    plt.figure(figsize=(12, 18))
    plt.bar(top_installers.index, top_installers.values)
    plt.title('Top Installers')
    plt.xlabel('installers')
    plt.ylabel('Frequency of wells')
    plt.xticks(rotation=45)
    plt.show()

    return top_installers


funders = display_installers(df_mode, n=18)
```



From the graphical representation above, we see that the top installer is DWE, seconded by the

Year of construction of wells.

In [28]:
```
df_mode['construction_year'].value_counts()
```

Out[28]:
```
0       20709
2010     2645
2008     2613
2009     2533
2000     2091
2007     1587
2006     1471
2003     1286
2011     1256
2004     1123
2012     1084
2002     1075
1978     1037
1995     1014
2005     1011
1999      979
1998      966
1990      954
1985      945
1980      811
1996      811
1984      779
1982      744
1994      738
1972      708
1974      676
1997      644
1992      640
1993      608
2001      540
1988      521
1983      488
1975      437
1986      434
1976      414
1970      411
1991      324
1989      316
1987      302
1981      238
1977      202
1979      192
1973      184
2013      176
1971      145
1960      102
1967       88
1963       85
1968       77
1969       59
1964       40
1962       30
1961       21
1965       19
1966       17
Name: construction_year, dtype: int64
```

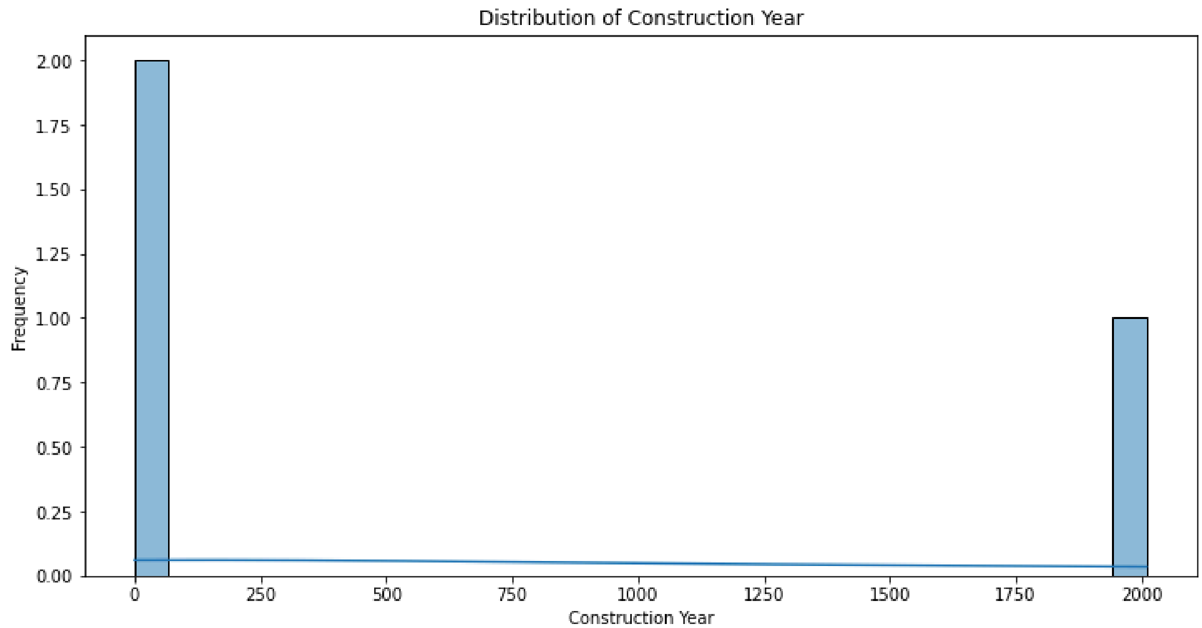
The output listed shows the number of constructions done in various years but since 0 is invalid year then it is likely to be the place holder of missing values but we will not drop it,or even convert it into mdeian because it has majority of the dataset .We will have to use it as a new value.

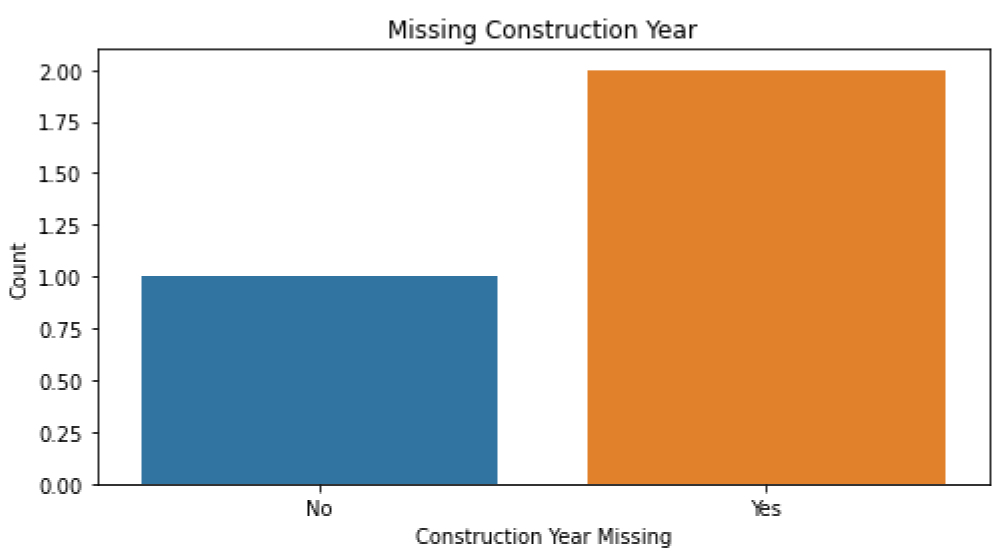
We shall create a new column and we will not alter with the original data.

In [29]:
```
data = {
    'amount_tsh': [0, 10, 20],
    'gps_height': [10, 20, 30],
    'longitude': [35.0, 36.0, 37.0],
    'latitude': [-10.0, -9.0, -8.0],
    'population': [100, 200, 300],
    'age': [5, 10, 15],  # Add more features as needed
    'region': ['A', 'B', 'A'],
    'installer': ['Installer1', 'Installer2', 'Installer1'],
    'construction_year': [0, 2010, 0]
}
df = pd.DataFrame(data)

# Create a new column 'construction_year_missing' to flag the missing yea
df['construction_year_missing'] = df['construction_year'] == 0

# Display the updated DataFrame
print(df)
```

```
   amount_tsh  gps_height  longitude  latitude  population  age region  
\
0           0          10       35.0     -10.0         100    5      A   
1          10          20       36.0      -9.0         200   10      B   
2          20          30       37.0      -8.0         300   15      A   

    installer  construction_year  construction_year_missing  
0  Installer1                  0                       True  
1  Installer2               2010                      False  
2  Installer1                  0                       True  
```

In [30]:
```
# Plotting the construction_year distribution
plt.figure(figsize=(12, 6))
sns.histplot(df['construction_year'], bins=30, kde=True)
plt.title('Distribution of Construction Year')
plt.xlabel('Construction Year')
plt.ylabel('Frequency')
plt.show()

# Plotting the construction_year_missing column
plt.figure(figsize=(8, 4))
sns.countplot(x='construction_year_missing', data=df)
plt.title('Missing Construction Year')
plt.xlabel('Construction Year Missing')
plt.ylabel('Count')
plt.xticks([0, 1], ['No', 'Yes'])
plt.show()
```





Came up with two graphical representations of both the construction year and the missing construction year.

Population

Let us take a look at the population so that we can compare with functionality of the wells and know whether it is a factor.

In [31]:
```
df_mode['population'].value_counts()
```

Out[31]:
```
0       21381
1        7025
200      1940
150      1892
250      1681
        ...
3241        1
1960        1
1685        1
2248        1
1439        1
Name: population, Length: 1049, dtype: int64
```

In [32]:
```
df_mode.loc[df_mode['population']==0].groupby('status_group').count()
```

Out[32]:

	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wpt_na
status_group								
functional	11274	11274	11274	11274	11274	11274	11274	11
functional needs repair	1775	1775	1775	1775	1775	1775	1775	1
non functional	8332	8332	8332	8332	8332	8332	8332	8

3 rows × 37 columns

In [33]:
```
pop = df_mode['population'].describe()
print("Total number of people around the wells is",pop[0])
print("\n")
print("The average number of people living around the wells is",pop[1])
print("\n")
print("The minimum population value is",pop[3])
print("\n")
print("The maximum population value is",pop[7])
```

```
Total number of people around the wells is 59400.0


The average number of people living around the wells is 179.90998316498
317


The minimum population value is 0.0


The maximum population value is 30500.0
```

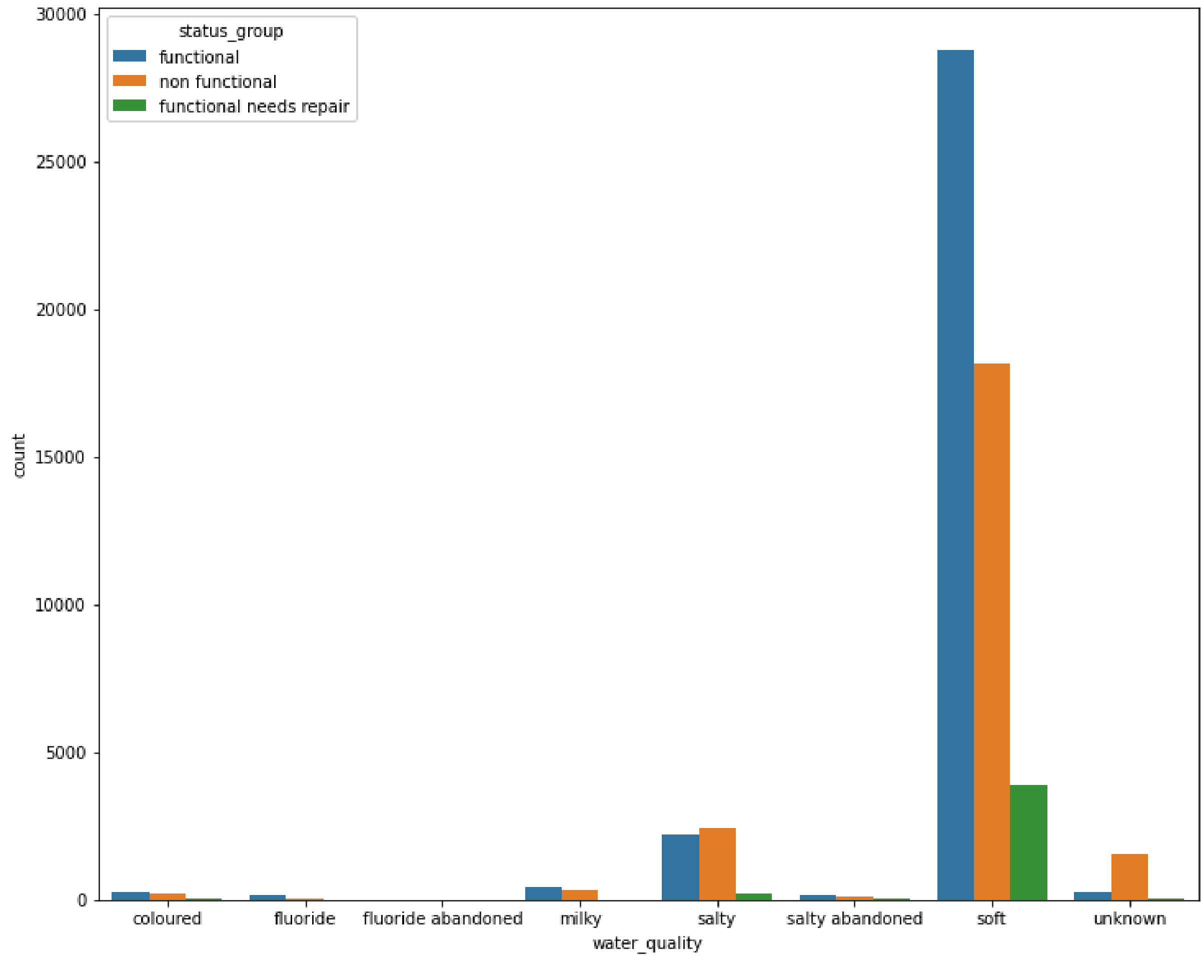
Water Quality.

In [34]:
```
df_mode['water_quality'].value_counts()
```

Out[34]:
```
soft                  50818
salty                  4856
unknown                1876
milky                   804
coloured                490
salty abandoned         339
fluoride                200
fluoride abandoned       17
Name: water_quality, dtype: int64
```

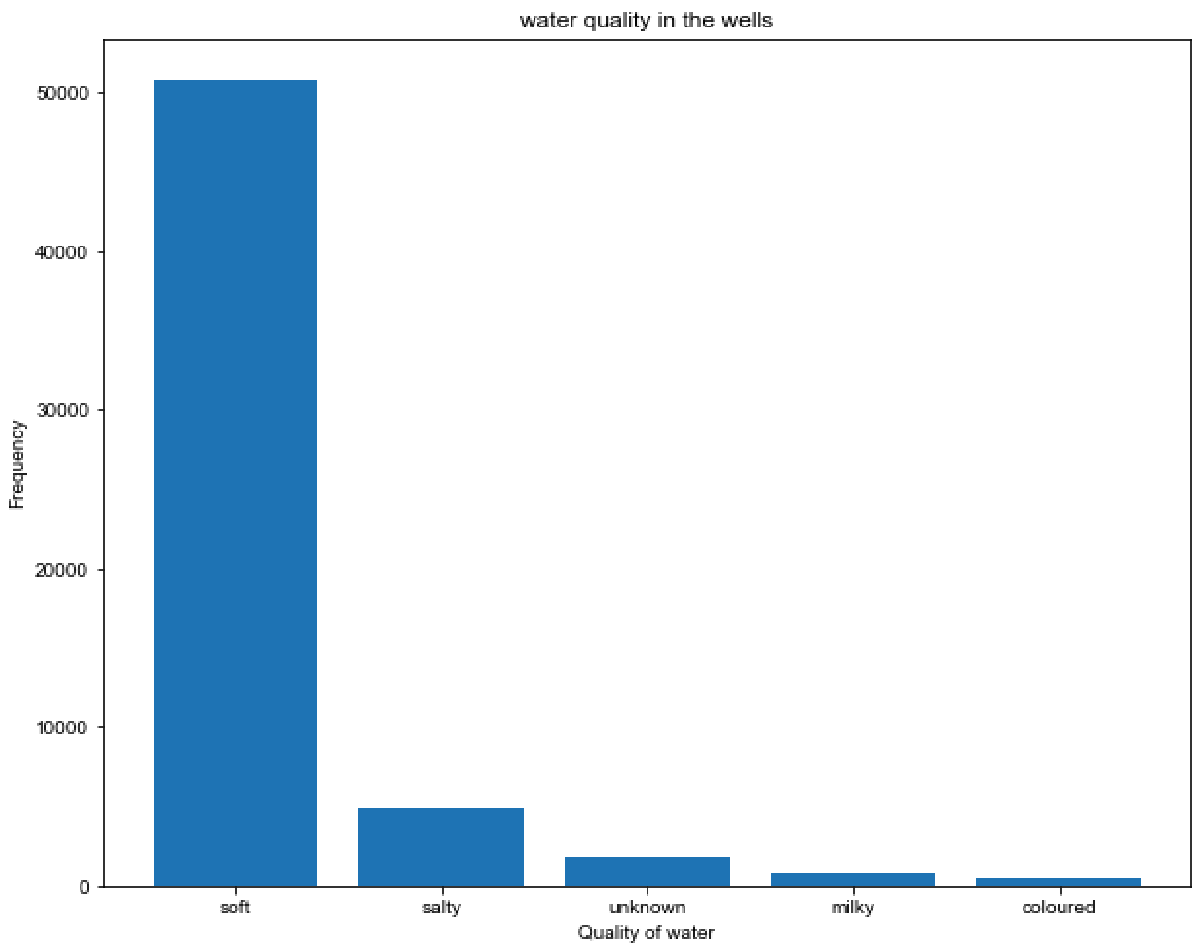
In [35]:

```
plt.figure(figsize=(12,10))
ax = sns.countplot(x='water_quality', hue="status_group", data=df_mode)
```



OR

In [36]:
```
fig, ax = plt.subplots(figsize=(10,8))
quality_count = train_df["water_quality"].value_counts().sort_values(asc
plt.bar(quality_count.index, quality_count.values)
plt.xlabel("Quality of water")
plt.ylabel("Frequency")
plt.title("water quality in the wells")
sns.set_style("darkgrid")
plt.show()
```

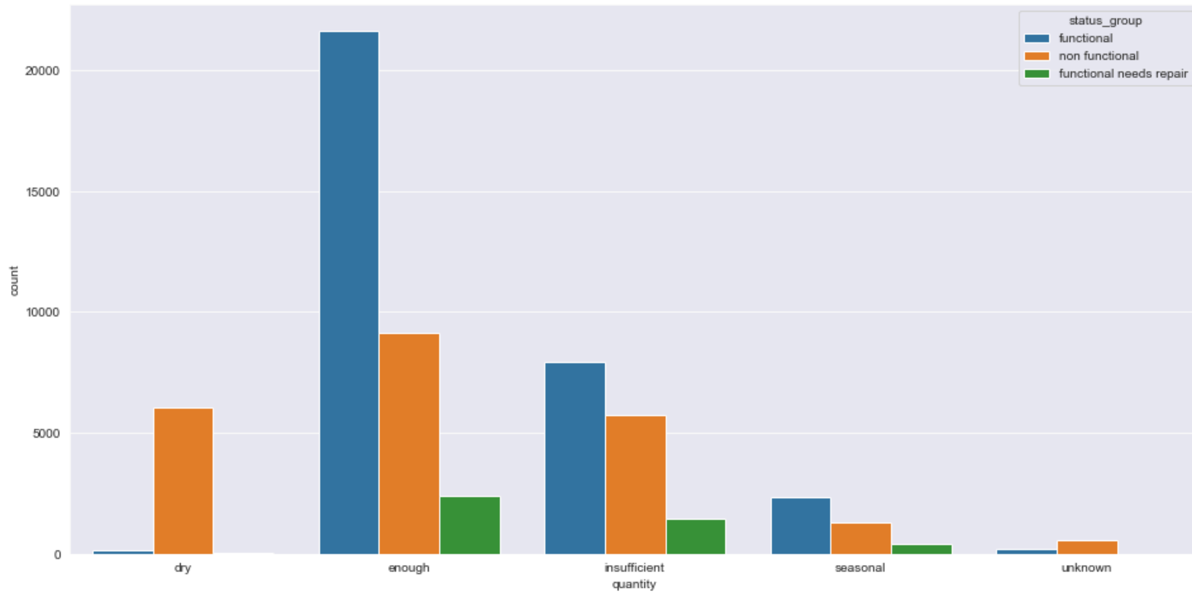


Quantity

In [37]:
```
df_mode['quantity'].value_counts()
```

Out[37]:
```
enough          33186
insufficient    15129
dry              6246
seasonal         4050
unknown           789
Name: quantity, dtype: int64
```

In [38]:
```
plt.figure(figsize=(16,8))
ax = sns.countplot(x='quantity', hue="status_group", data=df_mode)
```

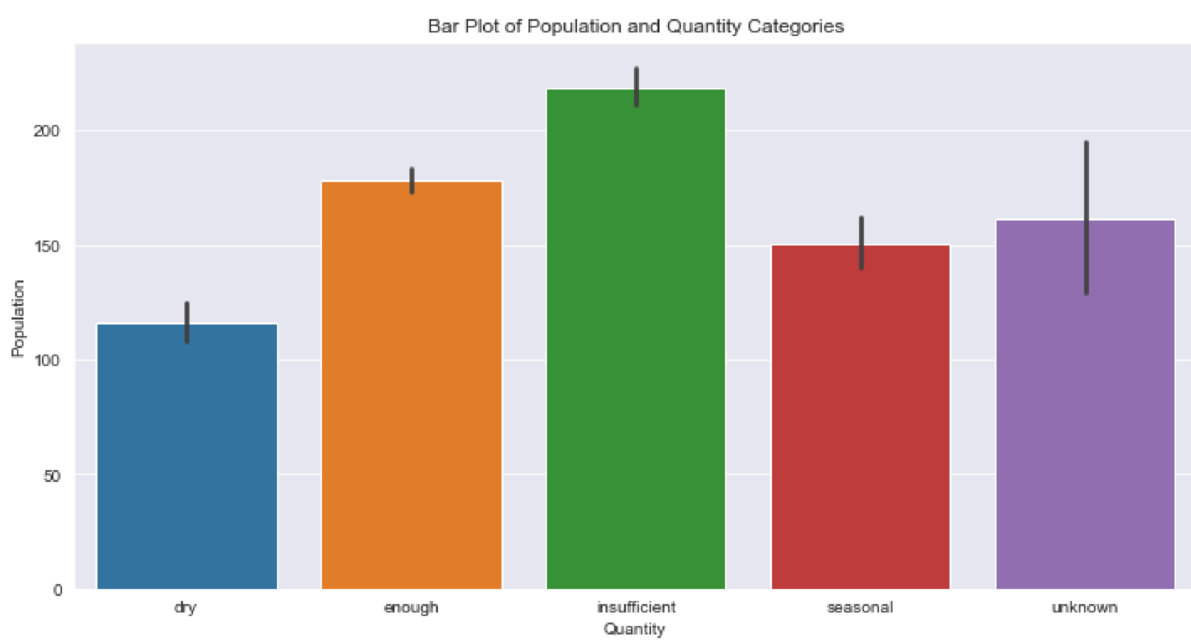


Bivariate Analysis

Relationship between the population and the quantity

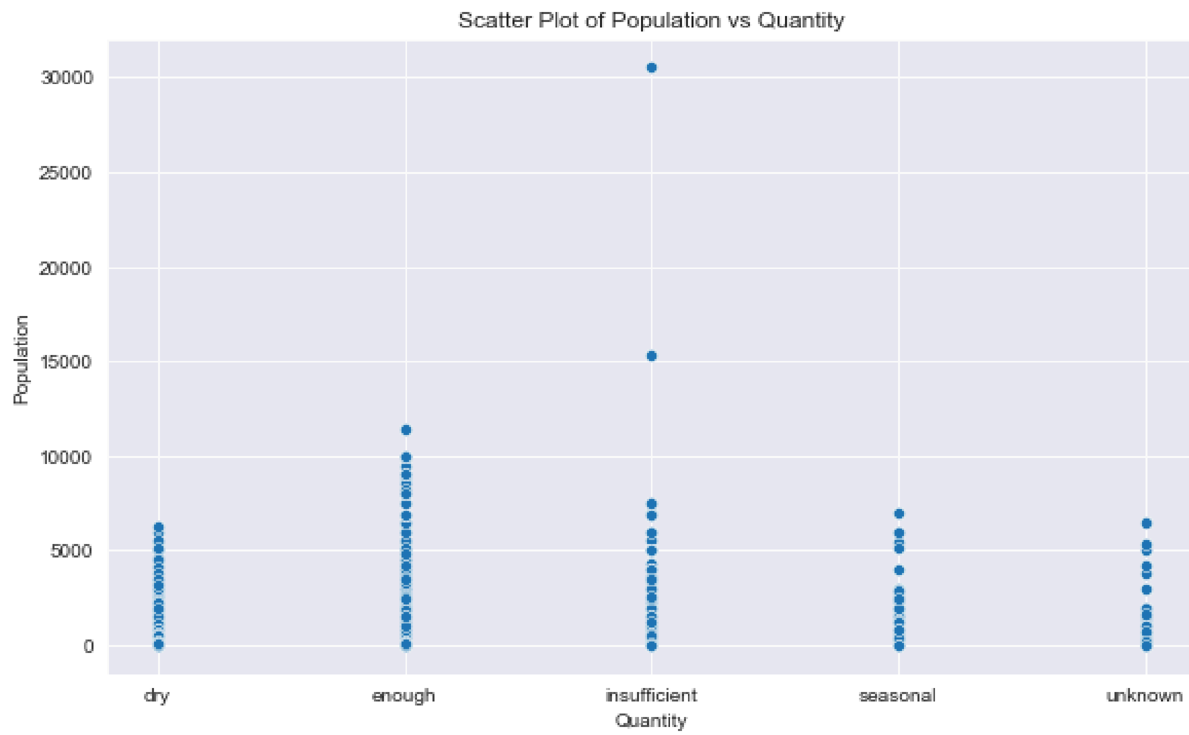
In [39]:
```
plt.figure(figsize=(12, 6))
sns.barplot(x='quantity', y='population', data=df_mode)
plt.title('Bar Plot of Population and Quantity Categories')
plt.xlabel('Quantity')
plt.ylabel('Population')
```

Out[39]:
```
Text(0, 0.5, 'Population')
```



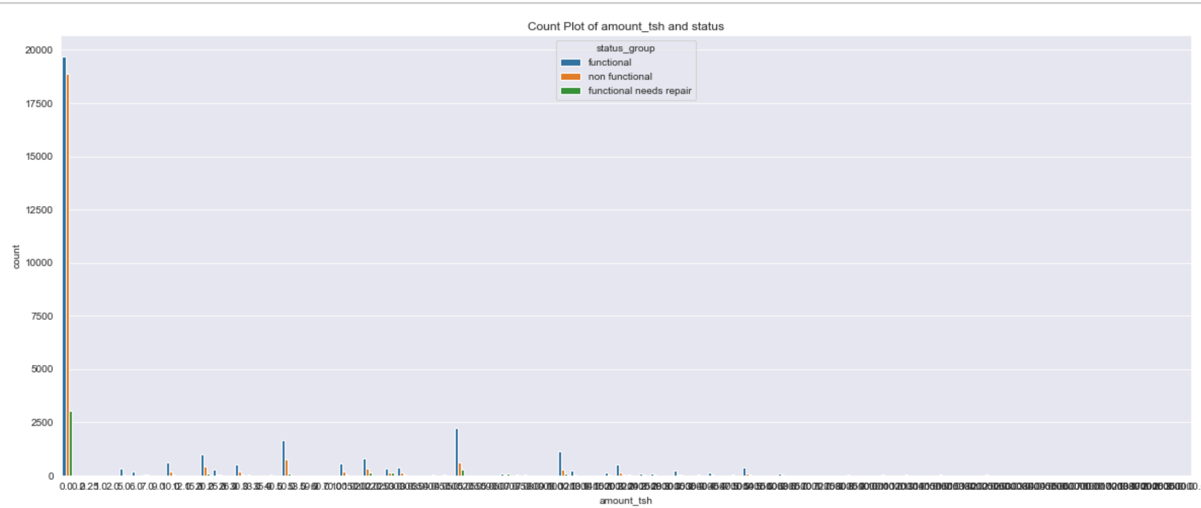
Below is a scatterplot of the same.

In [40]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_mode, x='quantity', y='population')
plt.title('Scatter Plot of Population vs Quantity')
plt.xlabel('Quantity')
plt.ylabel('Population')
plt.show()
```

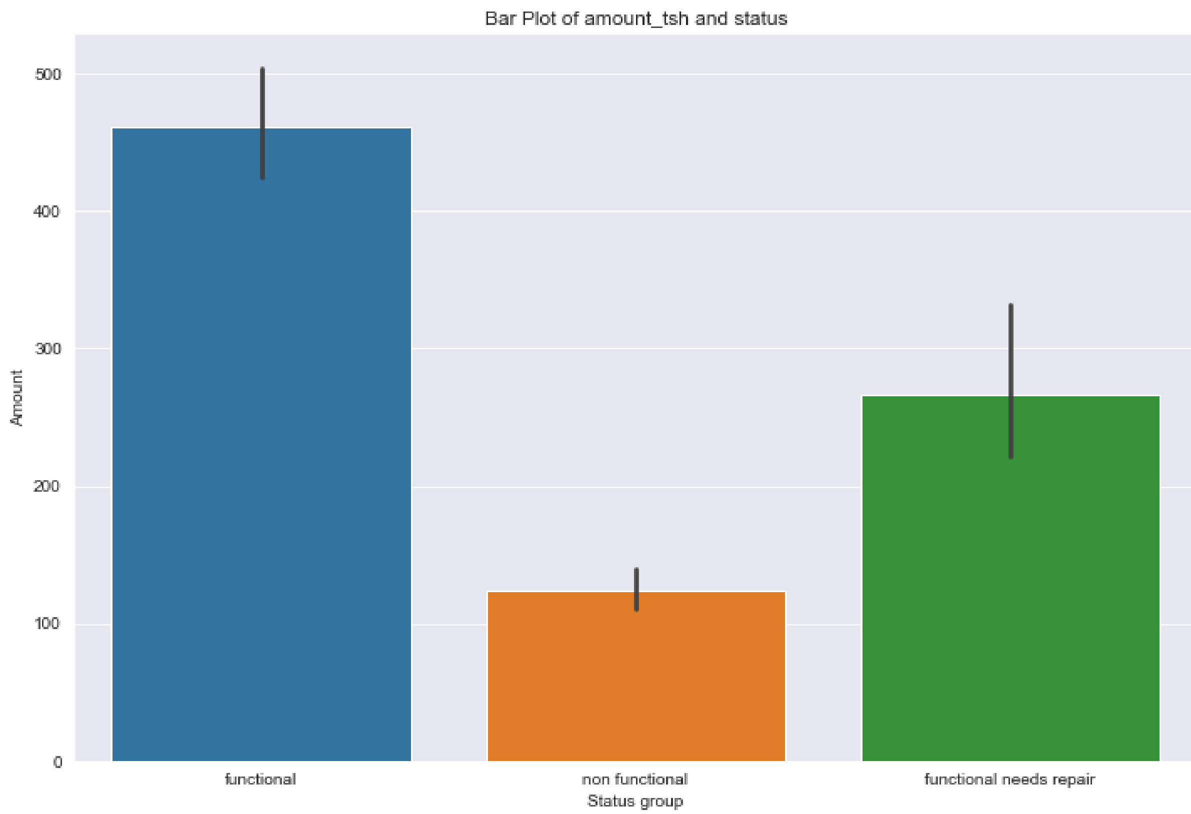


Relationship between amount_tsh and status

In [41]:
```
bar, ax = plt.subplots(figsize=(20,8))
ax = sns.countplot(data = df_mode, x="amount_tsh", hue="status_group")
ax.set_title("Count Plot of amount_tsh and status")
sns.set_style("darkgrid")
```

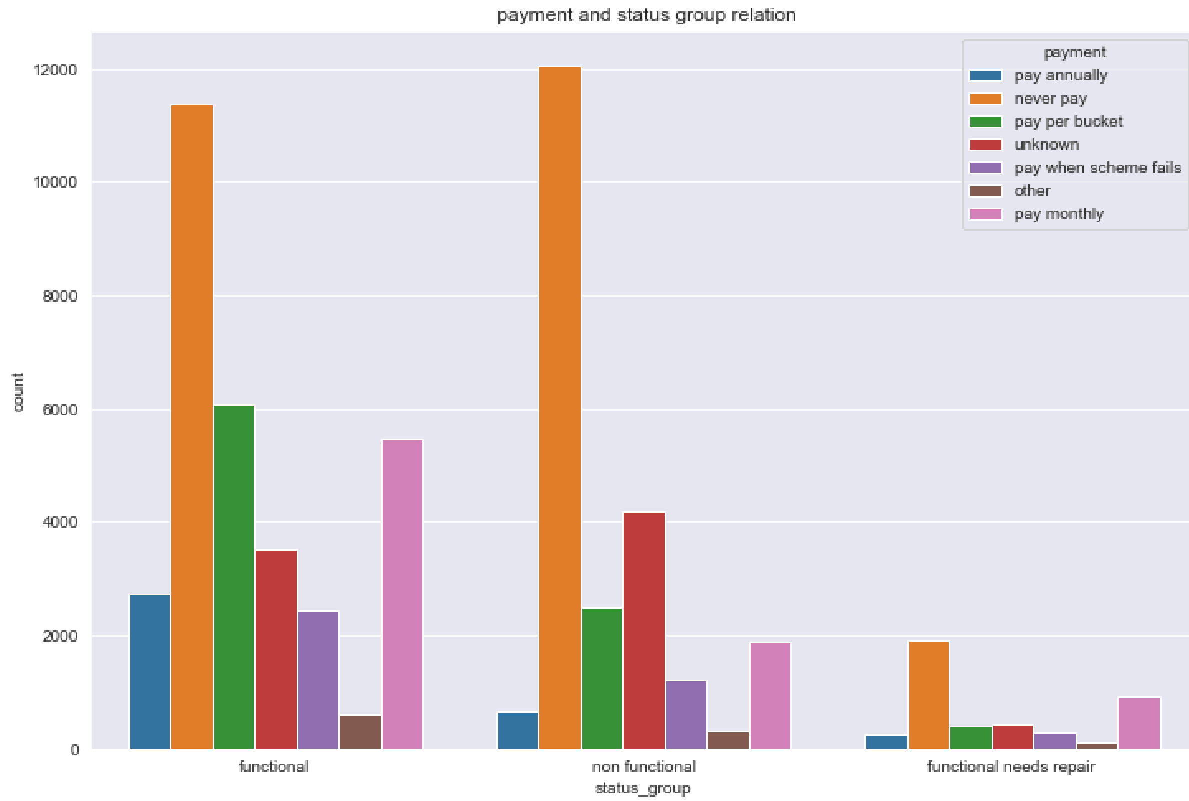


In [42]:
```
plt.figure(figsize=(12,8))
sns.barplot(x='status_group', y='amount_tsh', data=df_mode)
plt.title('Bar Plot of amount_tsh and status')
plt.xlabel('Status group')
plt.ylabel('Amount')
plt.show()
```



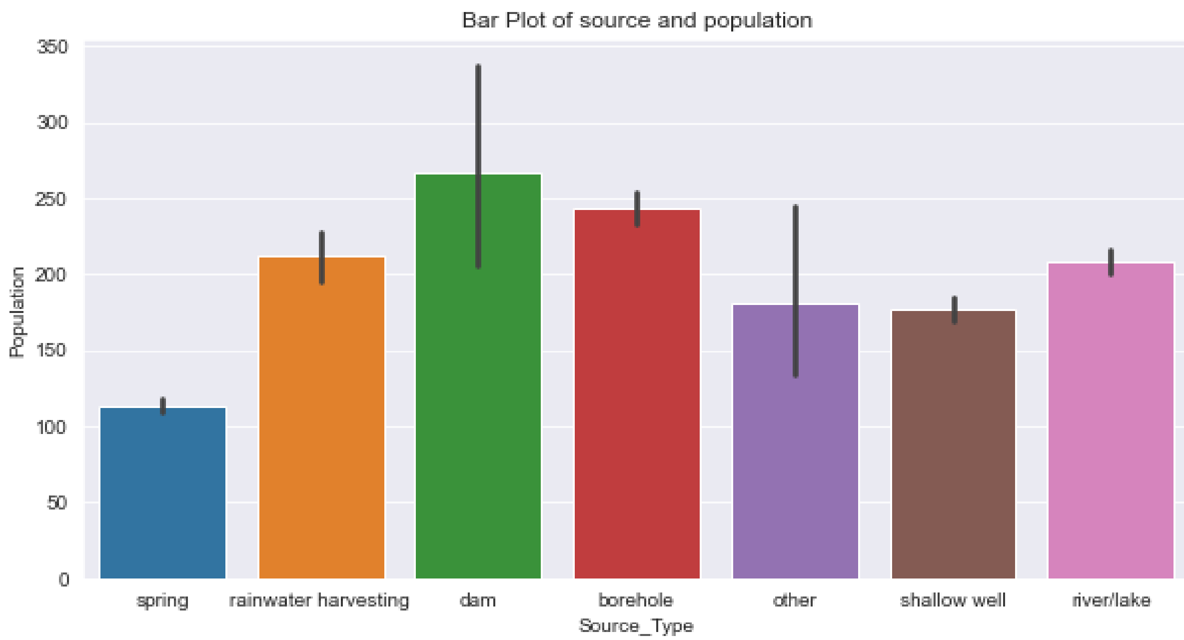
Relationship between payment and status_group

In [43]:
```
#payment and status group
ax, fig = plt.subplots(figsize=(12,8))
ax = sns.countplot(data=df_mode, x="status_group", hue="payment")
ax.set_title(" payment and status group relation")
sns.set_style("darkgrid")
```



Relationship between source type and population

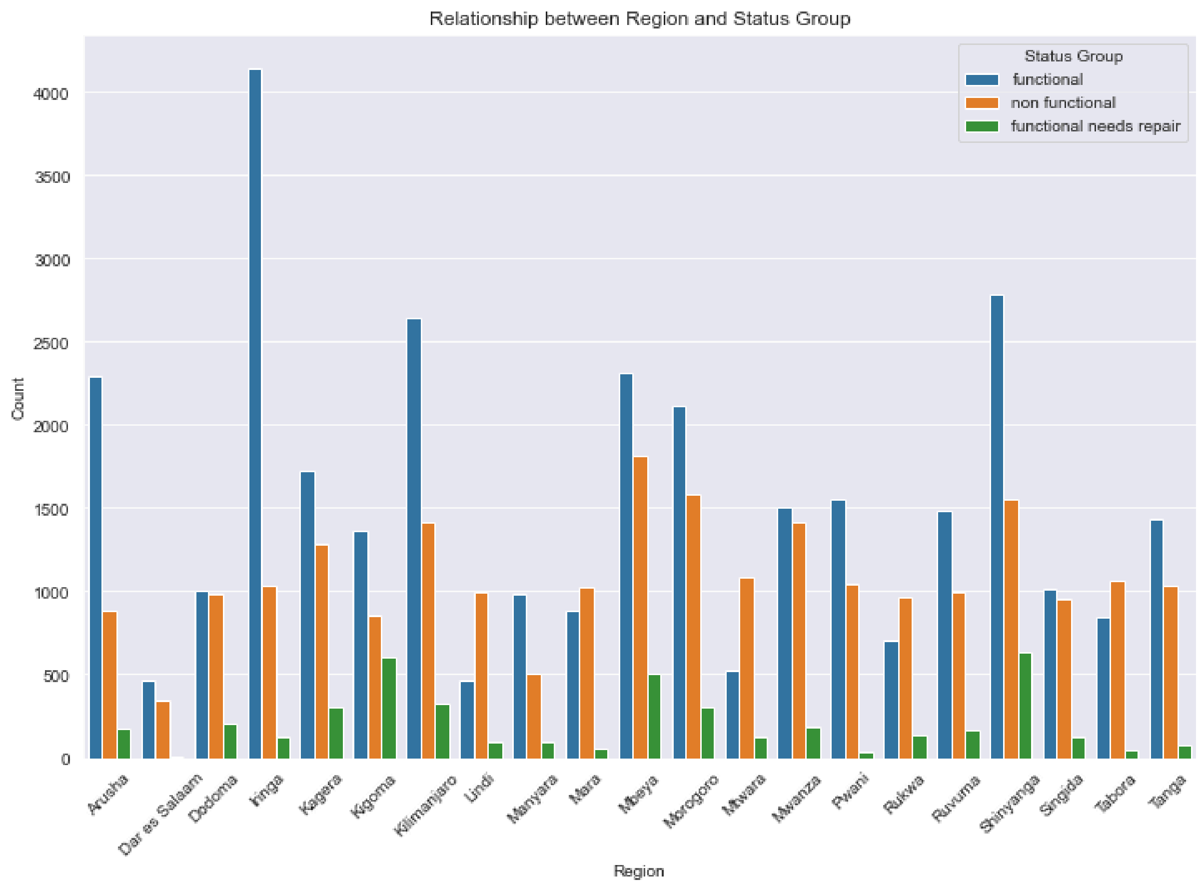
In [44]:
```
plt.figure(figsize=(10,5))
sns.barplot(x='source_type', y='population', data=df_mode)
plt.title('Bar Plot of source and population')
plt.xlabel('Source_Type')
plt.ylabel('Population')
plt.show()
```



Relationship between region and status group

In [45]:

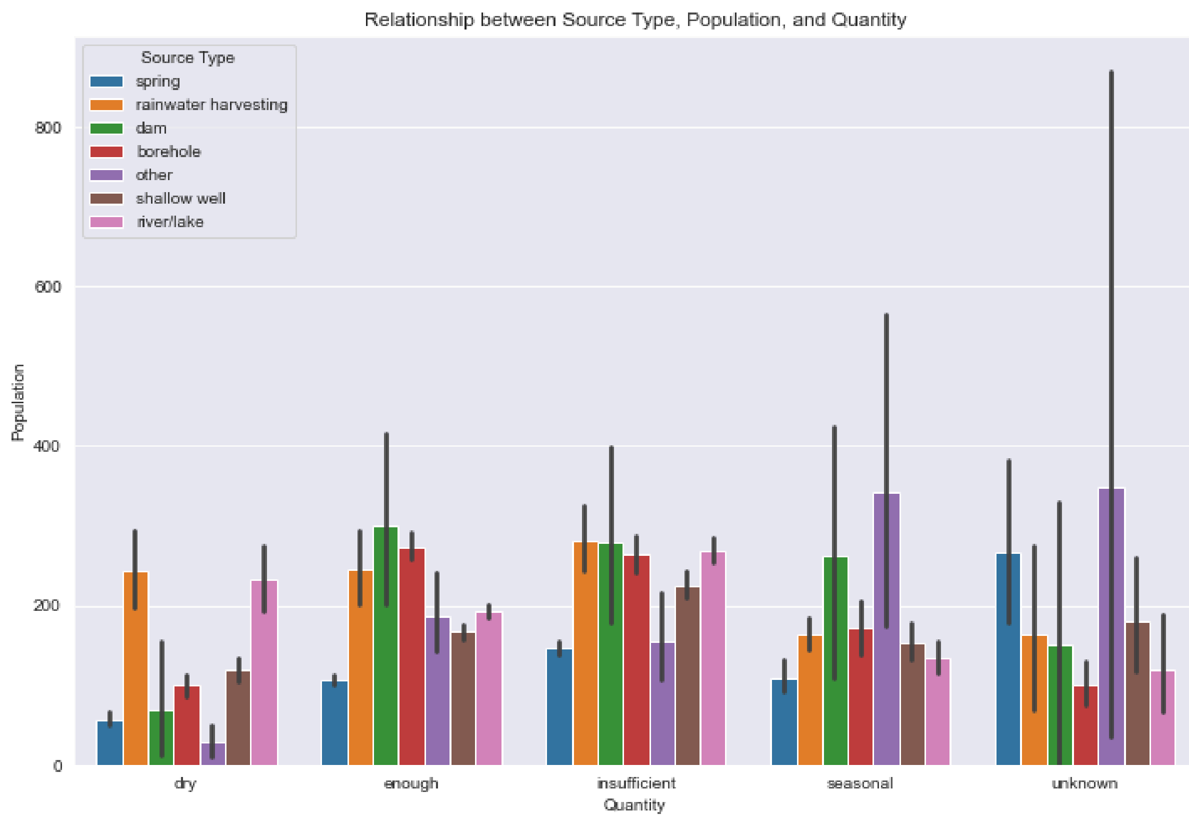
```
plt.figure(figsize=(12, 8))
sns.countplot(x='region', hue='status_group', data=df_mode)
plt.title('Relationship between Region and Status Group')
plt.xlabel('Region')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.legend(title='Status Group')
plt.show()
```



Mulitivariate Analysis

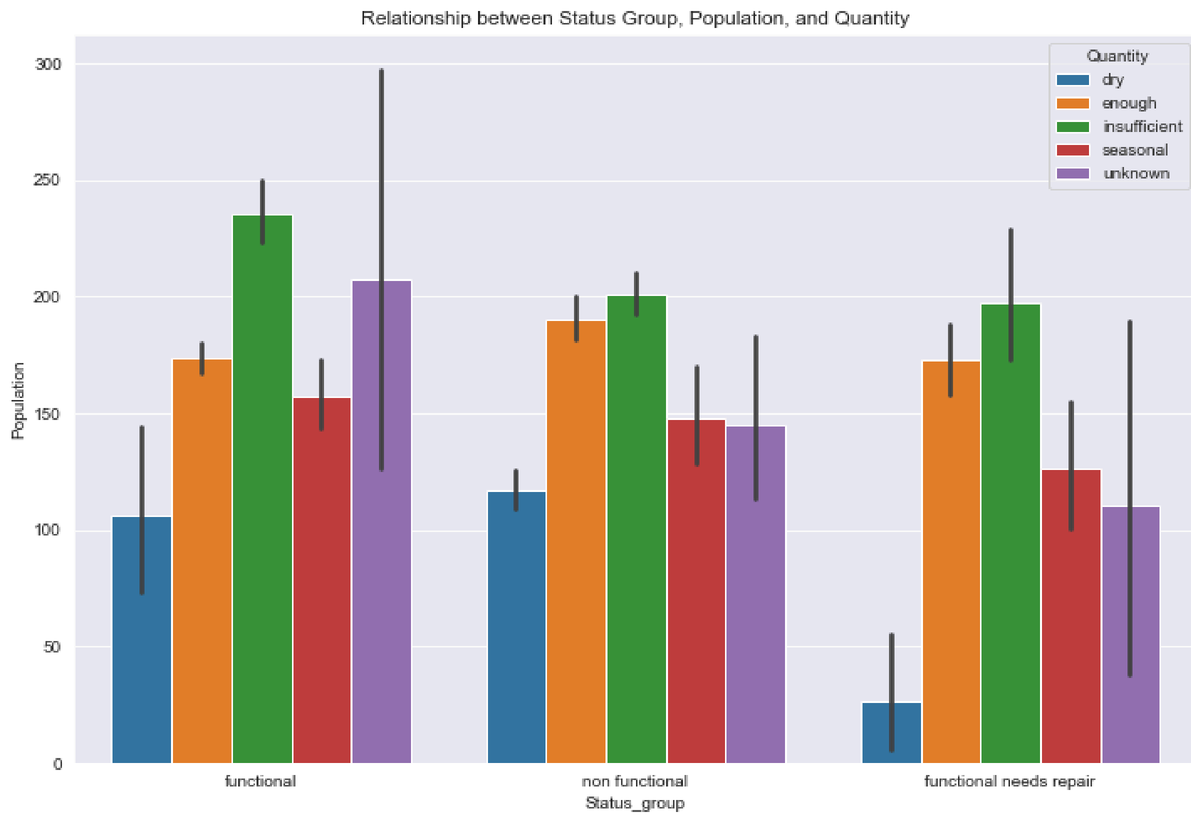
Relationship between source_type,population with respect to quantity

In [46]:
```
# Bar plot
plt.figure(figsize=(12, 8))
sns.barplot(x='quantity', y='population', hue='source_type', data=df_mode
plt.title('Relationship between Source Type, Population, and Quantity')
plt.xlabel('Quantity')
plt.ylabel('Population')
plt.legend(title='Source Type')
plt.show()
```



Relationship between status,population with respect to quantity

In [47]:
```
plt.figure(figsize=(12, 8))
sns.barplot(x='status_group', y='population', hue='quantity', data=df_mo
plt.title('Relationship between Status Group, Population, and Quantity')
plt.xlabel('Status_group')
plt.ylabel('Population')
plt.legend(title='Quantity')
plt.show()
```



Findings

1. Iringa is observed to be the region with the highest number of functional water points.
2. Many functional water points seem to be providing insuffficent amounts of water.
3. A lot of non-functional water sources are not paid for.
4. Factors that contribute functionality of the water sources include:Region, source type and water quality.
5. The functional water points wer the most populated.
6. Many wells are found in areas that are highly populated.

Modeling

Pre -Processing Data

This involves cleaning, transforming, and preparing the data to make it suitable for analysis or feeding into machine learning algorithms.

In [48]:
```
df_mode = train_df.copy()
df_mode.columns
```

Out[48]:
```
Index(['amount_tsh', 'date_recorded', 'funder', 'gps_height', 'installe
r',
       'longitude', 'latitude', 'wpt_name', 'basin', 'subvillage', 'reg
ion',
       'region_code', 'district_code', 'lga', 'ward', 'population',
       'public_meeting', 'scheme_management', 'scheme_name', 'permit',
       'construction_year', 'extraction_type', 'extraction_type_group',
       'extraction_type_class', 'management', 'management_group', 'paym
ent',
       'payment_type', 'water_quality', 'quality_group', 'quantity',
       'quantity_group', 'source', 'source_type', 'source_class',
       'waterpoint_type', 'waterpoint_type_group', 'status_group'],
      dtype='object')
```

Drop unnecessary columns that will not be used in modelling.

In [49]:
```
df_mode.drop(columns=['date_recorded','funder','latitude','longitude','su
```

In [50]:
```
df_mode.dtypes
```

Out[50]:
```
amount_tsh                  float64
gps_height                    int64
basin                      category
region                     category
region_code                category
district_code              category
population                    int64
permit                         bool
extraction_type              object
extraction_type_group        object
extraction_type_class        object
management                   object
management_group             object
payment                      object
payment_type               category
water_quality              category
quality_group                object
quantity                   category
quantity_group               object
source                     category
source_type                  object
source_class                 object
waterpoint_type            category
status_group                 object
dtype: object
```

We will go ahead and carry out train and test split

In [51]:
```
from sklearn.model_selection import train_test_split

# Define the target variable 'y'
y = df_mode['status_group']

# Define the feature variables 'X' by dropping the target column
X = df_mode.drop('status_group', axis=1)

# Perform train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

#  shapes of the resulting datasets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (47520, 23)
X_test shape: (11880, 23)
y_train shape: (47520,)
y_test shape: (11880,)
```

In [52]:
```
# One-hot encode categorical variables
df_encoded = pd.get_dummies(df_mode.drop('status_group', axis=1))

# Define the target variable 'y'
y = df_mode['status_group']

# Use the encoded DataFrame as features 'X'
X = df_encoded

# Perform train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# Display the shapes of the resulting datasets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (47520, 201)
X_test shape: (11880, 201)
y_train shape: (47520,)
y_test shape: (11880,)
```

Encoding

We will go ahead and create dummy variables or carry out one-hot-encoding for categorical columns in both datsets,the testing and training and ensure that both datsets have columns that are consistent.

In [53]:
```
# Create dummy variables for both training and testing data
X_train_encoded = pd.get_dummies(X_train)
X_test_encoded = pd.get_dummies(X_test)

# Align the columns of X_train and X_test to ensure they have the same co
X_train_encoded, X_test_encoded = X_train_encoded.align(X_test_encoded, 

# Display the shapes of the resulting datasets
print("X_train_encoded shape:", X_train_encoded.shape)
print("X_test_encoded shape:", X_test_encoded.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train_encoded shape: (47520, 201)
X_test_encoded shape: (11880, 201)
y_train shape: (47520,)
y_test shape: (11880,)
```

They have shown they have consistent columns so we will go ahead and start building and evaluating machine learning models.

Scaling

Important step especially for algorithms in machine learning that are sensitive to scale of the data i.e KNN, RandomForest.

In [54]:
```
# Instantiate the scaler
scaler = StandardScaler()

# Fit the scaler and transform the training data
X_train_scaled = scaler.fit_transform(X_train_encoded)

# Transform the test data
X_test_scaled = scaler.transform(X_test_encoded)

# Output the shapes to verify
print("X_train_scaled shape:", X_train_scaled.shape)
print("X_test_scaled shape:", X_test_scaled.shape)
```

```
X_train_scaled shape: (47520, 201)
X_test_scaled shape: (11880, 201)
```

'StandardScaler' standardizes features by removing the mean and scaling it to unit variance

Label Encoding

Involves converting categorical data into numerical values and here each unique category is assigned an interger value.

In [55]:
```
from sklearn.preprocessing import LabelEncoder
l_encoder = LabelEncoder()

# Fit encoder on  target variable and transform it
y_encoded = l_encoder.fit_transform(y_train)
y_encoded = l_encoder.transform(y_test)
```

In [56]:
```
# Fit encoder on the features in X
categorical_cols = X.select_dtypes(include=['category', 'object']).colum

# Apply label encoding to each categorical feature in the training and t
for col in categorical_cols:
    l_encoder = LabelEncoder()
    X_train[col] = l_encoder.fit_transform(X_train[col])
    X_test[col] = l_encoder.transform(X_test[col])
```

In [57]:
```
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder

# Instantiate SMOTE
smote = SMOTE(random_state=42)

# Use a smaller subset of the data for testing
X_train_subset = X_train_scaled[:10000]
y_train_subset = y_encoded[:10000]

# Apply SMOTE to the subset
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_subset

print(f"Shape of X_train_resampled: {X_train_resampled.shape}")
print(f"Shape of y_train_resampled: {y_train_resampled.shape}")
```

```
Shape of X_train_resampled: (16245, 201)
Shape of y_train_resampled: (16245,)
```

Let us look at the accuracy:

This section here is meant to determine most suitable machine learning model.

1. Logistic regression
2. Decision Tree
3. KNearest Neighbours
4. Random Forest

Building base models

In [58]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier

randomforest = RandomForestClassifier(random_state = 42)
knn = KNeighborsClassifier()
logistic_model = LogisticRegression(random_state = 42)
decisiontree = DecisionTreeClassifier(random_state = 42)
```

In [59]:
```
accuracy_train = []
accuracy_test = []
models = [randomforest,logistic_model, decisiontree, knn]
for i in models:
    i = i.fit(X_train_resampled, y_train_resampled)
    ytrain_pred = i.predict(X_train_resampled)
    ytest_pred = i.predict(X_test_scaled)
    accuracy_train.append(accuracy_score(ytrain_pred, y_train_resampled)
    accuracy_test.append(accuracy_score(ytest_pred, y_encoded))
```

```
/Users/mac/anaconda3/envs/learn-env/lib/python3.8/site-packages/sklear
n/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to co
nverge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html (http
s://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression (https://scikit-learn.org/stable/modules/linear_model.html#l
ogistic-regression)
  n_iter_i = _check_optimize_result(
```

In [61]:
```
accuracy_train
```

Out[61]:
```
[0.9378885811018774,
 0.42179132040627887,
 0.9378885811018774,
 0.723237919359803]
```

In [62]:
```
accuracy_test
```

Out[62]:
```
[0.4558922558922559,
 0.3538720538720539,
 0.4436026936026936,
 0.42163299663299664]
```

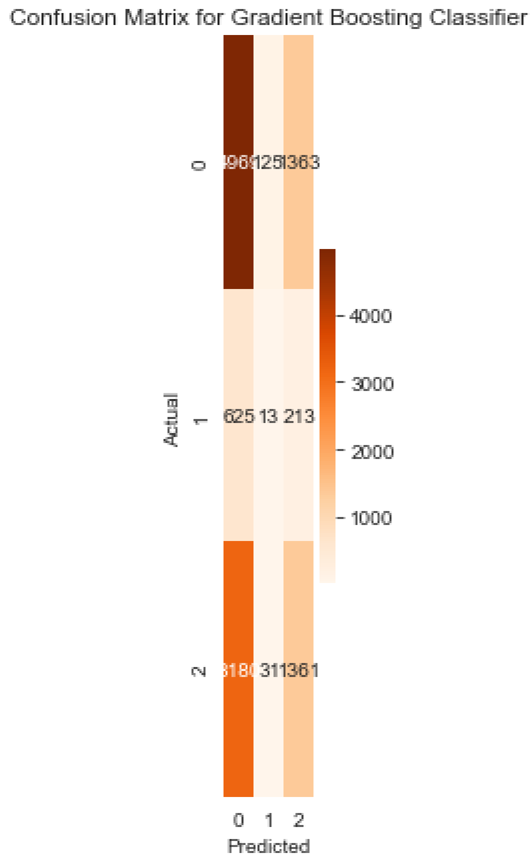
Accuracies test: RandomForest: 0.4559 (45.59%) LogisticRegression: 0.3539 (35.39%) DecisionTree: 0.4436 (44.36%) KNeighbors: 0.4216 (42.16%)

Accuracies Train: RandomForest: 0.9379 (93.79%) LogisticRegression: 0.4218 (42.18%) DecisionTree: 0.9379 (93.79%) KNeighbors: 0.7232 (72.32%)

This shows that our models are overfitting,underfitting and performing incosistently. We can improve perfromance of our models by implementing Gradient Boosting.

In [65]:
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report, confu
import seaborn as sns
import matplotlib.pyplot as plt

# train the Gradient Boosting Classifier
gradient_b = GradientBoostingClassifier(random_state=42)
gradient_b.fit(X_train_resampled, y_train_resampled)

# Predict on training and test sets
gradient_train_preds = gradient_b.predict(X_train_resampled)
gradient_test_preds = gradient_b.predict(X_test_scaled)

# Calculate accuracy for training and test sets
gradient_train_accuracy = accuracy_score(y_train_resampled, gradient_tra
gradient_test_accuracy = accuracy_score(y_encoded, gradient_test_preds)

print(f"Gradient Training Accuracy: {gradient_train_accuracy:.4f}")
print(f"Gradient Test Accuracy: {gradient_test_accuracy:.4f}")

# Classification report
print("Classification Report for Gradient Boosting on Test Set")
print(classification_report(y_encoded, gradient_test_preds))

# Plot confusion matrix
def plot_confusion_matrix(y_true, y_pred, model_name):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(1, 7))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Oranges')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'Confusion Matrix for {model_name}')
    plt.show()

plot_confusion_matrix(y_encoded, gradient_test_preds, 'Gradient Boosting
```

```
Gradient Training Accuracy: 0.6302
Gradient Test Accuracy: 0.5339
Classification Report for Gradient Boosting on Test Set
              precision    recall  f1-score   support

           0       0.57      0.77      0.65      6457
           1       0.08      0.02      0.03       851
           2       0.46      0.30      0.36      4572

    accuracy                           0.53     11880
   macro avg       0.37      0.36      0.35     11880
weighted avg       0.49      0.53      0.50     11880
```



Compared to previous models Gradient Boosting Classifier provides a more balanced performance in testing and training accuracy.

Training Accuracy: 63.02% Test Accuracy: 53.39%

In [66]:
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for Gradient Boosting
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.05],
    'max_depth': [3, 4, 5],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'subsample': [0.8, 1.0]
}

# Instantiate the Gradient Boosting Classifier
gradient_b = GradientBoostingClassifier(random_state=42)

# Perform Grid Search with Cross-Validation
grid_search = GridSearchCV(estimator=gradient_b, param_grid=param_grid,
grid_search.fit(X_train_resampled, y_train_resampled)

# Get the best parameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Cross-Validation Score:", best_score)

# Evaluate the best model on the test set
best_gradient_b_model = grid_search.best_estimator_
test_predictions = best_gradient_b_model.predict(X_test_scaled)
accuracy_test = accuracy_score(y_encoded, test_predictions)

print("Test Accuracy with Best Gradient Boosting Model:", accuracy_test)

# Classification report
print(classification_report(y_encoded, test_predictions))
```

Evaluation

The best working model in our case was Gradient Boosting since it training and testing accuracy values were not far apart as compared to other models. This model shows that there is balance between avoiding overfitting and underfitting .

Conclusion

Gradient Boosting model should be used because of balance.

Recommendations

1. The government of Tanzania and other top financiers being the main funders should be approached to fund for repairs and also add more water wells.
2. For the people to enjoy or have water the government should impose a rule where it will be mandatory for them to pay since it has being observed that there are many functional where they are paid for.

Improve the model for well pump prediction to optimize maintenance resource allocation.