Phase 3 Project

1. Project Setup

- Import necessary libraries (pandas, numpy, sklearn, matplotlib, seaborn)
- Load the dataset
- Set random seed for reproducibility

2. Data Exploration and Preprocessing

- Examine the dataset (head, info, describe)
- Check for missing values and handle them
- Explore data distributions and correlations
- Perform feature engineering if necessary
- Split the data into features (X) and target variable(s) (y)

3. Logistic Regression

- Prepare the data (ensure binary target variable)
- Create and train the model
- Make predictions on the test set
- Evaluate the model (accuracy, precision, recall, F1-score)
- Plot the ROC curve and calculate AUC
- Analyze coefficients and their significance

4. Decision Trees

- Prepare the data
- Create and train the model
- Make predictions on the test set

- Evaluate the model (accuracy, precision, recall, F1-score)
- Visualize the tree structure
- Analyze feature importance

5. Model Comparison and Conclusion

- Compare performance metrics across all models
- Discuss strengths and weaknesses of each approach
- Recommend the best model(s) for the problem at handk
- Summarize key findings
- Discuss limitations of the current approach
- Suggest potential improvements or additional models to try

Student details

Nduku Kitenge DSC-PT07

Business Problem

The goal of this project is to predict which pumps are functional, which need some repairs and which dont work at all.

The challenge from DrivenData. (2015). Pump it Up: Data Mining the Water Table. Retrieved [Month Day Year] from https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table with data from Taarifa and the Tanzanian Ministry of Water. The goal of this project is to predict one of these three classes based on a number of variables about what kind of pump is operating, when it was installed, and how it is managed. A smart understanding of which waterpoints will fail can improve maintenance operations and ensure that clean, potable water is available to communities across Tanzania.

1. Project Setup

- Import necessary libraries (pandas, numpy, sklearn, matplotlib, seaborn)
- Load the dataset

In [ ]:
```
#import neccessary libraries
import pandas as pd
```

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.feature_selection import RFE
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
```

In [ ]:
```python
#Load training dataset
sub_merged_df = pd.read_csv('./SubmissionFormat.csv',index_col=0)
training_merged_df = pd.read_csv('./TrainingSetValues.csv',index_col=0)
test_merged_df = pd.read_csv('./TestSetValues.csv',index_col=0)
t_label_merged_df = pd.read_csv('TrainingSetLabels.csv',index_col=0)
```

2. Data Exploration and Preprocessing

- Examine the dataset (head, info, describe)
- Check for missing values and handle them
- Explore data distributions and correlations
- Perform feature engineering eg. add pump age
- Split the data into features (X) and target variable(s) (y)

In [ ]:
```python
# Function to display dataset info
def display_dataset_info(merged_df, name):
    print(f"\n=== {name} ===")
    print(f"Shape: {merged_df.shape}")
    print("\nInfo:")
    print(merged_df.info())
    print("\nDescription:")
    print(merged_df.describe())
    print("\nHead:")
    print(merged_df.head())
```

```
    print("\n" + "="*40)

# Display info for each dataset
display_dataset_info(sub_merged_df, "Submission Format")
display_dataset_info(training_merged_df, "Training Set Values")
display_dataset_info(test_merged_df, "Test Set Values")
display_dataset_info(t_label_merged_df, "Training Set Labels")
```

```
=== Submission Format ===
Shape: (14850, 1)

Info:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14850 entries, 50785 to 68707
Data columns (total 1 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   status_group  14850 non-null  object
dtypes: object(1)
memory usage: 232.0+ KB
None

Description:
           status_group
count             14850
unique                1
top     predicted label
freq              14850

Head:
          status_group
id                    
50785  predicted label
51630  predicted label
17168  predicted label
45559  predicted label
49871  predicted label

========================================

=== Training Set Values ===
Shape: (59400, 39)

Info:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59400 entries, 69572 to 26348
Data columns (total 39 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
```

```
 0   amount_tsh             59400 non-null  float64
 1   date_recorded          59400 non-null  object
 2   funder                 55765 non-null  object
 3   gps_height             59400 non-null  int64
 4   installer              55745 non-null  object
 5   longitude              59400 non-null  float64
 6   latitude               59400 non-null  float64
 7   wpt_name               59400 non-null  object
 8   num_private            59400 non-null  int64
 9   basin                  59400 non-null  object
 10  subvillage             59029 non-null  object
 11  region                 59400 non-null  object
 12  region_code            59400 non-null  int64
 13  district_code          59400 non-null  int64
 14  lga                    59400 non-null  object
 15  ward                   59400 non-null  object
 16  population             59400 non-null  int64
 17  public_meeting         56066 non-null  object
 18  recorded_by            59400 non-null  object
 19  scheme_management      55523 non-null  object
 20  scheme_name            31234 non-null  object
 21  permit                 56344 non-null  object
 22  construction_year      59400 non-null  int64
 23  extraction_type        59400 non-null  object
 24  extraction_type_group  59400 non-null  object
 25  extraction_type_class  59400 non-null  object
 26  management             59400 non-null  object
 27  management_group       59400 non-null  object
 28  payment                59400 non-null  object
 29  payment_type           59400 non-null  object
 30  water_quality          59400 non-null  object
 31  quality_group          59400 non-null  object
 32  quantity               59400 non-null  object
 33  quantity_group         59400 non-null  object
 34  source                 59400 non-null  object
 35  source_type            59400 non-null  object
 36  source_class           59400 non-null  object
 37  waterpoint_type        59400 non-null  object
 38  waterpoint_type_group  59400 non-null  object
dtypes: float64(3), int64(6), object(30)
memory usage: 18.1+ MB
None

Description:
          amount_tsh    gps_height     longitude      latitude   num_private  \
count   59400.000000  59400.000000  59400.000000  5.940000e+04  59400.000000
mean      317.650385    668.297239     34.077427 -5.706033e+00      0.474141
std      2997.574558    693.116350      6.567432  2.946019e+00     12.236230
min         0.000000    -90.000000      0.000000 -1.164944e+01      0.000000
```

```
25%         0.000000      0.000000     33.090347 -8.540621e+00      0.000000
50%         0.000000    369.000000     34.908743 -5.021597e+00      0.000000
75%        20.000000   1319.250000     37.178387 -3.326156e+00      0.000000
max    350000.000000   2770.000000     40.345193 -2.000000e-08   1776.000000

        region_code  district_code    population  construction_year
count  59400.000000   59400.000000  59400.000000       59400.000000
mean      15.297003       5.629747    179.909983        1300.652475
std       17.587406       9.633649    471.482176         951.620547
min        1.000000       0.000000      0.000000           0.000000
25%        5.000000       2.000000      0.000000           0.000000
50%       12.000000       3.000000     25.000000        1986.000000
75%       17.000000       5.000000    215.000000        2004.000000
max       99.000000      80.000000  30500.000000        2013.000000

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

       longitude   latitude              wpt_name  num_private  \
id
69572  34.938093  -9.856322                  none            0
8776   34.698766  -2.147466              Zahanati            0
34310  37.460664  -3.821329           Kwa Mahundi            0
67743  38.486161 -11.155298  Zahanati Ya Nanyumbu            0
19728  31.130847  -1.825359               Shuleni            0

                         basin  ... payment_type water_quality  quality_group  \
id                              ...
69572               Lake Nyasa  ...     annually          soft           good
8776             Lake Victoria  ...    never pay          soft           good
34310                  Pangani  ...   per bucket          soft           good
67743  Ruvuma / Southern Coast  ...    never pay          soft           good
19728            Lake Victoria  ...    never pay          soft           good

           quantity quantity_group                source  \
id
69572        enough         enough                spring
8776   insufficient   insufficient  rainwater harvesting
34310        enough         enough                   dam
67743           dry            dry           machine dbh
19728      seasonal       seasonal  rainwater harvesting

                source_type source_class              waterpoint_type  \
```

```
id
69572                  spring  groundwater            communal standpipe
8776   rainwater harvesting      surface            communal standpipe
34310                     dam      surface  communal standpipe multiple
67743                borehole  groundwater  communal standpipe multiple
19728  rainwater harvesting      surface            communal standpipe

      waterpoint_type_group
id
69572    communal standpipe
8776     communal standpipe
34310    communal standpipe
67743    communal standpipe
19728    communal standpipe

[5 rows x 39 columns]

========================================

=== Test Set Values ===
Shape: (14850, 39)

Info:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14850 entries, 50785 to 68707
Data columns (total 39 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   amount_tsh             14850 non-null  float64
 1   date_recorded          14850 non-null  object
 2   funder                 13981 non-null  object
 3   gps_height             14850 non-null  int64
 4   installer              13973 non-null  object
 5   longitude              14850 non-null  float64
 6   latitude               14850 non-null  float64
 7   wpt_name               14850 non-null  object
 8   num_private            14850 non-null  int64
 9   basin                  14850 non-null  object
 10  subvillage             14751 non-null  object
 11  region                 14850 non-null  object
 12  region_code            14850 non-null  int64
 13  district_code          14850 non-null  int64
 14  lga                    14850 non-null  object
 15  ward                   14850 non-null  object
 16  population             14850 non-null  int64
 17  public_meeting         14029 non-null  object
 18  recorded_by            14850 non-null  object
 19  scheme_management      13881 non-null  object
 20  scheme_name            7758 non-null   object
```

```
 21  permit                 14113 non-null  object
 22  construction_year      14850 non-null  int64
 23  extraction_type        14850 non-null  object
 24  extraction_type_group  14850 non-null  object
 25  extraction_type_class  14850 non-null  object
 26  management             14850 non-null  object
 27  management_group       14850 non-null  object
 28  payment                14850 non-null  object
 29  payment_type           14850 non-null  object
 30  water_quality          14850 non-null  object
 31  quality_group          14850 non-null  object
 32  quantity               14850 non-null  object
 33  quantity_group         14850 non-null  object
 34  source                 14850 non-null  object
 35  source_type            14850 non-null  object
 36  source_class           14850 non-null  object
 37  waterpoint_type        14850 non-null  object
 38  waterpoint_type_group  14850 non-null  object
dtypes: float64(3), int64(6), object(30)
memory usage: 4.5+ MB
None

Description:
          amount_tsh    gps_height     longitude      latitude   num_private  \
count   14850.000000  14850.000000  14850.000000  1.485000e+04  14850.000000
mean      322.826983    655.147609     34.061605 -5.684724e+00      0.415084
std      2510.968644    691.261185      6.593034  2.940803e+00      8.167910
min         0.000000    -57.000000      0.000000 -1.156459e+01      0.000000
25%         0.000000      0.000000     33.069455 -8.443970e+00      0.000000
50%         0.000000    344.000000     34.901215 -5.049750e+00      0.000000
75%        25.000000   1308.000000     37.196594 -3.320594e+00      0.000000
max    200000.000000   2777.000000     40.325016 -2.000000e-08    669.000000

        region_code  district_code    population  construction_year
count  14850.000000   14850.000000  14850.000000       14850.000000
mean      15.139057       5.626397    184.114209        1289.708350
std       17.191329       9.673842    469.499332         955.241087
min        1.000000       0.000000      0.000000           0.000000
25%        5.000000       2.000000      0.000000           0.000000
50%       12.000000       3.000000     20.000000        1986.000000
75%       17.000000       5.000000    220.000000        2004.000000
max       99.000000      80.000000  11469.000000        2013.000000

Head:
       amount_tsh date_recorded                 funder  gps_height  \
id
50785         0.0    2013-02-04                   Dmdd        1996
51630         0.0    2013-02-04  Government Of Tanzania        1569
17168         0.0    2013-02-01                    NaN        1567
```

```
45559         0.0    2013-01-22             Finn Water         267
49871       500.0    2013-03-27                 Bruder        1260

        installer  longitude   latitude                 wpt_name  num_private  \
id
50785        DMDD  35.290799  -4.059696  Dinamu Secondary School            0
51630         DWE  36.656709  -3.309214                  Kimnyak            0
17168         NaN  34.767863  -5.004344           Puma Secondary            0
45559  FINN WATER  38.058046  -9.418672           Kwa Mzee Pange            0
49871      BRUDER  35.006123 -10.950412          Kwa Mzee Turuka            0

                         basin  ... payment_type water_quality  quality_group  \
id                              ...
50785                 Internal  ...    never pay          soft           good
51630                  Pangani  ...    never pay          soft           good
17168                 Internal  ...    never pay          soft           good
45559  Ruvuma / Southern Coast  ...      unknown          soft           good
49871  Ruvuma / Southern Coast  ...      monthly          soft           good

           quantity quantity_group               source  \
id
50785      seasonal       seasonal  rainwater harvesting
51630  insufficient   insufficient                spring
17168  insufficient   insufficient  rainwater harvesting
45559           dry            dry          shallow well
49871        enough         enough                spring

                source_type source_class     waterpoint_type  \
id
50785  rainwater harvesting      surface               other
51630                spring  groundwater  communal standpipe
17168  rainwater harvesting      surface               other
45559          shallow well  groundwater               other
49871                spring  groundwater  communal standpipe

      waterpoint_type_group
id
50785                 other
51630    communal standpipe
17168                 other
45559                 other
49871    communal standpipe

[5 rows x 39 columns]

========================================

=== Training Set Labels ===
Shape: (59400, 1)
```

```
Info:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59400 entries, 69572 to 26348
Data columns (total 1 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   status_group  59400 non-null  object
dtypes: object(1)
memory usage: 928.1+ KB
None

Description:
       status_group
count         59400
unique            3
top      functional
freq          32259

Head:
         status_group
id                   
69572      functional
8776       functional
34310      functional
67743  non functional
19728      functional

========================================
```

In [ ]:

```
#check for missing values in training dataset
training_merged_df.isna().sum()
```

Out[ ]:

```
amount_tsh                   0
date_recorded                0
funder                    3635
gps_height                   0
installer                 3655
longitude                    0
latitude                     0
wpt_name                     0
num_private                  0
basin                        0
subvillage                 371
region                       0
region_code                  0
district_code                0
lga                          0
ward                         0
```

```
population                   0
public_meeting            3334
recorded_by                  0
scheme_management         3877
scheme_name              28166
permit                    3056
construction_year            0
extraction_type              0
extraction_type_group        0
extraction_type_class        0
management                   0
management_group             0
payment                      0
payment_type                 0
water_quality                0
quality_group                0
quantity                     0
quantity_group               0
source                       0
source_type                  0
source_class                 0
waterpoint_type              0
waterpoint_type_group        0
dtype: int64
```

Dropping columns and reasons why:

- date_recorded: since there is a contruction year, this is not needed to make a prediction on well functionality.- funder: some rows have matching values with installer column, and many more rows only have different spellings of the same entry. The installer has a higher influence on the functionality of a well.
- wpt_name: too many unique values
- subvillage: can be represented by the region
- lga: can be represented by the region
- ward: can be represented by the region
- recorded_by: it has the same value for all rows
- scheme_name: too many unique values and nulls
- extraction_type: too similar to extraction type class
- extraction_type_group: too similar to extraction type class
- management: management_group are categories of management so I will drop management
- payment: payment and payment_type are duplicates, so I will drop payment
- quality_group: correlated with water quality. I'll retain water quality since it has unique rows that need to be dropped

- quantity: quantity_group are categories of quantity, so I will drop quantity.
- source:can be source class.
- source_type: can be source class.
- waterpoint_type: similar to waterpoint_type_group.

Numerical columns:

- num private: dont know what this means.
- region_code: a duplicate of region. 3.) district_code: can be the region.

Observations

- Funder and installer are similar, but funder has less null values than installer.
- Waterpoint type and water point are similar - Will drop waterpoint type.

In [ ]:
```
#Check categorical columns
training_merged_df.select_dtypes(include=['object']).columns
```

Out[ ]:
```
Index(['date_recorded', 'funder', 'installer', 'wpt_name', 'basin',
       'subvillage', 'region', 'lga', 'ward', 'public_meeting', 'recorded_by',
       'scheme_management', 'scheme_name', 'permit', 'extraction_type',
       'extraction_type_group', 'extraction_type_class', 'management',
       'management_group', 'payment', 'payment_type', 'water_quality',
       'quality_group', 'quantity', 'quantity_group', 'source', 'source_type',
       'source_class', 'waterpoint_type', 'waterpoint_type_group'],
      dtype='object')
```

In [ ]:
```
#check numerical columns
training_merged_df.select_dtypes(include=['int64', 'float64']).columns
```

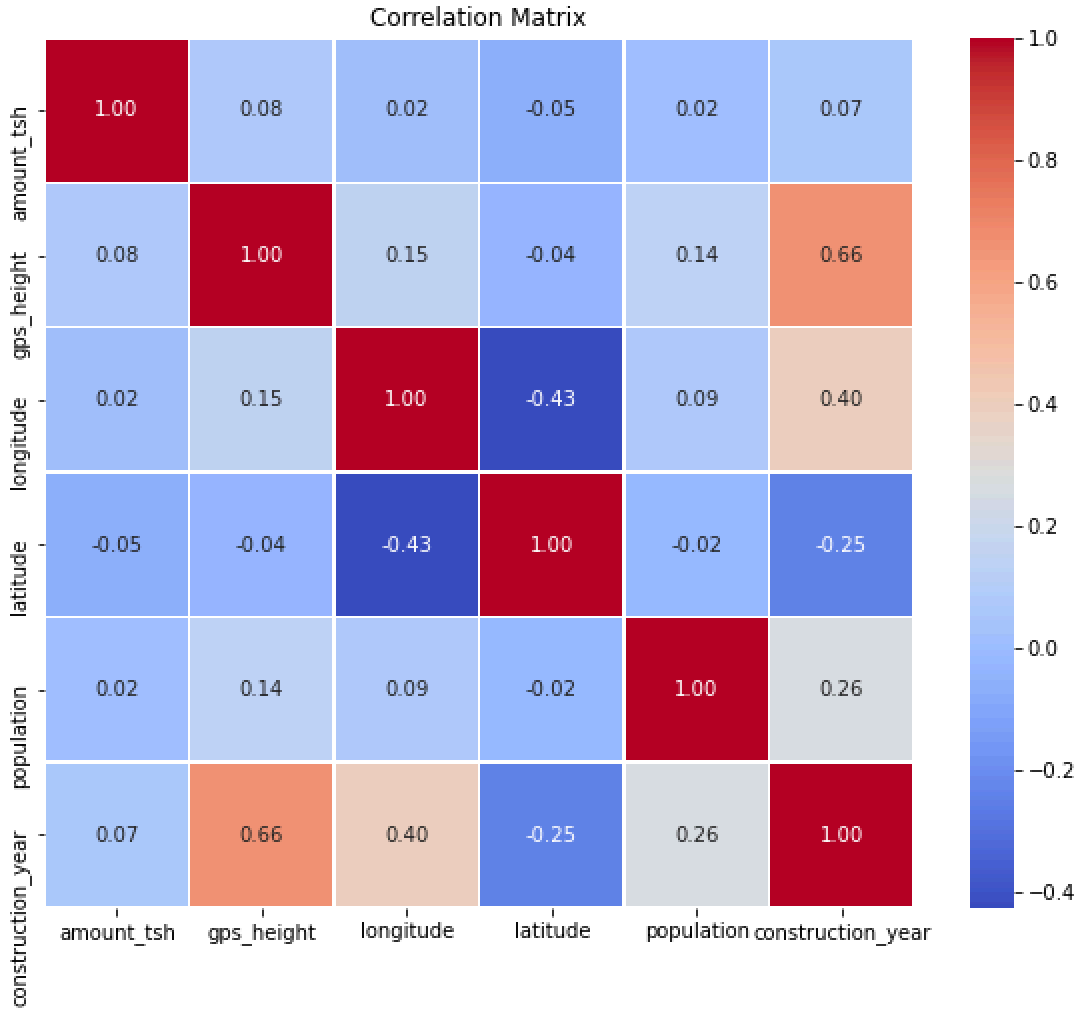
Out[ ]:
```
Index(['amount_tsh', 'gps_height', 'longitude', 'latitude', 'num_private',
       'region_code', 'district_code', 'population', 'construction_year'],
      dtype='object')
```

In [ ]:
```
#Define the list of columns to drop
columns_drop = ['date_recorded', 'funder', 'wpt_name', 'subvillage', 'lga',
 'ward', 'recorded_by', 'scheme_name', 'extraction_type',
 'extraction_type_group', 'management', 'payment', 'quality_group',
 'quantity', 'source', 'source_type', 'waterpoint_type', 'num_private',
 'region_code', 'district_code']

# Drop unnecessary columns
training_merged_df = training_merged_df.drop(columns_drop,axis=1)
```

```
#Print remaining columns
print(training_merged_df.columns)
```

```
Index(['amount_tsh', 'gps_height', 'installer', 'longitude', 'latitude',
       'basin', 'region', 'population', 'public_meeting', 'scheme_management',
       'permit', 'construction_year', 'extraction_type_class',
       'management_group', 'payment_type', 'water_quality', 'quantity_group',
       'source_class', 'waterpoint_type_group'],
      dtype='object')
```

In [ ]:
```
#explore data distrubitions and correlations of the numerical- training data
corr_matrix = training_merged_df.corr()

# Plot the correlation matrix using Seaborn for numerical values.
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm', square=True, linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



In [ ]:

```python
# merge with training labels data using ID as connector
merged_df = pd.merge(training_merged_df, t_label_merged_df, on='id', how='outer', indicator=True)
print(merged_df.head(20))
```

```
       amount_tsh  gps_height          installer  longitude   latitude  \
id
69572      6000.0        1390              Roman  34.938093  -9.856322
8776          0.0        1399            GRUMETI  34.698766  -2.147466
34310        25.0         686       World vision  37.460664  -3.821329
67743         0.0         263             UNICEF  38.486161 -11.155298
19728         0.0           0            Artisan  31.130847  -1.825359
9944         20.0           0                DWE  39.172796  -4.765587
19816         0.0           0               DWSP  33.362410  -3.766365
```

```
54551        0.0          0                   DWE  32.620617  -4.226198
53934        0.0          0             Water Aid  32.711100  -5.146712
46144        0.0          0               Artisan  30.626991  -1.257051
49056        0.0         62               Private  39.209518  -7.034139
50409      200.0       1062                DANIDA  35.770258 -10.574175
36957        0.0          0          World vision  33.798106  -3.290194
50495        0.0       1368  Lawatefuka water sup  37.092574  -3.181783
53752        0.0          0                WEDECO  34.364073  -3.629333
61848        0.0       1645                   DWE  31.444121  -8.274962
48451      500.0       1703                   DWE  34.642439  -9.106185
58155        0.0       1656                   DWE  34.569266  -9.085515
34169        0.0       1162                   DWE  32.920154  -1.947868
18274      500.0       1763                 Danid  34.508967  -9.894412

                         basin       region  population public_meeting  \
id
69572               Lake Nyasa       Iringa         109           True
8776              Lake Victoria        Mara         280            NaN
34310                  Pangani      Manyara         250           True
67743  Ruvuma / Southern Coast       Mtwara          58           True
19728            Lake Victoria       Kagera           0           True
9944                   Pangani        Tanga           1           True
19816                 Internal    Shinyanga           0           True
54551          Lake Tanganyika    Shinyanga           0           True
53934          Lake Tanganyika       Tabora           0           True
46144            Lake Victoria       Kagera           0           True
49056              Wami / Ruvu        Pwani         345           True
50409               Lake Nyasa       Ruvuma         250           True
36957                 Internal    Shinyanga           0           True
50495                  Pangani  Kilimanjaro           1           True
53752                 Internal    Shinyanga           0           True
61848          Lake Tanganyika        Rukwa         200           True
48451                   Rufiji       Iringa          35           True
58155                   Rufiji       Iringa          50           True
34169            Lake Victoria       Mwanza        1000            NaN
18274               Lake Nyasa       Iringa           1           True

      scheme_management  ... construction_year  extraction_type_class  \
id                       ...
69572               VWC  ...              1999                gravity
8776              Other  ...              2010                gravity
34310               VWC  ...              2009                gravity
67743               VWC  ...              1986            submersible
19728               NaN  ...                 0                gravity
9944                VWC  ...              2009            submersible
19816               VWC  ...                 0               handpump
54551               NaN  ...                 0               handpump
53934               VWC  ...                 0               handpump
46144               NaN  ...                 0               handpump
```

```
49056  Private operator  ...             2011           submersible
50409               WUG  ...             1987              handpump
36957               WUG  ...                0              handpump
50495       Water Board  ...             2009               gravity
53752               WUG  ...                0              handpump
61848               VWC  ...             1991              handpump
48451               WUA  ...             1978               gravity
58155               WUA  ...             1978               gravity
34169               NaN  ...             1999                 other
18274               VWC  ...             1992               gravity

      management_group payment_type water_quality quantity_group source_class  \
id
69572       user-group     annually          soft         enough  groundwater
8776        user-group    never pay          soft   insufficient      surface
34310       user-group   per bucket          soft         enough      surface
67743       user-group    never pay          soft            dry  groundwater
19728            other    never pay          soft       seasonal      surface
9944        user-group   per bucket         salty         enough      unknown
19816       user-group    never pay          soft         enough  groundwater
54551       user-group      unknown         milky         enough  groundwater
53934       user-group    never pay         salty       seasonal  groundwater
46144       user-group    never pay          soft         enough  groundwater
49056       commercial    never pay         salty         enough  groundwater
50409       user-group   on failure          soft   insufficient  groundwater
36957       user-group        other          soft         enough  groundwater
50495       user-group      monthly          soft         enough  groundwater
53752       user-group    never pay          soft         enough  groundwater
61848       user-group    never pay          soft         enough  groundwater
48451       user-group      monthly          soft            dry      surface
58155       user-group   on failure          soft            dry      surface
34169       user-group    never pay         milky   insufficient  groundwater
18274       user-group     annually          soft         enough  groundwater

      waterpoint_type_group             status_group _merge
id
69572    communal standpipe               functional   both
8776     communal standpipe               functional   both
34310    communal standpipe               functional   both
67743    communal standpipe           non functional   both
19728    communal standpipe               functional   both
9944     communal standpipe               functional   both
19816             hand pump           non functional   both
54551             hand pump           non functional   both
53934             hand pump           non functional   both
46144             hand pump               functional   both
49056                 other               functional   both
50409             hand pump               functional   both
36957             hand pump               functional   both
```

```
50495    communal standpipe                functional    both
53752             hand pump                functional    both
61848             hand pump                functional    both
48451    communal standpipe            non functional    both
58155    communal standpipe            non functional    both
34169                 other  functional needs repair    both
18274    communal standpipe                functional    both

[20 rows x 21 columns]
```

In [ ]:
```
#check for missing values in merged dataset
merged_df.isna().sum()
```

Out[ ]:
```
amount_tsh                  0
gps_height                  0
installer                3655
longitude                   0
latitude                    0
basin                       0
region                      0
population                  0
public_meeting           3334
scheme_management        3877
permit                   3056
construction_year           0
extraction_type_class       0
management_group            0
payment_type                0
water_quality               0
quantity_group              0
source_class                0
waterpoint_type_group       0
status_group                0
_merge                      0
dtype: int64
```

In [ ]:
```
#Treat null values
missing_value_columns = ['installer', 'public_meeting', 'scheme_management', 'permit']

# Check the value counts
for col in missing_value_columns:
    print(merged_df[col].value_counts())
```

```
DWE                 17402
Government           1825
RWE                  1206
Commu                1060
DANIDA               1050
```

```
                    ...
TCRS /DWE               1
WAMA                    1
Misri Government        1
Chacha                  1
George mtoto            1
Name: installer, Length: 2145, dtype: int64
True     51011
False     5055
Name: public_meeting, dtype: int64
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
True     38852
False    17492
Name: permit, dtype: int64
```

In [ ]:

```
# Remove rows with missing values in 'funder', 'installer' and 'scheme_management' columns
merged_df.dropna(subset=['installer', 'scheme_management'], axis=0, inplace=True)
```

Replacing the missing values for public meeting and permit with False. Assuming that the information doesnt exist.

In [ ]:

```
# Fill missing values in public meeting and permit'
for col in ['public_meeting', 'permit']:
    merged_df[col] = merged_df[col].fillna(False)
```

Remove dimensionality of unique values in installer column:

In [ ]:

```
# Replace close variations and misspellings in the installer column

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('Central government', 'Tanzania Government',
                                          'Cental Government','Tanzania government','Cebtral Government',
                                          'Centra Government', 'central government', 'CENTRAL GOVERNMENT',
                                          'TANZANIA GOVERNMENT', 'TANZANIAN GOVERNMENT', 'Central govt',
                                          'Centr', 'Centra govt', 'Tanzanian Government', 'Tanzania',
                                          'Tanz', 'Tanza', 'GOVERNMENT',
                                          'GOVER', 'GOVERNME', 'GOVERM', 'GOVERN', 'Gover', 'Gove',
```

```
                                           'Governme', 'Governmen', 'Got', 'Serikali', 'Serikari', 'Government',
                                           'Central Government'),
                                           value = 'Central Government')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('IDARA', 'Idara ya maji', 'MINISTRY OF WATER',
                                           'Ministry of water', 'Ministry of water engineer', 'MINISTRYOF WATER',
                                           'MWE &', 'MWE', 'Wizara ya maji', 'WIZARA', 'wizara ya maji',
                                           'Ministry of Water'),
                                           value ='Ministry of Water')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('District COUNCIL', 'DISTRICT COUNCIL',
                                           'Counc','District council','District Counci',
                                           'Council', 'COUN', 'Distri', 'Halmashauri ya wilaya',
                                           'Halmashauri wilaya', 'District Council'),
                                           value = 'District  Council')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('District water depar', 'District Water Department'
                                           'District water department', 'Distric Water Department'),
                                           value = 'District Water Department')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('villigers', 'villager', 'villagers', 'Villa', 'Vil
                                           'Villi', 'Village Council', 'Village Counil', 'Villages', 'Vill',
                                           'Village community', 'Villaers', 'Village Community', 'Villag',
                                           'Villege Council', 'Village council', 'Villege Council', 'Villagerd',
                                           'Villager', 'VILLAGER', 'Villagers',  'Villagerd', 'Village Technician',
                                           'Village water attendant', 'Village Office', 'VILLAGE COUNCIL',
                                           'VILLAGE COUNCIL .ODA', 'VILLAGE COUNCIL Orpha', 'Village community members',
                                           'VILLAG', 'VILLAGE', 'Village Government', 'Village government',
                                           'Village Govt', 'Village govt', 'VILLAGERS', 'VILLAGE WATER COMMISSION',
                                           'Village water committee', 'Commu', 'Communit', 'commu', 'COMMU', 'COMMUNITY',
                                            'Comunity', 'Communit', 'Kijiji', 'Serikali ya kijiji', 'Community'),
                                           value ='Community')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('FinW', 'Fini water', 'FINI WATER', 'FIN WATER',
                                           'Finwater', 'FINN WATER', 'FinW', 'FW', 'FinWater', 'FiNI WATER',
                                           'FinWate', 'FINLAND', 'Fin Water', 'Finland Government'),
                                           value ='Finnish Government')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('RC CHURCH', 'RC Churc', 'RC', 'RC Ch', 'RC C', 'RC
                                           'RC church', 'RC CATHORIC', 'Roman Church', 'Roman Catholic',
                                           'Roman catholic', 'Roman Ca', 'Roman', 'Romam', 'Roma',
                                           'ROMAN CATHOLIC', 'Kanisa', 'Kanisa katoliki'),
                                           value ='Roman Catholic Church')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('Dmdd', 'DMDD'), value ='DMDD')
```

```
merged_df['installer'] = merged_df['installer'].replace(to_replace = ('TASA', 'Tasaf', 'TASAF 1', 'TASAF/', 'TASF',
                                          'TASSAF', 'TASAF'), value ='TASAF')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('RW', 'RWE'), value ='RWE')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('SEMA CO LTD', 'SEMA Consultant', 'SEMA'), value ='

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('DW E', 'DW#', 'DW$', 'DWE&', 'DWE/', 'DWE}',
                                         'DWEB', 'DWE'), value ='DWE')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('No', 'NORA', 'Norad', 'NORAD/', 'NORAD'),
                                          value ='NORAD')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('Ox', 'OXFARM', 'OXFAM'), value ='OXFAM')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('PRIV', 'Priva', 'Privat', 'private', 'Private comp
                                          'Private individuals', 'PRIVATE INSTITUTIONS', 'Private owned',
                                          'Private person', 'Private Technician', 'Private'),
                                          value ='Private')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('Ch', 'CH', 'Chiko', 'CHINA', 'China',
                                            'China Goverment'), value ='Chinese Goverment')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('Unisef','Unicef', 'UNICEF'), value ='UNICEF')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('Wedeco','WEDEKO', 'WEDECO'), value ='WEDECO')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('Wo','WB', 'Word Bank', 'Word bank', 'WBK',
                                          'WORDL BANK', 'World', 'world', 'WORLD BANK', 'World bank',
                                          'world banks', 'World banks', 'WOULD BANK', 'World Bank'),
                                          value ='World Bank')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('Lga', 'LGA'), value ='LGA')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('World Division', 'World Visiin',
                                         'World vision', 'WORLD VISION', 'world vision', 'World Vission',
                                          'World Vision'),
                                          value ='World Vision')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('Local', 'Local technician', 'Local  technician',
                                         'local  technician', 'LOCAL CONTRACT', 'local fundi',
                                         'Local l technician', 'Local te', 'Local technical', 'Local technical tec',
                                         'local technical tec', 'local technician', 'Local technitian',
                                         'local technitian', 'Locall technician', 'Localtechnician',
```

```
                                          'Local Contractor'),
                                           value ='Local Contractor')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('DANID', 'DANNY', 'DANNIDA', 'DANIDS',
                                          'DANIDA CO', 'DANID', 'Danid', 'DANIAD', 'Danda', 'DA',
                                          'DENISH', 'DANIDA'),
                                           value ='DANIDA')

merged_df['installer'] = merged_df['installer'].replace(to_replace =('Adrs', 'Adra', 'ADRA'), value ='ADRA')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('Hesawa', 'hesawa', 'HESAW', 'hesaw',
                                           'HESAWQ', 'HESAWS', 'HESAWZ', 'hesawz', 'hesewa', 'HSW',
                                           'HESAWA'),
                                           value ='HESAWA')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('Jaica', 'JAICA', 'Jica', 'Jeica', 'JAICA CO', 'JAL
                                           'Japan', 'JAPAN', 'JAPAN EMBASSY', 'Japan Government', 'Jicks',
                                           'JIKA', 'jika', 'jiks', 'Embasy of Japan in Tanzania', 'JICA'),
                                           value ='JICA')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('KKT', 'KK', 'KKKT Church', 'KkKT', 'KKT C',
                                           'KKKT'), value ='KKKT')

merged_df['installer'] = merged_df['installer'].replace(to_replace = ('0', 'Not Known', 'not known', 'Not kno'), value ='
```

In [ ]:

```
# Retain top 20 installers as unique entries
top_20_installers = merged_df['installer'].value_counts(normalize=True).head(20).index.tolist()


merged_df['installer'] = [value if value in top_20_installers else "OTHER" for value in merged_df['installer']]
```

In [ ]:

```
# Confirm there are no more missing values
merged_df.isna().sum()
```

Out[ ]:

```
amount_tsh           0
gps_height           0
installer            0
longitude            0
latitude             0
basin                0
region               0
population           0
public_meeting       0
scheme_management    0
permit               0
```

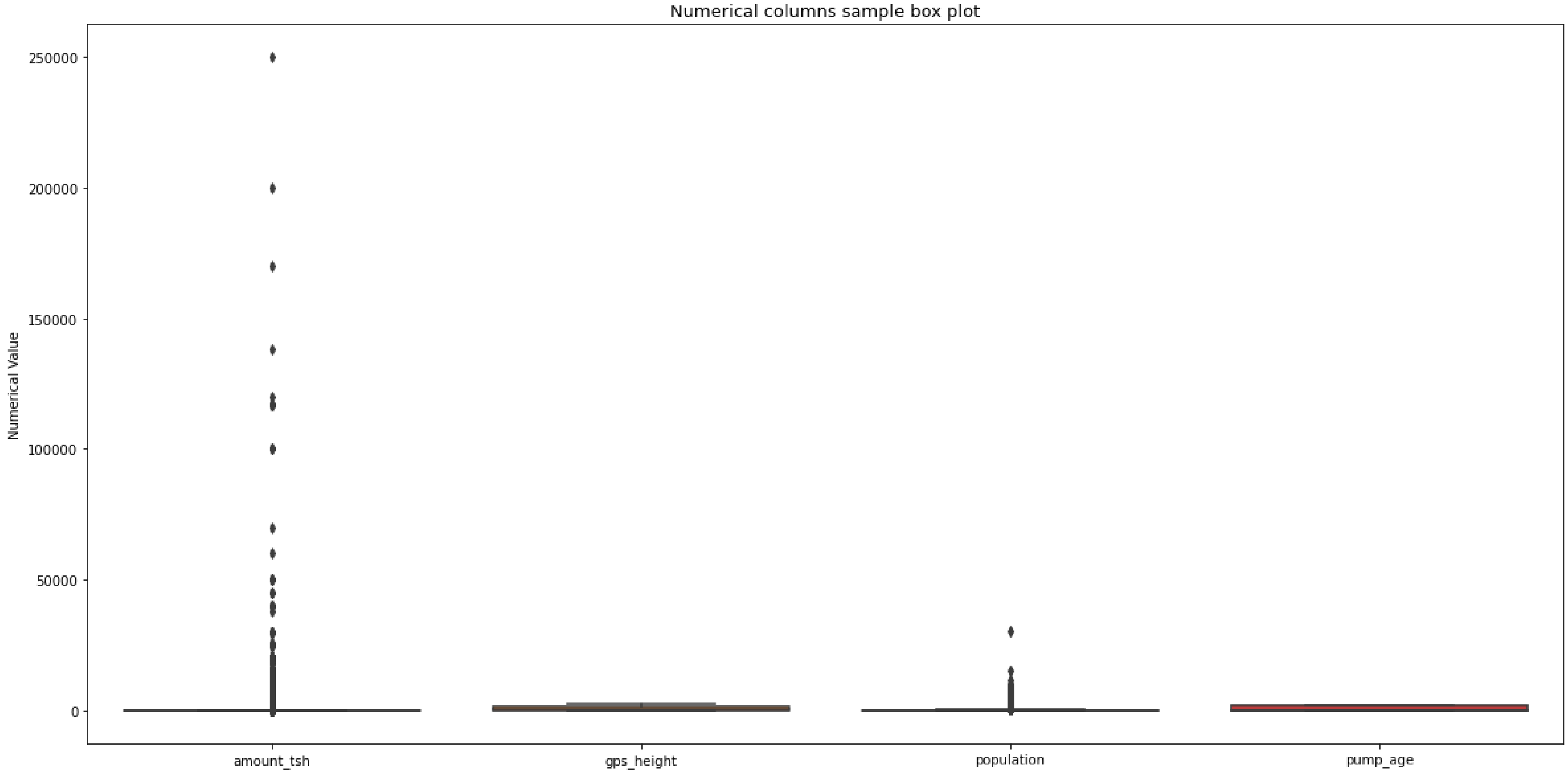
```
construction_year       0
extraction_type_class   0
management_group        0
payment_type            0
water_quality           0
quantity_group          0
source_class            0
waterpoint_type_group   0
status_group            0
_merge                  0
dtype: int64
```

In [ ]:

```
# Ensure the 'installation_year' is in numeric format (not datetime, just the year)
merged_df['construction_year'] = pd.to_numeric(training_merged_df['construction_year'], errors='coerce')

# Calculate pump age
current_year = pd.Timestamp.now().year
merged_df['pump_age'] = current_year - merged_df['construction_year']

# Handle cases where installation_year might be missing or incorrect (e.g., 0 or negative values)
merged_df['pump_age'] = merged_df['pump_age'].apply(lambda x: x if x > 0 else None)
```

In [ ]:

```
# Plotting box plots of some numerical columns
columns = ['amount_tsh', 'gps_height', 'population','pump_age']
plt.figure(figsize=(20, 10))
sns.boxplot(data=[merged_df[col] for col in columns])
plt.title("Numerical columns sample box plot", fontsize=13)
plt.ylabel("Numerical Value")
plt.xticks(range(0,4), columns)
plt.show()
```



In the above I found that the outliers were minimal and opted not to treat them.

In [ ]:
```
# Check whether there are duplicates
merged_df.duplicated(keep = 'first').sum()
```

Out[ ]: 1288

In [ ]:
```
# Change the data type of public_meeting and permit columns to binary for classification
merged_df[['public_meeting', 'permit']] = merged_df[['public_meeting', 'permit']].astype(int)
# Check the new data types
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 51927 entries, 69572 to 26348
Data columns (total 22 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   amount_tsh             51927 non-null  float64
```

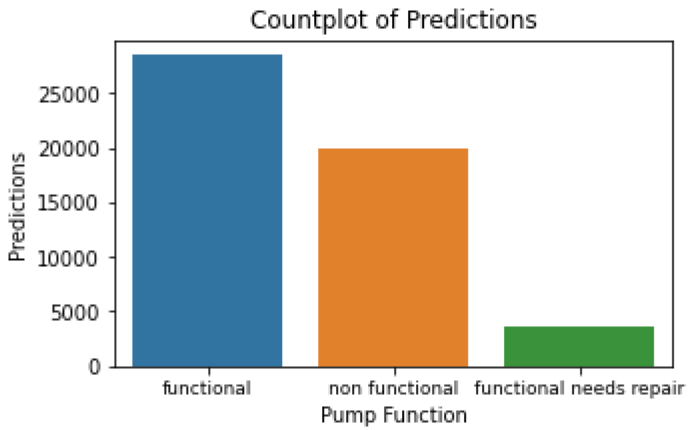
```
 1   gps_height             51927 non-null  int64
 2   installer              51927 non-null  object
 3   longitude              51927 non-null  float64
 4   latitude               51927 non-null  float64
 5   basin                  51927 non-null  object
 6   region                 51927 non-null  object
 7   population             51927 non-null  int64
 8   public_meeting         51927 non-null  int32
 9   scheme_management      51927 non-null  object
 10  permit                 51927 non-null  int32
 11  construction_year      51927 non-null  int64
 12  extraction_type_class  51927 non-null  object
 13  management_group       51927 non-null  object
 14  payment_type           51927 non-null  object
 15  water_quality          51927 non-null  object
 16  quantity_group         51927 non-null  object
 17  source_class           51927 non-null  object
 18  waterpoint_type_group  51927 non-null  object
 19  status_group           51927 non-null  object
 20  _merge                 51927 non-null  category
 21  pump_age               51927 non-null  int64
dtypes: category(1), float64(3), int32(2), int64(4), object(12)
memory usage: 8.4+ MB
```

Explore data distributions:

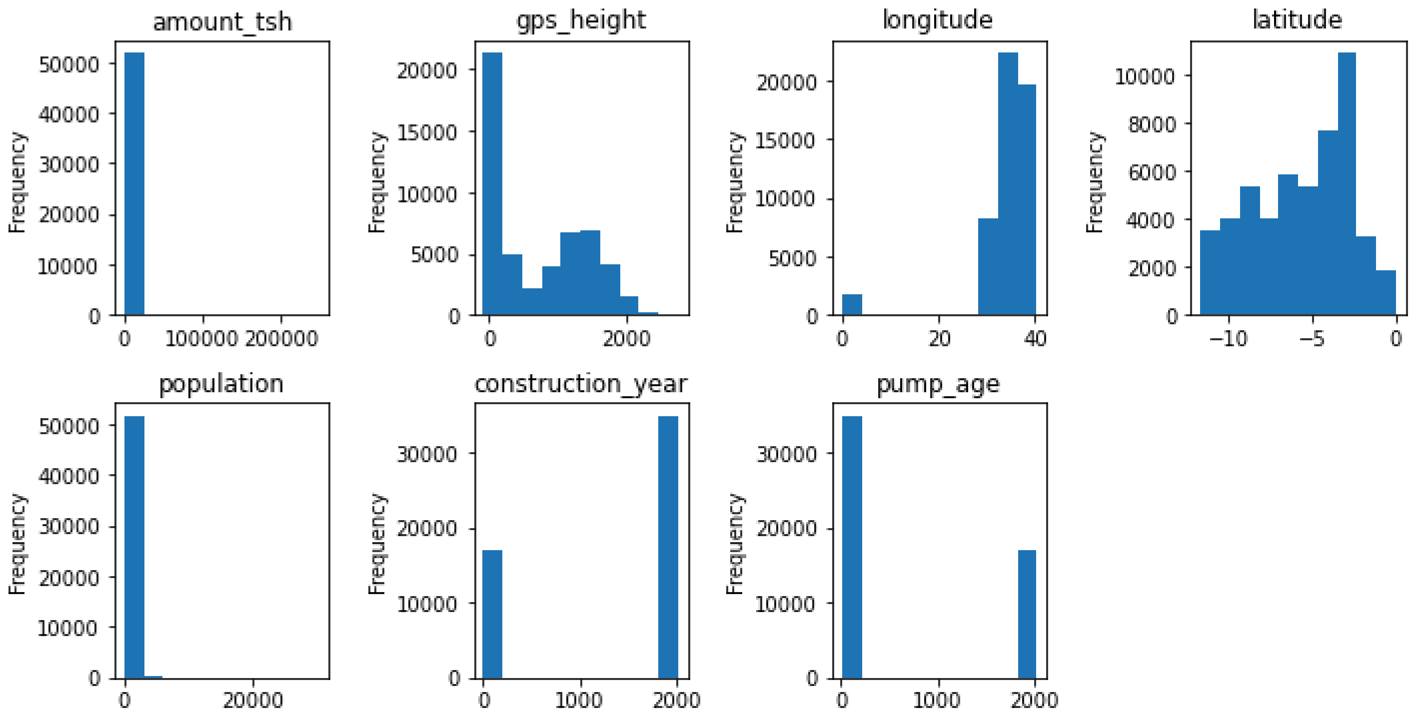
In [ ]:

```
# Plot distribution of target variable.
fig, ax = plt.subplots(figsize=(5, 3))
sns.countplot(merged_df['status_group'])
x_labels = merged_df['status_group'].unique()

# Add labels
plt.title('Countplot of Predictions')
plt.xlabel('Pump Function')
ax.set_xticklabels(x_labels, fontsize=9)
plt.ylabel('Predictions')
plt.show()
```

```
c:\Users\User\anaconda3\envs\learn-env\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following var
iable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other argume
nts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [ ]:
```
# Histogram of continuous variables
continuous = ['amount_tsh','gps_height','longitude','latitude','population','construction_year','pump_age']
fig = plt.figure(figsize=(10, 10))
for i, col in enumerate(continuous):
    ax = plt.subplot(4, 4, i+1)
    merged_df[col].plot(kind='hist', ax=ax, title=col)
plt.tight_layout()
plt.show()
```



3. Logistic Regression

- Prepare the data (ensure binary target variable)
- Create and train the model
- Make predictions on the test set
- Evaluate the model (accuracy, precision, recall, F1-score)
- Plot the ROC curve and calculate AUC
- Analyze coefficients and their significance

In [ ]:
```
# Assign status_group column to y series
y = merged_df['status_group']

# Drop status_group and _merge to create X dataframe
X = merged_df.drop(['status_group','_merge'], axis=1)

# Print first 5 rows of X
X.head()
```

Out[ ]:

id	amount_tsh	gps_height	installer	longitude	latitude	basin	region	population	public_meeting	scheme_management	permit
69572	6000.0	1390	Roman Catholic Church	34.938093	-9.856322	Lake Nyasa	Iringa	109	1	VWC	0
8776	0.0	1399	OTHER	34.698766	-2.147466	Lake Victoria	Mara	280	0	Other	1
34310	25.0	686	World Vision	37.460664	-3.821329	Pangani	Manyara	250	1	VWC	1
67743	0.0	263	OTHER	38.486161	-11.155298	Ruvuma / Southern Coast	Mtwara	58	1	VWC	1
9944	20.0	0	DWE	39.172796	-4.765587	Pangani	Tanga	1	1	VWC	1

In [ ]:

```
#Check categorical columns in merged set
merged_df.select_dtypes(include=['object']).columns
```

Out[ ]:

```
Index(['installer', 'basin', 'region', 'scheme_management',
       'extraction_type_class', 'management_group', 'payment_type',
       'water_quality', 'quantity_group', 'source_class',
       'waterpoint_type_group', 'status_group'],
      dtype='object')
```

In [ ]:

```
#check numerical columns in merged set
merged_df.select_dtypes(include=['int64', 'float64']).columns
```

Out[ ]:

```
Index(['amount_tsh', 'gps_height', 'longitude', 'latitude', 'population',
       'construction_year', 'pump_age'],
      dtype='object')
```

In [ ]:

```
# Create lists of categorical, numerical, and binary columns
category_column = ['installer', 'basin', 'region', 'scheme_management',
       'extraction_type_class', 'management_group', 'payment_type',
       'water_quality', 'quantity_group', 'source_class',
       'waterpoint_type_group',]

numerical_column = ['amount_tsh', 'gps_height', 'longitude', 'latitude', 'population',
       'construction_year', 'pump_age']
```

```
binary_column = ['public_meeting', 'permit']
```

In [ ]:
```
#create dummies for categorical colums
X= pd.get_dummies(X, columns=category_column)
X
```

Out[ ]:

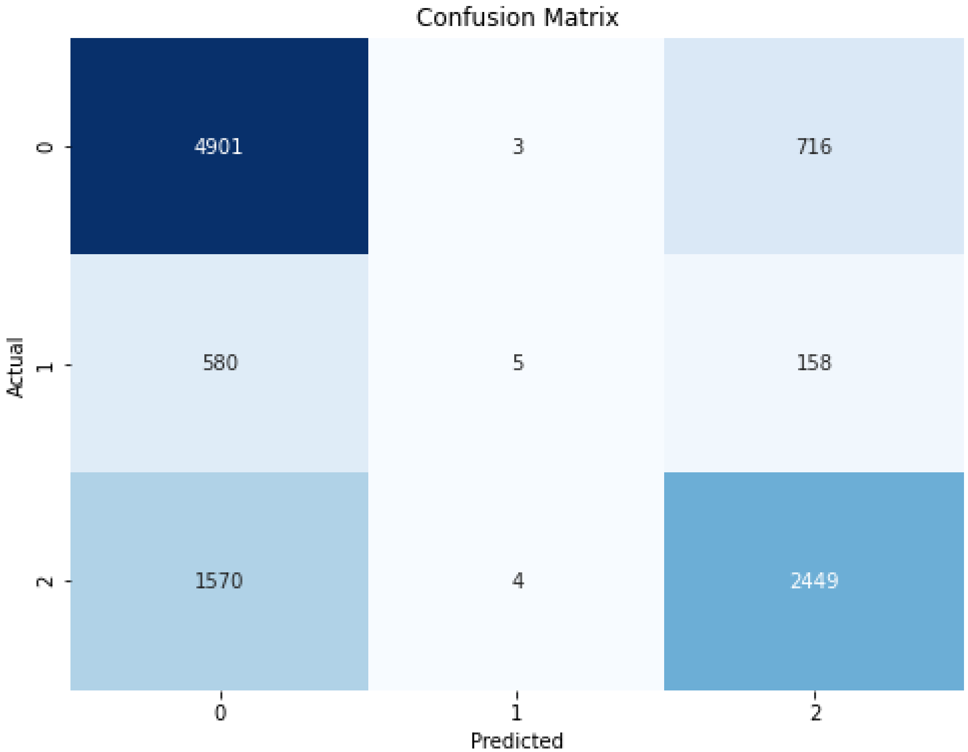
	amount_tsh	gps_height	longitude	latitude	population	public_meeting	permit	construction_year	pump_age	installer_CES	...	quai
id												
69572	6000.0	1390	34.938093	-9.856322	109	1	0	1999	25	0	...	
8776	0.0	1399	34.698766	-2.147466	280	0	1	2010	14	0	...	
34310	25.0	686	37.460664	-3.821329	250	1	1	2009	15	0	...	
67743	0.0	263	38.486161	-11.155298	58	1	1	1986	38	0	...	
9944	20.0	0	39.172796	-4.765587	1	1	1	2009	15	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
11164	500.0	351	37.634053	-6.124830	89	1	1	2007	17	0	...	
60739	10.0	1210	37.169807	-3.253847	125	1	1	1999	25	1	...	
27263	4700.0	1212	35.249991	-9.070629	56	1	1	1996	28	0	...	
31282	0.0	0	35.861315	-6.378573	0	1	1	0	2024	0	...	
26348	0.0	191	38.104048	-6.747464	150	1	1	2002	22	0	...	

51927 rows × 113 columns

In [ ]:
```
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Logistic Regression model
log_reg = LogisticRegression(max_iter=1000)

# Train the model
log_reg.fit(X_train, y_train)

# Predict on the test set
```

```
y_pred_logreg = log_reg.predict(X_test)
# Plotting Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred_logreg), annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred_logreg)
print(f"Accuracy: {accuracy}")
print("Classification Report:\n", classification_report(y_test, y_pred_logreg))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_logreg))
```

```
c:\Users\User\anaconda3\envs\learn-env\lib\site-packages\sklearn\linear_model\_logistic.py:762: ConvergenceWarning: lbfgs
failed to converge (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
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```




```
Accuracy: 0.7081648372809551
Classification Report:
                          precision    recall  f1-score   support

             functional       0.70      0.87      0.77      5620
functional needs repair       0.42      0.01      0.01       743
         non functional       0.74      0.61      0.67      4023

               accuracy                           0.71     10386
              macro avg       0.62      0.50      0.48     10386
           weighted avg       0.69      0.71      0.68     10386

Confusion Matrix:
 [[4901    3  716]
 [ 580    5  158]
 [1570    4 2449]]
```

I interpret the results of the logistic regression as follows:

1. Overall Accuracy:

The model correctly classifies about 70.8% of the cases in the test set, which is moderately good but shows room for improvement.

2. Class-wise Performance:

- **Functional (Majority Class)**:
  - **Precision**: 0.70 – When the model predicts a pump as "functional," it is correct 70% of the time.
  - **Recall**: 0.87 – The model captures 87% of all actual "functional" pumps.
  - **F1-Score**: 0.77 – This represents a balance between precision and recall, indicating good performance for this class.
- **Functional Needs Repair (Minority Class)**:
  - **Precision**: 0.42 – Only 42% of the time when the model predicts "functional needs repair" is it correct.
  - **Recall**: 0.01 – The model only captures 1% of actual "needs repair" cases. This shows a major issue in identifying pumps that need repair.
  - **F1-Score**: 0.01 – A very low score, suggesting the model is poor at distinguishing this class.
- **Non-functional**:
  - **Precision**: 0.74 – 74% of the predicted "non-functional" pumps are correctly classified.
  - **Recall**: 0.61 – The model identifies 61% of actual "non-functional" pumps.
  - **F1-Score**: 0.67 – Indicates decent performance but lower recall.

3. Confusion Matrix:

- **Functional**: The model correctly classified 4901 out of 5620 "functional" pumps, but misclassified 716 as "non-functional."
- **Functional Needs Repair**: The model struggles heavily here, correctly identifying only 5 out of 743 pumps that need repair, misclassifying most as either "functional" (580) or "non-functional" (158).
- **Non-functional**: The model correctly identifies 2449 of the 4023 non-functional pumps, but misclassifies 1570 as "functional."

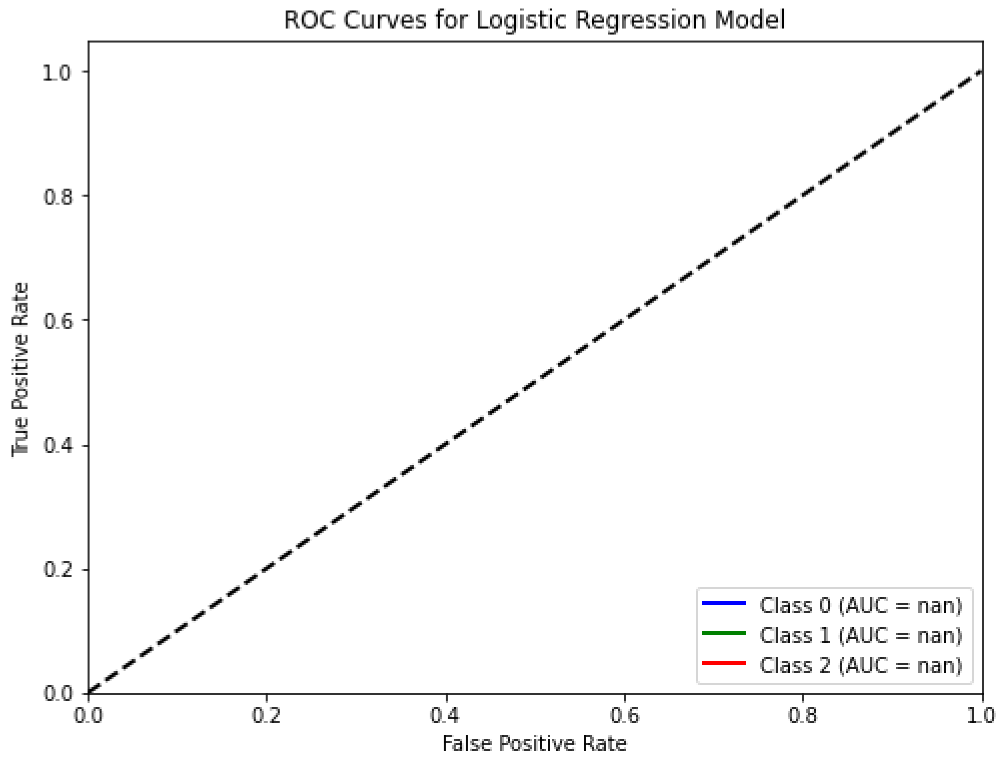
Insights:

- The model does well at predicting "functional" and "non-functional" pumps but performs very poorly at identifying pumps that need repair.
- The **class imbalance** (with the "functional needs repair" class being underrepresented) is likely contributing to the poor performance on this class.

- Since recall for "functional needs repair" is very low, this indicates that most pumps requiring repair are misclassified as either "functional" or "non-functional."

In [ ]:
```
# Binarize the output labels (important for multi-class ROC)
#n_classes = len(set(y_test))  # Number of classes
#y_test_bin = label_binarize(y_test, classes=[0, 1, 2])

# Get probability estimates for the classes
#y_prob = log_reg.predict_proba(X_test)

# Compute ROC curve and AUC for each class
#fpr = dict()
#tpr = dict()
#roc_auc = dict()

#for i in range(n_classes):
   # fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_prob[:, i])
   # roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve for each class
#plt.figure(figsize=(8, 6))

#colors = ['blue', 'green', 'red']
#for i, color in zip(range(n_classes), colors):
   # plt.plot(fpr[i], tpr[i], color=color, lw=2,
         #    label=f'Class {i} (AUC = {roc_auc[i]:.2f})')

# Plot diagonal line for random guessing
#plt.plot([0, 1], [0, 1], 'k--', lw=2)

#plt.xlim([0.0, 1.0])
#plt.ylim([0.0, 1.05])
#plt.xlabel('False Positive Rate')
#plt.ylabel('True Positive Rate')
#plt.title('ROC Curves for Logistic Regression Model')
#plt.legend(loc='lower right')
#plt.show()
```

```
c:\Users\User\anaconda3\envs\learn-env\lib\site-packages\sklearn\metrics\_ranking.py:811: UndefinedMetricWarning: No posi
tive samples in y_true, true positive value should be meaningless
  warnings.warn("No positive samples in y_true, "
```

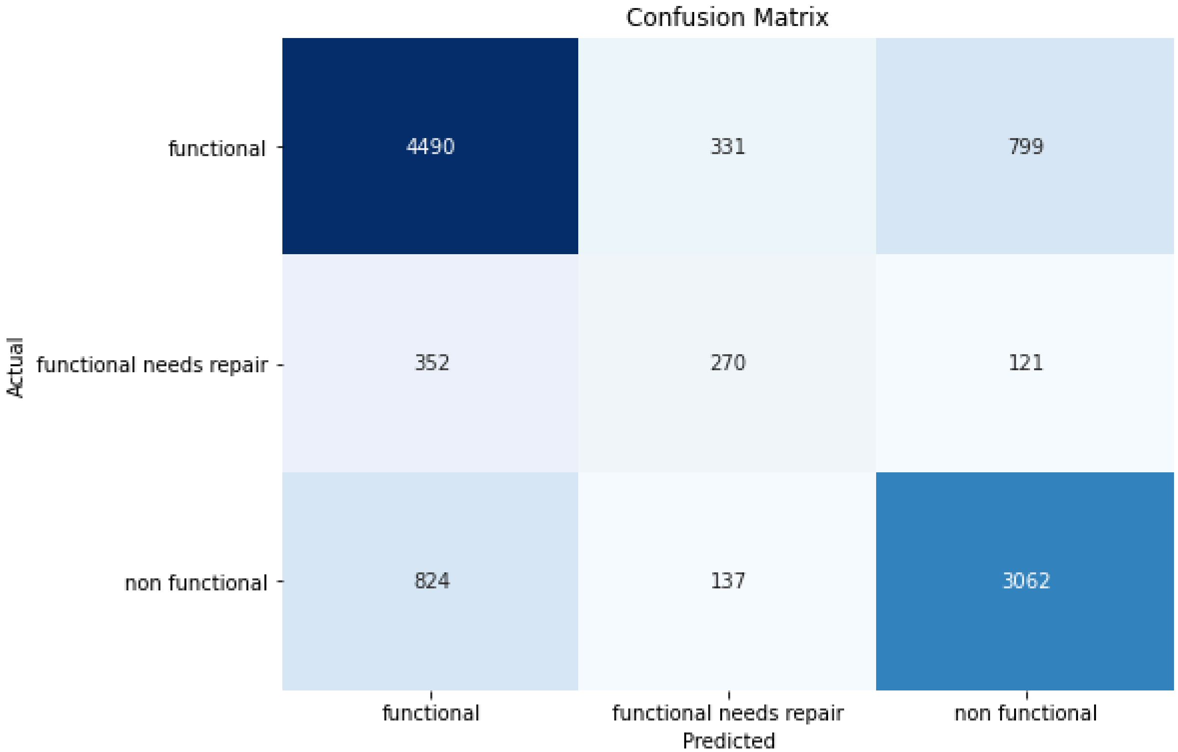


4. Decision Trees

- Prepare the data
- Create and train the model
- Make predictions on the test set
- Evaluate the model (accuracy, precision, recall, F1-score)
- Visualize the tree structure
- Analyze feature importance

In [ ]:
```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Initialize the Decision Tree model
dt = DecisionTreeClassifier(random_state=42)

# Train the model
```

```
dt.fit(X_train, y_train)

# Predict on the test set
y_pred_dt = dt.predict(X_test)

# Generate the confusion matrix
cm_dt = confusion_matrix(y_test, y_pred_dt)

# Plot the confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm_dt, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=dt.classes_, yticklabels=dt.classes_)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Classification Report:\n", classification_report(y_test, y_pred_dt))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_dt))
```




```
Accuracy: 0.7531292124013095
Classification Report:
                         precision    recall  f1-score   support

             functional       0.79      0.80      0.80      5620
functional needs repair       0.37      0.36      0.36       743
         non functional       0.77      0.76      0.77      4023

               accuracy                           0.75     10386
              macro avg       0.64      0.64      0.64     10386
           weighted avg       0.75      0.75      0.75     10386

Confusion Matrix:
 [[4490  331  799]
 [ 352  270  121]
 [ 824  137 3062]]
```

Interpretation of findings from decision tree:

1. Overall Accuracy:

- **Accuracy**: 0.7531 (75.31%) means that 75.31% of the water pumps were correctly classified as either functional, needing repair, or non-functional. While this is a decent result, it can still improve, particularly in distinguishing between the different classes.

2. Precision, Recall, and F1-Score:

- **Functional Pumps**:
  - **Precision**: 0.79 (79%) means that, when the model predicted a pump to be functional, 79% of the time, it was correct.
  - **Recall**: 0.80 (80%) means that the model correctly identified 80% of the truly functional pumps.
  - **F1-Score**: 0.80 indicates a good balance between precision and recall, reflecting strong performance for this class.
- **Functional but Needs Repair**:
  - **Precision**: 0.37 (37%) shows that when the model predicted a pump to need repairs, it was only correct 37% of the time. This indicates a high number of false positives for this class.
  - **Recall**: 0.36 (36%) means that the model only identified 35% of the pumps that actually need repair.
  - **F1-Score**: 0.36 indicates weak performance, as the model struggles to accurately classify pumps that need repair. This suggests that this class is the most challenging for the model to predict.
- **Non-Functional Pumps**:
  - **Precision**: 0.77 (77%) means that when the model predicted a pump was non-functional, 77% of the predictions were correct.
  - **Recall**: 0.76 (76%) shows that the model correctly identified 77% of the truly non-functional pumps.
  - **F1-Score**: 0.77 indicates solid performance for this class, similar to the functional pumps.

3. Macro and Weighted Averages:

- **Macro Average**: These metrics are the unweighted averages across all classes:
  - **Precision**: 0.64 (64%)
  - **Recall**: 0.64 (64%)
  - **F1-Score**: 0.64 (64%)
  - These lower values indicate that the model performs worse on minority classes, particularly on pumps that need repair.
- **Weighted Average**: These metrics are weighted by the number of samples in each class:
  - **Precision**: 0.75 (75%)

- **Recall**: 0.75 (75%)
- **F1-Score**: 0.75 (75%)
- These values are higher since the model performs well on the majority class (`functional`), which has a larger representation in the dataset.

4. Confusion Matrix:

- **Functional Pumps**: O4,490 pumps were correctly classified as functional, but 331 were misclassified as "functional needs repair" and 799 as "non-functional."
- **Functional but Needs Repair**: Only 270 were correctly classified, with 352 misclassified as functional and 121 as non-functional. This class has the highest misclassification rate, indicating difficulty in distinguishing between functional needs repair and the other two categories.
- **Non-Functional Pumps**: 3,062 pumps were correctly identified as non-functional, but 824 were incorrectly classified as functional, and 137 as needing repair.

Key Insights:

1. **Strength**: The model performs well on the majority classes (`functional` and `non-functional` pumps) .

2. **Weakness**: The model struggles significantly with the `functional needs repair` class. This indicates that the model has difficulty distinguishing pumps that require minor repairs from those that are functional or non-functional.

5. Model Comparison and Conclusion

- Compare performance metrics across all models
- Discuss strengths and weaknesses of each approach
- Recommend the best model(s) for the problem at handk
- Summarize key findings
- Discuss limitations of the current approach
- Suggest potential improvements or additional models to try

In [ ]:
```
# Define the data for the table
data = {
```

```
    'Metric': ['Accuracy', 'Functional Precision', 'Functional Recall', 'Functional F1-Score',
               'Functional Needs Repair Precision', 'Functional Needs Repair Recall', 'Functional Needs Repair F1-Score',
               'Non-functional Precision', 'Non-functional Recall', 'Non-functional F1-Score'],
    'Decision Tree': [0.757, 0.79, 0.80, 0.80,
                      0.37, 0.35, 0.36,
                      0.77, 0.77, 0.77],
    'Logistic Regression': [0.708, 0.70, 0.87, 0.77,
                            0.42, 0.01, 0.01,
                            0.74, 0.61, 0.67]
}

# Create the DataFrame
results_df = pd.DataFrame(data)

# Display the DataFrame
results_df
```

Out[ ]:

	Metric	Decision Tree	Logistic Regression
0	Accuracy	0.757	0.708
1	Functional Precision	0.790	0.700
2	Functional Recall	0.800	0.870
3	Functional F1-Score	0.800	0.770
4	Functional Needs Repair Precision	0.370	0.420
5	Functional Needs Repair Recall	0.350	0.010
6	Functional Needs Repair F1-Score	0.360	0.010
7	Non-functional Precision	0.770	0.740
8	Non-functional Recall	0.770	0.610
9	Non-functional F1-Score	0.770	0.670

When we compare the results of the Logistic Regression and Decision Tree models. We find the following:

1. **Accuracy**:
   - The Decision Tree (75.8%) outperforms Logistic Regression (70.8%) in terms of accuracy.
2. **Class Performance**:

- **Functional Pumps**:
  - Both models perform reasonably well, but the Decision Tree model has a slightly better balance between precision and recall, with a higher F1-score.
- **Functional Needs Repair**:
  - Both models struggle, but the Decision Tree performs better, with a recall of 0.35 compared to just 0.01 for Logistic Regression.
  - Logistic Regression is almost entirely ineffective at identifying "needs repair" pumps, whereas the Decision Tree, while still underperforming, does capture a small portion.
- **Non-functional Pumps**:
  - The Decision Tree is also better at identifying "non-functional" pumps, with both higher recall and F1-score.

3. **Confusion Matrix**:

- The Decision Tree distributes misclassifications more effectively across classes, particularly in the minority class ("needs repair").
- Logistic Regression over-classifies pumps as functional and fails to capture "needs repair" instances.

Strengths & Weaknesses:

- **Decision Tree**:
  - **Strengths**:
    - Performs better at handling class imbalance.
    - More interpretable for understanding complex decision boundaries.
    - Handles non-linear relationships well.
  - **Weaknesses**:
    - Prone to overfitting, especially with noisy data or small datasets.
    - May require tuning to avoid over-complex models.
- **Logistic Regression**:
  - **Strengths**:
    - Simpler and computationally efficient.
    - Well-suited for linearly separable data.
  - **Weaknesses**:
    - Struggles with imbalanced data, especially in the "needs repair" class.

- Assumes linear relationships between features and target, which may not fit this dataset.

Recommendation:

Given the results, the **Decision Tree** is the better model for this problem, particularly because:

- It outperforms Logistic Regression in overall accuracy.
- It handles the "functional needs repair" class significantly better.
- It captures more complex, non-linear relationships, which are likely present in the dataset.

Possible Next Steps:

1. **Addressing Class Imbalance**: Techniques like oversampling the "functional needs repair" class, undersampling the "functional" class, or using a more balanced metric such as the F1-score might help improve the model's ability to correctly identify pumps needing repair.
2. **Feature Engineering**: I would explore creating more features or transforming existing ones to improve model performance.
3. **Try Other Models**: I would consider using more complex models like Random Forest or Gradient Boosting to improve classification performance, particularly for the minority class.

In [ ]: