1. Overview

This competition, "Pump it Up: Data Mining the Water Table," hosted on DrivenData, challenges participants to predict the functional status of water pumps across Tanzania using a provided dataset. The contest spans from 2024 and aims to enhance access to clean, potable water by identifying malfunctioning water pumps. Participants are supplied with extensive data on various characteristics of the water points, from construction year to water quality. The primary goal is to classify each water point into one of three categories: functional, functional needs repair, and non-functional. This analysis could guide strategic decisions for improving water access and infrastructure investments in developing regions.

2. Business Understanding

The core objective of the "Pump it Up: Data Mining the Water Table" competition is to enable the identification of water pumps in Tanzania that are functional, require repairs, or are non-functional. The insights derived from this analysis will directly influence decisions regarding maintenance, investments, and resource allocation in the water infrastructure sector. Stakeholders, including government agencies and NGOs, will use these findings to prioritize and streamline efforts towards ensuring reliable water access. By effectively categorizing water points, the project aims to enhance operational efficiencies and reduce downtime due to pump failures. The ultimate goal is to support sustainable water management practices that can significantly impact public health and economic development in Tanzania.

Primary stakeholders for this project are the Tanzanian government and international development organizations focused on improving water access in the region.

3. Data Understanding

3.1 Data Description

Drawing from a comprehensive dataset provided by the "Pump it Up: Data Mining the Water Table" competition on DrivenData, our analysis is centered around extensive information regarding water points across Tanzania. This dataset includes:

- Geographic data such as location coordinates, altitude, and administrative divisions (region, district, and ward).
- Water point specifics such as the type, construction year, funding organization, and managing entity.
- Operational data including the water source, extraction type, water quality, and current functional status of each water pump.

Our investigation targets three key objectives: identifying patterns of pump functionality, understanding factors leading to pump failures or repairs, and assessing the impacts of management practices on pump operability. By analyzing these elements, we aim to derive actionable insights that can guide infrastructural improvements and strategic investments in water resource management. The outcome of this analysis will inform decision-making processes for stakeholders involved in Tanzanian water supply, optimizing interventions for enhanced water accessibility and reliability. This focused approach empowers our stakeholders to efficiently address the most critical needs, leveraging data-driven strategies to improve public health and community resilience.

3.2 Import Necessary Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import re  # Import regular expressions library

from IPython.display import display
```

3.3 Define global variables

In [2]:
```
INPUT_PATH_Submission_Format = "../Data/SubmissionFormat.csv"
INPUT_PATH_Test_set_values = "../Data/Test_set_values.csv"
INPUT_PATH_Training_set_labels = "../Data/Training_set_labels.csv"
INPUT_PATH_Training_set_values = "../Data/Training_set_values.csv"
```

3.4 Functions

In [3]:
```
def categorize_funder(funder):
    """
    Categorizes a funder name into specific groups based on keywords.

    Args:
    funder (str): A string representing the name of the funder to categorize.

    Returns:
    str: A category name representing the type of organization the funder belo

    This function takes a funder name, converts it to lowercase, removes leadi
    and categorizes it into predefined groups like 'Government', 'Religious Or
    'International Aid', 'Private Companies', or 'Individual/Other' based on k
    """
    funder = funder.lower().strip()  # convert to lowercase and strip whitespa
    if any(x in funder for x in ['government','ministry','gov','minis']):
        return 'Government'
    elif any(x in funder for x in ['church', 'muslim','mus', 'islamic','islam'
        return 'Religious Organizations'
    elif any(x in funder for x in ['ngo', 'foundation', 'fund', 'trust', 'soci
        return 'NGO'
    elif any(x in funder for x in ['international','internatio', 'un', 'world
        return 'International Aid'
    elif any(x in funder for x in ['ltd', 'company','compa', 'group', 'enterpr
        return 'Private Companies'
    else:
        return 'Individual/Other'
```

In [4]:
```
def categorize_installer(installer):
    """
    Categorizes an installer name into specific groups based on keywords.

    Args:
    installer (str): A string representing the name of the installer to catego

    Returns:
    str: A category name representing the type of entity the installer belongs

    This function processes an installer name by converting it to lowercase an
    any leading/trailing whitespace. It categorizes the name into predefined g
    'DWE', 'Government', 'Community', 'NGO', 'Private Company', 'Institutional
    based on specific keywords present in the installer's name. This helps in
    installer data for better analysis and insight extraction.
    """
    installer = installer.lower().strip()  # convert to lowercase and strip wh
    if 'dw' in installer:
        return 'DWE'
    elif any(x in installer for x in ['government', 'govt', 'gove']):
        return 'Government'
    elif any(x in installer for x in ['resource']):
        return 'Other'
    elif any(x in installer for x in ['community', 'villagers', 'village','com
        return 'Community'
    elif any(x in installer for x in ['ngo', 'unicef', 'foundat']):
        return 'NGO'
    elif 'company' in installer or 'contractor' in installer:
        return 'Private Company'
    elif any(x in installer for x in ['school','schoo','church', 'rc']):
        return 'Institutional'
    else:
        return 'Other'
```

In [5]:
```
def group_scheme_management(value):
    """
    Categorizes scheme management types into broader, more generalized groups.

    Args:
    value (str): A string representing the scheme management type to categoriz

    Returns:
    str: A generalized category name representing the type of scheme managemen

    This function takes a specific scheme management type and categorizes it i
    more generalized groups such as 'Government', 'Community', 'Private Sector
    'Water Board', or 'Other'. This categorization aids in simplifying the ana
    and understanding of the data by reducing the number of distinct categorie
    making trends and patterns more discernible.
    """
    if value in ['VWC', 'Water authority', 'Parastatal']:
        return 'Government'
    elif value in ['WUG', 'WUA']:
        return 'Community'
    elif value in ['Company', 'Private operator']:
        return 'Private Sector'
    elif value == 'Water Board':
        return 'Water Board'  # Retain this as a separate category if distinct
    else:
        return 'Other'
```

In [6]:
```
def clean_text(text):
    """
    Cleans a text string by converting to lowercase, removing non-alphanumeric
    and replacing multiple spaces with a single space. If the input is solely

    Args:
    text (str or NaN): The text to be cleaned; can be a string, numeric, or Na

    Returns:
    str or NaN: The cleaned text, with all characters in lowercase, non-alphan
                and multiple spaces collapsed to a single space, or the origin

    This function standardizes a text string by making it lowercase, stripping
    and then replacing sequences of spaces with a single space, facilitating u
    is numeric, it is assumed to be standardized already and is returned witho
    """
    if pd.isna(text):
        return text
    if isinstance(text, (int, float)):  # Check if the input is numeric
        return text
    text = text.lower()  # Convert to lowercase
    text = ''.join(char for char in text if char.isalpha() or char.isspace())
    text = re.sub(r'\s+', ' ', text)  # Replace multiple spaces with a single
    return text
```

In [7]:
```
def analyze_numeric_stats_and_plots(df, columns):
    """
    Calculates and prints descriptive statistics, and generates boxplots and h

    Args:
    df (pd.DataFrame): The DataFrame containing the data.
    columns (list): List of numeric column names to analyze.

    The function computes the mean, median, standard deviation, coefficient of
    kurtosis, and quartiles for the specified columns. It also generates a box
    """
    for column in columns:
        if column in df.columns and pd.api.types.is_numeric_dtype(df[column]):
            print(f"Stats for {column}:")

            # Calculate statistics
            max_value = df[column].max()
            min_value = df[column].min()
            mean = df[column].mean()
            median = df[column].median()
            std_dev = df[column].std()
            coeff_variation = std_dev / mean if mean != 0 else np.nan
            skewness = df[column].skew()
            kurtosis = df[column].kurtosis()
            quartiles = df[column].quantile([0.25, 0.5, 0.75])

            print(f"Max: {max_value}")
            print(f"Min: {min_value}")
            print(f"Mean: {mean}")
            print(f"Median: {median}")
            print(f"Standard Deviation: {std_dev}")
            print(f"Coefficient of Variation: {coeff_variation}")
            print(f"Skewness: {skewness}")
            print(f"Kurtosis: {kurtosis}")
            print(f"25th percentile (Q1): {quartiles[0.25]}")
            print(f"50th percentile (Median): {quartiles[0.5]}")
            print(f"75th percentile (Q3): {quartiles[0.75]}")

            # Plotting
            plt.figure(figsize=(12, 6))

            # Boxplot
            plt.subplot(1, 2, 1)  # 1 row, 2 columns, 1st subplot
            sns.boxplot(y=df[column])
            plt.title(f'Boxplot of {column}')

            # Histogram
            plt.subplot(1, 2, 2)  # 1 row, 2 columns, 2nd subplot
            sns.histplot(df[column], kde=False, bins=100)
            plt.title(f'Histogram of {column}')

            plt.show()
```

In [8]:
```
def plot_categorical_proportions(df):
    """
    Plots bar charts for each categorical variable in a DataFrame, showing the
    ordered by proportion in descending order. Each bar is labeled with its pe

    Args:
    df (pd.DataFrame): The DataFrame to analyze.

    This function identifies categorical variables, calculates the proportion
    and plots a bar chart for each categorical variable. Labels on the bars di
    """
    # Identifying categorical columns in the DataFrame
    categorical_columns = df.select_dtypes(include=['object', 'category']).col

    for col in categorical_columns:
        # Calculating proportions
        value_counts = df[col].value_counts(normalize=True).sort_values(ascend
        percentages = value_counts * 100  # Convert proportions to percentages

        # Plotting
        plt.figure(figsize=(10, 6))
        ax = percentages.plot(kind='bar')
        ax.set_title(f'Proportion of Categories in {col}')
        ax.set_ylabel('Percentage')

        # Adding percentage labels on the bars
        for p in ax.patches:
            ax.annotate(f'{p.get_height():.2f}%', (p.get_x() + p.get_width() /
                        ha='center', va='center', xytext=(0, 10), textcoords='

        plt.show()
```

In [9]:
```
def plot_categorical_proportions(df, columns):
    """
    Plots the proportions of categories in specified categorical columns of a

    Args:
        df (pd.DataFrame): The DataFrame containing the data.
        columns (list of str): List of categorical column names to plot.
    """
    for col in columns:
        # Calculating proportions
        value_counts = df[col].value_counts(normalize=True).sort_values(ascend
        percentages = value_counts * 100  # Convert proportions to percentages

        # Plotting
        plt.figure(figsize=(10, 6))
        ax = percentages.plot(kind='bar')
        ax.set_title(f'Proportion of Categories in {col}')
        ax.set_ylabel('Percentage')

        # Adding percentage labels on the bars
        for p in ax.patches:
            ax.annotate(f'{p.get_height():.2f}%', (p.get_x() + p.get_width() /
                        ha='center', va='center', xytext=(0, 10), textcoords='

        plt.show()
```

In [10]:
```
def plot_grouped_charts(df, status_col, cols):
    """
    Creates combined plots for each column in the DataFrame based on their dat
    For numeric columns, histograms for all statuses are combined in one plot,
    For categorical columns, grouped bar charts are created.

    Args:
        df (pd.DataFrame): The DataFrame containing the data.
        status_col (str): The name of the column to group data by.
        cols (list of str): List of column names to plot, both categorical and
    """
    unique_statuses = df[status_col]. unique()
    colors = plt.get_cmap('tab10')  # Fetches a colormap with distinct colors

    for col in cols:
        if df[col].dtype in ['int64', 'float64']:  # Numeric Columns
            plt.figure(figsize=(12, 6))

            # Histogram for all statuses
            for i, status in enumerate(unique_statuses):
                sns.histplot(df[df[status_col] == status][col], kde=True, elem
                             stat='density', label=str(status), color=colors(i

            plt.title(f'Combined Histogram of {col} by {status_col}')
            plt.legend(title=status_col)
            plt.show()

            # Boxplot for all statuses
            plt.figure(figsize=(12, 6))
            sns.boxplot(x=status_col, y=col, data=df, palette='tab10')
            plt.title(f'Combined Boxplot of {col} by {status_col}')
            plt.show()

        elif df[col].dtype == 'object':  # Categorical Columns
            plt.figure(figsize=(10, 6))
            sns.countplot(data=df, x=status_col, hue=col)
            plt.title(f'Grouped Bar Chart of {status_col} by {col}')
            plt.ylabel('Count')
            plt.xlabel(status_col)
            plt.legend(title=col, loc='upper right')
            plt.xticks(rotation=45)
            plt.show()
```

In [11]:
```
def generate_proportion_contingency_tables(df, status_col, categorical_cols):
    """
    Generates two-way contingency tables of proportions for the specified stat

    Args:
        df (pd.DataFrame): The DataFrame containing the data.
        status_col (str): The column name to use as one axis of the contingenc
        categorical_cols (list of str): List of categorical column names to in

    Returns:
        dict: A dictionary of pandas DataFrame objects where each key is the c
    """
    tables = {}
    for col in categorical_cols:
        if col != status_col:  # Ensure the status column is not included in t
            # Compute the contingency table with proportions normalized over a
            contingency_table = pd.crosstab(df[status_col], df[col], normalize
            contingency_table_df = pd.DataFrame(contingency_table)
            contingency_table_df = contingency_table_df.round(4) * 100  # Conv
            tables[col] = contingency_table_df

    return tables
```

3.5 Code

3.5.1 Exploratory Analysis

3.5.1.1 Looking at the train and labels dataset

In [12]:
```
df_train = pd.read_csv(INPUT_PATH_Training_set_values)
df_train.head()
```

Out[12]:

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wp
0	69572	6000.0	2011-03-14	Roman	1390	Roman	34.938093	-9.856322	
1	8776	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.698766	-2.147466	Z
2	34310	25.0	2013-02-25	Lottery Club	686	World vision	37.460664	-3.821329	N
3	67743	0.0	2013-01-28	Unicef	263	UNICEF	38.486161	-11.155298	Z Na
4	19728	0.0	2011-07-13	Action In A	0	Artisan	31.130847	-1.825359	

5 rows × 40 columns

In [13]:
```
df_train.shape
```
Out[13]: (59400, 40)

In [14]:
```
df_labels = pd.read_csv(INPUT_PATH_Training_set_labels)
df_labels.head()
```
Out[14]:

	id	status_group
0	69572	functional
1	8776	functional
2	34310	functional
3	67743	non functional
4	19728	functional

In [15]:
```
df_labels.shape
```
Out[15]: (59400, 2)

3.5.1.2 Merge both datasets

In [16]:
```
df_train_merge = pd.merge(df_train, df_labels)
df_train_merge.head()
```
Out[16]:

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wp
0	69572	6000.0	2011-03-14	Roman	1390	Roman	34.938093	-9.856322	
1	8776	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.698766	-2.147466	Z
2	34310	25.0	2013-02-25	Lottery Club	686	World vision	37.460664	-3.821329	N
3	67743	0.0	2013-01-28	Unicef	263	UNICEF	38.486161	-11.155298	Z Na
4	19728	0.0	2011-07-13	Action In A	0	Artisan	31.130847	-1.825359	

5 rows × 41 columns

In [17]:
```
df_train_merge.shape
```
Out[17]: (59400, 41)

As we can see above the merge has been done correctly because the number of rows is intact and the training set values has just one more column containing the training set labels

3.5.1.3 - Data Types

In [18]:
```
# Let's start by having a look at the type of each column
df_train_merge.dtypes
```

Out[18]:
```
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                     object
subvillage                object
region                    object
region_code                int64
district_code              int64
lga                       object
ward                      object
population                 int64
public_meeting            object
recorded_by               object
scheme_management         object
scheme_name               object
permit                    object
construction_year          int64
extraction_type           object
extraction_type_group     object
extraction_type_class     object
management                object
management_group          object
payment                   object
payment_type              object
water_quality             object
quality_group             object
quantity                  object
quantity_group            object
source                    object
source_type               object
source_class              object
waterpoint_type           object
waterpoint_type_group     object
status_group              object
dtype: object
```

3.5.1.4 - Null Values

In [19]:
```
# Let's see how the proportion of null values
(df_train_merge.isna().sum()/len(df_train_merge))*100
```

Out[19]:
```
id                        0.000000
amount_tsh                0.000000
date_recorded             0.000000
funder                    6.119529
gps_height                0.000000
installer                 6.153199
longitude                 0.000000
latitude                  0.000000
wpt_name                  0.000000
num_private               0.000000
basin                     0.000000
subvillage                0.624579
region                    0.000000
region_code               0.000000
district_code             0.000000
lga                       0.000000
ward                      0.000000
population                0.000000
public_meeting            5.612795
recorded_by               0.000000
scheme_management         6.526936
scheme_name              47.417508
permit                    5.144781
construction_year         0.000000
extraction_type           0.000000
extraction_type_group     0.000000
extraction_type_class     0.000000
management                0.000000
management_group          0.000000
payment                   0.000000
payment_type              0.000000
water_quality             0.000000
quality_group             0.000000
quantity                  0.000000
quantity_group            0.000000
source                    0.000000
source_type               0.000000
source_class              0.000000
waterpoint_type           0.000000
waterpoint_type_group     0.000000
status_group              0.000000
dtype: float64
```

In this case, we are going to fill NaN values just for categorical variables. In the next scrip (01_data_preprocessing) is where we will fill NaN values with calculated values such as the mode, mean, etc.

Column 'funder'

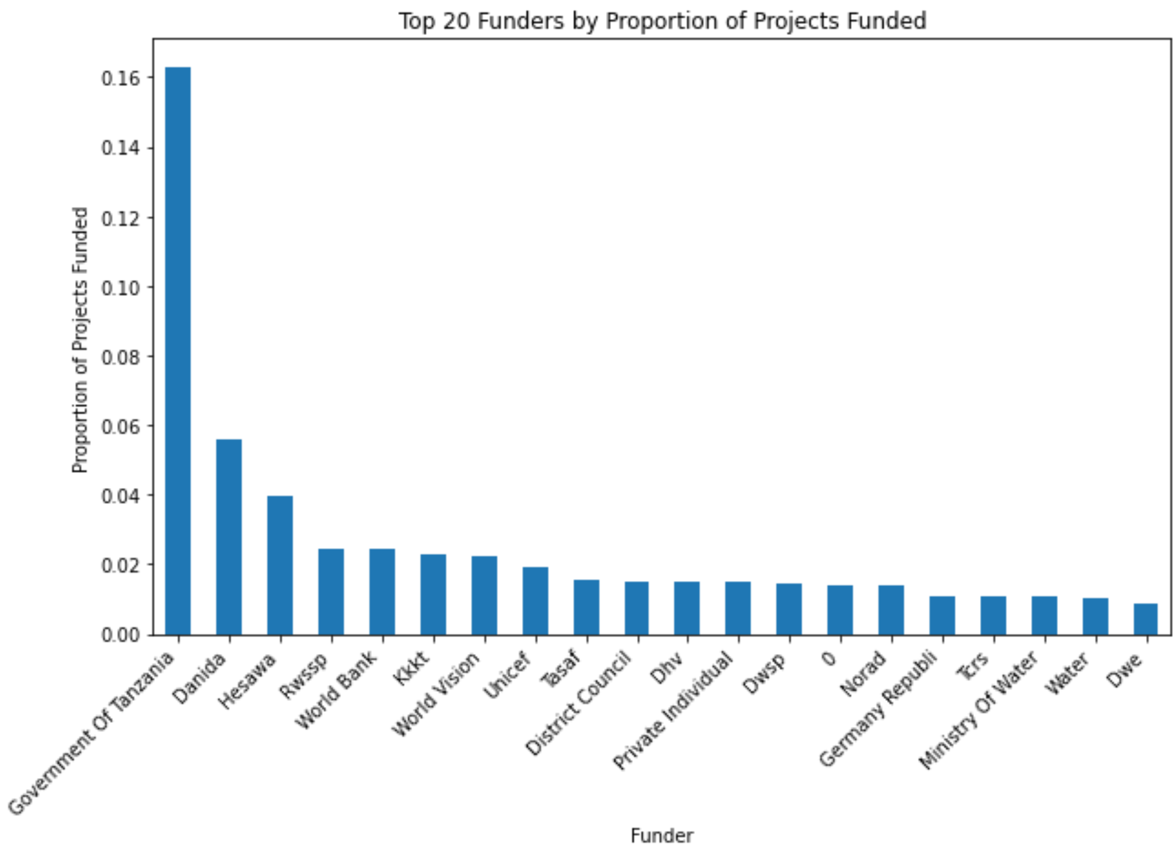
In [20]:
```
# Calculate value counts and then the proportions
funder_value_counts = df_train_merge['funder'].value_counts(normalize=True)

# Create a bar plot
plt.figure(figsize=(10,6))  # Sets the size of the figure
funder_value_counts.head(20).plot(kind='bar')  # Display the top 20 funders fo

# Set the title and labels
plt.title('Top 20 Funders by Proportion of Projects Funded')
plt.xlabel('Funder')
plt.ylabel('Proportion of Projects Funded')

# Rotate x-axis labels for better readability if needed
plt.xticks(rotation=45, ha='right')

# Show the plot
plt.show()
```



In [21]:
```
df_train_merge["funder"].value_counts(normalize=True)
```

Out[21]:
```
Government Of Tanzania    0.162898
Danida                    0.055841
Hesawa                    0.039487
Rwssp                     0.024639
World Bank                0.024191
                            ...
Kdpa                      0.000018
Dmk Anglican              0.000018
Kikundi Cha Akina Mama    0.000018
Dacp                      0.000018
Bukwang Church Saints     0.000018
Name: funder, Length: 1897, dtype: float64
```

In [22]:
```
# Looking at all the values in the funder column

# for valor in df_train_merge["funder"].unique():
#     print(valor)
```

In [23]:
```
# Handling NaN values with a filler string like 'Unknown'
df_train_merge['funder'] = df_train_merge['funder'].fillna('Unknown').astype(s

# Apply the mapping function to the 'funder' column
df_train_merge['funder_type'] = df_train_merge['funder'].apply(categorize_fund

# Check the categorized data
print(df_train_merge['funder_type'].value_counts())
```

```
Individual/Other          39410
Government                10017
International Aid          8468
Religious Organizations    1299
NGO                         146
Private Companies            60
Name: funder_type, dtype: int64
```

For the time being, we will advance with this categorization and decide later if we want to further investigate the Individual/Other category if necessary

Column 'installer'

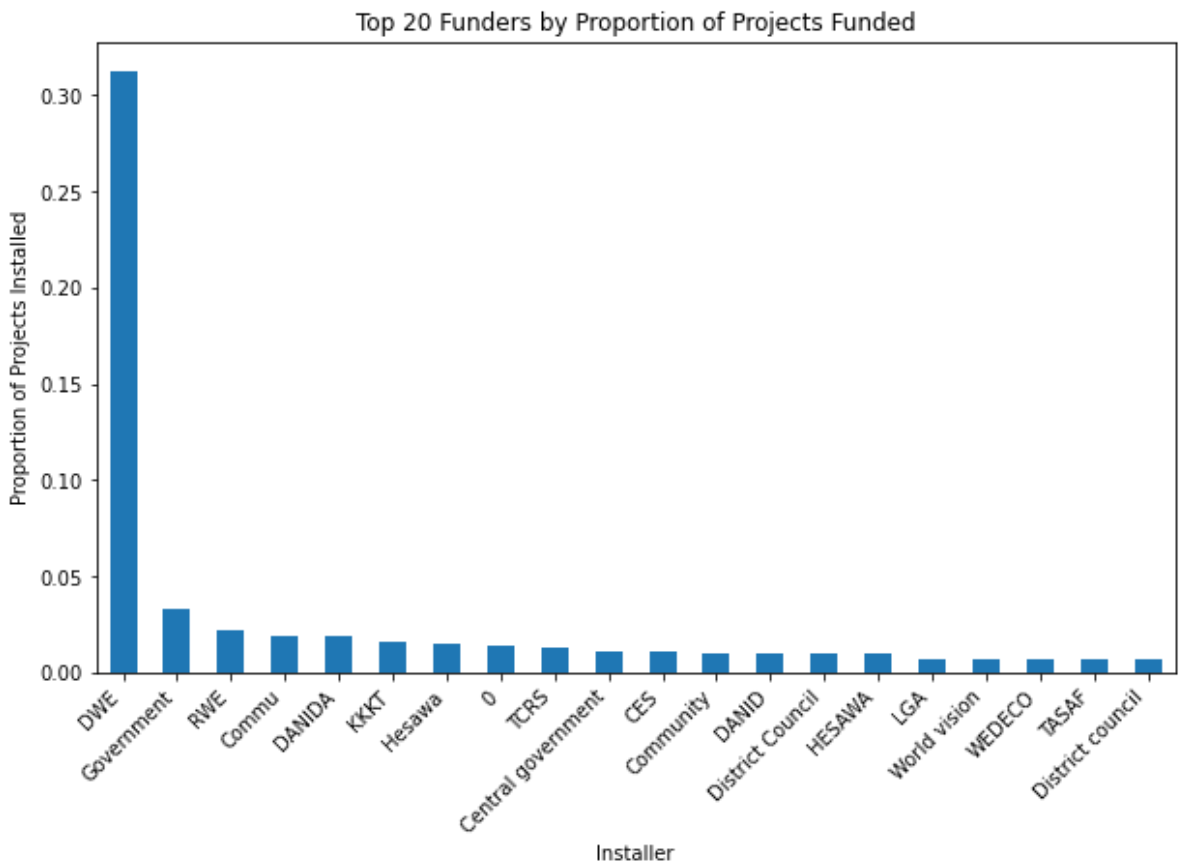
In [24]:
```
# Calculate value counts and then the proportions
funder_value_counts = df_train_merge['installer'].value_counts(normalize=True)

# Create a bar plot
plt.figure(figsize=(10,6))  # Sets the size of the figure
funder_value_counts.head(20).plot(kind='bar')  # Display the top 20 funders fo

# Set the title and labels
plt.title('Top 20 Funders by Proportion of Projects Funded')
plt.xlabel('Installer')
plt.ylabel('Proportion of Projects Installed')

# Rotate x-axis labels for better readability if needed
plt.xticks(rotation=45, ha='right')

# Show the plot
plt.show()
```



In [25]:
```
# Looking at all the values in the installer column

# for valor in df_train_merge["installer"].unique():
#     print(valor)
```

In [26]:
```
# Handling NaN values with a filler string like 'Unknown'
df_train_merge['installer'] = df_train_merge['installer'].fillna('Unknown').as

# Apply the mapping function to the 'installer' column
df_train_merge['installer_type'] = df_train_merge['installer'].apply(categoriz

# Now you can check your categorized data
print(df_train_merge['installer_type'].value_counts())
```

```
Other              34031
DWE                18121
Government          3753
Community           2338
Institutional        701
NGO                  327
Private Company      129
Name: installer_type, dtype: int64
```

For the time being, we will advance with this categorization and decide later if we want to further investigate the Individual/Other category if necessary

Column 'scheme_management'

In [27]:
```
df_train_merge["scheme_management"].value_counts(normalize=True)
```

Out[27]:
```
VWC                 0.662662
WUG                 0.093763
Water authority     0.056787
WUA                 0.051924
Water Board         0.049493
Parastatal          0.030258
Private operator    0.019145
Company             0.019109
Other               0.013796
SWC                 0.001747
Trust               0.001297
None                0.000018
Name: scheme_management, dtype: float64
```

We will categorize, based on this classification:

- Governmental Entities: Combine 'VWC' (Village Water Committee), 'Water authority', and 'Parastatal' into a single 'Government' category. These typically represent different layers or types of governmental involvement.
- Community Managed: Merge 'WUG' (Water User Group) and 'WUA' (Water User Association) into 'Community'. These are likely community-based management structures.
- Commercial Entities: Group 'Company' and 'Private operator' into 'Private Sector'. These likely represent privately managed schemes.
- Institutional Boards: Keep 'Water Board' as is if they represent formal institutional water management boards that don't fit into other categories.
- Other and Miscellaneous: Combine 'SWC', 'Trust', 'None', and 'Other' into 'Other'. These

categories might represent less common or unclear management structures.

In [28]:
```
# Apply the grouping function to the 'scheme_management' column
df_train_merge['scheme_management_grouped'] = df_train_merge['scheme_managemen

# Check the new value counts to see the grouped data
print(df_train_merge['scheme_management_grouped'].value_counts(normalize=True)
```

```
Government        0.700774
Community         0.136178
Other             0.081027
Water Board       0.046263
Private Sector    0.035758
Name: scheme_management_grouped, dtype: float64
```

Column 'scheme_name'

In [29]:
```
df_train_merge["scheme_name"].value_counts(normalize=True)
```

Out[29]:
```
K                          0.021835
None                       0.020619
Borehole                   0.017481
Chalinze wate              0.012967
M                          0.012807
                             ...
Fufu                       0.000032
Sola                       0.000032
Shigala Water Supply       0.000032
Kigondo                    0.000032
Kitangwi Water Supply      0.000032
Name: scheme_name, Length: 2696, dtype: float64
```

Given that there is almost 50% of unknown data, and the widespread of data, we will eliminate this column directly

In [30]:
```
# Start creating our drop list
drop_column_list = ['scheme_name']
```

Column 'num_private'

In [31]:
```
df_train_merge['num_private'].value_counts()
```

Out[31]:
```
0      58643
6         81
1         73
5         46
8         46
       ...
180        1
213        1
23         1
55         1
94         1
Name: num_private, Length: 65, dtype: int64
```

Given that num_private has no description and given that it has many values, we are going to add this to the drop list column

In [32]:
```
drop_column_list.append('num_private')
drop_column_list
```

Out[32]:
```
['scheme_name', 'num_private']
```

Column 'wpt_name '

In [33]:
```
df_train_merge['wpt_name'].value_counts()
```

Out[33]:
```
none                     3563
Shuleni                  1748
Zahanati                  830
Msikitini                 535
Kanisani                  323
                         ...
Lusenga Primary School      1
Kwa Deshi Kasugi            1
Nyamadebe                   1
Jumanne Mwansenga           1
Kwa Edga Ulaya              1
Name: wpt_name, Length: 37400, dtype: int64
```

No further information is added with this wpt_name column as it is the name of the waterpoint. We will add this to the drop_list

In [34]:
```
drop_column_list.append('wpt_name')
drop_column_list
```

Out[34]:
```
['scheme_name', 'num_private', 'wpt_name']
```

Column 'construction_year'

Converting 'construction_year' to object

In [35]:
```
df_train_merge['construction_year'] = df_train_merge['construction_year'].asty
print(df_train_merge['construction_year'].dtype)
```

```
object
```

In [36]:
```
df_train_merge.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59400 entries, 0 to 59399
Data columns (total 44 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                   59400 non-null  object 
 4   gps_height               59400 non-null  int64  
 5   installer                59400 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  object 
 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 
 41  funder_type              59400 non-null  object 
 42  installer_type           59400 non-null  object 
 43  scheme_management_grouped  59400 non-null  object 
dtypes: float64(3), int64(6), object(35)
memory usage: 20.4+ MB
```

Columns: 'subvillage' and 'region'

In [37]:
```
df_train_merge['subvillage'].value_counts()
```

Out[37]:
```
Madukani      508
Shuleni       506
Majengo       502
Kati          373
Mtakuja       262
             ...
Kaulungu B      1
Atsin           1
Mbizi A         1
Manankwa        1
Ntakaingo       1
Name: subvillage, Length: 19287, dtype: int64
```

In [38]:
```
df_train_merge['region'].value_counts()
```

Out[38]:
```
Iringa           5294
Shinyanga        4982
Mbeya            4639
Kilimanjaro      4379
Morogoro         4006
Arusha           3350
Kagera           3316
Mwanza           3102
Kigoma           2816
Ruvuma           2640
Pwani            2635
Tanga            2547
Dodoma           2201
Singida          2093
Mara             1969
Tabora           1959
Rukwa            1808
Mtwara           1730
Manyara          1583
Lindi            1546
Dar es Salaam     805
Name: region, dtype: int64
```

Having subvillage wouldn't give more insights to the model. There are more than 19k registrations of subvillages. Column 'region' alredy is a categorization of column 'subvillage' and so, we decide to add this column to the drop_list

In [39]:
```
drop_column_list.append('subvillage')
drop_column_list
```

Out[39]:
```
['scheme_name', 'num_private', 'wpt_name', 'subvillage']
```

Columns: 'lga', 'ward'

In [40]:
```
df_train_merge['lga'].value_counts()
```

Out[40]:
```
Njombe          2503
Arusha Rural    1252
Moshi Rural     1251
Bariadi         1177
Rungwe          1106
                ... 
Moshi Urban       79
Kigoma Urban      71
Arusha Urban      63
Lindi Urban       21
Nyamagana          1
Name: lga, Length: 125, dtype: int64
```

In [41]:
```
df_train_merge['ward'].value_counts()
```

Out[41]:
```
Igosi        307
Imalinyi     252
Siha Kati    232
Mdandu       231
Nduruma      217
            ... 
Kitete         1
Burungura      1
Simbay         1
Mkumbi         1
Mitole         1
Name: ward, Length: 2092, dtype: int64
```

As we already have column 'region' and columns: 'lga' and 'ward' are geographic locations. To avoid multiollinearity we will add 'lga'and 'ward' to the drop_list

In [42]:
```
drop_column_list.append('lga')
drop_column_list.append('ward')

drop_column_list
```

Out[42]:
```
['scheme_name', 'num_private', 'wpt_name', 'subvillage', 'lga', 'ward']
```

Columns: 'recorded_by'

In [43]:
```
df_train_merge['recorded_by'].value_counts()
```

Out[43]:
```
GeoData Consultants Ltd    59400
Name: recorded_by, dtype: int64
```

In [44]:
```
# Drop recorded_by column since it's constant and should be ignored
drop_column_list.append('recorded_by')
drop_column_list
```

Out[44]:
```
['scheme_name',
 'num_private',
 'wpt_name',
 'subvillage',
 'lga',
 'ward',
 'recorded_by']
```

Dropping the columns list

In [45]:
```
# Carry out the dropping
df_train_merge = df_train_merge.drop(drop_column_list, axis=1)
```

In [46]:
```
df_train_merge.columns
```

Out[46]:
```
Index(['id', 'amount_tsh', 'date_recorded', 'funder', 'gps_height',
       'installer', 'longitude', 'latitude', 'basin', 'region', 'region_cod
e',
       'district_code', 'population', 'public_meeting', 'scheme_management',
       'permit', 'construction_year', 'extraction_type',
       'extraction_type_group', 'extraction_type_class', 'management',
       'management_group', 'payment', 'payment_type', 'water_quality',
       'quality_group', 'quantity', 'quantity_group', 'source', 'source_typ
e',
       'source_class', 'waterpoint_type', 'waterpoint_type_group',
       'status_group', 'funder_type', 'installer_type',
       'scheme_management_grouped'],
      dtype='object')
```

3.5.1.3.2 - Transforming column types

In [47]:
```
df_train_merge.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59400 entries, 0 to 59399
Data columns (total 37 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                    59400 non-null  object
 4   gps_height                59400 non-null  int64
 5   installer                 59400 non-null  object
 6   longitude                 59400 non-null  float64
 7   latitude                  59400 non-null  float64
 8   basin                     59400 non-null  object
 9   region                    59400 non-null  object
 10  region_code               59400 non-null  int64
 11  district_code             59400 non-null  int64
 12  population                59400 non-null  int64
 13  public_meeting            56066 non-null  object
```

Column 'date_recorded'

In [48]:
```
df_train_merge['date_recorded']
```

Out[48]:
```
0        2011-03-14
1        2013-03-06
2        2013-02-25
3        2013-01-28
4        2011-07-13
            ...
59395    2013-05-03
59396    2011-05-07
59397    2011-04-11
59398    2011-03-08
59399    2011-03-23
Name: date_recorded, Length: 59400, dtype: object
```

In [49]:
```
df_train_merge['date_recorded'] = pd.to_datetime(df_train_merge['date_recorded

print(df_train_merge['date_recorded'].dtype)
```

```
datetime64[ns]
```

As can be seen, the date_recorded column has to be converted to date type

Column 'public_meeting'

In [50]:
```
print(df_train_merge['public_meeting'].dtype)

object
```

Column 'permit'

In [51]:
```
print(df_train_merge['permit'].dtype)
```

```
object
```

3.5.1.3.3 - Cleaning the dataset

In [52]:
```
# Apply the cleaning function to each object-type column in the DataFrame
for col in df_train_merge.select_dtypes(include='object').columns:
    df_train_merge[col] = df_train_merge[col].apply(clean_text)

# Display the cleaned DataFrame
df_train_merge.head()
```

Out[52]:

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	ba
0	69572	6000.0	2011-03-14	roman	1390	roman	34.938093	-9.856322	l nya
1	8776	0.0	2013-03-06	grumeti	1399	grumeti	34.698766	-2.147466	l victo
2	34310	25.0	2013-02-25	lottery club	686	world vision	37.460664	-3.821329	pang
3	67743	0.0	2013-01-28	unicef	263	unicef	38.486161	-11.155298	ruvu south cc
4	19728	0.0	2011-07-13	action in a	0	artisan	31.130847	-1.825359	l victo

5 rows × 37 columns

3.5.2 Descriptive Analysis

3.5.2.1 Univaried Analysis

Numerical columns

In [53]:
```
numeric_columns = df_train_merge.select_dtypes(include=[np.number])

# Let's exclude certain columns of numerical columns
numeric_columns
```

Out[53]:

	id	amount_tsh	gps_height	longitude	latitude	region_code	district_code	popula
0	69572	6000.0	1390	34.938093	-9.856322	11	5	
1	8776	0.0	1399	34.698766	-2.147466	20	2	
2	34310	25.0	686	37.460664	-3.821329	21	4	
3	67743	0.0	263	38.486161	-11.155298	90	63	
4	19728	0.0	0	31.130847	-1.825359	18	1	
...	...	...	...	...	...	...	...	
59395	60739	10.0	1210	37.169807	-3.253847	3	5	
59396	27263	4700.0	1212	35.249991	-9.070629	11	4	
59397	37057	0.0	0	34.017087	-8.750434	12	7	
59398	31282	0.0	0	35.861315	-6.378573	1	4	
59399	26348	0.0	191	38.104048	-6.747464	5	2	

59400 rows × 9 columns

In [54]:
```
numeric_columns = numeric_columns.drop(['id','longitude','latitude','region_co
```

In [55]:
```
numeric_columns
```

Out[55]:

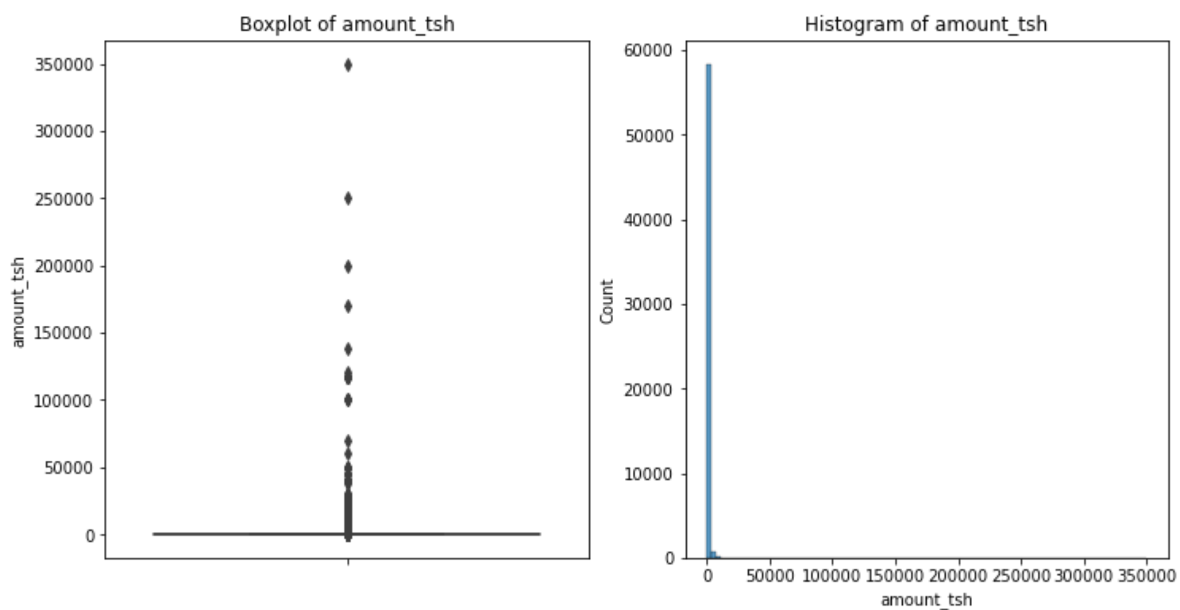
	amount_tsh	gps_height	population
0	6000.0	1390	109
1	0.0	1399	280
2	25.0	686	250
3	0.0	263	58
4	0.0	0	0
...	...	...	...
59395	10.0	1210	125
59396	4700.0	1212	56
59397	0.0	0	0
59398	0.0	0	0
59399	0.0	191	150

59400 rows × 3 columns

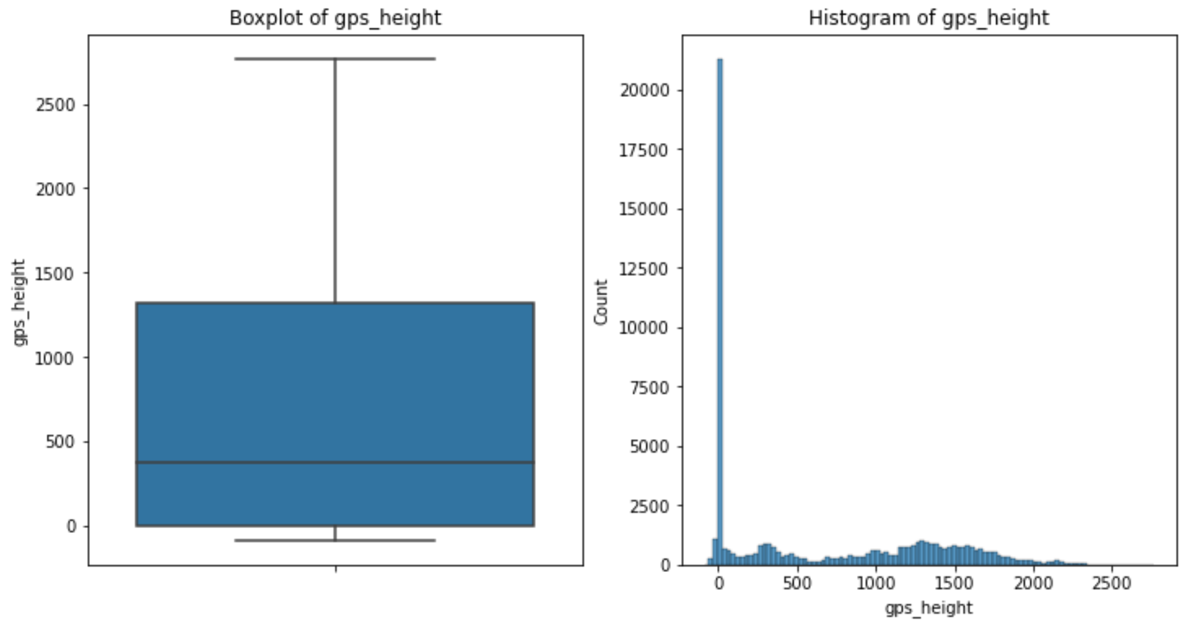
In [56]:
```
# Now let's analyze these numeric columns

analyze_numeric_stats_and_plots(df_train_merge, numeric_columns)
```

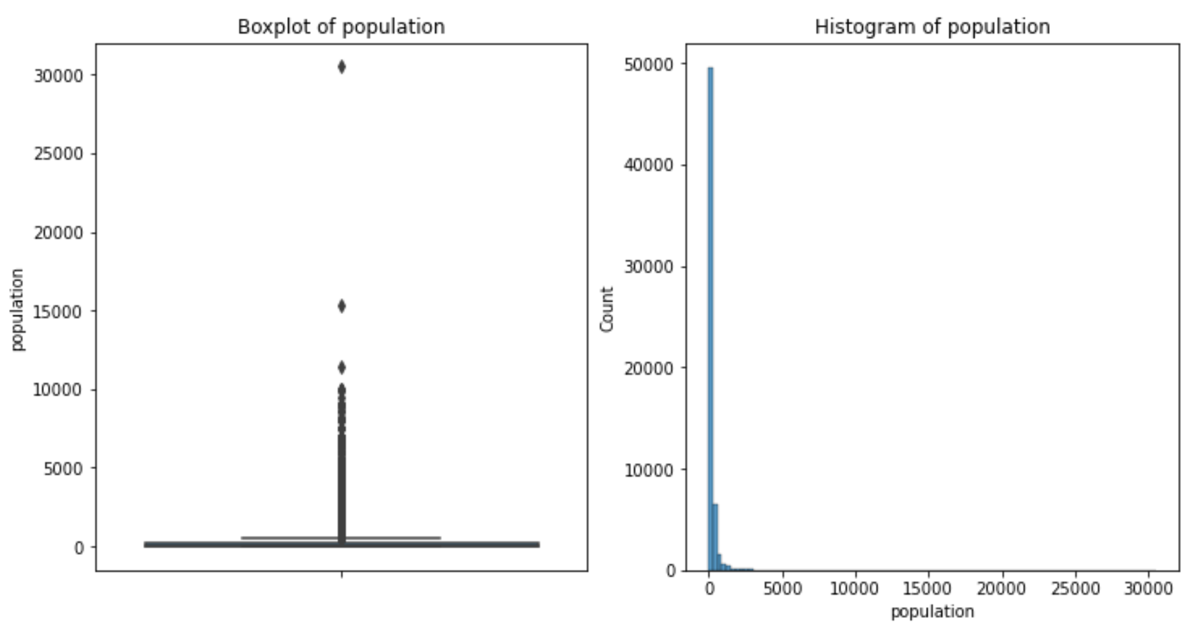
```
Stats for amount_tsh:
Max: 350000.0
Min: 0.0
Mean: 317.6503846801347
Median: 0.0
Standard Deviation: 2997.574558142169
Coefficient of Variation: 9.436709989067523
Skewness: 57.8077999458852
Kurtosis: 4903.543101955053
25th percentile (Q1): 0.0
50th percentile (Median): 0.0
75th percentile (Q3): 20.0
```




```
Stats for gps_height:
Max: 2770
Min: -90
Mean: 668.297239057239
Median: 369.0
Standard Deviation: 693.11635032505
Coefficient of Variation: 1.037137833013979
Skewness: 0.4624020849809572
Kurtosis: -1.2924401348688863
25th percentile (Q1): 0.0
50th percentile (Median): 369.0
75th percentile (Q3): 1319.25
```




```
Stats for population:
Max: 30500
Min: 0
Mean: 179.90998316498317
Median: 25.0
Standard Deviation: 471.48217573848035
Coefficient of Variation: 2.620655993870647
Skewness: 12.660713588843592
Kurtosis: 402.28011526096975
25th percentile (Q1): 0.0
50th percentile (Median): 25.0
75th percentile (Q3): 215.0
```



Categorical columns

In [57]:
```
categorical_columns = df_train_merge.select_dtypes(include=['object', 'categor
categorical_columns
```

Out[57]:
```
Index(['funder', 'installer', 'basin', 'region', 'public_meeting',
       'scheme_management', 'permit', 'extraction_type',
       'extraction_type_group', 'extraction_type_class', 'management',
       'management_group', 'payment', 'payment_type', 'water_quality',
       'quality_group', 'quantity', 'quantity_group', 'source', 'source_typ
e',
       'source_class', 'waterpoint_type', 'waterpoint_type_group',
       'status_group', 'funder_type', 'installer_type',
       'scheme_management_grouped'],
      dtype='object')
```

In [58]:
```
categorical_columns = categorical_columns.drop(['funder','installer','scheme_m
```

In [59]:
```
# Plot the distributions

# plot_categorical_proportions(df_train_merge, categorical_columns)
```

Here are the observations from the categorical columns:

- extraction_type_group and extraction_type are already classified in extraction_type_group_class so we will use the latter group
- we see that in management, VWC contains most of the data. management_group is then a grouping of all the data and thus has fewer categories. So we will use management_group instead of the management column
- columns: payment and payment_type have the same categories and distributions. We will thus keep one only. We chose payment_type.
- we see that in water_quality, soft contains most of the data. quality_group is then a grouping of all the data and thus has fewer categories. So we will use quality_group instead of the water_quality column
- columns: quantity and quantity_group have the same categories and distributions. We will thus keep one only. We chose quantity_group.
- looking at the column source we see different categories and then when looking at source_type we can see that it has joined certain groups from the source column successfully. We consider source_type to have a better description of the data. Moreover, we will disregard source_class because we consider that source_type will give us interesting insights to the data. If in the future we see that the model is not convincing with the number of categories from source_type, we will use source_class
- We will keep waterpoint_type_group over waterpoint_type because we consider it beneficial to have grouped communal standpipe and communal standpipe multiple together.

In [60]:
```
drop_categorical_columns = ['extraction_type_group', 'extraction_type', 'manag
```

In [61]:
```
# Drop the list of columns from df_train_merge
df_train_merge = df_train_merge.drop(drop_categorical_columns, axis=1)
```

3.5.1.5 Multivaried Analysis

In [62]:
```
numeric_columns.columns
```

Out[62]:
```
Index(['amount_tsh', 'gps_height', 'population'], dtype='object')
```

In [63]:
```
categorical_columns = categorical_columns.drop(drop_categorical_columns)
drop_categorical_columns
```

Out[63]:
```
['extraction_type_group',
 'extraction_type',
 'management',
 'payment',
 'water_quality',
 'quantity',
 'source',
 'source_class',
 'waterpoint_type_group']
```

In [64]:
```
# Let's join together numeric_columns and categorical_columns into a list that
# analysis function

combined_columns = numeric_columns.columns.tolist() + categorical_columns.toli
combined_columns
```

Out[64]:
```
['amount_tsh',
 'gps_height',
 'population',
 'basin',
 'region',
 'public_meeting',
 'permit',
 'extraction_type_class',
 'management_group',
 'payment_type',
 'quality_group',
 'quantity_group',
 'source_type',
 'waterpoint_type',
 'funder_type',
 'installer_type',
 'scheme_management_grouped']
```

In [65]:
```
df_train_merge.columns
```

Out[65]:
```
Index(['id', 'amount_tsh', 'date_recorded', 'funder', 'gps_height',
       'installer', 'longitude', 'latitude', 'basin', 'region', 'region_cod
e',
       'district_code', 'population', 'public_meeting', 'scheme_management',
       'permit', 'construction_year', 'extraction_type_class',
       'management_group', 'payment_type', 'quality_group', 'quantity_group',
       'source_type', 'waterpoint_type', 'status_group', 'funder_type',
       'installer_type', 'scheme_management_grouped'],
      dtype='object')
```

In [66]:
```
# Apply the function for multivaried analysis

# plot_grouped_charts(df_train_merge, 'status_group', combined_columns)
```

Contingency Tables

In [67]:
```
categorical_columns
```

Out[67]:
```
Index(['basin', 'region', 'public_meeting', 'permit', 'extraction_type_clas
s',
       'management_group', 'payment_type', 'quality_group', 'quantity_group',
       'source_type', 'waterpoint_type', 'funder_type', 'installer_type',
       'scheme_management_grouped'],
      dtype='object')
```

In [68]:
```
tables = generate_proportion_contingency_tables(df_train_merge, 'status_group'
```

In [69]:
```
# for i in categorical_columns:
#     print(f"Table for {i}:")
#     display(tables[i])
#     print("\n")  # Adds a newline for better separation
```

From the study above, we can have an insight of which variables are going to be determinant when running the classification model. These variables are: region, extraction_type_class, payment_type, quantity_group, waterpoint_type, scheme_management_grouped

4. Exporting the data

In [70]:
```
combined_columns.append('status_group')
```

In [71]:
```
combined_columns
```

Out[71]:
```
['amount_tsh',
 'gps_height',
 'population',
 'basin',
 'region',
 'public_meeting',
 'permit',
 'extraction_type_class',
 'management_group',
 'payment_type',
 'quality_group',
 'quantity_group',
 'source_type',
 'waterpoint_type',
 'funder_type',
 'installer_type',
 'scheme_management_grouped',
 'status_group']
```

In [72]:
```
df_train_merge[combined_columns].to_excel('df_train_transform.xlsx', index=Fal
```