1 Final Project Submission

Please fill out:

- Student name: **Brian Bentson**
- Student pace: self paced / part time / full time: **Full Time**
- Scheduled project review date/time:
- Instructor name: **James Irving**
- Blog post URL:
- Video of 5-min Non-Technical Presentation:

2 Table of Contents

Click to jump to matching Markdown Header.

3 INTRODUCTION

The Tanzanian Ministry of Water tracks vital information on water wells in its country to best ensure citizens are provided with a continual source of fresh water. A dataset housing this crucial information can be found HERE (https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/page/23/).

I will be utilizing this dataset to train a classification model to accurately predict which water wells are not functional and also to gain insights into potential reasons for water well failures.

3.1 Business Statement

It is undoubtedly obvious how crucial a consistent water supply is to every living thing in this world. Without it, life is not sustainable. A human can survive without food on average for about 1 to 2 months. However, a human can only survive 3 days without water! This time-frame without water can be shortened even further in climates which are extremely hot and humid. Tanzania, located on the east coast of Africa on the Indian Ocean, has parts of the country that are extremely hot and

humid. The average high and low temperature in the most populous city of Dar es Salaam during the year is 86˚F and 70˚F, respectively. Dar es Salaam is located on the coast and has a average relative humidity of 70%.

It is, therefore, extremely imperative to be able to supply a consistent source of fresh water for sustainment of human life in Tanzania. This begins with the evaluation of water wells in Tanzania with an emphasis on how reliability can be maximized to ensure a consistent supply of water is attainable. Unfortunately, our reliance on equipment (in this case a mechanical pump) means that 100% reliability can never be achieved. It is best to consider both reliability of the equipment and how quickly we can respond to an equipment failure and get it back to a running state. The response time to fixing a mechanical failure can be shortened with first predicting which water wells will fail. This key information can help maintenance organizations to ensure they have labor, tools and supplies ready to be mobilized in case of a failure. I will use machine learning to build a model to best predict water well failures in an attempt to understand what improvements can be made to factors such as funding, technology and maintenance operations.

3.2 Analysis Methodology

The dataset has information on 59,400 water wells in Tanzania, for which only 55% are fully operational based on this dataset. Information on these water wells includes many important factors that impact their operability and will be explored in order to provide insight into how reliability, and therefore accessibility, can be maximized. I will clean and explore the data to best be utilized with a classification machine learning model to predict failure.

More specifically, I believe the best model will prioritize determining which wells are not functioning even if that means that there are false positives. This is primarily due to the high cost of leaving a subset of the population without water for any given time. This approach will mean more money and time is spent on preventative maintenance ahead of the well failing, but should ensure the best reliability. In the context of a classification model, the model will be evaluated to maximize recall. Recall aims to maximize identifying true positives even if it means there will be false negatives.

4 DATA COLLECTION

4.1 Import Packages

In [622]:
```
#data wrangling and visualization packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('fivethirtyeight')
import statsmodels.api as sm
import scipy.stats as stats
#feature engineering packages
from sklearn.impute import SimpleImputer
from feature_engine import imputation as mdi

#feature selection packages
from feature_engine.selection import DropConstantFeatures, DropDuplicat

#modeling packages
from sklearn.model_selection import train_test_split
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression, LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier

#modeling evaluation packages
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import plot_roc_curve, roc_curve, auc
from sklearn.metrics import get_scorer

#optimization packages
from sklearn.model_selection import GridSearchCV
```

executed in 16ms, finished 20:49:43 2021-05-23

In [623]:
```
#notebook settings
pd.set_option("display.max_columns", 40)
pd.options.display.float_format = '{:,}'.format

import warnings
warnings.filterwarnings('ignore')
```

executed in 2ms, finished 20:49:43 2021-05-23

4.2 Global Functions

In [624]:

```
def col_eval(df, num_col=None, cat_cols=None, y_col='status_group', lab
    '''
    This function evaluates a single numeric feature or a list of categ
    features for feature engineering. It takes in a dataframe, a featur
    and provides detailed descriptive statistics and draws 4 informativ
    Keyword Arguments:
    - df: A dataframe
    - num_col: A single column of numeric data
    - cat_cols: A list of categorical columns
    - y_col: A dependent variable column. Default is 'status_group'
    - label_count: Number of labels to draw in bar graph
    - thresh: A threshold line drawn on the bar graph to indicate perce
    '''
    if num_col != None:
        #print the column name
        print(f'Column Name: {num_col}')
        print('\n')
        #print the number of unique values
        print(f'Number of unique values: {df[num_col].nunique()}')
        print('\n')
        #print the number of duplicate values
        print(f'There are {df[num_col].duplicated().sum()} duplicates')
        print('\n')
        #print the number of null values
        print(f'There are {df[num_col].isna().sum()} null values')
        print('\n')
        #print the number of values equal to 0
        print(f'There are {(df[num_col] == 0).sum()} zeros')
        print('\n')
        #print the value counts percentage
        print('Value Counts Percentage', '\n',
              df[num_col].value_counts(normalize=True, dropna=False).ro
        print('\n')
        #print descriptive statistics
        print('Descriptive Metrics:','\n',
              df[num_col].describe())
        #plot boxplot, histogram
        fig, ax = plt.subplots(ncols=2, nrows=2, figsize=(15,10))

        histogram = df[num_col].hist(ax=ax[0,0])
        ax[0,0].set_title(f'Distribution of {num_col}');

        scatter = df.plot(kind='scatter', x=num_col, y=y_col,ax=ax[0,1]
        ax[0,1].set_title(f'{y_col} vs {num_col}');

        boxplot = df.boxplot(column=num_col, ax=ax[1,0]);
        ax[1,0].set_title(f'Boxplot of {num_col}');

        sm.graphics.qqplot(df[num_col], dist=stats.norm, line='45', fit
        ax[1,1].set_title(f'QQ plot of {num_col}');
        plt.tight_layout()

        plt.show()
        return

    else:
```

```
57
58          for col in cat_cols:
59              print('=====================================================
60              #print the column name
61              print(f'Column Name: {col}')
62              print('\n')
63              #print the number of unique values
64              print(f'Number of unique values: {df[col].nunique()}')
65              print('\n')
66              #print the number of duplicate values
67              print(f'There are {df[col].duplicated().sum()} duplicates')
68              print('\n')
69              #print the number of null values
70              print(f'There are {df[col].isna().sum()} null values')
71              print('\n')
72              #print the number of values equal to '0'
73              print(f'There are {(df[col] == "0").sum()} zeros')
74              print('\n')
75              #print the value counts percentage
76              print('Value Counts Percentage', '\n',
77                    df[col].value_counts(dropna=False).round(2))
78              print('\n')
79
80              #plot barplot, histogram
81              fig, ax = plt.subplots(figsize=(15,10))
82
83              bar_graph = df[col].value_counts(normalize=True,
84                                               dropna=False)[:label_count
85              ax.axhline(y=thresh, color='red', linestyle='--',
86                         label=f'{thresh*100}% Threshold')
87              ax.set_title(f'{col} Value Counts')
88              ax.set_xlabel(f'{col} Labels')
89              ax.set_ylabel('Percentage')
90              ax.legend()
91
92              plt.tight_layout()
93
94              plt.show()
95          return
```

executed in 8ms, finished 20:49:43 2021-05-23

In [625]:

```
def rare_labels(df, col_name, thresh=.01):
    '''
    Function taken from Feature Engineering course on Udemy to group al
    labels below a certain threshold as "Rare".
    - df: A dataframe
    - col_name: A single column of data
    - thresh: A threshold of what to call a "Rare" label. Everything be
    equal to thethreshold will be deemed "Rare".
    '''
    df_temp = pd.Series(df[col_name].value_counts(normalize=True))

    rare_dict = {
        label: ('Rare' if label not in df_temp[df_temp >= thresh].index
        for label in df_temp.index
    }

    # now I replace the rare categories
    tmp = df[col_name].map(rare_dict)

    return tmp
```

executed in 2ms, finished 20:49:43 2021-05-23

In [626]:

```
#function to look at plots and stats of column with or without outlier
def model_eval(model, X_train, y_train, X_test, y_test,
               prev_model=None, prev_X_train=None, prev_y_train=None,
               prev_X_test=None, prev_y_test=None):
    '''
    This function takes in a fit model and provides classification eval
    metrics on that model. Optionally, a previous model can be supplied
    improvement metrics between the current model and the previous mode
    Keyword Arguments:
    - model: A fit model
    - X_train, y_train, X_test, y_test: Training and testing dataframes
    the "model" stated above was fit on.
    - prev_model: Another fit model
    - prev_X_train, prev_y_train, prev_X_test, prev_y_test: Training an
    testing dataframes which the previous model was fit on.
    '''
    #current model predictions on testing dataframe
    y_hat_test = model.predict(X_test)
    #get scores of current model on testing dataframe
    recall_model = get_scorer('recall')(model, X_test, y_test).round(2
    f1_model = get_scorer('f1')(model, X_test, y_test).round(2)
    accuracy_model = get_scorer('accuracy')(model, X_test, y_test).rou
    auc_model = get_scorer('roc_auc')(model, X_test, y_test).round(2)

    recall_model_train = get_scorer('recall')(model, X_train, y_train)
    #if statement to check for availability of a previous model and tra
    #and testing dataframes
    if prev_model != None:
        if prev_X_train is None:
            #if prevoius model has a different training and testing da

            #get previous model predictions and scores
            y_hat_test_prev = prev_model.predict(X_test)

            recall_prev = get_scorer('recall')(prev_model, X_test, y_te
            f1_prev = get_scorer('f1')(prev_model, X_test, y_test).rou
            accuracy_prev = get_scorer('accuracy')(prev_model, X_test,
            auc_prev = get_scorer('roc_auc')(prev_model, X_test, y_tes

            print('MODEL EVAL VS PREVIOUS (TEST)')
            print('==================================================

            #create dataframe comparing current and previous model
            df = pd.DataFrame(index=['Recall','F1','Accuracy','AUC'],
                              columns=['Previous Model','Current Model

            df.loc['Recall','Current Model'] = recall_model
            df.loc['Recall','Previous Model'] = recall_prev
            df.loc['F1','Current Model'] = f1_model
            df.loc['F1','Previous Model'] = f1_prev
            df.loc['Accuracy','Current Model'] = accuracy_model
            df.loc['Accuracy','Previous Model'] = accuracy_prev
            df.loc['AUC','Current Model'] = auc_model
            df.loc['AUC','Previous Model'] = auc_prev
            df.loc['Recall','Delta'] = (recall_model - recall_prev).rou
            df.loc['F1','Delta'] = (f1_model - f1_prev).round(2)
```

```
            df.loc['Accuracy','Delta'] = (accuracy_model - accuracy_pr
            df.loc['AUC','Delta'] = (auc_model - auc_prev).round(2)

            display(df)
            print('\n')

            #display current model scores, reports and graphs
            print(f"CURRENT MODEL: {'Overfit (Recall)' if recall_model_
            print('==========================================================
            print('\n')

            print(f'Recall on Training: {recall_model_train}')
            print(f'Recall on Test: {recall_model}')
            print('\n')

            print('Classification Reports--------------------------------
            print(classification_report(y_test, y_hat_test))

            print('Test Graphs-------------------------------------------
            fig, ax = plt.subplots(ncols=2, figsize=(15,8))

            plot_confusion_matrix(model, X_test, y_test, cmap='Blues',
                                  normalize='true',
                                  display_labels=['functional','non fu
                                  ax=ax[0]);

            plot_roc_curve(model, X_test, y_test, ax=ax[1]).ax_.plot([(
            plt.show()

            #display previous model graphs
            print("PREVIOUS MODEL")
            print('==========================================================
            fig, ax = plt.subplots(ncols=2, figsize=(15,8));

            plot_confusion_matrix(prev_model, X_test, y_test, cmap='Blu
                                  normalize='true',
                                  display_labels=['functional','non fu
                                  ax=ax[0]);

            plot_roc_curve(prev_model, X_test, y_test, ax=ax[1]).ax_.pl
            plt.show()

            plt.tight_layout()
        else:
            #if previous model has the same dataframes as current model
            y_hat_test_prev = prev_model.predict(prev_X_test)

            recall_prev = get_scorer('recall')(prev_model, prev_X_test,
                                               prev_y_test).round(2)
            f1_prev = get_scorer('f1')(prev_model, prev_X_test,
                                       prev_y_test).round(2)
            accuracy_prev = get_scorer('accuracy')(prev_model, prev_X_t
                                                   prev_y_test).round(2
            auc_prev = get_scorer('roc_auc')(prev_model, prev_X_test,
                                             prev_y_test).round(2)

            print('MODEL EVAL VS PREVIOUS (TEST)')
```

```
            print('=================================================

            df = pd.DataFrame(index=['Recall','F1','Accuracy','AUC'],
                              columns=['Previous Model','Current Model

            df.loc['Recall','Current Model'] = recall_model
            df.loc['Recall','Previous Model'] = recall_prev
            df.loc['F1','Current Model'] = f1_model
            df.loc['F1','Previous Model'] = f1_prev
            df.loc['Accuracy','Current Model'] = accuracy_model
            df.loc['Accuracy','Previous Model'] = accuracy_prev
            df.loc['AUC','Current Model'] = auc_model
            df.loc['AUC','Previous Model'] = auc_prev
            df.loc['Recall','Delta'] = (recall_model - recall_prev).rou
            df.loc['F1','Delta'] = (f1_model - f1_prev).round(2)
            df.loc['Accuracy','Delta'] = (accuracy_model - accuracy_pre
            df.loc['AUC','Delta'] = (auc_model - auc_prev).round(2)

            display(df)
            print('\n')


            print(f"CURRENT MODEL: {'Overfit (Recall)' if recall_model_
            print('=================================================
            print('\n')

            print(f'Recall on Training: {recall_model_train}')
            print(f'Recall on Test: {recall_model}')
            print('\n')

            print('Classification Reports-------------------------------
            print(classification_report(y_test, y_hat_test))

            print('Test Graphs-----------------------------------------
            fig, ax = plt.subplots(ncols=2, figsize=(15,8))

            plot_confusion_matrix(model, X_test, y_test, cmap='Blues',
                                  normalize='true',
                                  display_labels=['functional','non fu
                                  ax=ax[0]);

            plot_roc_curve(model, X_test, y_test, ax=ax[1]).ax_.plot([(
            plt.show()

            print("PREVIOUS MODEL")
            print('=================================================
            fig, ax = plt.subplots(ncols=2, figsize=(15,8));

            plot_confusion_matrix(prev_model, prev_X_test, prev_y_test
                                  normalize='true',
                                  display_labels=['functional','non fu
                                  ax=ax[0]);

            plot_roc_curve(prev_model, prev_X_test, prev_y_test,
                           ax=ax[1]).ax_.plot([0,1],[0,1]);
            plt.show()
```

```
            plt.tight_layout()
    else:
        #if there is no previous model, get current model metrics and
        print(f"CURRENT MODEL: {'Overfit (Recall)' if recall_model_tra
        print('=========================================================
        print('\n')
        print('Classification Reports-------------------------------
        print(classification_report(y_test, y_hat_test))

        print('Test Graphs-----------------------------------------
        fig, ax = plt.subplots(ncols=2, figsize=(15,8))

        plot_confusion_matrix(model, X_test, y_test, cmap='Blues',
                                     normalize='true',
                              display_labels=['functional','non functi
                              ax=ax[0]);

        plot_roc_curve(model, X_test, y_test, ax=ax[1]).ax_.plot([0,1]
        plt.show()


    return
```

executed in 17ms, finished 20:49:43 2021-05-23

In [627]:

```
def get_coefficients(model,X_train):
    '''
    Function to get coefficients of logistic regression models from cla
    '''


    coeffs = pd.Series(model.coef_.flatten(), index=X_train.columns)
    coeffs['intercept'] = model.intercept_[0]
    return coeffs
```

executed in 2ms, finished 20:49:43 2021-05-23

4.3 Import Data into Pandas

In [628]:
```
#create dataframe
df_original = pd.read_csv('Data/water_well_data.csv')
df_original.head()
```

executed in 319ms, finished 20:49:44 2021-05-23

Out[628]:

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	w
0	69572	6,000.0	2011-03-14	Roman	1390	Roman	34.93809275	-9.85632177	
1	8776	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.6987661	-2.14746569	
2	34310	25.0	2013-02-25	Lottery Club	686	World vision	37.46066446	-3.82132853	
3	67743	0.0	2013-01-28	Unicef	263	UNICEF	38.48616088	-11.15529772	N
4	19728	0.0	2011-07-13	Action In A	0	Artisan	31.13084671	-1.82535885	

5 rows × 41 columns

4.4 Data Schema

Taken from: https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/page/25/ (https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/page/25/)

- `amount_tsh` - Total static head (amount water available to waterpoint)
- `date_recorded` - The date the row was entered
- `funder` - Who funded the well
- `gps_height` - Altitude of the well
- `installer` - Organization that installed the well
- `longitude` - GPS coordinate
- `latitude` - GPS coordinate
- `wpt_name` - Name of the waterpoint if there is one
- `num_private` -
- `basin` - Geographic water basin
- `subvillage` - Geographic location
- `region` - Geographic location
- `region_code` - Geographic location (coded)
- `district_code` - Geographic location (coded)
- `lga` - Geographic location
- `ward` - Geographic location

- `population` - Population around the well
- `public_meeting` - True/False
- `recorded_by` - Group entering this row of data
- `scheme_management` - Who operates the waterpoint
- `scheme_name` - Who operates the waterpoint
- `permit` - If the waterpoint is permitted
- `construction_year` - Year the waterpoint was constructed
- `extraction_type` - The kind of extraction the waterpoint uses
- `extraction_type_group` - The kind of extraction the waterpoint uses
- `extraction_type_class` - The kind of extraction the waterpoint uses
- `management` - How the waterpoint is managed
- `management_group` - How the waterpoint is managed
- `payment` - What the water costs
- `payment_type` - What the water costs
- `water_quality` - The quality of the water
- `quality_group` - The quality of the water
- `quantity` - The quantity of water
- `quantity_group` - The quantity of water
- `source` - The source of the water
- `source_type` - The source of the water
- `source_class` - The source of the water
- `waterpoint_type` - The kind of waterpoint
- `waterpoint_type_group` - The kind of waterpoint

Supplemental Resources

Column abbrevation meanings: https://pdf.usaid.gov/pdf_docs/PA00JZJ5.pdf (https://pdf.usaid.gov/pdf_docs/PA00JZJ5.pdf)

4.5 Investigate Data

I will briefly investigate null values, column names and data types before diving into more specific preprocessing.

In [629]:
```
#evaluate dataframe null values, column names and data types
df_original.info()
```

executed in 65ms, finished 20:49:44 2021-05-23

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59400 entries, 0 to 59399
Data columns (total 41 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     59400 non-null  int64  
 1   amount_tsh             59400 non-null  float64
 2   date_recorded          59400 non-null  object 
 3   funder                 55765 non-null  object 
 4   gps_height             59400 non-null  int64  
 5   installer              55745 non-null  object 
 6   longitude              59400 non-null  float64
 7   latitude               59400 non-null  float64
 8   wpt_name               59400 non-null  object 
 9   num_private            59400 non-null  int64  
 10  basin                  59400 non-null  object 
 11  subvillage             59029 non-null  object 
 12  region                 59400 non-null  object 
 13  region_code            59400 non-null  int64  
 14  district_code          59400 non-null  int64  
 15  lga                    59400 non-null  object 
 16  ward                   59400 non-null  object 
 17  population             59400 non-null  int64  
 18  public_meeting         56066 non-null  object 
 19  recorded_by            59400 non-null  object 
 20  scheme_management      55523 non-null  object 
 21  scheme_name            31234 non-null  object 
 22  permit                 56344 non-null  object 
 23  construction_year      59400 non-null  int64  
 24  extraction_type        59400 non-null  object 
 25  extraction_type_group  59400 non-null  object 
 26  extraction_type_class  59400 non-null  object 
 27  management             59400 non-null  object 
 28  management_group       59400 non-null  object 
 29  payment                59400 non-null  object 
 30  payment_type           59400 non-null  object 
 31  water_quality          59400 non-null  object 
 32  quality_group          59400 non-null  object 
 33  quantity               59400 non-null  object 
 34  quantity_group         59400 non-null  object 
 35  source                 59400 non-null  object 
 36  source_type            59400 non-null  object 
 37  source_class           59400 non-null  object 
 38  waterpoint_type        59400 non-null  object 
 39  waterpoint_type_group  59400 non-null  object 
 40  status_group           59400 non-null  object 
dtypes: float64(3), int64(7), object(31)
memory usage: 18.6+ MB
```

OBSERVATIONS

- Many columns to explore for null value imputation
- Column names are already standardized
- Data types will require further evaluation during feature engineering

In [630]:
```
#evaluate numerical data descriptive statistics
df_original.describe()
```

executed in 30ms, finished 20:49:44 2021-05-23

Out[630]:

	id	amount_tsh	gps_height	longitude	
count	59,400.0	59,400.0	59,400.0	59,400.0	
mean	37,115.131767676765	317.6503846801347	668.297239057239	34.077426692028794	-5.7060326
std	21,453.12837131775	2,997.574558142169	693.11635032505	6.567431845646531	2.9460190
min	0.0	0.0	-90.0	0.0	-11.
25%	18,519.75	0.0	0.0	33.09034738	-8.5
50%	37,061.5	0.0	369.0	34.90874343	-5.0215966
75%	55,656.5	20.0	1,319.25	37.17838657	-3.32615563
max	74,247.0	350,000.0	2,770.0	40.34519307	

OBSERVATIONS

- Many of these numerical features should be transformed into a categorical feature
- `num_private`, `construction_year`, `population`, and `district_code` has a minimum of 0 which may be a placeholder for unknown
- `amount_tsh` needs to be explored further as there seem to be a lot of 0's as the median is 0 while the mean is 317.

5 DATA CLEANING

In this section, I will focus on understanding the raw data quality and cleaning the data in preparation for data exploration, visualization and modeling.

In [631]:
```
#create df_clean dataframe
df_clean = df_original.copy()
```

executed in 13ms, finished 20:49:44 2021-05-23

5.1 Feature Evaluation

In this section I will focus on understanding each column and outlining the actions I will take in dedicated preprocessing sections later on.

In [632]:
```
#create list of all columns
num_cols = df_clean.select_dtypes(exclude='object').columns
cat_cols = df_clean.select_dtypes(include='object').columns
print(f'There are {len(num_cols)} numerical columns: \n {num_cols}')
print('\n')
print(f'There are {len(cat_cols)} numerical columns: \n {cat_cols}')
```

executed in 20ms, finished 20:49:44 2021-05-23

```
There are 10 numerical columns:
 Index(['id', 'amount_tsh', 'gps_height', 'longitude', 'latitude',
       'num_private', 'region_code', 'district_code', 'population',
       'construction_year'],
      dtype='object')


There are 31 numerical columns:
 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_t
ype',
       'source_class', 'waterpoint_type', 'waterpoint_type_group',
       'status_group'],
      dtype='object')
```

In [633]:
```
1 #display first 5 rows of numeric columns
2 df_clean.head()
```

executed in 18ms, finished 20:49:44 2021-05-23

Out[633]:

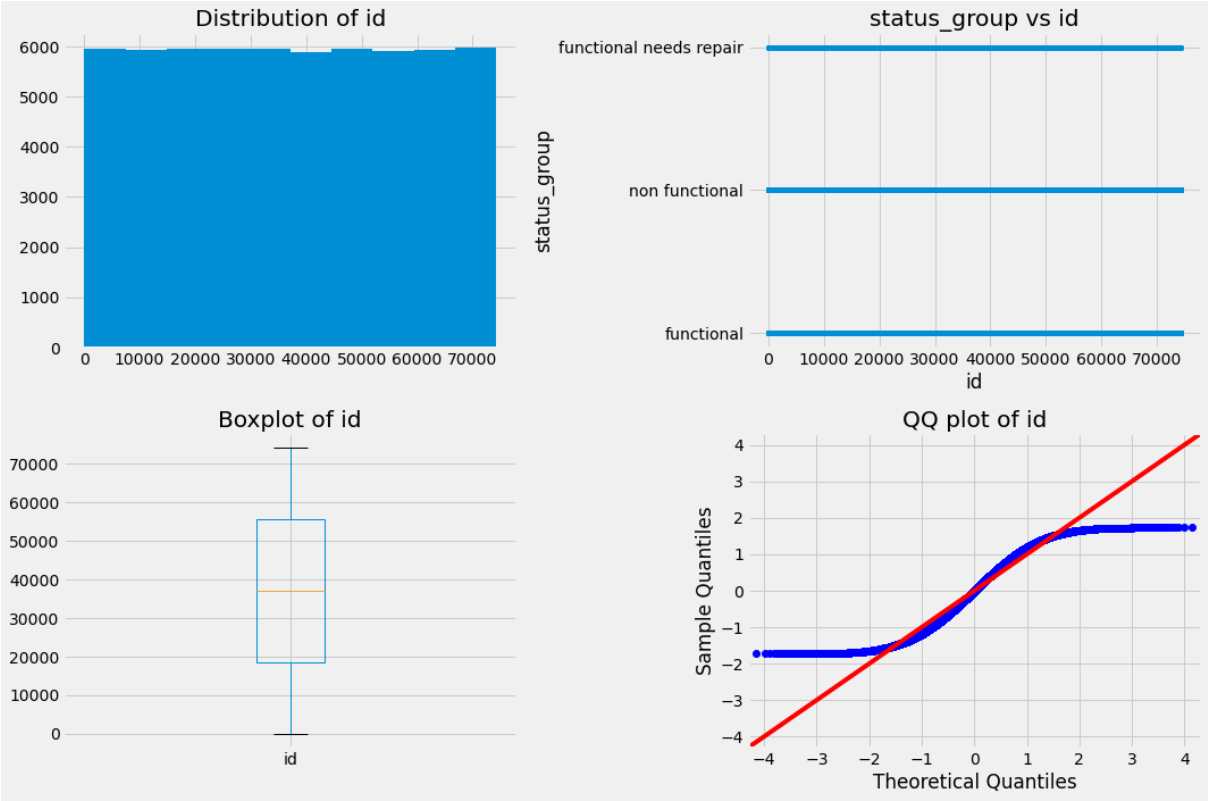
	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	w
0	69572	6,000.0	2011-03-14	Roman	1390	Roman	34.93809275	-9.85632177	
1	8776	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.6987661	-2.14746569	
2	34310	25.0	2013-02-25	Lottery Club	686	World vision	37.46066446	-3.82132853	
3	67743	0.0	2013-01-28	Unicef	263	UNICEF	38.48616088	-11.15529772	N
4	19728	0.0	2011-07-13	Action In A	0	Artisan	31.13084671	-1.82535885	

5 rows × 41 columns

In [634]:
```
#eval id feature
col_eval(df_clean, num_col='id')
```

executed in 642ms, finished 20:49:45 2021-05-23

```
Column Name: id


Number of unique values: 59400


There are 0 duplicates


There are 0 null values


There are 1 zeros


Value Counts Percentage
 2047    0.0
72310   0.0
49805   0.0
51852   0.0
62091   0.0
         ..
46396   0.0
36155   0.0
34106   0.0
38200   0.0
0       0.0
Name: id, Length: 59400, dtype: float64


Descriptive Metrics:
 count              59,400.0
mean    37,115.131767676765
std      21,453.12837131775
min                     0.0
25%               18,519.75
50%                37,061.5
75%                55,656.5
max                74,247.0
Name: id, dtype: float64
```



OBSERVATIONS

- `id` should be changed to categorical as it is a unique identifier for each well

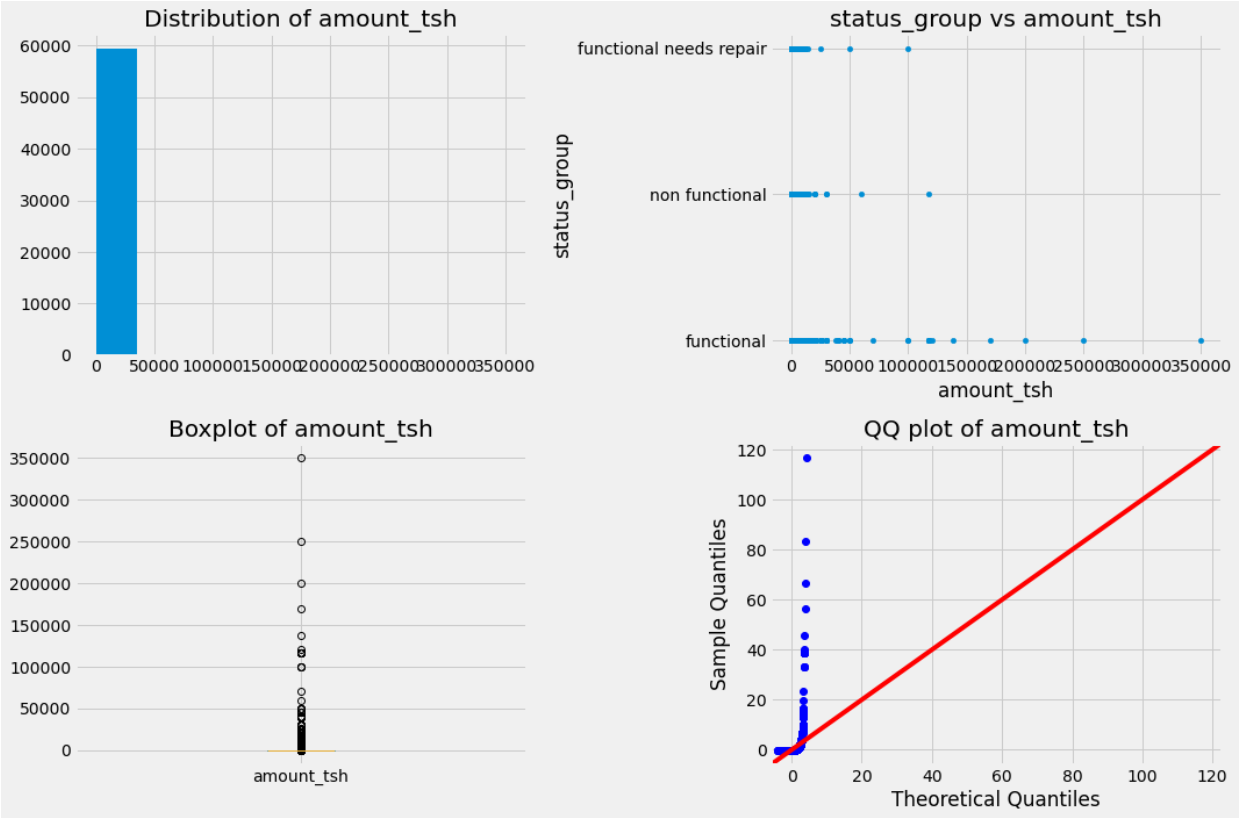
ACTIONS

- Will recast `id` as categorical

In [635]:
```
#eval amount_tsh feature
col_eval(df_clean, num_col='amount_tsh')
```

executed in 597ms, finished 20:49:45 2021-05-23

```
Column Name: amount_tsh


Number of unique values: 98


There are 59302 duplicates


There are 0 null values


There are 41639 zeros


Value Counts Percentage
 0.0          70.0
500.0         5.0
50.0          4.0
1,000.0       3.0
20.0          2.0
              ...
8,500.0       0.0
6,300.0       0.0
220.0         0.0
138,000.0     0.0
12.0          0.0
Name: amount_tsh, Length: 98, dtype: float64


Descriptive Metrics:
 count             59,400.0
mean      317.6503846801347
std      2,997.574558142169
min                     0.0
25%                     0.0
50%                     0.0
75%                    20.0
max                350,000.0
Name: amount_tsh, dtype: float64
```



OBSERVATIONS

- `amount_tsh` has many 0's and needs to be evaluated for imputation and outliers
- there are extreme outliers on the high end

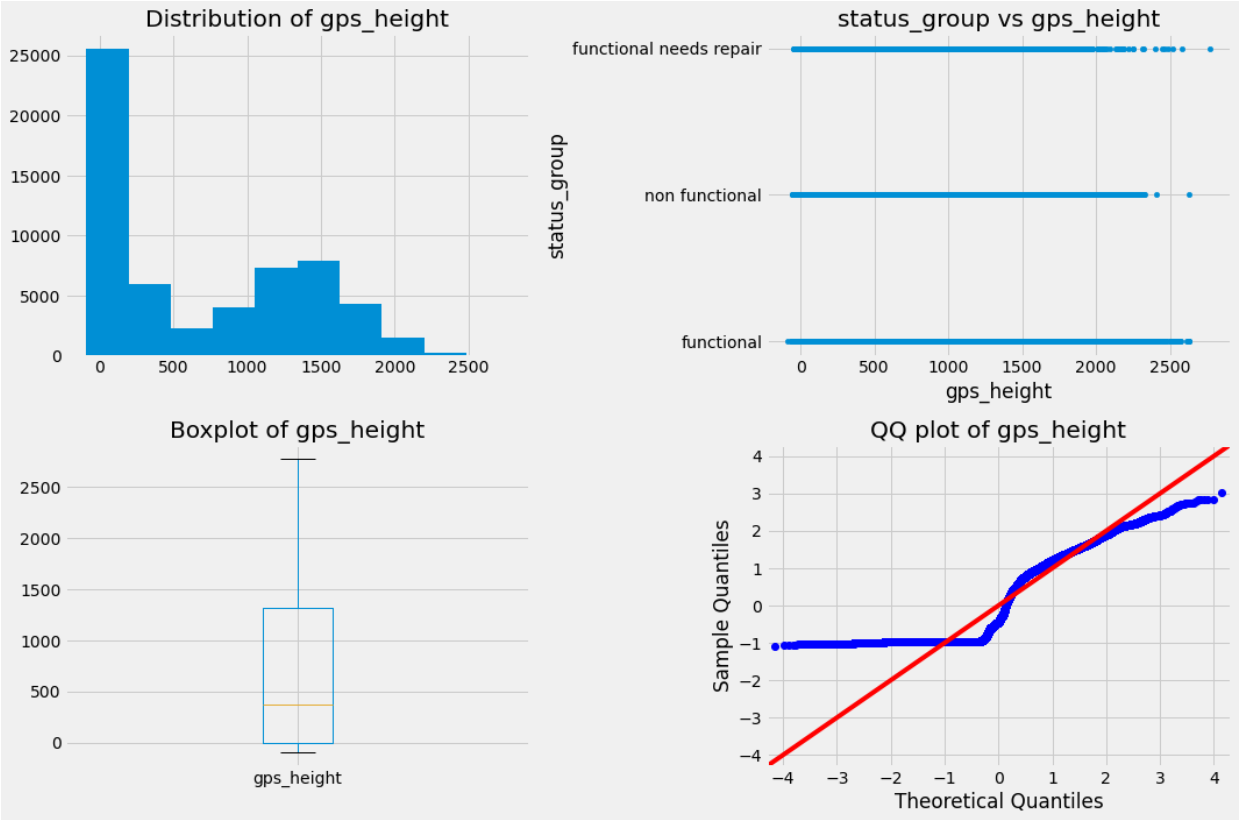
ACTIONS

- I will remove outliers but keep 0's as I believe they do appropriately represent the amount of head on the waterpoint, especially because some of the water comes from rain water capture, lakes and rivers, all of which would have 0 static head.

In [636]:
```
#eval gps_height
col_eval(df_clean, num_col='gps_height')
```

executed in 568ms, finished 20:49:46 2021-05-23

```
Column Name: gps_height


Number of unique values: 2428


There are 56972 duplicates


There are 0 null values


There are 20438 zeros


Value Counts Percentage
  0      34.0
-15      0.0
-16      0.0
-13      0.0
-20      0.0
        ...
 2285    0.0
 2424    0.0
 2552    0.0
 2413    0.0
 2385    0.0
Name: gps_height, Length: 2428, dtype: float64


Descriptive Metrics:
 count           59,400.0
mean    668.297239057239
std      693.11635032505
min                -90.0
25%                  0.0
50%                369.0
75%             1,319.25
max              2,770.0
Name: gps_height, dtype: float64
```



OBSERVATIONS

- There are many 0's which could mean the well is at sea level or it is a placeholder for unknown
- Does not seem to have extreme outliers

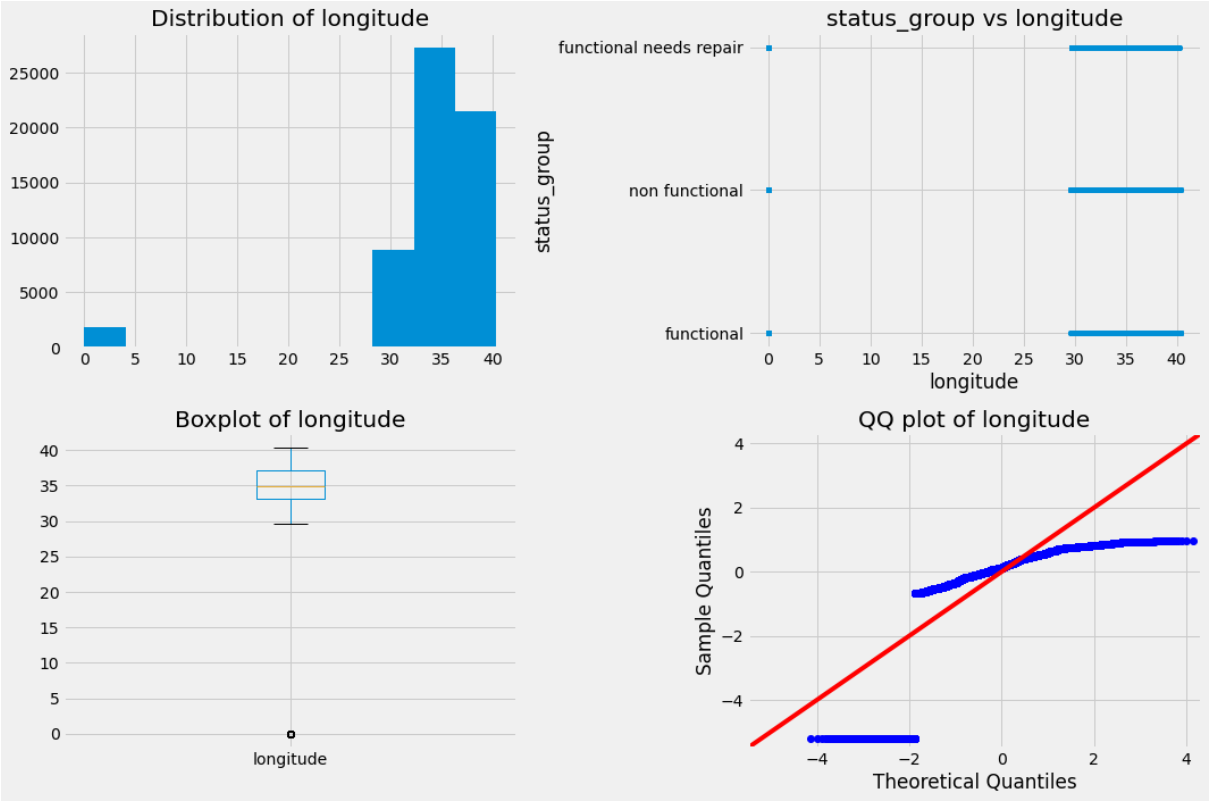
ACTIONS

- I will keep the 0's as I believe they indicate wells which are at sea level and not missing values.

In [637]:
```
1 col_eval(df_clean, num_col='longitude')
```

executed in 578ms, finished 20:49:46 2021-05-23

```
Column Name: longitude


Number of unique values: 57516


There are 1884 duplicates


There are 0 null values


There are 1812 zeros


Value Counts Percentage
 0.0           3.0
39.08887513   0.0
39.10530661   0.0
37.54340145   0.0
38.18053774   0.0
               ..
38.71052037   0.0
40.11702941   0.0
34.67296206   0.0
39.43360353   0.0
34.89083819   0.0
Name: longitude, Length: 57516, dtype: float64


Descriptive Metrics:
 count             59,400.0
mean    34.077426692028794
std      6.567431845646531
min                     0.0
25%             33.09034738
50%             34.90874343
75%             37.17838657
max             40.34519307
Name: longitude, dtype: float64
```



OBSERVATIONS

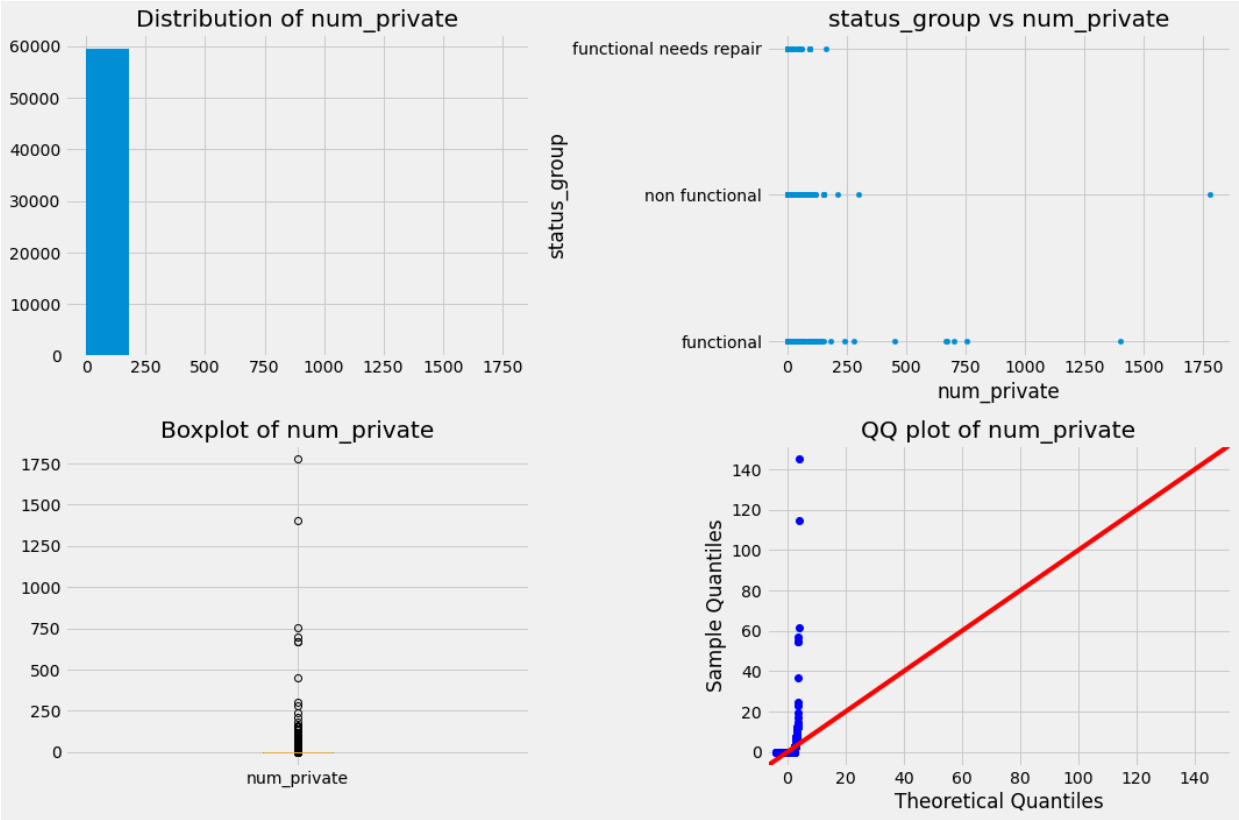
- Every numerical column is 0 when longitude is 0 (1812 rows or 3% of the data). I do not see a valid way to impute these columns.

- Drop 1812 rows

In [638]:
```
#eval num_private
col_eval(df_clean, num_col='num_private')
```

executed in 614ms, finished 20:49:47 2021-05-23

```
Column Name: num_private


Number of unique values: 65


There are 59335 duplicates


There are 0 null values


There are 58643 zeros


Value Counts Percentage
 0      99.0
6       0.0
1       0.0
5       0.0
8       0.0
       ...
180     0.0
213     0.0
23      0.0
55      0.0
94      0.0
Name: num_private, Length: 65, dtype: float64


Descriptive Metrics:
 count              59,400.0
mean    0.47414141414141414
std      12.236229810496686
min                      0.0
25%                      0.0
50%                      0.0
75%                      0.0
max                  1,776.0
Name: num_private, dtype: float64
```



OBSERVATIONS

- Because `num_private` is dominated by a single value (98.7% zero) and the data schema does not clearly state what it means, I will drop this column from the analysis.

ACTIONS

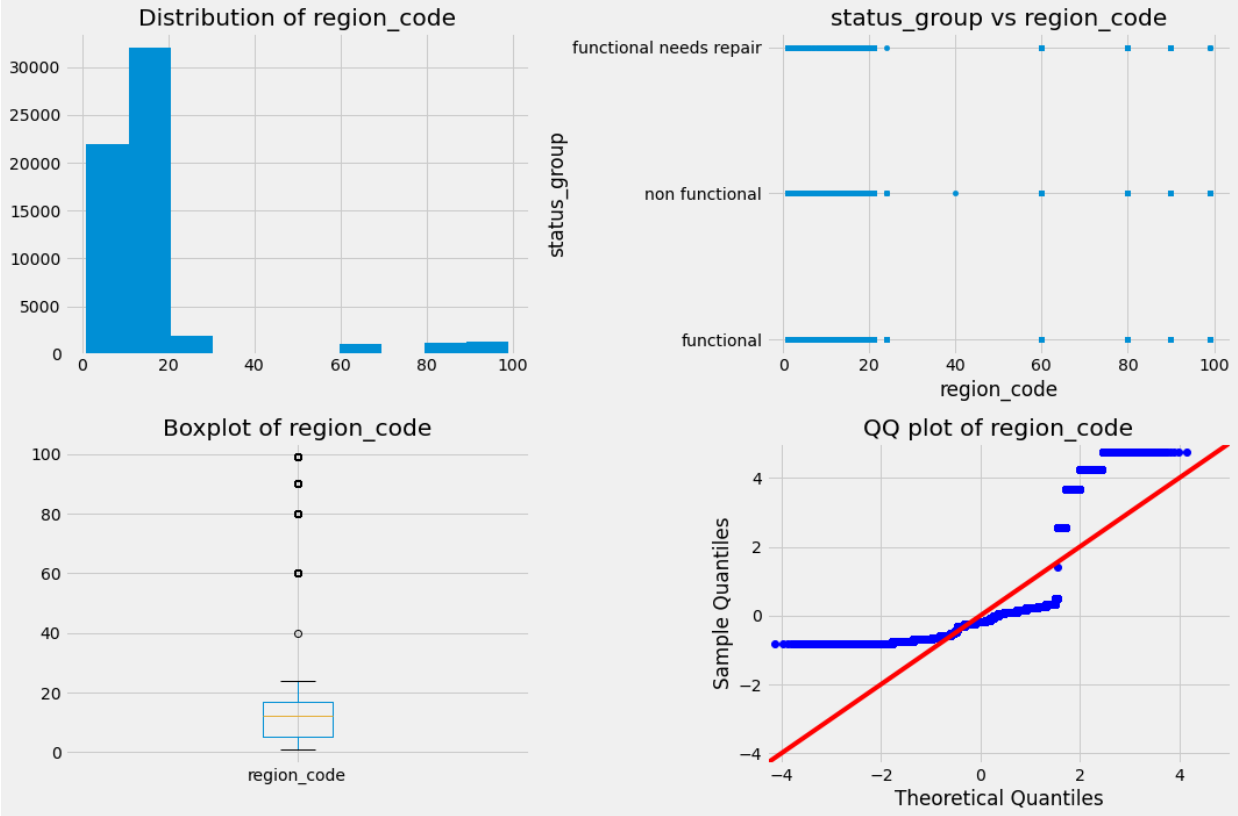
- Drop `num_private` column

In [639]:
```
#eval region_code
col_eval(df_clean, num_col='region_code')
```

executed in 464ms, finished 20:49:47 2021-05-23

```
Column Name: region_code


Number of unique values: 27


There are 59373 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 11                 9.0
17                 8.0
12                 8.0
3    7.000000000000001
5    7.000000000000001
18                 6.0
19                 5.0
2                  5.0
16                 5.0
10                 4.0
4                  4.0
1                  4.0
13                 4.0
14                 3.0
20                 3.0
15                 3.0
6                  3.0
21                 3.0
80                 2.0
60                 2.0
90                 2.0
7                  1.0
99                 1.0
9                  1.0
24                 1.0
8                  1.0
40                 0.0
Name: region_code, dtype: float64


Descriptive Metrics:
 count             59,400.0
mean    15.297003367003366
std      17.58740633733205
min                    1.0
25%                    5.0
50%                   12.0
```

```
75%                     17.0
max                     99.0
Name: region_code, dtype: float64
```



OBSERVATIONS

- `region_code` seems to represent a specific region and does not represent an actual numerical value.

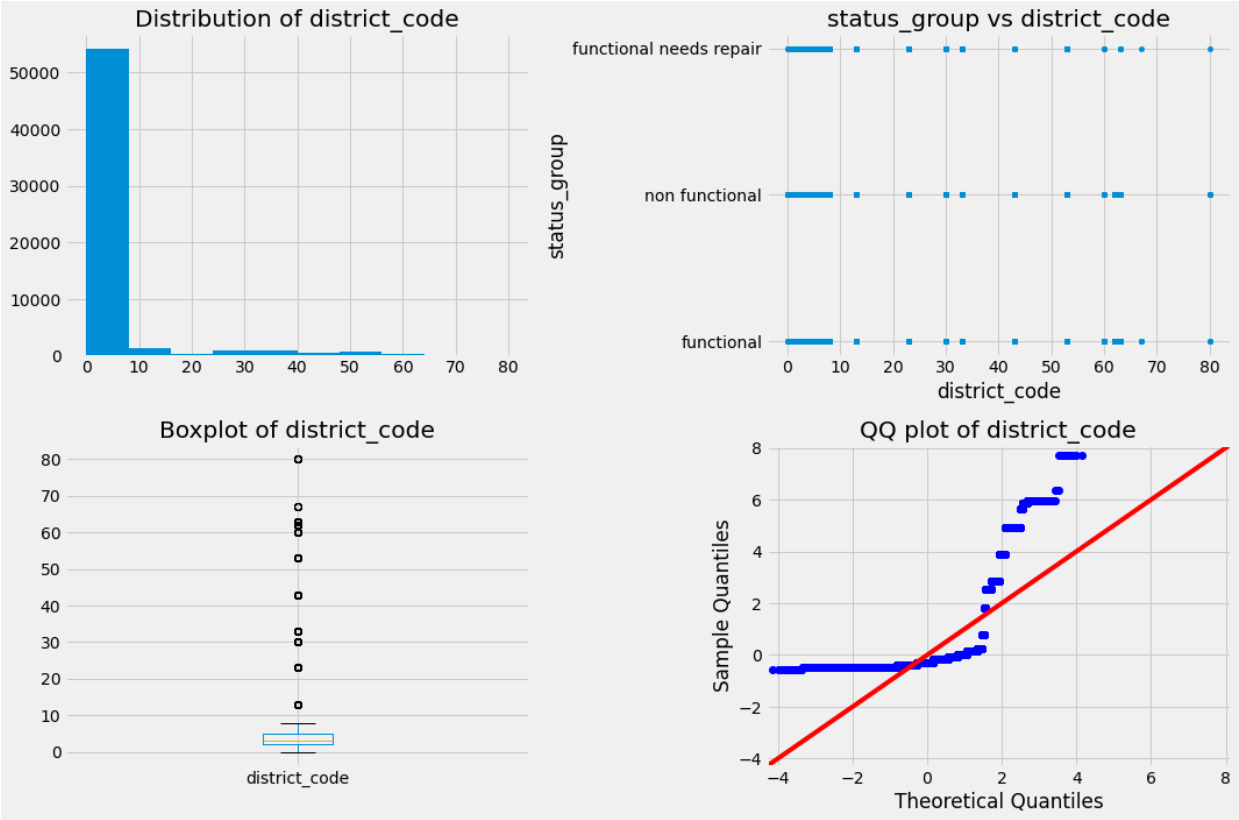
ACTIONS

- convert `region_code` to a categorical feature

In [640]:
```
#eval district_code
col_eval(df_clean, num_col='district_code')
```

executed in 681ms, finished 20:49:48 2021-05-23

```
Column Name: district_code


Number of unique values: 20


There are 59380 duplicates


There are 0 null values


There are 23 zeros


Value Counts Percentage
 1                 21.0
2                 19.0
3                 17.0
4                 15.0
5    7.000000000000001
6    7.000000000000001
7                  6.0
8                  2.0
30                 2.0
33                 1.0
53                 1.0
43                 1.0
13                 1.0
23                 0.0
63                 0.0
62                 0.0
60                 0.0
0                  0.0
80                 0.0
67                 0.0
Name: district_code, dtype: float64


Descriptive Metrics:
 count           59,400.0
mean    5.629747474747475
std     9.633648629454566
min                   0.0
25%                   2.0
50%                   3.0
75%                   5.0
max                  80.0
Name: district_code, dtype: float64
```



OBSERVATIONS

- `district_code` seems to represent a specific region and does not represent an actual numerical value.

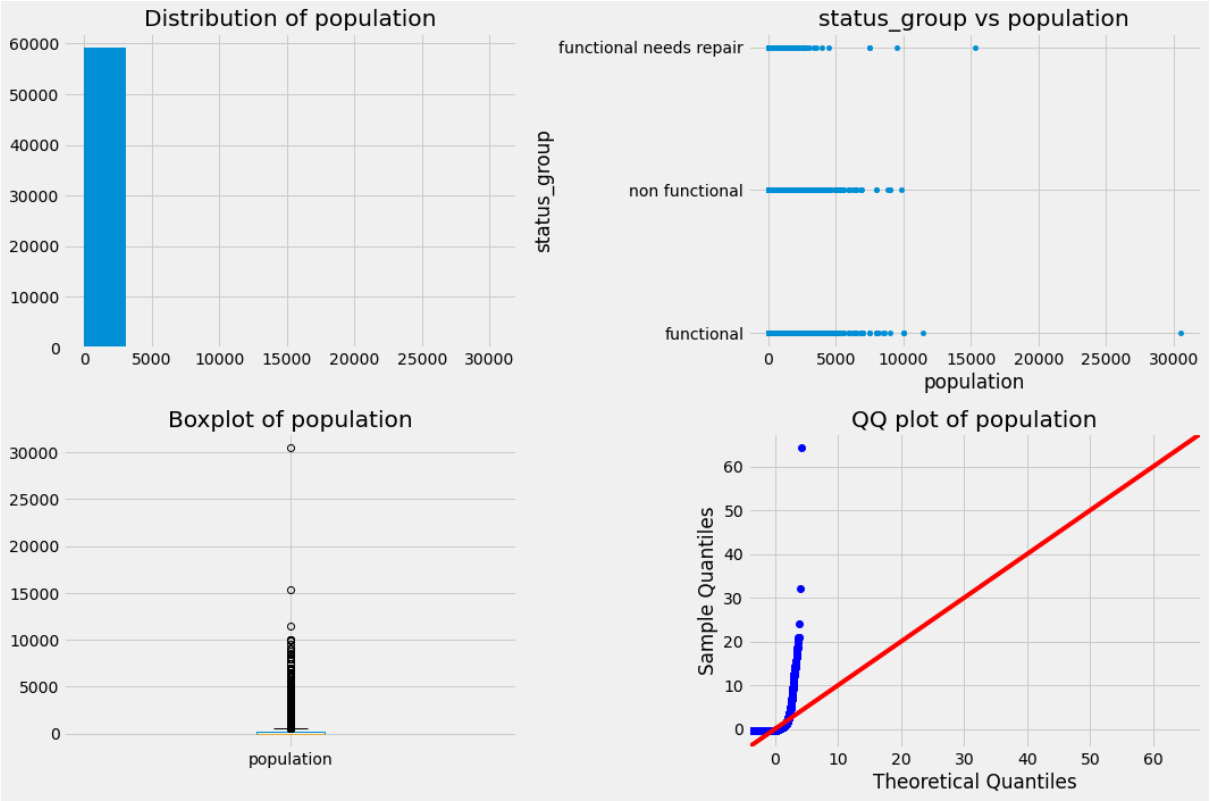
ACTIONS

- convert `district_code` to a categorical feature

In [641]:
```
#eval population
col_eval(df_clean, num_col='population')
```

executed in 492ms, finished 20:49:49 2021-05-23

```
Column Name: population


Number of unique values: 1049


There are 58351 duplicates


There are 0 null values


There are 21381 zeros


Value Counts Percentage
 0       36.0
1       12.0
200      3.0
150      3.0
250      3.0
        ...
3241     0.0
1960     0.0
1685     0.0
2248     0.0
1439     0.0
Name: population, Length: 1049, dtype: float64


Descriptive Metrics:
 count             59,400.0
mean    179.90998316498317
std     471.48217573848035
min                    0.0
25%                    0.0
50%                   25.0
75%                  215.0
max               30,500.0
Name: population, dtype: float64
```



OBSERVATIONS

- `population` has many 0's which could either represent wells with no population around them or a placeholder for unknown.

ACTIONS

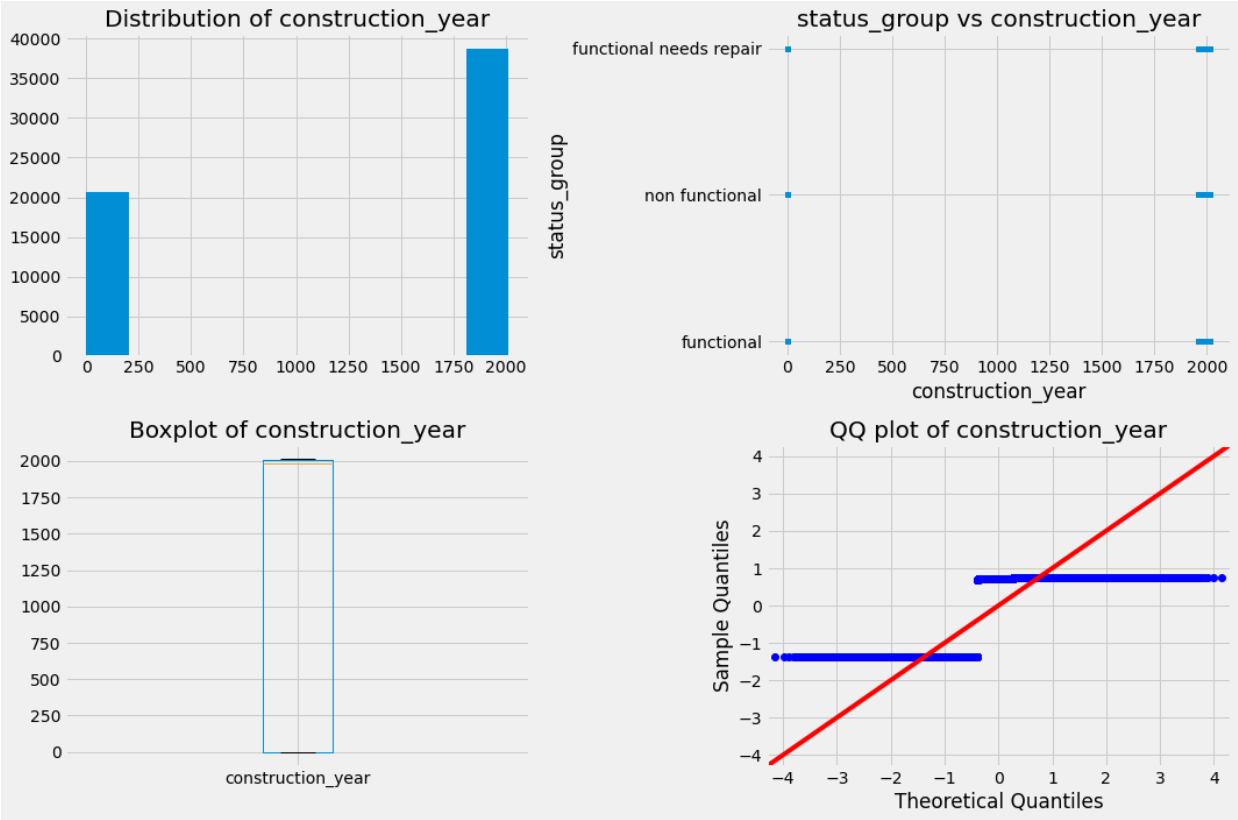
- I will leave the 0's as they are logical values

In [642]:
```
1 #eval construction_year
2 col_eval(df_clean, num_col='construction_year')
```

executed in 619ms, finished 20:49:49 2021-05-23

```
Column Name: construction_year


Number of unique values: 55


There are 59345 duplicates


There are 0 null values


There are 20709 zeros


Value Counts Percentage
 0       35.0
2010     4.0
2008     4.0
2009     4.0
2000     4.0
2007     3.0
2006     2.0
2003     2.0
2011     2.0
2004     2.0
2012     2.0
2002     2.0
1978     2.0
1995     2.0
2005     2.0
1999     2.0
1998     2.0
1990     2.0
1985     2.0
1980     1.0
1996     1.0
1984     1.0
1982     1.0
1994     1.0
1972     1.0
1974     1.0
1997     1.0
1992     1.0
1993     1.0
2001     1.0
1988     1.0
1983     1.0
1975     1.0
1986     1.0
1976     1.0
1970     1.0
1991     1.0
```

```
1989    1.0
1987    1.0
1981    0.0
1977    0.0
1979    0.0
1973    0.0
2013    0.0
1971    0.0
1960    0.0
1967    0.0
1963    0.0
1968    0.0
1969    0.0
1964    0.0
1962    0.0
1961    0.0
1965    0.0
1966    0.0
Name: construction_year, dtype: float64


Descriptive Metrics:
 count              59,400.0
mean    1,300.6524747474748
std       951.6205473151729
min                     0.0
25%                     0.0
50%                 1,986.0
75%                 2,004.0
max                 2,013.0
Name: construction_year, dtype: float64
```



OBSERVATIONS

- `construction_year` has many 0's which have to be placeholders for unknown. Imputation could be done by geographic location or based on most common construction years that are known.

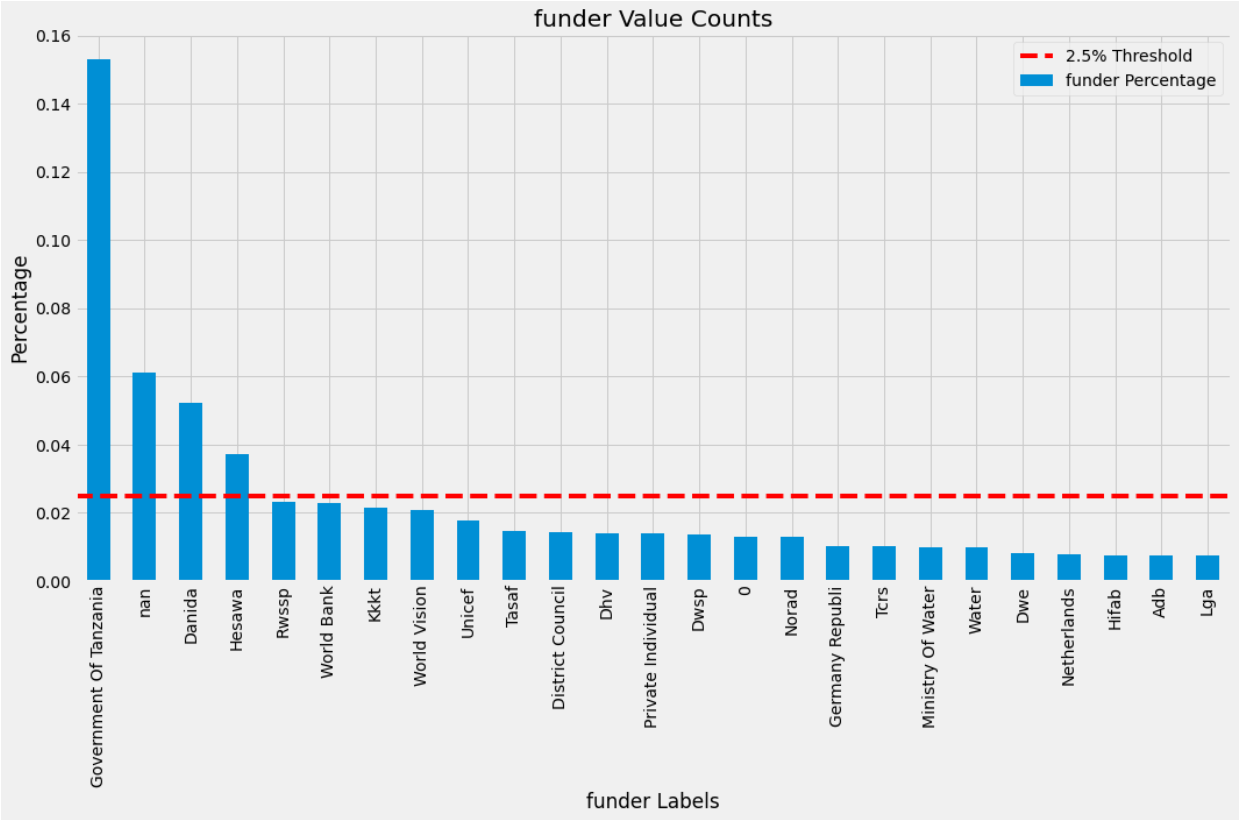
ACTIONS

- Impute `construction_year` to eliminate the 0's

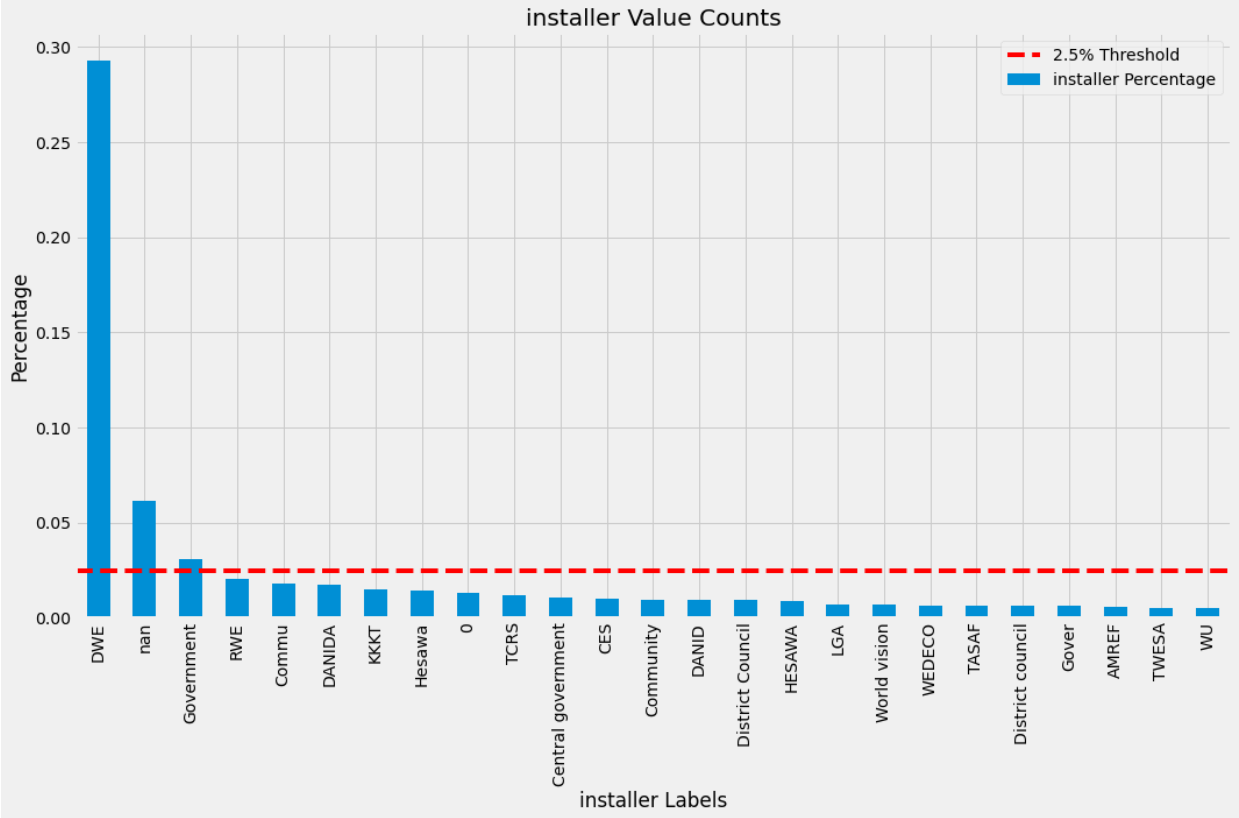
In [643]:
```
#eval funder
col_eval(df_clean, cat_cols=['funder','installer'])
```

executed in 478ms, finished 20:49:50 2021-05-23

```
==============================================================
Column Name: funder


Number of unique values: 1897


There are 57502 duplicates


There are 3635 null values


There are 777 zeros


Value Counts Percentage
 Government Of Tanzania      9084
NaN                          3635
Danida                       3114
Hesawa                       2202
Rwssp                        1374
                             ...
Rc Missi                        1
Makanya Sisal Estate            1
Lusajo                          1
Sister Makulata                 1
Grail Mission Kiseki Bar        1
Name: funder, Length: 1898, dtype: int64
```




```
============================================================
Column Name: installer


Number of unique values: 2145


There are 57254 duplicates


There are 3655 null values


There are 777 zeros


Value Counts Percentage
 DWE             17402
NaN              3655
Government       1825
RWE              1206
Commu            1060
                ...
MSIGWA              1
Livi                1
SIMBA LODGE         1
The Co              1
GERMAN              1
Name: installer, Length: 2146, dtype: int64
```



OBSERVATIONS

- These 2 columns are not similar but it does have a high cardinality (1897 unique labels) and rare labels.

- `funder` has 3,635 missing values, which is 6% of the rows while `installer` has 3,655 missing values, which is 6% of the rows.
- There is also 777 "0" labels for both which could indicate a missing value.

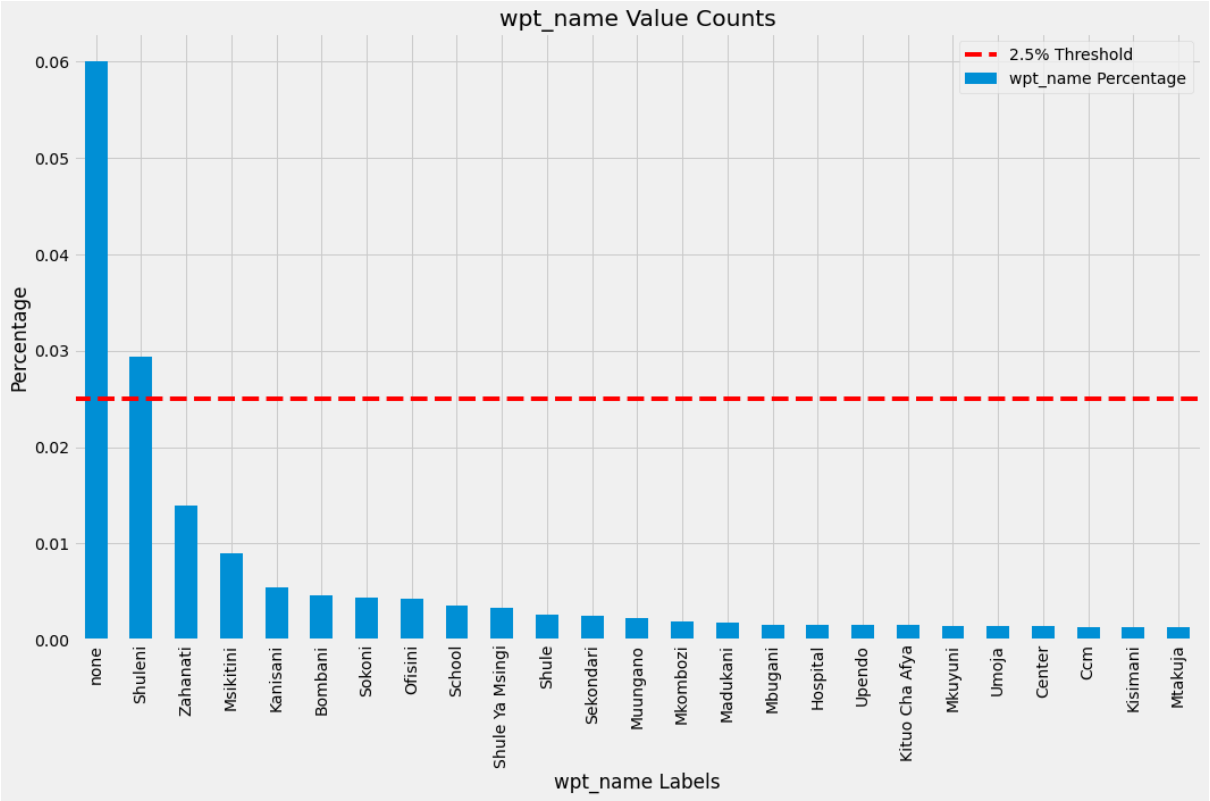
ACTIONS

- I will combine both of the 0's with the "Unknown" label
- I will combine labels in order to eliminate the high cardinality and rare labels.

In [644]:
```
#eval wpt_name
col_eval(df_clean, cat_cols=['wpt_name'])
```

executed in 266ms, finished 20:49:50 2021-05-23

```
============================================================
Column Name: wpt_name


Number of unique values: 37400


There are 22000 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 none                        3563
Shuleni                     1748
Zahanati                     830
Msikitini                    535
Kanisani                     323
                            ...
Chaludewa Primary School       1
Zahanati Ya Ndilimal           1
Monika                         1
Kwa Shundi                     1
Wilson Kalimo                  1
Name: wpt_name, Length: 37400, dtype: int64
```



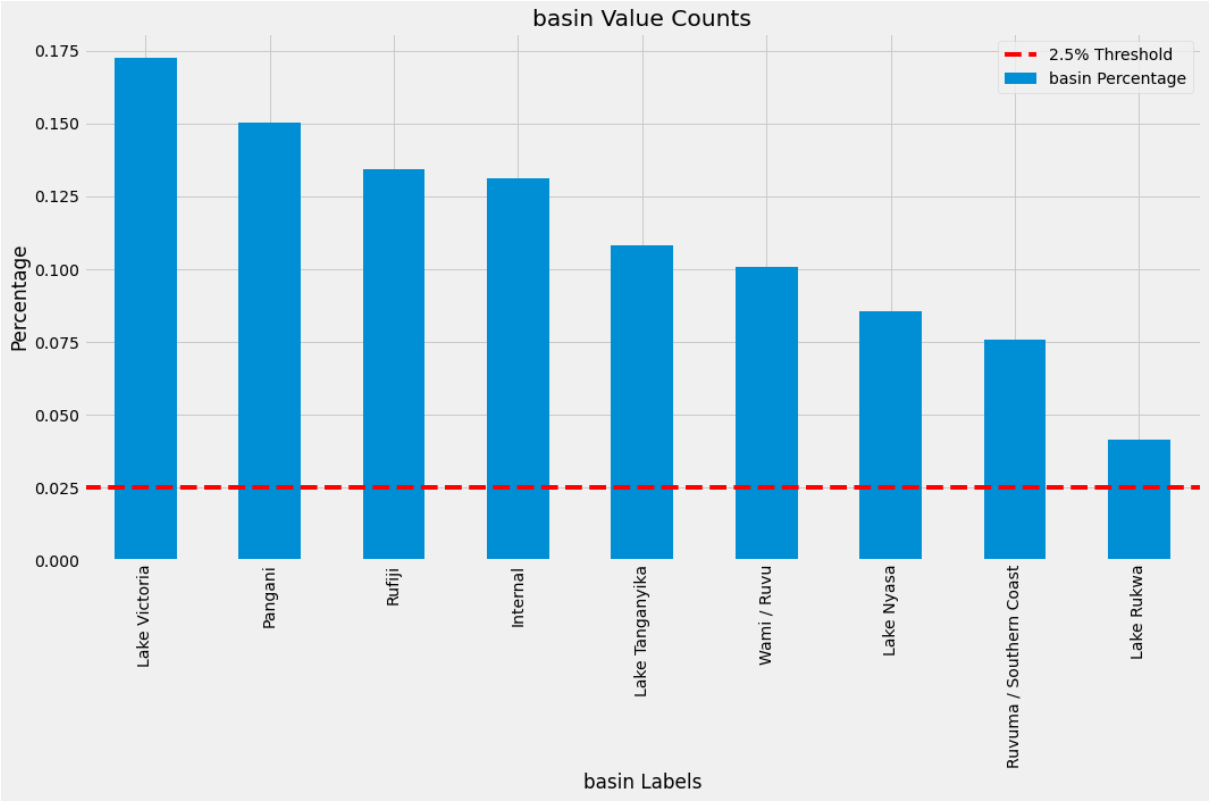
OBSERVATIONS

- No issues with `wpt_name` .

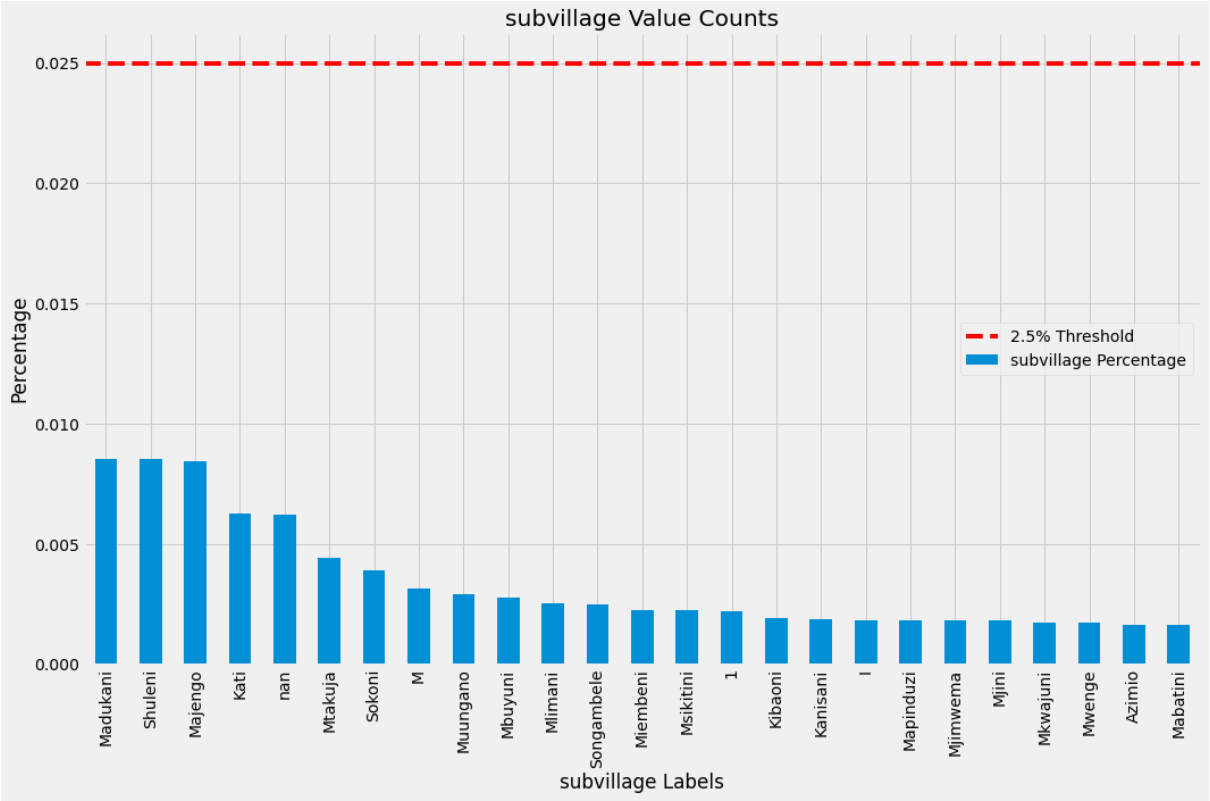
In [645]:
```
#eval geographic features
col_eval(df_clean, cat_cols=['basin','subvillage','region','region_code
```

executed in 1.50s, finished 20:49:51 2021-05-23

```
============================================================
Column Name: basin


Number of unique values: 9


There are 59391 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 Lake Victoria              10248
Pangani                     8940
Rufiji                      7976
Internal                    7785
Lake Tanganyika             6432
Wami / Ruvu                 5987
Lake Nyasa                  5085
Ruvuma / Southern Coast     4493
Lake Rukwa                  2454
Name: basin, dtype: int64
```

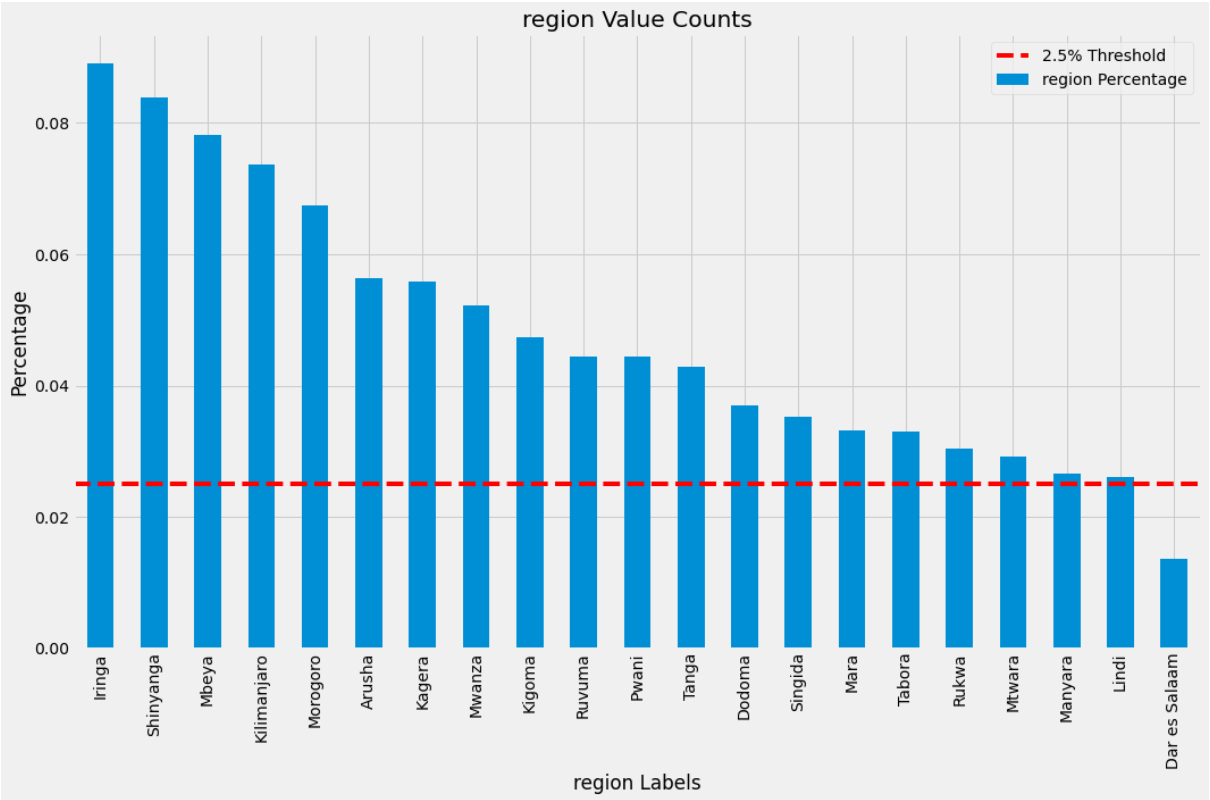



```
============================================================
Column Name: subvillage


Number of unique values: 19287


There are 40112 duplicates


There are 371 null values


There are 0 zeros


Value Counts Percentage
 Madukani        508
Shuleni          506
Majengo          502
Kati             373
NaN              371
                ...
Semaki A           1
Namienje           1
Sangu Jineli       1
Iphanda            1
Igumbiro           1
Name: subvillage, Length: 19288, dtype: int64
```

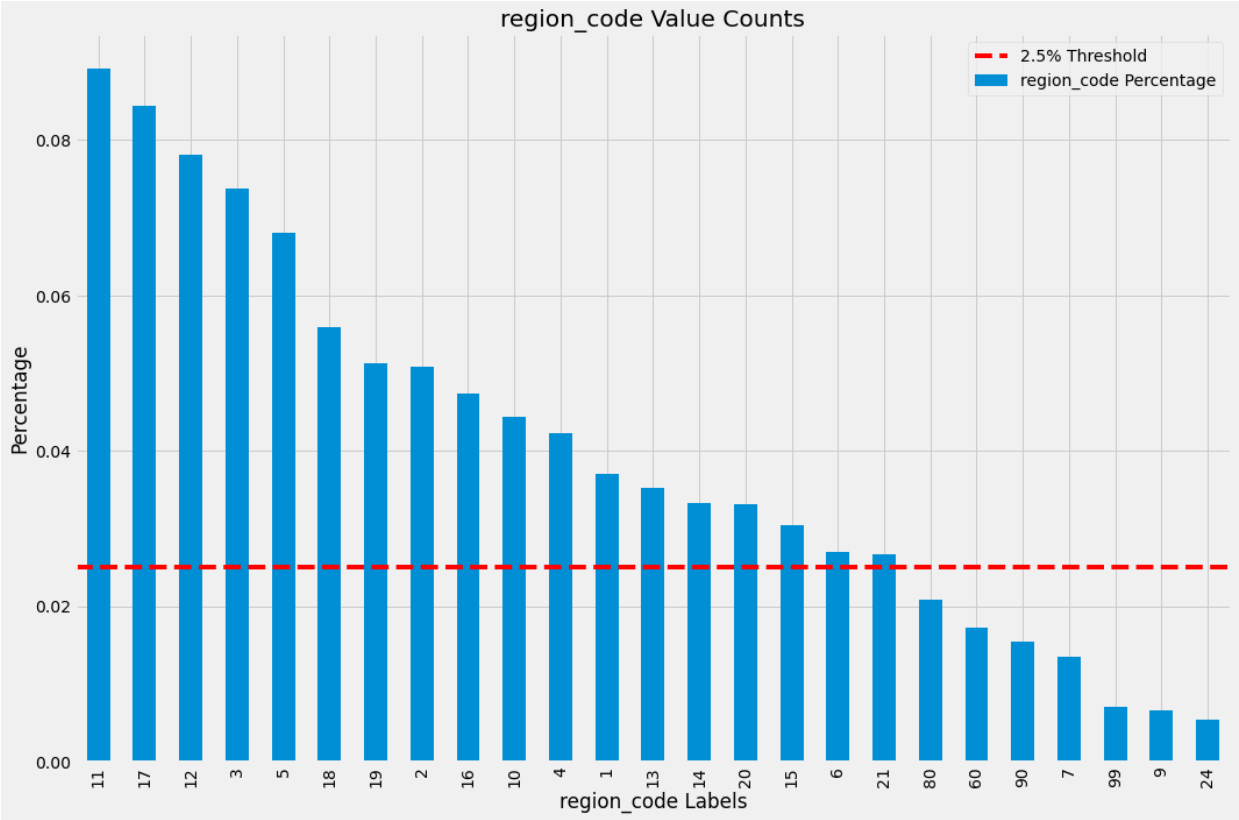



```
============================================================
Column Name: region

Number of unique values: 21

There are 59379 duplicates

There are 0 null values
```

```
There are 0 zeros


Value Counts Percentage
 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
```



```
============================================================
Column Name: region_code


Number of unique values: 27


There are 59373 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 11    5300
17    5011
12    4639
3     4379
5     4040
18    3324
19    3047
2     3024
16    2816
10    2640
4     2513
1     2201
13    2093
14    1979
20    1969
15    1808
6     1609
21    1583
80    1238
60    1025
90     917
7      805
99     423
9      390
24     326
8      300
40       1
Name: region_code, dtype: int64
```



==========

Column Name: district_code

Number of unique values: 20

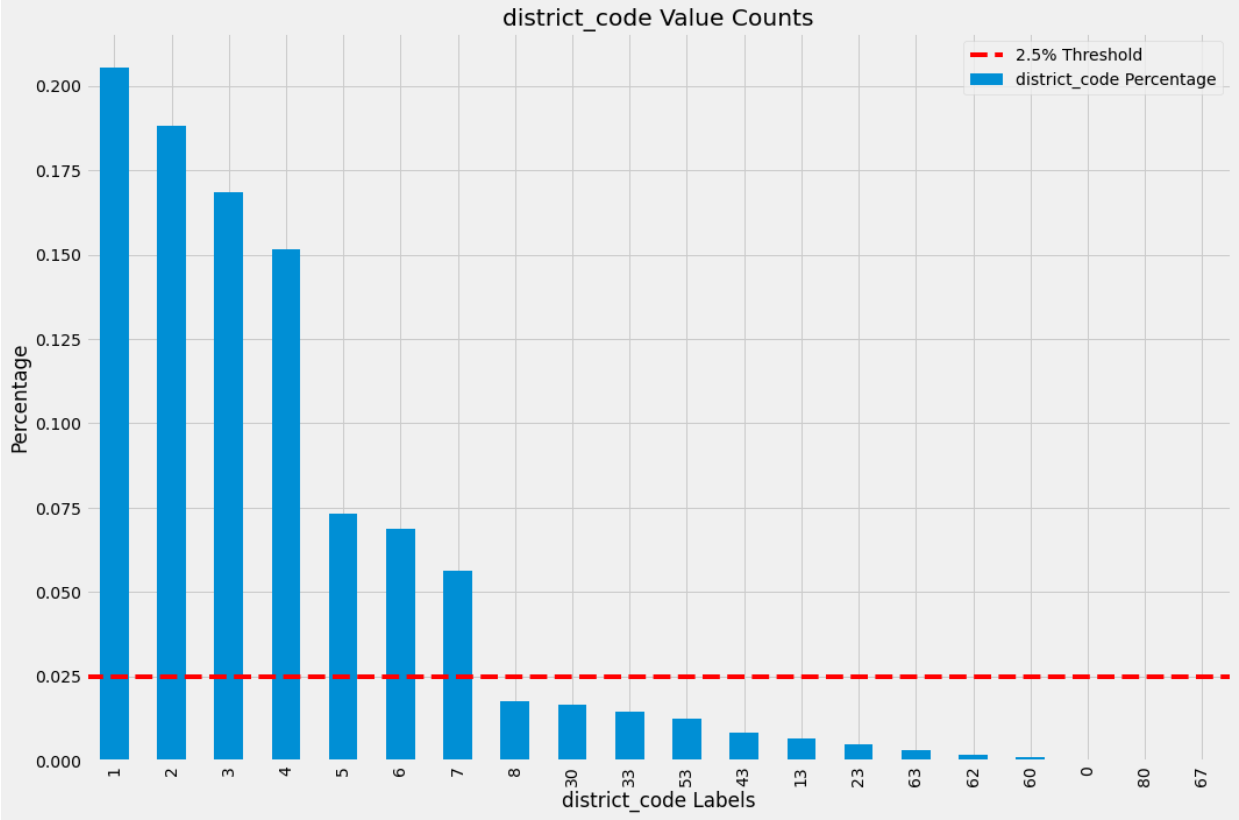
There are 59380 duplicates

There are 0 null values

There are 0 zeros

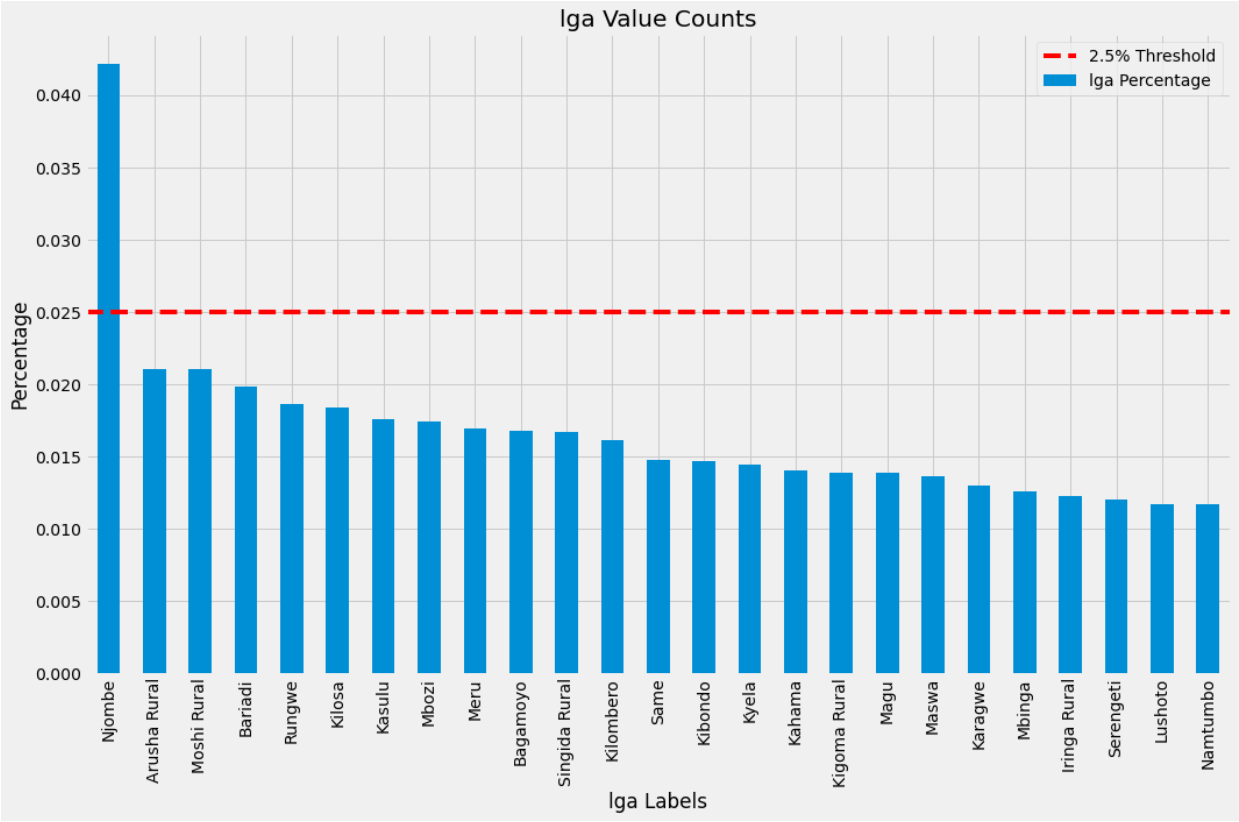
Value	Counts	Percentage
1	12203	
2	11173	
3	9998	
4	8999	
5	4356	
6	4074	
7	3343	
8	1043	
30	995	
33	874	
53	745	
43	505	
13	391	
23	293	
63	195	
62	109	

```
60        63
0         23
80        12
67         6
Name: district_code, dtype: int64
```



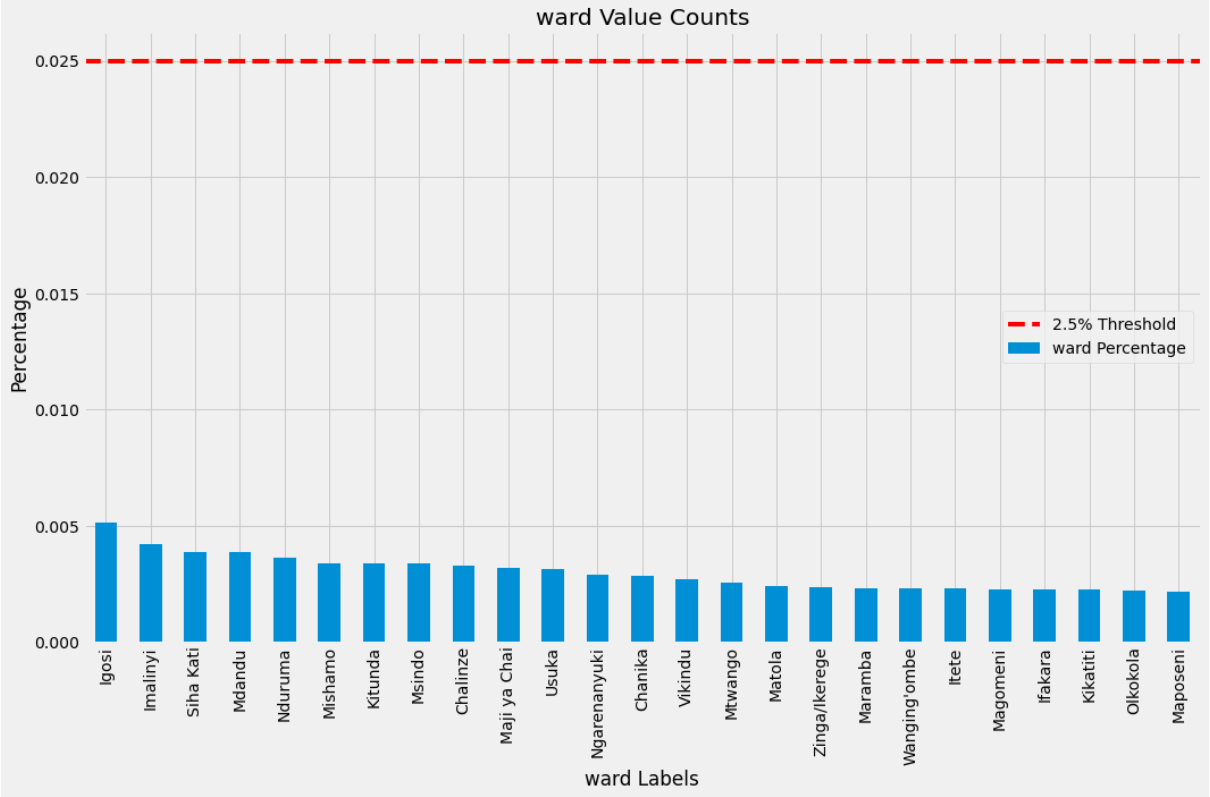

```
============================================================
Column Name: lga


Number of unique values: 125


There are 59275 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 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
```




```
============================================================
Column Name: ward


Number of unique values: 2092


There are 57308 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 Igosi           307
Imalinyi         252
Siha Kati        232
Mdandu           231
Nduruma          217
                ...
Mkumbi             1
Machinjioni        1
```

```
Ukata            1
Kapilula         1
Themi            1
Name: ward, Length: 2092, dtype: int64
```



OBSERVATIONS

- `basin` seems normal
- `subvillage` has 371 null values and high cardinality (19,286 unique values)
- `region` seems normal

- `region_code` is redundant
- `district_code` seems normal
- `lga` seems normal
- `ward` has high cardinality (2,092 unique values)

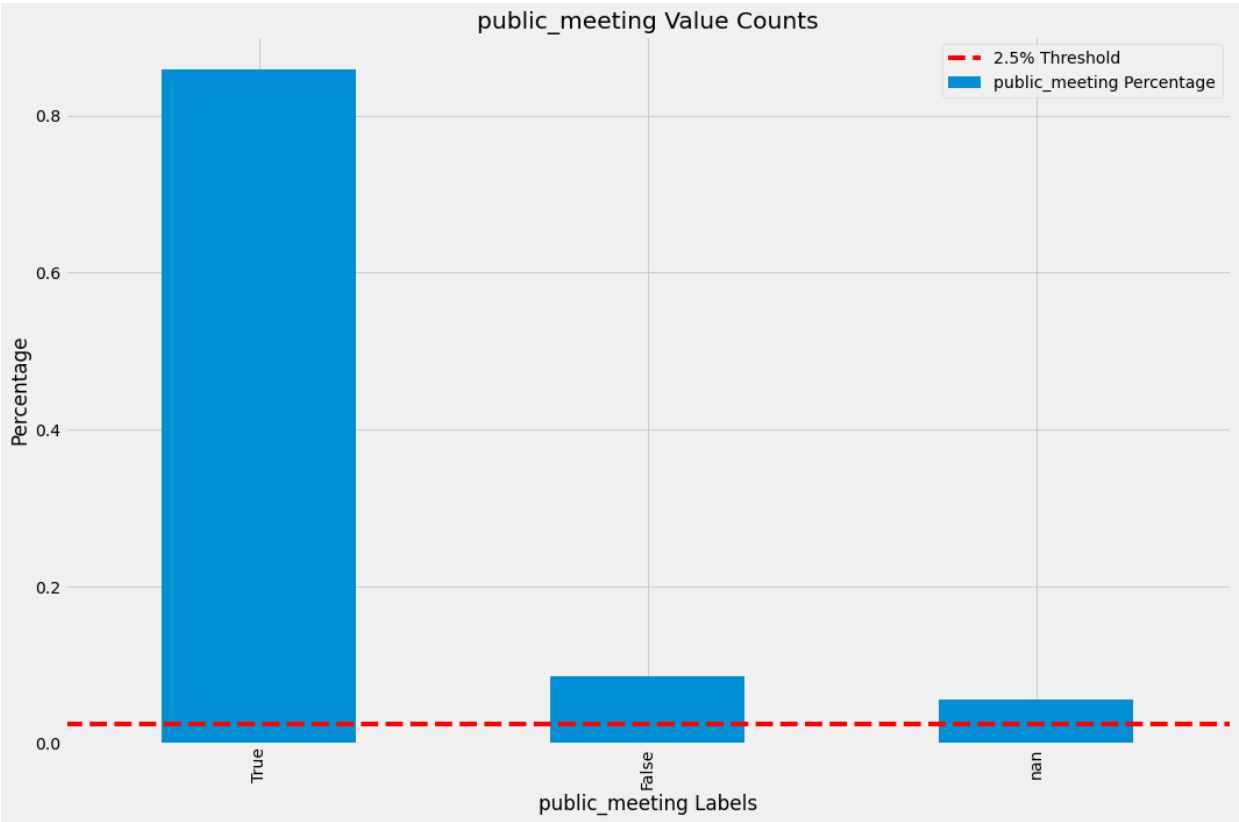
ACTIONS

- drop `subvillage` since it has too many unique values
- drop `region_code`

In [646]:
```
1 #eval public_meeting
2 col_eval(df_clean, cat_cols=['public_meeting'])
```

executed in 133ms, finished 20:49:52 2021-05-23

```
================================================================
Column Name: public_meeting


Number of unique values: 2


There are 59397 duplicates


There are 3334 null values


There are 0 zeros


Value Counts Percentage
 True     51011
False     5055
NaN       3334
Name: public_meeting, dtype: int64
```



OBSERVATIONS

- `public_meeting` looks to be a boolean feature
- there are 3,333 missing values

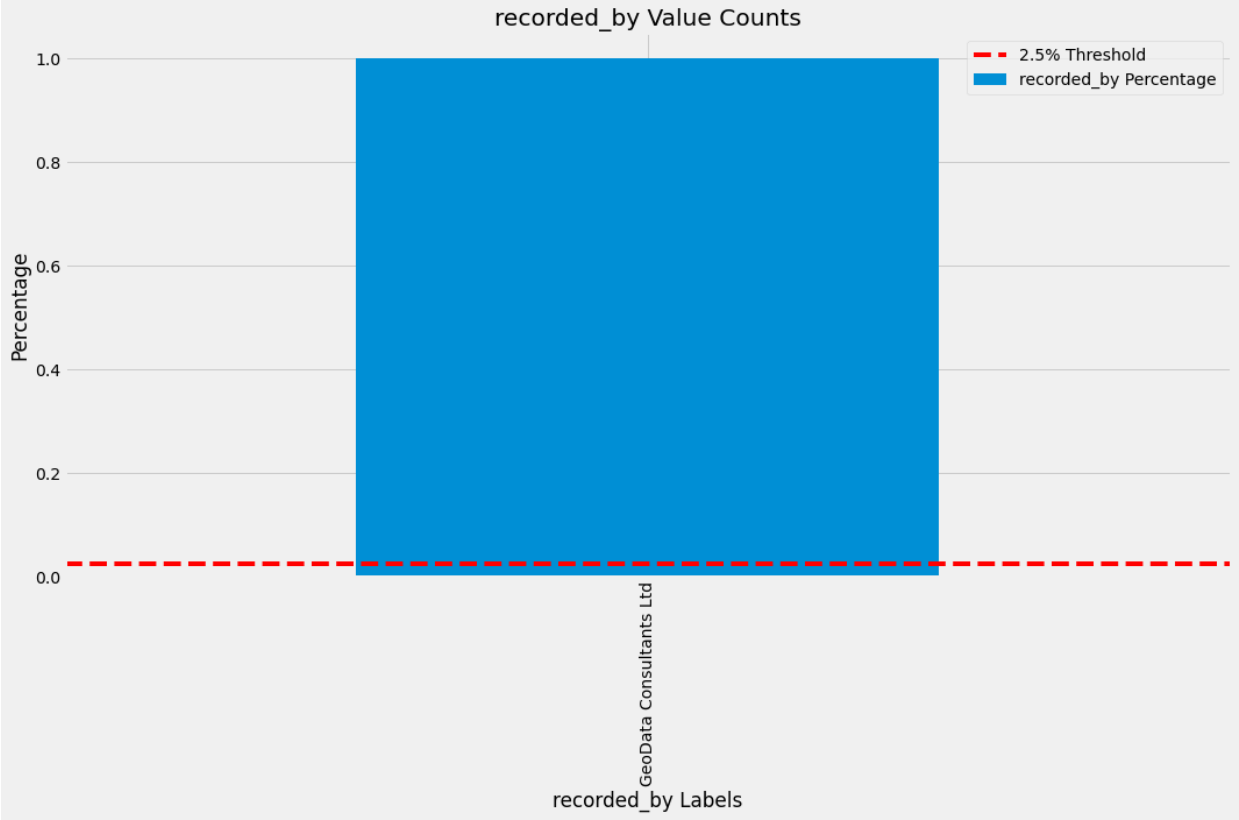
ACTIONS

- convert to string
- impute missing values as "unknown"

In [647]:
```
#eval recorded_by
col_eval(df_clean, cat_cols=['recorded_by'])
```

executed in 137ms, finished 20:49:52 2021-05-23

```
============================================================
Column Name: recorded_by


Number of unique values: 1


There are 59399 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 GeoData Consultants Ltd    59400
Name: recorded_by, dtype: int64
```



OBSERVATIONS

- `recorded_by` has a single value and will not be beneficial for EDA or modeling

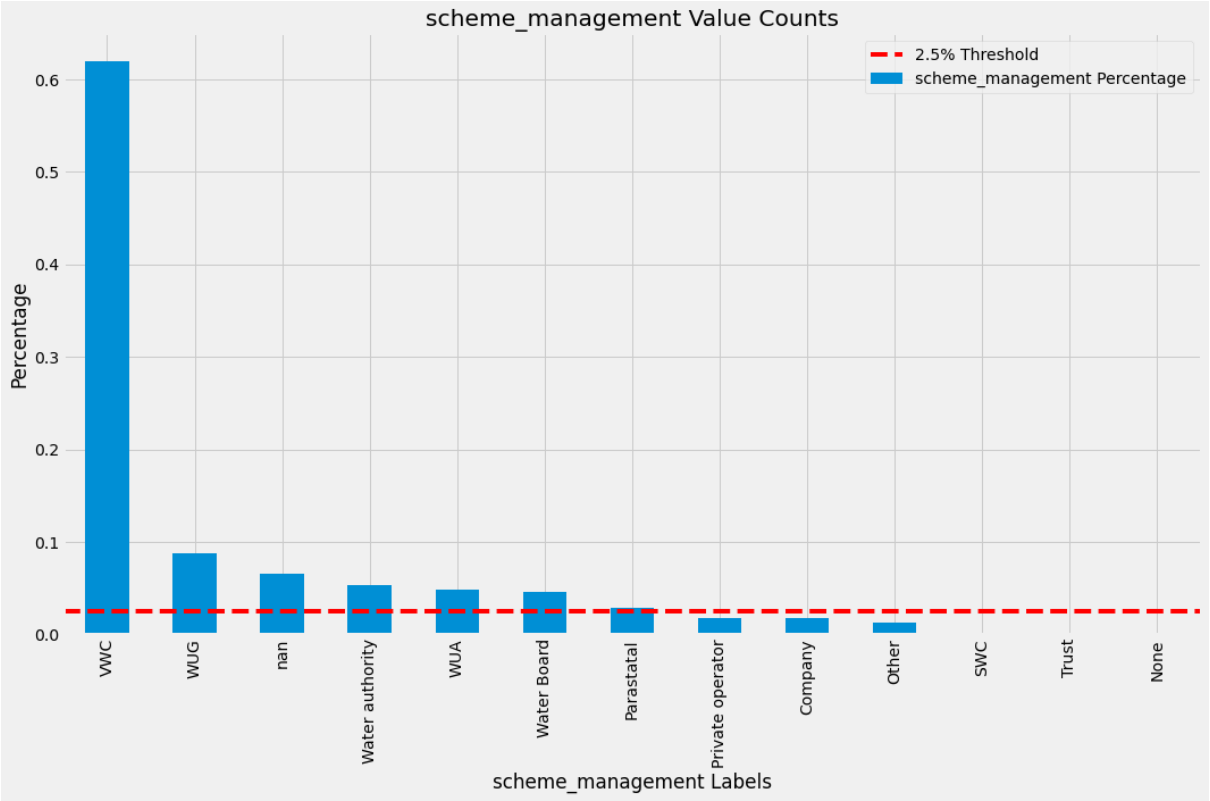
ACTIONS

- drop the feature

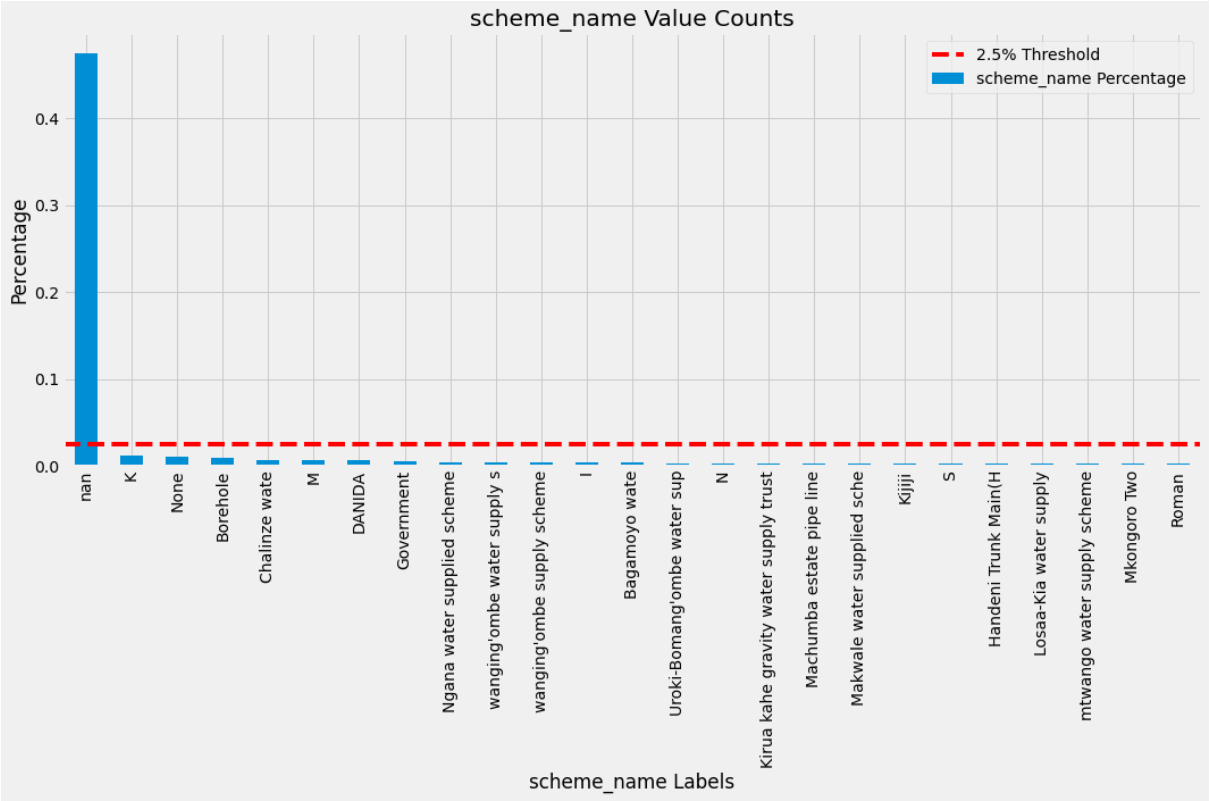
In [648]:
```
#eval scheme_name, scheme_management
col_eval(df_clean, cat_cols=['scheme_management','scheme_name'])
```

executed in 430ms, finished 20:49:52 2021-05-23

```
==============================================================
Column Name: scheme_management


Number of unique values: 12


There are 59387 duplicates


There are 3877 null values


There are 0 zeros


Value Counts Percentage
 VWC                 36793
WUG                  5206
NaN                  3877
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
```




```
============================================================
Column Name: scheme_name


Number of unique values: 2696


There are 56703 duplicates


There are 28166 null values


There are 0 zeros


Value Counts Percentage
 NaN                              28166
K                                  682
None                               644
Borehole                           546
Chalinze wate                      405
                                  ...
Tove - Mtwango                       1
NYA/ MAK/ BUK piped  scheme          1
lgongolo gravity water sche          1
Tanload                              1
Mfinga Water Supply                  1
Name: scheme_name, Length: 2697, dtype: int64
```



OBSERVATIONS

- `scheme_management` has 3,877 missing values
- `scheme_name` has 28,160 missing values

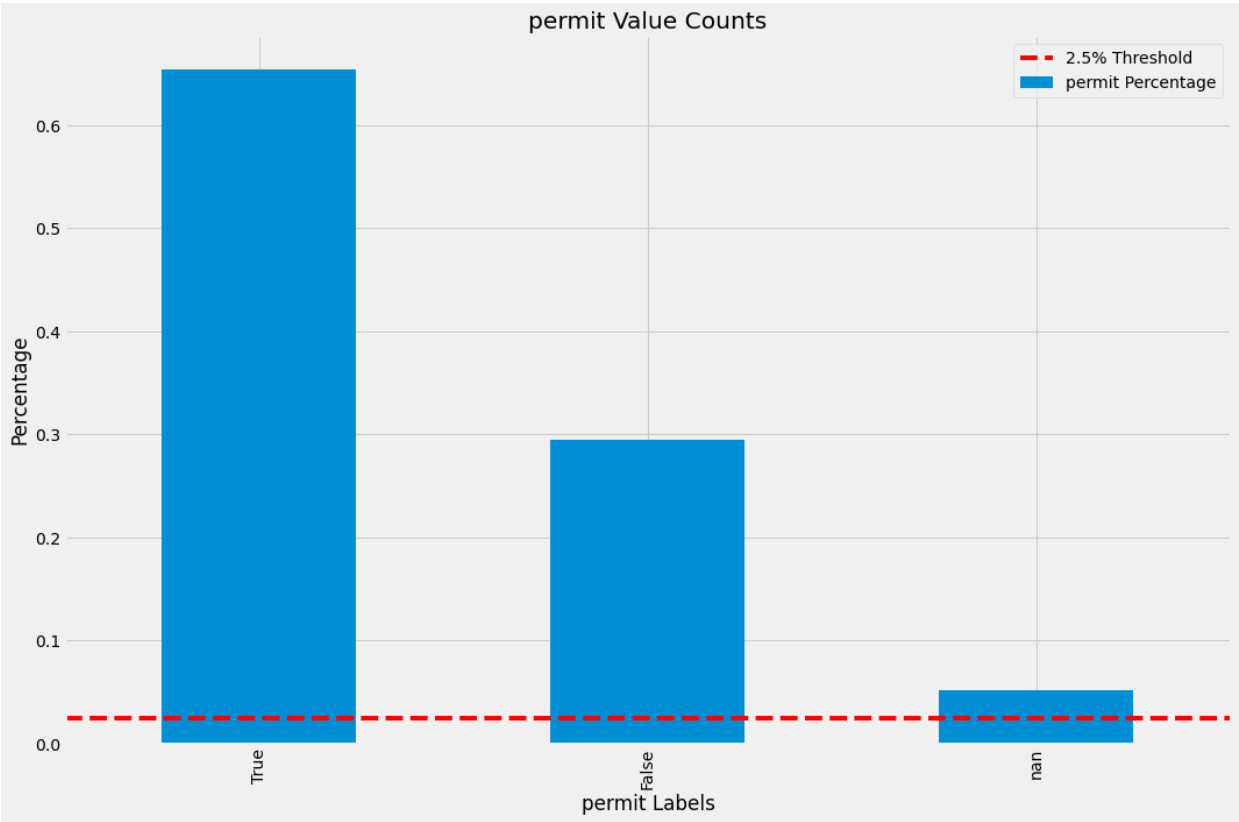
ACTIONS

- drop `scheme_name`
- consolidate missing values in `scheme_management` with "unknown"

In [649]:
```
#eval permit
col_eval(df_clean, cat_cols=['permit'])
```

executed in 142ms, finished 20:49:52 2021-05-23

```
================================================================
Column Name: permit


Number of unique values: 2


There are 59397 duplicates


There are 3056 null values


There are 0 zeros


Value Counts Percentage
 True     38852
False    17492
NaN       3056
Name: permit, dtype: int64
```



OBSERVATIONS

- `permit` looks like a boolean feature
- There are 3056 original missing values which were converted to string 'nan', which is 5% of the rows.

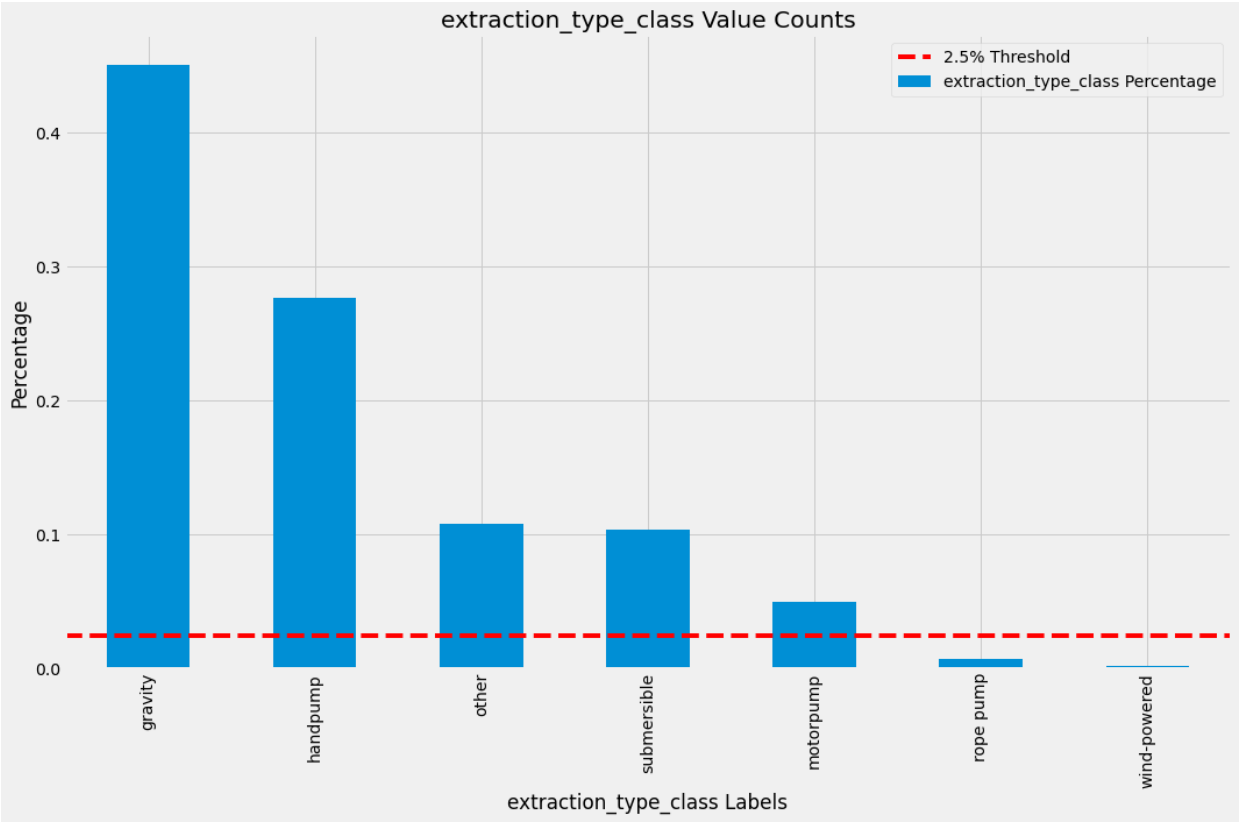
ACTIONS

- I will rename these "unknown"
- I will also recast as a string

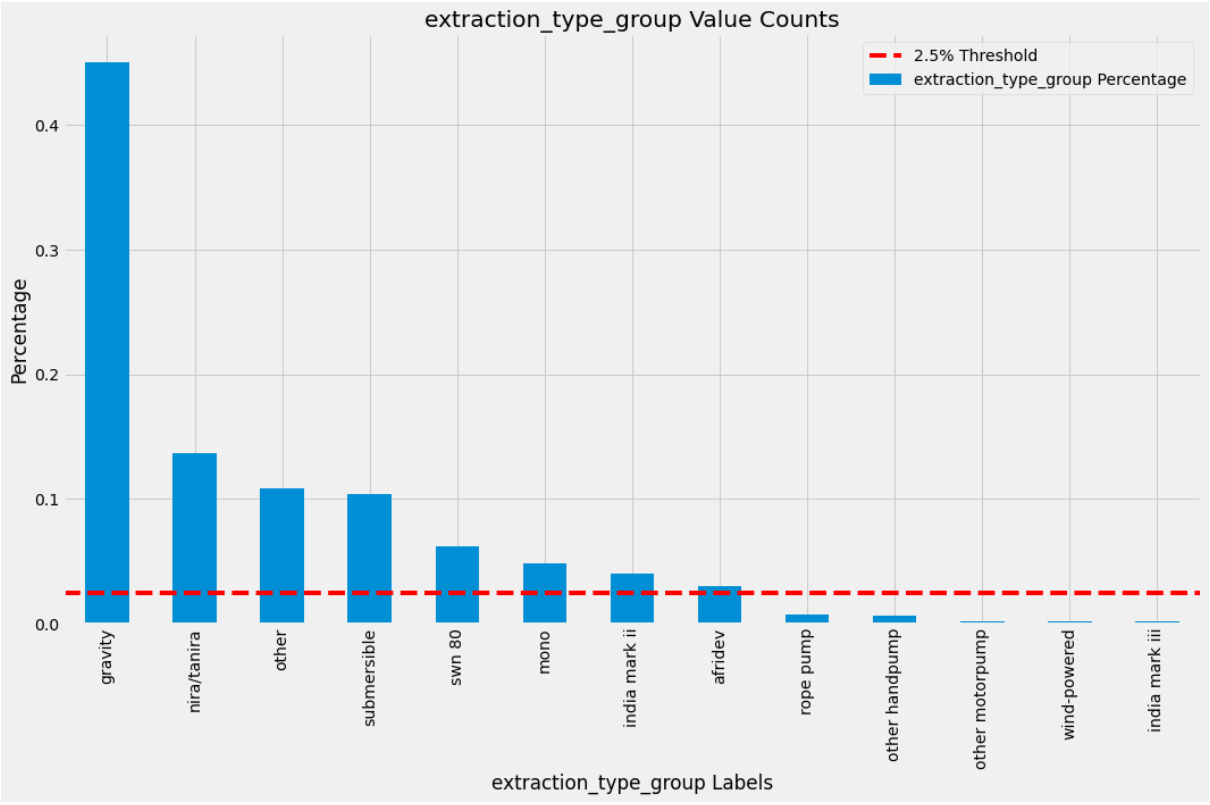
In [650]:
```
#eval extraction_type, extraction_type_group, extraction_type_class
col_eval(df_clean, cat_cols=['extraction_type_class','extraction_type_g
```

executed in 541ms, finished 20:49:53 2021-05-23

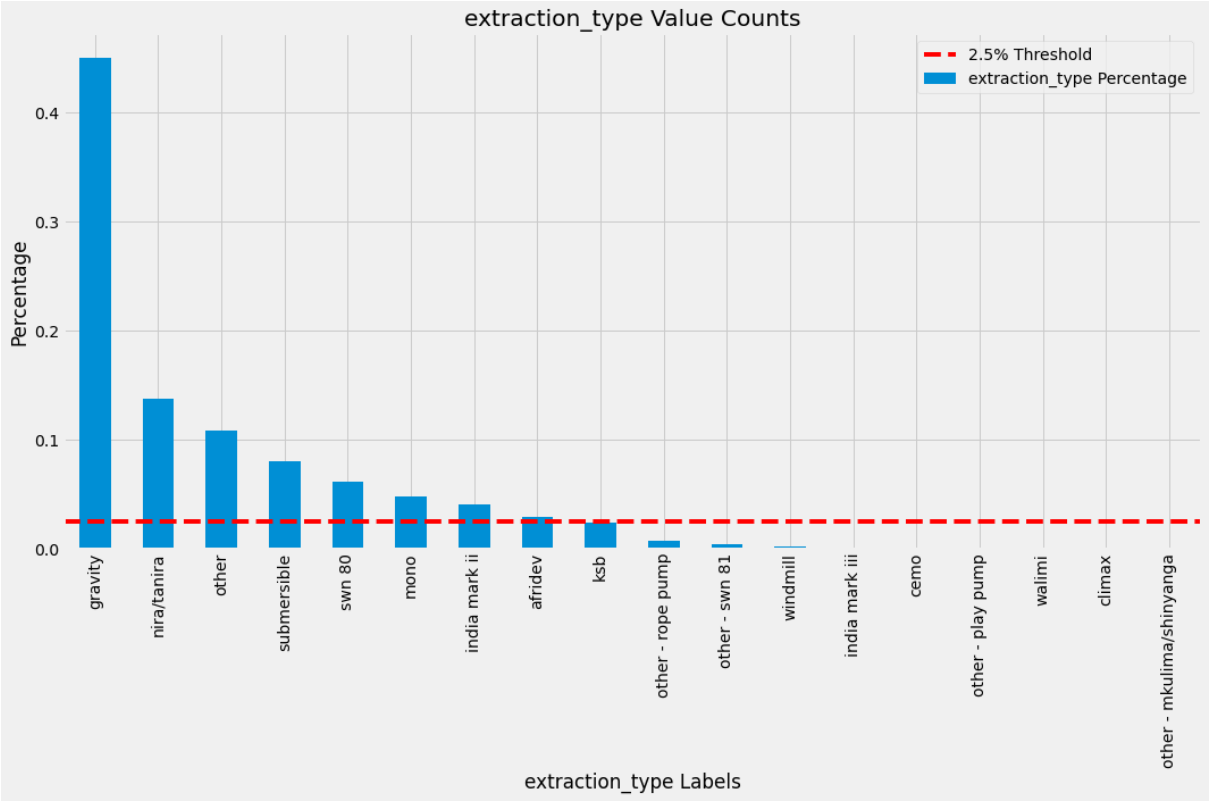
```
============================================================
Column Name: extraction_type_class


Number of unique values: 7


There are 59393 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 gravity          26780
handpump         16456
other             6430
submersible       6179
motorpump         2987
rope pump          451
wind-powered       117
Name: extraction_type_class, dtype: int64
```



```
============================================================
Column Name: extraction_type_group


Number of unique values: 13


There are 59387 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 gravity             26780
nira/tanira          8154
other                6430
submersible          6179
swn 80               3670
mono                 2865
india mark ii        2400
afridev              1770
rope pump             451
other handpump        364
other motorpump       122
wind-powered          117
india mark iii         98
Name: extraction_type_group, dtype: int64
```



```
============================================================
Column Name: extraction_type


Number of unique values: 18


There are 59382 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 gravity                      26780
nira/tanira                   8154
other                         6430
submersible                   4764
swn 80                        3670
mono                          2865
india mark ii                 2400
afridev                       1770
ksb                           1415
other - rope pump              451
other - swn 81                 229
windmill                       117
india mark iii                  98
cemo                            90
other - play pump               85
walimi                          48
climax                          32
other - mkulima/shinyanga        2
Name: extraction_type, dtype: int64
```


extraction_type Value Counts
2.5% Threshold
extraction_type Percentage
Percentage
0.0
0.1
0.2
0.3
0.4
gravity
nira/tanira
other
submersible
swn 80
mono
india mark ii
afridev
ksb
other - rope pump
other - swn 81
windmill
india mark iii
cemo
other - play pump
walimi
climax
other - mkulima/shinyanga
extraction_type Labels

In [651]:
```
#eval extraction_type_class, extraction_type_group and extraction_type
df_clean.groupby(by=['extraction_type_class','extraction_type_group','e
```

executed in 102ms, finished 20:49:53 2021-05-23

Out[651]:

			id	amount_tsh	date_recorded
extraction_type_class	**extraction_type_group**	**extraction_type**			
gravity	**gravity**	**gravity**	26780	26780	26780
handpump	**afridev**	**afridev**	1770	1770	1770
	india mark ii	**india mark ii**	2400	2400	2400
	india mark iii	**india mark iii**	98	98	98
	nira/tanira	**nira/tanira**	8154	8154	8154
	other handpump	**other - mkulima/shinyanga**	2	2	2
		other - play pump	85	85	85
		other - swn 81	229	229	229
		walimi	48	48	48
	swn 80	**swn 80**	3670	3670	3670
motorpump	**mono**	**mono**	2865	2865	2865
	other motorpump	**cemo**	90	90	90
		climax	32	32	32
other	**other**	**other**	6430	6430	6430
rope pump	**rope pump**	**other - rope pump**	451	451	451
submersible	**submersible**	**ksb**	1415	1415	1415
		submersible	4764	4764	4764
wind-powered	**wind-powered**	**windmill**	117	117	117

OBSERVATIONS

- These columns are similar as they represent a hierarchical structure.

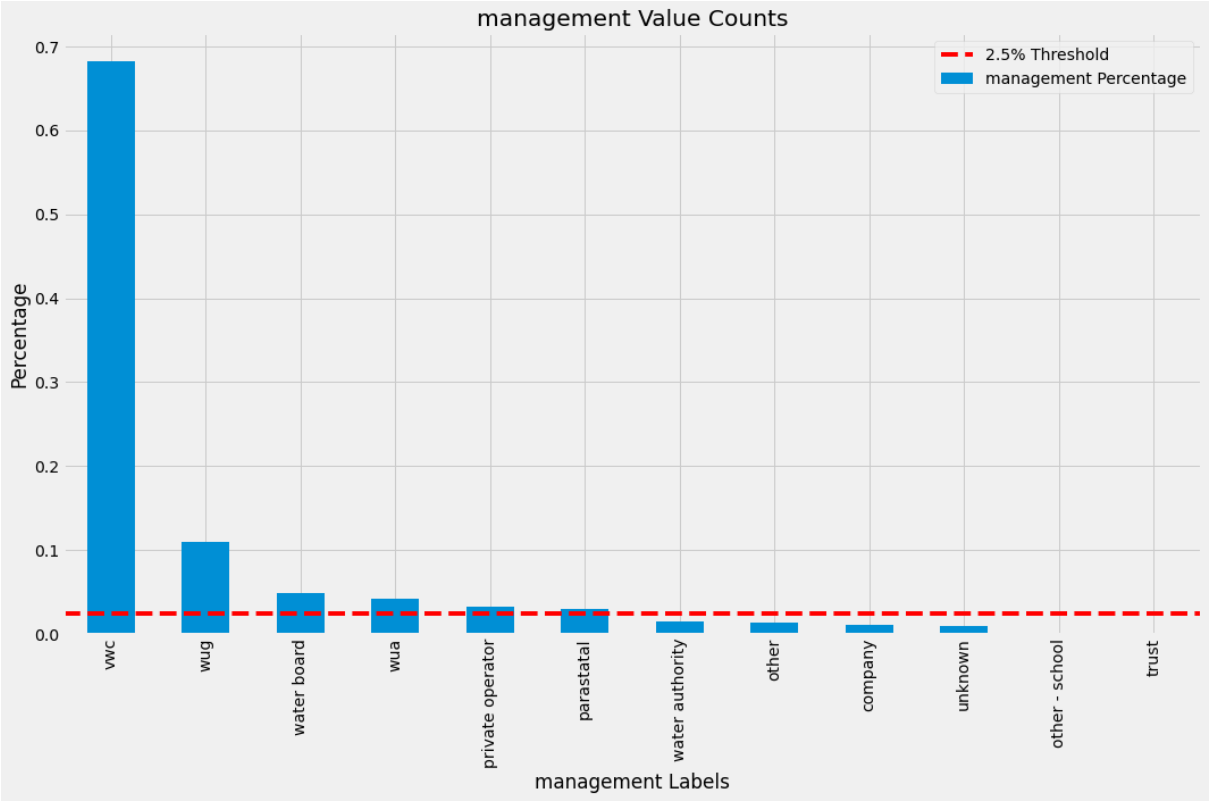
ACTIONS

- I will consolidate by removing the `extraction_type_group` feature as the labels are captured in more detail in the `extraction_type` feature. I will rename the columns and will also clean up the naming of the labels.

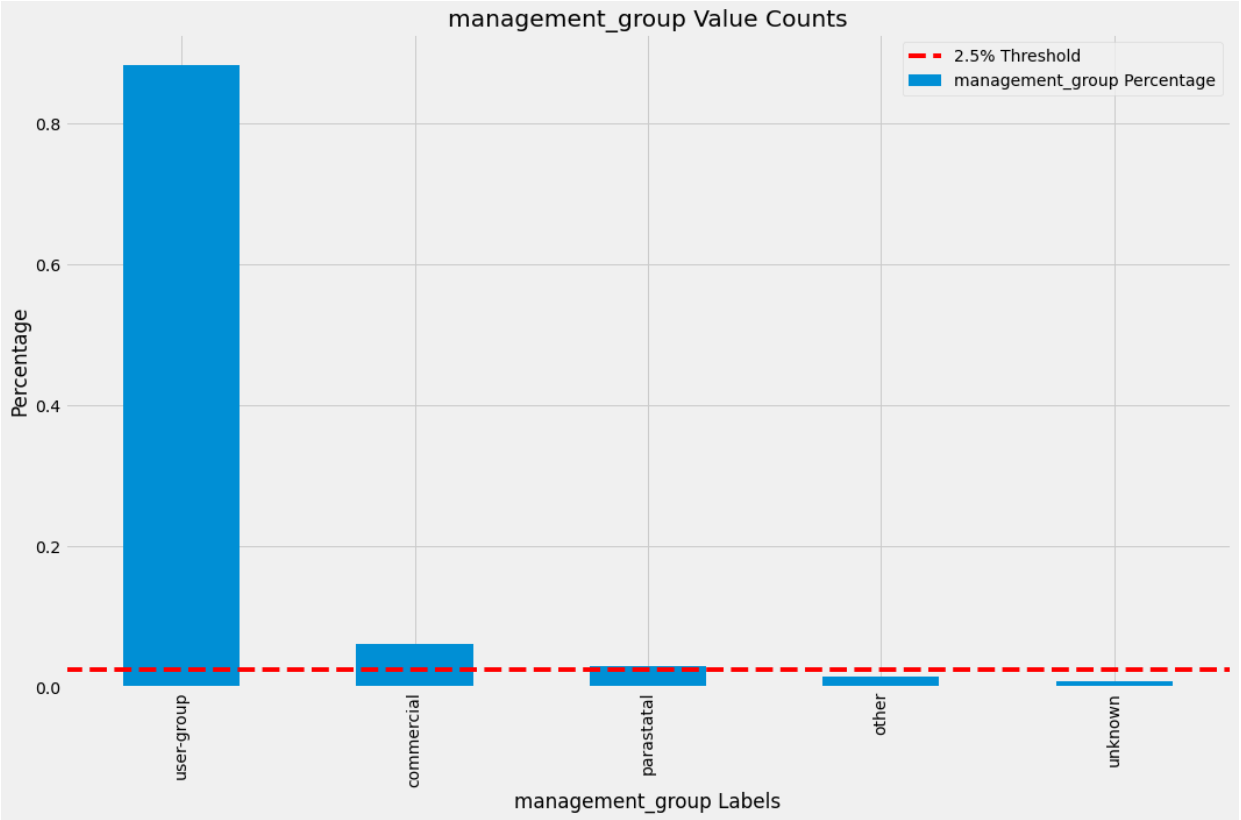
In [652]:
```
#evaluate management and management_group for dropping
col_eval(df_clean, cat_cols=['management','management_group'])
```

executed in 340ms, finished 20:49:53 2021-05-23

```
============================================================
Column Name: management


Number of unique values: 12


There are 59388 duplicates


There are 0 null values


There are 0 zeros


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




```
============================================================
Column Name: management_group


Number of unique values: 5


There are 59395 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 user-group    52490
commercial     3638
parastatal     1768
other           943
unknown         561
Name: management_group, dtype: int64
```



In [653]:
```
#evaluate management and management_group for dropping
df_clean.groupby(by=['management','management_group']).count().index
```

executed in 84ms, finished 20:49:53 2021-05-23

Out[653]:
```
MultiIndex([(         'company', 'commercial'),
            (           'other',      'other'),
            (  'other - school',      'other'),
            (      'parastatal', 'parastatal'),
            ('private operator', 'commercial'),
            (           'trust', 'commercial'),
            (         'unknown',    'unknown'),
            (             'vwc', 'user-group'),
            ( 'water authority', 'commercial'),
            (     'water board', 'user-group'),
            (             'wua', 'user-group'),
            (             'wug', 'user-group')],
           names=['management', 'management_group'])
```

OBSERVATIONS

- `management_group` is redundant

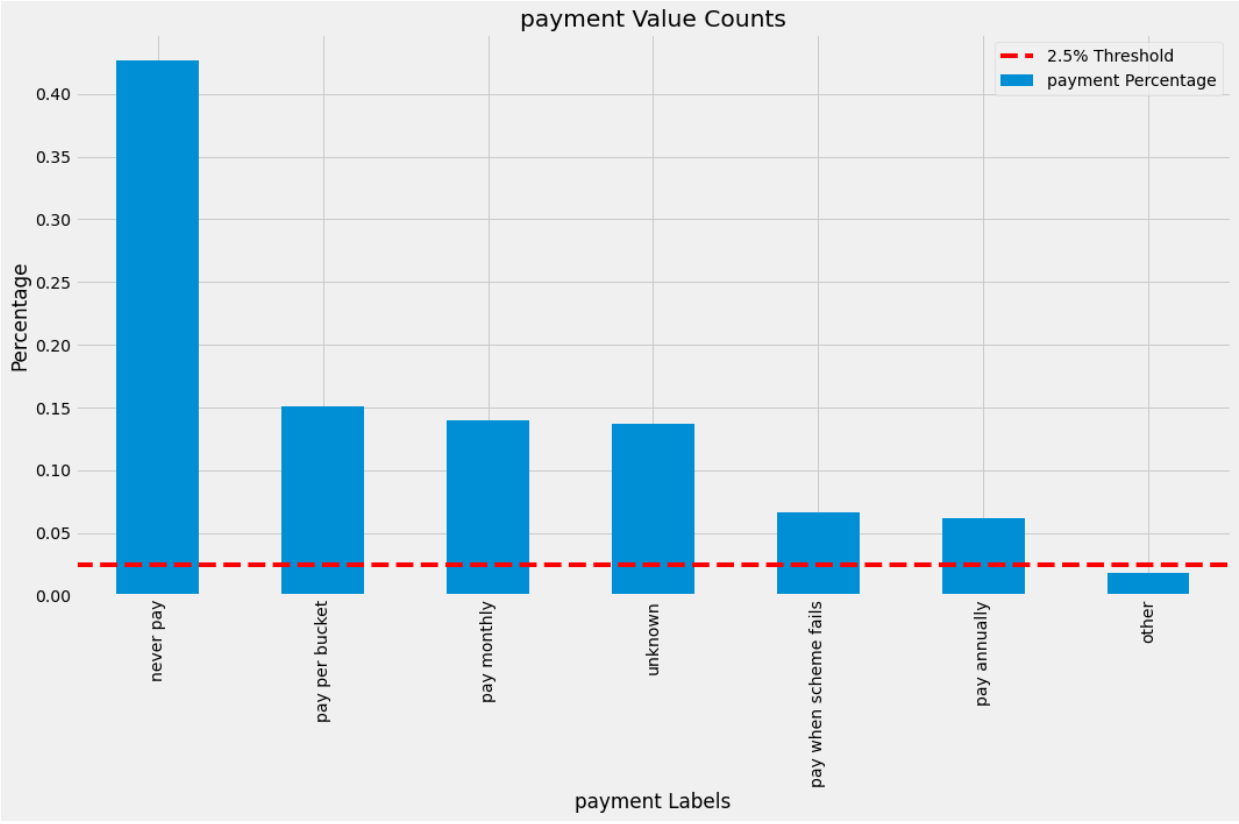
ACTIONS

- drop `management_group`

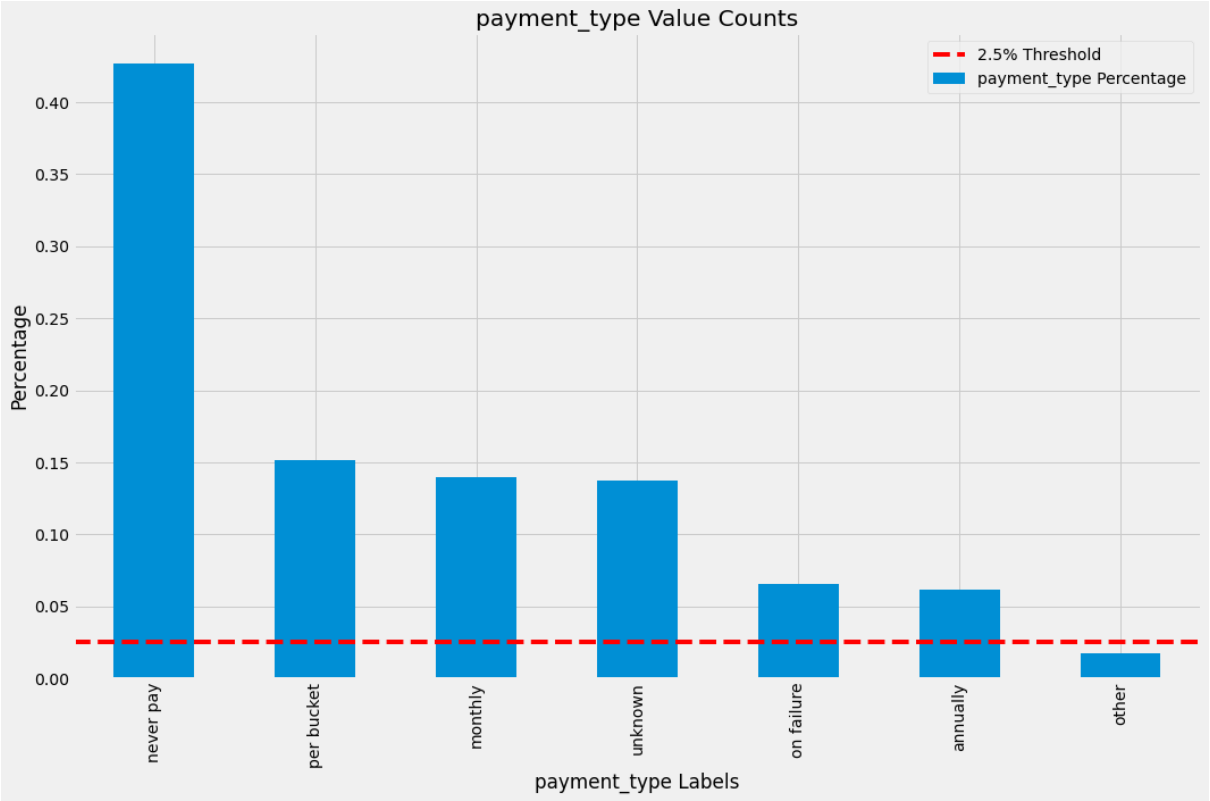
In [654]:
```
#evaluate payment, payment_type
col_eval(df_clean, cat_cols=['payment','payment_type'])
```

executed in 332ms, finished 20:49:54 2021-05-23

```
============================================================
Column Name: payment


Number of unique values: 7


There are 59393 duplicates


There are 0 null values


There are 0 zeros


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



```
============================================================
Column Name: payment_type


Number of unique values: 7


There are 59393 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 never pay     25348
per bucket     8985
monthly        8300
unknown        8157
on failure     3914
annually       3642
other          1054
Name: payment_type, dtype: int64
```



In [655]:
```
#evaluate payment and payment_type
df_clean.groupby(by=['payment','payment_type']).count().index
```

executed in 82ms, finished 20:49:54 2021-05-23

Out[655]:
```
MultiIndex([(            'never pay',  'never pay'),
            (                'other',      'other'),
            (          'pay annually',   'annually'),
            (           'pay monthly',    'monthly'),
            (        'pay per bucket', 'per bucket'),
            ('pay when scheme fails', 'on failure'),
            (              'unknown',    'unknown')],
           names=['payment', 'payment_type'])
```

OBSERVATIONS

- `payment_type` is redundant

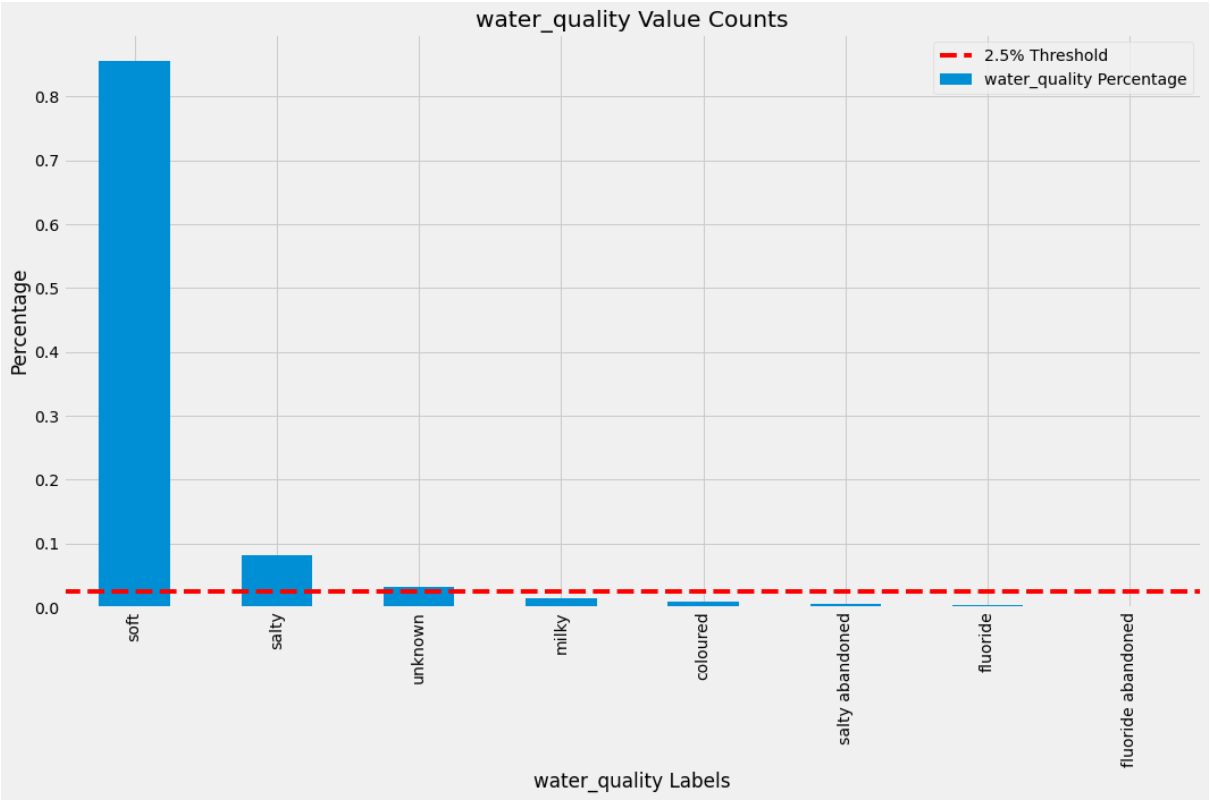
ACTIONS

- drop `payment_type`

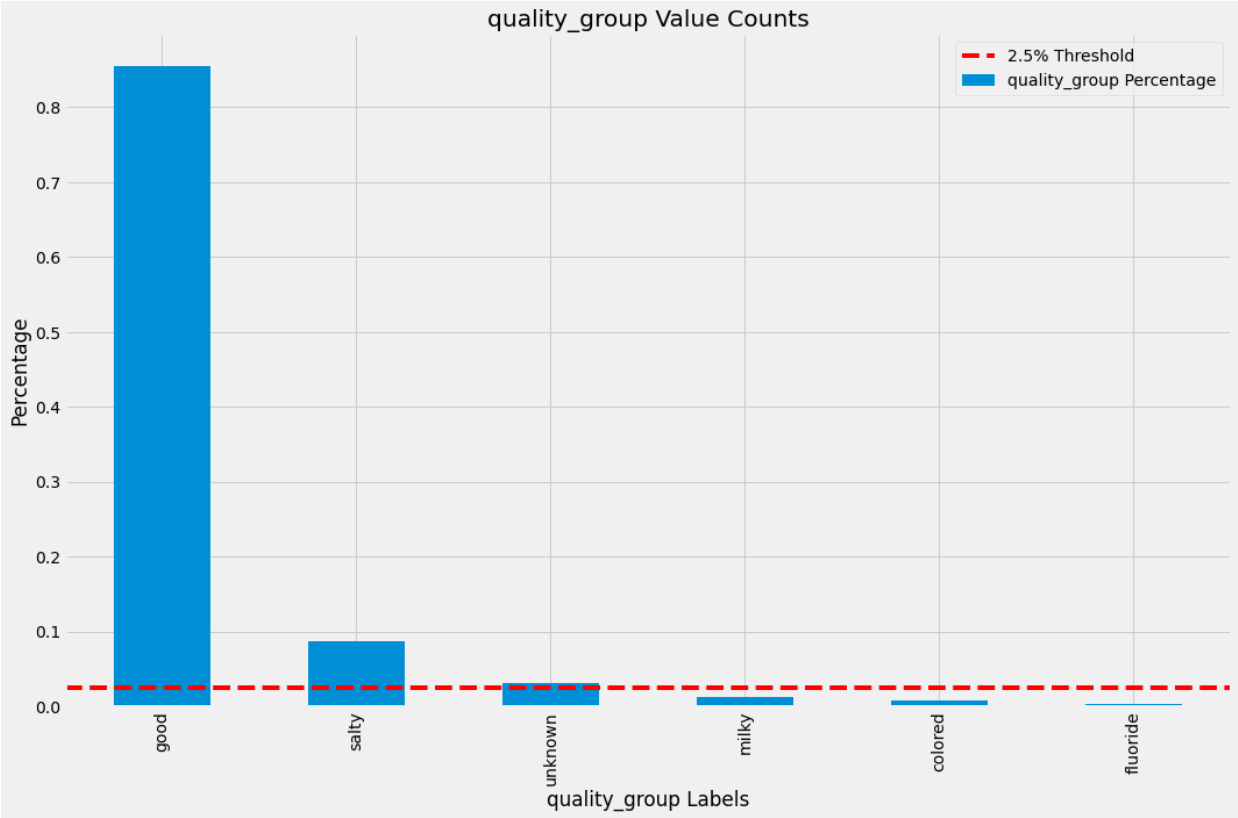
In [656]:
```
#eval water_quality and quality_group for dropping
col_eval(df_clean,cat_cols=['water_quality','quality_group'])
```

executed in 335ms, finished 20:49:54 2021-05-23

```
============================================================
Column Name: water_quality


Number of unique values: 8


There are 59392 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 soft                  50818
salty                  4856
unknown                1876
milky                   804
coloured                490
salty abandoned         339
fluoride                200
fluoride abandoned       17
Name: water_quality, dtype: int64
```



```
================================================================
Column Name: quality_group


Number of unique values: 6


There are 59394 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 good         50818
salty         5195
unknown       1876
milky          804
colored        490
fluoride       217
Name: quality_group, dtype: int64
```



In [657]:
```
1 #eval water_quality and quality_group
2 df_clean.groupby(by=['water_quality','quality_group']).count().index
```

executed in 83ms, finished 20:49:54 2021-05-23

Out[657]:
```
MultiIndex([(          'coloured',  'colored'),
            (          'fluoride', 'fluoride'),
            ('fluoride abandoned', 'fluoride'),
            (             'milky',    'milky'),
            (             'salty',    'salty'),
            (   'salty abandoned',    'salty'),
            (              'soft',     'good'),
            (           'unknown',  'unknown')],
           names=['water_quality', 'quality_group'])
```

OBSERVATIONS

- `quality_group` is redundant, need to update "soft" label to "soft-good".

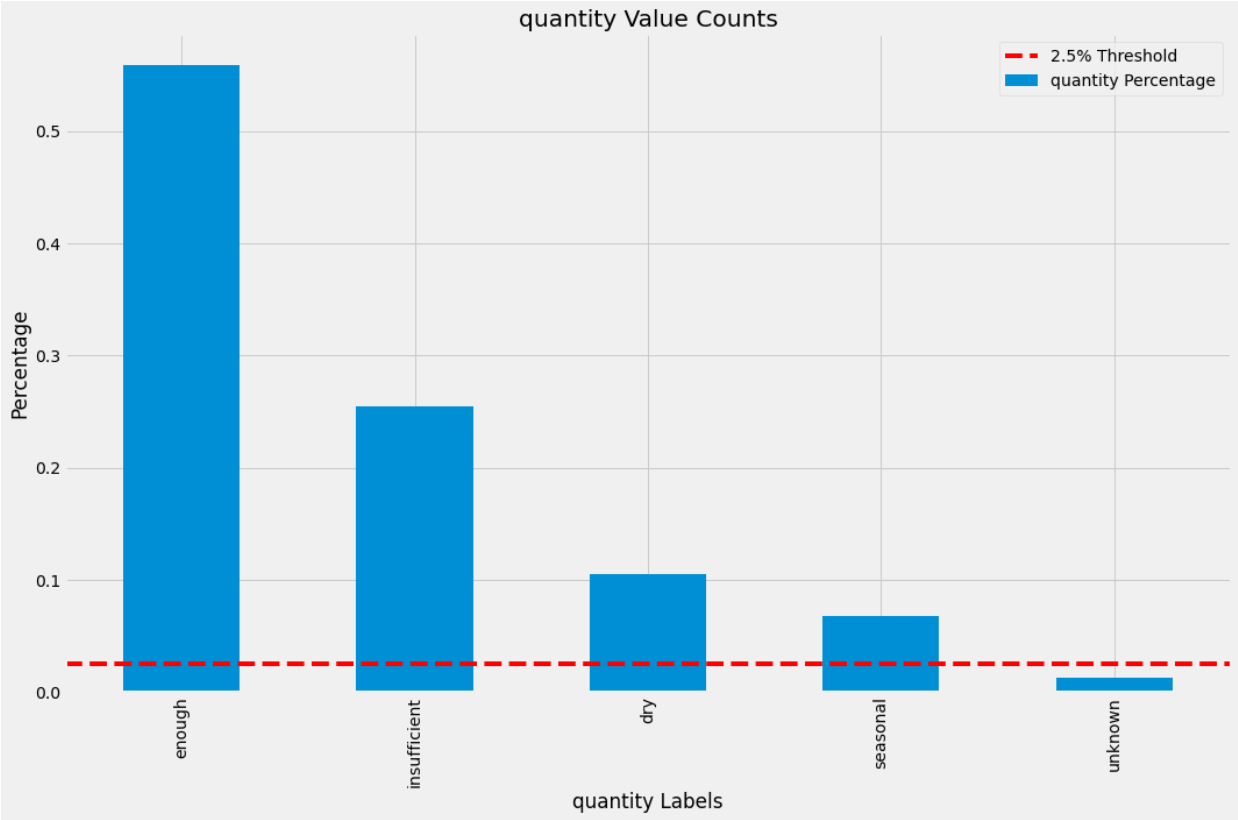
ACTIONS

- drop `quality_group` , update "soft" label to "soft-good" on `water_quality` feature

In [658]:
```
#evaluate quantity and quantity group
col_eval(df_clean, cat_cols=['quantity','quantity_group'])
```

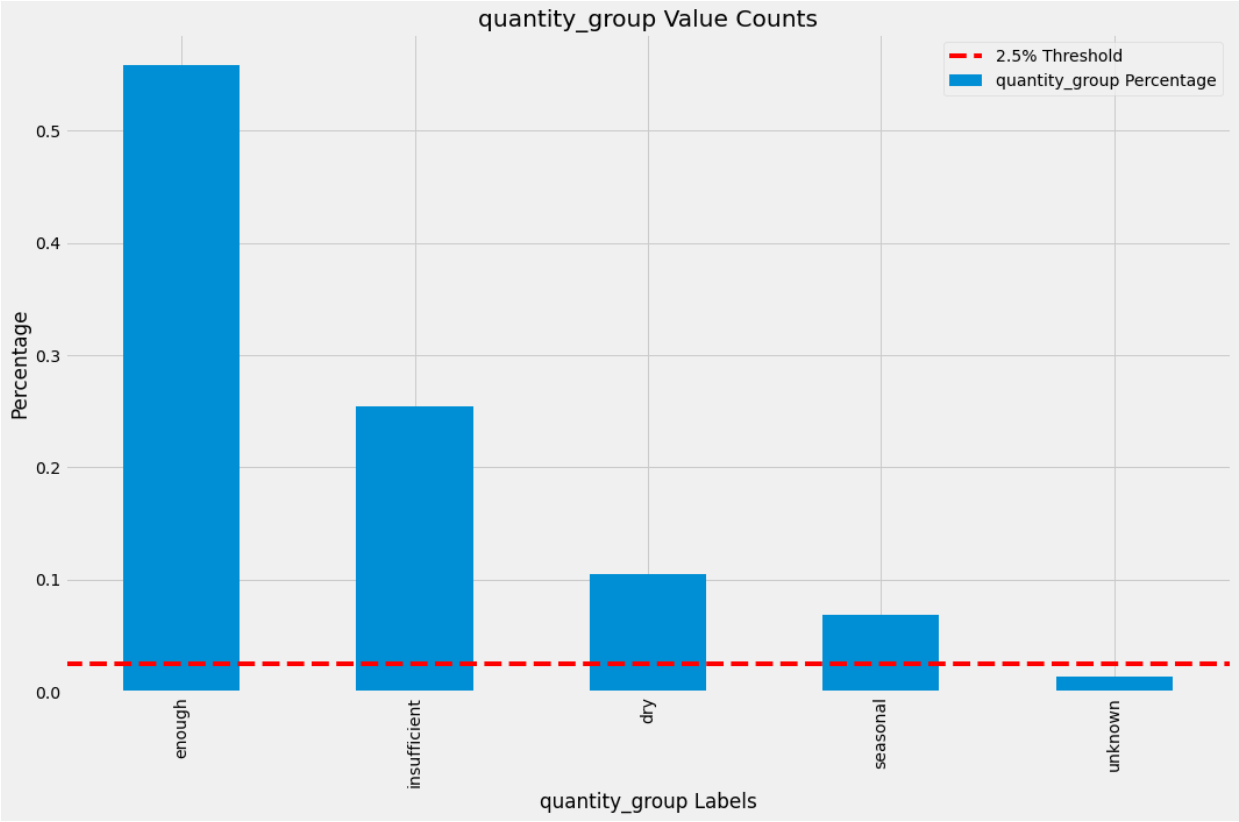
executed in 389ms, finished 20:49:55 2021-05-23

```
============================================================
Column Name: quantity


Number of unique values: 5


There are 59395 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 enough          33186
insufficient    15129
dry              6246
seasonal         4050
unknown           789
Name: quantity, dtype: int64
```




```
============================================================
Column Name: quantity_group
```

```
Number of unique values: 5


There are 59395 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 enough          33186
insufficient    15129
dry              6246
seasonal         4050
unknown           789
Name: quantity_group, dtype: int64
```



In [659]:
```
#eval quantity and quantity_group
df_clean.groupby(by=['quantity','quantity_group']).count().index
```

executed in 83ms, finished 20:49:55 2021-05-23

Out[659]:
```
MultiIndex([(         'dry',           'dry'),
            (      'enough',        'enough'),
            ('insufficient', 'insufficient'),
            (    'seasonal',      'seasonal'),
            (     'unknown',       'unknown')],
           names=['quantity', 'quantity_group'])
```

OBSERVATIONS

- `quantity` and `quantity_group` are the same.

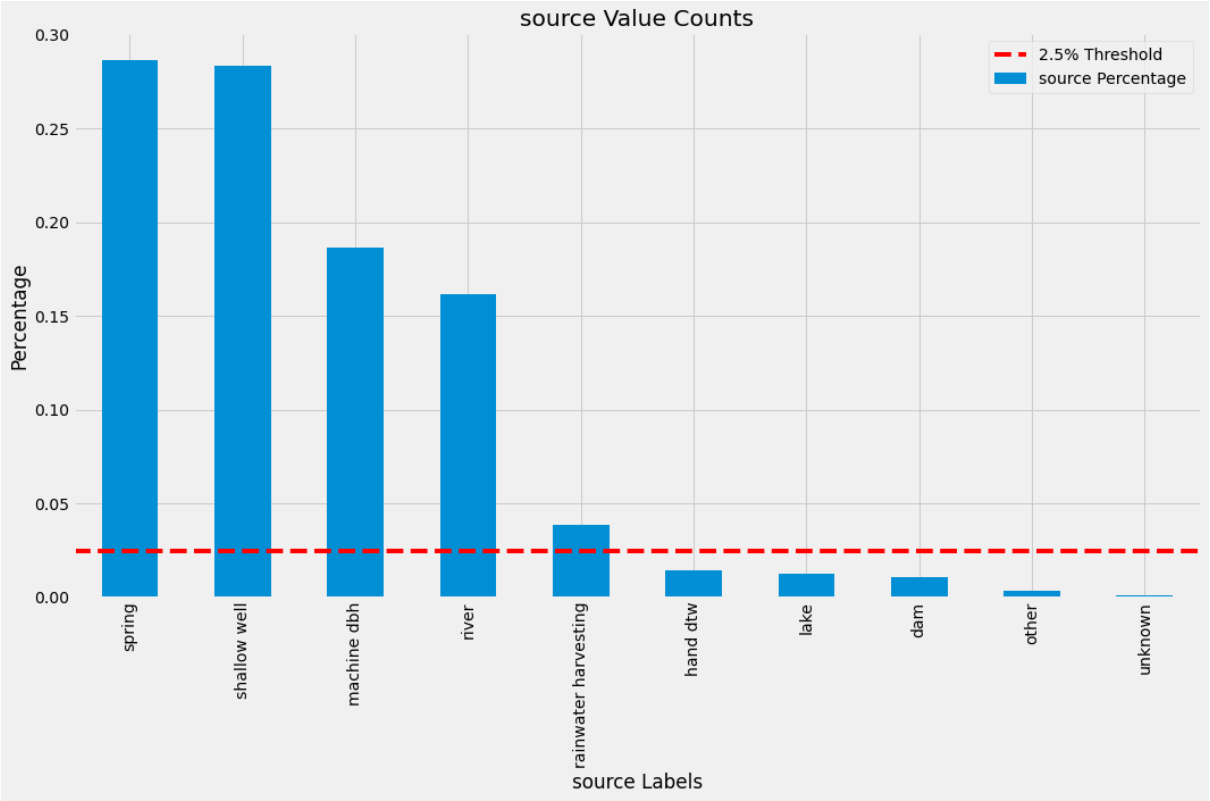
ACTIONS

- I will drop `quantity_group`

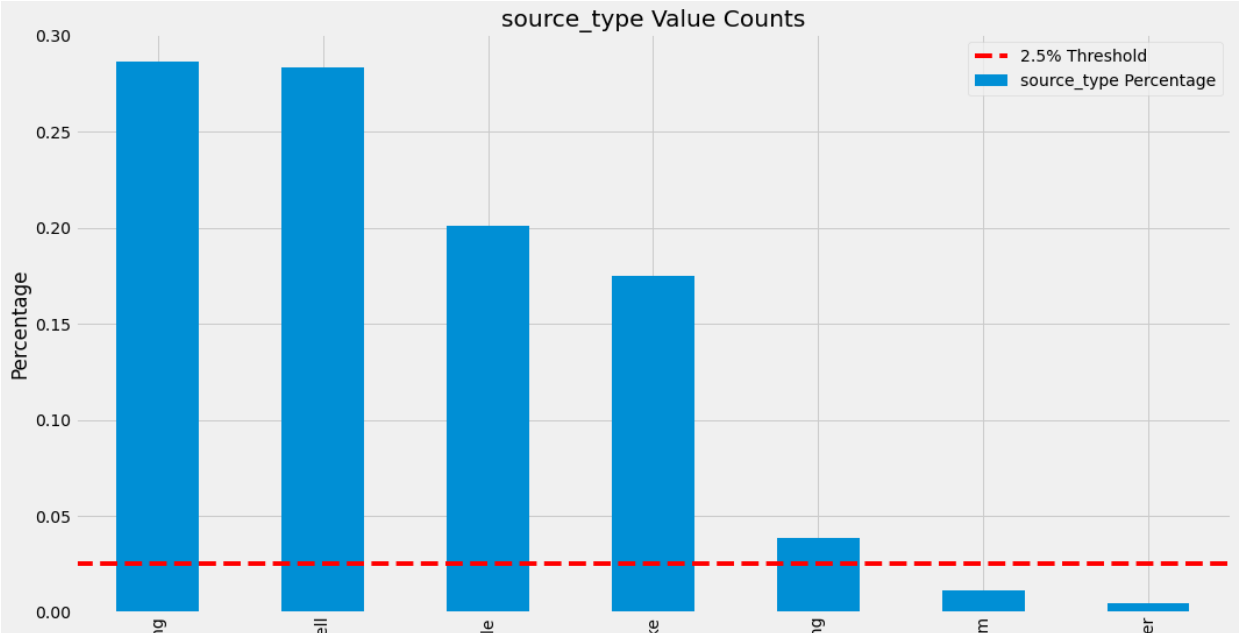
In [660]:
```
#evaluate the source, source_type, source_class features
col_eval(df_clean, cat_cols=['source','source_type','source_class'])
```

executed in 487ms, finished 20:49:55 2021-05-23

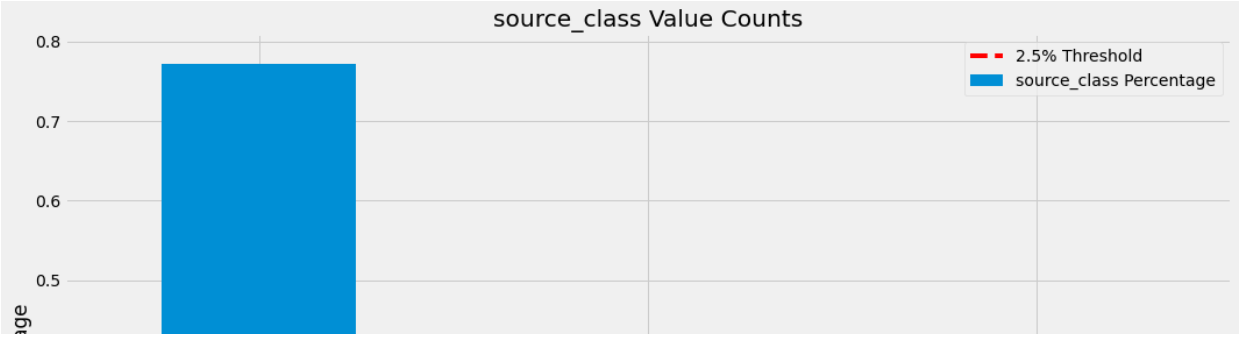
```
================================================================
Column Name: source


Number of unique values: 10


There are 59390 duplicates


There are 0 null values


There are 0 zeros


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




```
============================================================
Column Name: source_type


Number of unique values: 7


There are 59393 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 spring                  17021
shallow well            16824
borehole                11949
river/lake              10377
rainwater harvesting     2295
dam                       656
other                     278
Name: source_type, dtype: int64
```




```
============================================================
Column Name: source_class


Number of unique values: 3


There are 59397 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 groundwater    45794
surface         13328
unknown           278
Name: source_class, dtype: int64
```



In [661]:
```
#groupby to see redundancy
df_clean.groupby(by=['source_class','source_type','source']).count().in
```

executed in 83ms, finished 20:49:55 2021-05-23

Out[661]:
```
MultiIndex([('groundwater',             'borehole',             'hand dt
w'),
            ('groundwater',             'borehole',          'machine db
h'),
            ('groundwater',         'shallow well',         'shallow wel
l'),
            ('groundwater',               'spring',               'sprin
g'),
            (    'surface',                  'dam',                  'da
m'),
            (    'surface', 'rainwater harvesting', 'rainwater harvestin
g'),
            (    'surface',           'river/lake',                'lak
e'),
            (    'surface',           'river/lake',                'rive
r'),
            (    'unknown',                'other',                'othe
r'),
            (    'unknown',                'other',             'unknow
n')],
           names=['source_class', 'source_type', 'source'])
```

OBSERVATIONS

- `source_type` is redundant information

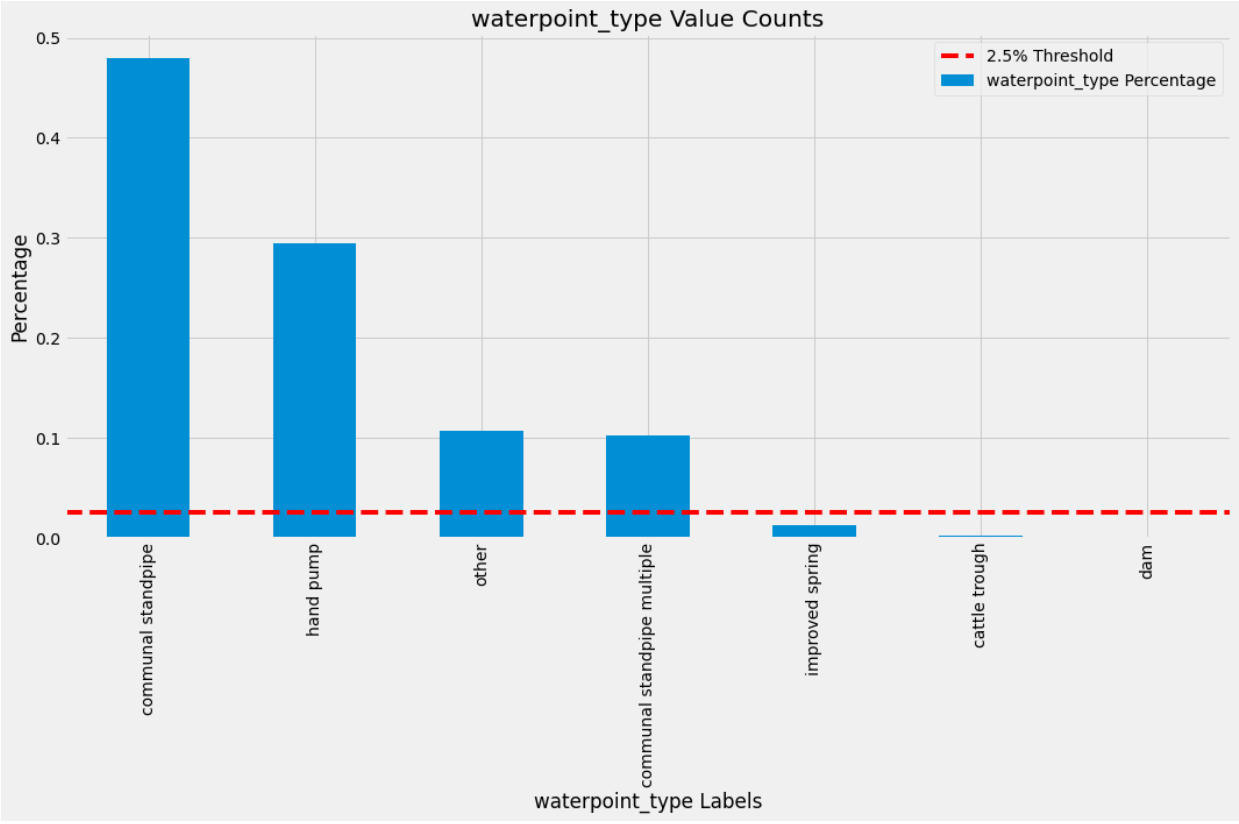
ACTIONS

- I will remove `source_type` and rename `source_class` as `source_type_1` and `source` as `source_type_2`

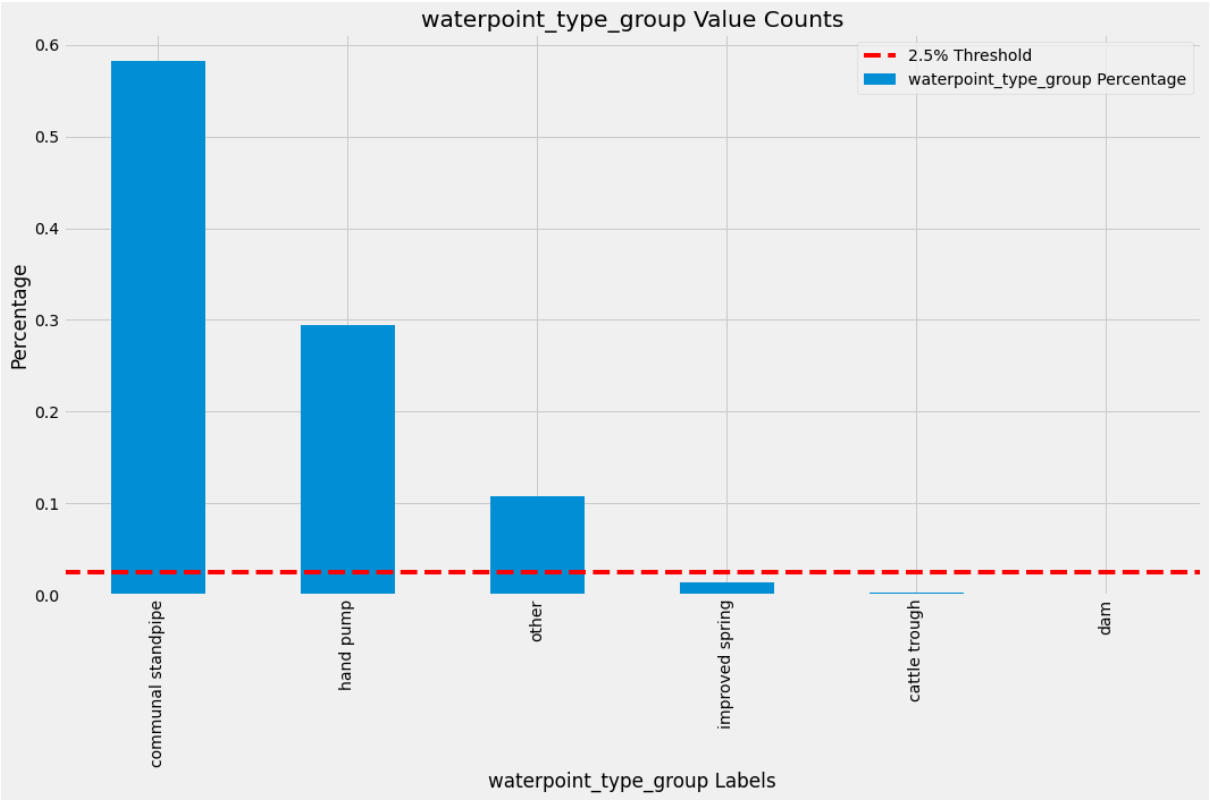
In [662]:
```
#evaluation waterpoint_type and waterpoint_type_group
col_eval(df_clean, cat_cols=['waterpoint_type','waterpoint_type_group'])
```

executed in 323ms, finished 20:49:56 2021-05-23

```
================================================================
Column Name: waterpoint_type


Number of unique values: 7


There are 59393 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 communal standpipe             28522
hand pump                       17488
other                            6380
communal standpipe multiple      6103
improved spring                   784
cattle trough                     116
dam                                 7
Name: waterpoint_type, dtype: int64
```



```
============================================================
Column Name: waterpoint_type_group

Number of unique values: 6

There are 59394 duplicates

There are 0 null values

There are 0 zeros

Value Counts Percentage
 communal standpipe    34625
hand pump             17488
other                  6380
improved spring         784
cattle trough           116
dam                       7
Name: waterpoint_type_group, dtype: int64
```



In [663]:
```
#groupby to see redundancy
df_clean.groupby(by=['waterpoint_type','waterpoint_type_group']).count(
```

executed in 84ms, finished 20:49:56 2021-05-23

Out[663]:
```
MultiIndex([(              'cattle trough',      'cattle trough'),
            (         'communal standpipe', 'communal standpipe'),
            ('communal standpipe multiple', 'communal standpipe'),
            (                        'dam',               'dam'),
            (                 'hand pump',         'hand pump'),
            (            'improved spring',    'improved spring'),
            (                      'other',             'other')],
           names=['waterpoint_type', 'waterpoint_type_group'])
```

OBSERVATIONS

- features are the essentially the same. `waterpoint_type` has more granularity so will keep

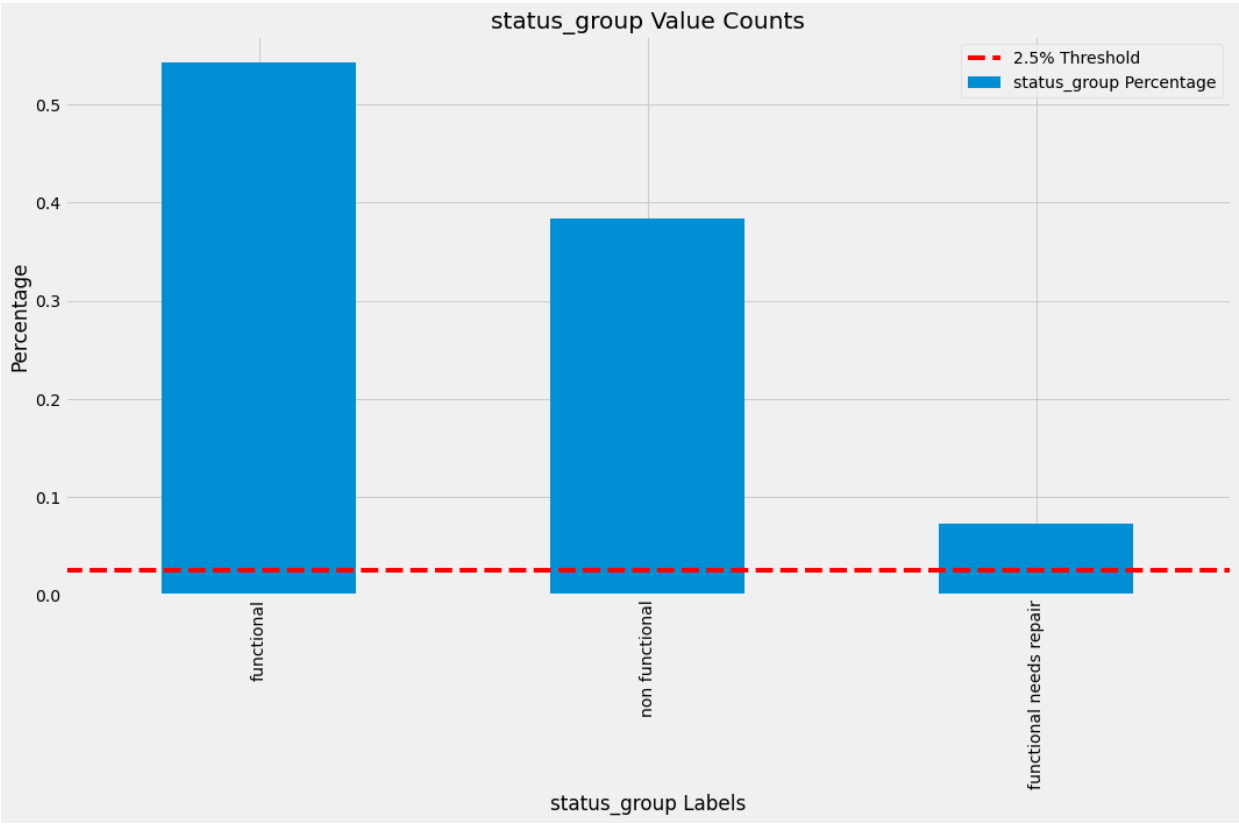
ACTIONS

- Remove `waterpoint_type_group` feature

In [664]:
```
1 col_eval(df_clean, cat_cols=['status_group'])
```

executed in 141ms, finished 20:49:56 2021-05-23

```
============================================================
Column Name: status_group


Number of unique values: 3


There are 59397 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 functional                 32259
non functional              22824
functional needs repair      4317
Name: status_group, dtype: int64
```



OBSERVATIONS

- I believe "functional needs repair" is equivalent to "non functional" since it suggests the waterpoint has a failure but is still somehow functional. To me, functional with a failure means that the waterpoint is operating at a degraded state and not as optimal as it could.

ACTIONS

- I will combine "functional needs repair" to the "non functional" group which will simplify the modeling as well.

==========

SUMMARY OF ACTIONS TO TAKE

- recast `public_meeting` as string
- recast `permit` as a string
- recast `id` as categorical
- recast `region_code` as categorical
- recast `district_code` as categorical
- drop `num_private`
- drop `region_code`
- drop `recorded_by`
- drop `extraction_type_group`
- drop `management_group`
- drop `payment_type`
- drop `quality_group`
- drop `quantity_group`

- drop `source_type`
- drop `waterpoint_type_group`
- drop `scheme_name`
- drop `subvillage`
- impute `construction_year`
- impute `subvillage` and consolidate rare labels
- impute 0's in `funder` and `installer` as "Unknown"
- impute `public_meeting` missing values as "unknown"
- impute null values in the `scheme_management` and `scheme_name` features as "Unknown"
- impute `permit` nulls as "unknown"
- impute missing values in `scheme_management` with "unknown"
- rename `extraction_type_class` as `extraction_type_1` and `extraction_type` as `extraction_type_2` and rename labels
- rename "soft" label to "soft-good" on `water_quality` feature
- rename `source_class` as `source_type_1` and `source` as `source_type_2`
- rename "functional needs repair" to "non functional" for the `status_group` feature
- rename `payment` to `water_cost`
- rename `quantity` as `water_quantity`
- reduce cardinality in `funder` and `installer` features by consolidating rare labels
- remove rows where longitude = 0

5.2 Data Type Recasting

I will change data types of features here.

In [665]:
```
df_clean.dtypes
```
executed in 4ms, finished 20:49:56 2021-05-23

Out[665]:
```
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
```

In [666]:
```
#convert id to categorical
df_clean['id'] = df_clean['id'].astype('object')
```
executed in 4ms, finished 20:49:56 2021-05-23

In [667]:
```
#convert region_code to categorical
df_clean['region_code'] = df_clean['region_code'].astype('object')
```
executed in 3ms, finished 20:49:56 2021-05-23

In [668]:
```
#convert district_code to categorical
df_clean['district_code'] = df_clean['district_code'].astype('object')
```
executed in 3ms, finished 20:49:56 2021-05-23

In [669]:
```
df_clean['public_meeting'].value_counts(dropna=False)
```
executed in 8ms, finished 20:49:56 2021-05-23

Out[669]:
```
True     51011
False     5055
NaN       3334
Name: public_meeting, dtype: int64
```

In [670]:
```
#convert public meeting to string type
df_clean['public_meeting'] = df_clean['public_meeting'].astype(str)

#convert 'nan' to nulls
df_clean['public_meeting'].replace('nan',np.nan, inplace=True)
```
executed in 14ms, finished 20:49:56 2021-05-23

In [671]:
```
df_clean['public_meeting'].value_counts(dropna=False)
```
executed in 8ms, finished 20:49:56 2021-05-23

Out[671]:
```
True     51011
False     5055
NaN       3334
Name: public_meeting, dtype: int64
```

In [672]:
```
df_clean['permit'].value_counts(dropna=False)
```
executed in 8ms, finished 20:49:56 2021-05-23

Out[672]:
```
True     38852
False    17492
NaN       3056
Name: permit, dtype: int64
```

In [673]:
```
#convert permit to string type
df_clean['permit'] = df_clean['permit'].astype(str)

#convert 'nan' to nulls
df_clean['permit'].replace('nan',np.nan, inplace=True)
```
executed in 14ms, finished 20:49:56 2021-05-23

In [674]:
```
df_clean['permit'].value_counts(dropna=False)
```
executed in 8ms, finished 20:49:56 2021-05-23

Out[674]:
```
True     38852
False    17492
NaN       3056
Name: permit, dtype: int64
```

In [675]:

```
#extract the year from date_recorded feature
df_clean['date_recorded_yr'] = pd.to_datetime(df_clean['date_recorded']
df_clean['date_recorded_yr'] = pd.to_datetime(df_clean['date_recorded']
df_clean[['date_recorded_yr','date_recorded']].head()
```

executed in 45ms, finished 20:49:56 2021-05-23

Out[675]:

	date_recorded_yr	date_recorded
0	2011	2011-03-14
1	2013	2013-03-06
2	2013	2013-02-25
3	2013	2013-01-28
4	2011	2011-07-13

In [676]:
```
#review dataframe
df_clean
```

executed in 105ms, finished 20:49:56 2021-05-23

Out[676]:

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitud
0	69572	6,000.0	2011-03-14	Roman	1390	Roman	34.93809275	-9.8563217
1	8776	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.6987661	-2.147465
2	34310	25.0	2013-02-25	Lottery Club	686	World vision	37.46066446	-3.821328
3	67743	0.0	2013-01-28	Unicef	263	UNICEF	38.48616088	-11.155297
4	19728	0.0	2011-07-13	Action In A	0	Artisan	31.13084671	-1.825358
...	...	...	...	...	...	...	...	
59395	60739	10.0	2013-05-03	Germany Republi	1210	CES	37.16980689	-3.253847
59396	27263	4,700.0	2011-05-07	Cefa-njombe	1212	Cefa	35.24999126	-9.07062
59397	37057	0.0	2011-04-11	NaN	0	NaN	34.01708706	-8.750434
59398	31282	0.0	2011-03-08	Malec	0	Musa	35.86131531	-6.378573
59399	26348	0.0	2011-03-23	World Bank	191	World	38.10404822	-6.747464

59400 rows × 42 columns

5.3 Feature/Row Drop

I will drop any unecessary features here.

In [677]:
```
#drop all features outlined in feature evaluation for training set
df_clean = df_clean.drop(columns=['region_code','recorded_by','extracti
                         'management_group','payment_type','quality_group
                         'source_type','waterpoint_type_group','scheme_na
                        'num_private','subvillage'])
```
executed in 12ms, finished 20:49:56 2021-05-23

In [678]:
```
#remove rows where longitude == 0
df_clean.drop(df_clean.loc[df_clean['longitude'] == 0].index, inplace=T
```
executed in 16ms, finished 20:49:56 2021-05-23

In [679]:
```
#check for date_recorded_year year earlier than construction year
df_clean.loc[df_clean['date_recorded_yr'] < df_clean['construction_year
                         ['id']].count()
```
executed in 5ms, finished 20:49:56 2021-05-23

Out[679]:
```
id    9
dtype: int64
```

In [680]:
```
#drop date_recorded
df_clean.drop(columns='date_recorded', inplace=True)
```
executed in 13ms, finished 20:49:56 2021-05-23

OBSERVATIONS

- There are some records where the `date_recorded_yr` is prior to the `construction_year`.

ACTIONS

- I will remove these rows.

In [681]:
```
#drop rows
df_clean.drop(df_clean.loc[df_clean['date_recorded_yr'] <
                           df_clean['construction_year']].index, inplac
```
executed in 16ms, finished 20:49:56 2021-05-23

In [682]:
```
#check for date_recorded_year year earlier than construction year
df_clean.loc[df_clean['date_recorded_yr'] < df_clean['construction_year
                         ['id']].count()
```
executed in 4ms, finished 20:49:56 2021-05-23

Out[682]:
```
id    0
dtype: int64
```

5.4 Feature and Label Renaming

I will rename features and labels here to make descriptions more concise.

In [683]:
```
#rename features
df_clean.rename(columns={'amount_tsh':'head','gps_height':'well_elevati
                inplace=True)
```
executed in 2ms, finished 20:49:56 2021-05-23

In [684]:
```
#rename extraction_type_class feature as extraction_type_1
df_clean.rename(columns={'extraction_type_class':'extraction_type_1'},
df_clean.rename(columns={'extraction_type':'extraction_type_2'}, inplac
```
executed in 2ms, finished 20:49:56 2021-05-23

In [685]:
```
#rename labels in extraction_type_2
df_clean.loc[df_clean['extraction_type_2'] == 'other - mkulima/shinyang
             ['extraction_type_2']] = 'mkulima/shinyanga'

df_clean.loc[df_clean['extraction_type_2'] == 'other - play pump',
             ['extraction_type_2']] = 'play pump'

df_clean.loc[df_clean['extraction_type_2'] == 'other - swn 81',
             ['extraction_type_2']] = 'swn 81'

df_clean.loc[df_clean['extraction_type_2'] == 'other - rope pump',
             ['extraction_type_2']] = 'rope pump'
```
executed in 19ms, finished 20:49:56 2021-05-23

In [686]:
```
#rename soft label as soft-good for water_quality feature
df_clean.loc[df_clean['water_quality'] == 'soft','water_quality'] = 'so
```
executed in 6ms, finished 20:49:56 2021-05-23

In [687]:
```
#rename source features
df_clean.rename(columns={'source_class':'source_type_1',
                         'source':'source_type_2'},inplace=True)
```
executed in 2ms, finished 20:49:56 2021-05-23

In [688]:
```
#update status_group labels
status_group_dict = {'functional':'functional',
                     'non functional':'non functional',
                     'functional needs repair':'non functional'}
df_clean['status_group'] = df_clean['status_group'].map(status_group_di
df_clean['status_group'].value_counts()
```
executed in 13ms, finished 20:49:56 2021-05-23

Out[688]:
```
functional        31385
non functional    26194
Name: status_group, dtype: int64
```

In [689]:
```
#rename payment to water_cost
df_clean.rename(columns={'payment':'water_cost'},inplace=True)
```
executed in 3ms, finished 20:49:56 2021-05-23

In [690]:

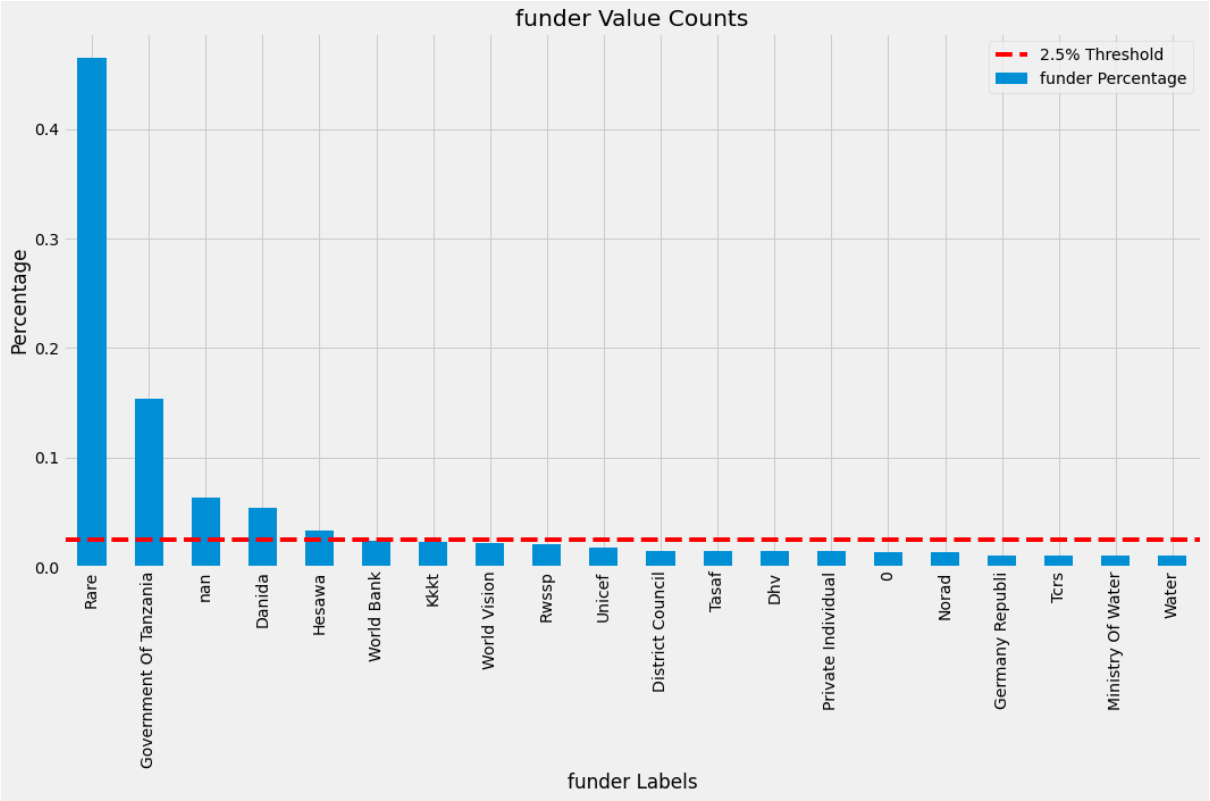
```
#rename quantity to water_quantity
df_clean.rename(columns={'quantity':'water_quantity'}, inplace=True)
```

executed in 2ms, finished 20:49:56 2021-05-23

In [691]:
```
#create 'rare' consolidated label for funder
df_clean['funder'] = rare_labels(df_clean, 'funder', thresh=.01)
col_eval(df_clean, cat_cols=['funder'])
```

executed in 396ms, finished 20:49:57 2021-05-23

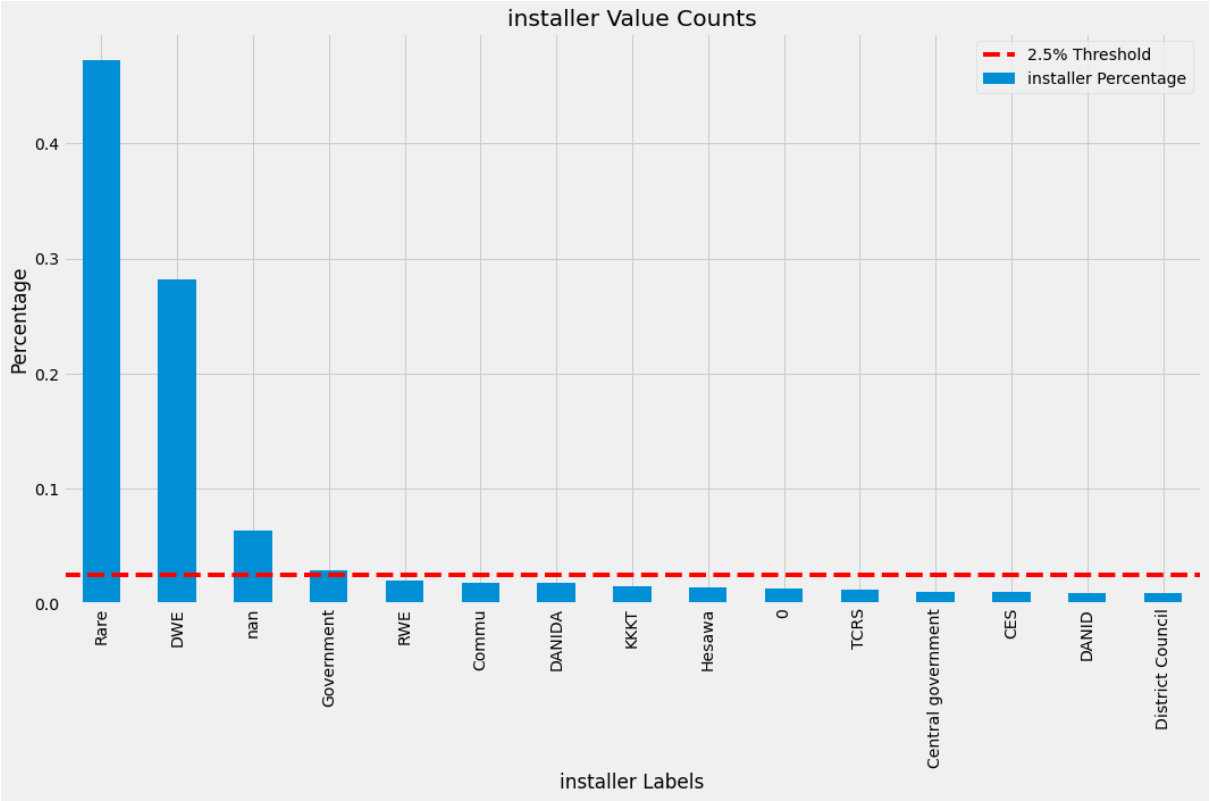
```
============================================================
Column Name: funder


Number of unique values: 19


There are 57559 duplicates


There are 3622 null values


There are 777 zeros


Value Counts Percentage
 Rare                        26754
Government Of Tanzania       8841
NaN                          3622
Danida                       3114
Hesawa                       1914
World Bank                   1345
Kkkt                         1286
World Vision                 1224
Rwssp                        1187
Unicef                       1035
District Council              843
Tasaf                         834
Dhv                           829
Private Individual            824
0                             777
Norad                         765
Germany Republi               610
Tcrs                          602
Ministry Of Water             590
Water                         583
Name: funder, dtype: int64
```


funder Value Counts
2.5% Threshold
funder Percentage
Percentage
0.0
0.1
0.2
0.3
0.4
Rare
Government Of Tanzania
nan
Danida
Hesawa
World Bank
Kkkt
World Vision
Rwssp
Unicef
District Council
Tasaf
Dhv
Private Individual
0
Norad
Germany Republi
Tcrs
Ministry Of Water
Water
funder Labels

In [692]:
```
#create 'rare' consolidated label for installer
df_clean['installer'] = rare_labels(df_clean, 'installer', thresh=.01)
col_eval(df_clean, cat_cols=['installer'])
```

executed in 406ms, finished 20:49:57 2021-05-23

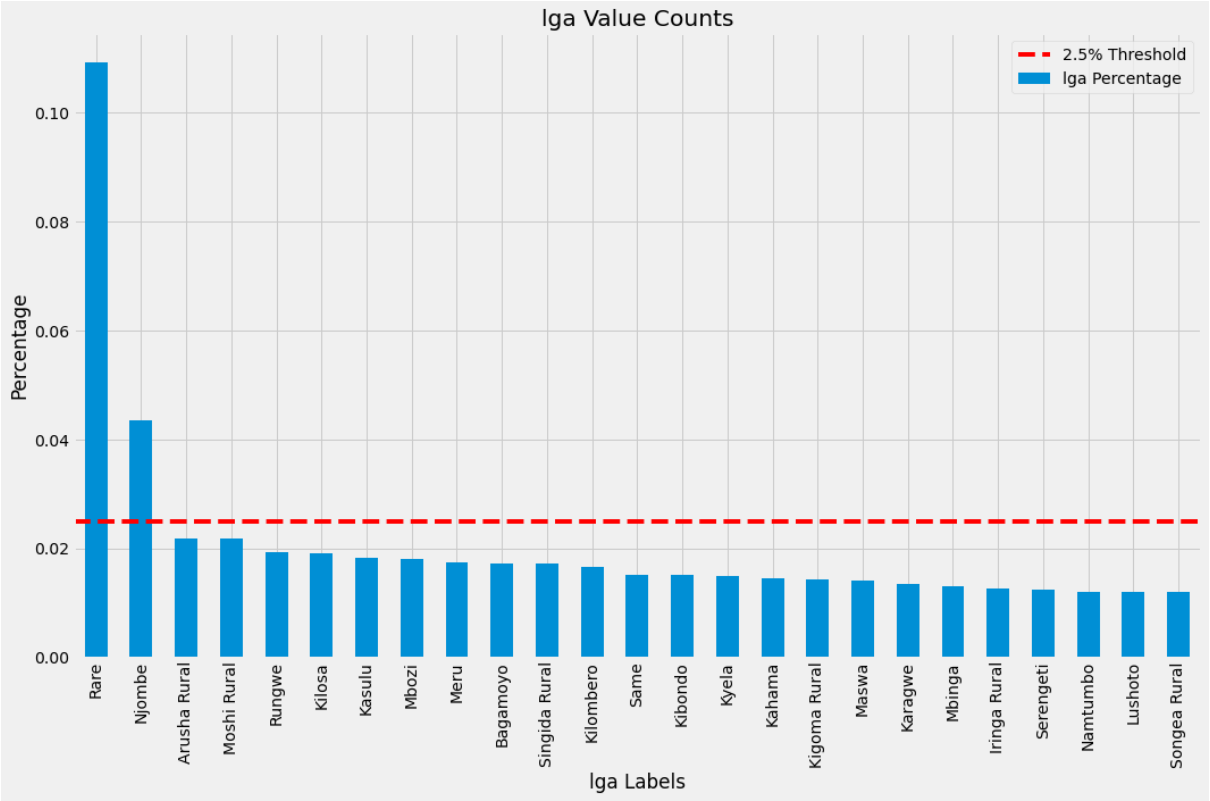
```
============================================================
Column Name: installer


Number of unique values: 14


There are 57564 duplicates


There are 3636 null values


There are 777 zeros


Value Counts Percentage
 Rare                  27213
DWE                   16253
NaN                    3636
Government             1670
RWE                    1181
Commu                  1060
DANIDA                 1050
KKKT                    897
Hesawa                  803
0                       777
TCRS                    707
Central government      619
CES                     610
DANID                   552
District Council        551
Name: installer, dtype: int64
```


installer Value Counts
2.5% Threshold
installer Percentage
Percentage
0.0
0.1
0.2
0.3
0.4
Rare
DWE
nan
Government
RWE
Commu
DANIDA
KKKT
Hesawa
0
TCRS
Central government
CES
DANID
District Council
installer Labels

In [693]:
```
#create 'rare' consolidated label for lga
df_clean['lga'] = rare_labels(df_clean, 'lga', thresh=.005)
col_eval(df_clean, cat_cols=['lga'])
```

executed in 256ms, finished 20:49:57 2021-05-23

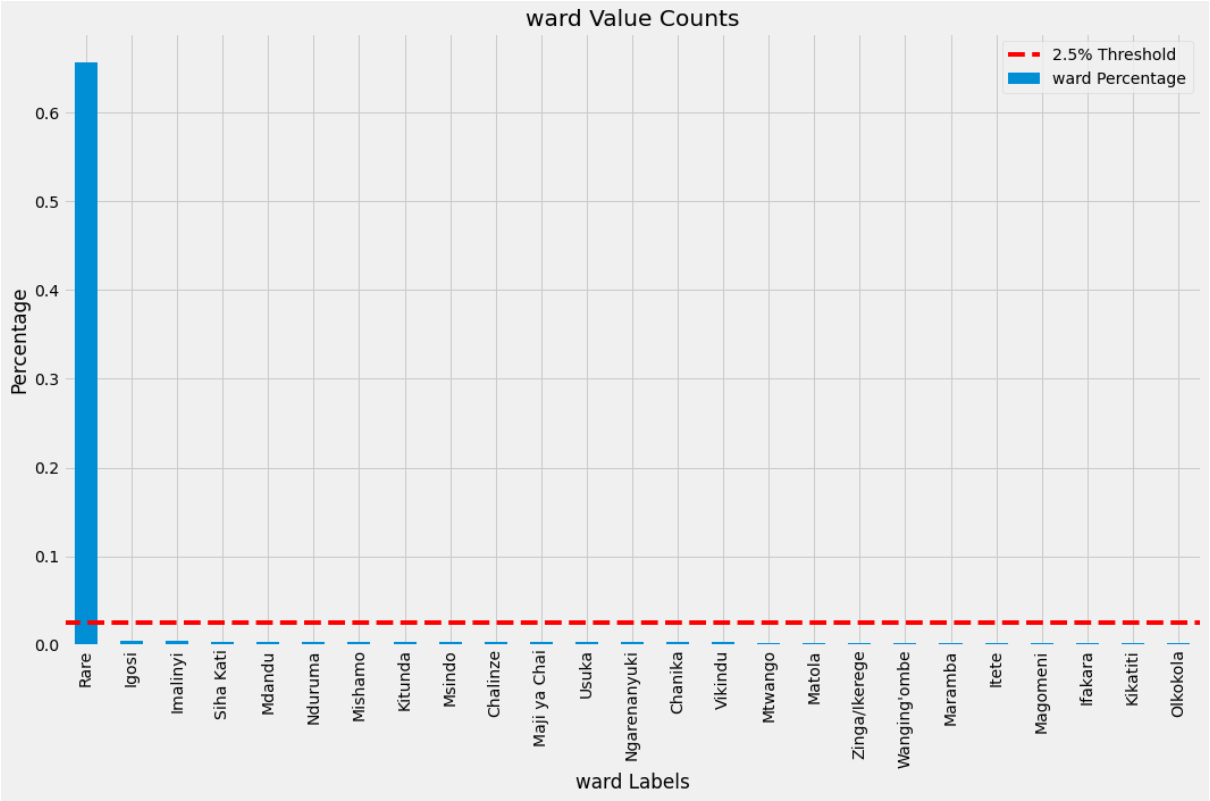
```
============================================================
Column Name: lga


Number of unique values: 87


There are 57492 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 Rare            6288
Njombe          2502
Arusha Rural    1252
Moshi Rural     1251
Rungwe          1106
                ...
Nachingwea       300
Chunya           298
Mbulu            297
Ruangwa          291
Mkinga           288
Name: lga, Length: 87, dtype: int64
```


lga Value Counts
2.5% Threshold
lga Percentage
Percentage
0.00
0.02
0.04
0.06
0.08
0.10
Rare
Njombe
Arusha Rural
Moshi Rural
Rungwe
Kilosa
Kasulu
Mbozi
Meru
Bagamoyo
Singida Rural
Kilombero
Same
Kibondo
Kyela
Kahama
Kigoma Rural
Maswa
Karagwe
Mbinga
Iringa Rural
Serengeti
Namtumbo
Lushoto
Songea Rural
lga Labels

In [694]:
```
#create 'rare' consolidated label for ward
df_clean['ward'] = rare_labels(df_clean, 'ward', thresh=.001)
col_eval(df_clean, cat_cols=['ward'])
```

executed in 470ms, finished 20:49:58 2021-05-23

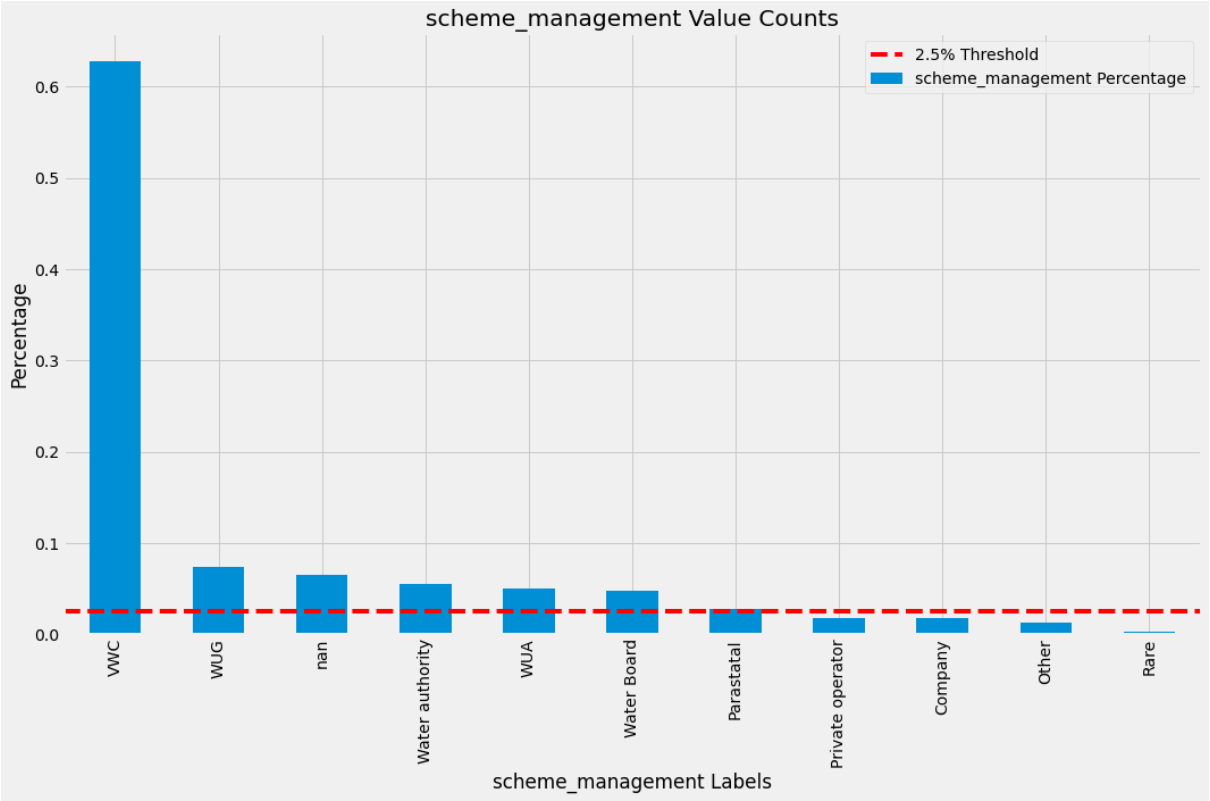
```
============================================================
Column Name: ward


Number of unique values: 223


There are 57356 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 Rare               37837
Igosi                307
Imalinyi             252
Siha Kati            232
Mdandu               231
                    ...
Kiruruma              58
Siha Kaskazini        58
Kisemu                58
Madibira              58
Lulindi               58
Name: ward, Length: 223, dtype: int64
```


ward Value Counts
2.5% Threshold
ward Percentage
Percentage
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Rare
Igosi
Imalinyi
Siha Kati
Mdandu
Nduruma
Mishamo
Kitunda
Msindo
Chalinze
Maji ya Chai
Usuka
Ngarenanyuki
Chanika
Vikindu
Mtwango
Matola
Zinga/Ikerege
Wanging'ombe
Maramba
Itete
Magomeni
Ifakara
Kikatiti
Olkokola
ward Labels

In [695]:
```
#create 'rare' consolidated label for scheme_management
df_clean['scheme_management'] = rare_labels(df_clean, 'scheme_managemen
col_eval(df_clean, cat_cols=['scheme_management'])
```

executed in 189ms, finished 20:49:58 2021-05-23

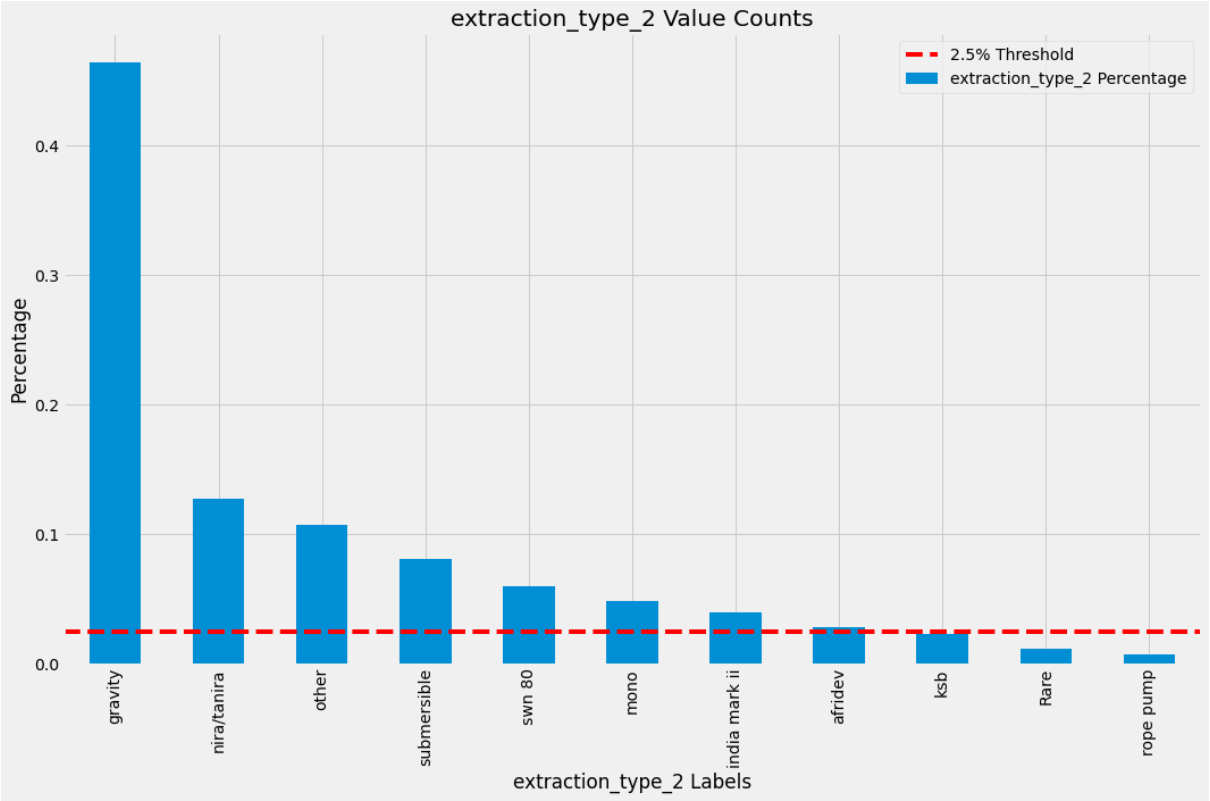
```
============================================================
Column Name: scheme_management


Number of unique values: 10


There are 57568 duplicates


There are 3750 null values


There are 0 zeros


Value Counts Percentage
 VWC                 36136
WUG                  4249
NaN                  3750
Water authority      3151
WUA                  2882
Water Board          2747
Parastatal           1607
Private operator     1062
Company              1061
Other                 764
Rare                  170
Name: scheme_management, dtype: int64
```


scheme_management Value Counts
2.5% Threshold
scheme_management Percentage
Percentage
0.0
0.1
0.2
0.3
0.4
0.5
0.6
VWC
WUG
nan
Water authority
WUA
Water Board
Parastatal
Private operator
Company
Other
Rare
scheme_management Labels

In [696]:
```
#create 'rare' consolidated label for extraction_type_2
df_clean['extraction_type_2'] = rare_labels(df_clean, 'extraction_type_
col_eval(df_clean, cat_cols=['extraction_type_2'])
```

executed in 184ms, finished 20:49:58 2021-05-23

```
============================================================
Column Name: extraction_type_2


Number of unique values: 11


There are 57568 duplicates


There are 0 null values


There are 0 zeros


Value Counts Percentage
 gravity          26693
nira/tanira       7360
other             6159
submersible       4688
swn 80            3447
mono              2817
india mark ii     2283
afridev           1659
ksb               1358
Rare               665
rope pump          450
Name: extraction_type_2, dtype: int64
```



In [697]:
```
#change status_group to binary
df_clean['status_group'] = df_clean['status_group'].map({'functional':0
```

executed in 11ms, finished 20:49:58 2021-05-23

In [698]:
```
#reorder features
df_clean = df_clean[['id','wpt_name','construction_year','waterpoint_ty
                     'water_quality','water_quantity','head',
                     'source_type_1','source_type_2','extraction_type_1
                     'extraction_type_2','well_elevation','population',
                     'status_group','latitude','longitude','basin',
                     'district_code','region','lga','ward',
                     'funder','installer','permit','public_meeting',
                     'scheme_management','management','water_cost',
                     'date_recorded_yr']]
```

executed in 15ms, finished 20:49:58 2021-05-23

In [699]:
```
#state of the dataframe
df_clean
```
executed in 19ms, finished 20:49:58 2021-05-23

Out[699]:

	id	wpt_name	construction_year	waterpoint_type	water_quality	water_quantity	head
0	69572	none	1999	communal standpipe	soft-good	enough	6,000.0
1	8776	Zahanati	2010	communal standpipe	soft-good	insufficient	0.0
2	34310	Kwa Mahundi	2009	communal standpipe multiple	soft-good	enough	25.0
3	67743	Zahanati Ya Nanyumbu	1986	communal standpipe multiple	soft-good	dry	0.0
4	19728	Shuleni	0	communal standpipe	soft-good	seasonal	0.0
...	...	...	...	...	...	...	...
59395	60739	Area Three Namba 27	1999	communal standpipe	soft-good	enough	10.0
59396	27263	Kwa Yahona Kuvala	1996	communal standpipe	soft-good	enough	4,700.0
59397	37057	Mashine	0	hand pump	fluoride	enough	0.0
59398	31282	Mshoro	0	hand pump	soft-good	insufficient	0.0
59399	26348	Kwa Mzee Lugawa	2002	hand pump	salty	enough	0.0

57579 rows × 29 columns

5.5 Train-Test Split

In [700]:
```
# #create train-test split
X = df_clean.drop(columns='status_group')
y = df_clean['status_group']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3
```
executed in 30ms, finished 20:49:58 2021-05-23

In [701]:
```
1 X_train.shape, X_test.shape, y_train.shape, y_test.shape
```
executed in 3ms, finished 20:49:58 2021-05-23

Out[701]:
```
((40305, 28), (17274, 28), (40305,), (17274,))
```

In [702]:
```
1 X_train_tf = X_train.copy()
2 X_test_tf = X_test.copy()
```
executed in 9ms, finished 20:49:58 2021-05-23

In [703]:
```
1 print(X_train_tf.shape,y_train.shape)
```
executed in 3ms, finished 20:49:58 2021-05-23

```
(40305, 28) (40305,)
```

5.6 Imputation

In [704]:
```
1 #check for null values
2 X_train_tf.isnull().sum()
```
executed in 32ms, finished 20:49:58 2021-05-23

Out[704]:
```
id                     0
wpt_name               0
construction_year      0
waterpoint_type        0
water_quality          0
water_quantity         0
head                   0
source_type_1          0
source_type_2          0
extraction_type_1      0
extraction_type_2      0
well_elevation         0
population             0
latitude               0
longitude              0
basin                  0
district_code          0
region                 0
lga                    0
ward                   0
funder              2532
installer           2549
permit              2140
public_meeting      2100
scheme_management   2642
management             0
water_cost             0
date_recorded_yr       0
dtype: int64
```

OBSERVATIONS

- null values in `funder`, `installer`, `permit`, `public_meeting` and `scheme_management`

ACTIONS

- see if any of the columns have 0's for placeholders to convert to nan and then impute null values with "Unknown" since they are all categorical variables

In [705]:
```
#see how many columns have values equal to 0
for col in X_train_tf:
    count_num = (X_train_tf[col] == 0).sum()
    count_cat = (X_train_tf[col] == '0').sum()
    print(col,count_num, count_cat)

for col in X_test_tf:
    count_num = (X_test_tf[col] == 0).sum()
    count_cat = (X_test_tf[col] == '0').sum()
    print(col,count_num, count_cat)
```

executed in 154ms, finished 20:49:58 2021-05-23

```
id 0 0
wpt_name 0 0
construction_year 13241 0
waterpoint_type 0 0
water_quality 0 0
water_quantity 0 0
head 27848 0
source_type_1 0 0
source_type_2 0 0
extraction_type_1 0 0
extraction_type_2 0 0
well_elevation 13064 0
population 13738 0
latitude 0 0
longitude 0 0
basin 0 0
district_code 15 0
region 0 0
lga 0 0
ward 0 0
funder 0 544
installer 0 544
permit 0 0
public_meeting 0 0
scheme_management 0 0
management 0 0
water_cost 0 0
date_recorded_yr 0 0
id 1 0
wpt_name 0 0
construction_year 5656 0
waterpoint_type 0 0
water_quality 0 0
water_quantity 0 0
head 11974 0
source_type_1 0 0
source_type_2 0 0
extraction_type_1 0 0
extraction_type_2 0 0
well_elevation 5562 0
population 5831 0
latitude 0 0
longitude 0 0
basin 0 0
district_code 8 0
```

```
region 0 0
lga 0 0
ward 0 0
funder 0 233
installer 0 233
permit 0 0
public_meeting 0 0
scheme_management 0 0
management 0 0
water_cost 0 0
date_recorded_yr 0 0
```

OBSERVATIONS

- `construction_year`, `funder` and `installer` are the only feature that shouldn't have 0's

ACTIONS

- convert 0's to nulls

In [706]:
```
#convert 0's to nulls in construction_year
X_train_tf['construction_year'].replace(0, np.nan, inplace=True)
X_test_tf['construction_year'].replace(0, np.nan, inplace=True)
```

executed in 5ms, finished 20:49:58 2021-05-23

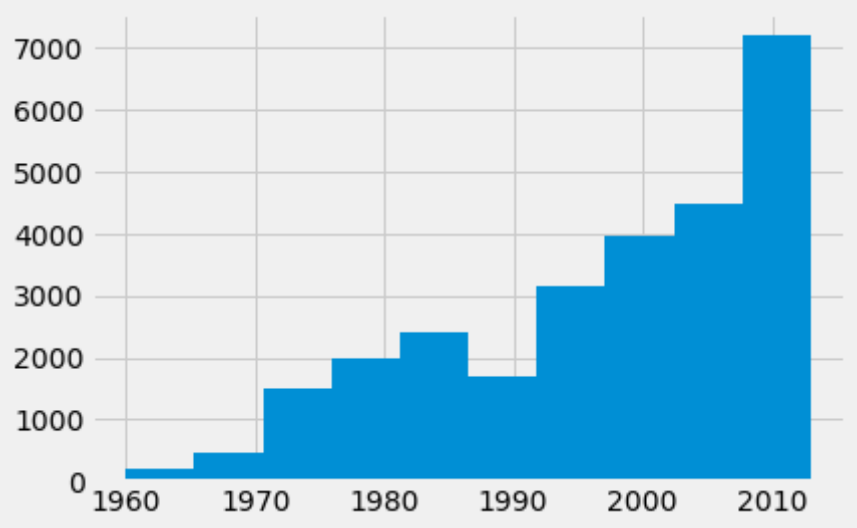
In [707]:
```
#convert original 0's to nulls in training set
cat_conversion_list = ['installer','funder']
for col in cat_conversion_list:
    X_train_tf[col].replace('0', np.nan, inplace=True)

#convert original 0's to nulls in test set
for col in cat_conversion_list:
    X_test_tf[col].replace('0', np.nan, inplace=True)
```

executed in 6ms, finished 20:49:58 2021-05-23

5.6.1 Impute `construction_year`

In [708]:
```
#current distribution of construction_year
X_train_tf['construction_year'].hist()
```
executed in 103ms, finished 20:49:58 2021-05-23

Out[708]: <AxesSubplot:>



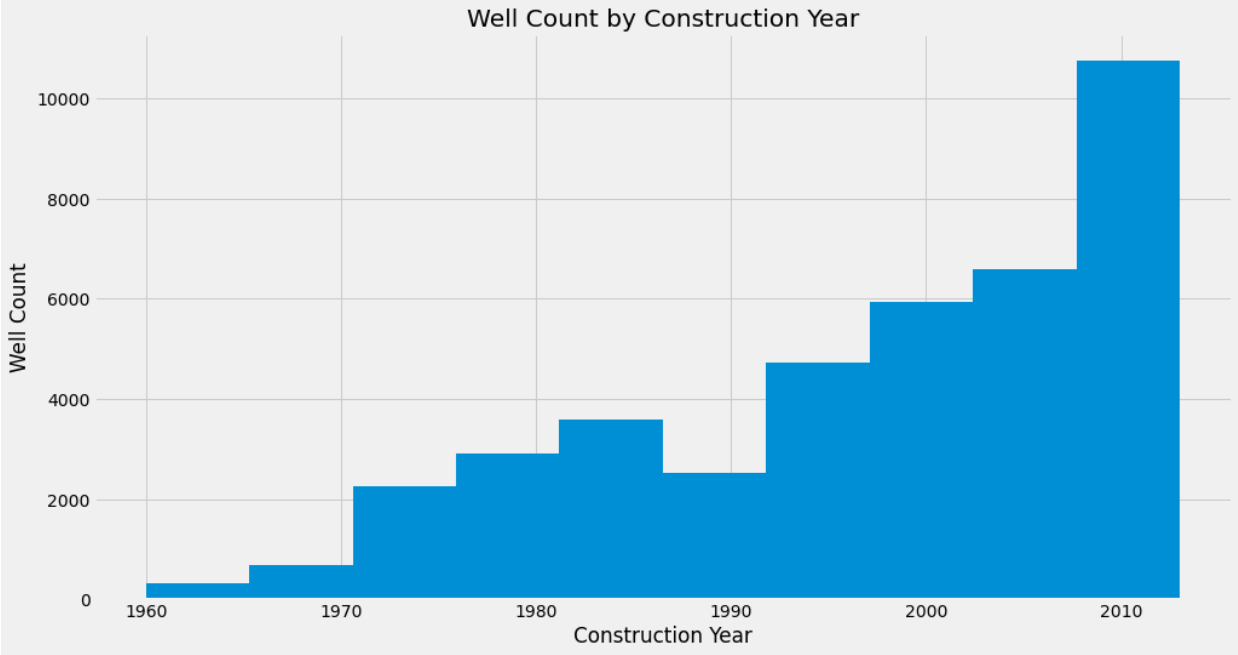
In [709]:
```
#slice out construction_year
impute_num = ['construction_year']
```
executed in 2ms, finished 20:49:58 2021-05-23

In [710]:
```
#random sample imputation
rs_imputer = mdi.RandomSampleImputer(random_state = 123)
X_train_tf[impute_num] = rs_imputer.fit_transform(X_train_tf[impute_num
X_test_tf[impute_num] = rs_imputer.transform(X_test_tf[impute_num])
```
executed in 9ms, finished 20:49:58 2021-05-23

In [711]:
```
#review construction_year after imputation
fig, ax = plt.subplots(figsize=(15,8))
X_train_tf['construction_year'].hist(ax=ax)
ax.set_title("Well Count by Construction Year")
ax.set_xlabel("Construction Year")
ax.set_ylabel("Well Count")
```

executed in 99ms, finished 20:49:59 2021-05-23

Out[711]:
```
Text(0, 0.5, 'Well Count')
```



In [712]:
```
1 #check for missing values
2 X_train_tf.isna().sum()
```

executed in 31ms, finished 20:49:59 2021-05-23

Out[712]:
```
id                     0
wpt_name               0
construction_year      0
waterpoint_type        0
water_quality          0
water_quantity         0
head                   0
source_type_1          0
source_type_2          0
extraction_type_1      0
extraction_type_2      0
well_elevation         0
population             0
latitude               0
longitude              0
basin                  0
district_code          0
region                 0
lga                    0
ward                   0
funder              3076
installer           3093
permit              2140
public_meeting      2100
scheme_management   2642
management             0
water_cost             0
date_recorded_yr       0
dtype: int64
```

In [713]:

```
#see how many columns have values equal to 0
for col in X_train_tf:
    count_num = (X_train_tf[col] == 0).sum()
    count_cat = (X_train_tf[col] == '0').sum()
    print(col,count_num, count_cat)

for col in X_test_tf:
    count_num = (X_test_tf[col] == 0).sum()
    count_cat = (X_test_tf[col] == '0').sum()
    print(col,count_num,'\n', count_cat)
```

executed in 148ms, finished 20:49:59 2021-05-23

```
id 0 0
wpt_name 0 0
construction_year 0 0
waterpoint_type 0 0
water_quality 0 0
water_quantity 0 0
head 27848 0
source_type_1 0 0
source_type_2 0 0
extraction_type_1 0 0
extraction_type_2 0 0
well_elevation 13064 0
population 13738 0
latitude 0 0
longitude 0 0
basin 0 0
district_code 15 0
region 0 0
lga 0 0
ward 0 0
funder 0 0
installer 0 0
permit 0 0
public_meeting 0 0
scheme_management 0 0
management 0 0
water_cost 0 0
date_recorded_yr 0 0
id 1
 0
wpt_name 0
 0
construction_year 0
 0
waterpoint_type 0
 0
water_quality 0
 0
water_quantity 0
 0
head 11974
 0
source_type_1 0
 0
source_type_2 0
```

```
 0
extraction_type_1 0
 0
extraction_type_2 0
 0
well_elevation 5562
 0
population 5831
 0
latitude 0
 0
longitude 0
 0
basin 0
 0
district_code 8
 0
region 0
 0
lga 0
 0
ward 0
 0
funder 0
 0
installer 0
 0
permit 0
 0
public_meeting 0
 0
scheme_management 0
 0
management 0
 0
water_cost 0
 0
date_recorded_yr 0
 0
```

5.6.2 Impute Categorical Variables

In [714]:
```
#set columns to impuate
impute_cat = ['funder', 'installer', 'permit', 'public_meeting', 'schem
```

executed in 2ms, finished 20:49:59 2021-05-23

In [715]:
```
#categorical imputer with "Unknown"
unknown_imputer = mdi.CategoricalImputer(fill_value='Unknown')
X_train_tf[impute_cat] = unknown_imputer.fit_transform(X_train_tf[imput
X_test_tf[impute_cat] = unknown_imputer.transform(X_test_tf[impute_cat]
```

executed in 30ms, finished 20:49:59 2021-05-23

In [716]:
```
#check for missing values
X_train_tf[impute_cat].isna().sum()
```

executed in 12ms, finished 20:49:59 2021-05-23

Out[716]:
```
funder               0
installer            0
permit               0
public_meeting       0
scheme_management    0
dtype: int64
```

5.7 Feature Engineering

In this section I will develop new features which I believe will improve the ability to gain insights into the data and possibly help modeling.

5.7.1 `well_age`

`well_age` will allow insight into how age of the well impacts operability. This feature will be created by taking the max of `recorded_date` and subtracting the `construction_year`.

In [717]:
```
1 X_train_tf['construction_year'].value_counts()
```

executed in 4ms, finished 20:49:59 2021-05-23

Out[717]:

2,008.0	2729
2,010.0	2710
2,009.0	2642
2,000.0	2195
2,007.0	1647
2,006.0	1422
2,003.0	1336
2,011.0	1328
2,004.0	1173
2,012.0	1166
2,002.0	1137
1,985.0	1061
1,995.0	1060
1,978.0	1052
1,998.0	1040
1,999.0	1036
2,005.0	1018
1,990.0	991
1,996.0	875
1,984.0	819
1,980.0	797
1,994.0	776
1,982.0	759
1,972.0	740
1,997.0	706
1,974.0	692
1,992.0	692
1,993.0	627
1,988.0	555
2,001.0	538
1,983.0	513
1,975.0	501
1,986.0	436
1,976.0	417
1,970.0	405
1,991.0	337
1,989.0	330
1,987.0	312
1,981.0	236
1,977.0	229
1,979.0	184
2,013.0	178
1,973.0	162
1,971.0	159
1,960.0	105
1,967.0	98
1,968.0	97
1,963.0	91
1,969.0	63
1,964.0	35
1,962.0	32
1,961.0	25
1,965.0	22

```
1,966.0      19
Name: construction_year, dtype: int64
```

In [718]:
```
#create well_age feature
max_date_recorded = df_clean['date_recorded_yr'].max()

X_train_tf['well_age'] = max_date_recorded - X_train_tf['construction_y
X_test_tf['well_age'] = max_date_recorded - X_test_tf['construction_yea
X_train_tf['well_age'].value_counts()
```

executed in 7ms, finished 20:49:59 2021-05-23

Out[718]:
```
5.0     2729
3.0     2710
4.0     2642
13.0    2195
6.0     1647
7.0     1422
10.0    1336
2.0     1328
9.0     1173
1.0     1166
11.0    1137
28.0    1061
18.0    1060
35.0    1052
15.0    1040
14.0    1036
8.0     1018
23.0     991
17.0     875
29.0     819
33.0     797
19.0     776
31.0     759
41.0     740
16.0     706
39.0     692
21.0     692
20.0     627
25.0     555
12.0     538
30.0     513
38.0     501
27.0     436
37.0     417
43.0     405
22.0     337
24.0     330
26.0     312
32.0     236
36.0     229
34.0     184
0.0      178
40.0     162
42.0     159
53.0     105
46.0      98
45.0      97
50.0      91
44.0      63
```

```
49.0      35
51.0      32
52.0      25
48.0      22
47.0      19
Name: well_age, dtype: int64
```

In [719]:
```
#drop date_recorded_yr feature
X_train_tf.drop(columns='date_recorded_yr', inplace=True)
X_test_tf.drop(columns='date_recorded_yr', inplace=True)
```

executed in 16ms, finished 20:49:59 2021-05-23

5.7.2 impact_rating [...]

6 DATA EXPLORATION

In this section I will explore the clean data to try and examine insights that will help my stakeholder understand the reliability of the water wells.

In [720]:
```
#create data exploration df
df_explore = pd.concat([X_train_tf, X_test_tf], axis=0)
df_explore['status_group'] = pd.concat([y_train, y_test])
```

executed in 23ms, finished 20:49:59 2021-05-23

In [721]:
```
#make a copy for linear assumptions checks
df_explore_binary = df_explore.copy()
```

executed in 16ms, finished 20:49:59 2021-05-23

In [722]:
```
#convert status group back to words for EDA
df_explore['status_group'] = df_explore['status_group'].map({0:'functio
```

executed in 5ms, finished 20:49:59 2021-05-23

In [723]:
```
#explore the dataframe
df_explore
```
executed in 21ms, finished 20:49:59 2021-05-23

Out[723]:

	id	wpt_name	construction_year	waterpoint_type	water_quality	water_quantity	
16350	2659	Shuleni	1,982.0	communal standpipe multiple	soft-good	enough	1,
19439	1593	Daniel Kituma Mahewa	2,012.0	communal standpipe	salty	insufficient	
44958	2548	none	2,000.0	communal standpipe	soft-good	enough	3,
55544	25039	Kwa Luduka	2,000.0	other	soft-good	dry	
34351	11325	Tahani	2,006.0	communal standpipe	soft-good	enough	
...	...	...	...	...	...	...	
52171	63328	Saiti	1,975.0	other	salty	dry	
18684	3266	Sekondari	2,002.0	communal standpipe	soft-good	insufficient	
9988	26740	Teresia Isack[Kong'Eng'I]	2,009.0	communal standpipe	soft-good	enough	
38372	34973	Shule Ya Sekondari Ndono	2,002.0	communal standpipe	soft-good	seasonal	
27689	6388	Kwa Mohamed Mdee	2,008.0	communal standpipe	soft-good	enough	

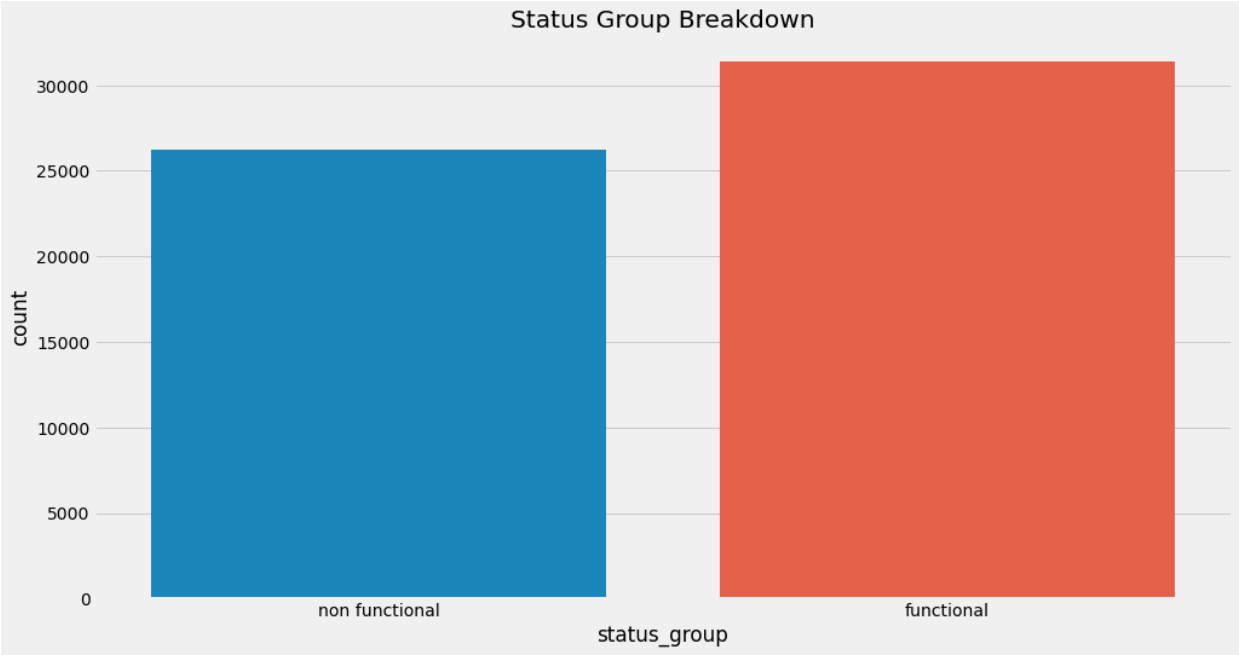
57579 rows × 29 columns

In [724]:
```
#export original dataframe to Tableau
# df_explore.to_csv(r'Data/water_well_data_for_tableau_imputed.csv', in
```
executed in 2ms, finished 20:49:59 2021-05-23

6.1 Functional vs Not Functional

In [725]:
```
#create plot
fig, ax = plt.subplots(figsize=(15,8))
ax = sns.countplot(data=df_explore, x='status_group');
ax.set_title('Status Group Breakdown');
```

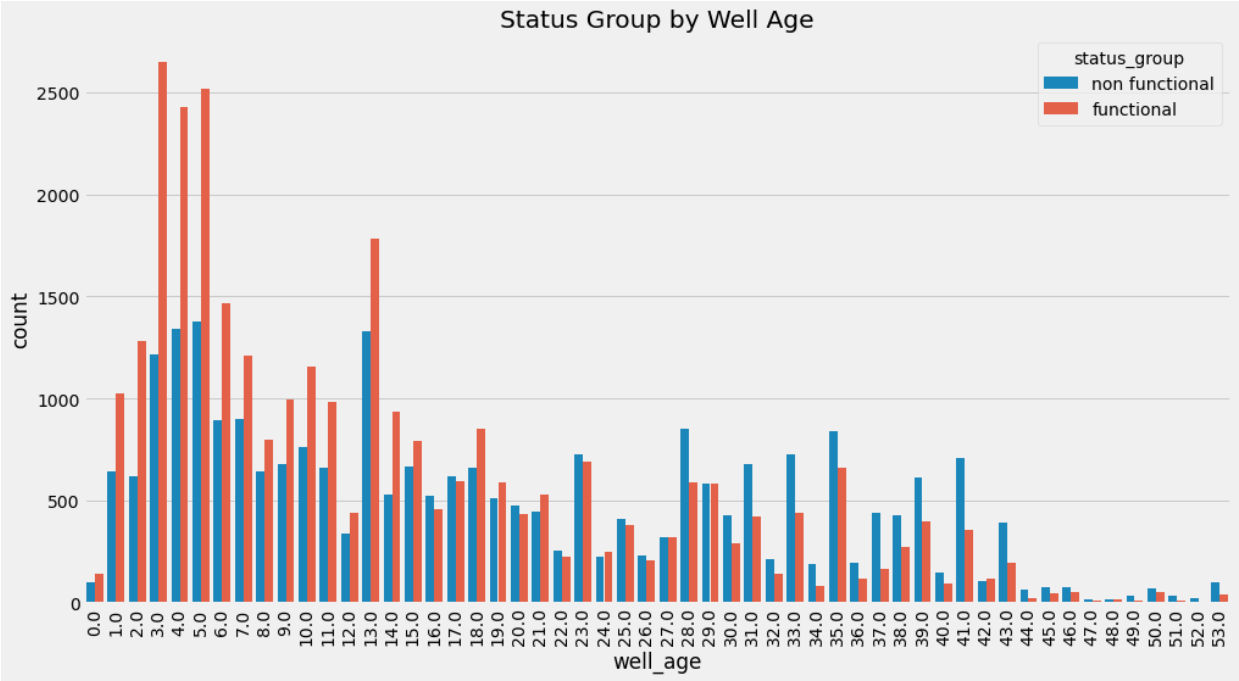
executed in 112ms, finished 20:49:59 2021-05-23



6.2 Number of Functional/Not Functional Water Wells by Well Age

In [726]:
```
#create a barplot
fig, ax = plt.subplots(figsize=(15,8))
ax = sns.countplot(data=df_explore, x='well_age', hue='status_group');
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
ax.set_title('Status Group by Well Age');
```

executed in 735ms, finished 20:50:00 2021-05-23

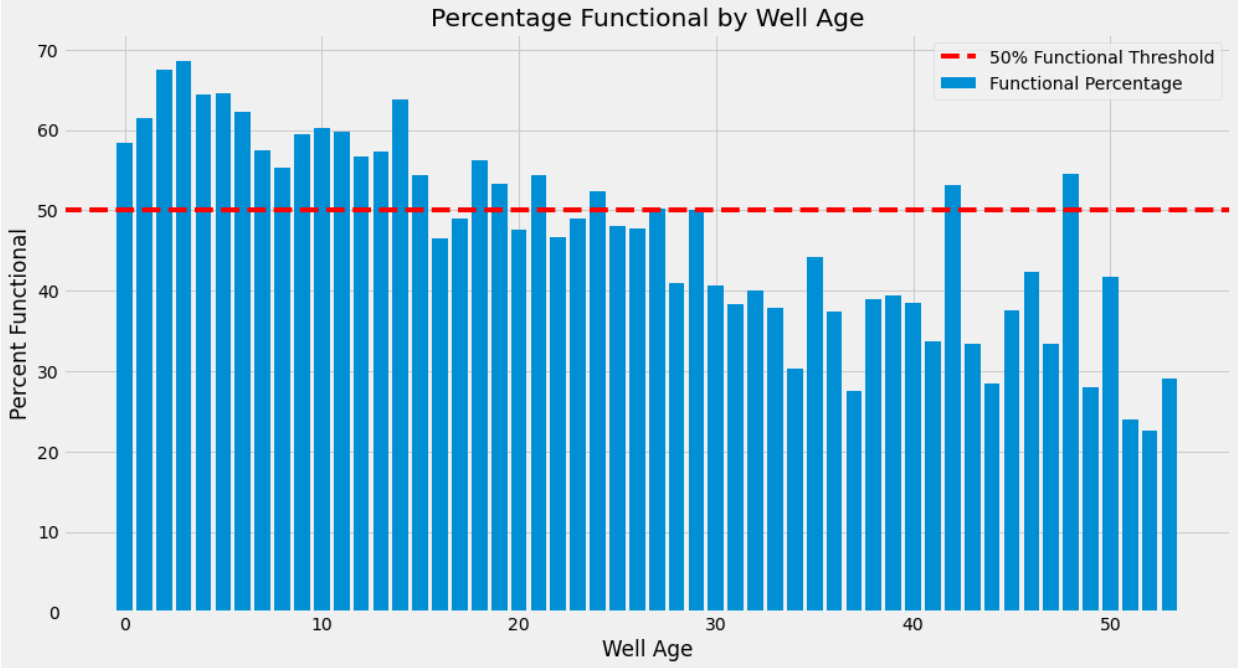


6.3 What age do wells start to not function more than they function?

In [727]:

```
fig, ax = plt.subplots(figsize=(15,8))
status_group_ratio_df = pd.DataFrame(df_explore.groupby(by=['well_age',
percent_df = status_group_ratio_df.groupby(level=0).apply(lambda x: x /
percent_df.reset_index(inplace=True)
functional_percent_df = percent_df.loc[(percent_df['status_group'] == '
functional_percent_df
plt.bar(data=functional_percent_df, x='well_age',height='id', label='Fu
ax.axhline(y=50, color='red', linestyle='--',
                        label=f'50% Functional Threshold');
ax.set_title('Percentage Functional by Well Age')
ax.set_xlabel('Well Age')
ax.set_ylabel('Percent Functional')
plt.legend();
```

executed in 211ms, finished 20:50:00 2021-05-23



OBSERVATIONS

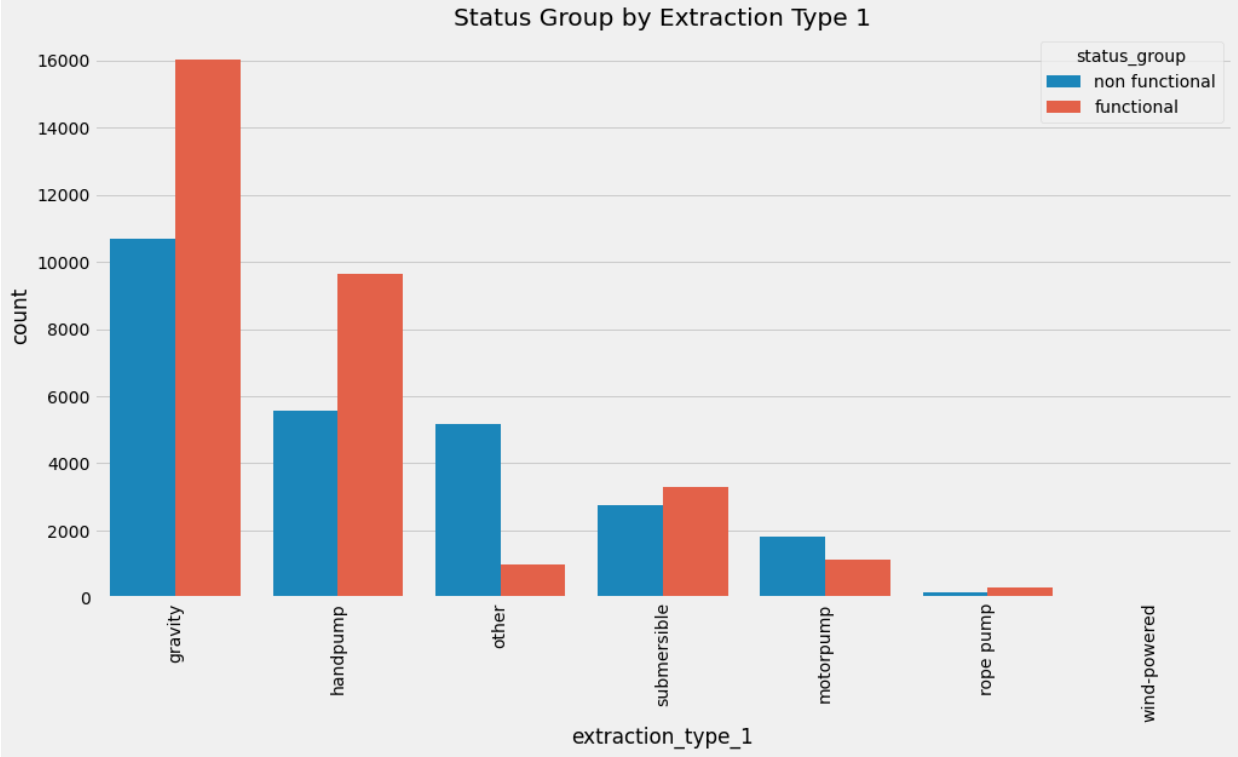
- The ratio of functional to non-functional wells consistently goes below 50% around a well age of 25 years.

Type *Markdown* and LaTeX: α^2

6.4 Which technologies are most reliable?

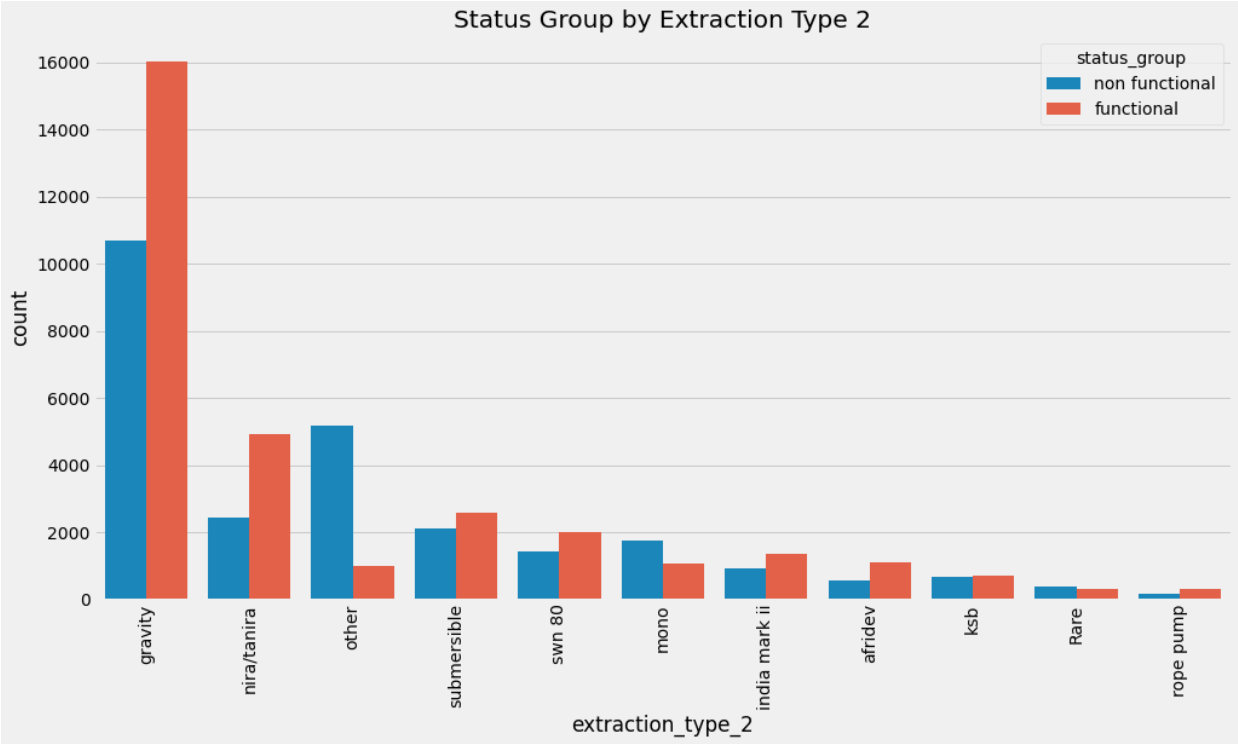
In [728]:
```
#create a barplot
fig, ax = plt.subplots(figsize=(15,8))
ax = sns.countplot(data=df_explore, x='extraction_type_1', hue='status_
                   order=df_explore['extraction_type_1'].value_counts()
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
ax.set_title('Status Group by Extraction Type 1');
```

executed in 182ms, finished 20:50:00 2021-05-23



In [729]:
```
#create a barplot
fig, ax = plt.subplots(figsize=(15,8))
ax = sns.countplot(data=df_explore, x='extraction_type_2', hue='status_
                   order=df_explore['extraction_type_2'].value_counts().
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
ax.set_title('Status Group by Extraction Type 2');
```

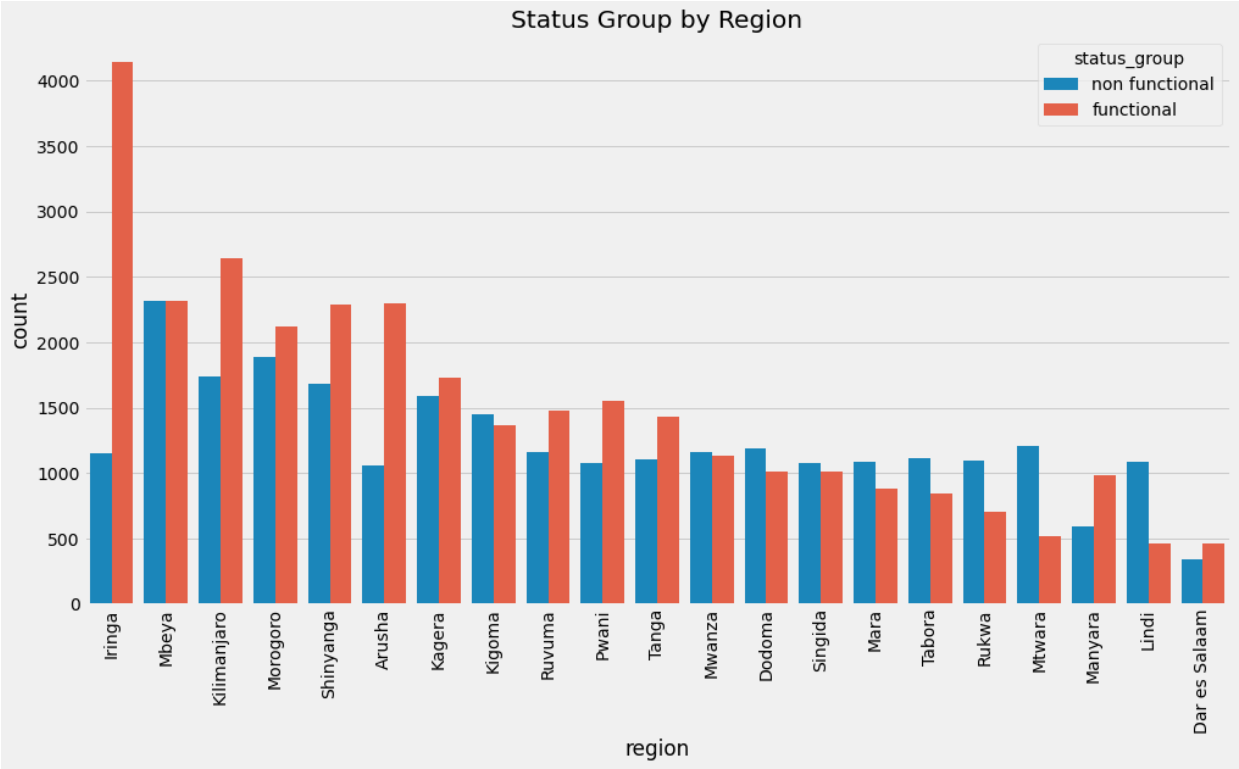
executed in 200ms, finished 20:50:00 2021-05-23



6.5 What regions have the most failures

In [730]:
```
#create a barplot
fig, ax = plt.subplots(figsize=(15,8))
ax = sns.countplot(data=df_explore, x='region', hue='status_group',
                   order=df_explore['region'].value_counts().index);
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
ax.set_title('Status Group by Region');
```

executed in 246ms, finished 20:50:01 2021-05-23

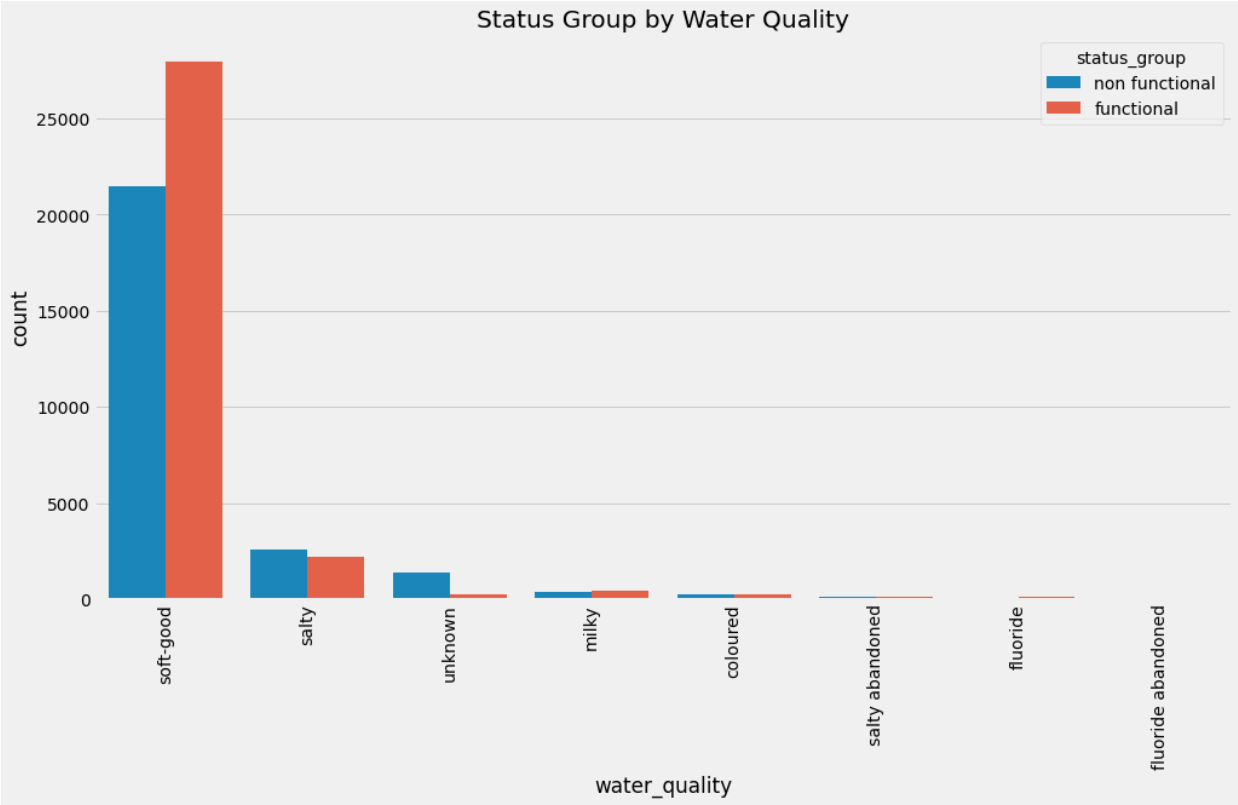


6.6 How does water quality affect reliability?

In [731]:

```
#create a barplot
fig, ax = plt.subplots(figsize=(15,8))
ax = sns.countplot(data=df_explore, x='water_quality', hue='status_grou
                   order=df_explore['water_quality'].value_counts().inde
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
ax.set_title('Status Group by Water Quality');
```

executed in 179ms, finished 20:50:01 2021-05-23



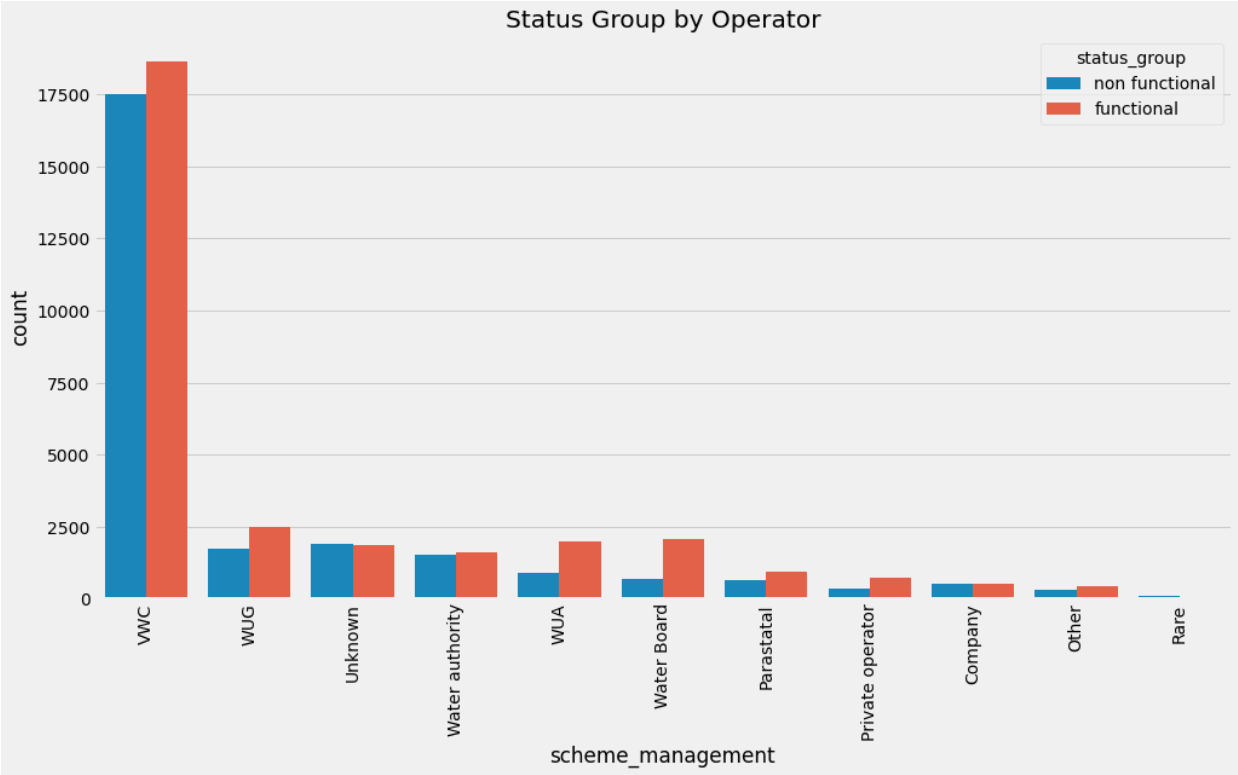
OBSERVATIONS

- It seems that reliability goes down as your water quality degrades beyond 'soft-good'. Salty water specifically is highly corrosive and will lead to the corrosion of any metal components.

6.7 How does the operator affect reliability?

In [732]:
```
#create a barplot
fig, ax = plt.subplots(figsize=(15,8))
ax = sns.countplot(data=df_explore, x='scheme_management', hue='status_
                   order=df_explore['scheme_management'].value_counts().
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
ax.set_title('Status Group by Operator');
```

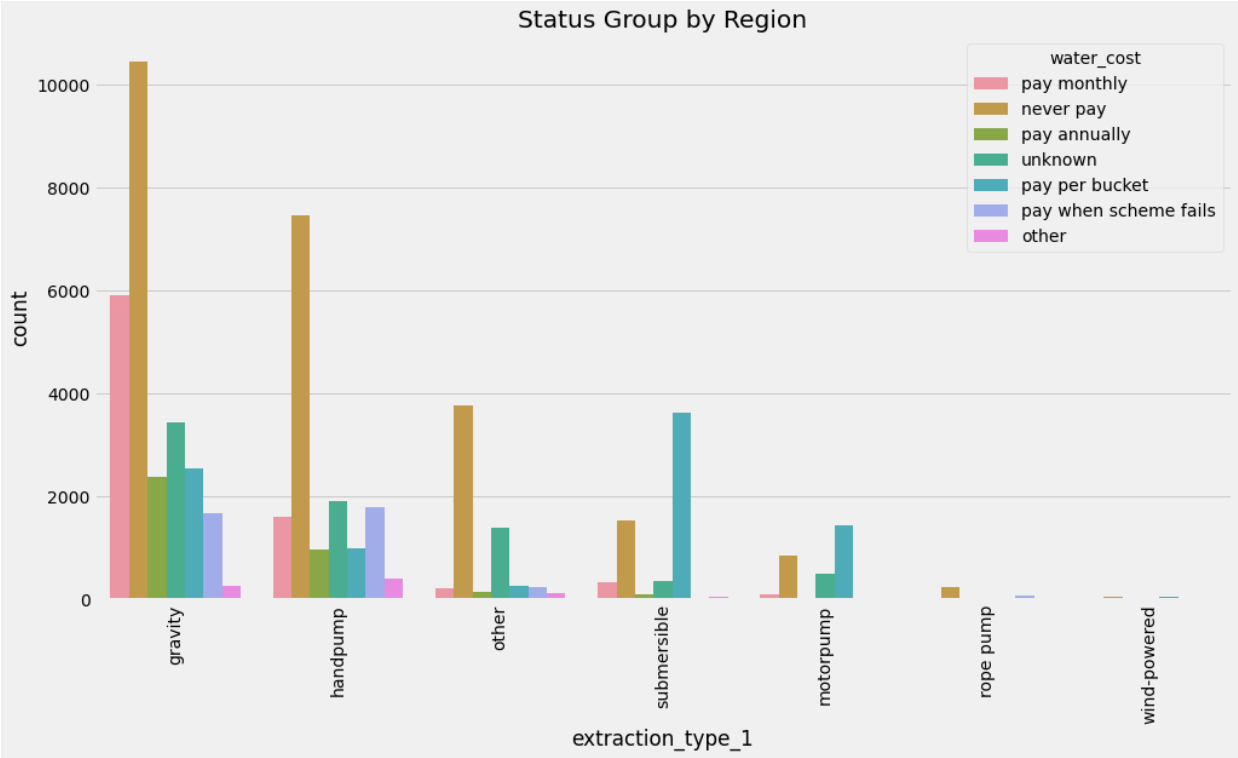
executed in 193ms, finished 20:50:01 2021-05-23



6.8 Which Technologies cost the most

In [734]:
```
#create a barplot
fig, ax = plt.subplots(figsize=(15,8))
ax = sns.countplot(data=df_explore, x='extraction_type_1', hue='water_c
                   order=df_explore['extraction_type_1'].value_counts().
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
ax.set_title('Status Group by Region');
```

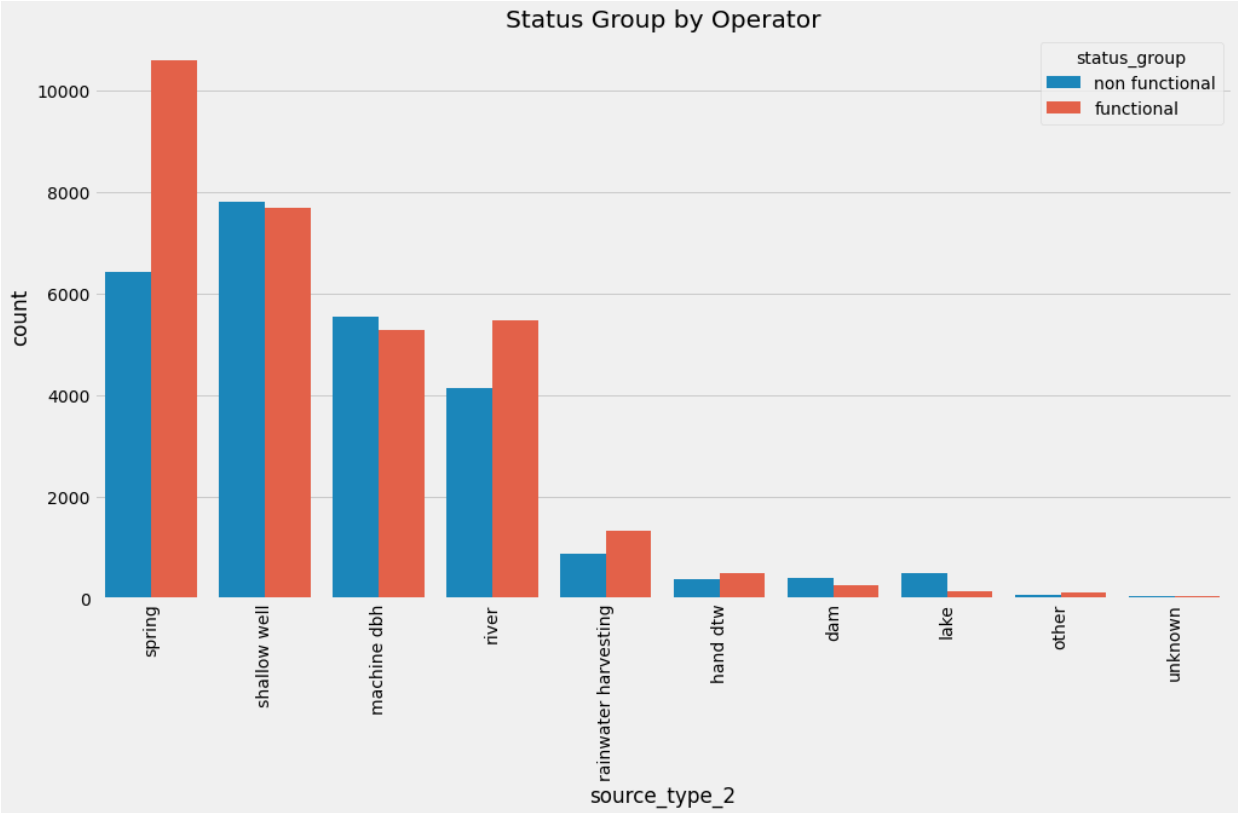
executed in 251ms, finished 20:50:01 2021-05-23



6.9 How does water source affect reliability?

In [735]:
```
#create a barplot
fig, ax = plt.subplots(figsize=(15,8))
ax = sns.countplot(data=df_explore, x='source_type_2', hue='status_grou
                   order=df_explore['source_type_2'].value_counts().inde
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
ax.set_title('Status Group by Operator');
```

executed in 188ms, finished 20:50:01 2021-05-23

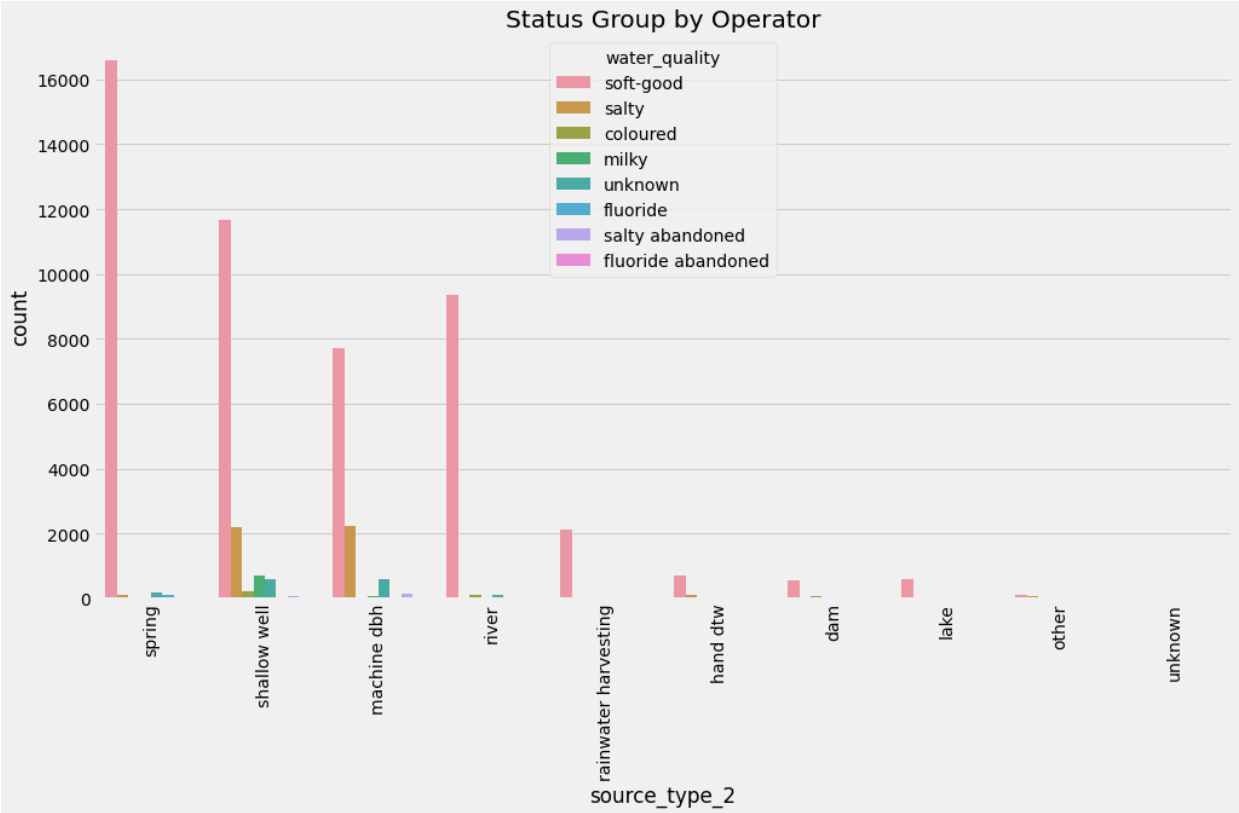


6.10 How does water source affect water quality

In [736]:

```
#create a barplot
fig, ax = plt.subplots(figsize=(15,8))
ax = sns.countplot(data=df_explore, x='source_type_2', hue='water_quali
                   order=df_explore['source_type_2'].value_counts().inde
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
ax.set_title('Status Group by Operator');
```

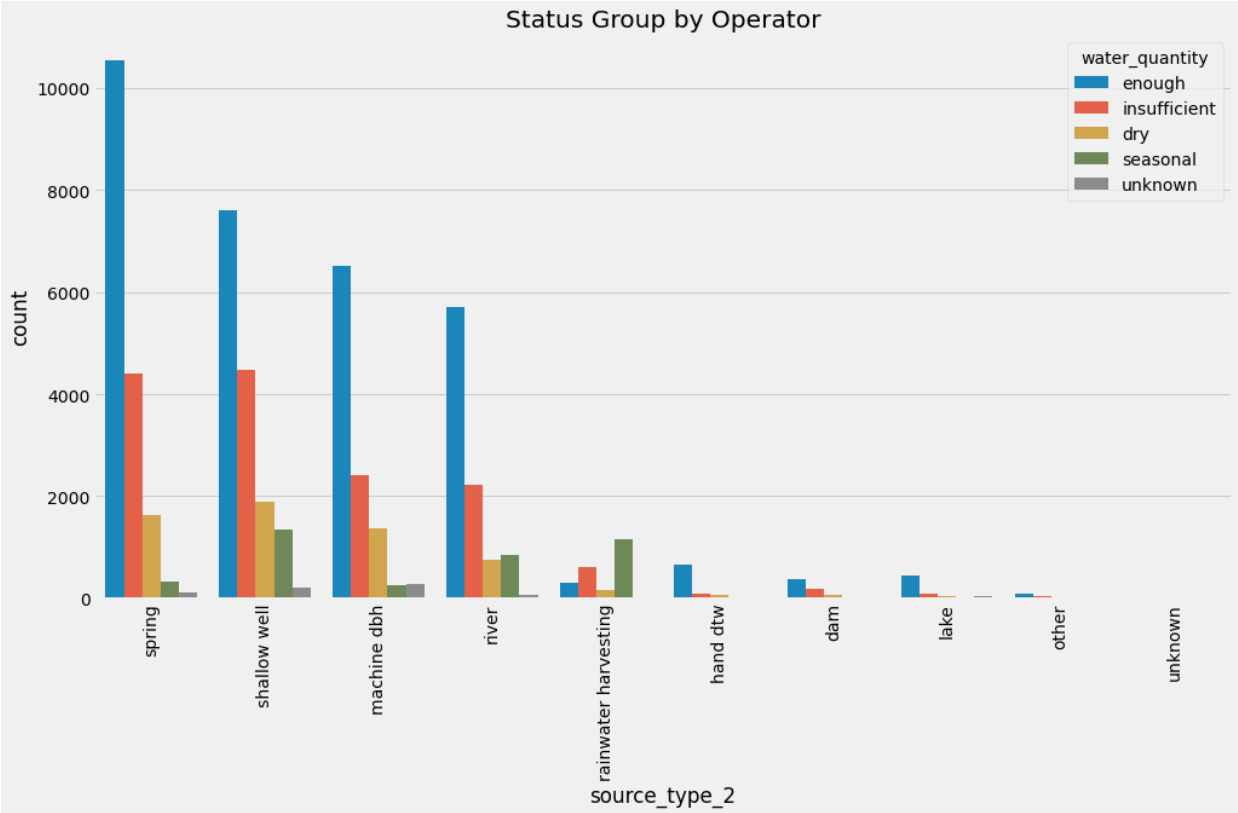
executed in 322ms, finished 20:50:02 2021-05-23



6.11 How does water source affect water quantity

In [737]:
```
#create a barplot
fig, ax = plt.subplots(figsize=(15,8))
ax = sns.countplot(data=df_explore, x='source_type_2', hue='water_quant
                   order=df_explore['source_type_2'].value_counts().inde
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
ax.set_title('Status Group by Operator');
```

executed in 247ms, finished 20:50:02 2021-05-23



6.12 Size of population left without water (around a non-functional well)

In [738]:
```
df_explore.loc[df_explore['status_group'] == 'non functional',['populat
```

executed in 13ms, finished 20:50:02 2021-05-23

Out[738]:
```
population    4634437
dtype: int64
```

7 DATA MODELING

As stated before, I will be modeling whether or not a water well is functional or not functional based on characteristics of the well. Since it is more important to identify the water wells that are not functional as opposed to functional, I will be tuning a model to ensure that the it correctly identifies non functional wells even if it means that the model predicts a false positive. Although this will mean more funding and time will be allocated to water well maintenance, it will ensure that the water wells are as reliable as they can be, and subsequently, that the Tanzanian citizens have a reliable water supply.

7.1 Model Preprocessing

I will take 2 major steps in preprocessing my data to prepare for modeling:

1. Scale numerical data
2. Encode categorical data

In [739]:
```
#training columns
X_train_tf.columns
```
executed in 3ms, finished 20:50:02 2021-05-23

Out[739]:
```
Index(['id', 'wpt_name', 'construction_year', 'waterpoint_type',
       'water_quality', 'water_quantity', 'head', 'source_type_1',
       'source_type_2', 'extraction_type_1', 'extraction_type_2',
       'well_elevation', 'population', 'latitude', 'longitude', 'basin',
       'district_code', 'region', 'lga', 'ward', 'funder', 'installer',
       'permit', 'public_meeting', 'scheme_management', 'management',
       'water_cost', 'well_age'],
      dtype='object')
```

In [740]:
```
#remove features
X_train_tf.drop(columns=['id','wpt_name'],
                inplace=True)
X_test_tf.drop(columns=['id','wpt_name'],
               inplace=True)
```
executed in 12ms, finished 20:50:02 2021-05-23

In [741]:
```
#create dataframe for decision tree models
X_train_dt = X_train_tf.copy()
X_test_dt = X_test_tf.copy()
```
executed in 4ms, finished 20:50:02 2021-05-23

In [742]:
```
#separate out categorical and numerical data
cat_cols = X_train_tf.select_dtypes(include='object').columns
num_cols = X_train_tf.select_dtypes(exclude='object').columns
cat_cols
```

executed in 12ms, finished 20:50:02 2021-05-23

Out[742]:
```
Index(['waterpoint_type', 'water_quality', 'water_quantity', 'source_type
_1',
       'source_type_2', 'extraction_type_1', 'extraction_type_2', 'basi
n',
       'district_code', 'region', 'lga', 'ward', 'funder', 'installer',
       'permit', 'public_meeting', 'scheme_management', 'management',
       'water_cost'],
      dtype='object')
```

In [743]:
```
#one hot encode categorical columns
ohe = OneHotEncoder(sparse=False, drop='first')
ohe.fit(X_train_tf[cat_cols])

train_ohe_df = pd.DataFrame(ohe.transform(X_train_tf[cat_cols]),
                            columns=ohe.get_feature_names(cat_cols),
                            index=X_train_tf.index)

test_ohe_df = pd.DataFrame(ohe.transform(X_test_tf[cat_cols]),
                            columns=ohe.get_feature_names(cat_cols),
                            index=X_test_tf.index)

train_ohe_df
```

executed in 363ms, finished 20:50:02 2021-05-23

Out[743]:

	waterpoint_type_communal standpipe	waterpoint_type_communal standpipe multiple	waterpoint_type_dam	waterpoint_type
16350	0.0	1.0	0.0	
19439	1.0	0.0	0.0	
44958	1.0	0.0	0.0	
55544	0.0	0.0	0.0	
34351	1.0	0.0	0.0	
...	...	...	...	
34029	0.0	0.0	0.0	
6523	0.0	0.0	0.0	
33122	0.0	0.0	0.0	
26755	1.0	0.0	0.0	
4054	0.0	1.0	0.0	

40305 rows × 461 columns

In [744]:

```
#scale numeric data
scaler = StandardScaler()
scaler.fit(X_train_tf[num_cols])

train_scale_df = pd.DataFrame(scaler.transform(X_train_tf[num_cols]),
                              columns=num_cols, index=X_train_tf.index)

test_scale_df = pd.DataFrame(scaler.transform(X_test_tf[num_cols]),
                             columns=num_cols, index=X_test_tf.index)

train_scale_df
```

executed in 16ms, finished 20:50:02 2021-05-23

Out[744]:

	construction_year	head	well_elevation	population	
16350	-1.1884272448163955	0.2009153779868061	0.20926304070087273	0.13756573010598272	-1
19439	1.2203490337488778	-0.10215745328534168	1.0324582255785268	-0.09487626185688926	1
44958	0.25683852232276855	0.8070610405311016	1.982520619333998	-0.16672196846359513	-
55544	0.25683852232276855	-0.10215745328534168	-0.9902087523257616	-0.39071152435509	
34351	0.7385937780358233	-0.10215745328534168	0.8983826525599247	-0.13713844221377508	0
...	...	...	...	...	
34029	-1.99135267100482	-0.10215745328534168	1.1362586692058316	2.356330198842488	0
6523	0.7385937780358233	-0.10215745328534168	1.048316626688254	0.560187533674841	1
33122	0.25683852232276855	-0.10215745328534168	-0.9902087523257616	-0.39071152435509	-1
26755	-1.0278421595787106	-0.04154288703091212	-0.2030553881520328	0.24322118099819728	-0
4054	0.9794714058923506	-0.10215745328534168	-0.9902087523257616	-0.39071152435509	-0.

40305 rows × 7 columns

In [745]:
```
1 #review scaling of numeric features
2 train_scale_df.describe()
```
executed in 23ms, finished 20:50:02 2021-05-23

Out[745]:

	construction_year	head	well_elevation	population	
count	40,305.0	40,305.0	40,305.0	40,305.0	
mean	-7.775863785859447e-15	-1.8158020539211094e-17	-2.327047292403752e-17	-1.7981729077665356e-17	-:
std	1.0000124056396043	1.0000124056396043	1.0000124056396043	1.0000124056396043	
min	-2.954863182430929	-0.10215745328534168	-1.1199593068598928	-0.39071152435509	
25%	-0.7066719891033408	-0.10215745328534168	-0.9902087523257616	-0.39071152435509	
50%	0.25683852232276855	-0.10215745328534168	-0.3890311829842874	-0.3167527087305398	(
75%	0.8991788632735082	-0.09306526834717724	0.9272161091230648	0.09530354974909691	
max	1.3006415763677204	105.97333349196637	3.0032249816691627	64.05911351989577	

In [746]:
```
1 #concat num and cat
2 X_train_tf = pd.concat([train_ohe_df,train_scale_df],axis=1)
3 X_test_tf = pd.concat([test_ohe_df,test_scale_df],axis=1)
```
executed in 141ms, finished 20:50:03 2021-05-23

In [747]:
```
1 #review dataframe
2 X_train_tf
```
executed in 28ms, finished 20:50:03 2021-05-23

Out[747]:

	waterpoint_type_communal standpipe	waterpoint_type_communal standpipe multiple	waterpoint_type_dam	waterpoint_type
16350	0.0	1.0	0.0	
19439	1.0	0.0	0.0	
44958	1.0	0.0	0.0	
55544	0.0	0.0	0.0	
34351	1.0	0.0	0.0	
...	...	...	...	
34029	0.0	0.0	0.0	
6523	0.0	0.0	0.0	
33122	0.0	0.0	0.0	
26755	1.0	0.0	0.0	
4054	0.0	1.0	0.0	

40305 rows × 468 columns

7.2 Dummy Classifier

The dummy model will predict the exact same distribution for the target variable, `status_group`, as is in the data. This model will be equivalent to randomly guessing and will be a good baseline to evaluate other models against.

7.2.1 Model Creation

In [748]:
```
#current class values
y_train.value_counts(normalize=True)
```

executed in 4ms, finished 20:50:03 2021-05-23

Out[748]:
```
0    0.5452673365587396
1    0.4547326634412604
Name: status_group, dtype: float64
```

In [749]:
```
#create dummy classifier model as a baseline
dummy = DummyClassifier()
#fit the dummy model
dummy.fit(X_train_tf,y_train)
```

executed in 4ms, finished 20:50:03 2021-05-23

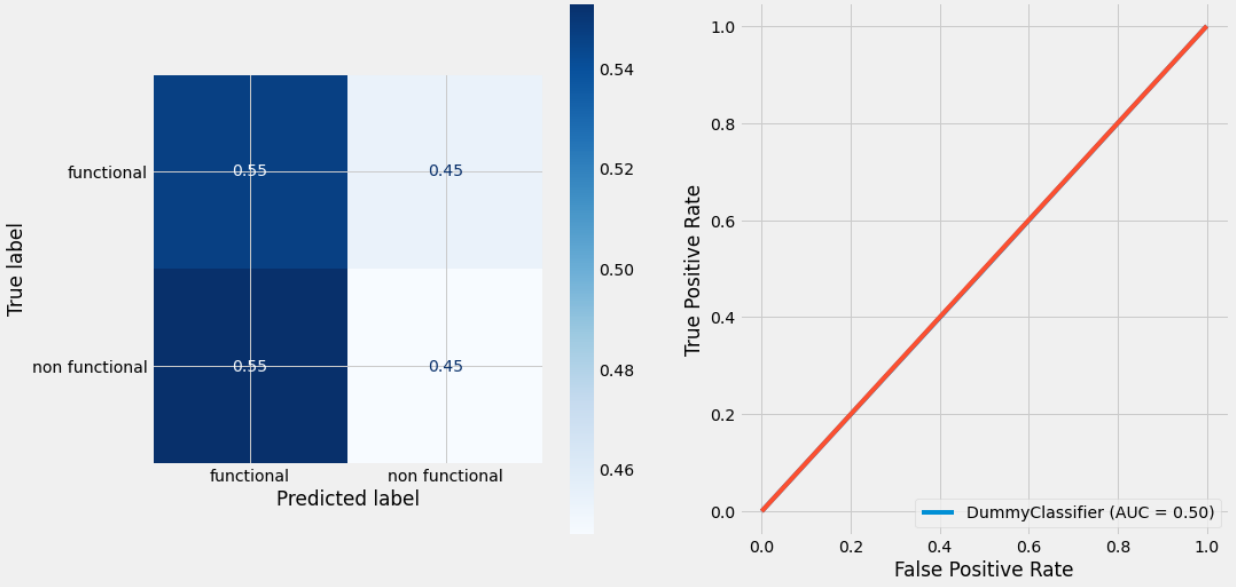
Out[749]:
```
DummyClassifier()
```

7.2.2 Model Evaluation

In [750]:
```
#eval model
model_eval(dummy, X_train_tf, y_train, X_test_tf, y_test)
```

executed in 242ms, finished 20:50:03 2021-05-23

```
CURRENT MODEL: Not Overfit (Recall)
================================================================


Classification Reports----------------------------------------
              precision    recall  f1-score   support

           0       0.55      0.55      0.55      9408
           1       0.46      0.46      0.46      7866

    accuracy                           0.50     17274
   macro avg       0.50      0.50      0.50     17274
weighted avg       0.50      0.50      0.50     17274

Test Graphs---------------------------------------------------
```



7.3 Feature Selection (Model Agnostic-Filter Methods)

In this section I will be removing features based solely on the characteristics of the data and therefore, will be model agnostic. Removing features using filter methods eliminates features which are constant, almost constant or duplicated and will not add value to any model. Also, the lower the number of features, the less cost in computational time and hardware requirements.

In [751]:
```
#remove constant features
sel_const = DropConstantFeatures(tol=1, missing_values='raise')
#fit the model
sel_const.fit(X_train_tf)
```

executed in 487ms, finished 20:50:03 2021-05-23

Out[751]:
```
DropConstantFeatures(variables=['waterpoint_type_communal standpipe',
                                'waterpoint_type_communal standpipe multi
ple',
                                'waterpoint_type_dam',
                                'waterpoint_type_hand pump',
                                'waterpoint_type_improved spring',
                                'waterpoint_type_other',
                                'water_quality_fluoride',
                                'water_quality_fluoride abandoned',
                                'water_quality_milky', 'water_quality_sal
ty',
                                'water_quality_salty abandoned...
                                'water_quantity_unknown',
                                'source_type_1_surface',
                                'source_type_1_unknown',
                                'source_type_2_hand dtw', 'source_type_2_
lake',
                                'source_type_2_machine dbh',
                                'source_type_2_other',
                                'source_type_2_rainwater harvesting',
                                'source_type_2_river',
                                'source_type_2_shallow well',
                                'source_type_2_spring', 'source_type_2_un
known',
                                'extraction_type_1_handpump',
                                'extraction_type_1_motorpump', ...])
```

In [752]:
```
#number of constant features
len(sel_const.features_to_drop_)
```

executed in 3ms, finished 20:50:03 2021-05-23

Out[752]:
```
0
```

OBSERVATIONS

- No constant features to remove

In [753]:
```
#remove quasi-constant features
quasi_const = DropConstantFeatures(tol=0.998, missing_values='raise')
#fit the model
quasi_const.fit(X_train_tf)
```

executed in 598ms, finished 20:50:04 2021-05-23

Out[753]:
```
DropConstantFeatures(tol=0.998,
                     variables=['waterpoint_type_communal standpipe',
                                'waterpoint_type_communal standpipe multi
ple',
                                'waterpoint_type_dam',
                                'waterpoint_type_hand pump',
                                'waterpoint_type_improved spring',
                                'waterpoint_type_other',
                                'water_quality_fluoride',
                                'water_quality_fluoride abandoned',
                                'water_quality_milky', 'water_quality_sal
ty',
                                'water_quality_salt...
                                'water_quantity_unknown',
                                'source_type_1_surface',
                                'source_type_1_unknown',
                                'source_type_2_hand dtw', 'source_type_2_
lake',
                                'source_type_2_machine dbh',
                                'source_type_2_other',
                                'source_type_2_rainwater harvesting',
                                'source_type_2_river',
                                'source_type_2_shallow well',
                                'source_type_2_spring', 'source_type_2_un
known',
                                'extraction_type_1_handpump',
                                'extraction_type_1_motorpump', ...])
```

In [754]:
```
#number of constant features
len(quasi_const.features_to_drop_)
```

executed in 2ms, finished 20:50:04 2021-05-23

Out[754]:
```
195
```

In [755]:
```
#example of a quasi-constant
quasi_ex = quasi_const.features_to_drop_[0]
X_train_tf[quasi_ex].value_counts(normalize=True)
```

executed in 4ms, finished 20:50:04 2021-05-23

Out[755]:
```
0.0      0.9999007567299343
1.0   9.924327006574867e-05
Name: waterpoint_type_dam, dtype: float64
```

In [756]:
```
#shape of dataframes before
X_train_tf.shape, X_test_tf.shape
```

executed in 2ms, finished 20:50:04 2021-05-23

Out[756]:
```
((40305, 468), (17274, 468))
```

In [757]:
```
#drop quasi-constants from train and test
X_train_tf = quasi_const.transform(X_train_tf)
X_test_tf = quasi_const.transform(X_test_tf)
```

executed in 306ms, finished 20:50:04 2021-05-23

In [758]:
```
#shape of dataframes after
X_train_tf.shape, X_test_tf.shape
```

executed in 2ms, finished 20:50:04 2021-05-23

Out[758]:
```
((40305, 273), (17274, 273))
```

In [759]:
```
#remove duplicate features
dup = DropDuplicateFeatures(missing_values='raise')
#fit the model
dup.fit(X_train_tf)
```

executed in 2.74s, finished 20:50:07 2021-05-23

Out[759]:
```
DropDuplicateFeatures(missing_values='raise',
                      variables=['waterpoint_type_communal standpipe',
                                 'waterpoint_type_communal standpipe mult
iple',
                                 'waterpoint_type_hand pump',
                                 'waterpoint_type_improved spring',
                                 'waterpoint_type_other',
                                 'water_quality_fluoride',
                                 'water_quality_milky', 'water_quality_sa
lty',
                                 'water_quality_salty abandoned',
                                 'water_quality_soft-good',
                                 'wat...
                                 'source_type_2_hand dtw', 'source_type_2
_lake',
                                 'source_type_2_machine dbh',
                                 'source_type_2_other',
                                 'source_type_2_rainwater harvesting',
                                 'source_type_2_river',
                                 'source_type_2_shallow well',
                                 'source_type_2_spring',
                                 'extraction_type_1_handpump',
                                 'extraction_type_1_motorpump',
                                 'extraction_type_1_other',
                                 'extraction_type_1_rope pump',
                                 'extraction_type_1_submersible', ...])
```

In [760]:
```
#duplicated features
dup.duplicated_feature_sets_
```

executed in 2ms, finished 20:50:07 2021-05-23

Out[760]:
```
[{'extraction_type_1_other', 'extraction_type_2_other'},
 {'extraction_type_1_rope pump', 'extraction_type_2_rope pump'}]
```

In [761]:
```
#simple check for duplicates between district 13 and lga=kilwa
df_clean.loc[df_clean['district_code'] == 13,['lga']].value_counts()
```
executed in 9ms, finished 20:50:07 2021-05-23

Out[761]:
```
lga
Kilwa    391
dtype: int64
```

In [762]:
```
#drop duplicate features
X_train_tf = dup.transform(X_train_tf)
X_test_tf = dup.transform(X_test_tf)
```
executed in 30ms, finished 20:50:07 2021-05-23

In [763]:
```
#shape of dataframes after
X_train_tf.shape, X_test_tf.shape
```
executed in 3ms, finished 20:50:07 2021-05-23

Out[763]:
```
((40305, 271), (17274, 271))
```

7.4 Logistic Regression

Logistic Regression will be my first model to classify water well failure. I will begin with a vanilla model before making improvements by evaluating hyperparameter tuning, class imbalances and feature selection.

In [764]:
```
#create copy of training and test set for logistic regression
X_train_lr = X_train_tf.copy()
X_test_lr = X_test_tf.copy()
```
executed in 12ms, finished 20:50:07 2021-05-23

7.4.1 Model Assumptions

7.4.1.1 Linearity with Target

In [765]:
```
#plot numerical features against the target
# num_cols = df_explore_binary.drop(columns='status_group').select_dtyp

# for col in num_cols:
#     sns.lmplot(data=df_explore_binary, x=col,y='status_group')
```
executed in 2ms, finished 20:50:07 2021-05-23

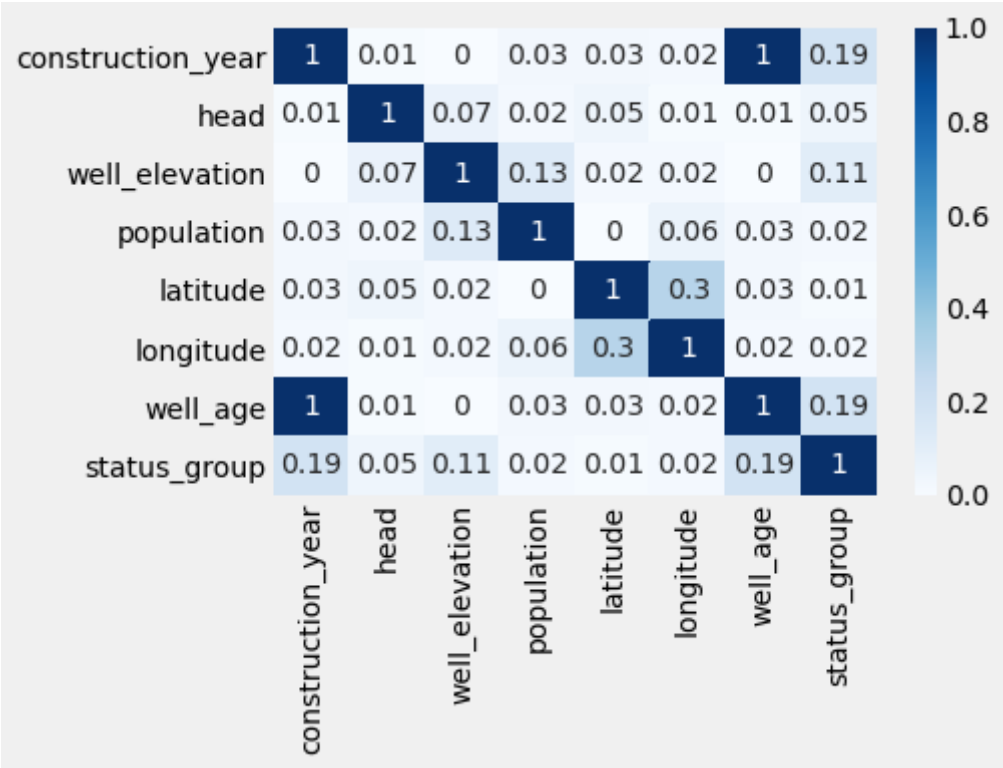
OBSERVATIONS

- The features seem to have a linear relationship with the target

7.4.1.2 Multicollinearity

In [766]:
```
1 sns.heatmap(df_explore_binary.corr().abs().round(2),annot=True, cmap='B
```
executed in 357ms, finished 20:50:07 2021-05-23

Out[766]: <AxesSubplot:>



OBSERVATIONS

- `construction_year` and `well_age` have perfect correlations which makes sense because `well_age` was created from `construction_year`

ACTIONS

- I will remove `construction_year`

In [767]:
```
1 #remove construction year from X_train and X_test
2 X_train_lr.drop(columns='construction_year', inplace=True)
3 X_test_lr.drop(columns='construction_year', inplace=True)
```
executed in 30ms, finished 20:50:07 2021-05-23

7.4.2 Model #1

In [768]:
```
#create a logistic regression model
lr_1 = LogisticRegression(C=1e12, max_iter=2500, n_jobs=-1)
#fit the model to the training data
lr_1.fit(X_train_lr, y_train)
```

executed in 36.0s, finished 20:50:43 2021-05-23

Out[768]:
```
LogisticRegression(C=1000000000000.0, max_iter=2500, n_jobs=-1)
```

7.4.3 Model Evaluation

Need to get:

- confusion matrix
- train score vs test score (< .05)

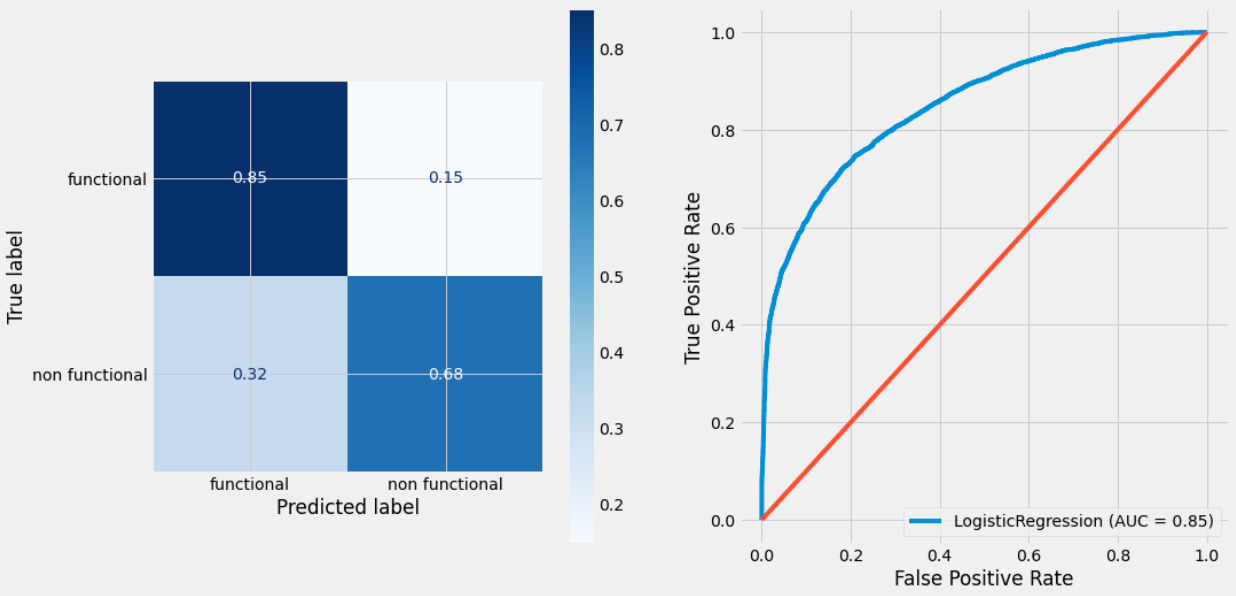
In [769]:
```
#eval model
model_eval(lr_1, X_train_lr, y_train, X_test_lr, y_test, prev_model=dum
```

executed in 529ms, finished 20:50:44 2021-05-23

```
MODEL EVAL VS PREVIOUS (TEST)
================================================================
```

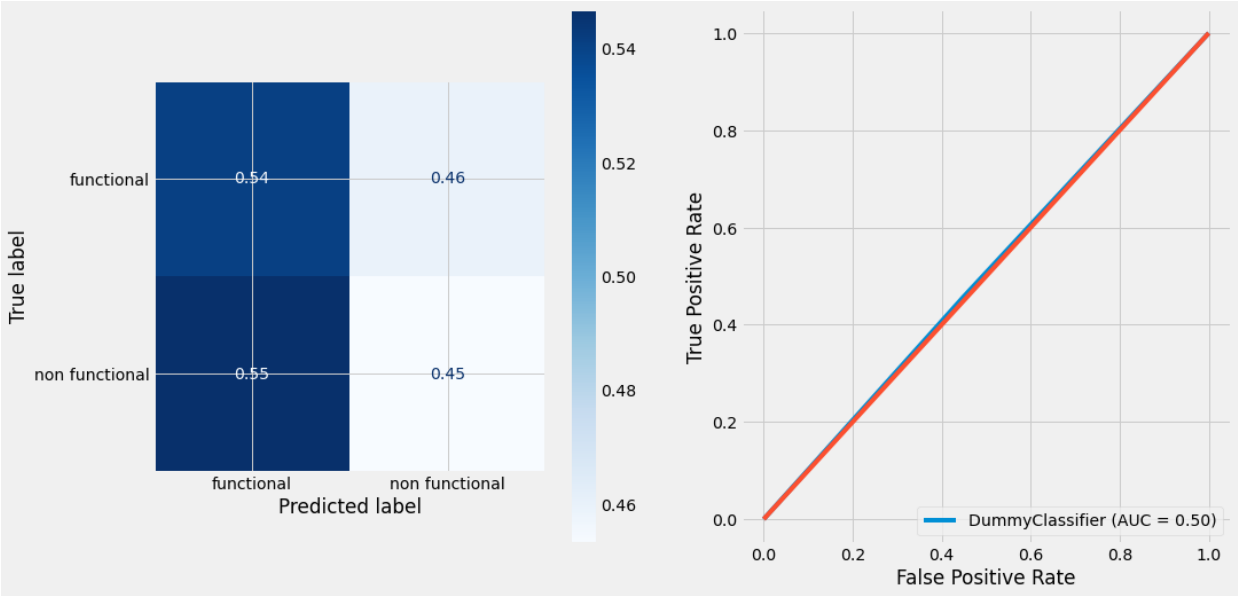
	Previous Model	Current Model	Delta
Recall	0.45	0.68	0.23
F1	0.46	0.73	0.27
Accuracy	0.51	0.77	0.26
AUC	0.5	0.85	0.35

```
CURRENT MODEL: Not Overfit (Recall)
================================================================

Recall on Training: 0.68
Recall on Test: 0.68

Classification Reports----------------------------------------
              precision    recall  f1-score   support

           0       0.76      0.85      0.80      9408
           1       0.80      0.68      0.73      7866

    accuracy                           0.77     17274
   macro avg       0.78      0.77      0.77     17274
weighted avg       0.78      0.77      0.77     17274

Test Graphs---------------------------------------------------
```




```
PREVIOUS MODEL
```

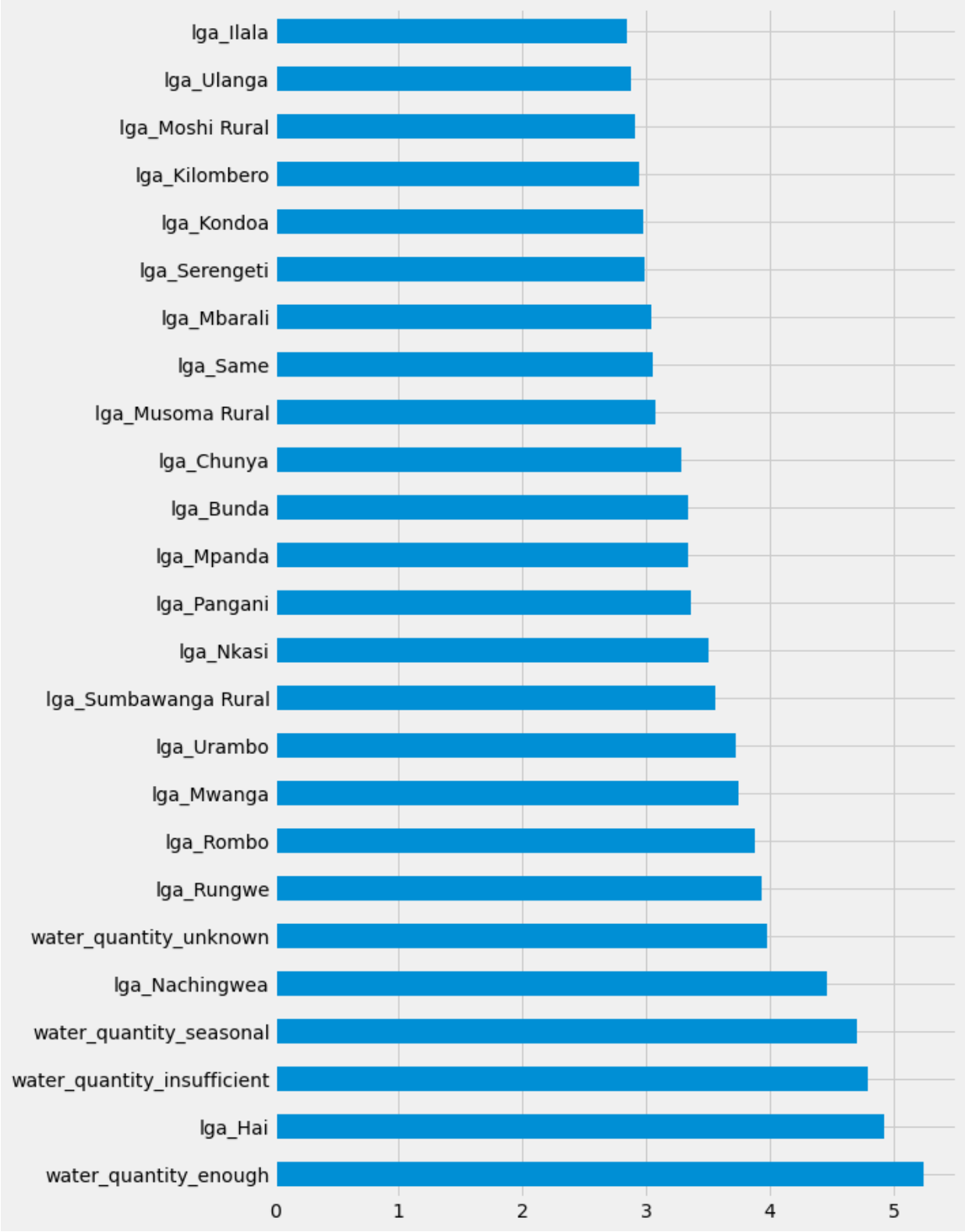
```
==================================================================
```




```
<Figure size 432x288 with 0 Axes>
```

In [770]:
```
#plot top coefficients
fig = plt.subplots(figsize=(8,15))
get_coefficients(lr_1,X_train_lr).abs().sort_values(ascending=False)[:2
```

executed in 165ms, finished 20:50:44 2021-05-23



OBSERVATIONS

- LR model accuracy improved on f1 by 30% and recall by 25%
- model is not overfit as it pertains to recall

ACTIONS

- Find the best parameters using gridsearch

7.4.4 Model #2

Use gridsearch to identify the optimal parameters for getting the best score for recall.

In [771]:
```
#create new lr model
lr_2 = LogisticRegression()
```

executed in 2ms, finished 20:50:44 2021-05-23

In [772]:
```
#set up parameters to iterate on
params_2 = {'C': [1e7, 1e8, 1e9, 1e10],
            'penalty': ['l1', 'l2'],
            'solver': ['liblinear']}
```
executed in 1ms, finished 20:50:44 2021-05-23

In [773]:
```
#perform a gridsearchCV in order to find the best parameters
lr_2_gridsearch = GridSearchCV(lr_2, params_2,scoring='recall',n_jobs=-
lr_2_gridsearch.fit(X_train_lr, y_train)
```
executed in 30.5s, finished 20:51:15 2021-05-23

Out[773]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(), n_jobs=-1,
             param_grid={'C': [10000000.0, 100000000.0, 1000000000.0,
                               10000000000.0],
                         'penalty': ['l1', 'l2'], 'solver': ['liblinea
r']},
             scoring='recall')
```

7.4.5 Model Evaluation

In [774]:
```
#get best parameters from gridsearch
lr_2_gridsearch.best_params_
```
executed in 3ms, finished 20:51:15 2021-05-23

Out[774]:
```
{'C': 10000000.0, 'penalty': 'l1', 'solver': 'liblinear'}
```

In [775]:
```
#get best coeffs
lr_2_gridsearch.best_estimator_.coef_
```

executed in 4ms, finished 20:51:15 2021-05-23

Out[775]:
```
array([[ 1.05087736e+00,  1.76335907e+00,  1.03341219e+00,
        -1.88827958e-01,  2.48332779e+00, -1.87105572e-01,
        -8.11281382e-01,  4.77996984e-02,  3.36263298e-01,
         9.03340897e-02,  7.04843321e-01, -5.23891670e+00,
        -4.78314862e+00, -4.69167850e+00, -3.96731074e+00,
         8.90203278e-01,  2.32522997e-01,  1.25461386e+00,
         3.88610634e-01,  1.05906942e+00, -1.61191242e-01,
        -9.91260789e-01, -4.18514408e-03,  1.06746027e+00,
         4.57008605e-01,  7.59569490e-01,  1.00464418e+00,
         1.60778238e+00,  5.30029249e-01,  6.00264119e-01,
         1.34155594e+00, -5.58610934e-01,  8.96383963e-01,
        -3.66508294e-01,  1.15415291e+00,  7.26065185e-02,
        -8.64075221e-01,  1.50951448e-01, -2.89605444e-01,
        -1.35292076e-01, -2.94743818e-02, -8.67306900e-02,
        -2.81643623e-01,  3.53020201e-01,  1.87632648e-01,
         7.27292546e-02,  8.89668234e-01,  2.92810241e-01,
        -4.05005387e-01, -6.70273488e-02, -4.91633223e-01,
        -2.97014899e-01,  9.59108910e-01,  1.23407517e-01,
         2.98788506e-01,  5.41166849e-01,  1.30779520e-01,
         2.36872337e-01,  1.30331602e+00,  1.15447775e-01,
         1.86335741e+00,  3.64815091e-01, -6.93236928e-01,
        -1.68110733e+00, -1.19981784e-01,  5.32258781e-01,
         1.03651041e+00, -2.64740296e+00, -1.55204287e+00,
        -2.87446711e-01, -8.03305979e-01, -4.39613833e-01,
        -1.44209050e+00,  9.00713381e-02,  3.65509391e-01,
        -9.14718159e-01, -4.95328757e-01, -4.32931497e-01,
         5.58432442e-01, -5.46669306e-01, -9.13263392e-01,
        -1.69739008e+00,  3.38319401e-01,  8.49581638e-01,
         1.58979822e+00,  1.92821792e+00,  1.29194735e+00,
         3.31607688e+00,  1.55486190e+00,  3.26248672e+00,
         1.54270293e+00,  4.88253765e+00,  2.35240809e+00,
         2.82180356e+00,  1.99046756e+00, -4.40225458e-01,
         5.13682221e-02,  4.38765756e-01, -1.67431945e+00,
         9.93229219e-01,  6.38259094e-01,  2.00874139e+00,
         1.53510209e+00,  2.92198539e+00,  2.01753127e+00,
         2.62982719e+00,  5.73812292e-01,  2.92179333e+00,
        -7.67172280e-01,  1.22631150e+00,  4.39108355e-01,
         7.08642210e-01,  2.40682992e+00, -6.40857766e-02,
         9.00305541e-01,  1.30674403e+00,  1.52013430e+00,
        -5.72599854e-01,  2.56003048e+00,  8.77300639e-01,
         1.16285283e+00,  3.01027049e+00,  2.27601506e+00,
         1.38098046e+00,  1.95626798e+00,  1.21946527e+00,
        -4.36819773e-01, -1.67720756e-02,  1.53145242e-01,
         9.11358405e-01,  1.46058982e+00,  2.11172437e+00,
         2.87775324e+00,  3.28294281e+00,  2.60876801e+00,
         1.06800052e+00,  1.36522202e+00,  5.83769847e-01,
         2.36815496e+00,  3.04363262e+00,  5.50174957e-01,
         3.71667727e+00,  4.40951017e+00,  2.71877957e+00,
         4.84217201e-01,  1.60126436e+00,  3.45370156e+00,
         1.85774800e+00,  3.31557992e+00,  1.48959900e+00,
         3.84681831e+00,  2.76289057e+00,  7.42463221e-01,
         3.89980995e+00,  3.01772772e+00,  1.10808395e+00,
         2.95699449e+00,  3.69342897e-01,  2.52345145e+00,
```

```
 1.17327388e+00,  1.94164844e+00,  1.32583729e+00,
 3.52461413e+00,  2.20228769e+00,  9.30780455e-01,
 2.87145985e+00,  3.68698414e+00,  2.65925790e+00,
 3.99174356e-01, -5.51868490e-01,  9.90687600e-02,
-1.24656951e+00, -7.99240773e-01,  6.84979814e-01,
-1.65237116e+00, -5.39686768e-01, -6.57101385e-01,
 1.07791010e+00, -7.52970114e-01, -7.85819330e-02,
-6.90683426e-01, -6.86681867e-02, -6.51755017e-01,
-4.76253269e-01, -8.19050950e-03, -1.18359878e-01,
-5.82617551e-01,  8.77594553e-02, -6.65685352e-01,
-2.43634468e-01, -6.13854172e-01,  1.88267480e-01,
 4.88329953e-01,  6.30962205e-01, -8.03728207e-01,
 2.66427687e-01,  1.29531568e-01, -2.06417175e-01,
-1.47609318e+00,  2.25914299e-01,  4.00165421e-01,
-1.60449299e+00,  2.74342244e-01,  2.53269442e-01,
-3.80947180e-01, -1.75251675e-01, -9.28518885e-01,
 6.92412964e-02,  3.99471300e-02, -1.20357361e-01,
-1.22476327e-01, -7.13713795e-03, -9.48625915e-01,
-4.35100787e-01, -3.96546494e-01, -3.27873334e-01,
-7.70816998e-01, -5.20440771e-01, -6.20320429e-01,
-3.37020833e-01, -1.04803191e-01, -2.38302330e-01,
 1.28080614e+00,  1.45790392e+00,  6.77581623e-02,
 5.51023771e-01,  9.60860041e-01,  6.64000923e-01,
 1.46978182e+00,  1.26300722e+00,  1.14904593e+00,
 1.37133830e+00,  8.91219389e-01,  1.07274895e+00,
 2.71883473e-01, -2.54103215e-01, -1.70134057e-01,
-3.64671377e-01, -3.60178704e-01, -5.66774413e-01,
-7.98377078e-02, -1.78667466e-02, -5.84139833e-01,
-3.54673775e-01, -3.53443373e-01, -6.20152904e-01,
-5.64412043e-01, -8.34238586e-01,  3.46668306e-01,
-5.53301731e-01, -5.20389788e-01, -1.22731210e+00,
-6.03787339e-01,  2.16607771e-01, -4.56353861e-01,
 1.78231521e-01,  9.46374574e-02,  8.96628658e-02,
-5.82079030e-01, -9.13819224e-01, -6.71686971e-01,
-1.15719224e+00, -5.72100093e-01,  6.65799615e-02,
-8.46543453e-02,  1.29459702e-02, -1.11735057e-01,
 3.54682279e-01,  4.28029022e-01,  1.94059401e-01]])
```

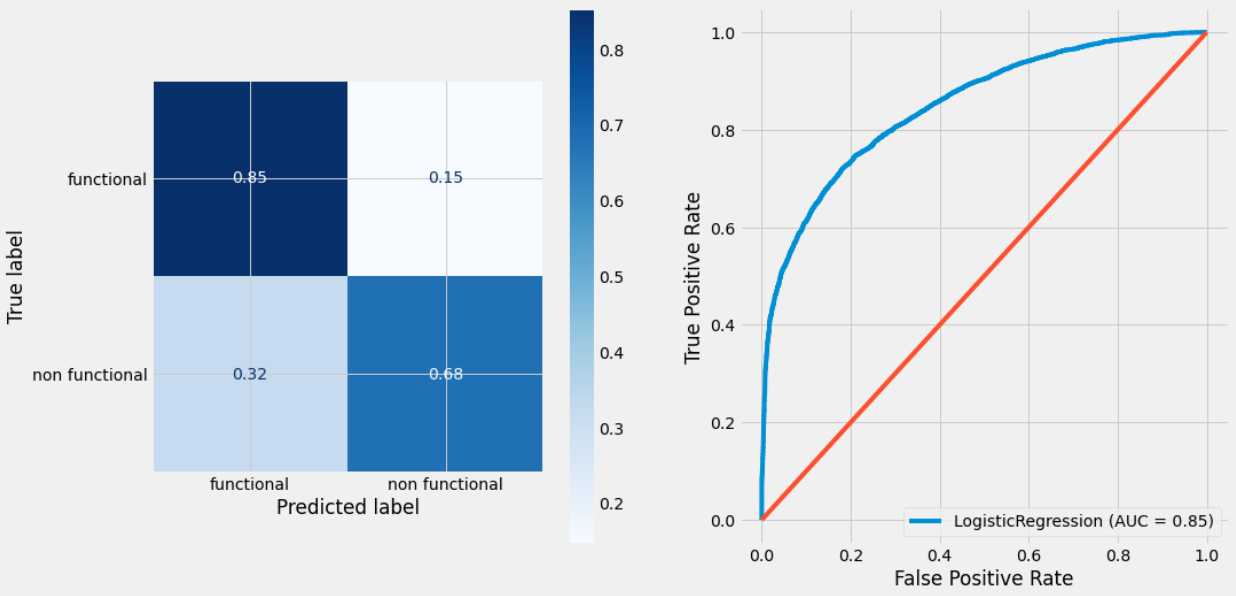
In [776]:
```
#eval the best estimator from gridsearch
model_eval(lr_2_gridsearch.best_estimator_, X_train_lr, y_train, X_test
           y_test, prev_model=lr_1)
```

executed in 587ms, finished 20:51:15 2021-05-23

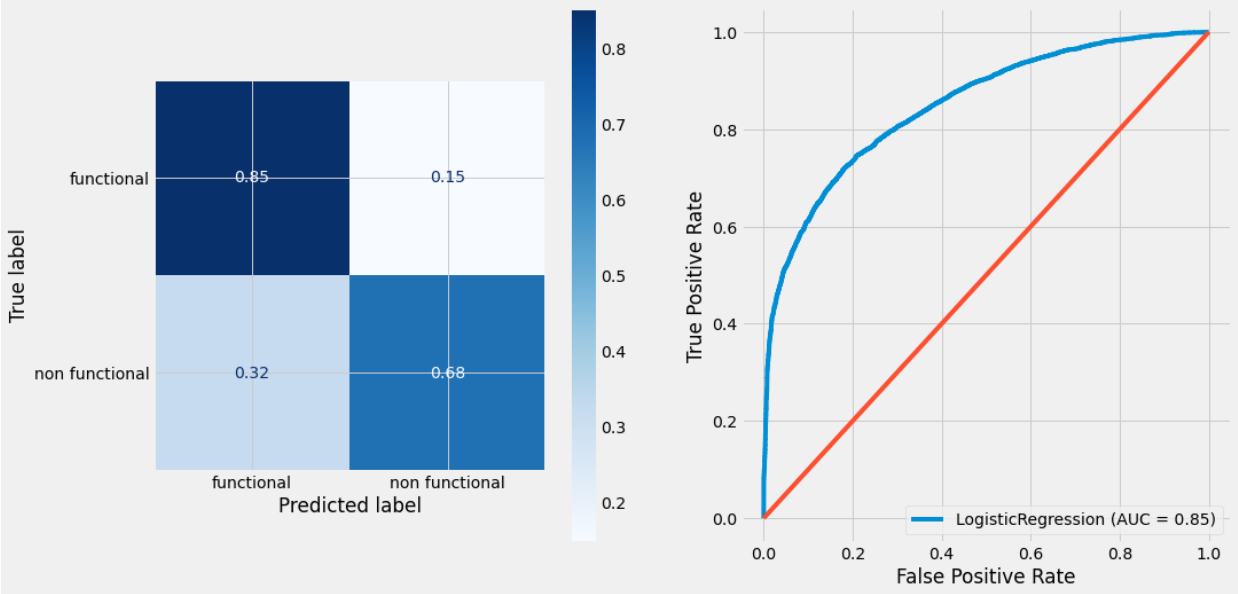
```
MODEL EVAL VS PREVIOUS (TEST)
================================================================
```

	Previous Model	Current Model	Delta
Recall	0.68	0.68	0.0
F1	0.73	0.73	0.0
Accuracy	0.77	0.77	0.0
AUC	0.85	0.85	0.0

```
CURRENT MODEL: Not Overfit (Recall)
================================================================


Recall on Training: 0.68
Recall on Test: 0.68


Classification Reports----------------------------------------
              precision    recall  f1-score   support

           0       0.76      0.85      0.80      9408
           1       0.80      0.68      0.73      7866

    accuracy                           0.77     17274
   macro avg       0.78      0.77      0.77     17274
weighted avg       0.78      0.77      0.77     17274

Test Graphs---------------------------------------------------
```



```
PREVIOUS MODEL
================================================================
```




```
<Figure size 432x288 with 0 Axes>
```

OBSERVATIONS

- gridsearching through different C values, penalties and solvers gave the exact same score as a vanilla logistic regression model.
- best model for optimizing recall used a penalty of ridge regularization(l2), the liblinear solver and a C value of 1e8.

ACTIONS

- see if fixing class imbalance will help the model

7.4.6 Model #3

I will run a gridsearch while making the target classes balanced to see if there is any improvement.

In [777]:
```
#create new lr model with target class balanced
lr_3 = LogisticRegression(class_weight='balanced')
```
executed in 2ms, finished 20:51:15 2021-05-23

In [778]:
```
#set up parameters to iterate on
params_3 = {'C': [1e7, 1e8, 1e9, 1e10],
            'penalty': ['l1', 'l2'],
            'solver': ['liblinear']}
```
executed in 2ms, finished 20:51:15 2021-05-23

In [779]:
```
#perform a gridsearchCV in order to find the best parameters
lr_3_gridsearch = GridSearchCV(lr_3, params_3,scoring='recall',n_jobs=-
lr_3_gridsearch.fit(X_train_lr, y_train)
```

executed in 31.1s, finished 20:51:46 2021-05-23

Out[779]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(class_weight='balanced'),
             n_jobs=-1,
             param_grid={'C': [10000000.0, 100000000.0, 1000000000.0,
                               10000000000.0],
                         'penalty': ['l1', 'l2'], 'solver': ['liblinea
r']},
             scoring='recall')
```

7.4.7 Model Evaluation

In [780]:
```
#get best parameters from gridsearch
lr_3_gridsearch.best_params_
```

executed in 2ms, finished 20:51:46 2021-05-23

Out[780]:
```
{'C': 100000000.0, 'penalty': 'l1', 'solver': 'liblinear'}
```

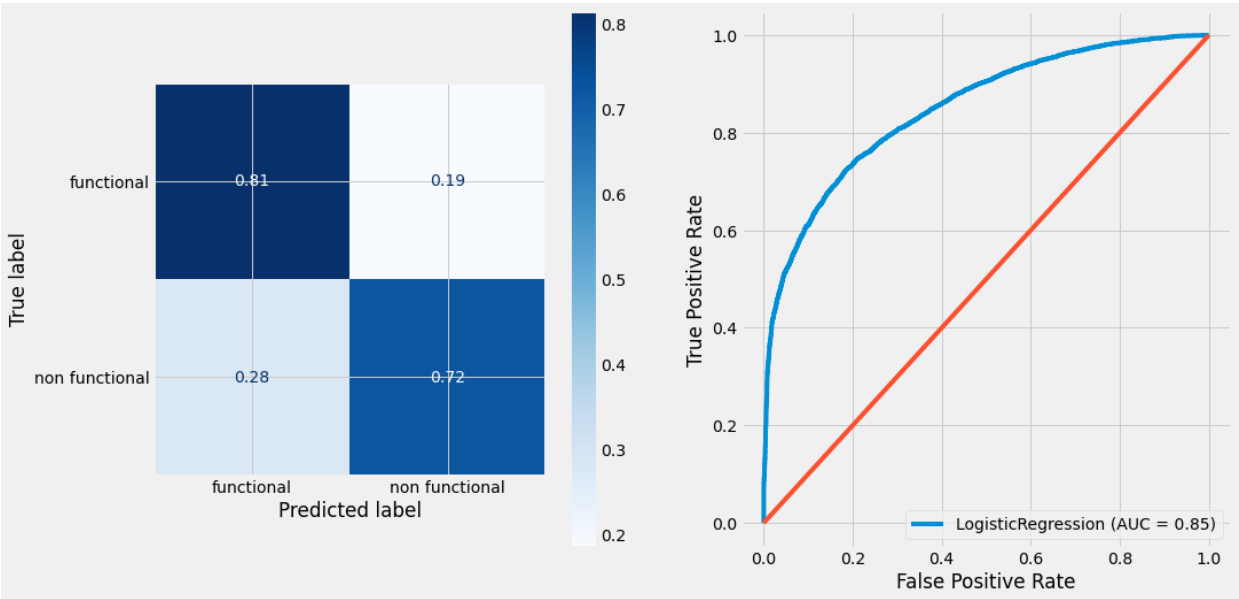
In [781]:
```
#eval the best estimator from gridsearch
model_eval(lr_3_gridsearch.best_estimator_, X_train_lr, y_train, X_test
           y_test, prev_model=lr_2_gridsearch.best_estimator_)
```

executed in 546ms, finished 20:51:47 2021-05-23

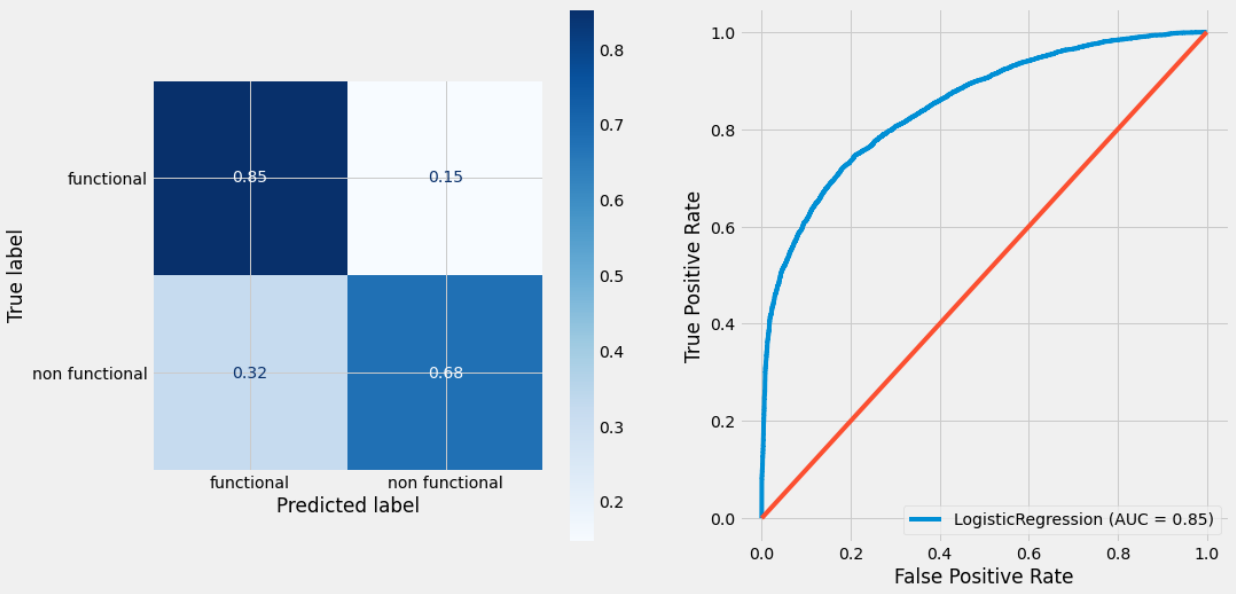
```
MODEL EVAL VS PREVIOUS (TEST)
================================================================
```

	Previous Model	Current Model	Delta
Recall	0.68	0.72	0.04
F1	0.73	0.74	0.01
Accuracy	0.77	0.77	0.0
AUC	0.85	0.85	0.0

```
CURRENT MODEL: Not Overfit (Recall)
================================================================

Recall on Training: 0.72
Recall on Test: 0.72

Classification Reports----------------------------------------
              precision    recall  f1-score   support

           0       0.78      0.81      0.80      9408
           1       0.77      0.72      0.74      7866

    accuracy                           0.77     17274
   macro avg       0.77      0.77      0.77     17274
weighted avg       0.77      0.77      0.77     17274

Test Graphs---------------------------------------------------
```



```
PREVIOUS MODEL
================================================================
```




```
<Figure size 432x288 with 0 Axes>
```

OBSERVATIONS

- Balancing the classes improved the model by increasing the recall score by 4% and the f1 by 1%

ACTIONS

- Eval feature selection

7.4.8 Model #4

For this model I will utilize feature selection in order to drop additional features based on lasso regularization. I will vary the penalty of the regularization and evaluate the models. The goal is to select the least amount of features without drastically hurting the model.

In [782]:
```
#use embedded methods for feature selection
lr_4 = LogisticRegression(C=10, penalty='l1',solver='liblinear')
sel_features = SelectFromModel(lr_4)
#fit the model
sel_features.fit(X_train_lr, y_train)
```

executed in 5.03s, finished 20:51:52 2021-05-23

Out[782]:
```
SelectFromModel(estimator=LogisticRegression(C=10, penalty='l1',
                                             solver='liblinear'))
```

In [783]:
```
#previous dataframe shape
X_train_lr.shape, X_test_lr.shape
```
executed in 2ms, finished 20:51:52 2021-05-23

Out[783]:
```
((40305, 270), (17274, 270))
```

In [784]:
```
#get number of features to keep
selected_feat = X_train_lr.columns[sel_features.get_support()]
print(f'Number of features selected to keep: {len(selected_feat)}')
```
executed in 2ms, finished 20:51:52 2021-05-23

```
Number of features selected to keep: 262
```

In [785]:
```
#get selected features to remove
len(sel_features.get_support())
print(f'Number of features shrank to 0: {(sel_features.estimator_.coef_
```
executed in 1ms, finished 20:51:52 2021-05-23

```
Number of features shrank to 0: 8
```

In [786]:
```
#create updated dataframes for train and test
X_train_lr_fs = sel_features.transform(X_train_lr)
X_test_lr_fs = sel_features.transform(X_test_lr)
```
executed in 30ms, finished 20:51:52 2021-05-23

In [787]:
```
#see shape of current dataframe
X_train_lr_fs.shape, X_test_lr_fs.shape
```
executed in 2ms, finished 20:51:52 2021-05-23

Out[787]:
```
((40305, 262), (17274, 262))
```

In [788]:
```
#set up parameters to iterate on
params_4 = {'C': [10],
            'penalty': ['l1'],
            'solver': ['liblinear']}
```
executed in 2ms, finished 20:51:52 2021-05-23

In [789]:
```
#perform a gridsearchCV in order to find the best parameters
lr_4_gridsearch = GridSearchCV(lr_4, params_4,scoring='recall',n_jobs=-
lr_4_gridsearch.fit(X_train_lr_fs, y_train)
```
executed in 9.94s, finished 20:52:02 2021-05-23

Out[789]:
```
GridSearchCV(cv=5,
             estimator=LogisticRegression(C=10, penalty='l1',
                                          solver='liblinear'),
             n_jobs=-1,
             param_grid={'C': [10], 'penalty': ['l1'], 'solver': ['liblin
ear']},
             scoring='recall')
```

7.4.9 Model Evaluation

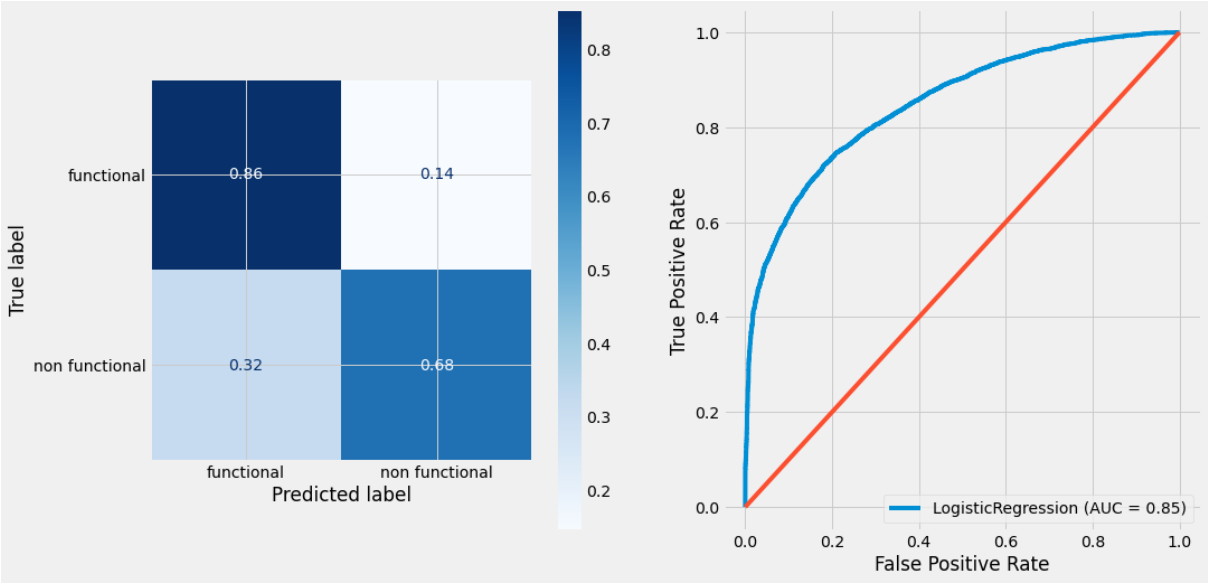
In [790]:
```
#eval the new model
model_eval(lr_4_gridsearch.best_estimator_, X_train_lr_fs, y_train,
           X_test_lr_fs, y_test, prev_model=lr_3_gridsearch.best_estima
          prev_X_train=X_train_lr, prev_y_train=y_train, prev_X_test=X_
           prev_y_test=y_test)
```

executed in 526ms, finished 20:52:02 2021-05-23

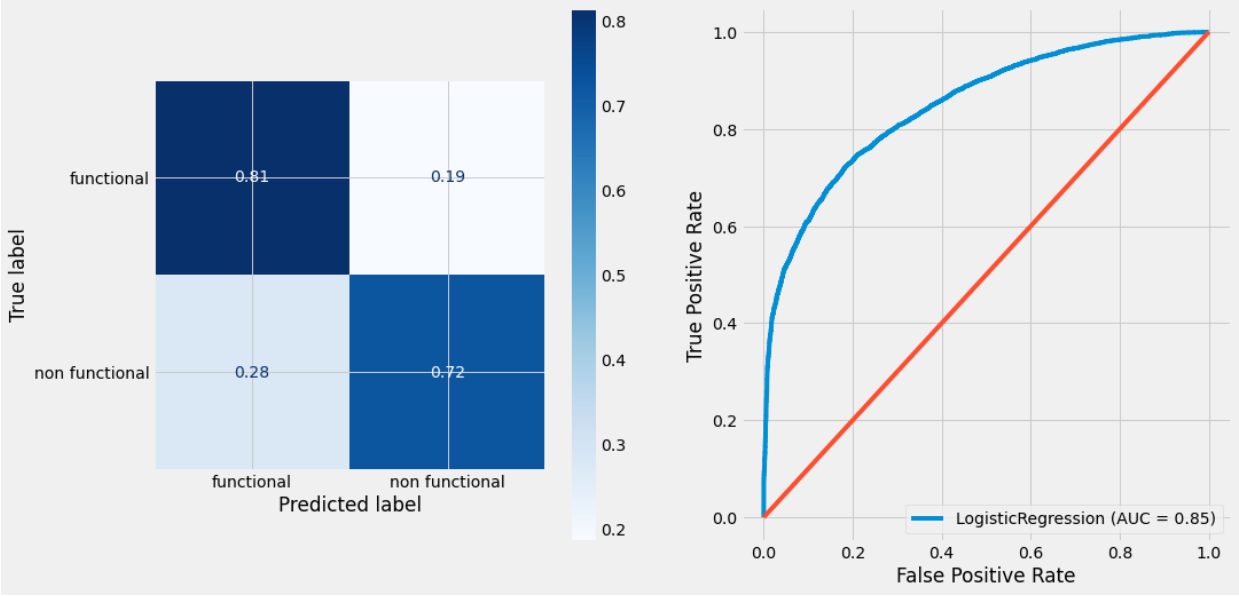
```
MODEL EVAL VS PREVIOUS (TEST)
================================================================
```

	Previous Model	Current Model	Delta
Recall	0.72	0.68	-0.04
F1	0.74	0.73	-0.01
Accuracy	0.77	0.77	0.0
AUC	0.85	0.85	0.0

```
CURRENT MODEL: Not Overfit (Recall)
================================================================


Recall on Training: 0.68
Recall on Test: 0.68


Classification Reports----------------------------------------
              precision    recall  f1-score   support

           0       0.76      0.86      0.81      9408
           1       0.80      0.68      0.73      7866

    accuracy                           0.77     17274
   macro avg       0.78      0.77      0.77     17274
weighted avg       0.78      0.77      0.77     17274

Test Graphs---------------------------------------------------
```




```
PREVIOUS MODEL
================================================================
```




```
<Figure size 432x288 with 0 Axes>
```

OBSERVATIONS

- The new model remove features through lasso regularization, however, the model recall takes a 5% hit. I believe the best Logistic Regression model is Model 3:
  - class balanced
  - C = 1e8
  - penalty = l2
  - solver = liblinear
- **Recall = .72, F1 = .74**

ACTIONS

- Look at most valuable coefficients for the model

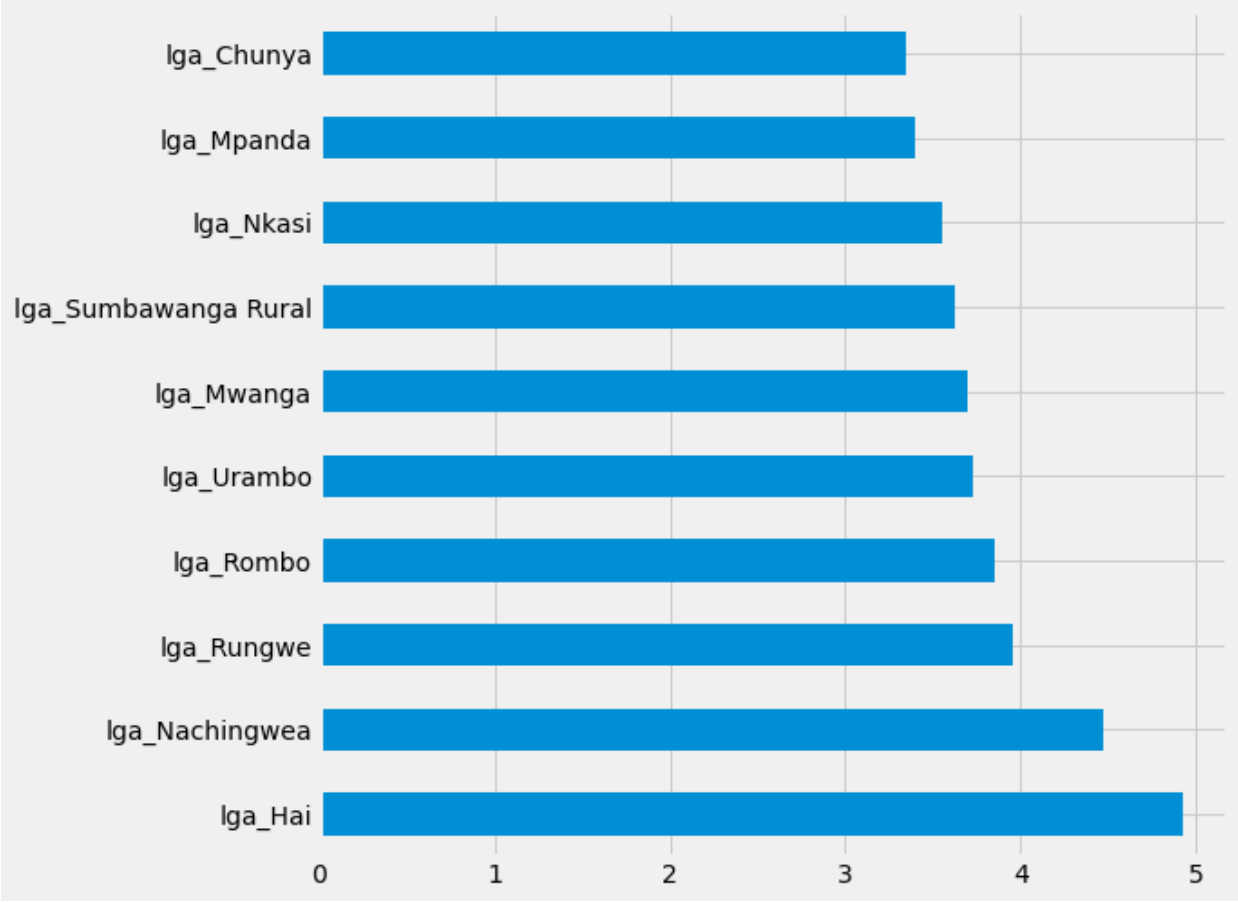
In [791]:
```
#find top coefficients
best_coeffs = get_coefficients(lr_3_gridsearch.best_estimator_,X_train_
best_coeffs_pos = best_coeffs.loc[best_coeffs >= 0].sort_values(ascendi
best_coeffs_neg = best_coeffs.loc[best_coeffs < 0].sort_values(ascendin
best_coeffs_overall = best_coeffs.abs().sort_values(ascending=False)
```

executed in 5ms, finished 20:52:02 2021-05-23

In [792]:
```
#top 10 positive coeffs
fig = plt.subplots(figsize=(8,8))
best_coeffs_pos[:10].plot.barh()
```

executed in 229ms, finished 20:52:03 2021-05-23

Out[792]: <AxesSubplot:>



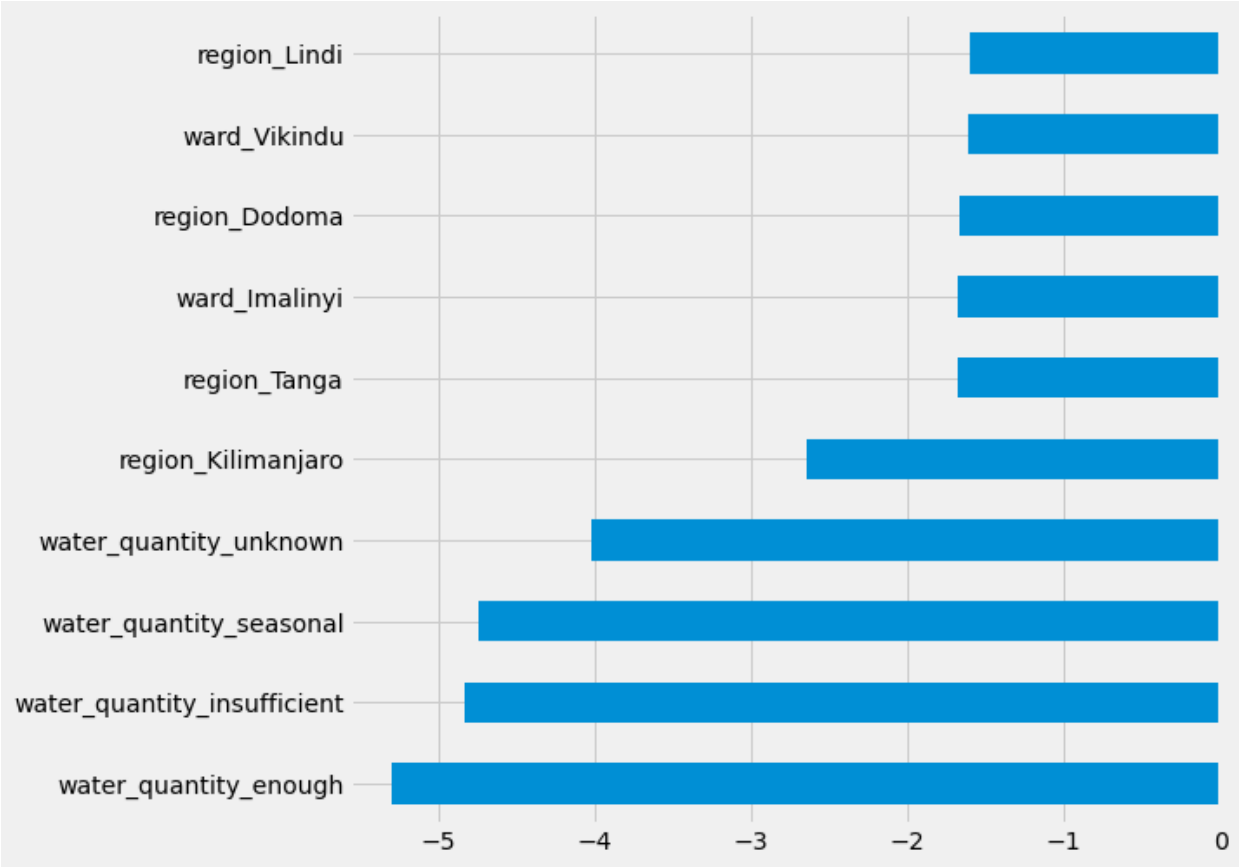
OBSERVATIONS

- The top coefficients which predict the target (non functional water well) most strongly are various local government authorities. There might be a strong correlation between how these lga's govern the water wells in this area that make them less reliable.

In [793]:
```
#top 10 negative coeffs
fig = plt.subplots(figsize=(8,8))
best_coeffs_neg[:10].plot.barh()
```

executed in 100ms, finished 20:52:03 2021-05-23

Out[793]: <AxesSubplot:>

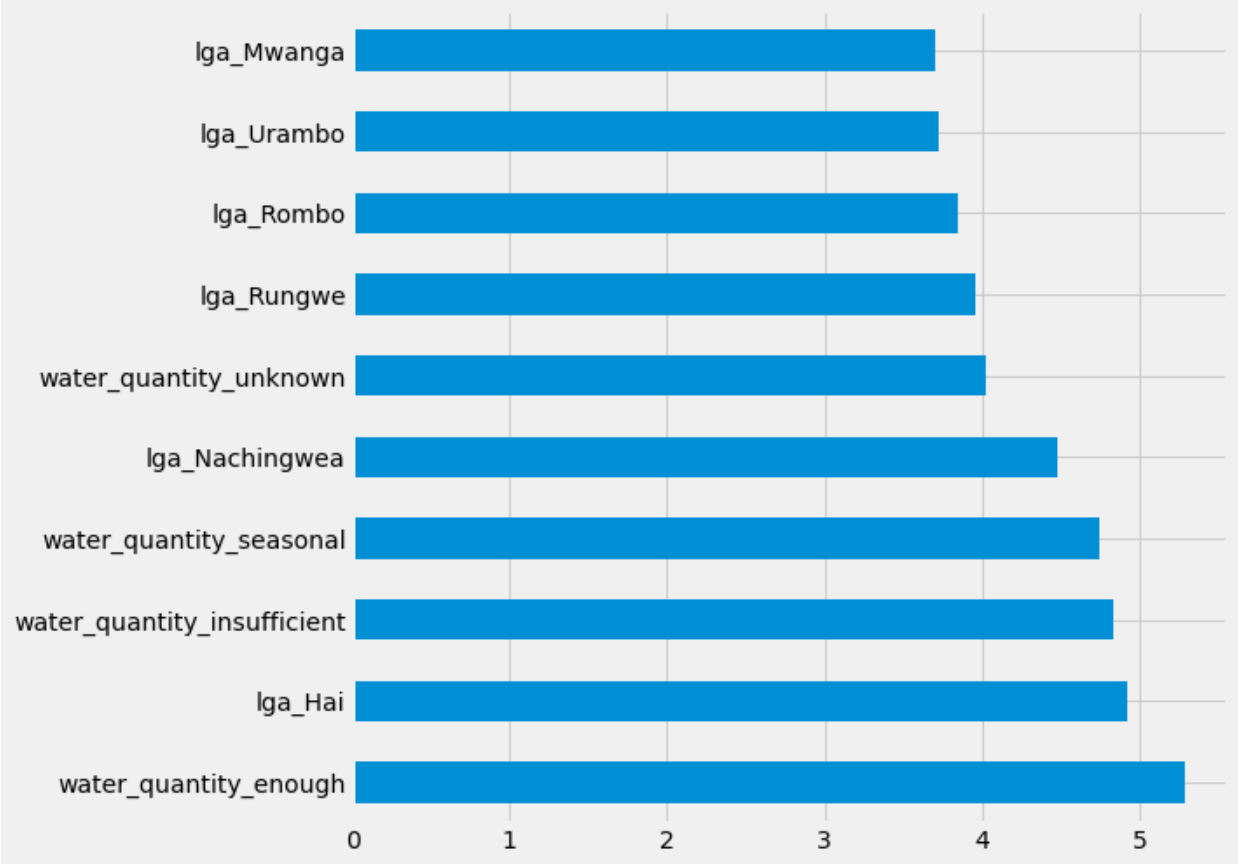


OBSERVATIONS

- The top coefficients which correlate negatively with predicting the target (non functional water well) are from the `water_quantity` feature. While it does make sense that having enough water does accurately predict if the well is functional (opposite of non functional), it is strange to see that having an insufficient water supply is also a good predictor of if the well is functional.

In [794]:
```
#top 10 coeffs overall
fig = plt.subplots(figsize=(8,8))
best_coeffs_overall[:10].plot.barh();
```

executed in 94ms, finished 20:52:03 2021-05-23



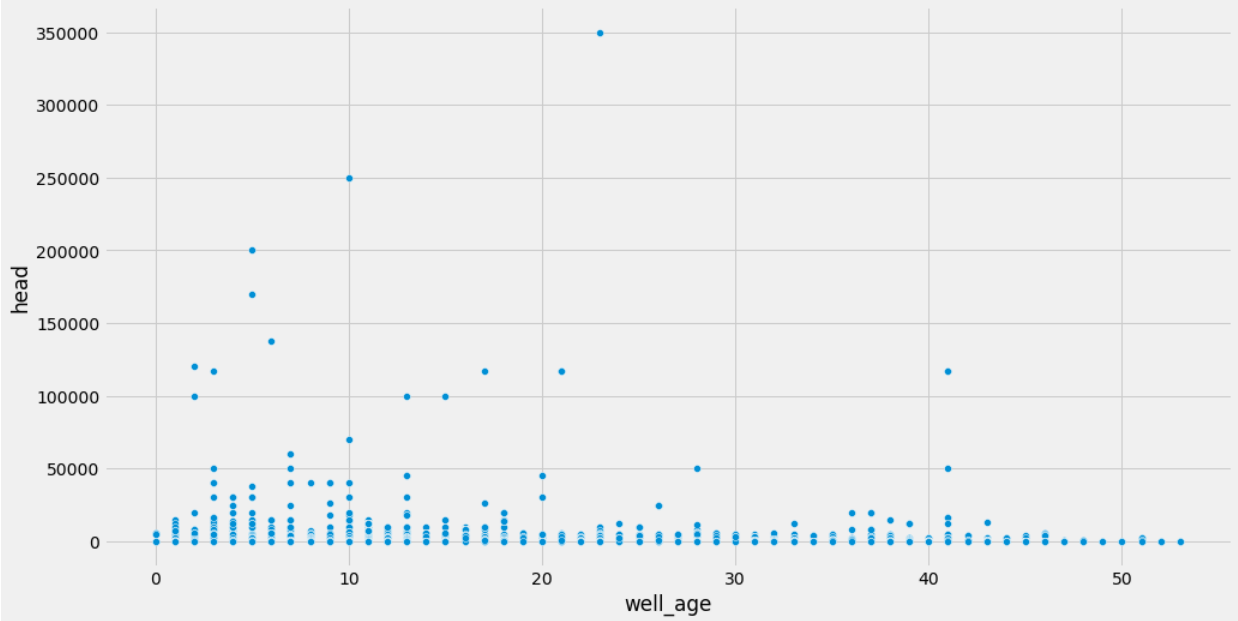
OBSERVATIONS

- The top features for making predictions overall remain the water quantity followed by the local government authorities.

In [795]:
```
1 plt.figure(figsize=(15,8))
2 sns.scatterplot(data=df_explore, x='well_age',y='head')
```

executed in 140ms, finished 20:52:03 2021-05-23

Out[795]: <AxesSubplot:xlabel='well_age', ylabel='head'>



7.5 Random Forrest

I now want to explore if a Random Forest classification model will perform better than the best logistic regression model I found.

In [796]:
```
1 #review training dataframe
2 X_train_dt
```
executed in 17ms, finished 20:52:03 2021-05-23

Out[796]:

	construction_year	waterpoint_type	water_quality	water_quantity	head	source_type_1	so
16350	1,982.0	communal standpipe multiple	soft-good	enough	1,000.0	surface	
19439	2,012.0	communal standpipe	salty	insufficient	0.0	groundwater	r
44958	2,000.0	communal standpipe	soft-good	enough	3,000.0	groundwater	
55544	2,000.0	other	soft-good	dry	0.0	groundwater	
34351	2,006.0	communal standpipe	soft-good	enough	0.0	groundwater	
...	...	...	...	...	...	...	
34029	1,972.0	other	soft-good	enough	0.0	groundwater	
6523	2,006.0	hand pump	soft-good	enough	0.0	groundwater	
33122	2,000.0	hand pump	soft-good	insufficient	0.0	groundwater	
26755	1,984.0	communal standpipe	soft-good	enough	200.0	surface	
4054	2,009.0	communal standpipe multiple	salty	dry	0.0	groundwater	r

40305 rows × 26 columns

OBSERVATIONS

- Need to encode categorical labels

7.5.1 Model Preprocessing

In [797]:
```
#encode the categorical features
X_train_dt = pd.get_dummies(X_train_dt)
X_test_dt = pd.get_dummies(X_test_dt)
X_train_dt.drop(columns=['construction_year'], inplace=True)
X_test_dt.drop(columns=['construction_year'], inplace=True)
X_train_dt
```

executed in 188ms, finished 20:52:03 2021-05-23

Out[797]:

	head	well_elevation	population	latitude	longitude	well_age	waterpoint_type_catt troug
16350	1,000.0	832	250	-10.46151284	36.13949617	31.0	
19439	0.0	1403	140	-2.14681927	34.68605048	1.0	
44958	3,000.0	2062	106	-9.71102678	34.73972738	13.0	
55544	0.0	0	0	-2.53441687	33.126675	13.0	
34351	0.0	1310	120	-4.20019169	35.85744301	7.0	
...	...	...	...	...	...	...	
34029	0.0	1475	1300	-3.70745784	37.65754363	41.0	
6523	0.0	1414	450	-2.94766891	34.39509404	7.0	
33122	0.0	0	0	-8.9994183	32.79584178	13.0	
26755	200.0	546	300	-7.34181122	36.29927858	29.0	
4054	0.0	0	0	-6.02477713	36.70740197	4.0	

40305 rows × 486 columns

7.5.2 Model #1

In [798]:
```
#produce a vanilla random forest model
rf_1 = RandomForestClassifier()
#fit the data
rf_1.fit(X_train_dt, y_train)
```

executed in 10.5s, finished 20:52:14 2021-05-23

Out[798]:
```
RandomForestClassifier()
```

7.5.3 Model Evaluation

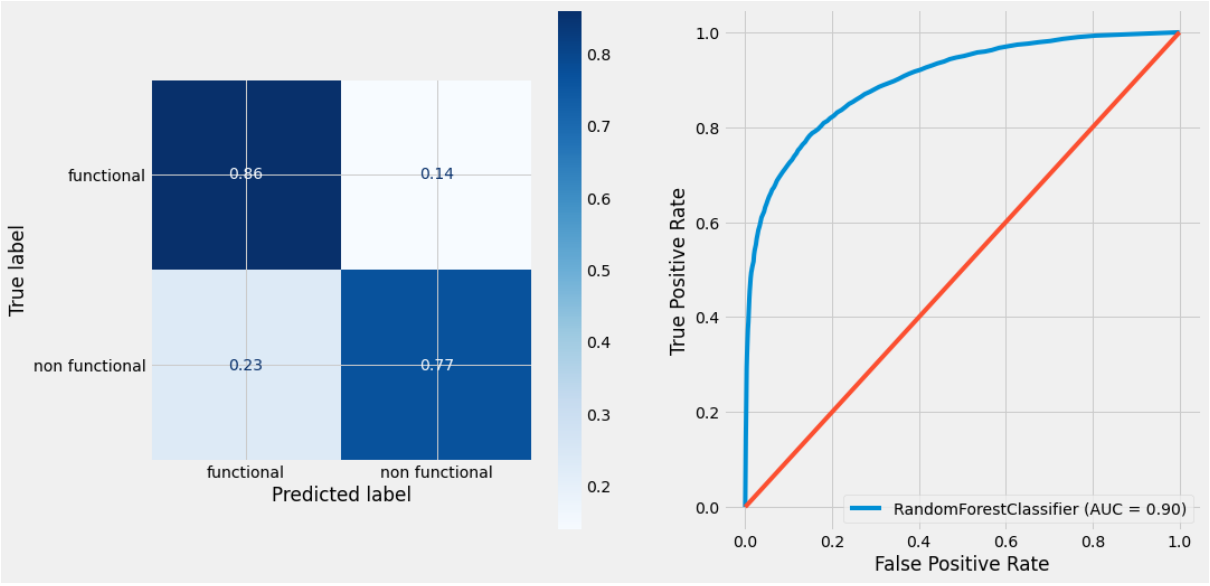
In [799]:
```
model_eval(rf_1, X_train_dt, y_train, X_test_dt, y_test,
           prev_model=lr_3_gridsearch.best_estimator_,
          prev_X_train=X_train_lr, prev_y_train=y_train, prev_X_test=X_
           prev_y_test=y_test)
```

executed in 4.05s, finished 20:52:18 2021-05-23

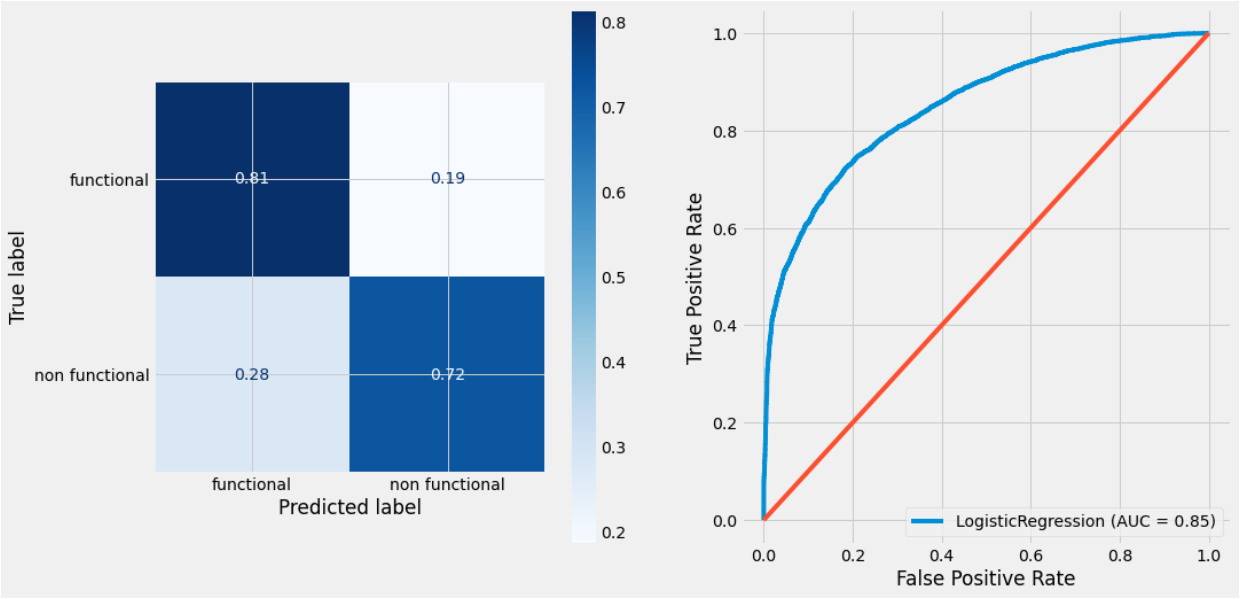
```
MODEL EVAL VS PREVIOUS (TEST)
================================================================
```

	Previous Model	Current Model	Delta
Recall	0.72	0.77	0.05
F1	0.74	0.8	0.06
Accuracy	0.77	0.82	0.05
AUC	0.85	0.9	0.05

```
CURRENT MODEL: Overfit (Recall)
================================================================


Recall on Training: 1.0
Recall on Test: 0.77


Classification Reports----------------------------------------
              precision    recall  f1-score   support

           0       0.82      0.86      0.84      9408
           1       0.82      0.77      0.80      7866

    accuracy                           0.82     17274
   macro avg       0.82      0.82      0.82     17274
weighted avg       0.82      0.82      0.82     17274

Test Graphs---------------------------------------------------
```




```
PREVIOUS MODEL
==============================================================================
```




```
<Figure size 432x288 with 0 Axes>
```

In [800]:
```
#number of leaves
rf_1.estimators_[0].get_n_leaves()
```

executed in 4ms, finished 20:52:18 2021-05-23

Out[800]: 9137

In [801]:
```
#max depth
depths = [m.get_depth() for m in rf_1.estimators_]
max(depths)
```
executed in 4ms, finished 20:52:18 2021-05-23

Out[801]: 77

OBSERVATIONS

- Vanilla RF model improved Recall by 5% and F1 by 6%. However, the model is overfit to the training data (1.0 vs .76 on recall). This is because I allowed the decision trees to go to completion.

ACTIONS

- I will perform a gridsearch to optimize hyperparameters to limit overfitting.

7.5.4 Model #2

In [802]:
```
#create RF model
rf_2 = RandomForestClassifier()
```
executed in 1ms, finished 20:52:18 2021-05-23

In [803]:
```
#create parameters for gridsearch
params = [{'n_estimators': [1200],
           'criterion': ['gini'],
           'max_features': ['auto'],
           'class_weight': ['balanced'],
           'max_depth':[65],
           'min_samples_split': [3],
           'min_samples_leaf': [1]}]
```
executed in 2ms, finished 20:52:18 2021-05-23

In [804]:
```
#create and fit gridsearch
rf_2_gridsearch = GridSearchCV(rf_2, params,scoring='recall',n_jobs=-1,
rf_2_gridsearch.fit(X_train_dt, y_train)
```
executed in 4m 54s, finished 20:57:12 2021-05-23

Out[804]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid=[{'class_weight': ['balanced'], 'criterion': ['gi
ni'],
                          'max_depth': [65], 'max_features': ['auto'],
                          'min_samples_leaf': [1], 'min_samples_split':
[3],
                          'n_estimators': [1200]}],
             scoring='recall')
```

7.5.5 Model Evaluation

In [805]:
```
#best parameters
rf_2_gridsearch.best_params_
```

executed in 15ms, finished 20:57:12 2021-05-23

Out[805]:
```
{'class_weight': 'balanced',
 'criterion': 'gini',
 'max_depth': 65,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 3,
 'n_estimators': 1200}
```

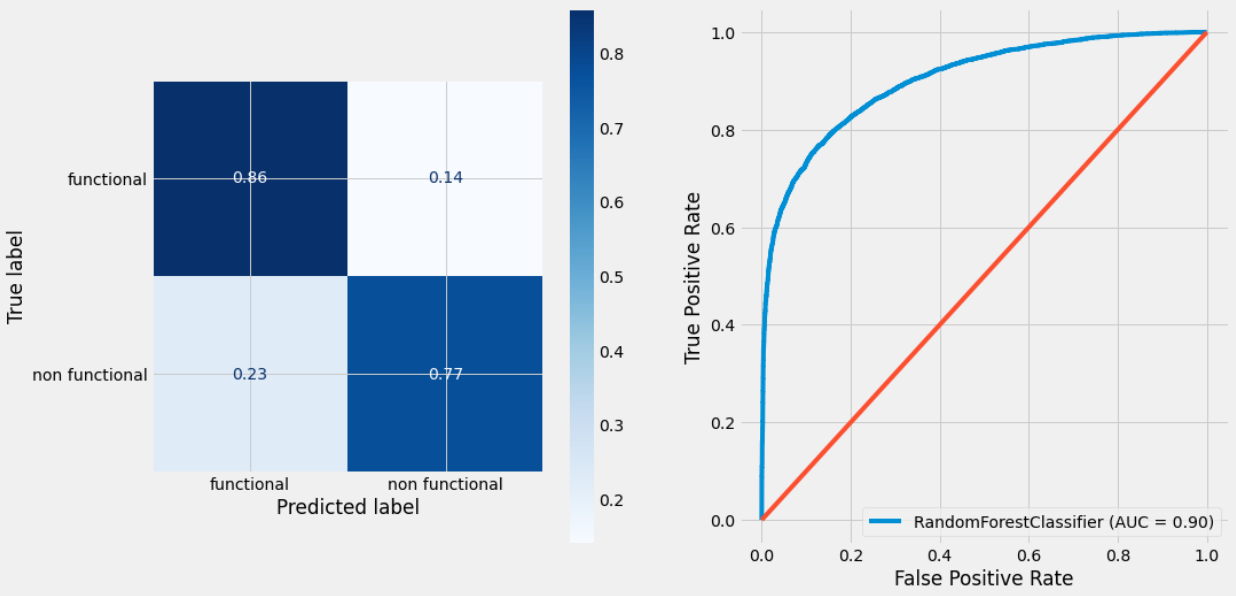
In [806]:
```
#evaluate model
model_eval(rf_2_gridsearch.best_estimator_, X_train_dt, y_train, X_test
           y_test, prev_model=rf_1)
```

executed in 43.2s, finished 20:57:55 2021-05-23

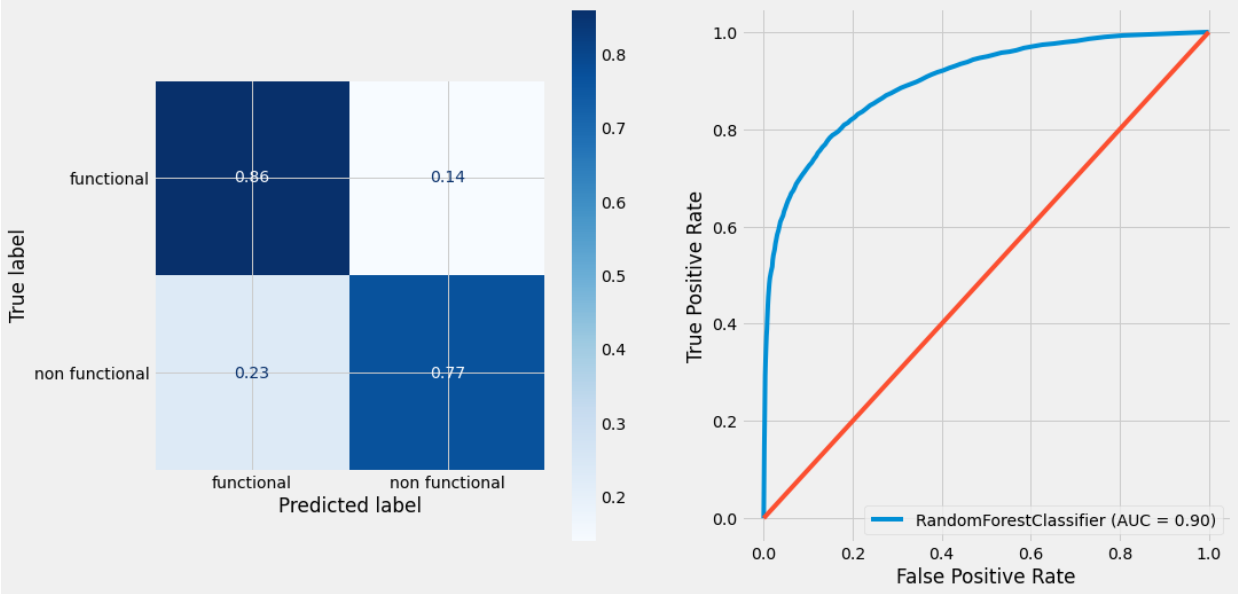
```
MODEL EVAL VS PREVIOUS (TEST)
================================================================
```

	Previous Model	Current Model	Delta
Recall	0.77	0.77	0.0
F1	0.8	0.8	0.0
Accuracy	0.82	0.82	0.0
AUC	0.9	0.9	0.0

```
CURRENT MODEL: Overfit (Recall)
================================================================


Recall on Training: 1.0
Recall on Test: 0.77


Classification Reports----------------------------------------
              precision    recall  f1-score   support

           0       0.82      0.86      0.84      9408
           1       0.82      0.77      0.80      7866

    accuracy                           0.82     17274
   macro avg       0.82      0.82      0.82     17274
weighted avg       0.82      0.82      0.82     17274

Test Graphs---------------------------------------------------
```



```
PREVIOUS MODEL
================================================================
```




```
<Figure size 432x288 with 0 Axes>
```

In [807]:
```
#get features
rf_2_fi = rf_2_gridsearch.best_estimator_.feature_importances_
```
executed in 179ms, finished 20:57:55 2021-05-23

In [808]:
```
#convert features into dataframe
rf_2_features = pd.DataFrame(data=rf_2_fi, index=X_train_dt.columns,
                             columns=['Feature Importances'])
rf_2_features_sorted = rf_2_features[:10].sort_values('Feature Importan
```
executed in 8ms, finished 20:57:55 2021-05-23

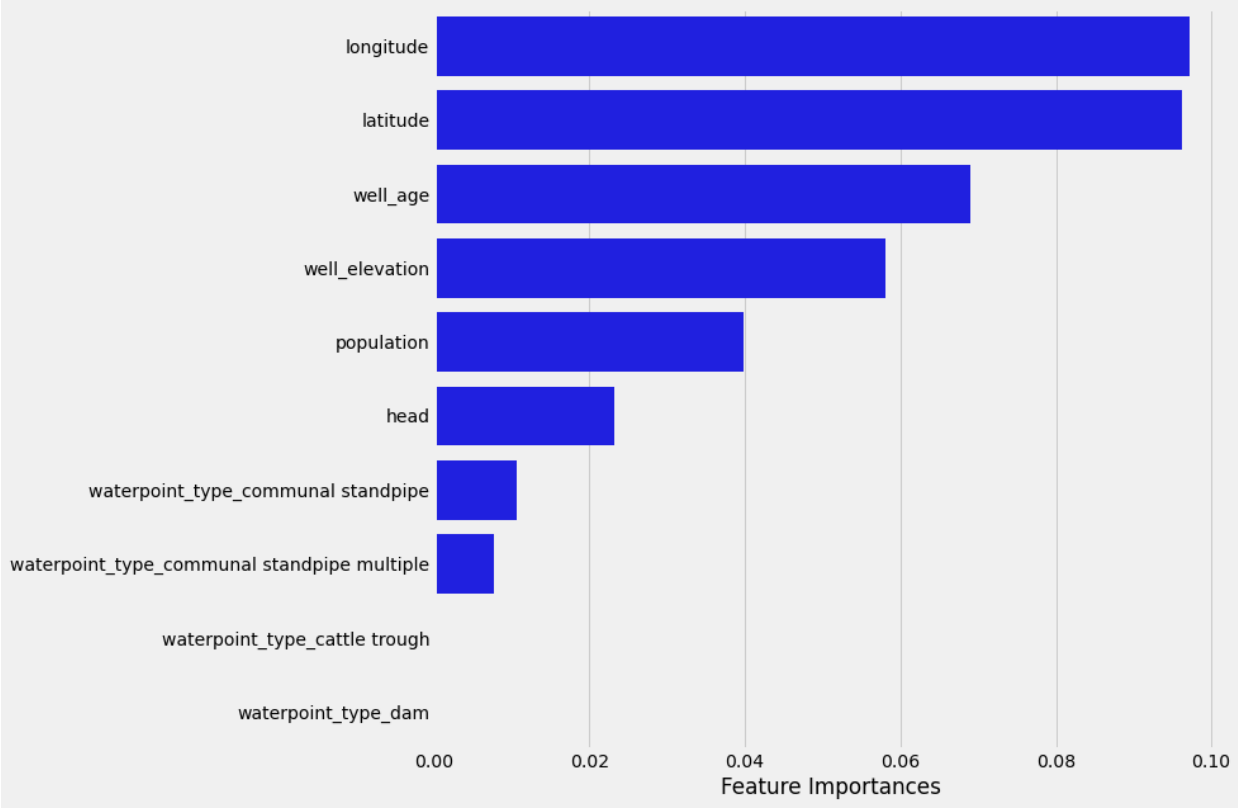
In [809]:
```
1 rf_2_features_sorted
```
executed in 6ms, finished 20:57:55 2021-05-23

Out[809]:

	Feature Importances
longitude	0.09726535199576049
latitude	0.0962103624358328
well_age	0.06905271345222457
well_elevation	0.05806936938815115
population	0.039821798923374924
head	0.02316975860274658
waterpoint_type_communal standpipe	0.01068801118045164
waterpoint_type_communal standpipe multiple	0.007755177084764425
waterpoint_type_cattle trough	0.00037045529106460526
waterpoint_type_dam	1.6186297717734977e-05

In [810]:
```
1 fig, ax = plt.subplots(figsize=(10,10))
2
3 sns.barplot(data=rf_2_features_sorted,
4             x=rf_2_features_sorted['Feature Importances'],
5             y=rf_2_features_sorted.index, orient='h', color='blue',ax=a
```
executed in 141ms, finished 20:57:56 2021-05-23

Out[810]: <AxesSubplot:xlabel='Feature Importances'>



In [811]:
```
# #export feature importances to Tableau
# rf_2_features_sorted.to_csv(r'Data/rf_feature_importances.csv', index
```

executed in 1ms, finished 20:57:56 2021-05-23

8 DATA INTERPRETATION

This dataset took a lot of cleaning, especially involving unknown values which were in the original dataset as zeros. After cleaning, a logistic regression and random forest classification model was trained to predict well status (functional or non-functional). Classification models were created and optimized for the recall of finding a non-functional well. As stated in the introduction, this is the most financially expensive option since we may determine a well is non-functional when it is functional, however, this is also the most effective options for ensuring citizens have a consistent water supply. The best model for Logistic Regression (model #3) after preprocessing the data to adhere to assumptions of linear models had a recall of 71%. I then developed a random forest model to try and improve the classification of recall for non-functional wells. The best random forest model, and the final model for this analysis, has a recall of 77%. This means the best model can find 77% of all non-functional wells. One of the benefits of choosing a Logistic Regression and Random Forest model is their interpretability. From the Random Forest model, I was able to extract what the most important features are for developing that model:

1. `latitude` and `longitude` shows how geography impacts well status
2. `well_age` shows that older wells tend to have more reliability issues
3. `well_elevation`, `population` and `head` are all impactful for determining well status

9 RECOMMENDATIONS AND CONCLUSIONS

Based on what the model showed were the most important features, I have 3 recommendations:

1. Because there are so many features which impact well reliability, the Tanzanian government should strive to develop a close working relationship with locals around water wells to ensure that when wells are malfunctioning or aren't supplying an adequate amount of water to fulfill the population demands.
2. Keep a close eye on wells after they hit age 24. The analysis shows that there is a higher likelihood of a well being non-functional after hitting age 24.
3. Improve data governance to increase data quality with respect to numerical features such as `head`, `well_elevation`, `population` and `construction_year`. Having accurate data will lead to a better classification model and a more efficient use of maintenance time and resources.

make docstring have args 47:30 on video of topic 29