phase-oo3-logistic-regression- / index.ipynb

jmbego locally worked on 05abdb8 · now

5296 lines (5296 loc) · 710 KB

phase-oo3-logistic-regression- / index.ipynb

Top

Preview Code Blame Raw

Tanzanian Water Wells Status Prediction

Overview

Tanzania is a developing country that struggles to get clean water to its population of 59 million people. According to WHO, 1 in 6 people in Tanzania lack access to safe drinking water and 29 million don't have access to improved sanitation. The focus of this project is to build a classification model to predict the functionality of waterpoints in Tanzania given data provided by Taarifa and the Tanzanian Ministry of Water. The model was built from a dataset containing information about the source of water and status of the waterpoint (functional, functional but needs repairs, and non functional) using an iterative approach and can be found here. The dataset contains 60,000 waterpoints in Tanzania and the following features:

- `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

- 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

The first sections focus on investigating, cleaning, wrangling, and reducing dimensionality for modeling. The next section contains 6 different classification models and evaluation of each, ultimately leading to us to select our best model for predicting waterpoint status based on the precision of the functional wells in the model. Finally, I will make recommendations to the Tanzanian Government and provide insight on predicting the status of waterpoints.

Business Problem

The Tanzanian government has a severe water crisis on their hands as a result of the vast number of non functional wells and they have asked for help. They want to be able to predict the statuses of which pumps are functional, functional but need repair, and non functional in order to improve their maintenance operations and ensure that it's residents have access to safe drinking water. The data has been collected by and is provided by Taarifa and the Tanzanian Ministry of Water with the hope that the information provided by each waterpoint can aid understanding in which waterpoints will fail.

I have partnered with the Tanzanian government to build a classification model to predict the status of the waterpoints using the dataset provided. I will use the precision of the functional wells as my main metric for model selection, as a non functional well being predicted as a functional well would be more detrimental to their case, but will provide and discuss several metrics for each model.

Project Objectives

1. Fix Waterpoints in Problem Areas:

Objective: Fix at least 30% of non-working waterpoints in the regions of Mtwara, Lindi, Mara, and Rukwa within the next year.

> ie . Send repair teams to these areas, make sure resources are used effectively, and set up systems to check the status of repairs regularly.

2. Repair and Maintain Waterpoints with Water:

Objective: Repair over 8,000 waterpoints that have water but are not working, and fix all the waterpoints needing repairs in Kigoma within 18 months.

> ie. Create and follow a plan to repair these waterpoints, start with those that already have water, and schedule regular checks to keep them working.

3. Improve Installer Performance and Payment:

Objective: Reduce pump failures caused by the government, district councils, and Fini Water by 25% in the next year.

> ie. Review why these installers have high failure rates, consider using different installers if needed, and look at how payment affects maintenance. Improve training and incentives for installers based on these findings.

Data Understanding

The dataset used for this analysis can be found here. It contains a wealth of information about waterpoints in Tanzania and the status of their operation. The target variable has 3 different options for it's status:

- `functional` - the waterpoint is operational and there are no repairs needed
- `functional needs repair` - the waterpoint is operational, but needs repairs
- `non functional` - the waterpoint is not operational

Below I will import the dataset and start my investigation of relevant information it may contain. Let's get started!

In [5]:

```
# Import standard packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_sc
from sklearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE, SMOTENC

# Classification Models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
#import xgboost as xgb
from sklearn.dummy import DummyClassifier
```

```
from sklearn.dummy import DummyClassifier
#from xgboost.sklearn import XGBClassifier

from sklearn.metrics import  accuracy_score, f1_score, precision_score, recall_s
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

# Scalers
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, label_binarize

# Categorical Create Dummies
from sklearn.preprocessing import OneHotEncoder
```

In [6]:

```
# Data Import Train Set
df_train_set = pd.read_csv('.\Data\Training set values.csv', index_col='id')
df_train_set
```

Out[6]:

id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitud
69572	6000.0	2011-03-14	Roman	1390	Roman	34.938093	-9.85632
8776	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.698766	-2.14746
34310	25.0	2013-02-25	Lottery Club	686	World vision	37.460664	-3.82132
67743	0.0	2013-01-28	Unicef	263	UNICEF	38.486161	-11.15529
19728	0.0	2011-07-13	Action In A	0	Artisan	31.130847	-1.82535
...	...	...	...	...	...	...	
60739	10.0	2013-05-03	Germany Republi	1210	CES	37.169807	-3.25384
27263	4700.0	2011-05-07	Cefa-njombe	1212	Cefa	35.249991	-9.07062
37057	0.0	2011-04-11	NaN	0	NaN	34.017087	-8.75043
31282	0.0	2011-03-08	Malec	0	Musa	35.861315	-6.37857
26348	0.0	2011-03-23	World Bank	191	World	38.104048	-6.74746

59400 rows × 39 columns

In [7]:
```
# Data import Training set labels
df_train_labels = pd.read_csv('.\Data\Training set labels.csv', index_col='id')
df_train_labels
```

Out[7]:

id	status_group
69572	functional
8776	functional
34310	functional
67743	non functional
19728	functional
...	...
60739	functional
27263	functional
37057	functional
31282	functional
26348	functional

59400 rows × 1 columns

In [8]:
```
#Merge datasets
df = pd.merge(df_train_labels, df_train_set, how = 'inner', on='id')
```

In [9]:
```
#Reset index
df.reset_index(inplace=True)
df.head()
```

Out[9]:

	id	status_group	amount_tsh	date_recorded	funder	gps_height	installer	long
0	69572	functional	6000.0	2011-03-14	Roman	1390	Roman	34.93
1	8776	functional	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.69
2	34310	functional	25.0	2013-02-25	Lottery Club	686	World vision	37.46

3	67743	non functional	0.0	2013-01-28	Unicef	263	UNICEF	38.48
4	19728	functional	0.0	2011-07-13	Action In A	0	Artisan	31.13

5 rows × 41 columns

In [10]:
```
# Check datatypes
df.info()
```

```
<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   status_group           59400 non-null  object 
 2   amount_tsh             59400 non-null  float64
 3   date_recorded          59400 non-null  object 
 4   funder                 55765 non-null  object 
 5   gps_height             59400 non-null  int64  
 6   installer              55745 non-null  object 
 7   longitude              59400 non-null  float64
 8   latitude               59400 non-null  float64
 9   wpt_name               59400 non-null  object 
 10  num_private            59400 non-null  int64  
 11  basin                  59400 non-null  object 
 12  subvillage             59029 non-null  object 
 13  region                 59400 non-null  object 
 14  region_code            59400 non-null  int64  
 15  district_code          59400 non-null  int64  
 16  lga                    59400 non-null  object 
 17  ward                   59400 non-null  object 
 18  population             59400 non-null  int64  
 19  public_meeting         56066 non-null  object 
 20  recorded_by            59400 non-null  object 
 21  scheme_management      55523 non-null  object 
 22  scheme_name            31234 non-null  object 
 23  permit                 56344 non-null  object 
 24  construction_year      59400 non-null  int64  
 25  extraction_type        59400 non-null  object 
 26  extraction_type_group  59400 non-null  object 
 27  extraction_type_class  59400 non-null  object 
 28  management             59400 non-null  object 
 29  management_group       59400 non-null  object 
 30  payment                59400 non-null  object 
 31  payment_type           59400 non-null  object 
 32  water_quality          59400 non-null  object 
 33  quality_group          59400 non-null  object 
 34  quantity               59400 non-null  object 
 35  quantity_group         59400 non-null  object 
 36  source                 59400 non-null  object 
 37  source_type            59400 non-null  object 
 38  source_class           59400 non-null  object 
 39  waterpoint_type        59400 non-null  object 
 40  waterpoint_type_group  59400 non-null  object 
```

```
dtypes: float64(3), int64(7), object(31)
memory usage: 18.6+ MB
```

In [11]:
```
#Get stats on numeric columns
df.describe()
```

Out[11]:

	id	amount_tsh	gps_height	longitude	latitude	num_pri
count	59400.000000	59400.000000	59400.000000	59400.000000	5.940000e+04	59400.00
mean	37115.131768	317.650385	668.297239	34.077427	-5.706033e+00	0.47
std	21453.128371	2997.574558	693.116350	6.567432	2.946019e+00	12.23
min	0.000000	0.000000	-90.000000	0.000000	-1.164944e+01	0.00
25%	18519.750000	0.000000	0.000000	33.090347	-8.540621e+00	0.00
50%	37061.500000	0.000000	369.000000	34.908743	-5.021597e+00	0.00
75%	55656.500000	20.000000	1319.250000	37.178387	-3.326156e+00	0.00
max	74247.000000	350000.000000	2770.000000	40.345193	-2.000000e-08	1776.00

In [12]:
```
#Check for duplicates
sum(df.duplicated())
```

Out[12]: 0

In [13]:
```
# Print all value counts to make observations
for col in df.columns:
    print(df[col].value_counts())
```

```
2047     1
72310    1
49805    1
51852    1
62091    1
        ..
46396    1
36155    1
34106    1
38200    1
0        1
Name: id, Length: 59400, dtype: int64
functional                 32259
non functional             22824
functional needs repair     4317
Name: status_group, dtype: int64
0.0         41639
500.0        3102
50.0         2472
1000.0       1488
20.0         1463
            ...
```

```
8500.0          1
6300.0          1
220.0           1
138000.0        1
12.0            1
Name: amount_tsh, Length: 98, dtype: int64
2011-03-15    572
2011-03-17    558
2013-02-03    546
2011-03-14    520
2011-03-16    513
              ...
2012-01-21      1
2011-09-19      1
2011-09-08      1
2002-10-14      1
2011-09-15      1
Name: date_recorded, Length: 356, dtype: int64
Government Of Tanzania    9084
Danida                    3114
Hesawa                    2202
Rwssp                     1374
World Bank                1349
                          ...
Rdws                         1
Mh.J S Sumari                1
Mgaya Masese                 1
Trc                          1
Mungaya                      1
Name: funder, Length: 1897, dtype: int64
 0       20438
-15         60
-16         55
-13         55
-20         52
          ...
 2285        1
 2424        1
 2552        1
 2413        1
 2385        1
Name: gps_height, Length: 2428, dtype: int64
DWE                   17402
Government             1825
RWE                    1206
Commu                  1060
DANIDA                 1050
                      ...
Simango Kihengu           1
Magul                     1
Nerthlands                1
DANIDS                    1
Pentecostal church        1
Name: installer, Length: 2145, dtype: int64
0.000000     1812
37.540901       2
33.010510       2
39.093484       2
32.972719       2
              ...
37.579803       1
```

```
33.196490       1
34.017119       1
33.788326       1
30.163579       1
Name: longitude, Length: 57516, dtype: int64
-2.000000e-08    1812
-6.985842e+00       2
-3.797579e+00       2
-6.981884e+00       2
-7.104625e+00       2
                  ...
-5.726001e+00       1
-9.646831e+00       1
-8.124530e+00       1
-2.535985e+00       1
-2.598965e+00       1
Name: latitude, Length: 57517, dtype: int64
none                      3563
Shuleni                   1748
Zahanati                   830
Msikitini                  535
Kanisani                   323
                          ...
Kwa Asajile Mwakwalaba       1
Chechenza A                  1
Kwa Shushe Lolouwo           1
Kwa Mzee Mgube               1
Kwa Bangaseo                 1
Name: wpt_name, Length: 37400, dtype: int64
0      58643
6         81
1         73
5         46
8         46
       ...
180        1
213        1
23         1
55         1
94         1
Name: num_private, Length: 65, dtype: int64
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
Madukani      508
Shuleni       506
Majengo       502
Kati          373
Mtakuja       262
              ...
Miyond          1
Kakerere        1
Liloya          1
```

```
Masemba         1
Nyitigasto      1
Name: subvillage, Length: 19287, dtype: int64
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
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
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
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
Igosi        307
Imalinyi     252
Siha Kati    232
Mdandu       231
Nduruma      217
            ...
Mawenzi        1
Uchindile      1
Chinugulu      1
Kinungu        1
Kitete         1
Name: ward, Length: 2092, dtype: int64
0       21381
1        7025
200      1940
150      1892
250      1681
        ...
3241        1
1960        1
1685        1
2248        1
1439        1
Name: population, Length: 1049, dtype: int64
True     51011
False     5055
Name: public_meeting, dtype: int64
GeoData Consultants Ltd    59400
Name: recorded_by, dtype: int64
VWC                 36793
WUG                  5206
Water authority      3153
WUA                  2883
Water Board          2748
Parastatal           1680
```

```
Private operator     1063
Company              1061
Other                 766
SWC                    97
Trust                  72
None                    1
Name: scheme_management, dtype: int64
K                            682
None                         644
Borehole                     546
Chalinze wate                405
M                            400
                            ...
Mradi wa maji wa izia          1
Rain water havest              1
Kasuguti irrigation scheme     1
Keza water supply              1
Bl Aziz water supply           1
Name: scheme_name, Length: 2696, dtype: int64
True     38852
False    17492
Name: permit, dtype: int64
0       20709
2010     2645
2008     2613
2009     2533
2000     2091
2007     1587
2006     1471
2003     1286
2011     1256
2004     1123
2012     1084
2002     1075
1978     1037
1995     1014
2005     1011
1999      979
1998      966
1990      954
1985      945
1980      811
1996      811
1984      779
1982      744
1994      738
1972      708
1974      676
1997      644
1992      640
1993      608
2001      540
1988      521
1983      488
1975      437
1986      434
1976      414
1970      411
1991      324
1989      316
```

```
1987      302
1981      238
1977      202
1979      192
1973      184
2013      176
1971      145
1960      102
1967       88
1963       85
1968       77
1969       59
1964       40
1962       30
1961       21
1965       19
1966       17
Name: construction_year, dtype: int64
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
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
gravity         26780
handpump        16456
other            6430
submersible      6179
motorpump        2987
rope pump         451
wind-powered      117
Name: extraction_type_class, dtype: int64
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
user-group    52490
commercial     3638
parastatal     1768
other           943
unknown         561
Name: management_group, dtype: int64
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
never pay     25348
per bucket     8985
monthly        8300
unknown        8157
on failure     3914
annually       3642
other          1054
Name: payment_type, dtype: int64
soft                  50818
salty                  4856
unknown                1876
milky                   804
coloured                490
salty abandoned         339
fluoride                200
fluoride abandoned       17
Name: water_quality, dtype: int64
good        50818
salty        5195
unknown      1876
milky         804
colored       490
fluoride      217
Name: quality_group, dtype: int64
enough          33186
insufficient    15129
dry              6246
seasonal         4050
unknown           789
Name: quantity, dtype: int64
enough          33186
insufficient    15129
dry              6246
seasonal         4050
unknown           789
```

```
Name: quantity_group, dtype: int64
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
spring                  17021
shallow well            16824
borehole                11949
river/lake              10377
rainwater harvesting     2295
dam                       656
other                     278
Name: source_type, dtype: int64
groundwater    45794
surface        13328
unknown          278
Name: source_class, dtype: int64
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
communal standpipe    34625
hand pump             17488
other                  6380
improved spring         784
cattle trough           116
dam                       7
Name: waterpoint_type_group, dtype: int64
```

In [14]:

```
# Check null values
df.isna().sum()
```

Out[14]:

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

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

In [15]:

```
# Check unique values for categorical data
obj_df = df.select_dtypes(include=['object'])
obj_df.nunique()
```

Out[15]:

```
status_group               3
date_recorded            356
funder                  1897
installer               2145
wpt_name               37400
basin                      9
subvillage             19287
region                    21
lga                      125
ward                    2092
public_meeting             2
recorded_by                1
scheme_management         12
scheme_name             2696
permit                     2
extraction_type           18
extraction_type_group     13
extraction_type_class      7
management                12
management_group           5
payment                    7
payment_type               7
water_quality              8
quality_group              6
quantity                   5
quantity_group             5
source                    10
source_type                7
source_class               3
```

```
waterpoint_type             7
waterpoint_type_group       6
dtype: int64
```

Initial Observations

Missing Values

scheme_name has the most missing values, followed by **funder**, **installer**, **public_meeting**, **scheme_management**, and **permit** with ~3,000 null values, and then **subvillage** with 371 null values. Several of these columns will be deleted as they appear to duplicate other columns, and I will investigate **installer**, **permit**, and **subvillage** further.

Data types

- **wpt_name**, **subvillage**, **ward**, **scheme_name**, **installer**, **funder**, and **date_recorded** are categorical features that have unique values in the thousands. This will be a problem with dummy variables, will likely remove or feature engineer.
- I will drop **recorded_by** as it has the same value for all rows.
- **num_private** is not defined on the DrivenData site, and it is not obvious what the feature indicates.
- **id** column will be dropped.
- **public_meeting** and **permit** are boolean.
- **construction_year**, **latitude**, **longitude**, **gps_height**, **amount_tsh**, and **population** all have thousands of rows of 0 entered. I will drop rows for most of these variables that have 0 entered, and will have to investigate further for real data on some columns.

Duplicate and Similar Data

The following columns all contain duplicate or similar data, will remove features that will cause multicollinearity:

- **extraction_type**, **extraction_type_group**, and **extraction_type_class**
- **payment** and **payment_type**
- **water_quality** and **quality_group**
- **quanitity** and **quantity_group**
- **source** and **source_type**
- **waterpoint_type** and **waterpoint_type_group**
- **region** and **region_code**

Data Cleaning

In this section, I will clean the dataset by removing similar and unnecessary columns and trim the dataset of remaining null values. I will also further investigate whether some columns contain the same information if it was not immediately obvious. There are several rows containing 0 entries in some column information. I will investigate whether I believe

rows containing 0 entries in some column information. I will investigate whether I believe the data to be real instead of a placeholder.

Drop duplicate and columns with similar information

I will keep **extraction_type_class** and remove **extraction_type** and **extraction_type_group** as it's columns values appear to be the most relevant for the project. **scheme_name** will be dropped for it's many null values. Other columns will be removed at this point due to irrelavancy, duplicates, null values, and some others will have to be investigated after the first drop.

In [16]:
```
# Columns to be dropped
dropped_columns = ['extraction_type', 'extraction_type_group', 'payment', 'schem
                   'quantity_group', 'source', 'waterpoint_type_group', 'recorde
                   'id', 'subvillage', 'wpt_name', 'ward', 'funder', 'date_recor
                   'region_code', 'district_code', 'lga', 'scheme_management', '
```

In [17]:
```
df = df.drop(dropped_columns, axis=1)
```

In [18]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59400 entries, 0 to 59399
Data columns (total 19 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   status_group           59400 non-null  object 
 1   amount_tsh             59400 non-null  float64
 2   gps_height             59400 non-null  int64  
 3   installer              55745 non-null  object 
 4   longitude              59400 non-null  float64
 5   latitude               59400 non-null  float64
 6   basin                  59400 non-null  object 
 7   region                 59400 non-null  object 
 8   population             59400 non-null  int64  
 9   permit                 56344 non-null  object 
 10  construction_year      59400 non-null  int64  
 11  extraction_type_class  59400 non-null  object 
 12  management             59400 non-null  object 
 13  management_group       59400 non-null  object 
 14  payment_type           59400 non-null  object 
 15  water_quality          59400 non-null  object 
 16  quantity               59400 non-null  object 
 17  source_type            59400 non-null  object 
 18  waterpoint_type        59400 non-null  object 
dtypes: float64(3), int64(3), object(13)
memory usage: 8.6+ MB
```

Dealing with null values

In [19]:
```
#Check for nulls
```

```
#Check for nulls
df.isna().sum()
```

Out[19]:
```
status_group               0
amount_tsh                 0
gps_height                 0
installer               3655
longitude                  0
latitude                   0
basin                      0
region                     0
population                 0
permit                  3056
construction_year          0
extraction_type_class      0
management                 0
management_group           0
payment_type               0
water_quality              0
quantity                   0
source_type                0
waterpoint_type            0
dtype: int64
```

In [20]:
```
# Drop all remaining null values from our dataset
df = df.dropna()
```

In [21]:
```
#Check to see that it worked
df.isna().sum()
```

Out[21]:
```
status_group            0
amount_tsh              0
gps_height              0
installer               0
longitude               0
latitude                0
basin                   0
region                  0
population              0
permit                  0
construction_year       0
extraction_type_class   0
management              0
management_group        0
payment_type            0
water_quality           0
quantity                0
source_type             0
waterpoint_type         0
dtype: int64
```

In [22]:
```
# Convert boolean permit to integers
df['permit'] = df['permit'].astype(int)
```

In [23]:
```
# Check to see that it worked
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 55102 entries, 0 to 59399
Data columns (total 19 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   status_group           55102 non-null  object
 1   amount_tsh             55102 non-null  float64
 2   gps_height             55102 non-null  int64
 3   installer              55102 non-null  object
 4   longitude              55102 non-null  float64
 5   latitude               55102 non-null  float64
 6   basin                  55102 non-null  object
 7   region                 55102 non-null  object
 8   population             55102 non-null  int64
 9   permit                 55102 non-null  int32
 10  construction_year      55102 non-null  int64
 11  extraction_type_class  55102 non-null  object
 12  management             55102 non-null  object
 13  management_group       55102 non-null  object
 14  payment_type           55102 non-null  object
 15  water_quality          55102 non-null  object
 16  quantity               55102 non-null  object
 17  source_type            55102 non-null  object
 18  waterpoint_type        55102 non-null  object
dtypes: float64(3), int32(1), int64(3), object(12)
memory usage: 8.2+ MB
```

Investigate management and management_group

I need to investigate these 2 columns further to see if they contain similar information.

In [24]:
```
df['management'].value_counts()
```

Out[24]:
```
vwc                37416
wug                 6314
water board         2705
wua                 2307
private operator    1891
parastatal          1588
water authority      825
other                733
company              656
unknown              491
other - school        99
trust                 77
Name: management, dtype: int64
```

In [25]:
```
df['management_group'].value_counts()
```

Out[25]:
```
user-group    48742
commercial     3449
parastatal     1588
other           832
unknown         491
Name: management_group, dtype: int64
```

The most data is contained in the user-group subcategory of **management_group**. I will groupby to investigate if the information is similar.

In [26]:
```
df.loc[df['management_group']=='user-group']['management'].value_counts()
```

Out[26]:
```
vwc            37416
wug             6314
water board     2705
wua             2307
Name: management, dtype: int64
```

The data is identical to the data contained in the management column in the subcategory of 'user-group'. I will drop **management_group** from our features.

In [27]:
```
#Drop column
df = df.drop('management_group', axis=1)
```

In [28]:
```
#Check to see that it worked
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 55102 entries, 0 to 59399
Data columns (total 18 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   status_group           55102 non-null  object 
 1   amount_tsh             55102 non-null  float64
 2   gps_height             55102 non-null  int64  
 3   installer              55102 non-null  object 
 4   longitude              55102 non-null  float64
 5   latitude               55102 non-null  float64
 6   basin                  55102 non-null  object 
 7   region                 55102 non-null  object 
 8   population             55102 non-null  int64  
 9   permit                 55102 non-null  int32  
 10  construction_year      55102 non-null  int64  
 11  extraction_type_class  55102 non-null  object 
 12  management             55102 non-null  object 
 13  payment_type           55102 non-null  object 
 14  water_quality          55102 non-null  object 
 15  quantity               55102 non-null  object 
 16  source_type            55102 non-null  object 
 17  waterpoint_type        55102 non-null  object 
dtypes: float64(3), int32(1), int64(3), object(11)
memory usage: 7.8+ MB
```

In [29]:
```
for col in df.columns:
    print(df[col].value_counts())
```

```
functional                29885
non functional            21381
functional needs repair    3836
Name: status_group, dtype: int64
0.0        37811
500.0       3071
```

```
50.0        2333
1000.0      1442
20.0        1427
           ...
38000.0        1
1400.0         1
8500.0         1
6300.0         1
26.0           1
Name: amount_tsh, Length: 95, dtype: int64
 0       18310
-15         54
 303        51
-16         51
-13         50
          ...
 2424        1
 2405        1
 2628        1
 2552        1
 2576        1
Name: gps_height, Length: 2426, dtype: int64
DWE                   17361
Government             1788
RWE                    1203
Commu                  1060
DANIDA                 1049
                      ...
Birage                    1
ter                       1
SDA CHURCH                1
Water use Group           1
Pentecostal church        1
Name: installer, Length: 2056, dtype: int64
0.000000     1793
32.984790       2
37.540901       2
37.328905       2
37.252194       2
             ...
39.002868       1
37.095964       1
36.658462       1
33.116994       1
38.592731       1
Name: longitude, Length: 53261, dtype: int64
-2.000000e-08    1793
-2.489378e+00       2
-2.519950e+00       2
-2.467137e+00       2
-6.956746e+00       2
                  ...
-6.405513e+00       1
-3.482213e+00       1
-9.438806e+00       1
-3.914628e+00       1
-2.598965e+00       1
Name: latitude, Length: 53263, dtype: int64
Lake Victoria              9705
Pangani                    8674
```

```
Rufiji                   7197
Internal                 6468
Lake Tanganyika          6406
Wami / Ruvu              5950
Ruvuma / Southern Coast  4481
Lake Nyasa               3769
Lake Rukwa               2452
Name: basin, dtype: int64
Iringa           5285
Shinyanga        4940
Kilimanjaro      4237
Morogoro         3995
Kagera           3224
Mwanza           3050
Arusha           2898
Kigoma           2805
Mbeya            2703
Ruvuma           2636
Tanga            2546
Pwani            2497
Dodoma           2199
Tabora           1942
Rukwa            1805
Mtwara           1725
Mara             1592
Manyara          1580
Lindi            1542
Singida          1124
Dar es Salaam     777
Name: region, dtype: int64
0       19250
1        6100
150      1854
200      1815
250      1605
        ...
406         1
1960        1
1685        1
2248        1
1439        1
Name: population, Length: 1026, dtype: int64
1    38195
0    16907
Name: permit, dtype: int64
0       18392
2008     2568
2009     2490
2010     2427
2000     1565
2007     1557
2006     1447
2003     1276
2011     1211
2004     1107
2002     1064
1978     1027
2012     1025
2005      983
1995      978
1999      950
```

```
1985      941
1998      921
1984      777
1996      766
1982      741
1972      705
1994      703
1974      675
1990      666
1980      647
1992      632
1997      612
1993      595
2001      530
1988      520
1983      487
1975      437
1986      431
1976      411
1991      322
1989      316
1970      310
1987      297
1981      237
1977      199
1979      192
1973      183
2013      173
1971      145
1963       84
1967       83
1968       68
1969       59
1960       45
1964       40
1962       29
1961       20
1965       19
1966       17
Name: construction_year, dtype: int64
gravity         24439
handpump        15779
other            5983
submersible      5759
motorpump        2689
rope pump         348
wind-powered      105
Name: extraction_type_class, dtype: int64
vwc                37416
wug                 6314
water board         2705
wua                 2307
private operator    1891
parastatal          1588
water authority      825
other                733
company              656
unknown              491
other - school        99
trust                 77
```

```
Name: management, dtype: int64
never pay     23097
per bucket     8666
monthly        8034
unknown        7021
on failure     3773
annually       3521
other           990
Name: payment_type, dtype: int64
soft                  47474
salty                  4652
unknown                1279
milky                   785
coloured                391
salty abandoned         329
fluoride                175
fluoride abandoned       17
Name: water_quality, dtype: int64
enough          31664
insufficient    13695
dry              5768
seasonal         3344
unknown           631
Name: quantity, dtype: int64
shallow well            16073
spring                  15792
borehole                10954
river/lake               9430
rainwater harvesting     1978
dam                       629
other                     246
Name: source_type, dtype: int64
communal standpipe             25551
hand pump                      16698
communal standpipe multiple     6012
other                           6004
improved spring                  745
cattle trough                     86
dam                                6
Name: waterpoint_type, dtype: int64
```

After our first round of cleaning, there are several features we need to examine further:

- **status_group** is an unbalanced target, may need to look into further during modeling and apply SMOTE.
- There are several columns with thousands of 0 entries - **amount_tsh**, **gps_height**, **longitude**, **latitude**, **population**, **construction_year**.

Construction year

In [30]:
```
df['construction_year'].value_counts()
```

Out[30]:
```
0       18392
2008     2568
2009     2490
2010     2427
2000     1565
```

```
2007     1557
2006     1447
2003     1276
2011     1211
2004     1107
2002     1064
1978     1027
2012     1025
2005      983
1995      978
1999      950
1985      941
1998      921
1984      777
1996      766
1982      741
1972      705
1994      703
1974      675
1990      666
1980      647
1992      632
1997      612
1993      595
2001      530
1988      520
1983      487
1975      437
1986      431
1976      411
1991      322
1989      316
1970      310
1987      297
1981      237
1977      199
1979      192
1973      183
2013      173
1971      145
1963       84
1967       83
1968       68
1969       59
1960       45
1964       40
1962       29
1961       20
1965       19
1966       17
Name: construction_year, dtype: int64
```

In [31]:

```
# Finding mean and median without zero values
df.loc[df['construction_year']!=0].describe()
```

Out[31]:

	amount_tsh	gps_height	longitude	latitude	population	per
count	36710.000000	36710.000000	36710.000000	36710.000000	36710.000000	36710.000

mean	471.881843	982.395015	36.015003	-6.358975	268.881694	0.717
std	3074.841656	623.784917	2.609370	2.762486	542.812926	0.450
min	0.000000	-63.000000	29.607122	-11.649440	0.000000	0.000
25%	0.000000	351.000000	34.671850	-8.855908	30.000000	0.000
50%	0.000000	1116.500000	36.691907	-6.351197	150.000000	1.000
75%	200.000000	1471.000000	37.896261	-3.731978	304.000000	1.000
max	250000.000000	2770.000000	40.345193	-1.042375	30500.000000	1.000

In [32]:
```
#Replace 0 values in construction_year with 1950 to aid visualization
df['construction_year'].replace(to_replace = 0, value = 1950, inplace=True)
```

In [33]:
```
#Check to see if it worked
df['construction_year'].value_counts()
```

Out[33]:
```
1950    18392
2008     2568
2009     2490
2010     2427
2000     1565
2007     1557
2006     1447
2003     1276
2011     1211
2004     1107
2002     1064
1978     1027
2012     1025
2005      983
1995      978
1999      950
1985      941
1998      921
1984      777
1996      766
1982      741
1972      705
1994      703
1974      675
1990      666
1980      647
1992      632
1997      612
1993      595
2001      530
1988      520
1983      487
1975      437
1986      431
1976      411
1991      322
1989      316
```

```
1970      310
1987      297
1981      237
1977      199
1979      192
1973      183
2013      173
1971      145
1963       84
1967       83
1968       68
1969       59
1960       45
1964       40
1962       29
1961       20
1965       19
1966       17
Name: construction_year, dtype: int64
```

It is unfortunate that there are 19,000 entries with 0 for the **construction_year**. These may be natural and spring fed sources that were never "constructed". I chose to replace the 0 values with 1950, so they are still the "oldest" in the dataset, but will aid in visualizing the functionality of the pumps by the year they were made.

Latitude/Longitude zeros

In [34]:
```
df.longitude.value_counts()
```

Out[34]:
```
0.000000     1793
32.984790       2
37.540901       2
37.328905       2
37.252194       2
               ...
39.002868       1
37.095964       1
36.658462       1
33.116994       1
38.592731       1
Name: longitude, Length: 53261, dtype: int64
```

In [35]:
```
# Investigate longitude entries that are 0
df.loc[df['longitude'] == 0]
```

Out[35]:

	status_group	amount_tsh	gps_height	installer	longitude	latitude	basi
21	functional	0.0	0	DWE	0.0	-2.000000e-08	Lak Victor
53	non functional	0.0	0	Government	0.0	-2.000000e-08	Lak Victor
168	functional	0.0	0	WVT	0.0	-2.000000e-	Lak

							08 Victor
177	non functional	0.0	0	DWE	0.0	-2.000000e-08	Lak Victor
253	functional needs repair	0.0	0	DWE	0.0	-2.000000e-08	Lak Victor
...	...	...	...	...	...	...	
59189	functional needs repair	0.0	0	DWE	0.0	-2.000000e-08	Lak Victor
59208	functional	0.0	0	DWE	0.0	-2.000000e-08	Lak Victor
59295	functional needs repair	0.0	0	DWE	0.0	-2.000000e-08	Lak Victor
59324	functional	0.0	0	World Vision	0.0	-2.000000e-08	Lak Victor
59374	functional	0.0	0	DWE	0.0	-2.000000e-08	Lak Victor

1793 rows × 18 columns

The 0s that are entered into the longitude column are also 0s in gps_height and -2e8 for latitude columns. I will drop these values from the dataset.

In [36]:
```
# Drop rows with 0 entered in longitude column
df = df.loc[df['longitude'] != 0]
```

In [37]:
```
# Check to see if it worked
df.describe()
```

Out[37]:

	amount_tsh	gps_height	longitude	latitude	population	per
count	53309.000000	53309.000000	53309.000000	53309.000000	53309.000000	53309.000
mean	337.580181	692.509670	35.186804	-5.849440	188.814515	0.702
std	2714.547122	691.264883	2.670974	2.806529	474.147131	0.457
min	0.000000	-90.000000	29.607122	-11.649440	0.000000	0.000
25%	0.000000	0.000000	33.167340	-8.441371	0.000000	0.000
50%	0.000000	438.000000	35.295878	-5.144420	45.000000	1.000
75%	40.000000	1322.000000	37.353028	-3.359390	240.000000	1.000
max	250000.000000	2770.000000	40.345193	-0.998464	30500.000000	1.000

In [38]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53309 entries, 0 to 59399
Data columns (total 18 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   status_group           53309 non-null  object 
 1   amount_tsh             53309 non-null  float64
 2   gps_height             53309 non-null  int64  
 3   installer              53309 non-null  object 
 4   longitude              53309 non-null  float64
 5   latitude               53309 non-null  float64
 6   basin                  53309 non-null  object 
 7   region                 53309 non-null  object 
 8   population             53309 non-null  int64  
 9   permit                 53309 non-null  int32  
 10  construction_year      53309 non-null  int64  
 11  extraction_type_class  53309 non-null  object 
 12  management             53309 non-null  object 
 13  payment_type           53309 non-null  object 
 14  water_quality          53309 non-null  object 
 15  quantity               53309 non-null  object 
 16  source_type            53309 non-null  object 
 17  waterpoint_type        53309 non-null  object 
dtypes: float64(3), int32(1), int64(3), object(11)
memory usage: 7.5+ MB
```

Looks like it all worked! I believe the **amount_tsh** and **population** 0 values are real so I will leave all data as is for vanilla models.

Installer - Several different spellings for same installer

In [39]:
```
#Check unique values after inital cleaning
df.nunique()
```

Out[39]:
```
status_group               3
amount_tsh                95
gps_height              2426
installer               2024
longitude              53260
latitude               53262
basin                      9
region                    21
population              1026
permit                     2
construction_year         55
extraction_type_class      7
management                12
payment_type               7
water_quality              8
quantity                   5
source_type                7
waterpoint_type            7
dtype: int64
```

Upon checking the unique values for our categorical variables after trimming the dataset,

installer still has 2024 unique entries, which will be a problem when we create dummies. We will need to cut down the amount of unique entries to not overload our model.

In [40]:
```
#Investigate 2024 unique values for installer
# pd.set_option("display.max_rows", None)
df['installer'].value_counts()
```

Out[40]:
```
DWE                     16214
Government               1633
RWE                      1178
Commu                    1060
DANIDA                   1049
                        ...
Hilfe Fur Bruder            1
ter                         1
Mama Kapwapwa               1
Ramadhani M. Mvugalo        1
Pentecostal church          1
Name: installer, Length: 2024, dtype: int64
```

There are several entries with typos and different variations of the same installer. I will attempt to fix some of the clerical errors and narrow down the amount of unique identifiers we will use for our model.

In [41]:
```
# Correct variations and misspellings in the installer column
df['installer'] = df['installer'].replace(to_replace = ('Central government', 'T
                                          'Cental Government','Tanzania governme
                                          'Centra Government', 'central governme
                                          'TANZANIA GOVERNMENT','Central govt',
                                          'Tanzanian Government', 'Tanzania'), v

df['installer'] = df['installer'].replace(to_replace = ('District COUNCIL', 'DIS
                                          'Counc','District council','District C
                                          'Council', 'COUN', 'Distri', 'District
                                          value = 'District Council')

df['installer'] = df['installer'].replace(to_replace = ('villigers', 'villager',
                                          'Villi', 'Village Council', 'Village C
                                          'Village community', 'Villaers', 'Vill
                                          'Villege Council', 'Village council',
                                          'Villager', 'Village Technician', 'Vil
                                          'Village community members', 'VILLAG',
                                          'Village govt', 'VILLAGERS', 'Village

df['installer'] = df['installer'].replace(to_replace = ('District Water Departme
                                          'Distric Water Department'), value ='D

df['installer'] = df['installer'].replace(to_replace = ('FinW', 'Fini water', 'F
                                          'Finwater', 'FINN WATER', 'FinW', 'FW'
                                          value ='Fini Water')

df['installer'] = df['installer'].replace(to_replace = ('RC CHURCH', 'RC Churc',
                                          'RC church', 'RC CATHORIC', 'Ch') , va

df['installer'] = df['installer'].replace(to_replace = ('world vision', 'World D
                                          'WORLD VISION', 'World Vission'), valu
```

```
df['installer'] = df['installer'].replace(to_replace = ('Unisef','Unicef'), valu

df['installer'] = df['installer'].replace(to_replace = 'DANID', value ='DANIDA')

df['installer'] = df['installer'].replace(to_replace =('Commu', 'Communit', 'com
                                          'Adra /Community', 'Communit', 'Adra/C
                                          value ='Community')

df['installer'] = df['installer'].replace(to_replace = ('GOVERNMENT', 'GOVER', '
                                          'Gover', 'Gove', 'Governme', 'Governme

df['installer'] = df['installer'].replace(to_replace = ('Hesawa', 'hesawa'), val

df['installer'] = df['installer'].replace(to_replace = ('JAICA', 'JICA', 'Jica',
                                          value ='Jaica')

df['installer'] = df['installer'].replace(to_replace = ('KKKT _ Konde and DWE',
                                          value ='KKKT')

df['installer'] = df['installer'].replace(to_replace = '0', value ='Unknown')
```

In [42]:

```
df['installer'].value_counts().head(20)
```

Out[42]:

```
DWE                   16214
Government             2468
Community              1791
DANIDA                 1601
HESAWA                 1180
RWE                    1178
District Council       1173
Central Government     1115
KKKT                   1102
Fini Water              952
Unknown                 780
TCRS                    702
World Vision            660
CES                     610
RC Church               484
Villagers               482
LGA                     408
WEDECO                  397
TASAF                   371
Jaica                   358
Name: installer, dtype: int64
```

Reduce Dimensionality for Installer

In [43]:

```
# Keep only top 20 installers as unique values
installer_20 = df.installer.value_counts(normalize=True).head(20).index.tolist()


df['installer'] = [type_ if type_ in installer_20
                   else "OTHER" for type_ in df['installer']]
```

In [44]:
```
df.installer.value_counts()
```

Out[44]:
```
OTHER                19283
DWE                  16214
Government            2468
Community             1791
DANIDA                1601
HESAWA                1180
RWE                   1178
District Council      1173
Central Government    1115
KKKT                  1102
Fini Water             952
Unknown                780
TCRS                   702
World Vision           660
CES                    610
RC Church              484
Villagers              482
LGA                    408
WEDECO                 397
TASAF                  371
Jaica                  358
Name: installer, dtype: int64
```

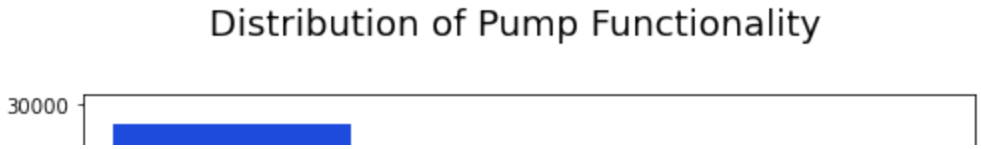
To reduce the dimensionality of the dataset, I made an "Other" category for installer if they were not in the top 20 installers of the dataset.

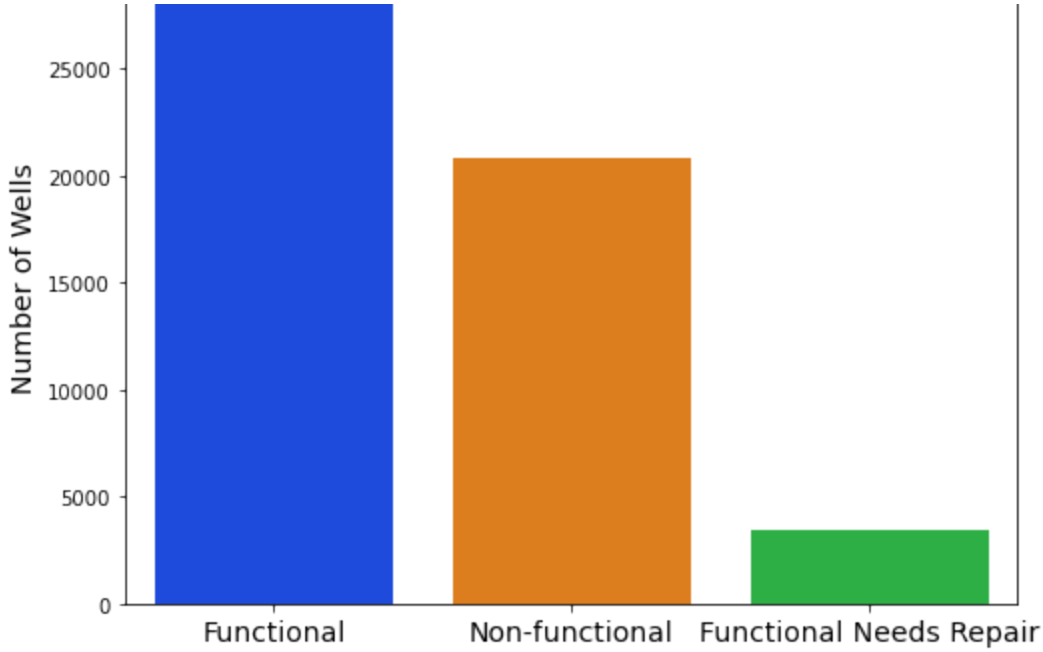
Modified Features Exploration

Column EDA

Target Feature Distribution

In [45]:
```
# plot
fig, ax = plt.subplots(figsize=(8,6))
ax = sns.countplot(x='status_group', palette='bright', data=df)

# Title and y-axis labels
fig.suptitle('Distribution of Pump Functionality', fontsize=18)
plt.ylabel('Number of Wells', fontsize=14)
plt.xlabel(' ', fontsize=14)

# X Labels
labels = ['Functional', 'Non-functional', 'Functional Needs Repair']
plt.xticks(ticks=(0,1,2), labels = labels, rotation = 0, size = 14)

plt.show();
```





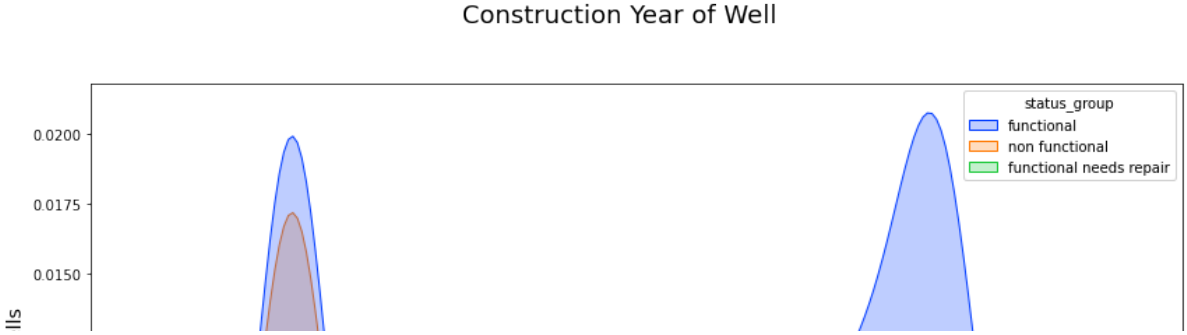
In [46]:
```
# Numner of wells in each category
df['status_group'].value_counts()
```

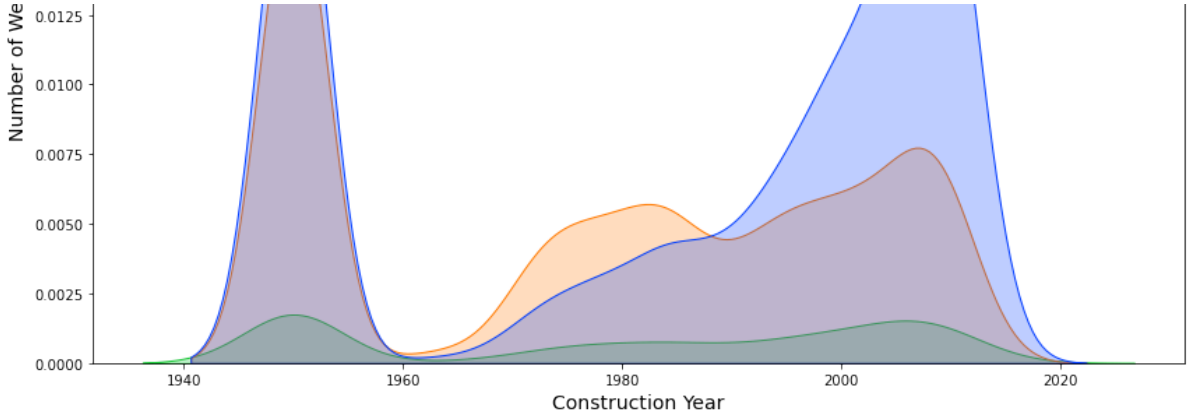
Out[46]:
```
functional                 29026
non functional             20829
functional needs repair     3454
Name: status_group, dtype: int64
```

We have the most functional wells at ~29,000, followed by non functional wells at ~21,000, and the minority class, functional needs repair at ~3,500.

Construction year

In [47]:
```
# Plot
fig, ax = plt.subplots(figsize=(14,8))
ax = sns.kdeplot(data=df, x='construction_year', hue='status_group', palette='br

# Title and axis labels
fig.suptitle('Construction Year of Well', fontsize=18)
plt.xlabel("Construction Year", fontsize=14)
plt.ylabel("Number of Wells", fontsize=14)
plt.show();
```

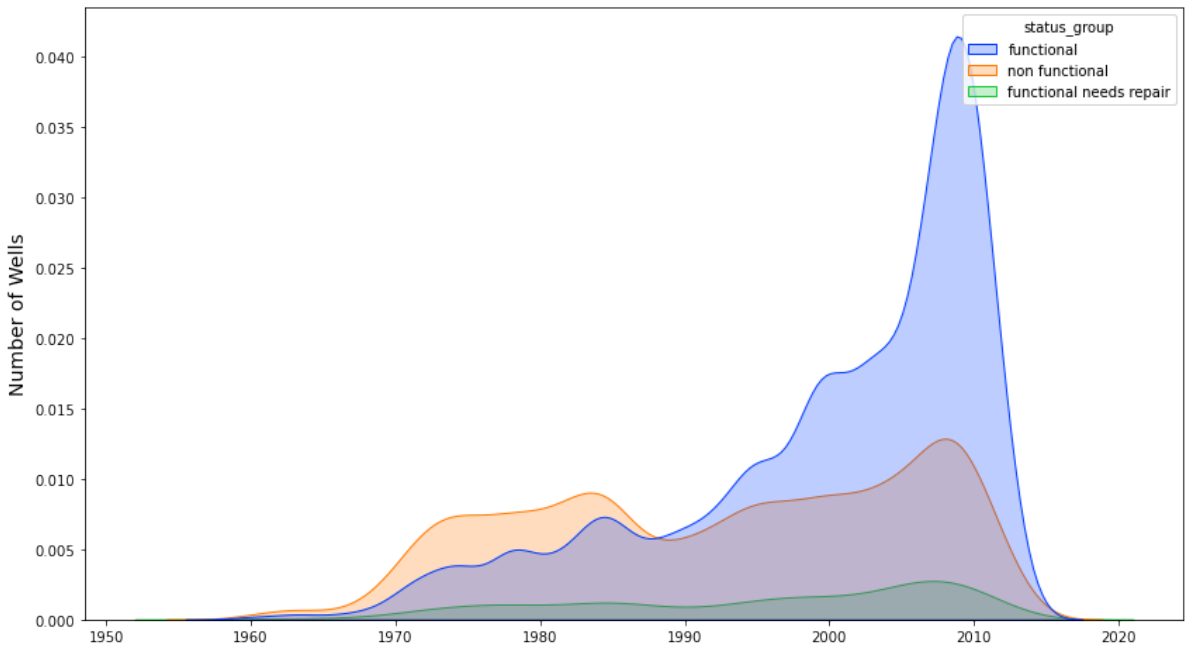




There is large amount of data in the year 1950 which were entered as 0 in the dataset, these may be natural sources and our distribution is normal for these sources. However, we can see the correlation of an older pump being more likely to be non functional and more functional newer pumps.

In [48]:
```
# Create df without entries in year 1950
const_year_df = df.loc[df['construction_year'] != 1950]
```

In [49]:
```
# Plot without waterpoints from year 1950
fig, ax = plt.subplots(figsize=(14,8))
ax = sns.kdeplot(data=const_year_df, x='construction_year', hue='status_group',

# Title and axis labels
fig.suptitle('Construction Year of Well', fontsize=18)
plt.xlabel("Construction Year", fontsize=14)
plt.ylabel("Number of Wells", fontsize=14)

plt.show();
```



Construction Year

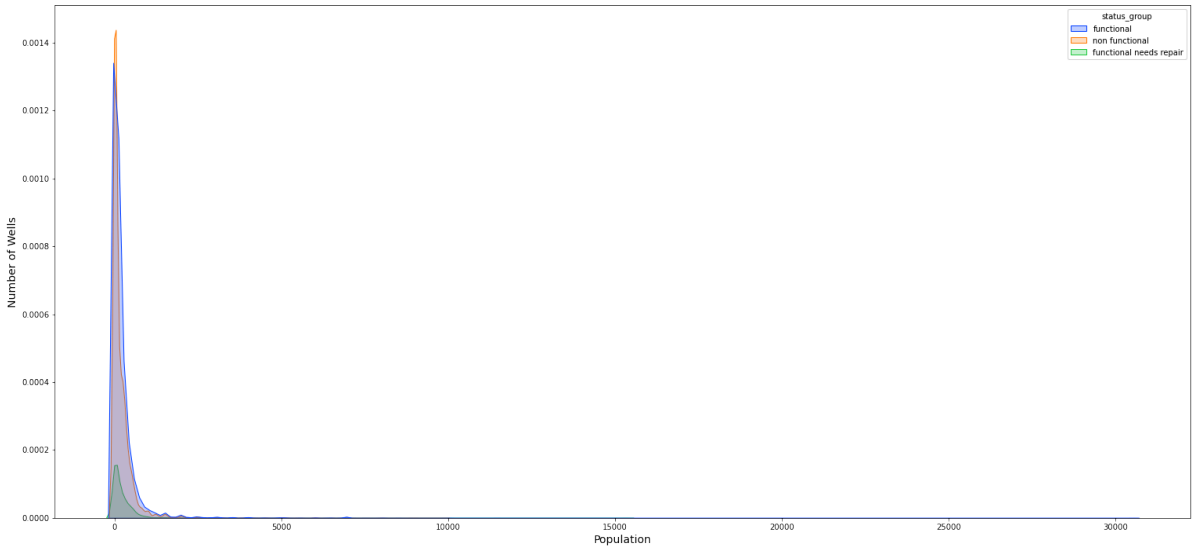
There are more non functional pumps than functional if they were built before 1988, but the rate of functionality keeps increasing after 1988.

Population

In [50]:

```
# Plot
fig, ax = plt.subplots(figsize=(26,12))
ax = sns.kdeplot(data=df, x='population', hue='status_group', palette='bright',

# Title and axis labels
fig.suptitle('Population at Wellsite', fontsize=18)
plt.xlabel("Population", fontsize=14)
plt.ylabel("Number of Wells", fontsize=14)
plt.show()
```

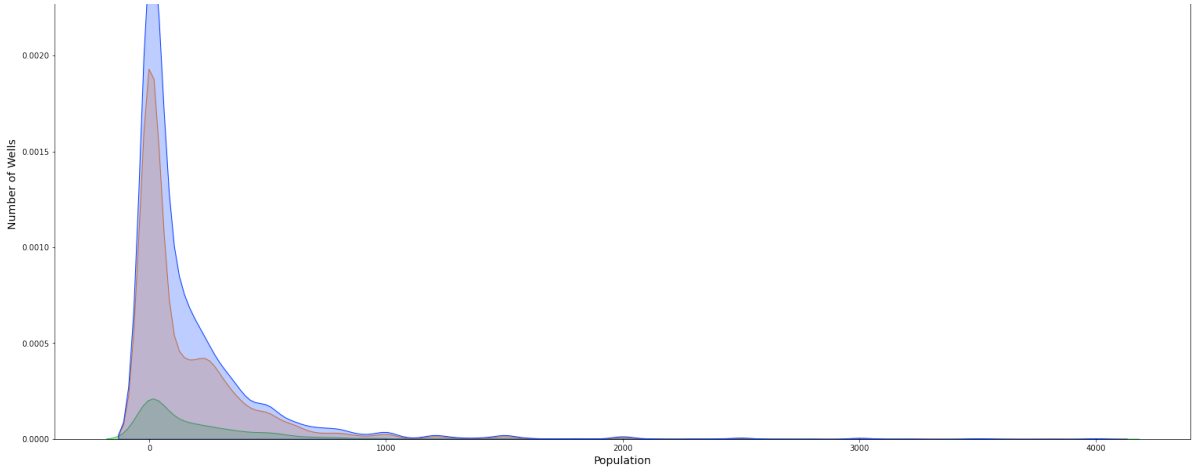
Population at Wellsite



In [51]:

```
# Get a closer look
pop_df = df.loc[df['population'] <= 4000]

# Plot
fig, ax = plt.subplots(figsize=(26,12))
ax = sns.kdeplot(data=pop_df, x='population', hue='status_group', palette='brigh

# Title and axis labels
fig.suptitle('Population at Wellsite', fontsize=18)
plt.xlabel("Population", fontsize=14)
plt.ylabel("Number of Wells", fontsize=14)
plt.show()
```

Population at Wellsite

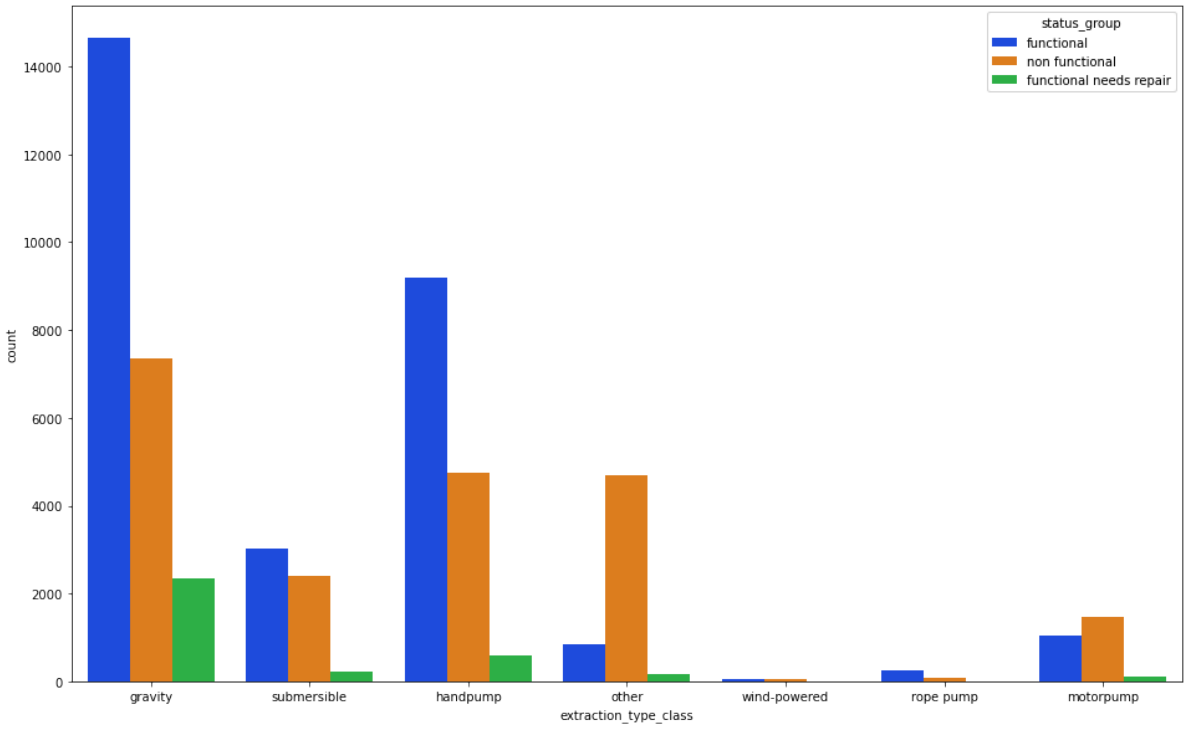


Overall, the distribution of pump functionality is similar across all population ranges and there isn't a lot of separation, with there being more functional wells than any other class. There isn't too much to draw from these graphs about population and functionality.

Extraction_type_class

In [52]:

```
# Plot
plt.figure(figsize=(16,10))
ax = sns.countplot(x='extraction_type_class', hue='status_group', palette='brigh
```

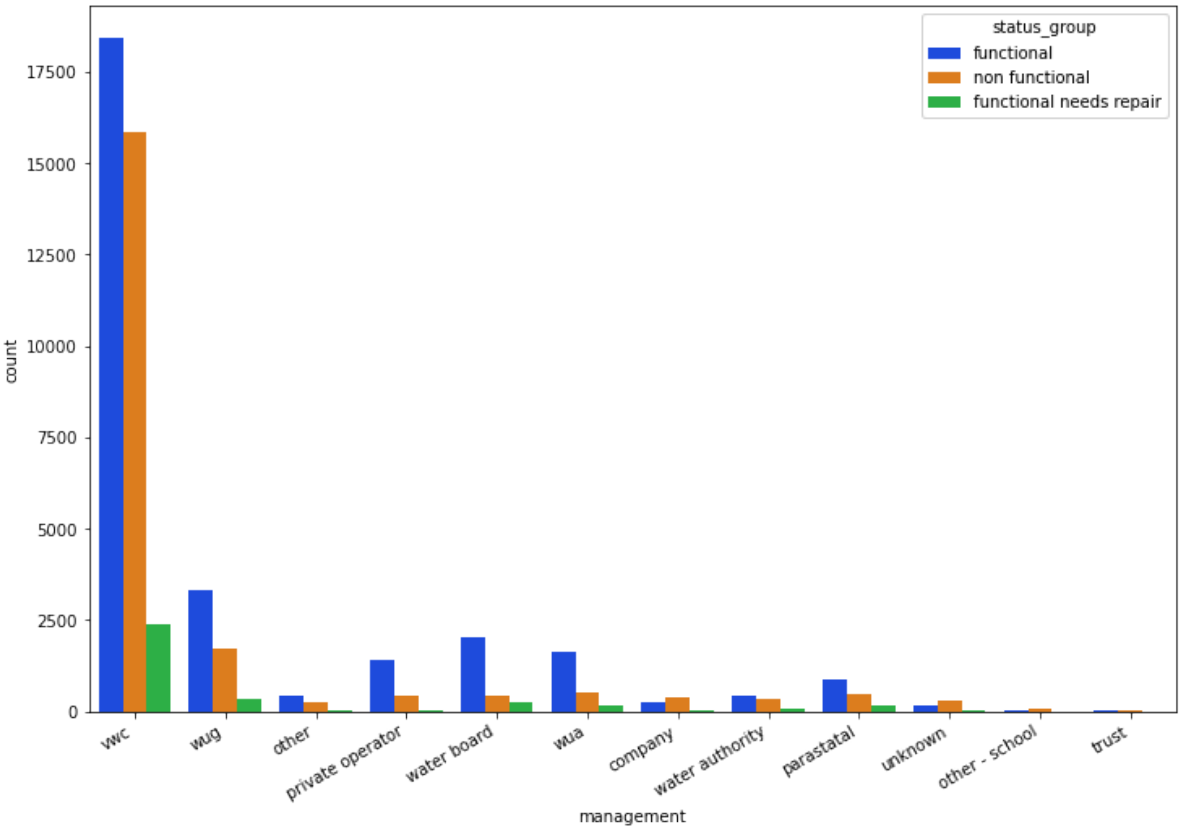


Other type and motorpump are especially non functioning. Gravity and handpump are the 2 largest types, and both have more functioning, but half non functioning.

Management

In [53]:

```
# Plot
plt.figure(figsize=(12,8))
ax = sns.countplot(x='management', hue='status_group', palette='bright', data=df
plt.xticks(rotation=30, ha='right');
```

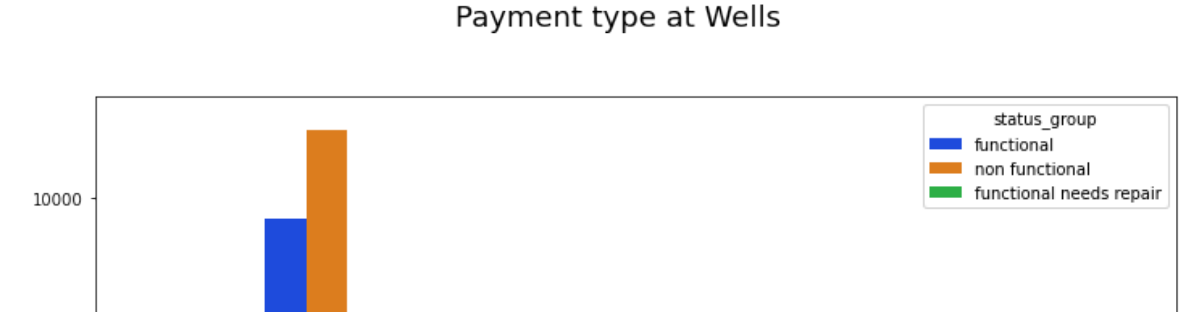


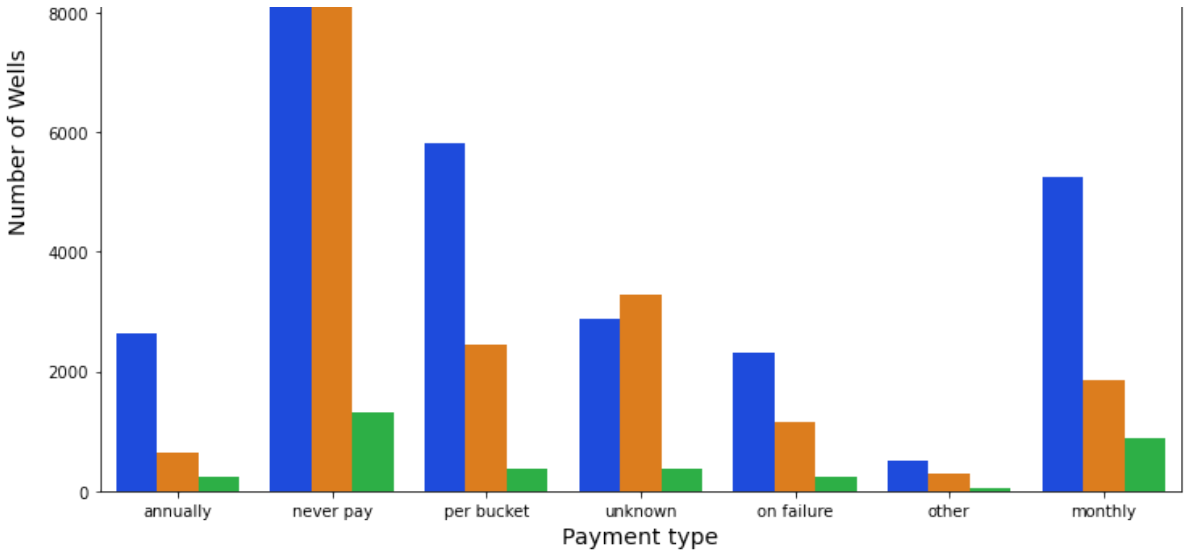
water board, wua, and private operators have a high rate of functionality.

Payment_type

In [54]:

```
# Plot
fig, ax = plt.subplots(figsize=(12,8))
ax = sns.countplot(x='payment_type', hue="status_group", palette='bright', data=

# Title and axis labels
fig.suptitle('Payment type at Wells', fontsize=18)
plt.xlabel("Payment type", fontsize=14)
plt.ylabel("Number of Wells", fontsize=14)
plt.show();
```

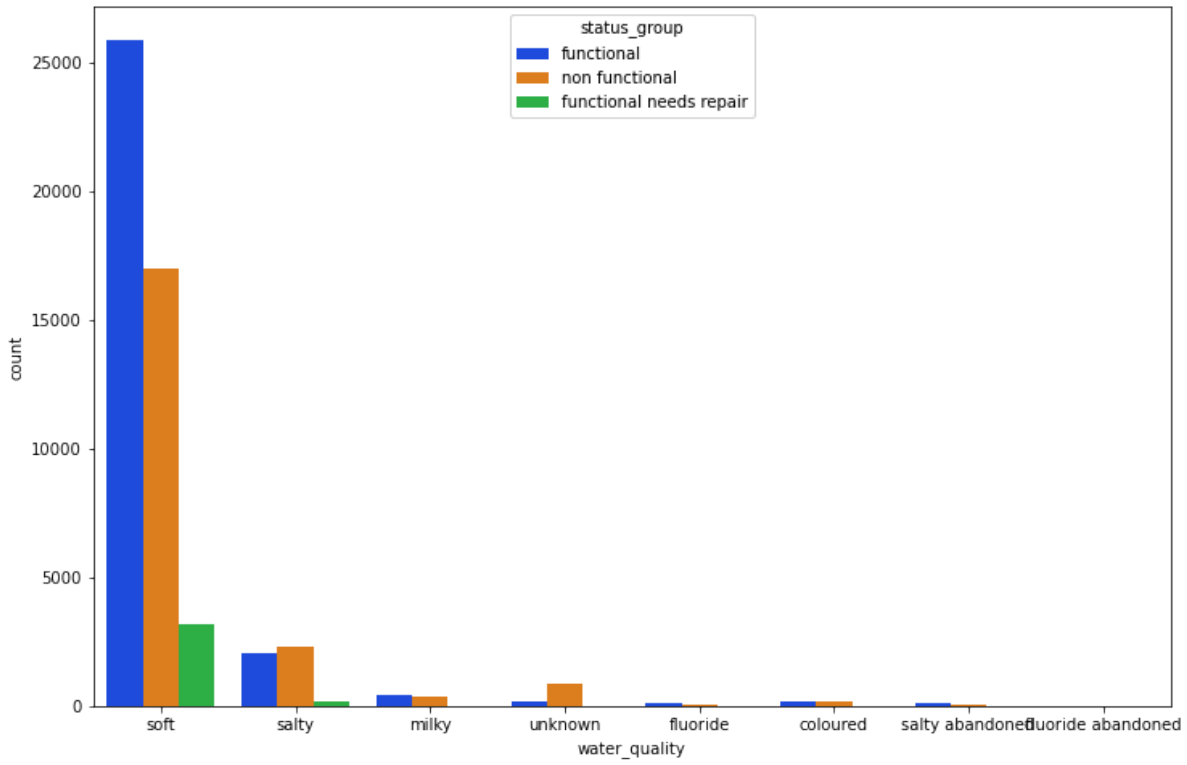




Never pay pumps have more non functioning waterpoints than functioning waterpoints. Some form of payment increases the functionality of the waterpoints.

Water quality

In [55]:
```
# Plot
plt.figure(figsize=(12,8))
ax = sns.countplot(x='water_quality', hue='status_group', palette='bright', data
```

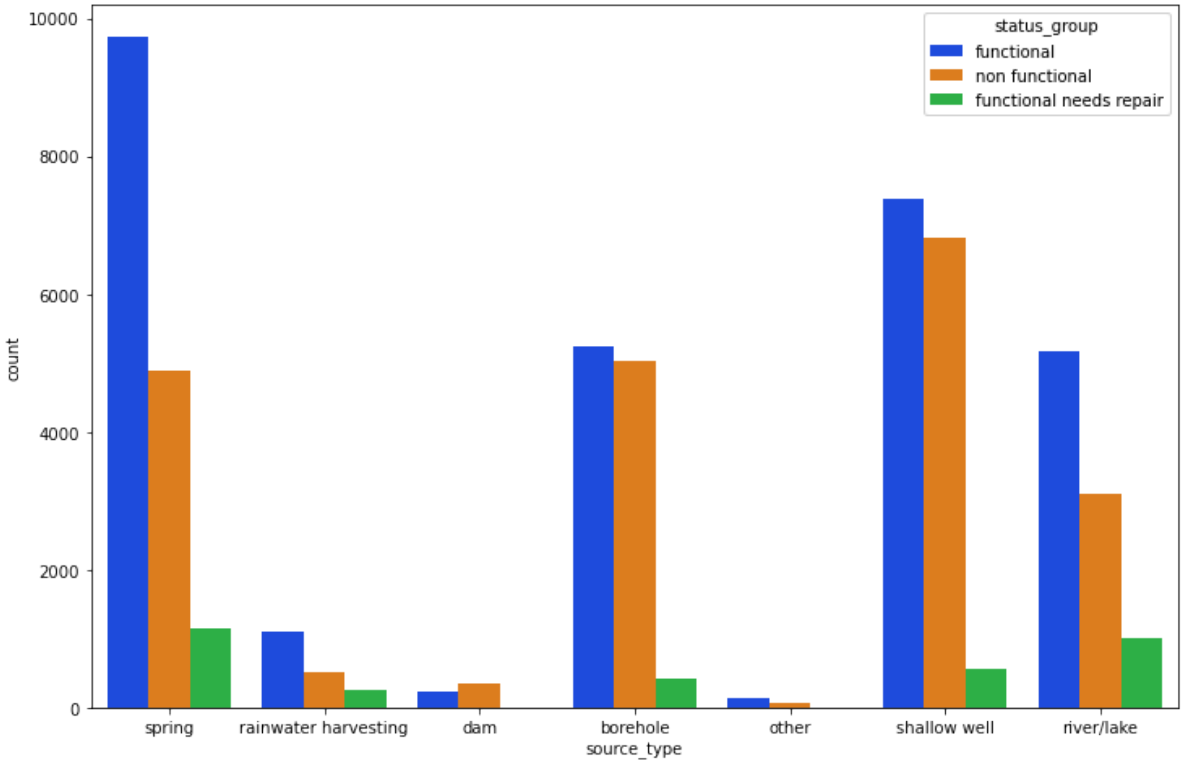


Soft water quality has a high rate of functional waterpoints, salty has a high rate of non functional waterpoints.

Source type

Source type

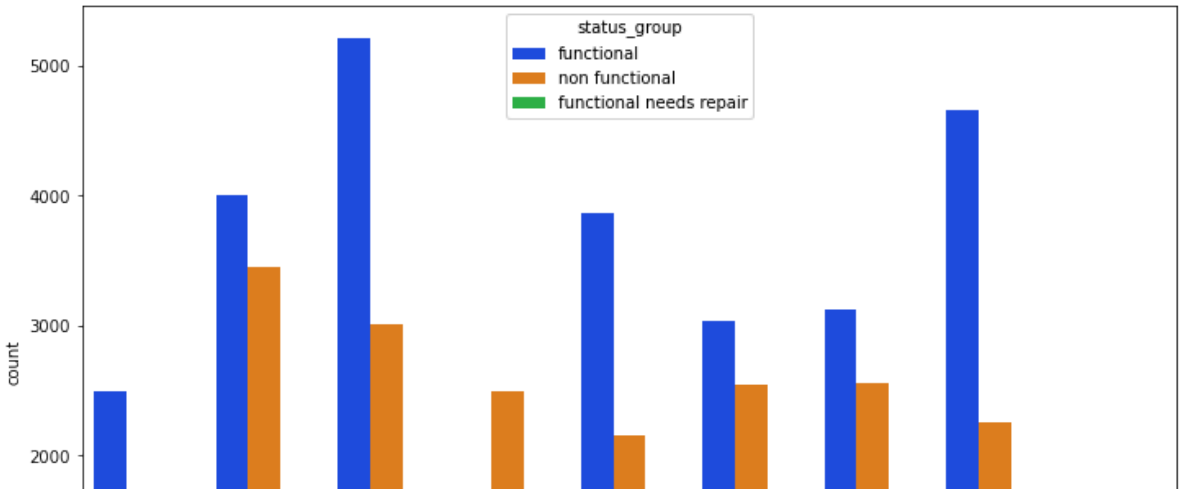
In [56]:
```
# Plot
plt.figure(figsize=(12,8))
ax = sns.countplot(x='source_type', hue='status_group', palette='bright', data=d
```

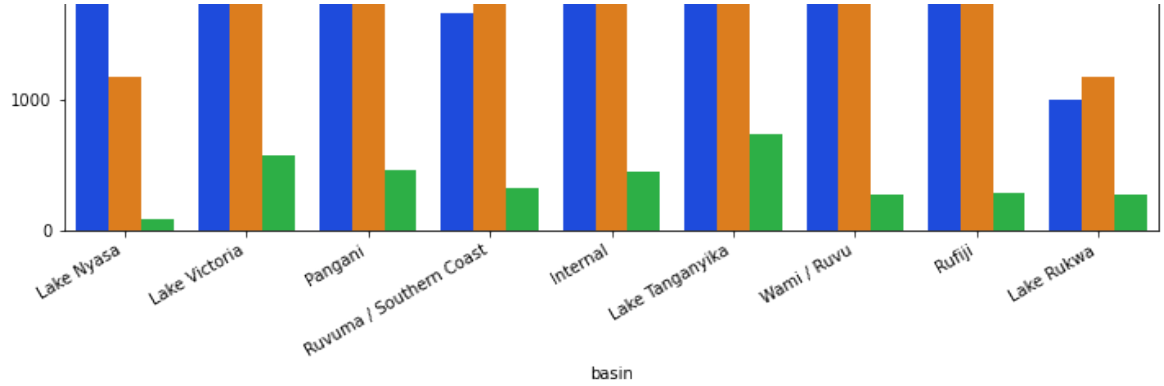


Even distribution of functional and nonfunctional boreholes. Many more functional springs and rivers than non functional.

Basin

In [57]:
```
# Plot
plt.figure(figsize=(12,8))
ax = sns.countplot(x='basin', hue='status_group', palette='bright', data=df)
plt.xticks(rotation=30, ha='right');
```

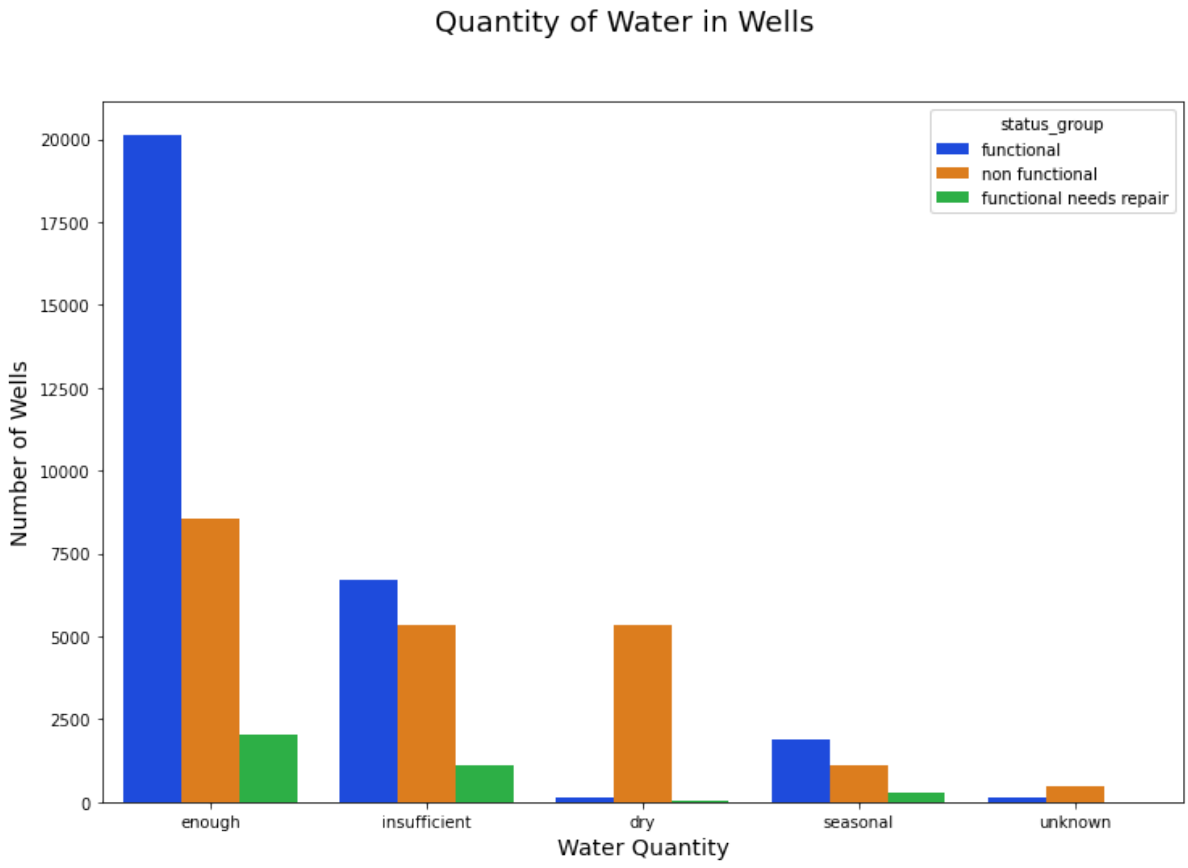




The Ruvuma/Southern Coast and Lake Rukwa basins have more non functioning wells than functional.

Quantity

In [58]:
```
# Plot
fig, ax = plt.subplots(figsize=(12,8))
ax = sns.countplot(x='quantity', hue="status_group", palette='bright', data=df)

# Axis and title
fig.suptitle('Quantity of Water in Wells', fontsize=18)
plt.xlabel("Water Quantity", fontsize=14)
plt.ylabel("Number of Wells", fontsize=14)

plt.show();
```



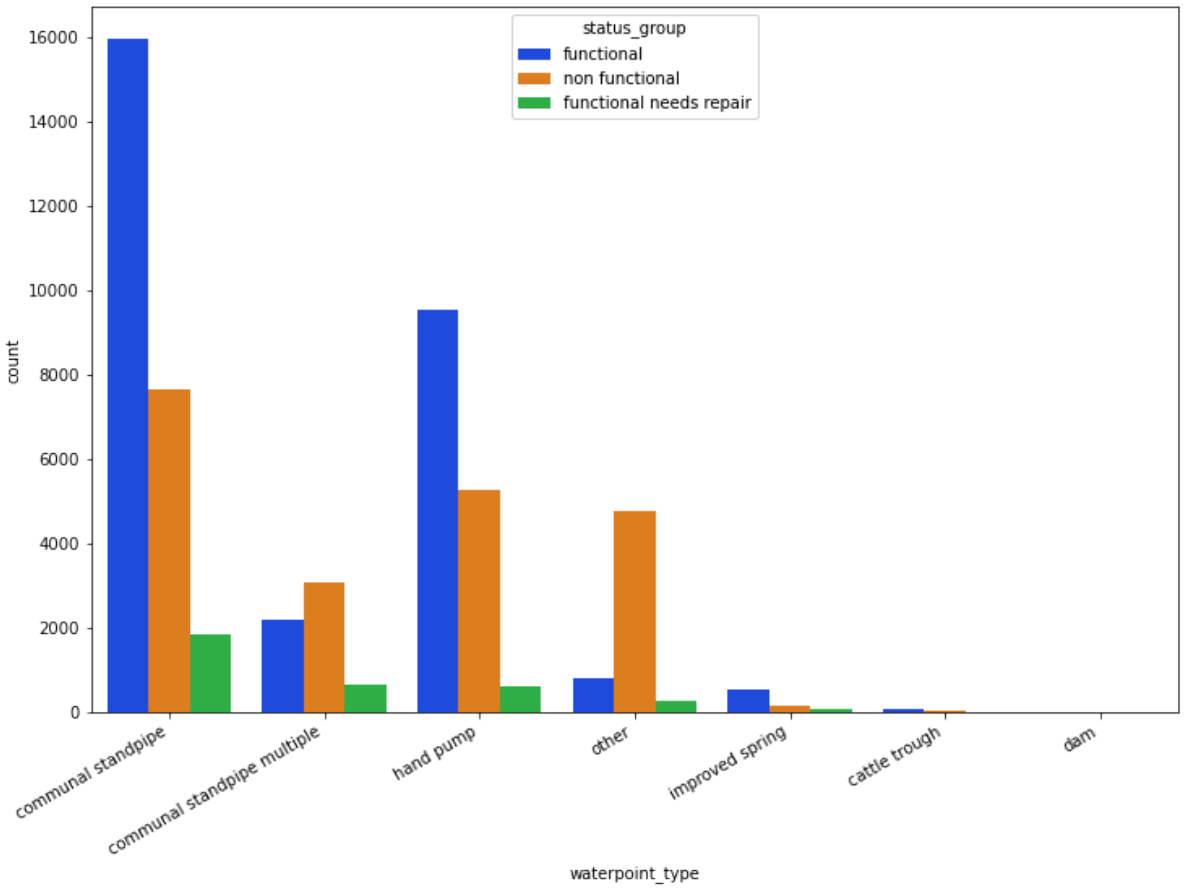
Dry waterpoints have a high chance of being non functional, as expected. If the waterpoint

has enough water, there is a high chance of functionality.

The Iringa region has a very high rate of functioning wells, followed by Kilimanjaro, Arusha, and Shinyanga. The worst regions for well perfomance are Mtwara, Mara, Rukwa, and Lindi.

Waterpoint type

In [59]:
```
# Plot
plt.figure(figsize=(12,8))
ax = sns.countplot(x='waterpoint_type', hue='status_group', palette='bright', da
plt.xticks(rotation=30, ha='right');
```

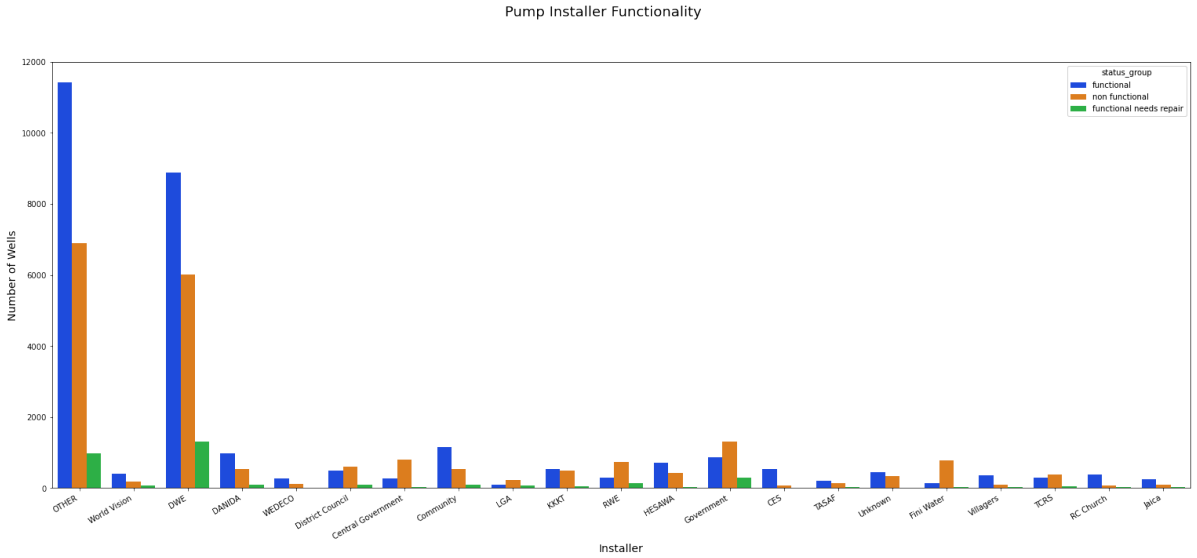


Other and communal standpipe multiple have the highest rate of being non functioning.

Installer

In [60]:
```
# Plot
fig, ax = plt.subplots(figsize=(26,10))
ax = sns.countplot(x='installer', hue="status_group", palette='bright', data=df)

# Title and axis labels
fig.suptitle('Pump Installer Functionality', fontsize=18)
plt.xlabel("Installer", fontsize=14)
plt.ylabel("Number of Wells", fontsize=14)
```

```
# Rotate x labels
plt.xticks(rotation=30, ha='right')

plt.show();
```



The government, Fini Water, RWE, and Distict Council have a high rate of non functioning wells. Other is out largest category.

Well Function map

In [61]:
```
# Import packages
import folium
from folium.plugins import FloatImage
```

In [62]:
```
# Create 3 dataframes for each status_group
df_f = df[df['status_group'] == 'functional']
df_nf = df[df['status_group'] == 'non functional']
df_fnr = df[df['status_group'] == 'functional needs repair']
```

In [63]:
```
# Create lists of latitude and longitude values
lat_f = [x for x in df_f['latitude']]
long_f = [x for x in df_f['longitude']]

lat_nf = [x for x in df_nf['latitude']]
long_nf = [x for x in df_nf['longitude']]

lat_fnr = [x for x in df_fnr['latitude']]
long_fnr = [x for x in df_fnr['longitude']]

lat_long_f = [(lat_f[i], long_f[i]) for i in range(len(lat_f))]
lat_long_nf = [(lat_nf[i], long_nf[i]) for i in range(len(lat_nf))]
lat_long_fnr = [(lat_fnr[i], long_fnr[i]) for i in range(len(lat_fnr))]
```

Create df['status'] with status_group in integer format

In [64]:
```
# Change status_group/target values to numeric values
df['status'] = df.status_group.map({"non functional":0, "functional needs repair
df.head()
```

Out[64]:

	status_group	amount_tsh	gps_height	installer	longitude	latitude	basin	reg
0	functional	6000.0	1390	OTHER	34.938093	-9.856322	Lake Nyasa	Ir
1	functional	0.0	1399	OTHER	34.698766	-2.147466	Lake Victoria	N
2	functional	25.0	686	World Vision	37.460664	-3.821329	Pangani	Man
3	non functional	0.0	263	OTHER	38.486161	-11.155298	Ruvuma / Southern Coast	Mtv
4	functional	0.0	0	OTHER	31.130847	-1.825359	Lake Victoria	Ka

In [65]:
```
df = df.drop('status_group', axis=1)
```

Modeling

Data Preprocessing

Following we will create our dummy variables for our categorical columns and perform train test split to prepare for modeling.

Create dummies

In [66]:
```
# Create lists of categorical and continuous columns
cat_col = ['installer','basin','region','extraction_type_class','management','pa
           'quantity','source_type','waterpoint_type']
cont_col = ['amount_tsh','gps_height','longitude','latitude','population','permi
```

In [67]:
```
#Create dummies
dummy_df = pd.get_dummies(df,columns=cat_col,drop_first=True)
```

Separate target and perform train test split

In [68]:

```
y = dummy_df['status']
X = dummy_df.drop(['status'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

Model Statistics Function

Precision will be our main metric used to track model performance, but we will calculate accuracy, recall, and f1 score to provide more detail using sklearn's *classification_report()* function.

In [69]:
```
# function to track model metrics and plot confusion matrix

def model_score(model, X, y_pred, y_true):
    target_names= ['non func', 'func need repair', 'functional']
    print(classification_report(y_true, y_pred, target_names=target_names))

    #Confusion matrix
    return plot_confusion_matrix(model, X, y_true, display_labels=target_names);
```

Dummy Classifier Model

In [70]:
```
from sklearn.dummy import DummyClassifier

dummy = DummyClassifier(random_state=42)
dummy.fit(X_train, y_train)
print(dummy.score(X_test, y_test))
```

```
0.5471768898893266
```

Our baseline dummy model performed very poorly with an accuracy score of 46%. Our data is heavily imbalanced, which explains how our ternary model performed close to 50%.

Logistic Regression

In [71]:
```
# train a logistic regression model on the training set
from sklearn.linear_model import LogisticRegression


# instantiate the model
logreg = LogisticRegression(solver='liblinear', random_state=0)


# fit the model
logreg.fit(X_train, y_train)
```

Out[71]:
```
LogisticRegression(random_state=0, solver='liblinear')
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page

with nbviewer.org.

In [72]:
```
y_pred_test = logreg.predict(X_test)

y_pred_test
```

Out[72]:
```
array([2, 0, 0, ..., 0, 2, 0], dtype=int64)
```

Our logistic regression model is improved to 75% accuracy over the dummy model. This model struggled to predict wells that were functional but needed repairs, likely due to class imbalances. The precision of the functional class is 73%.

In [73]:
```
# probability of getting output as 0 - no rain

logreg.predict_proba(X_test)[:,0]
```

Out[73]:
```
array([0.07636396, 0.97446781, 0.5685986 , ..., 0.99509324, 0.11839079,
       0.85914157])
```

Check accuracy score

In [74]:
```
from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred_tes
```

```
Model accuracy score: 0.7471
```

In [75]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.79      0.64      0.71      4157
           1       0.60      0.04      0.08       671
           2       0.73      0.90      0.81      5834

    accuracy                           0.75     10662
   macro avg       0.71      0.53      0.53     10662
weighted avg       0.74      0.75      0.72     10662
```

In [85]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Create the pipeline
pipe_lr = Pipeline([
    ('ss', StandardScaler()),
    ('lr', LogisticRegression())
])
```

```
# Fit the model
pipe_lr.fit(X_train, y_train)

# Predict on the test set
test_preds = pipe_lr.predict(X_test)

# Calculate the accuracy score
lr_score = accuracy_score(y_test, test_preds)

print("Test data model score:", lr_score)
```

```
Test data model score: 0.7499531044832114
```

Decision Tree Model

In [86]:

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn import tree
import graphviz

# Train a Decision Tree classifier
dc = DecisionTreeClassifier()
dc.fit(X_train, y_train)
```

Out[86]:

```
DecisionTreeClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [81]:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Create the pipeline
pipe_dt = Pipeline([
    ('ss', StandardScaler()),
    ('dt', DecisionTreeClassifier())
])

# Fit the model
pipe_dt.fit(X_train, y_train)

# Predict on the test set
test_preds = pipe_dt.predict(X_test)

# Calculate the accuracy score
dt_score = accuracy_score(y_test, test_preds)

print("Test data model score:", dt_score)
```

```
Test data model score: 0.7549240292628024
```

In [82]:
```
# Predict on training and test sets
training_preds = pipe_dt.predict(X_train)
test_preds = pipe_dt.predict(X_test)

# Accuracy of training and test sets
training_accuracy = accuracy_score(y_train, training_preds)
test_accuracy = accuracy_score(y_test, test_preds)

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

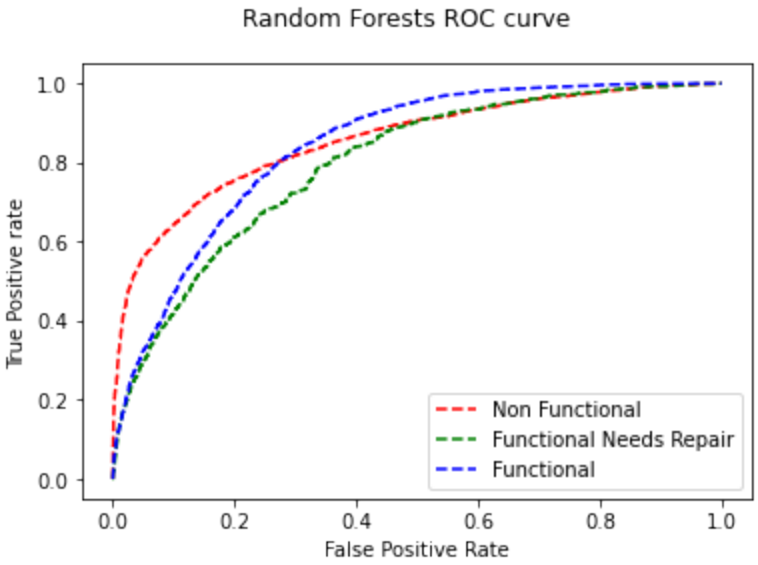
```
Training Accuracy: 100.0%
Validation accuracy: 75.49%
```

In [87]:
```
# Binarize for multiclass problem
y_multi_test = label_binarize(y_test, classes=[0, 1, 2])

# Predict probability for each model
dt_model_pred_proba = pipe_dt.predict_proba(X_test)
log_model_pred_proba = pipe_lr.predict_proba(X_test)



# Calculate AUC ROC score using predicted probablity for each model
dt_test = roc_auc_score(y_multi_test, dt_model_pred_proba, multi_class='ovr')
log_model_auc = roc_auc_score(y_multi_test, log_model_pred_proba, multi_class='o


print(f'The AUC score for our decision tree model is: {round(dt_test, 4)}')
print(f'The AUC score for our logistic regression model is: {round(log_model_auc
```

```
The AUC score for our decision tree model is: 0.7475
The AUC score for our logistic regression model is: 0.8309
```

In [90]:
```
# Plot
fig, ax = plt.subplots()

# Create dictionary
n_class = 3

fpr = {}
tpr = {}
thresh ={}

# Iterate through each class to create roc_curve
for i in range(n_class):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, log_model_pred_proba[:,i], pos

# plotting
plt.plot(fpr[0], tpr[0], linestyle='--',color='red', label='Non Functional')
plt.plot(fpr[1], tpr[1], linestyle='--',color='green', label='Functional Needs R
plt.plot(fpr[2], tpr[2], linestyle='--',color='blue', label='Functional')
plt.suptitle('Random Forests ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
```

Out[90]: <matplotlib.legend.Legend at 0x127a2728c40>



Conclusions

Data Utilization and Model Improvement

Analyze Key Features:

> Water Quantity: Since water quantity has proven crucial, ensure models account for this feature effectively. Payment Models: Investigate how payment influences functionality. Develop and promote strategies to implement or enhance fee structures to encourage regular maintenance. Address Non-Functional Waterpoints with Water:

> Prioritize over 8,000 non-functional waterpoints with sufficient water as they represent immediate opportunities for restoration with existing resources. Improve Data Quality:

> Future Data Collection: Ensure that data collected moving forward is accurate and comprehensive. This will enhance the reliability of predictions and recommendations. Continuous Monitoring and Updating: Regularly monitor the status of wells and update your models accordingly. This will help in refining predictions and adapting strategies.

Recommedation

> Prioritize Critical Areas:

Southeast (Mtwara, Lindi), North (Mara), and Southwest (Rukwa): Focus on these regions

Southeast (Mtwara, Lindi), North (Mara), and Southwest (Rukwa). Focus on these regions where the rate of non-functional waterpoints is high. Immediate intervention is crucial due to the severity of the situation.

> Assess and Repair Functional But Needing Maintenance Points:

Kigoma: Address the cluster of functional waterpoints needing repair to prevent costly future failures.

> Investigate and Improve Installer Performance:

Government, District Council, and Fini Water: Conduct a thorough review of why these entities have higher failure rates. Potential issues could include quality of installation, maintenance practices, or training deficits.