



Tanzania Machine Learning Water Pump Classification

Data Exploration Notebook

Author: Dylan Dey

This project is available on GitHub here: [GitHub Project Link](https://github.com/ddey117/Tanzanian_Water_Pump_Classification)
[\(https://github.com/ddey117/Tanzanian_Water_Pump_Classification\)](https://github.com/ddey117/Tanzanian_Water_Pump_Classification)

The Author can be reached by email: ddey2985@gmail.com (<mailto:ddey2985@gmail.com>)

Overview

The website '[DrivenData](https://www.drivendata.org/about/)' (<https://www.drivendata.org/about/>) finds partners to collaborate with in order to aggregate data to make it available in an open competition of Data Scientists from around the globe to come together to solve large social issues. For this project, DrivenData partnered with Taarifa and the Tanzanian Ministry of Water in order to predict the functionality of water pumps in the country of Tanzania. Machine learning classifiers from the '[scikit-learn](https://scikit-learn.org/stable/)' (<https://scikit-learn.org/stable/>) library were used to create a model for predictive maintenance of the water pumps.

Business Problem

Please feel free to check out my blog [here](#) for a more detailed dive into the problem.

Tanzania Water Scarcity

Several water points are established in Tanzania which currently supply the population with water, although this system is quite inefficient. Poor infrastructure and spotty maintenance plague the system with broken taps, broken pipes, and damaged supply sources. In addition, water pumps and plumbing make attractive targets for thieves. These factors and others block access to clean water from the established system.

The most common form of maintenance for these points is to repair them once they are no longer functional. This is not very efficient and is somewhat expensive, but it is magnitudes cheaper than establishing a new water point through drilling and installing of large equipment.

A better approach would be to prevent repairs from becoming necessary in the first place. Predictive maintenance constantly monitors the status of pumps in order to more efficiently maintain them. This is the goal of this project. Timely verification of the status of water pumps in the region can prevent further compounding of issues and can reduce maintenance costs significantly. Most importantly, it can bring fresh water to those who need it in a more efficient manner than before it was implemented.

For the past two decades, the Ministry of Water has been implementing sector reforms that aim at improving resource management and improving water supply in both rural and urban environments. An attempt to determine if predictive maintenance could help the Ministry of Water in the overall success of their mission statement.

Because the first goal is to predict if an unknown water pump is either functional or nonfunctional before ever sending anyone to physically check the pump I decided to simplify things for this project and only predict two labels instead of the three described by DrivenData in their competition. Future work could prove valuable to treat the issues as a tertiary classification problem, with focus on creating a more sophisticated model with proper class imbalance techniques. Focusing for now on creating a strong predictive model for a binary classification problem can be very beneficial for the immediate needs of the Ministry of Water.

A team will need different equipment, training, and other resources in order to do simple maintenance on a functional pump versus fully repairing a non-functional pump. It is critical to improve the efficiency of pump maintenance and repair to ensure more people can get what they need to survive as quickly as possible.

Without any model to make predictions, the Ministry of Water would have to physically check all of the pumps. The teams sent could bring just what they would need to maintain clean pumps, and only have what they need for a little over half of the pumps. Then they could record which pumps were maintained and which pumps still need to be repaired, go back to base, gather the proper supplies, and deploy again with the proper resources.

The ‘date recorded’ column in the dataset available from the DrivenData competition reveals that in 2011 nearly 28k? Water pumps were physically checked for their status. This seems to be the current capacity for just the exploratory part of maintenance before ever sending the proper equipment or supplies to fix what needs fixed. We could imagine how resource intensive this could

be through a little bit of speculation. If one team could check 3 pumps a day, worked five days a week with no holidays, and never took any additional time off, it would still take around 40 teams even under these unrealistic, intense conditions. This doesn't even consider the few months a year that Tanzania has heavy rains that could prove to slow progress even further. These 40 teams would need the proper equipment to navigate long distances. Tanzania has some of the most diverse geography in the world, with from some of the world's tallest mountain ranges and volcanoes, dense jungle, arid grassland with no shade from the intensity of the African sun, treacherous valleys, and seasonal flooding as just some features of Tanzania geography that can impede this first round of physical exploration before the second round in which resources can finally be directed to where they need to be. This burden can significantly lower whatever money and resources would even be left to buy and send the equipment that is needed after funding all of this exploration.

Thus, being able to reliably predict the status of a pump without physically checking the water pump could be extremely important. Reducing the amount of resources necessary for this crucial step can free up resources to return functionality to important water points to the communities that desperately need them, increasing the amount of people with access to fresh water. Furthermore, these communities will get access to fresh water on a much shorter timeline.

```
In [1]: 1 # Import standard packages
2 import pandas as pd
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6 %matplotlib inline
7 import pandas_profiling
8 from collections import Counter
9 from sklearn.preprocessing import OneHotEncoder
10
11 sns.set_theme(style="darkgrid")
```

Data Understanding

Target: status_group

There is a clear class imbalance when looking at the distribution of the three target labels. There are a lot fewer pumps that are functional but need repair than either the functional or non-functional pumps. This could create a bias in my models towards the heavier weighted samples unless I attempt to balance the data. For my project, I decided to simplify things and create a binary classification model that identifies pumps as either being functional or non-functional. All functional-needs-repair pumps will be binned with non-functional pumps. Future work will focus on more sophisticated methods to handle the class imbalance and build a model to solve a tertiary problem instead.

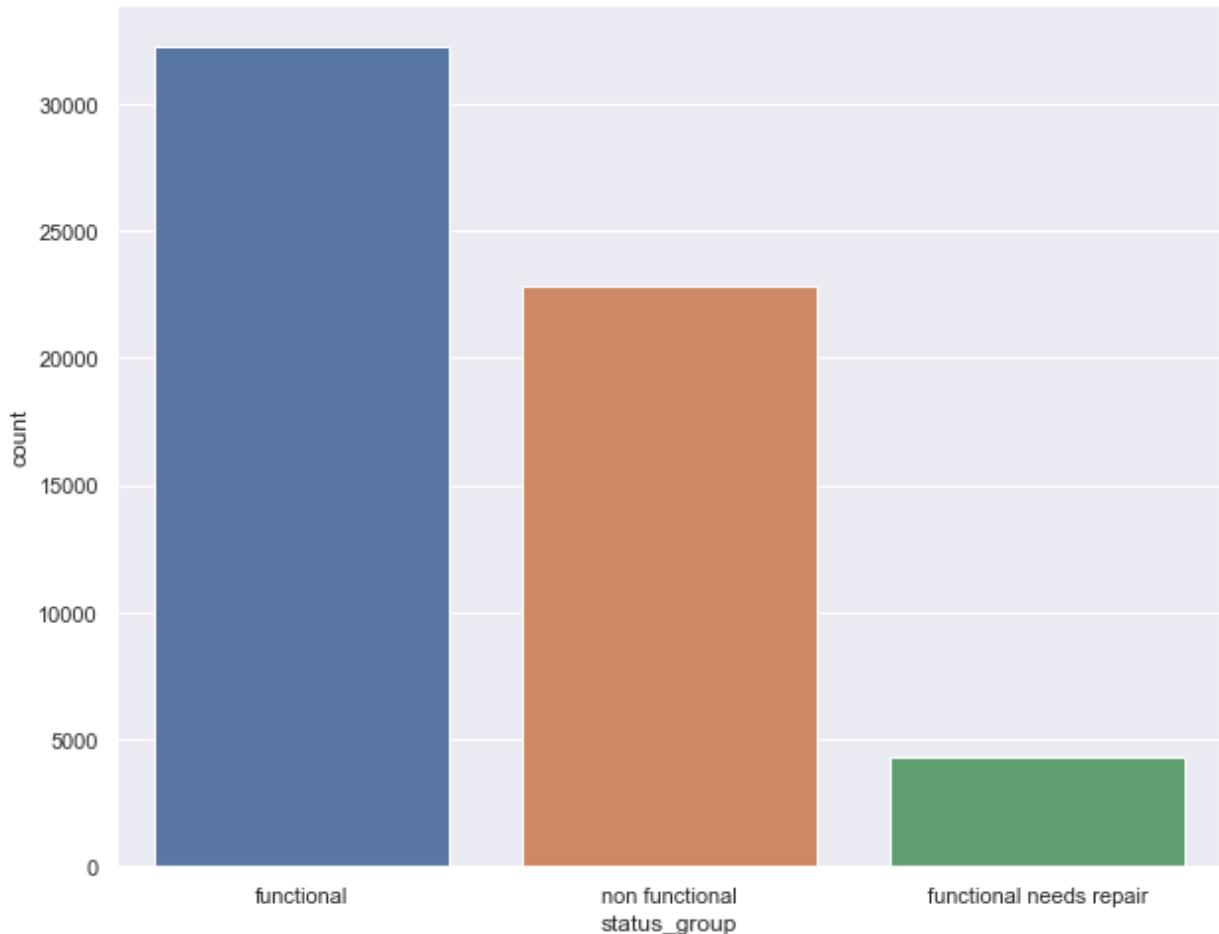
Data Understanding

```
In [2]: 1 train_labels = pd.read_csv('data/0bf8bc6e-30d0-4c50-956a-603fc693d966.c  
2 display(train_labels.head())  
3 train_labels.status_group.value_counts(normalize=True)
```

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

```
Out[2]: functional           0.543081  
non functional          0.384242  
functional needs repair  0.072677  
Name: status_group, dtype: float64
```

```
In [3]: 1 fig = plt.figure(figsize=(10,8))  
2 sns.countplot(x='status_group', data=train_labels)  
3 plt.show();
```



In [4]:

```

1 train_features = pd.read_csv('data/4910797b-ee55-40a7-8668-10efd5c1b960')
2 train_features.info()

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59400 entries, 0 to 59399
Data columns (total 40 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

dtypes: float64(3), int64(7), object(30)
memory usage: 18.1+ MB

In [5]:

```

1 #merge training features and training labels in order
2 #to explore the data
3 df = train_features.merge(train_labels, on='id').copy()

```

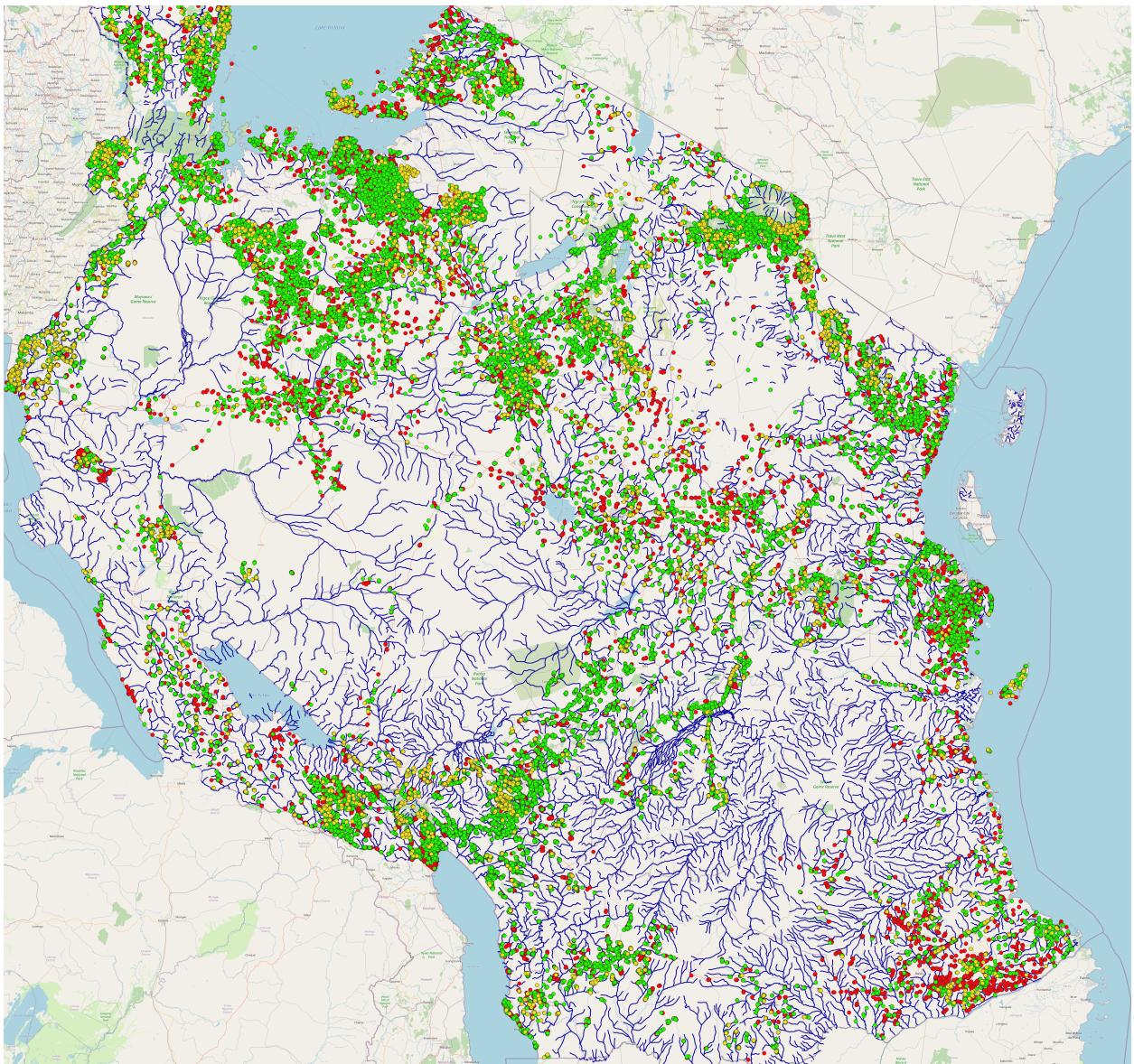
Geodata Exploration

River Distance using QGIS

In [6]:

```
1 #create a subset of data that just contains
2 #coordinates of the pumps
3 #in order to analyze using free open source
4 #QGIS software and open GIS data
5latlong = df[['latitude', 'longitude', 'status_group']]
6latlong.to_csv('data/latlong2.csv')
```

The map below was created using QGIS software. Green dots represent fully functional pumps. Yellow dots refer to pumps that are functional but in need of repair. The red dots are the pumps that are non-functional. A quick glance will reveal that there are some clusters that represent different distributions of functional to non functional pumps. However, it doesn't look like latitude or longitude alone would be very predictive variables to consider. Will drop latitude and longitude for modeling and keep other more broad geological predictors.



click below for a pdf version of map with zoom functionality.

[water_pump_pdf \(images/map_image.png.pdf\)](#)

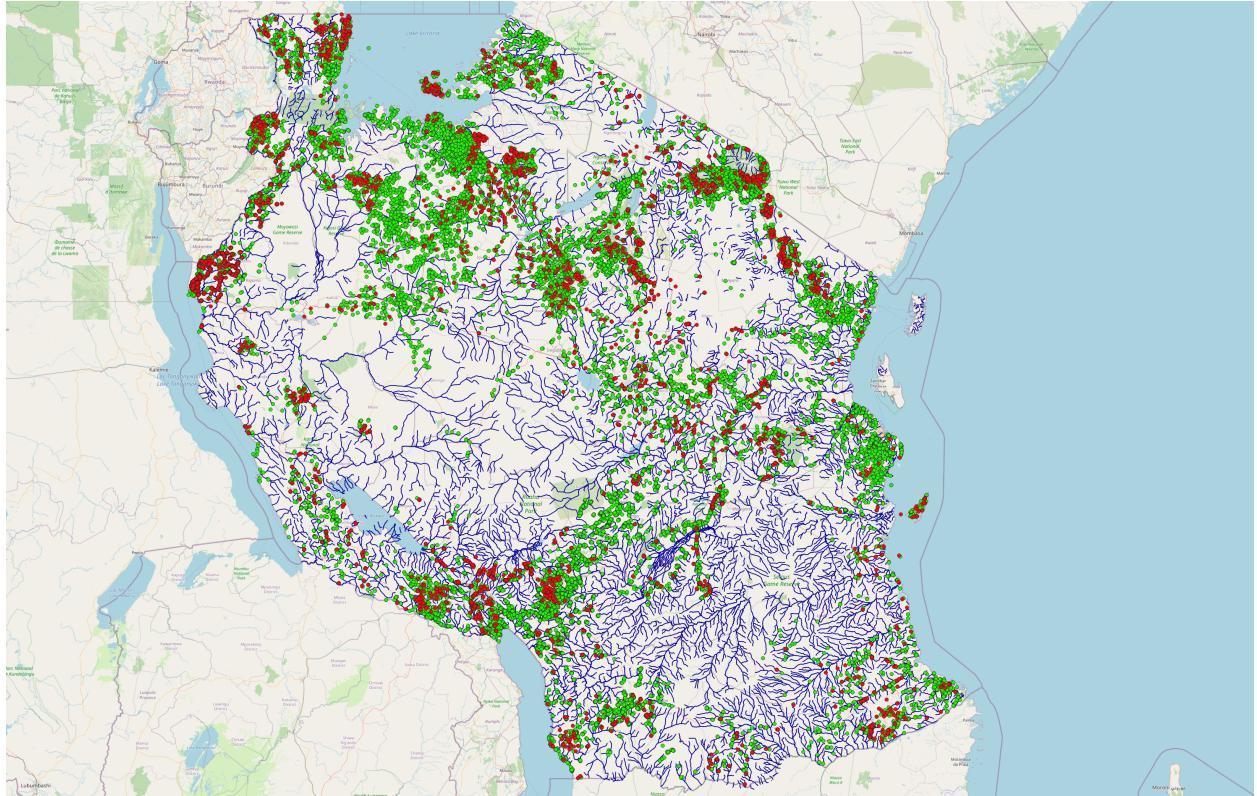
Knowing what pumps are functional/nonfunctional and being sure about it can prove essential for our predictive maintenance model. In this way, we can save resources by being sure which communities already have what they need vs communities that are more desperate for assistance and resources. There is a strong class imbalance in the tertiary model. In the future I would like to use more sophisticated methods to try to deal with the class imbalance and make predictions on all 3 labels. It could help to save even more money and resources to be able to accurately distinguish between functional needs repair and non functional. However, for now, I think building a reasonably accurate binary model is an excellent starting place for predictive maintenance.

```
In [7]: 1 need_repair_index = df['status_group'] == 'functional needs repair'
2 df_binary = df.copy()
3 df_binary.loc[need_repair_index, 'status_group'] = 'non functional'
```

```
In [8]: 1 df_binary['status_group'].value_counts()
```

```
Out[8]: functional      32259
non functional    27141
Name: status_group, dtype: int64
```

Below is an unupdated map for a binary classification model.

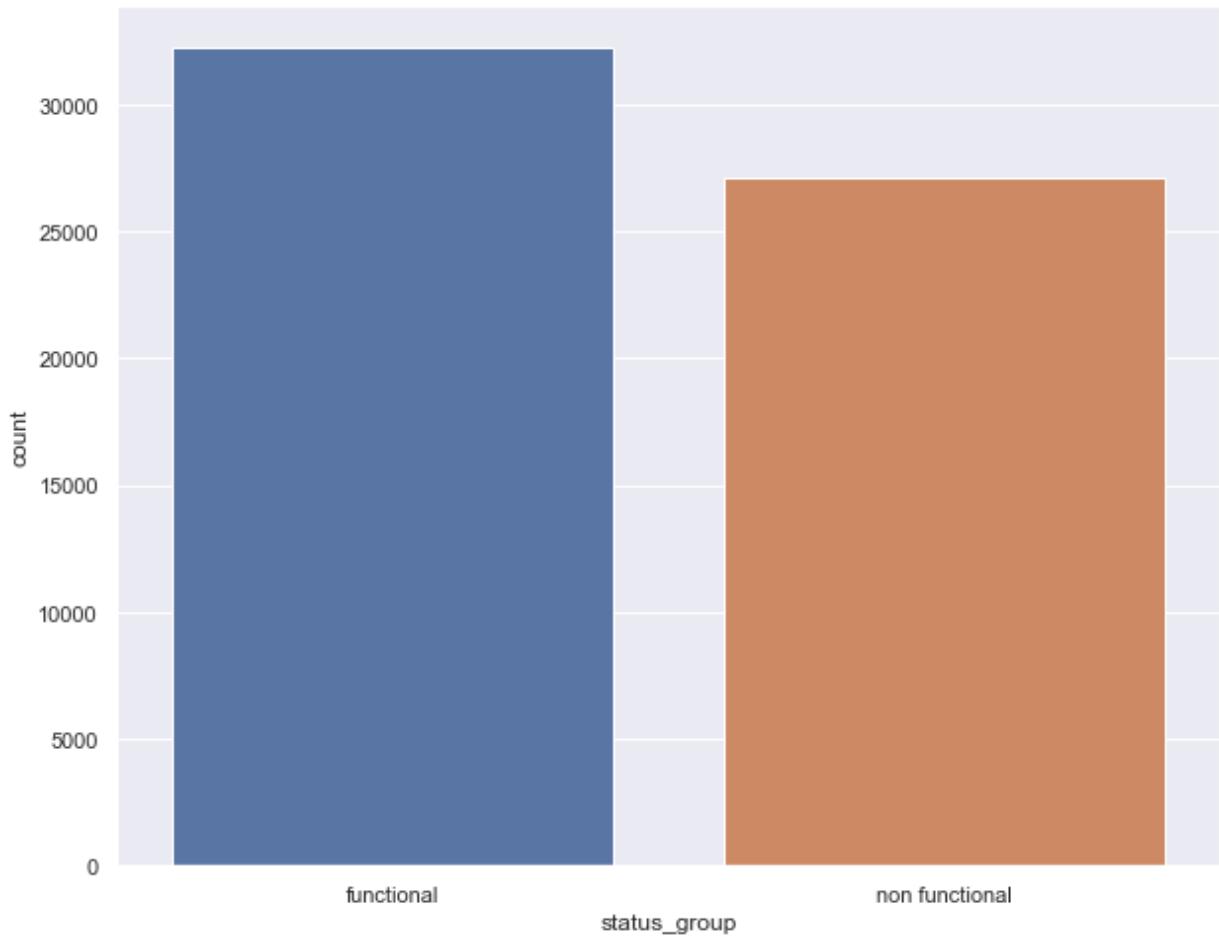


click below for a pdf version of map with zoom functionality.

[water_pump_pdf \(images/binary_map2.pdf\)](#)

In [9]:

```
1 fig = plt.figure(figsize=(10,8))
2 sns.countplot(x='status_group', data=df_binary)
3 #plt.savefig('images/binary_status.png')
4 plt.show();
```



In [10]:

```
1 river_df = pd.read_csv('data/river_dist2.csv')
2 river_df.head()
```

Out[10]:

	field_1	latitude	longitude	HubName	HubDist
0	0	-9.856322	34.938093	River/Stream	8.025861
1	1	-2.147466	34.698766	River/Stream	7.538778
2	2	-3.821329	37.460664	River/Stream	2.523575
3	3	-11.155298	38.486161	River/Stream	8.307942
4	4	-1.825359	31.130847	River/Stream	4.524439

In [11]:

```

1 #because some of the distance vectors were created with false coordinates
2 #I decided to fill in the incorrect distance values
3 #with the median distance for distance to nearest river
4 #there are about 1800 rows with incorrect lat/long coordinates
5 #these 1800 rows show lat/long values of zero which
6 #are obvious outliers when plotted on a map
7
8 index_riv = river_df[river_df['HubDist'] > 66].index
9 river_median = river_df['HubDist'].median()
10 river_df.loc[index_riv, 'HubDist'] = river_median
11 river_df['HubDist'].describe()

```

Out[11]:

	count	59400.000000
mean	8.520694	
std	6.833602	
min	0.035102	
25%	4.237760	
50%	7.115599	
75%	10.669175	
max	65.057665	
Name:	HubDist, dtype:	float64

In [12]:

```

1 #create a boolean feature describing whether
2 #a water pump is within 8 km of a river
3
4 river_s = river_df['HubDist'].copy()
5 river_s.rename('near_river', inplace=True)
6 near_river = river_s[river_s < 8].apply(lambda x: 1 if not pd.isnull(x)
7 df = df.join(near_river)
8 df.near_river.fillna(0, inplace=True)
9 df.near_river.value_counts(normalize=True)

```

Out[12]:

	0.0	0.597323
1.0	0.0	0.402677
Name:	near_river, dtype:	float64

In [13]:

```

1 display(df[df['near_river'] == 1]['status_group'].value_counts(normalize=True)
2 df[df['near_river'] == 0]['status_group'].value_counts(normalize=True)

```

status_group	value
functional	0.555311
non functional	0.369663
functional needs repair	0.075026
Name: status_group, dtype:	float64

Out[13]:

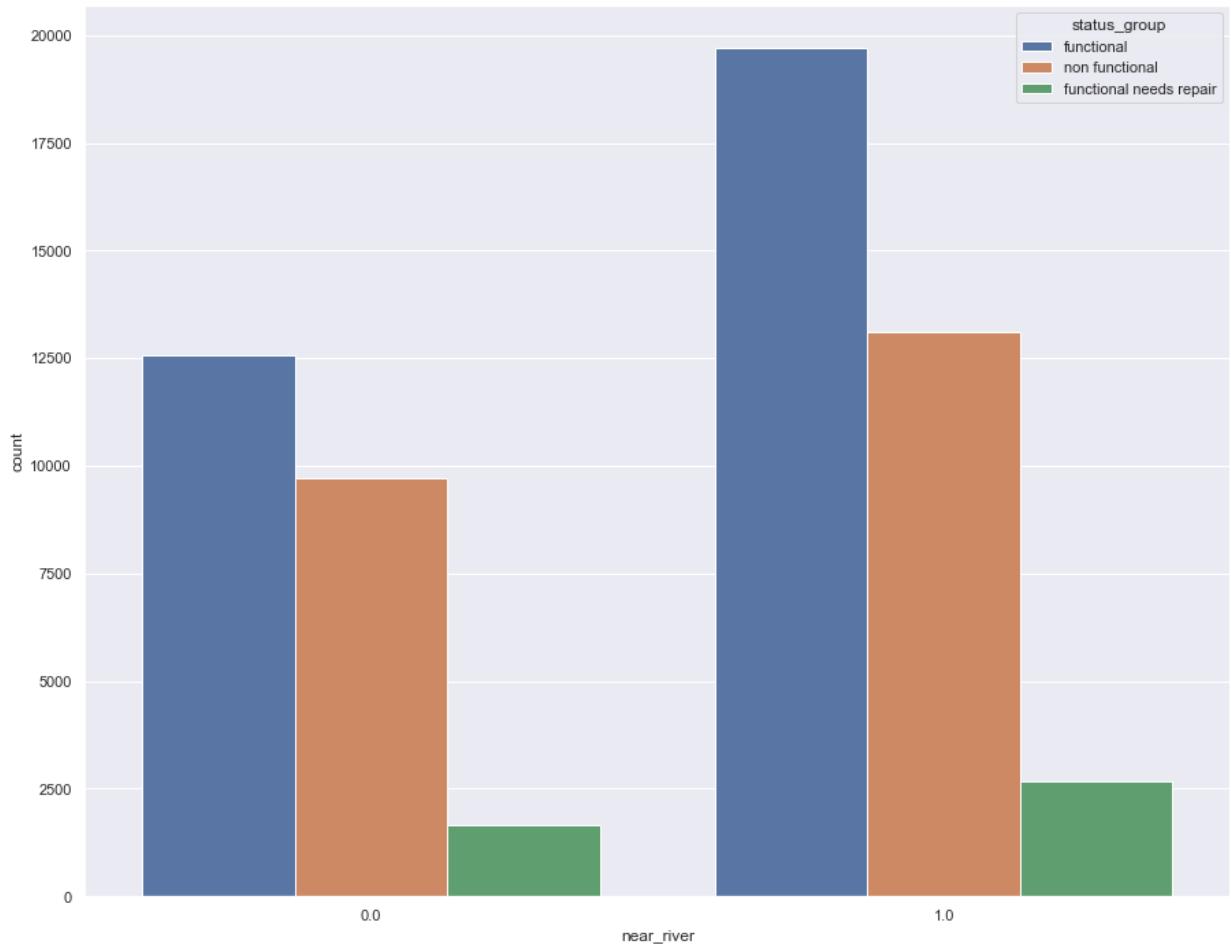
status_group	value
functional	0.524938
non functional	0.405870
functional needs repair	0.069192
Name: status_group, dtype:	float64

In [14]:

```

1 fig = plt.figure(figsize=(15,12))
2 sns.countplot(x='near_river', hue='status_group', data=df)
3 plt.show();

```



In [15]:

```

1 #profile = pandas_profiling.ProfileReport(df.copy())
2 #profile.to_file("output.html")

```

Adding Population data by ward

The column describing population is heavily skewed towards a value of zero, and it would seem very unlikely for so many wells to have a population of zero. Nearly 50% of the data is skewed towards a value of either zero or one. It is hard to trust the validity of this column. I decided to drop this column and bring in population information by ward collected by government census in 2012. This data has its own issues but I believe it to be a more complete dataset and give a better idea about the number of people who could benefit from the presence of a functional well in their area. Ward is a pretty high level division of geological boundaries for Tanzania. May have to revisit this population data at a later date.

In [16]:

```

1 df_pop = pd.read_excel('data/tza-pop-popn-nbs-baselinedata-xlsx-1.xlsx')
2 display(df_pop.head())
3 df_pop[ 'Ward_Name' ].isna().sum()

```

	Reg_Code	Reg_Name	Dis_Code	Dis_Name	Ward_Code	Ward_Name	Division	PCode	total_b
0	1	Dodoma	1	Kondoa	11	Bumbuta	NaN	101011	86
1	1	Dodoma	1	Kondoa	21	Pahi	NaN	101021	139
2	1	Dodoma	1	Kondoa	41	Haubi	NaN	101041	137
3	1	Dodoma	1	Kondoa	51	Kalamba	NaN	101051	139
4	1	Dodoma	1	Kondoa	61	Kwadelo	NaN	101061	115

5 rows × 21 columns

Out[16]: 0

In [17]:

```

1 #create a dictionary of values with format {Ward : Total Population}
2
3 pop_index = df_pop.groupby('Ward_Name')[ 'total_both' ].sum().index
4 pop_values = df_pop.groupby('Ward_Name')[ 'total_both' ].sum().values
5 pop_dict = dict(zip(pop_index, pop_values))
6
7 #create pandas Dataframe for merging
8 pop_dataframe = pd.DataFrame.from_dict(pop_dict, orient='index')
9 #rename column for clarity
10 pop_dataframe.rename(columns={0: 'ward_pop'}, inplace=True)

```

In [18]:

```

1 #merge dataframes
2 df_pop_merge = df.merge(pop_dataframe,
3                         how='left',
4                         left_on='ward',
5                         right_index=True)
6
7 df_pop_merge.info()

```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 59400 entries, 0 to 59399
Data columns (total 43 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  
 41  near_river          59400 non-null   float64 
 42  ward_pop            53000 non-null   float64 

```

```
dtypes: float64(5), int64(7), object(31)
memory usage: 22.4+ MB
```

```
In [19]: 1 #replace null values of ward population with
          2 #median ward population
          3
          4 ward_pop_median = df_pop_merge['ward_pop'].median()
          5 df_pop_merge.fillna(value=ward_pop_median, inplace=True)
          6 df_pop_merge['ward_pop'].isna().sum()
```

```
Out[19]: 0
```

In [20]:

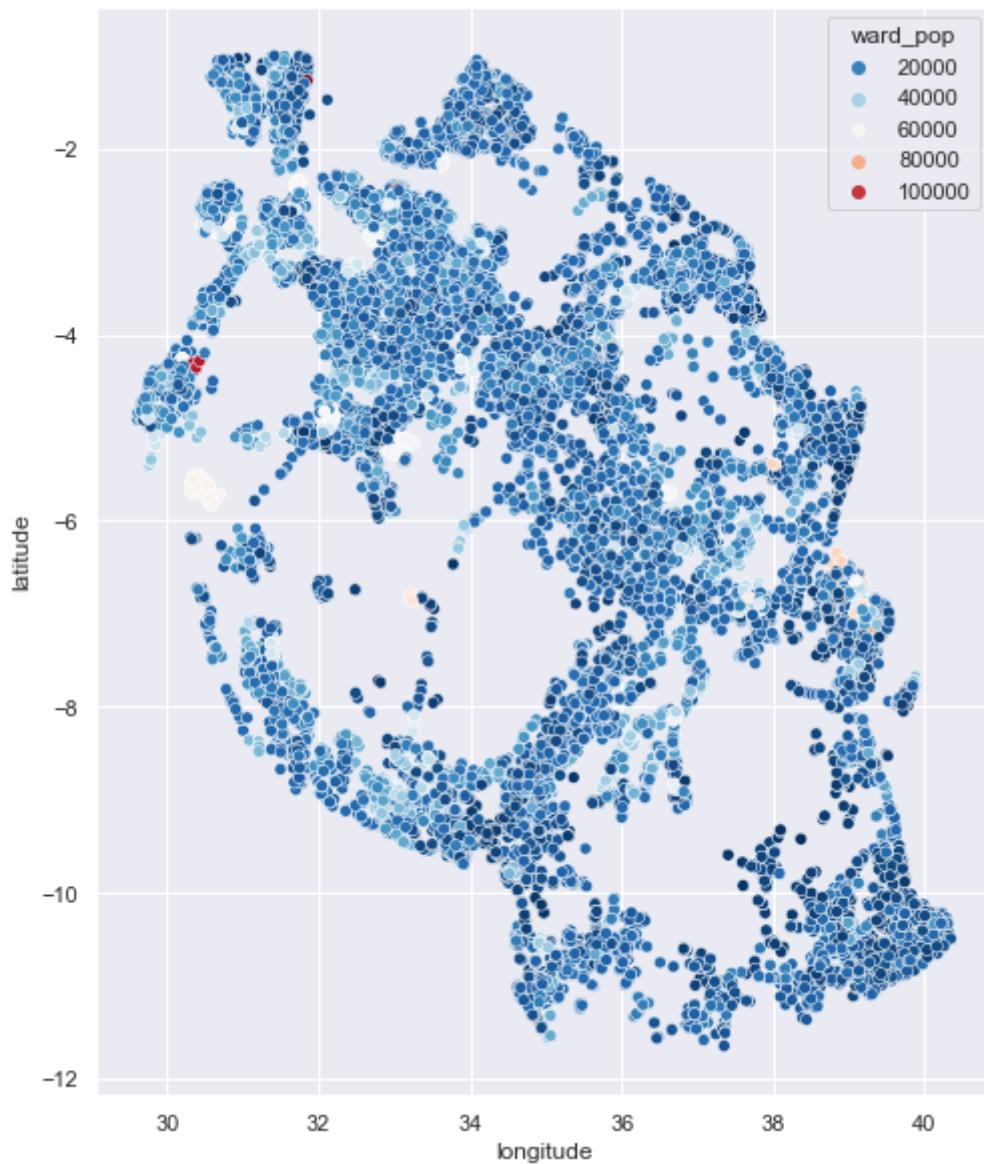
```
1 ward_pop_s = df_pop_merge['ward_pop'].copy()
2 df = df.join(ward_pop_s)
3 df.drop(columns=['population'], axis=1, inplace=True)
4 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59400 entries, 0 to 59399
Data columns (total 42 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  public_meeting   56066 non-null   object  
 18  recorded_by      59400 non-null   object  
 19  scheme_management 55523 non-null   object  
 20  scheme_name      31234 non-null   object  
 21  permit            56344 non-null   object  
 22  construction_year 59400 non-null   int64  
 23  extraction_type  59400 non-null   object  
 24  extraction_type_group 59400 non-null   object  
 25  extraction_type_class 59400 non-null   object  
 26  management        59400 non-null   object  
 27  management_group  59400 non-null   object  
 28  payment           59400 non-null   object  
 29  payment_type      59400 non-null   object  
 30  water_quality     59400 non-null   object  
 31  quality_group     59400 non-null   object  
 32  quantity          59400 non-null   object  
 33  quantity_group    59400 non-null   object  
 34  source            59400 non-null   object  
 35  source_type        59400 non-null   object  
 36  source_class       59400 non-null   object  
 37  waterpoint_type   59400 non-null   object  
 38  waterpoint_type_group 59400 non-null   object  
 39  status_group       59400 non-null   object  
 40  near_river         59400 non-null   float64 
 41  ward_pop          59400 non-null   float64 

dtypes: float64(5), int64(6), object(31)
memory usage: 22.0+ MB
```

```
In [21]: 1 fig_df = df.copy()
2 to_drop_index = fig_df[fig_df['longitude'] == 0].index
3 fig_df = fig_df.drop(to_drop_index)
```

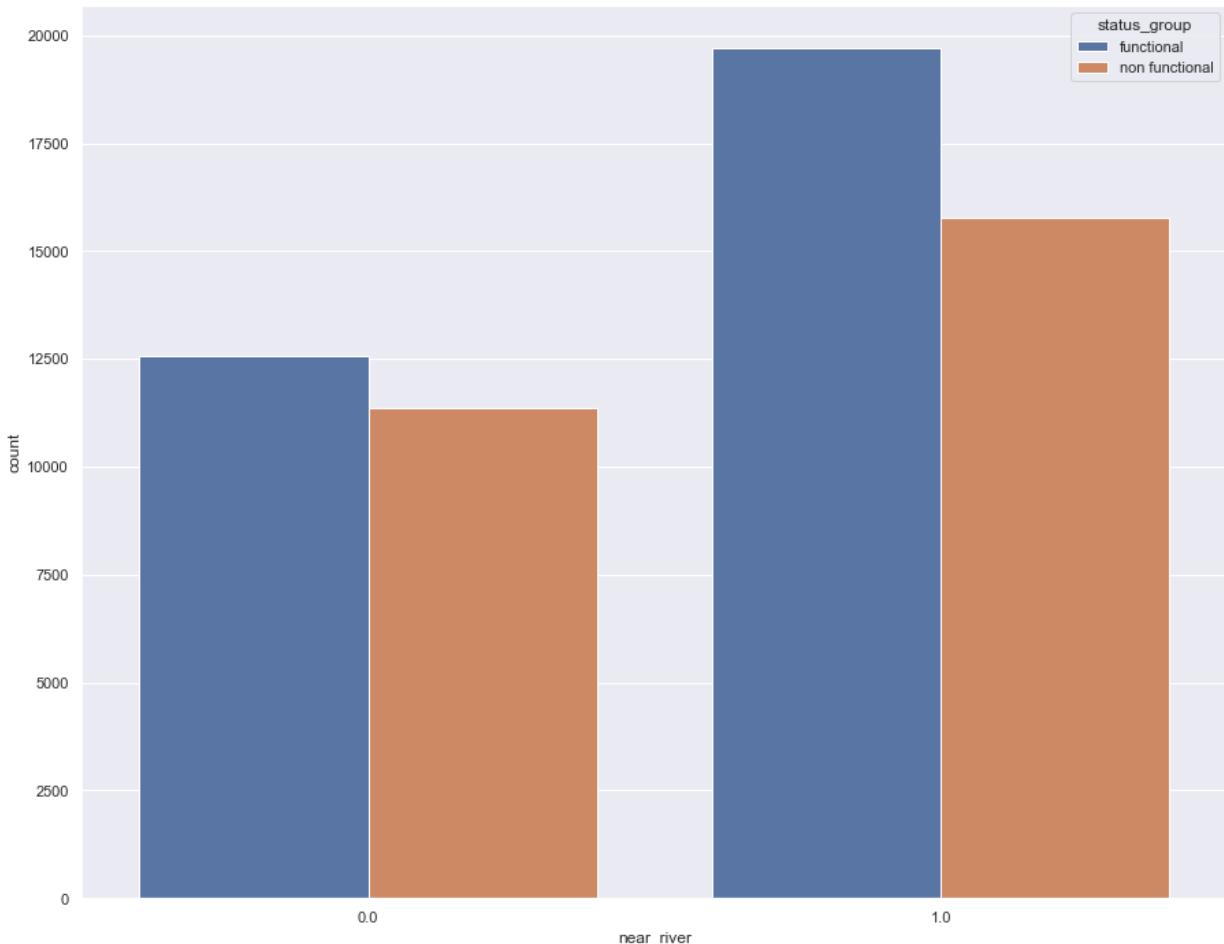
```
In [22]: 1 #quick visualization of the density of population around pumps
2 #derived from ward population government census data
3 pop_fig = plt.figure(figsize=(8,10))
4 sns.scatterplot(x='longitude',
5                  y='latitude',
6                  hue='ward_pop',
7                  data=fig_df,
8                  palette="RdBu_r",
9                  )
10
11 #plt.savefig('images/ward_pop_map.png')
12 plt.show();
```



Binary Classification Model Binning

```
In [23]: 1 df_binary = df.copy()
2 df_binary.loc[need_repair_index, 'status_group'] = 'non functional'
```

```
In [24]: 1 fig = plt.figure(figsize=(15,12))
2 sns.countplot(x='near_river', hue='status_group', data=df_binary)
3 plt.show();
```



At this point, I have my dataframe that I will treat as my raw data.

Data Cleaning

Dropping Redundant and Uninformative Columns Overview

Below are the columns that I decided to drop initially and the reasoning behind dropping the column. This step will help overall model speeds and should reduce the chance of overfitting by cutting down on multicollinearity.

id: uninformative

recorded_by: constant, unimportant for model

num_private: nearly 99% zeroes. Uninformative for model.

waterpoint_type_group: redundant. waterpoint_type covers same information.

source: Explains same information as source_type but with more variables. Keeping source_type.

source_class: Broadly explains same information as source and source_type. Keeping

source_type.

Source_type: This was NOT dropped. Kept it as a balance between the other two features of similar information.

extraction_type: explains similar information as extraction_type_group and extraction_type_class. Dropped.

extraction_type_group: explains similar information as extraction_type and extraction_type_class. Dropped.

extraction_type_class: broadly explains similar information as extraction_type and extraction_type_group. Kept.

payment: redundant information. Keep this and drop 'payment type'

payment_type: Same information as payment. Drop.

management: keep this and drop 'management_type'.

management_group: drop. Information contained within management column.

scheme_name: drop. Missing almost half of values. Large cardinality (nearly 2700 categories).

scheme_manager: drop. Contains similar information to management column.

water_quality: Keep this and drop 'quality group.' **water_quantity:** Keep this and drop water_quantity. They contain identical information. **date_recorded:** administrative. Not predictive.

latitude: No clear relationship between coordinate and pump status after viewing map. There is some obvious clustering though. Hope to get enough information from the other geological categorical features. Basin is likely to always show up in feature data as it is most general. Latitude and longitude for the pumps may even be imputed from mean or median values based on ward or some other common geological boundaries. Dropped. **longitude:** See above comments on latitude. Dropped.

In [25]: 1 df.columns

```
Out[25]: Index(['id', 'amount_tsh', 'date_recorded', 'funder', 'gps_height',
       'installer', 'longitude', 'latitude', 'wpt_name', 'num_private',
       'basin', 'subvillage', 'region', 'region_code', 'district_code',
       'lga',
       'ward', 'public_meeting', 'recorded_by', 'scheme_management',
       'scheme_name', 'permit', 'construction_year', 'extraction_type',
       'extraction_type_group', 'extraction_type_class', 'management',
       'management_group', 'payment', 'payment_type', 'water_quality',
       'quality_group', 'quantity', 'quantity_group', 'source', 'source_t
ype',
       'source_class', 'waterpoint_type', 'waterpoint_type_group',
       'status_group', 'near_river', 'ward_pop'],
      dtype='object')
```

In [26]:

```

1 #view dataframe in different orientation
2 #to check for redundant information
3 df.head(10).transpose()

```

Out[26]:

	0	1	2	3	4	5
id	69572	8776	34310	67743	19728	9944
amount_tsh	6000	0	25	0	0	20
date_recorded	2011-03-14	2013-03-06	2013-02-25	2013-01-28	2011-07-13	2011-03-13
funder	Roman	Grumeti	Lottery Club	Unicef	Action In A	Mkinga Distric Coun
gps_height	1390	1399	686	263	0	0
installer	Roman	GRUMETI	World vision	UNICEF	Artisan	DWE
longitude	34.9381	34.6988	37.4607	38.4862	31.1308	39.1728
latitude	-9.85632	-2.14747	-3.82133	-11.1553	-1.82536	-4.76559
wpt_name	none	Zahanati	Kwa Mahundi	Zahanati Ya Nanyumbu	Shuleni	Tajiri
num_private	0	0	0	0	0	0
basin	Lake Nyasa	Lake Victoria	Pangani	Ruvuma / Southern Coast	Lake Victoria	Pangani
subvillage	Mnyusi B	Nyamara	Majengo	Mahakamani	Kyanyamisa	Moa/Mwereme
region	Iringa	Mara	Manyara	Mtvara	Kagera	Tanga
region_code	11	20	21	90	18	4
district_code	5	2	4	63	1	8
lga	Ludewa	Serengeti	Simanjiro	Nanyumbu	Karagwe	Mkinga
ward	Mundindi	Natta	Ngorika	Nanyumbu	Nyakasimbi	Moa
public_meeting	True	NaN	True	True	True	True
recorded_by	GeoData Consultants Ltd	GeoData Consultants Ltd	GeoData Consultants Ltd	GeoData Consultants Ltd	GeoData Consultants Ltd	GeoData Consultants Ltd
scheme_management	VWC	Other	VWC	VWC	NaN	VWC
scheme_name	Roman	NaN	Nyumba ya mungu pipe scheme	NaN	NaN	Zingibali
permit	False	True	True	True	True	True
construction_year	1999	2010	2009	1986	0	2009
extraction_type	gravity	gravity	gravity	submersible	gravity	submersible
extraction_type_group	gravity	gravity	gravity	submersible	gravity	submersible
extraction_type_class	gravity	gravity	gravity	submersible	gravity	submersible

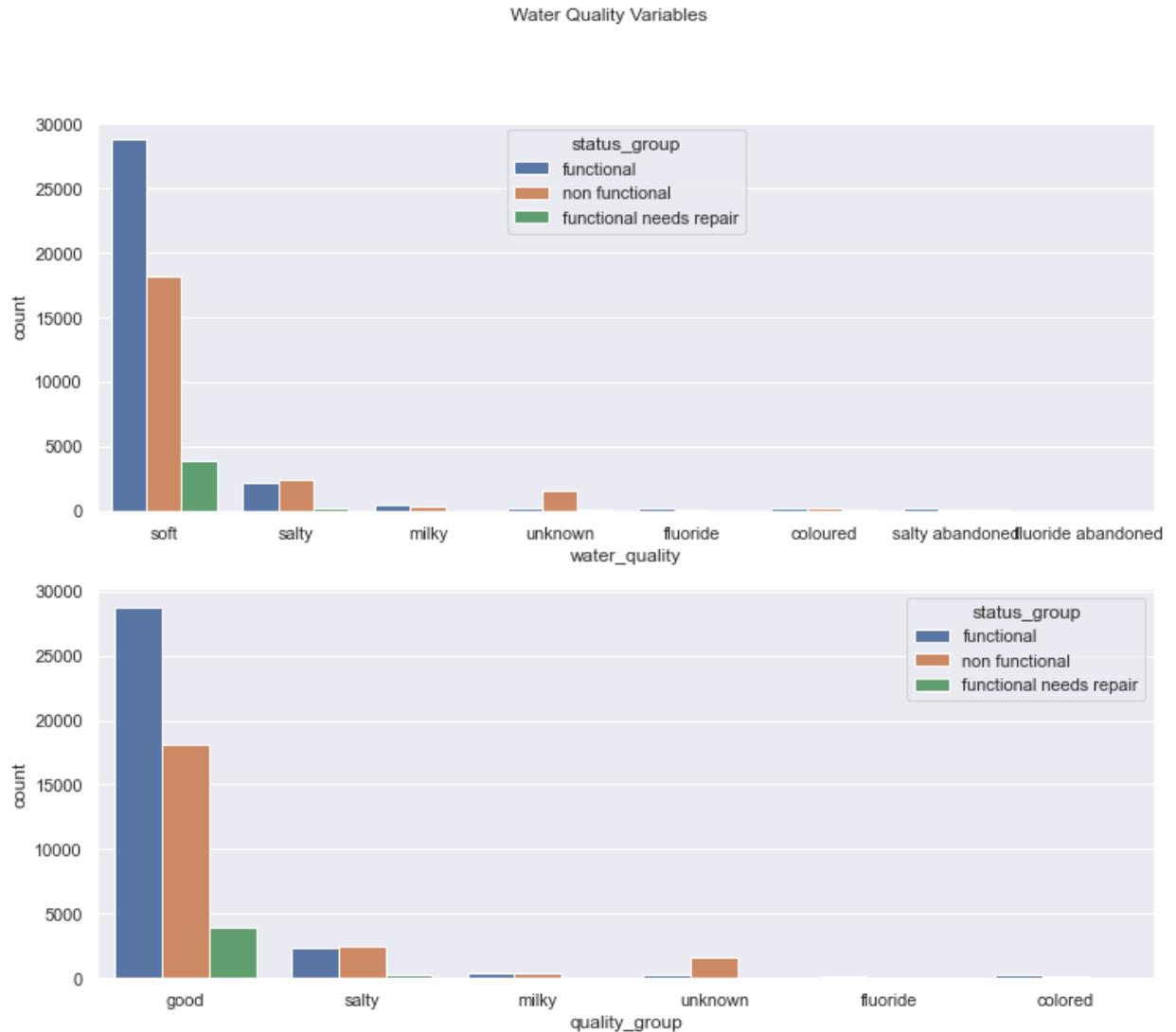
	0	1	2	3	4	5
management	vwc	wug	vwc	vwc	other	vwc
management_group	user-group	user-group	user-group	user-group	other	user-group
payment	pay annually	never pay	pay per bucket	never pay	never pay	pay per bucket
payment_type	annually	never pay	per bucket	never pay	never pay	per bucket
water_quality	soft	soft	soft	soft	soft	salty
quality_group	good	good	good	good	good	salty
quantity	enough	insufficient	enough	dry	seasonal	enough
quantity_group	enough	insufficient	enough	dry	seasonal	enough
source	spring	rainwater harvesting	dam	machine dbh	rainwater harvesting	other
source_type	spring	rainwater harvesting	dam	borehole	rainwater harvesting	other
source_class	groundwater	surface	surface	groundwater	surface	unknown
waterpoint_type	communal standpipe	communal standpipe	communal standpipe multiple	communal standpipe multiple	communal standpipe	communal standpipe multiple
waterpoint_type_group	communal standpipe	communal standpipe	communal standpipe	communal standpipe	communal standpipe	communal standpipe
status_group	functional	functional	functional	non functional	functional	functional
near_river	0	1	1	0	1	1
ward_pop	7346	12849	7417	10826	12803	4339

Water_quality vs Quality_group

Both of these columns contain essentially the same information, subgrouped differently. I decided to keep 'water_quality' and drop 'quality_group' in order to avoid overfitting issues and speed up the iterative modeling process.

In [27]:

```
1 fig, (ax1, ax2) = plt.subplots(2, figsize=(12,10))
2 fig.suptitle('Water Quality Variables')
3 sns.countplot(x='water_quality', hue='status_group', data=df, ax=ax1)
4 sns.countplot(x='quality_group', hue='status_group', data=df, ax=ax2)
5 plt.show();
```



In [28]:

```

1 display(df.groupby('quality_group')['status_group'].value_counts(normalize=True))
2 df.groupby('water_quality')['status_group'].value_counts(normalize=True)

```

quality_group	status_group	
colored	functional	0.502041
	non functional	0.387755
	functional needs repair	0.110204
fluoride	functional	0.723502
	non functional	0.216590
	functional needs repair	0.059908
good	functional	0.565941
	non functional	0.357236
	functional needs repair	0.076823
milky	functional	0.544776
	non functional	0.437811
	functional needs repair	0.017413
salty	non functional	0.482002
	functional	0.460828
	functional needs repair	0.057170
unknown	non functional	0.840618
	functional	0.140725
	functional needs repair	0.018657

Name: status_group, dtype: float64

Out[28]: water_quality

water_quality	status_group	
coloured	functional	0.502041
	non functional	0.387755
	functional needs repair	0.110204
fluoride	functional	0.755000
	non functional	0.180000
	functional needs repair	0.065000
fluoride abandoned	non functional	0.647059
	functional	0.352941
milky	functional	0.544776
	non functional	0.437811
	functional needs repair	0.017413
salty	non functional	0.496499
	functional	0.457166
	functional needs repair	0.046334
salty abandoned	functional	0.513274
	non functional	0.274336
	functional needs repair	0.212389
soft	functional	0.565941
	non functional	0.357236
	functional needs repair	0.076823
unknown	non functional	0.840618
	functional	0.140725
	functional needs repair	0.018657

Name: status_group, dtype: float64

In [29]:

```
1 print('-----')
2 display(df_binary.groupby('quality_group')['status_group'].value_counts)
3 df_binary.groupby('water_quality')['status_group'].value_counts(normali
```

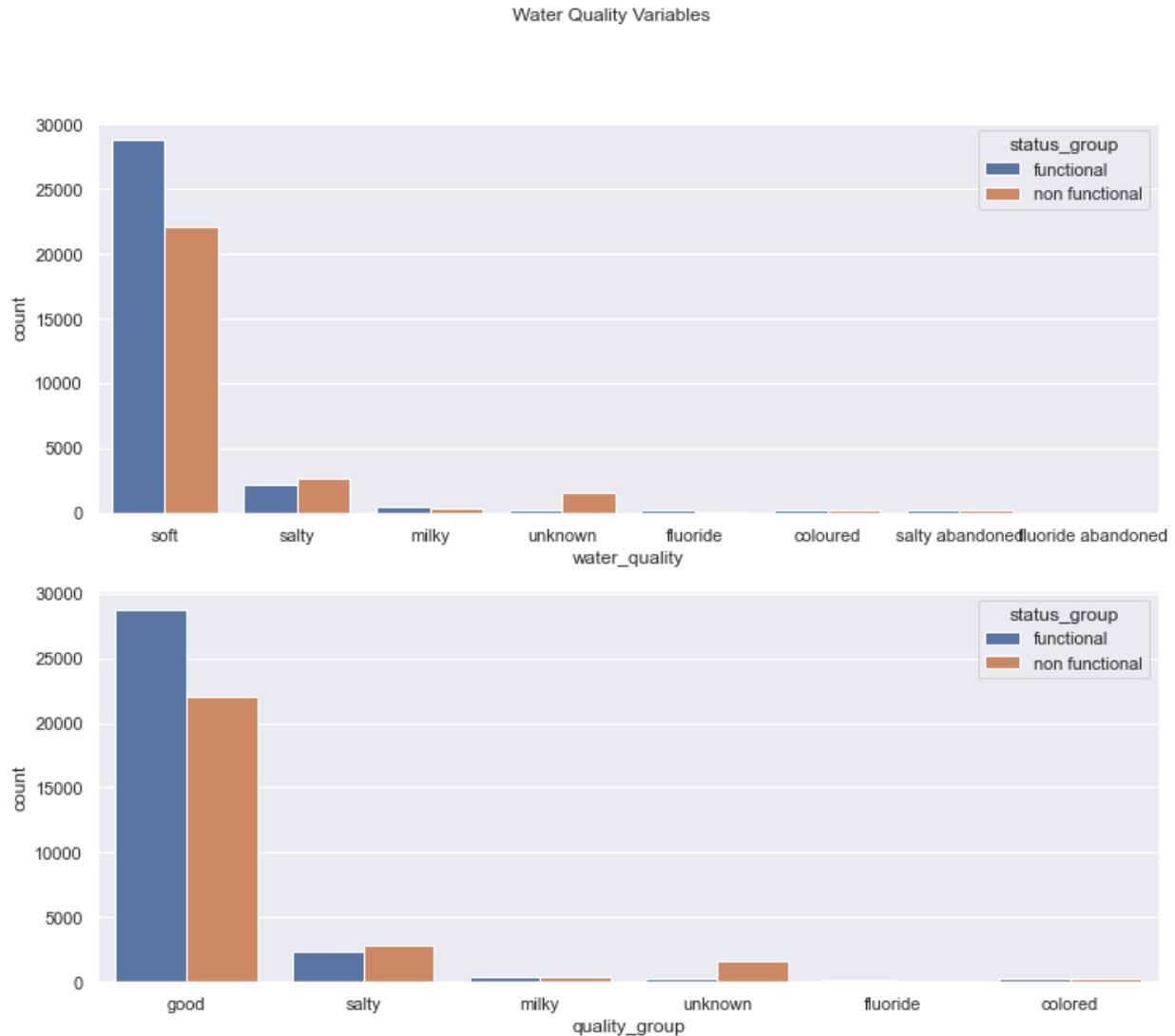
```
-----
quality_group  status_group
colored        functional      0.502041
                non functional  0.497959
fluoride       functional      0.723502
                non functional  0.276498
good           functional      0.565941
                non functional  0.434059
milky          functional      0.544776
                non functional  0.455224
salty           non functional 0.539172
                functional      0.460828
unknown         non functional 0.859275
                functional      0.140725
Name: status_group, dtype: float64
```

Out[29]: water_quality

```
status_group
coloured      functional      0.502041
                non functional  0.497959
fluoride       functional      0.755000
                non functional  0.245000
fluoride abandoned  non functional  0.647059
                functional      0.352941
milky          functional      0.544776
                non functional  0.455224
salty           non functional 0.542834
                functional      0.457166
salty abandoned  functional      0.513274
                non functional  0.486726
soft            functional      0.565941
                non functional  0.434059
unknown         non functional 0.859275
                functional      0.140725
Name: status_group, dtype: float64
```

In [30]:

```
1 fig, (ax1, ax2) = plt.subplots(2, figsize=(12,10))
2 fig.suptitle('Water Quality Variables')
3 sns.countplot(x='water_quality', hue='status_group', data=df_binary, ax=1)
4 sns.countplot(x='quality_group', hue='status_group', data=df_binary, ax=2)
5 plt.show();
```

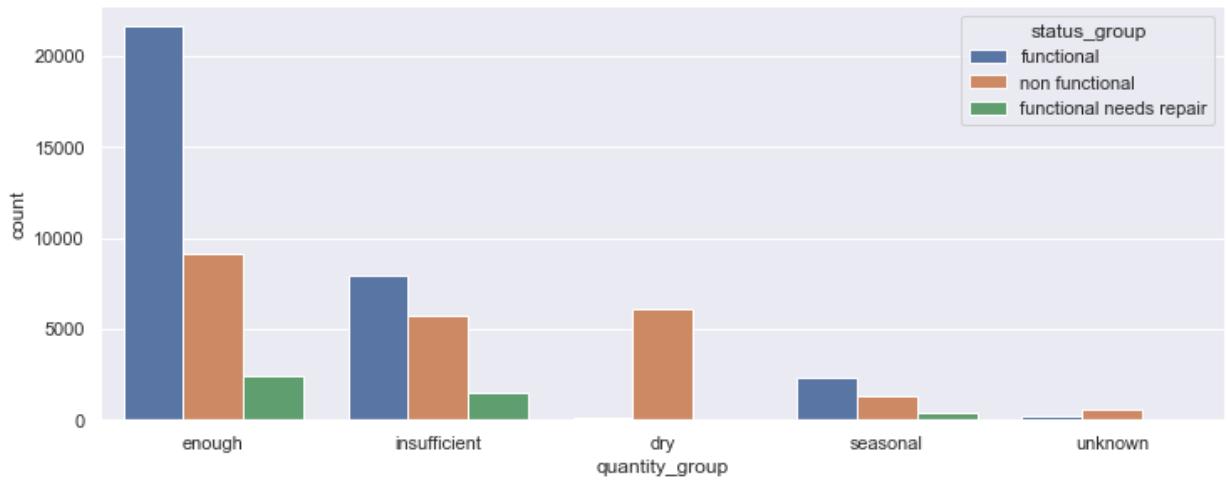
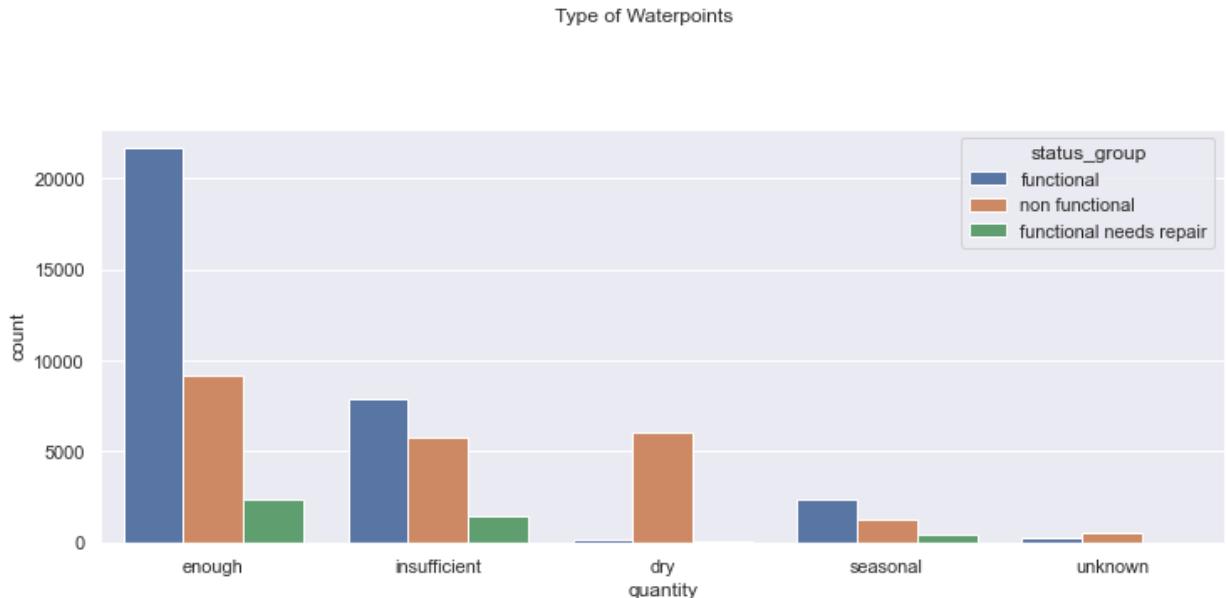


quantity and quantity_group

These columns are identical. Drop quantity group.

Quantity describes the amount of water available. Dry pumps are non-functional, as expected. It is crucial that each pump has a sufficient water supply.

```
In [31]: 1 fig, (ax1, ax2) = plt.subplots(2, figsize=(12,10))
2 fig.suptitle('Type of Waterpoints')
3 sns.countplot(x='quantity', hue='status_group', data=df, ax=ax1)
4 sns.countplot(x='quantity_group', hue='status_group', data=df, ax=ax2)
5 plt.show();
```



Above: tertiary

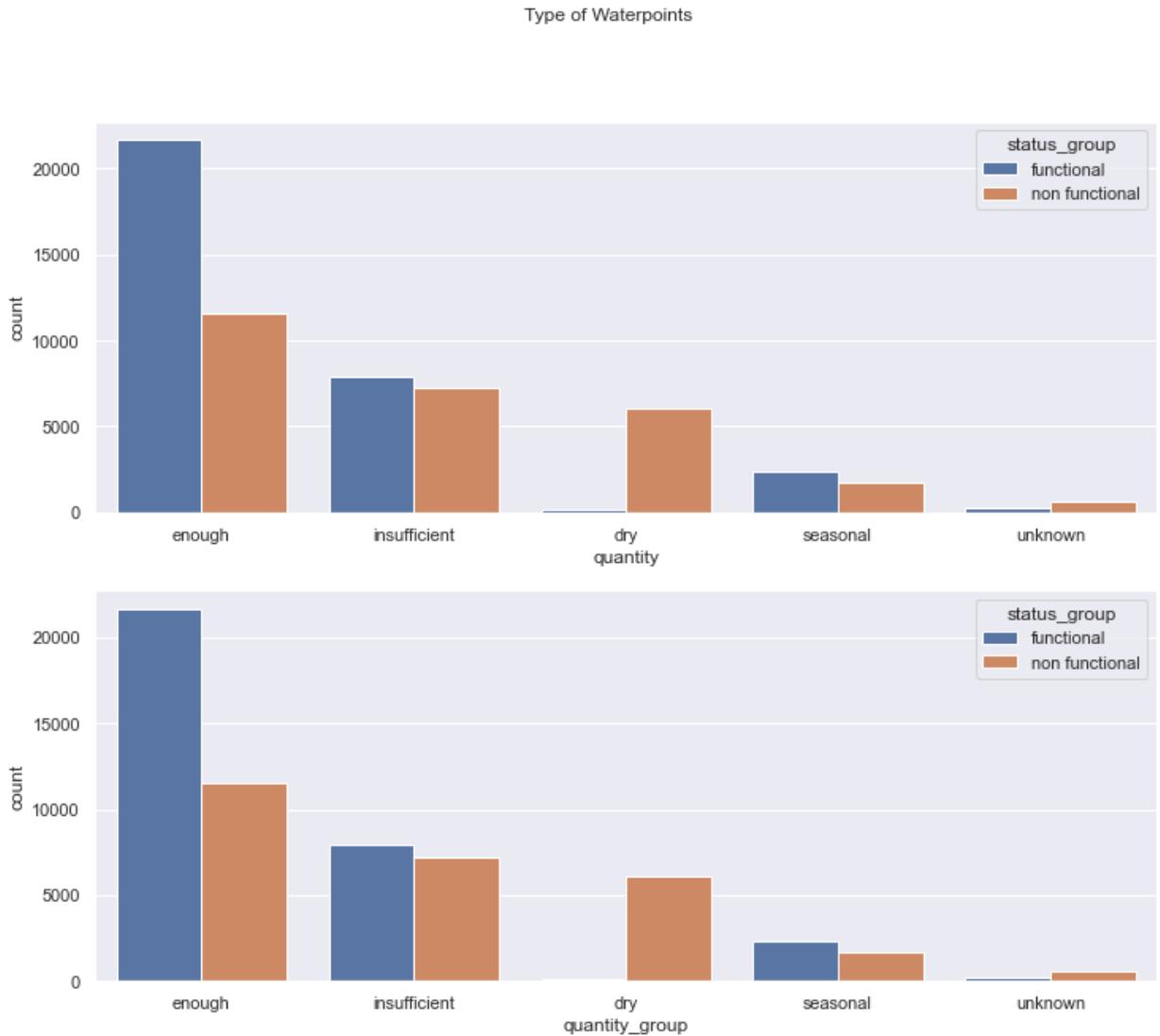
Below: Binary

In [32]:

```

1 fig, (ax1, ax2) = plt.subplots(2, figsize=(12,10))
2 fig.suptitle('Type of Waterpoints')
3 sns.countplot(x='quantity', hue='status_group', data=df_binary, ax=ax1)
4 sns.countplot(x='quantity_group', hue='status_group', data=df_binary, ax=ax2)
5 plt.show();

```



waterpoint_type and waterpoint_type_group

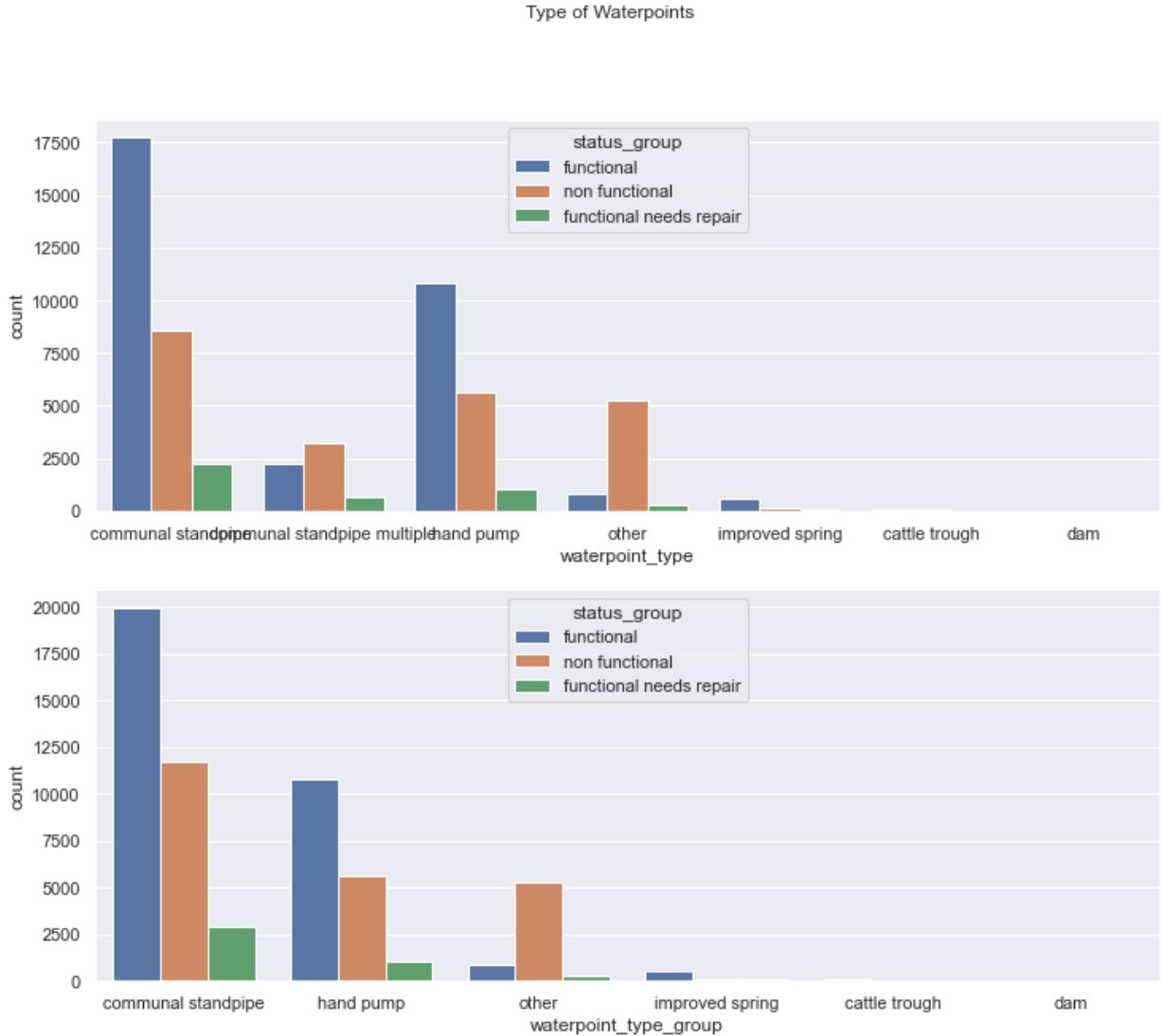
Similar to what we saw above with water quality, both of these columns contain essentially the same information, subgrouped differently. I decided to keep 'waterpoint_type' and drop 'waterpoint_type_group' in order to avoid overfitting issues and speed up the iterative modeling process.

In [33]:

```

1 fig, (ax1, ax2) = plt.subplots(2, figsize=(12,10))
2 fig.suptitle('Type of Waterpoints')
3 sns.countplot(x='waterpoint_type', hue='status_group', data=df, ax=ax1)
4 sns.countplot(x='waterpoint_type_group', hue='status_group', data=df, a
5 plt.show();

```



Above: tertiary

Below: Binary

source, source_class, and source_type

source: Explains same information as source_type but with more variables. Keeping source_type.

source_class: Broadly explains same information as source and source_type. Keeping source_type.

Source_type: This was NOT dropped. Kept it as a balance between the other two features of similar information.

See below for visualization of the different groupings in terms of functional water pumps.

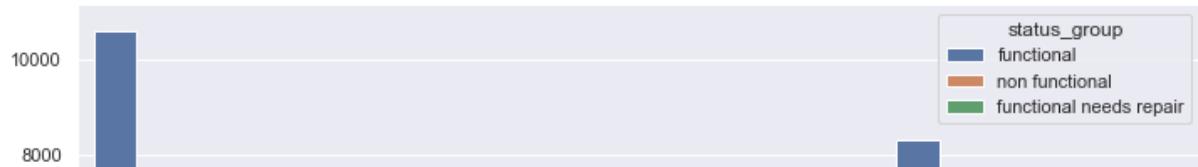
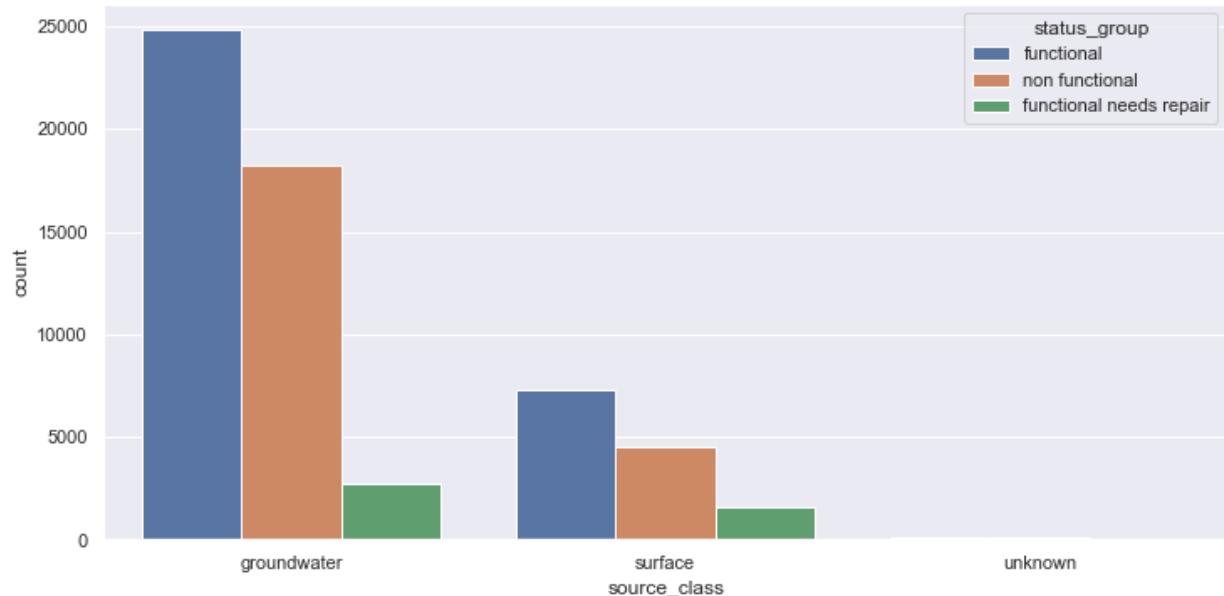
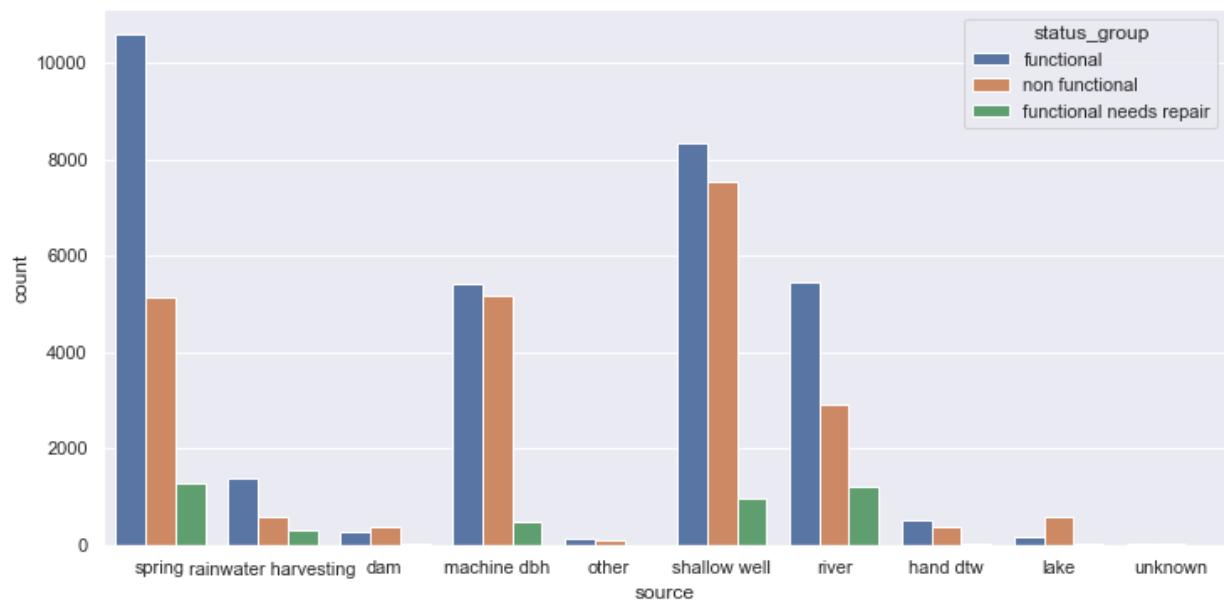
In [34]:

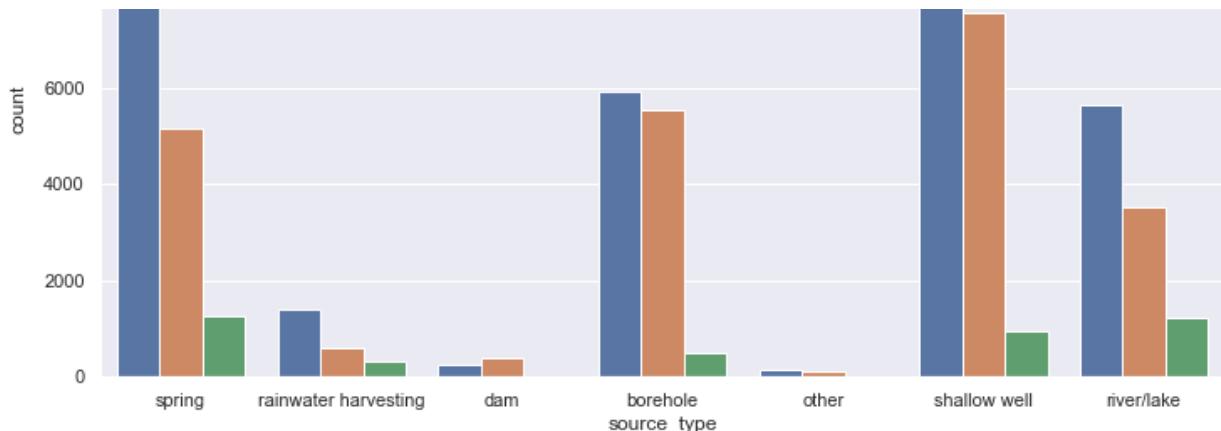
```

1 fig, (ax1, ax2, ax3) = plt.subplots(3, figsize=(12,20))
2 fig.suptitle('water source')
3 sns.countplot(x='source', hue='status_group', data=df, ax=ax1)
4 sns.countplot(x='source_class', hue='status_group', data=df, ax=ax2)
5 sns.countplot(x='source_type', hue='status_group', data=df, ax=ax3)
6 plt.show();

```

water source





Above: tertiary

Below: Binary

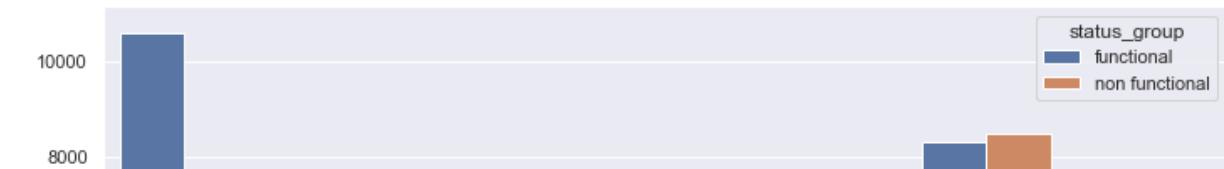
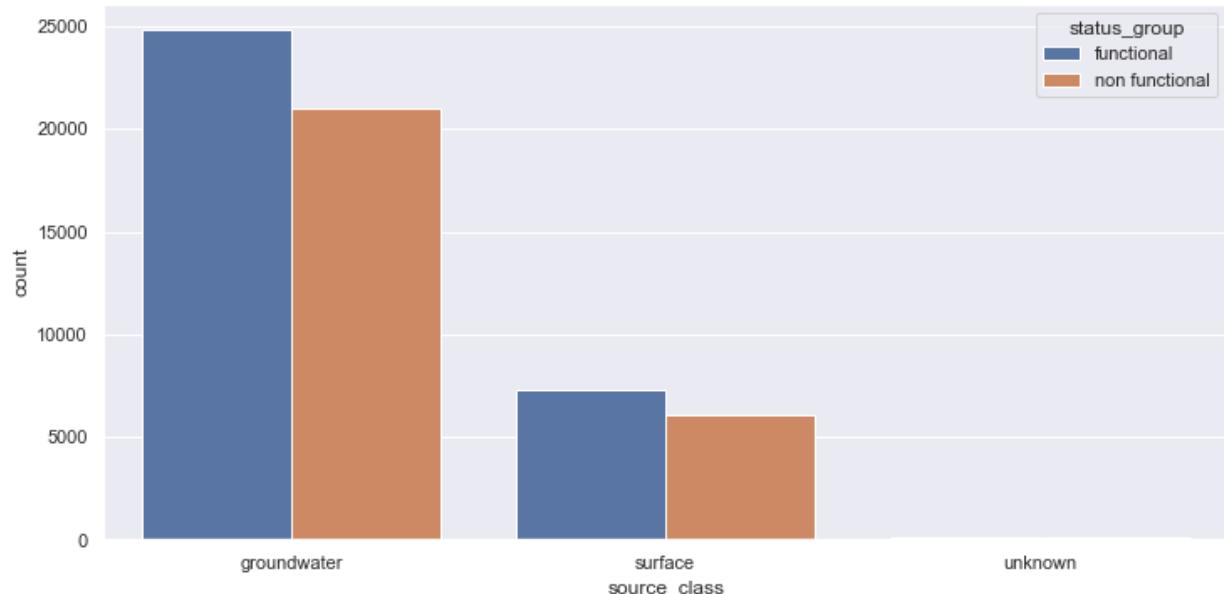
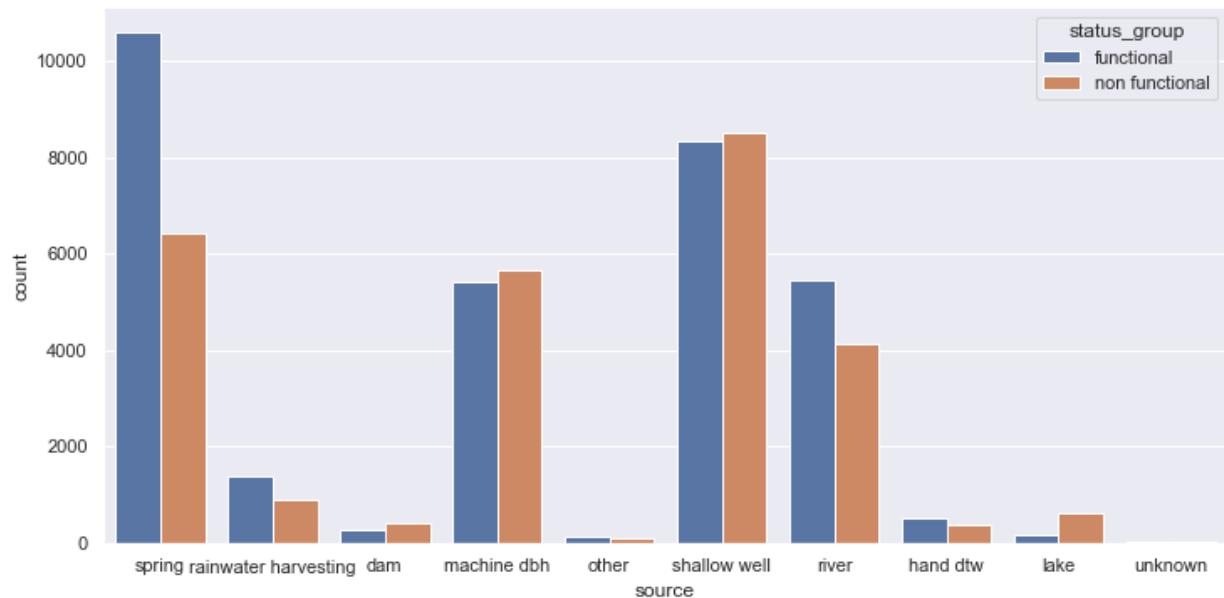
In [35]:

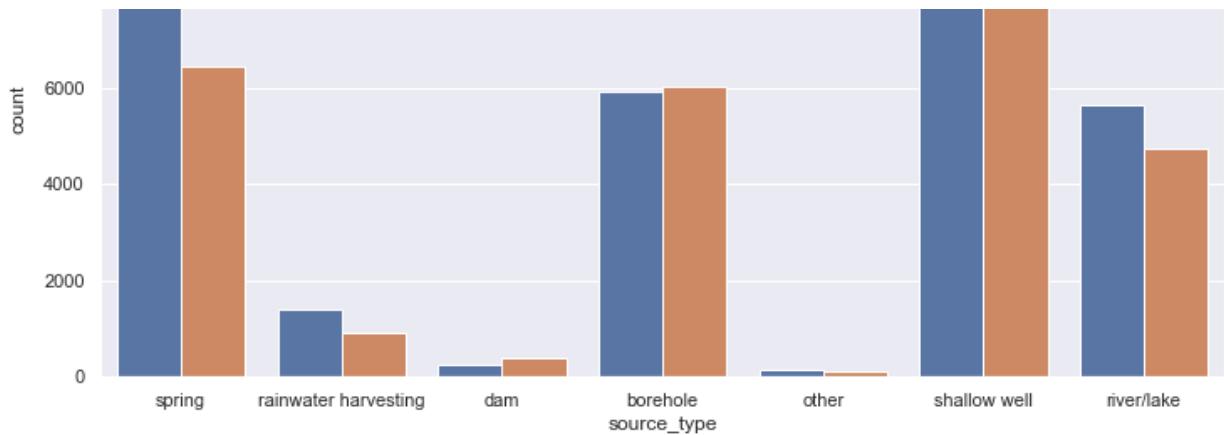
```

1 fig, (ax1, ax2, ax3) = plt.subplots(3, figsize=(12,20))
2 fig.suptitle('water source')
3 sns.countplot(x='source', hue='status_group', data=df_binary, ax=ax1)
4 sns.countplot(x='source_class', hue='status_group', data=df_binary, ax=ax2)
5 sns.countplot(x='source_type', hue='status_group', data=df_binary, ax=ax3)
6 plt.show();

```

water source





extraction_type, extraction_type_group, extraction_type_class

These three features explain a lot of the same information subgrouped in different ways. I decided to keep two and drop the third. The two that were dropped had columns with too little weight in the distribution for useful information.

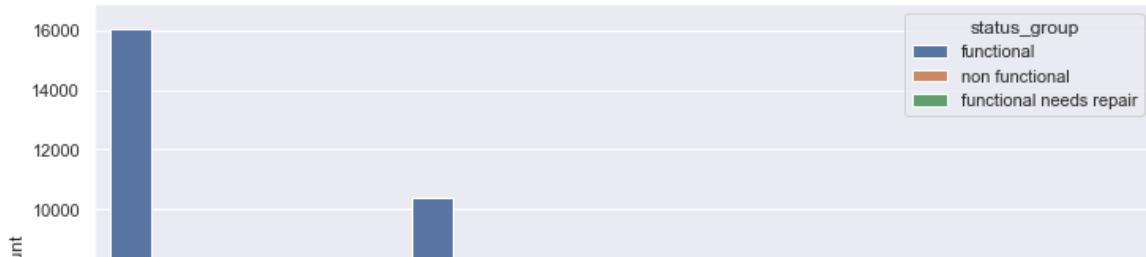
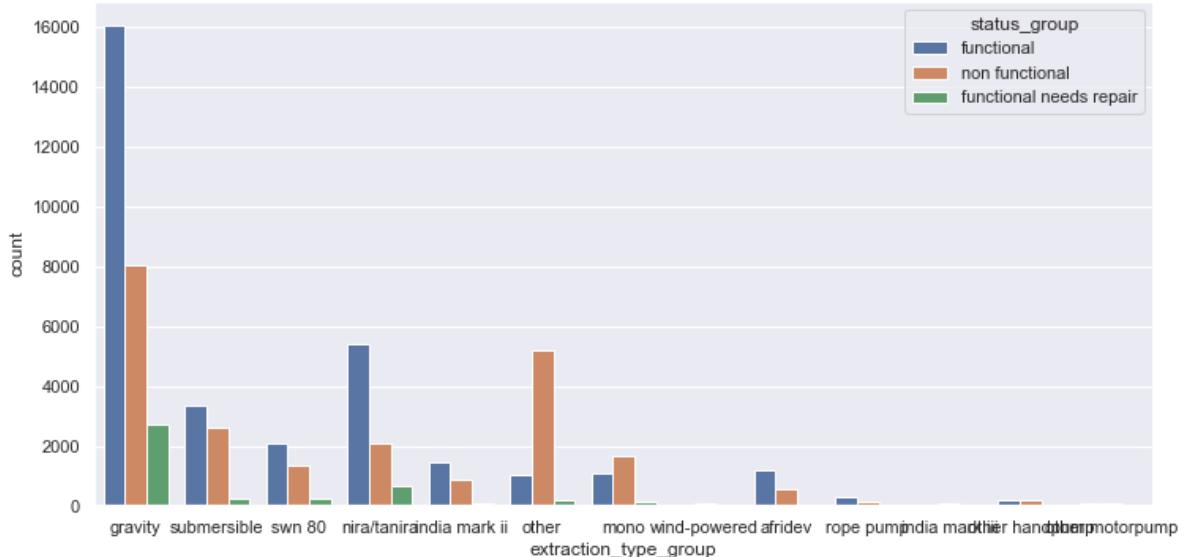
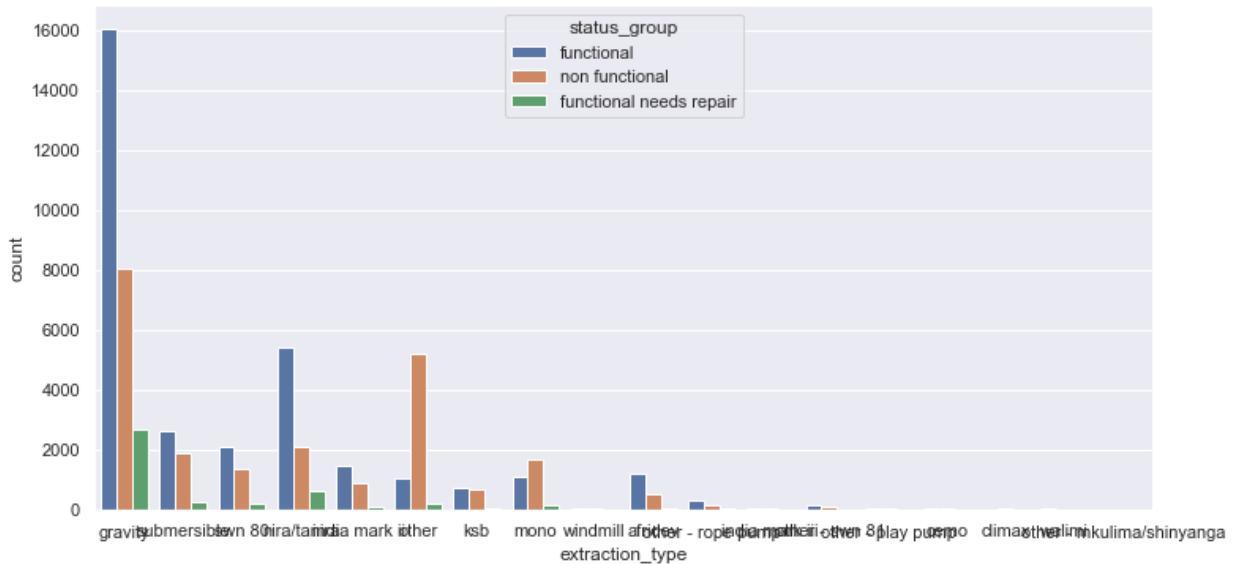
In [36]:

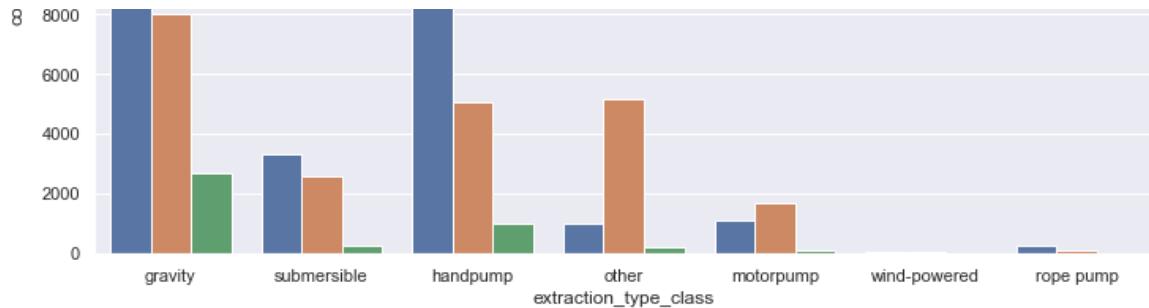
```

1 fig, (ax1, ax2, ax3) = plt.subplots(3, figsize=(12,20))
2 fig.suptitle('extraction process')
3 sns.countplot(x='extraction_type', hue='status_group', data=df, ax=ax1)
4 sns.countplot(x='extraction_type_group', hue='status_group', data=df, a
5 sns.countplot(x='extraction_type_class', hue='status_group', data=df, a
6 plt.show();

```

extraction process





Above: tertiary

Below: Binary

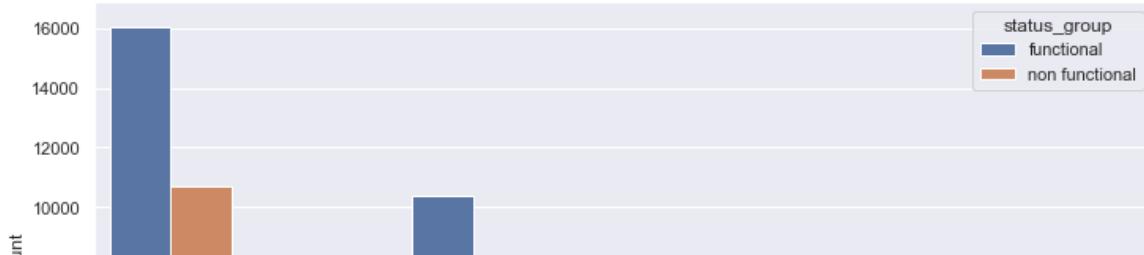
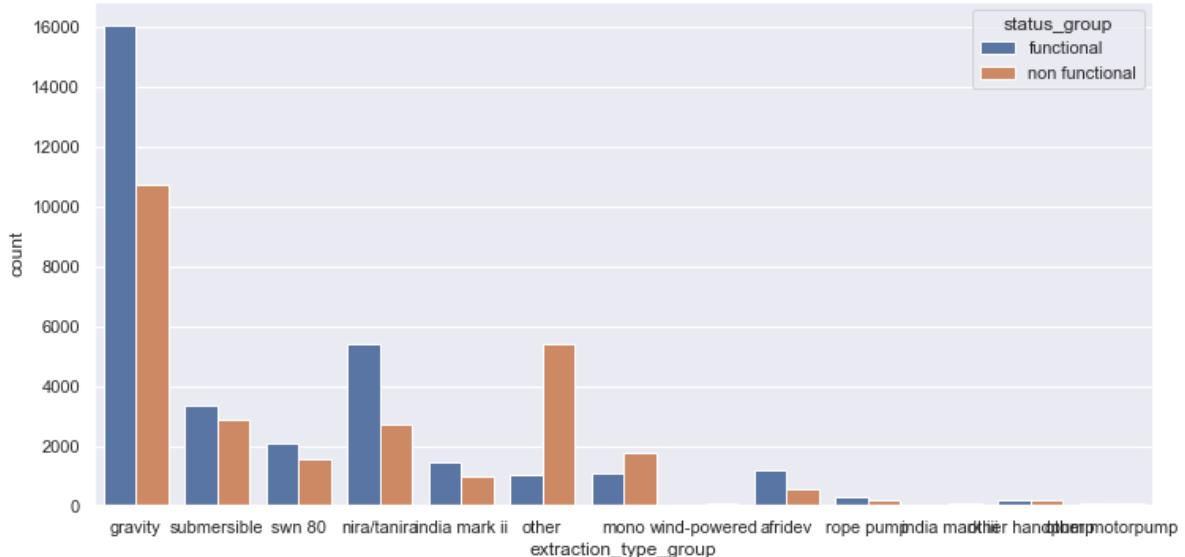
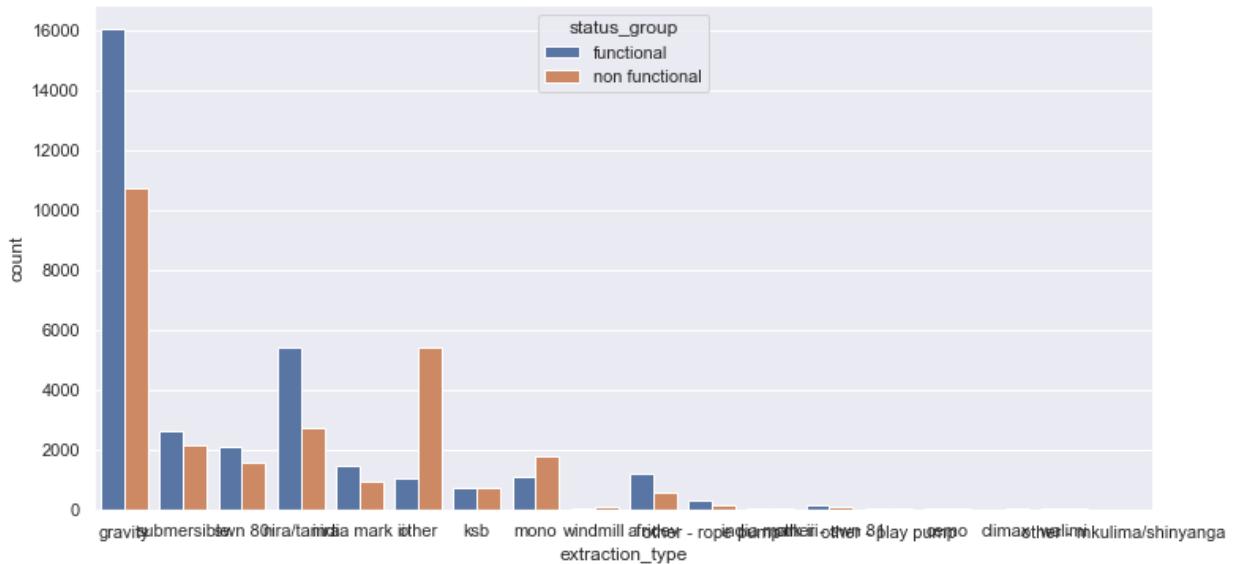
In [37]:

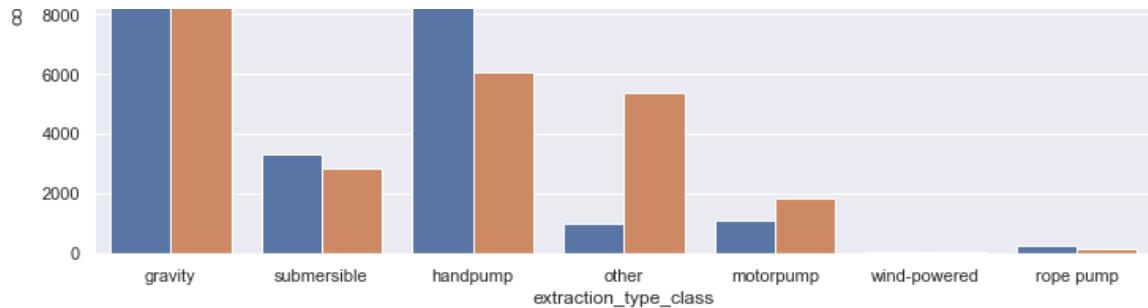
```

1 fig, (ax1, ax2, ax3) = plt.subplots(3, figsize=(12,20))
2 fig.suptitle('extraction process')
3 sns.countplot(x='extraction_type', hue='status_group', data=df_binary,
4 sns.countplot(x='extraction_type_group', hue='status_group', data=df_bi
5 sns.countplot(x='extraction_type_class', hue='status_group', data=df.bi
6 plt.show();

```

extraction process





payment and payment_type

The information for both of these columns is nearly exactly identical. Drop 'payment_type' and keep 'payment'.

Most points that are not paid are not working. This may be due to a lack of maintenance resources. When a payment is added, the status changes.

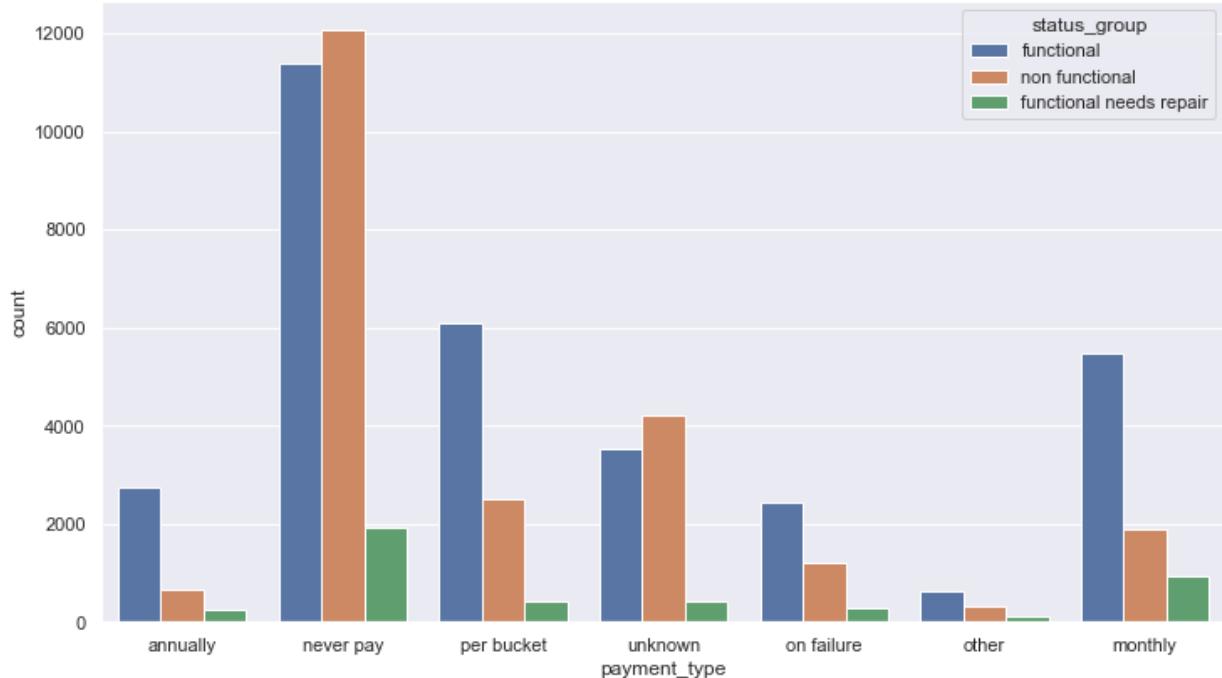
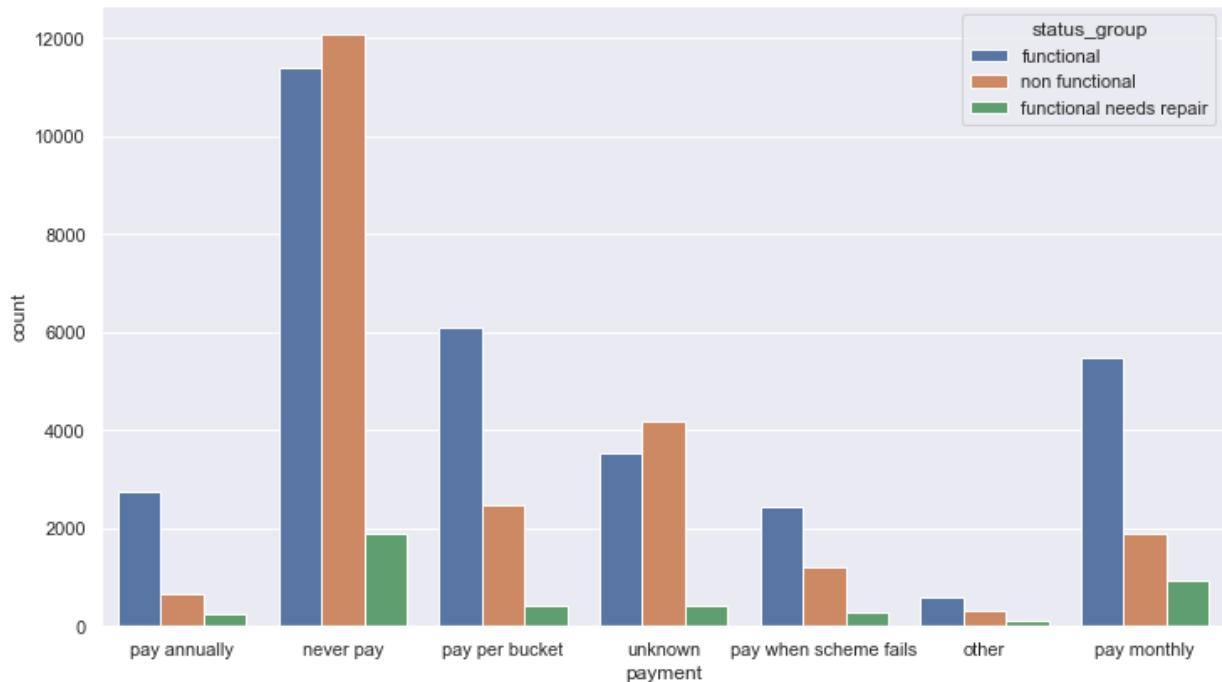
In [38]:

```

1 fig, (ax1, ax2) = plt.subplots(2, figsize=(12,15))
2 fig.suptitle('payment information')
3 sns.countplot(x='payment', hue='status_group', data=df, ax=ax1)
4 sns.countplot(x='payment_type', hue='status_group', data=df, ax=ax2)
5 plt.show();

```

payment information



Above: tertiary

Below: Binary

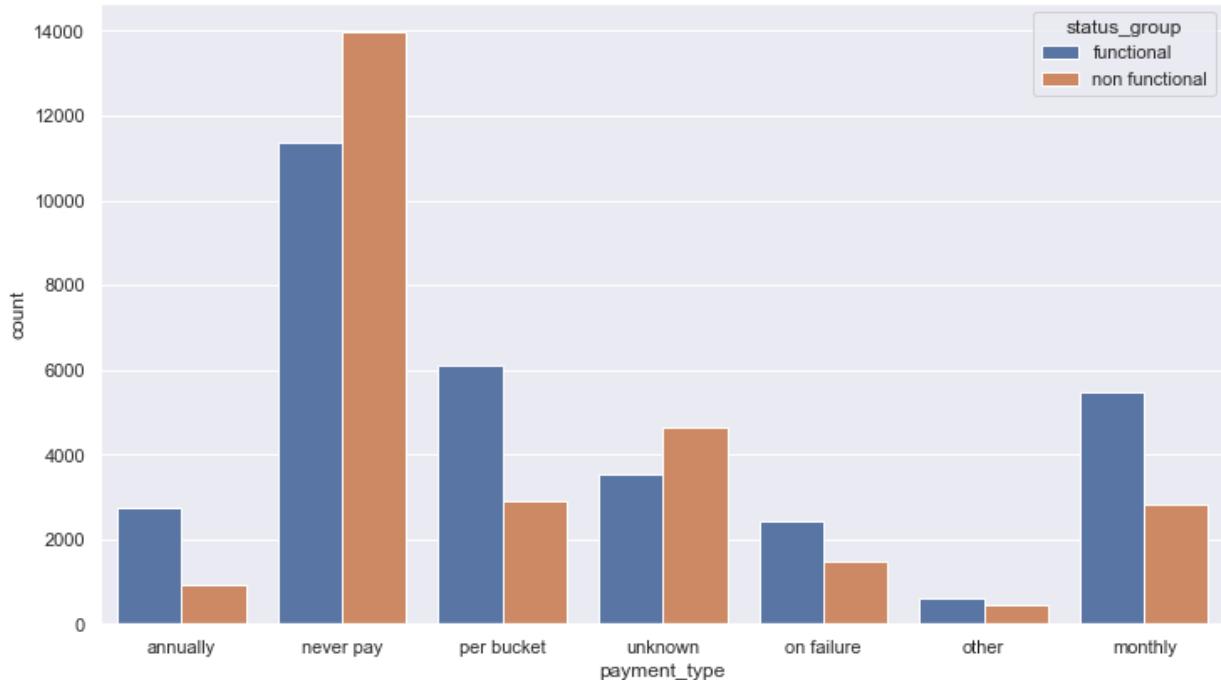
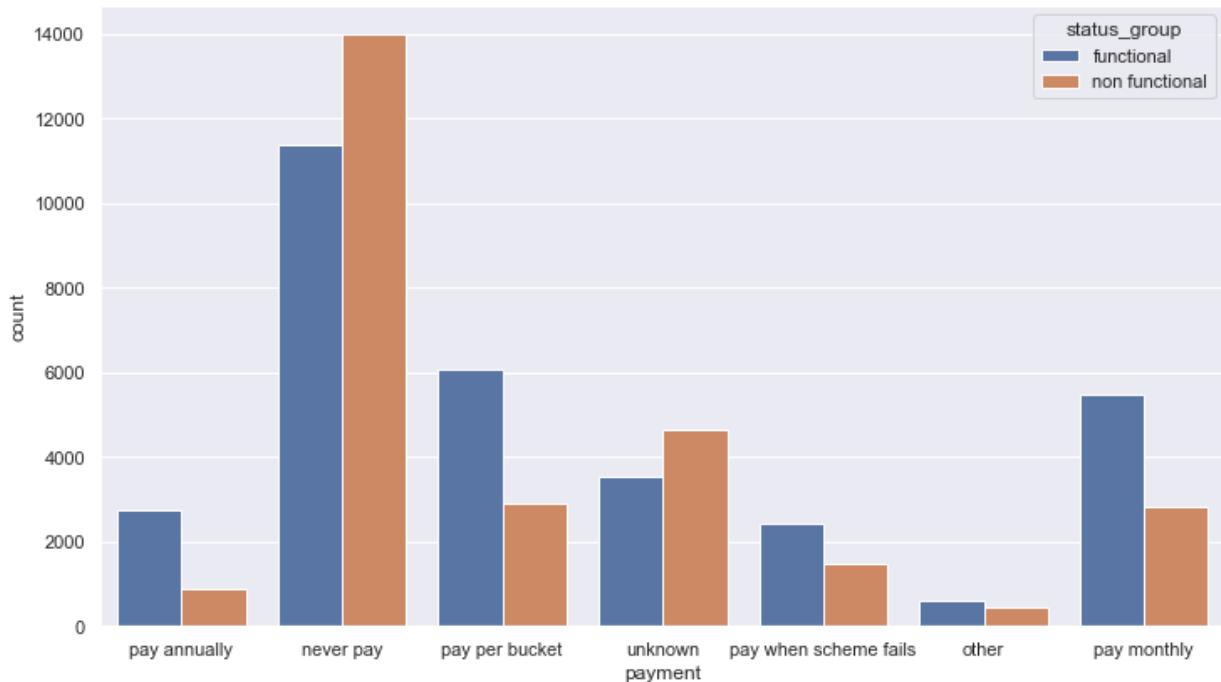
In [39]:

```

1 fig, (ax1, ax2) = plt.subplots(2, figsize=(12,15))
2 fig.suptitle('payment information')
3 sns.countplot(x='payment', hue='status_group', data=df_binary, ax=ax1)
4 sns.countplot(x='payment_type', hue='status_group', data=df_binary, ax=
5 plt.show();

```

payment information



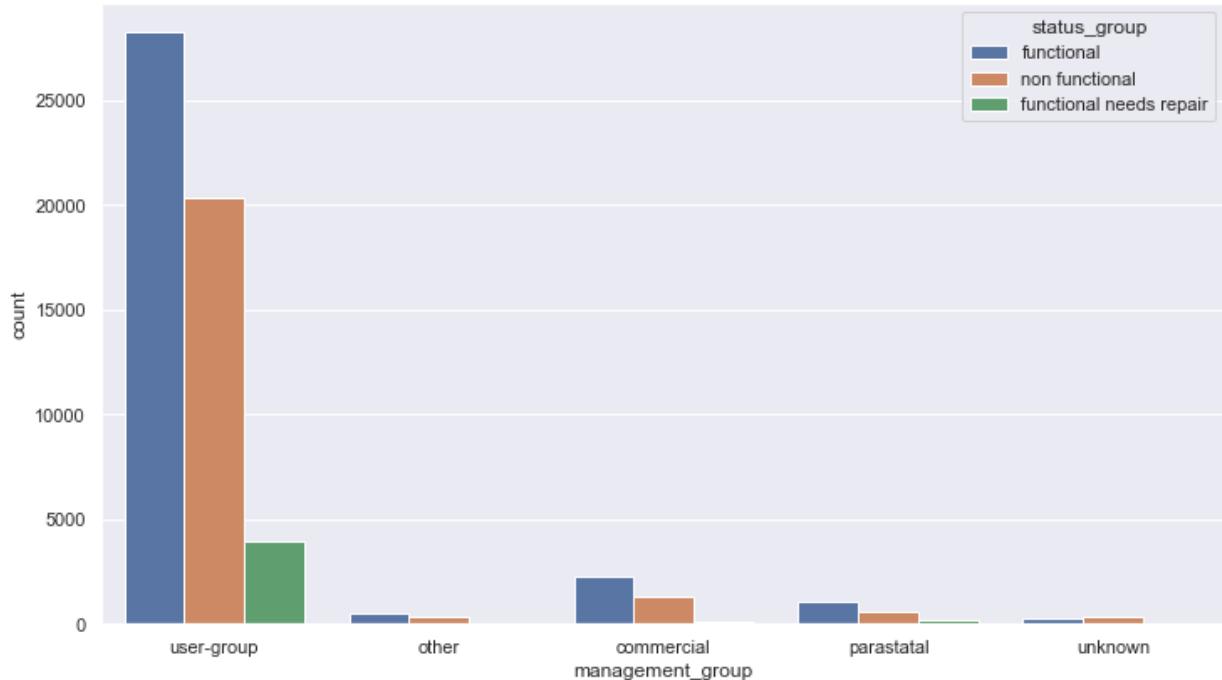
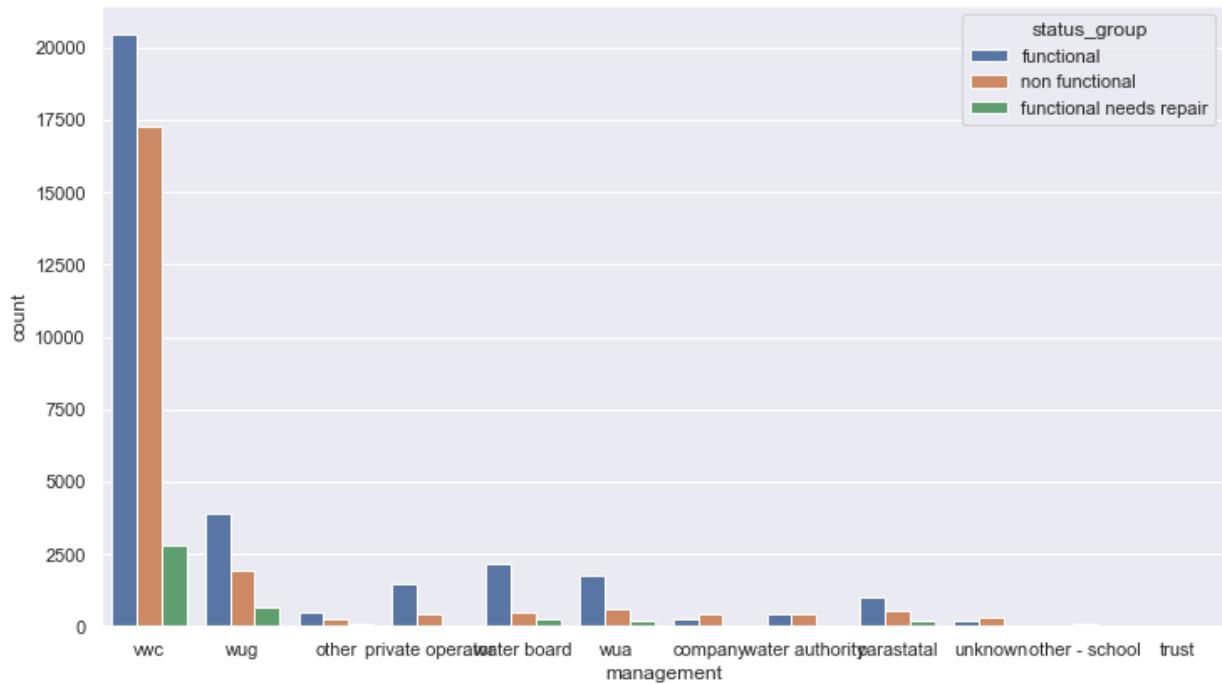
management and management_group

I decided to keep the 'management' column and group the less specific 'management_group' column. A lot of the columns in the management column should have enough information to help the model make better predictions.

In [40]:

```
1 fig, (ax1, ax2) = plt.subplots(2, figsize=(12,15))
2 fig.suptitle('management information')
3 sns.countplot(x='management', hue='status_group', data=df, ax=ax1)
4 sns.countplot(x='management_group', hue='status_group', data=df, ax=ax2)
5 plt.show();
```

management information



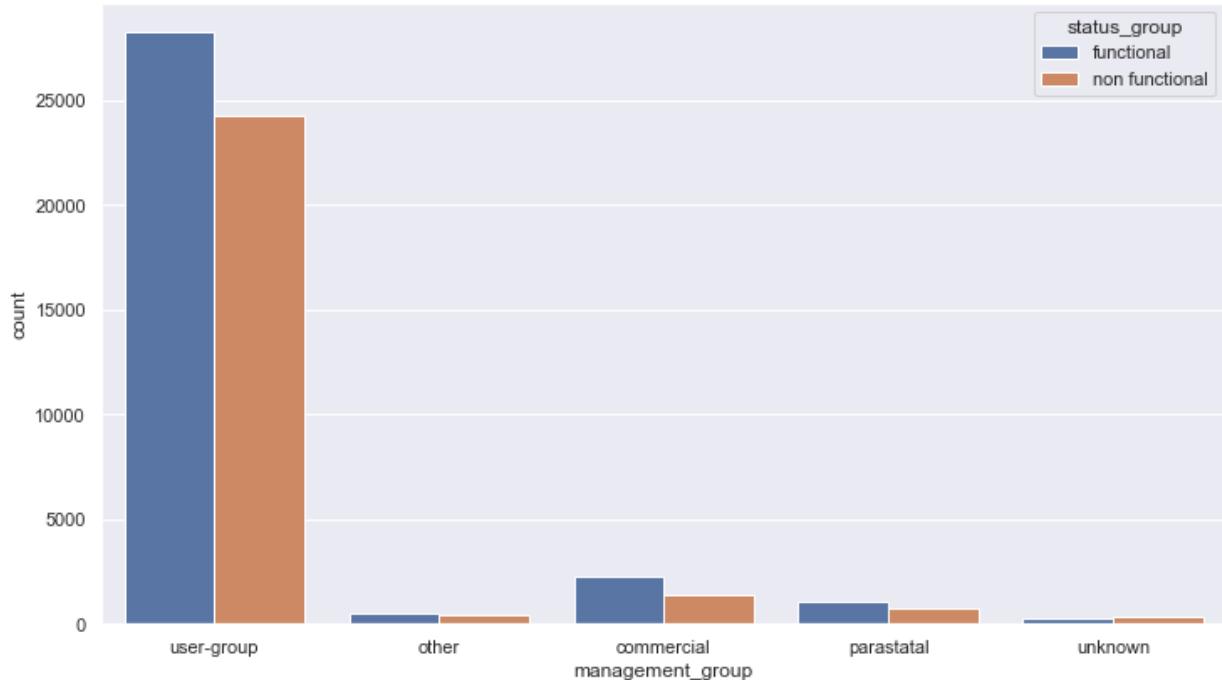
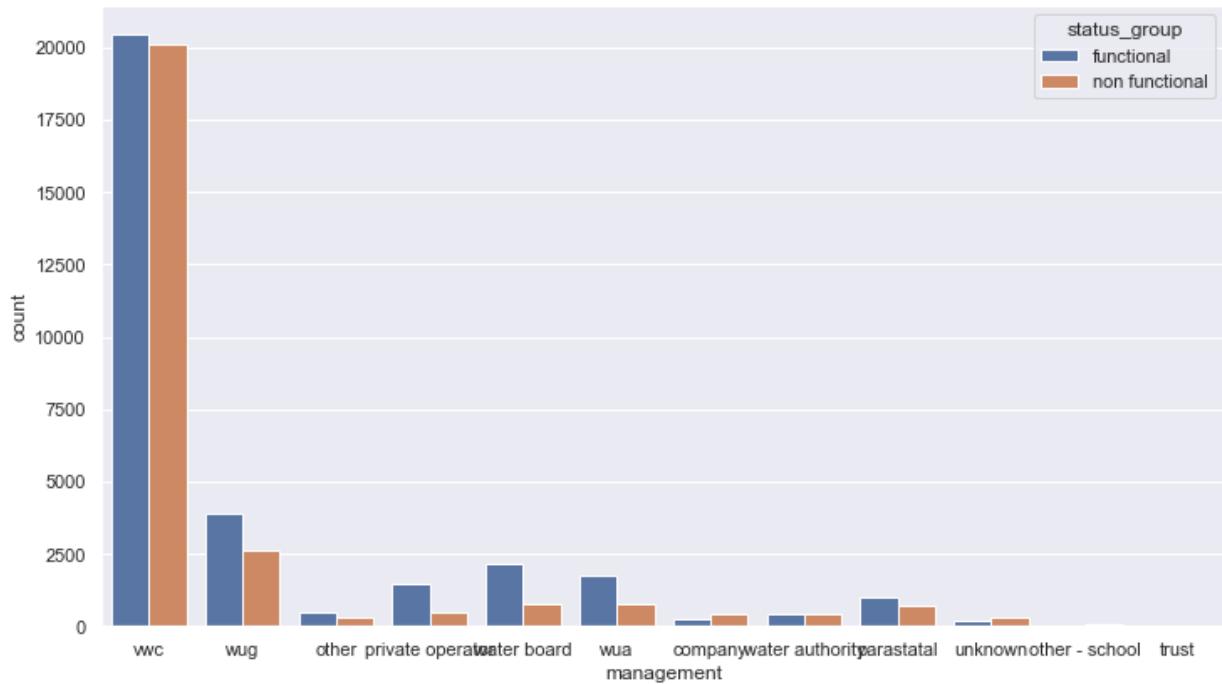
Above: tertiary

Below: Binary

In [41]:

```
1 fig, (ax1, ax2) = plt.subplots(2, figsize=(12,15))
2 fig.suptitle('management information')
3 sns.countplot(x='management', hue='status_group', data=df_binary, ax=ax1)
4 sns.countplot(x='management_group', hue='status_group', data=df_binary, ax=ax2)
5 plt.show();
```

management information



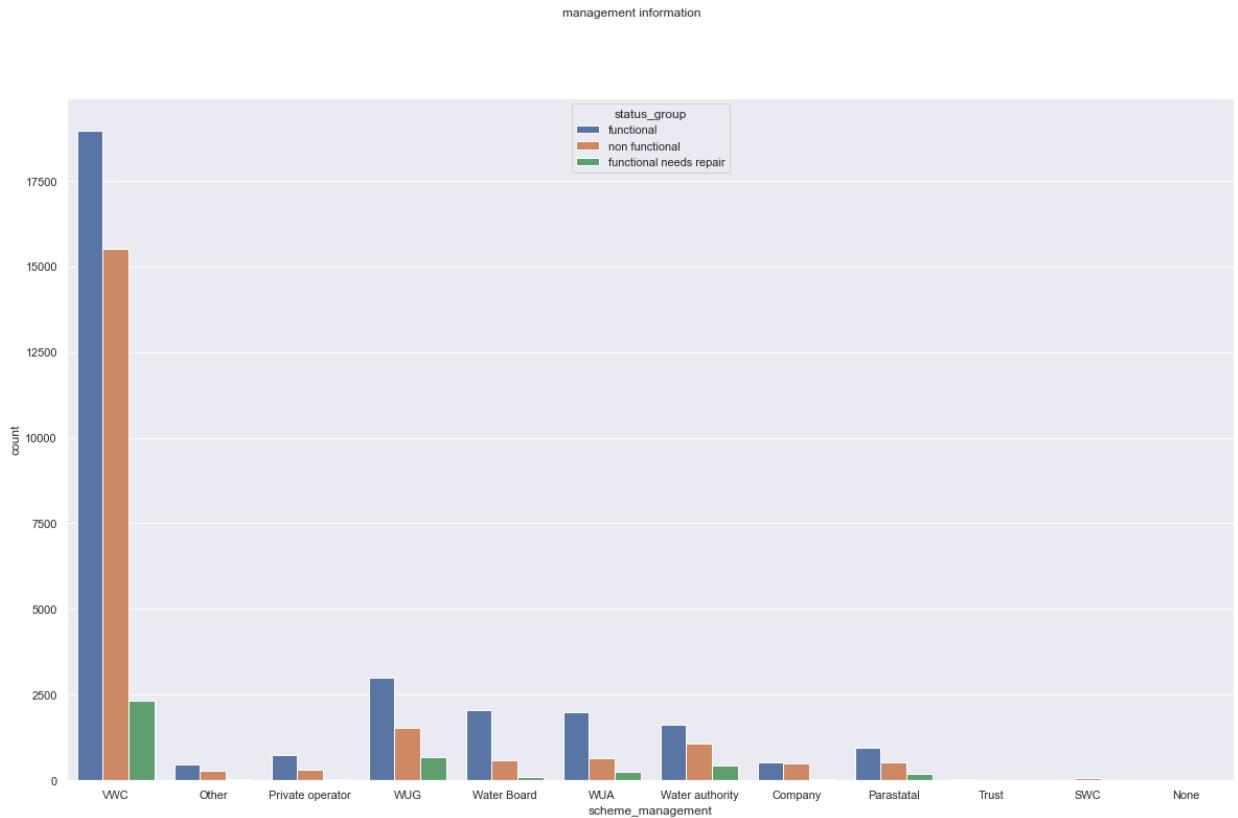
scheme_name and scheme_management

scheme_name: drop. Missing almost half of values. Large cardinality (nearly 2700 categories).

scheme_manager: drop. Contains similar information to management column.

In [42]:

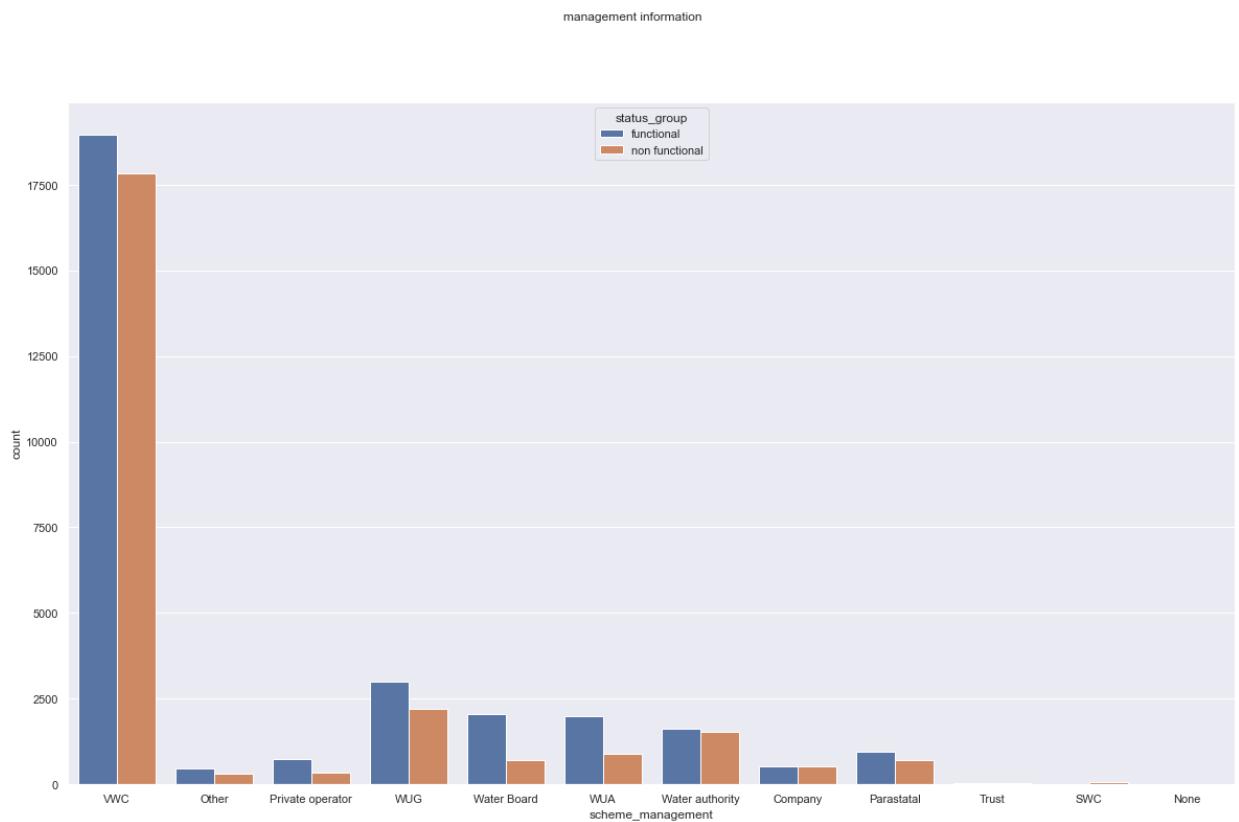
```
1 fig = plt.figure(figsize=(20,12))
2 fig.suptitle('management information')
3 sns.countplot(x='scheme_management', hue='status_group', data=df)
4 plt.show();
```



Above: tertiary

Below: Binary

```
In [43]: 1 fig = plt.figure(figsize=(20,12))
2 fig.suptitle('management information')
3 sns.countplot(x='scheme_management', hue='status_group', data=df_binary)
4 plt.show();
```



In [44]:

```

1 to_drop = ['id', 'recorded_by', 'num_private',
2           'waterpoint_type_group', 'source',
3           'source_class', 'extraction_type',
4           'extraction_type_group', 'payment_type',
5           'management_group', 'scheme_name',
6           'water_quality', 'quantity_group',
7           'scheme_management', 'longitude',
8           'latitude', 'date_recorded']
9 df2 = df.drop(to_drop, axis=1)
10 df2.head()

```

Out[44]:

	amount_tsh	funder	gps_height	installer	wpt_name	basin	subvillage	region	region_c
0	6000.0	Roman	1390	Roman	none	Lake Nyasa	Mnyusi B	Iringa	
1	0.0	Grumeti	1399	GRUMETI	Zahanati	Lake Victoria	Nyamara	Mara	
2	25.0	Lottery Club	686	World vision	Kwa Mahundi	Pangani	Majengo	Manyara	
3	0.0	Unicef	263	UNICEF	Zahanati Ya Nanyumbu	Ruvuma / Southern Coast	Mahakamani	Mtwara	
4	0.0	Action In A	0	Artisan	Shuleni	Lake Victoria	Kyanyamisa	Kagera	

5 rows × 25 columns

Date_Recorded: Understanding Time Frame of Data Collection and Current Capacity Limits

As shown in the bar graph below, almost all of the data was originally recorded between the years 2011 and 2013. The highest capacity for the Ministry of Water was in the year 2011 when 28,674 pumps were visited. It is important to consider the time-scale of the data when considering the cost/resource analysis of predictive modeling vs the current maintenance being conducted with no modeling.

In [45]:

```

1 df3=df.copy()
2 df3['date_recorded'] = pd.to_datetime(df3['date_recorded'])

```

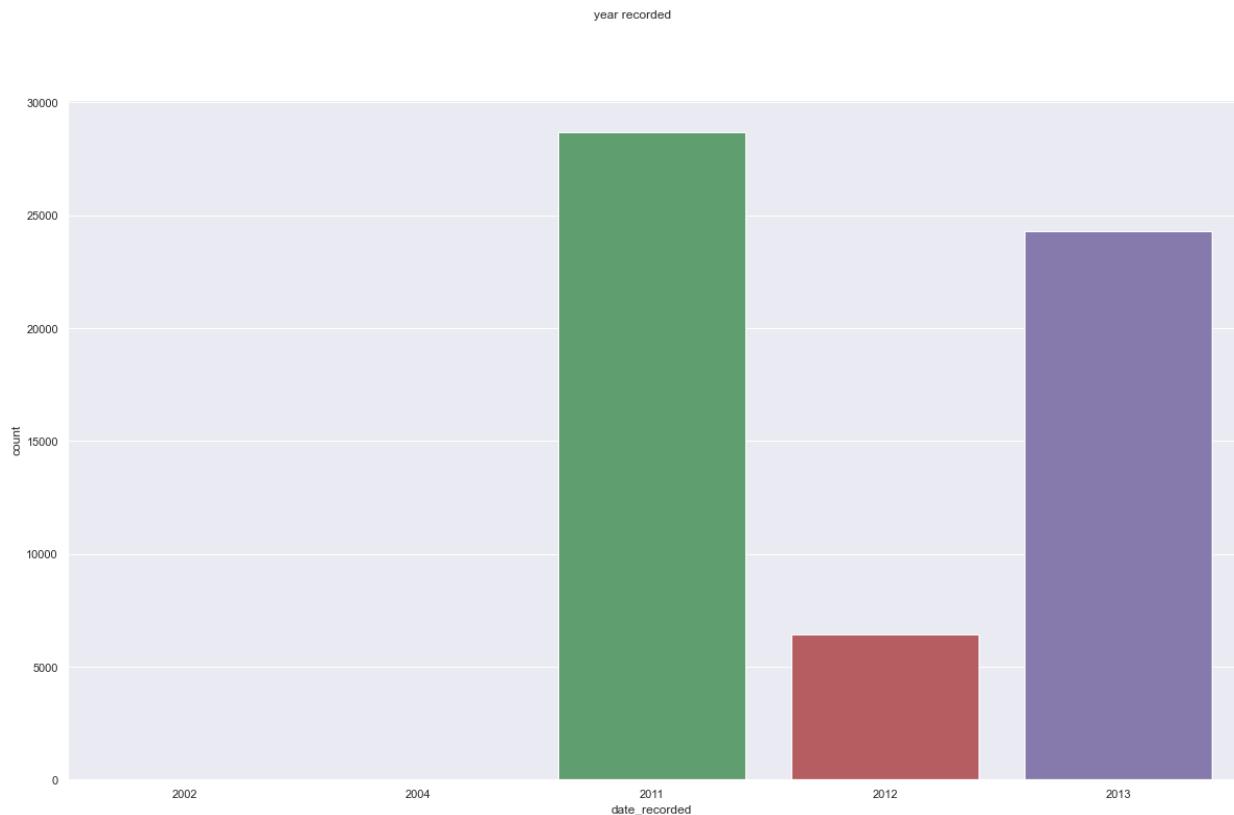
```
In [46]: 1 df3['date_recorded'].dt.year.value_counts()
```

```
Out[46]: 2011    28674
2013    24271
2012    6424
2004      30
2002       1
Name: date_recorded, dtype: int64
```

```
In [47]: 1 df3.shape
```

```
Out[47]: (59400, 42)
```

```
In [48]: 1 fig = plt.figure(figsize=(20,12))
2 fig.suptitle('year recorded')
3 sns.countplot(x=df3['date_recorded'].dt.year)
4 plt.show();
```



Speculate on the amount of labor required to physically check all of these pumps in a 2 to 3 year period. This will give a good understanding of the current cost and capacity without using any predictive maintenance.

Dealing with Null Values and '0' placeholder values

This section covers the handling of null values in the data in preparation for modeling.

funder and Installer null values

```
In [49]: 1 print(df2['funder'].isna().sum())
2 print(df2['installer'].isna().sum())
3 df2[df2['funder'].isna()].index.isin(df2[df2['installer'].isna()].index)
```

3635
3655

Out[49]: 3582

```
In [50]: 1 df2['installer'].value_counts()[:20]
```

```
Out[50]: DWE                17402
Government          1825
RWE                1206
Commu              1060
DANIDA              1050
KKKT                  898
Hesawa              840
0                      777
TCRS                  707
Central government    622
CES                  610
Community            553
DANID                552
District Council      551
HESAWA                539
World vision          408
LGA                  408
WEDECO                397
TASAF                  396
District council        392
Name: installer, dtype: int64
```

```
In [51]: 1 df2['funder'].value_counts()[:20]
```

```
Out[51]: Government Of Tanzania    9084
Danida          3114
Hesawa          2202
Rwssp           1374
World Bank      1349
Kkkt            1287
World Vision    1246
Unicef          1057
Tasaf           877
District Council 843
Dhv             829
Private Individual 826
Dwsp            811
0               777
Norad           765
Germany Republi 610
Tcrs            602
Ministry Of Water 590
Water           583
Dwe             484
Name: funder, dtype: int64
```

In [52]:

```

1 installer_index_0 = df2['installer'] == '0'
2 funder_index_0 = df2['funder'] == '0'
3 #replace zero placeholders with 'unknown'
4 df2.loc[funder_index_0, 'funder'] = 'unknown'
5 df2.loc[funder_index_0, 'installer'] = 'unknown'
6 #replace null values as 'unkown'
7 df2['installer'].fillna(value='unknown', inplace=True)
8 df2['funder'].fillna(value='unknown', inplace=True)
9 #sanity check
10 display(df2['funder'].value_counts()[:20])
11 display(df2['installer'].value_counts()[:20])
12 df2.isna().sum()

```

Government Of Tanzania	9084
unknown	4412
Danida	3114
Hesawa	2202
Rwssp	1374
World Bank	1349
Kkkt	1287
World Vision	1246
Unicef	1057
Tasaf	877
District Council	843
Dhv	829
Private Individual	826
Dwsp	811
Norad	765
Germany Republi	610
Tcrs	602
Ministry Of Water	590
Water	583
Dwe	484

Name: funder, dtype: int64

DWE	17402
unknown	4433
Government	1825
RWE	1206
Commu	1060
DANIDA	1050
KKKT	898
Hesawa	840
TCRS	707
Central government	622
CES	610
Community	553
DANID	552
District Council	551
HESAWA	539
World vision	408
LGA	408
WEDECO	397
TASAF	396
District council	392

Name: installer, dtype: int64

```
Out[52]: amount_tsh          0
funder                  0
gps_height               0
installer                0
wpt_name                 0
basin                   0
subvillage              371
region                  0
region_code              0
district_code            0
lga                      0
ward                     0
public_meeting           3334
permit                  3056
construction_year         0
extraction_type_class    0
management               0
payment                  0
quality_group             0
quantity                 0
source_type               0
waterpoint_type           0
status_group              0
near_river                0
ward_pop                  0
dtype: int64
```

public meeting null values

```
In [53]: 1 display(df2['public_meeting'].value_counts(normalize=True))
2 df2['public_meeting'].value_counts(dropna=False,normalize=True)
```

```
True      0.909838
False     0.090162
Name: public_meeting, dtype: float64
```

```
Out[53]: True      0.858771
False     0.085101
NaN       0.056128
Name: public_meeting, dtype: float64
```

In [54]:

```

1 #create copy of dataframe
2 df_pub_no_na = df2.copy()
3 #initialize boolean series for null values in pub meeting
4 filt = df_pub_no_na['public_meeting'].isna()
5 #initialize series in order to keep ratio of boolean the same
6 #after filling null values
7 probs = df2['public_meeting'].value_counts(normalize=True)
8 #fill null values with random selection of
9 #True or False but keep same probability as before filling NaNs
10 df_pub_no_na.loc[filt, 'public_meeting'] = np.random.choice([True, False],
11 size=int(filt.sum())),
12 p = [probs[True], probs[False]])
13
14 #check to see NaNs are filled without changing probabilities
15 df_pub_no_na['public_meeting'].value_counts(dropna=False, normalize = True)

```

Out[54]: True 0.910118
 False 0.089882
 Name: public_meeting, dtype: float64

permit null values

In [55]:

```

1 df3 = df_pub_no_na.copy()
2 df3.isna().sum()

```

amount_tsh	0
funder	0
gps_height	0
installer	0
wpt_name	0
basin	0
subvillage	371
region	0
region_code	0
district_code	0
lga	0
ward	0
public_meeting	0
permit	3056
construction_year	0
extraction_type_class	0
management	0
payment	0
quality_group	0
quantity	0
source_type	0
waterpoint_type	0
status_group	0
near_river	0
ward_pop	0
dtype:	int64

```
In [56]: 1 df3['permit'].value_counts(dropna=False, normalize=True)
```

```
Out[56]: True      0.654074
False     0.294478
NaN       0.051448
Name: permit, dtype: float64
```

```
In [57]: 1 #initialize boolean series for null values in permit
2 filt2 = df3['permit'].isna()
3 #initialize series in order to keep ratio of boolean the same
4 #after filling null values
5 probs2 = df3['permit'].value_counts(normalize=True)
6 #fill null values with random selection of
7 #True or False but keep same probability as before filling NaNs
8 df3.loc[filt2, 'permit'] = np.random.choice([True, False],
9                     size=int(filt2.sum())),
10                    p = [probs2[True], probs2[False]])
11
12 df3['permit'].value_counts(dropna=False, normalize=True)
```

```
Out[57]: True      0.689377
False     0.310623
Name: permit, dtype: float64
```

subvillage null values

```
In [58]: 1 #grab indexes of null values to fill for scheme_management column
2 subvillage_index_null = df3[df3['subvillage'].isna()].index
3 #fill null values with 'unknown'
4 df3.loc[subvillage_index_null, 'subvillage'] = 'unknown'
5
6 df3.isna().sum()
```

```
Out[58]: amount_tsh          0
funder              0
gps_height          0
installer           0
wpt_name            0
basin               0
subvillage          0
region              0
region_code         0
district_code       0
lga                 0
ward                0
public_meeting      0
permit              0
construction_year   0
extraction_type_class 0
management          0
payment              0
quality_group        0
quantity             0
source_type          0
waterpoint_type     0
status_group         0
near_river           0
ward_pop             0
dtype: int64
```

High Cardinality Categorical Data

There are numerous ways to deal with categorical data with high cardinality. It is important when creating machine learning models to pay attention to the dimensionality of the data. It can get out of hand really quickly if you do nothing to limit your cardinality. One of the simplest and easiest ways to follow is to take the categories with the highest frequencies (and therefore makeup the bulk of the distribution for that particular category) are left as is until a certain threshold of the data has been accounted for. After the threshold is met, all other categories for that column (which can sometimes be thousands of categories) will get lumped into a new category labeled as 'other.'

This simple trick can easily reduce dimensionality of categorical data. While it is not the most sophisticated method, it should be good enough for the models I want to create for my project.

Below is a function I created in order to easily reduce cardinality in my categorical data by using the method described above. My default threshold value is 75%, meaning after 75% of the data is accounted for, all other categories of lower frequencies will be lumped together into the 'other' category.

This code was inspired by author Raj Sangani in his blog: [Dealing with features that have high cardinality](https://towardsdatascience.com/dealing-with-features-that-have-high-cardinality-1c9212d7ff1b) (<https://towardsdatascience.com/dealing-with-features-that-have-high-cardinality-1c9212d7ff1b>). Thank you for the well commented and easy to follow along code!

Here is the documentation for the import necessary for the function below from python collections library: Counter. [collections.Counter](https://docs.python.org/3/library/collections.html#collections.Counter) (<https://docs.python.org/3/library/collections.html#collections.Counter>). Here we use the method 'most_common()' of collections.Counter.

From the documentation:

most_common([n]): Return a list of the n most common elements and their counts from the most common to the least. If n is omitted or None, most_common() returns all elements in the counter. Elements with equal counts are ordered in the order first encountered.

This function is ok for initial data exploration. However, it will need to be tweaked later in order to be implemented appropriately into a data processing pipeline. Instead of returning an updated column, future implementation will return a dictionary that will act as a value mapper in order to reduce cardinality of training and testing sets. The value mapper dictionary will be created by running the function on the entire dataset before carrying out the actual data processing/cleaning.

In [59]:

```

1 def cardinality_threshold(column,threshold=0.75,return_categories_list=
2     #calculate the threshold value using
3     #the frequency of instances in column
4     threshold_value=int(threshold*len(column))
5     #initialize a new list for lower cardinality column
6     categories_list=[]
7     #initialize a variable to calculate sum of frequencies
8     s=0
9     #Create a dictionary (unique_category: frequency)
10    counts=Counter(column)
11
12    #Iterate through category names and corresponding frequencies aft
13    #by descending order of frequency
14    for i,j in counts.most_common():
15        #Add the frequency to the total sum
16        s+=dict(counts)[i]
17        #append the category name to the categories list
18        categories_list.append(i)
19        #Check if the global sum has reached the threshold value, if so
20        if s >= threshold_value:
21            break
22    #append the new 'Other' category to list
23    categories_list.append('Other')
24
25    #Take all instances not in categories below threshold
26    #that were kept and lump them into the
27    #new 'Other' category.
28    new_column = column.apply(lambda x: x if x in categories_list else
29
30        #Return the transformed column and
31        #unique categories if return_categories = True
32        if(return_categories_list):
33            return new_column,categories_list
34        #Return only the transformed column if return_categories=False
35        else:
36            return new_column

```

funder categorical column

This column has a lot of categories that seem to be abbreviations or misspellings of the same funders. Ideally, I would correct all of these inconsistencies in the data in order to have my model perform the best that it can perform. However, for the scope of this project, I am going to put this tedious task off for now.

```
In [60]: 1 df3['funder'].sort_values().unique()[:200]
```

```
Out[60]: array(['A/co Germany', 'Aar', 'Abas Ka', 'Abasia',
   'Abc-ihushi Development Cent', 'Abd', 'Abdala', 'Abddwe', 'Abdul',
   'Abood', 'Abs', 'Aco/germany', 'Acord', 'Acord Ngo', 'Acra', 'Ac
   t',
   'Act Mara', 'Action Aid', 'Action Contre La Faim', 'Action In A',
   'Adap', 'Adb', 'Adf', 'Adp', 'Adp Bungu', 'Adp Mombo', 'Adp/w',
   'Adra', 'Af', 'Afdp', 'Afric', 'Africa',
   'Africa 2000 Network/undp', 'Africa Amini Alama',
   'Africa Project Ev Germany', 'African', 'African 2000 Network',
   'African Barrick Gold', 'African Development Bank',
   'African Development Foundation', 'African Muslim Agency',
   'African Realief Committe Of Ku', 'African Reflections Foundatio
   n',
   'African Relie', 'Africaone Ltd', 'Africare', 'Afriican Reli',
   'Afroz Ismail', 'Afya Department Lindi Rural', 'Agape Churc',
   'Agt Church', 'Ahmadia', 'Ai', 'Aic', 'Aic Church', 'Aic Kij',
   'Aict', 'Aimgold', 'Aixos', 'Alia', 'Ambwene Mwaikek', 'Amref',
   'Amrefe', 'Anglican Church', 'Angrikana', 'Anjuman E Seifee',
   'Answeer Muslim Grou', 'Apm', 'Apm[africa Precious Metals Lt',
   'Aqua Blues Angels', 'Arab Community', 'Arabi', 'Arabs Community',
   'Ardhi Instute', 'Area', 'Artisan', 'Asb', 'Asdp',
   'Asgerali N Bharwan', 'Auwasa', 'Awf', 'B.A.P', 'Ba As', 'Babtes
   t',
   'Babtist', 'Bahewasa', 'Bahresa', 'Bakari Chimkube', 'Bakwata',
   'Ballo', 'Balo', 'Balyehe', 'Banca Reale', 'Bank', 'Bao',
   'Baptist Church', 'Baric', 'Bathlomew Vicent', 'Batist Church',
   'Belgian Government', 'Belgij', 'Bened', 'Benguka', 'Bffs', 'Bfw
   d',
   'Bgm', 'Bgss', 'Bgssws', 'Bhws', 'Bilila', 'Bingo Foundation',
   'Bingo Foundation Germany', 'Bio Fuel Company', 'Biore', 'Birage',
   'Bkhws', 'Boazi', 'Boazi /o', 'Bobby', 'Bokera W', 'Boma Saving',
   'Bong-kug Ohh/choonla Lee', 'Bonite Bottles Ltd', 'Br', 'Bra',
   'Brad', 'Brdp', 'Bread For The Wor', 'Bread Of The Worl',
   'Bridge North', 'British Colonial Government', 'British Tanza',
   'Brown', 'Bruder', 'Bs', 'Bsf', 'Bukumbi', 'Bukwang Church Saint',
   'Bukwang Church Saints', 'Buluga Subvillage Community',
   'Bulyahunlu Gold Mine', 'Bumabu', 'Buptist', 'Busoga Trust', 'C',
   'Cafod', 'Caltas', 'Caltas Tanzania', 'Caltaz Kahama', 'Caltus',
   'Calvary Connect', 'Camartec', 'Camavita', 'Canada', 'Canada Aid',
   'Care Int', 'Care International', 'Care/cipro', 'Care/dwe',
   'Caritas', 'Carmatech', 'Cartas Tanzania', 'Cast', 'Cathoric',
   'Cbhi', 'Cc Motor Day 2010', 'Ccp', 'Ccpk', 'Ccps', 'Cct', 'Cdrg',
   'Cdft', 'Cdg', 'Cdtf', 'Cdtfdistrict Council', 'Cefa',
   'Cefa-njombe', 'Cefa/rcchurch', 'Ces (gmbh)', 'Ces(gmbh)', 'Cg',
   'Cg/rc', 'Cgc', 'Cgi', 'Ch', 'Chacha', 'Chacha Issame',
   'Chai Wazir', 'Chama Cha Ushirika', 'Chamavita', 'Chani',
   'Charlotte Well', 'Cheni', 'China Government', 'Chmavita',
   'Chongolo', 'Christan Outrich', 'Christian Outrich', 'Chuo',
   'Churc'], dtype=object)
```

```
In [61]: 1 transformed_funder = cardinality_threshold(df3['funder'],
2                                         threshold=0.65,
3                                         return_categories_list=False)
4
5 transformed_funder.value_counts()
```

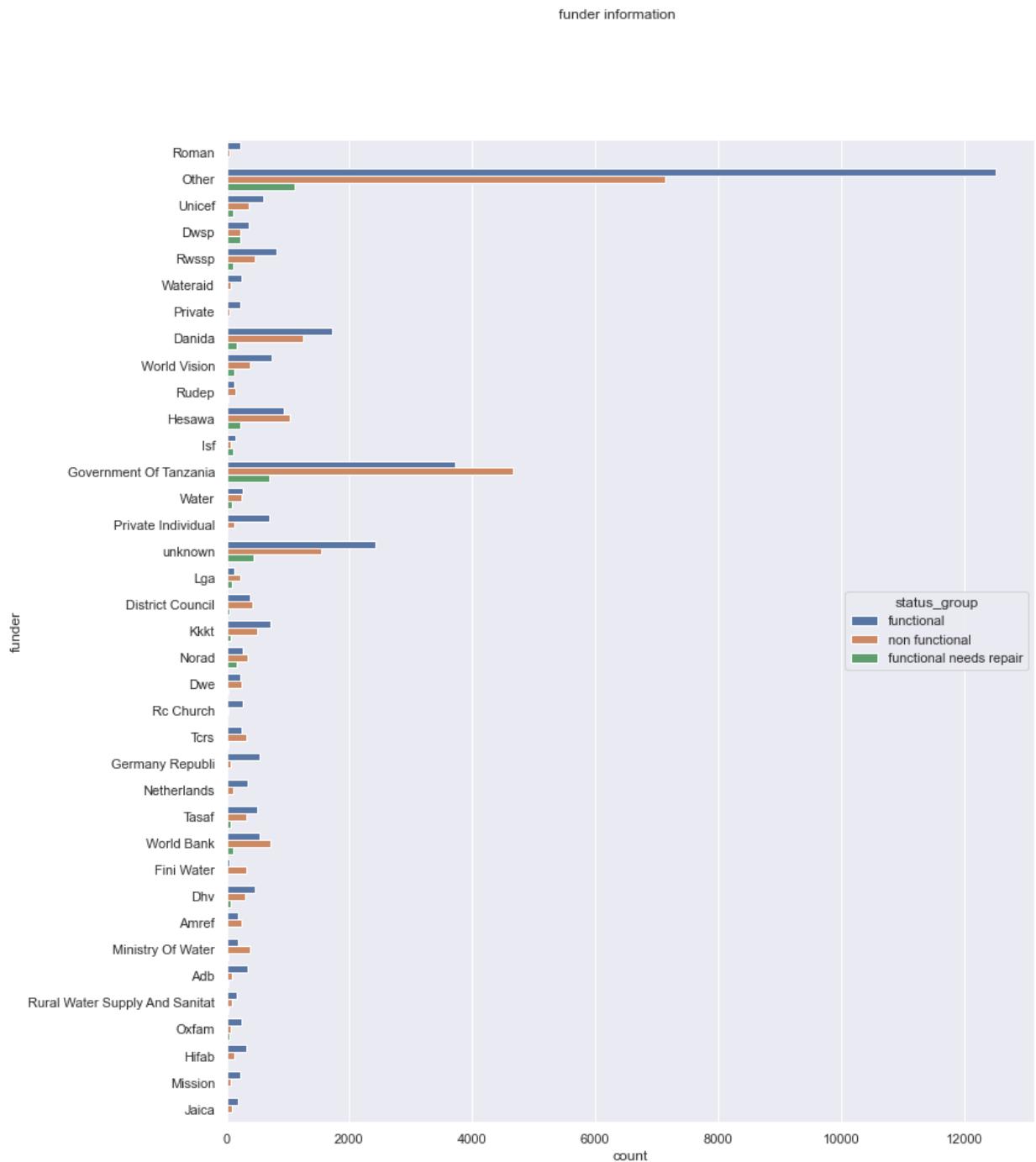
```
Out[61]: Other                20765
Government Of Tanzania        9084
unknown                         4412
Danida                          3114
Hesawa                          2202
Rwssp                           1374
World Bank                      1349
Kkkt                            1287
World Vision                    1246
Unicef                          1057
Tasaf                           877
District Council                 843
Dhv                            829
Private Individual               826
Dwsp                           811
Norad                           765
Germany Republi                 610
Tcrs                           602
Ministry Of Water                590
Water                           583
Dwe                            484
Netherlands                     470
Hifab                           450
Adb                            448
Lga                            442
Amref                           425
Fini Water                      393
Oxfam                           359
Wateraid                        333
Rc Church                        321
Isf                            316
Rudep                           312
Mission                          301
Private                         295
Jaica                           280
Roman                           275
Rural Water Supply And Sanitat   270
Name: funder, dtype: int64
```

In [62]:

```

1 df_transformed = df3.copy()
2 df_transformed['funder'] = transformed_funder
3
4 fig = plt.figure(figsize=(12,15))
5 fig.suptitle('funder information')
6 sns.countplot(y= 'funder', hue='status_group', data=df_transformed)
7 plt.show();

```



Transforming installer column

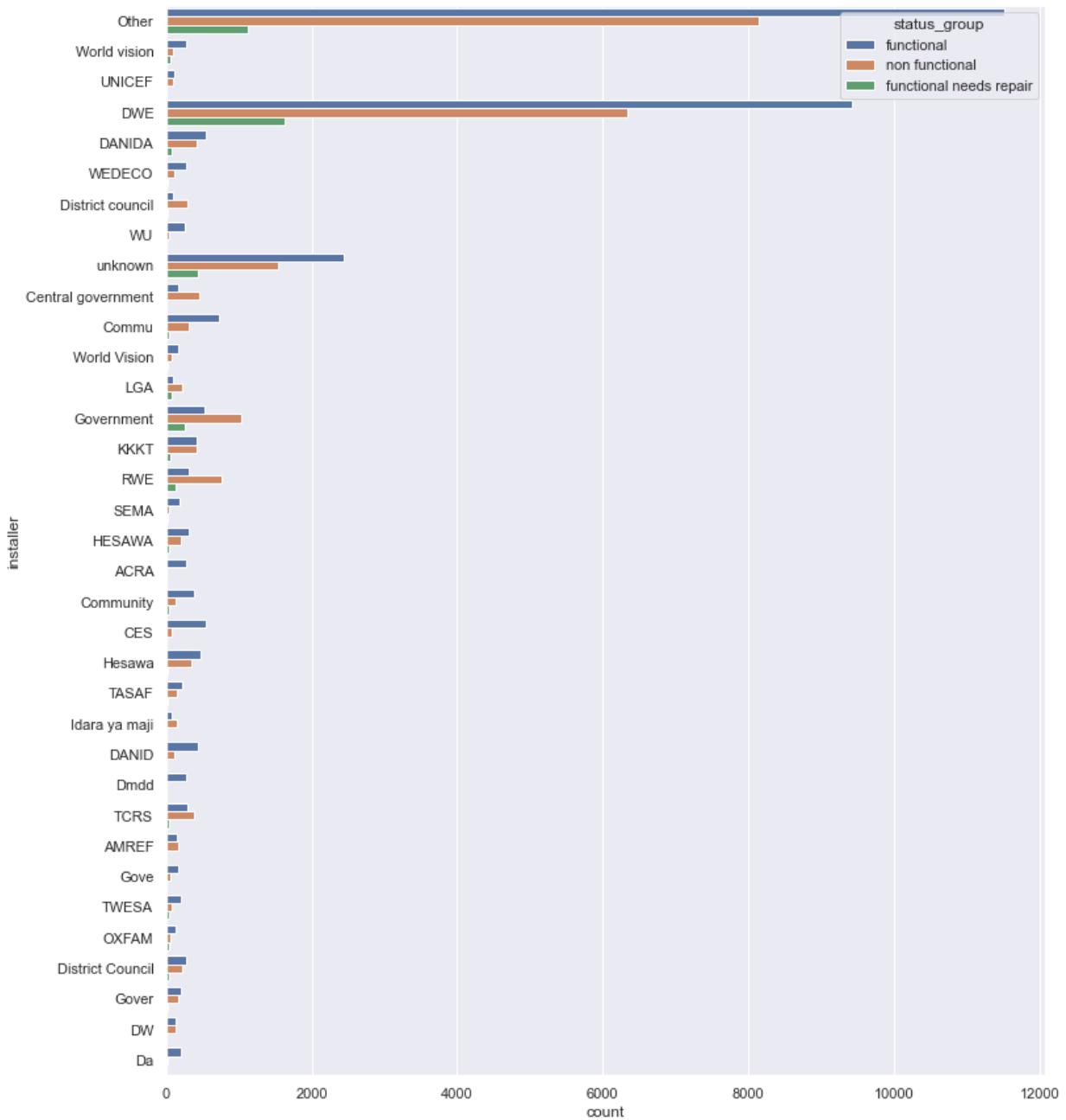
```
In [63]: 1 transformed_installer = cardinality_threshold(df3['installer'],
2                                         threshold=0.65,
3                                         return_categories_list=False)
4
5 transformed_installer.value_counts()
```

```
Out[63]: Other                20768
DWE                  17402
unknown              4433
Government           1825
RWE                  1206
Commu                1060
DANIDA               1050
KKKT                 898
Hesawa               840
TCRS                 707
Central government   622
CES                  610
Community            553
DANID                552
District Council     551
HESAWA               539
LGA                  408
World vision          408
WEDECO               397
TASAF                396
District council     392
Gover                383
AMREF                329
TWESA                316
WU                   301
Dmdd                 287
ACRA                 278
World Vision          270
SEMA                 249
DW                   246
OXFAM                234
Da                   224
UNICEF               222
Gove                 222
Idara ya maji        222
Name: installer, dtype: int64
```

In [64]:

```
1 df_transformed['installer'] = transformed_installer
2
3 fig = plt.figure(figsize=(12,15))
4 fig.suptitle('installer information')
5 sns.countplot(y= 'installer', hue='status_group', data=df_transformed)
6 plt.show();
```

installer information



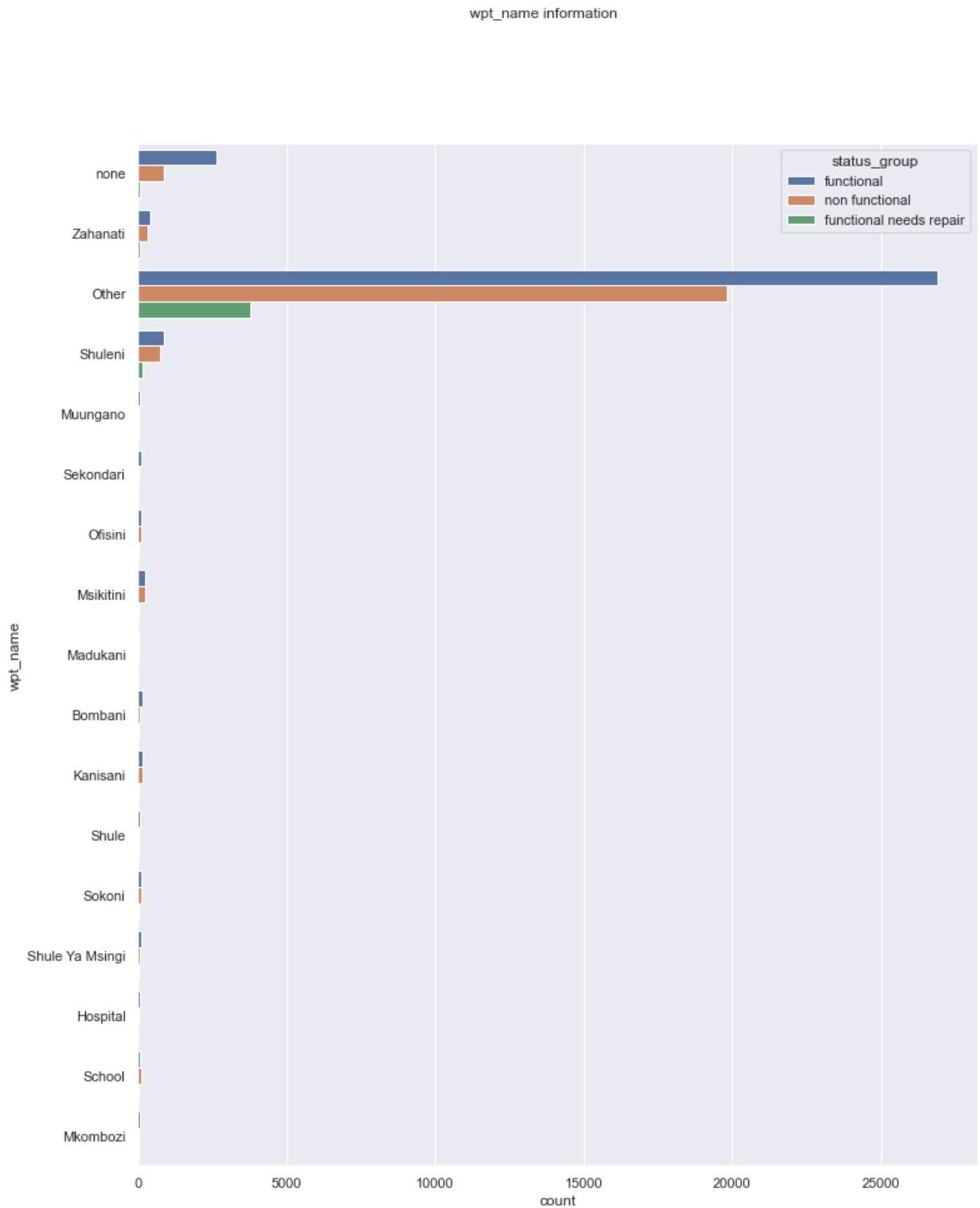
wpt_name transformation

```
In [65]: 1 transformed_wpt_name = cardinality_threshold(df3['wpt_name'],
2                                         threshold=0.15,
3                                         return_categories_list=False)
4
5 transformed_wpt_name.value_counts()
```

```
Out[65]: Other          50469
none            3563
Shulenzi        1748
Zahanati        830
Msikitini       535
Kanisani         323
Bombani          271
Sokoni           260
Ofisini          254
School           208
Shule Ya Msingi 199
Shule            152
Sekondari        146
Muungano         133
Mkombozi         111
Madukani          104
Hospital          94
Name: wpt_name, dtype: int64
```

In [66]:

```
1 df_transformed['wpt_name'] = transformed_wpt_name
2
3 fig = plt.figure(figsize=(12,15))
4 fig.suptitle('wpt_name information')
5 sns.countplot(y= 'wpt_name', hue='status_group', data=df_transformed)
6 plt.show();
```



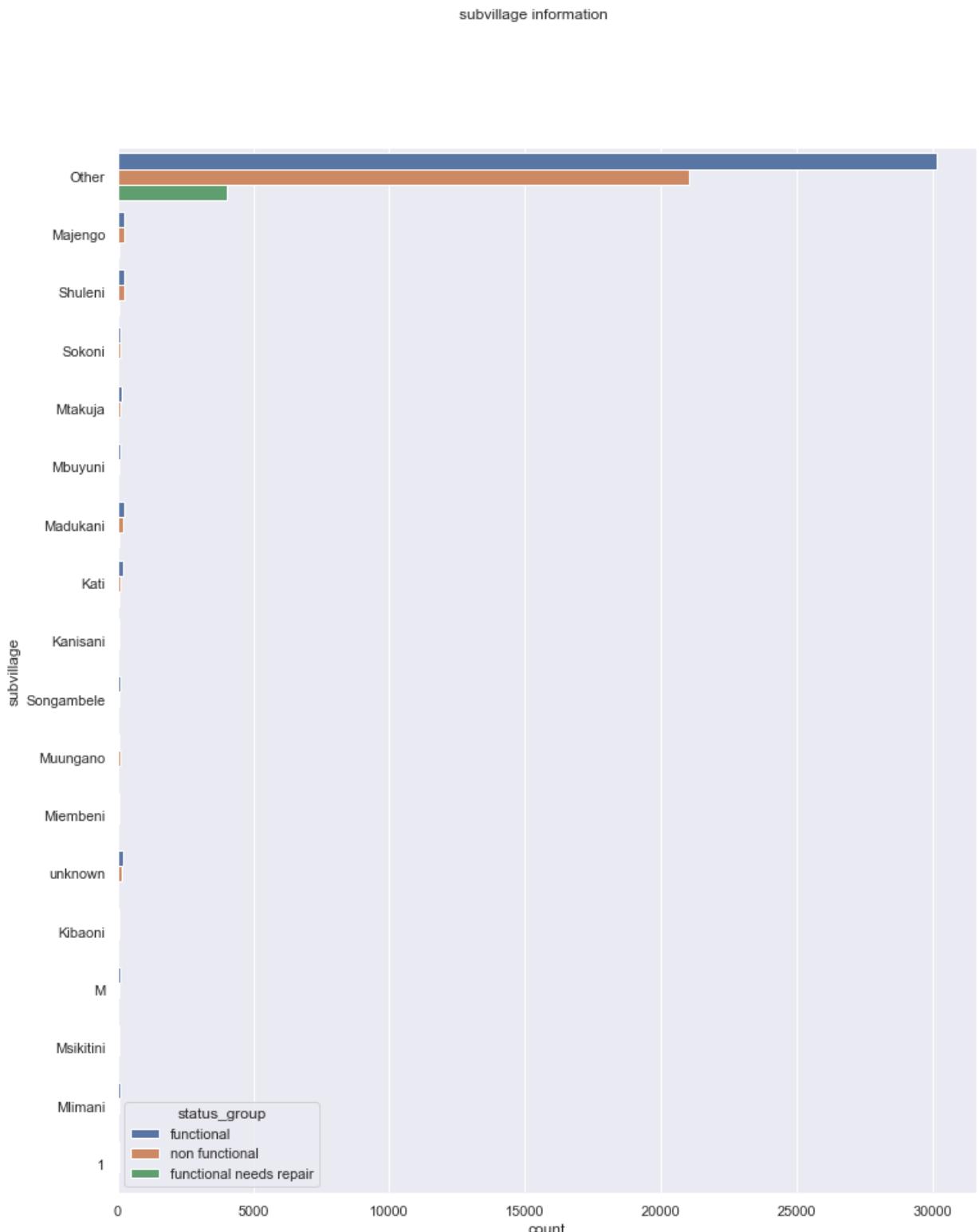
subvillage transformation

```
In [67]: 1 transformed_subvillage = cardinality_threshold(df3['subvillage'],
2                                         threshold=0.07,
3                                         return_categories_list=False)
4
5 transformed_subvillage.value_counts()
```

```
Out[67]: Other      55199
Madukani      508
Shuleni       506
Majengo       502
Kati          373
unknown        371
Mtakuja       262
Sokoni         232
M              187
Muungano      172
Mbuyuni        164
Mlimani        152
Songambele    147
Msikitini     134
Miembeni       134
1              132
Kibaoni        114
Kanisani       111
Name: subvillage, dtype: int64
```

In [68]:

```
1 df_transformed['subvillage'] = transformed_subvillage
2
3 fig = plt.figure(figsize=(12,15))
4 fig.suptitle('subvillage information')
5 sns.countplot(y= 'subvillage', hue='status_group', data=df_transformed)
6 plt.show();
```



Iga transformation

```
In [69]: 1 transformed_lga = cardinality_threshold(df3['lga'],
2                                         threshold=0.6,
3                                         return_categories_list=False)
4
5 transformed_lga.value_counts()
```

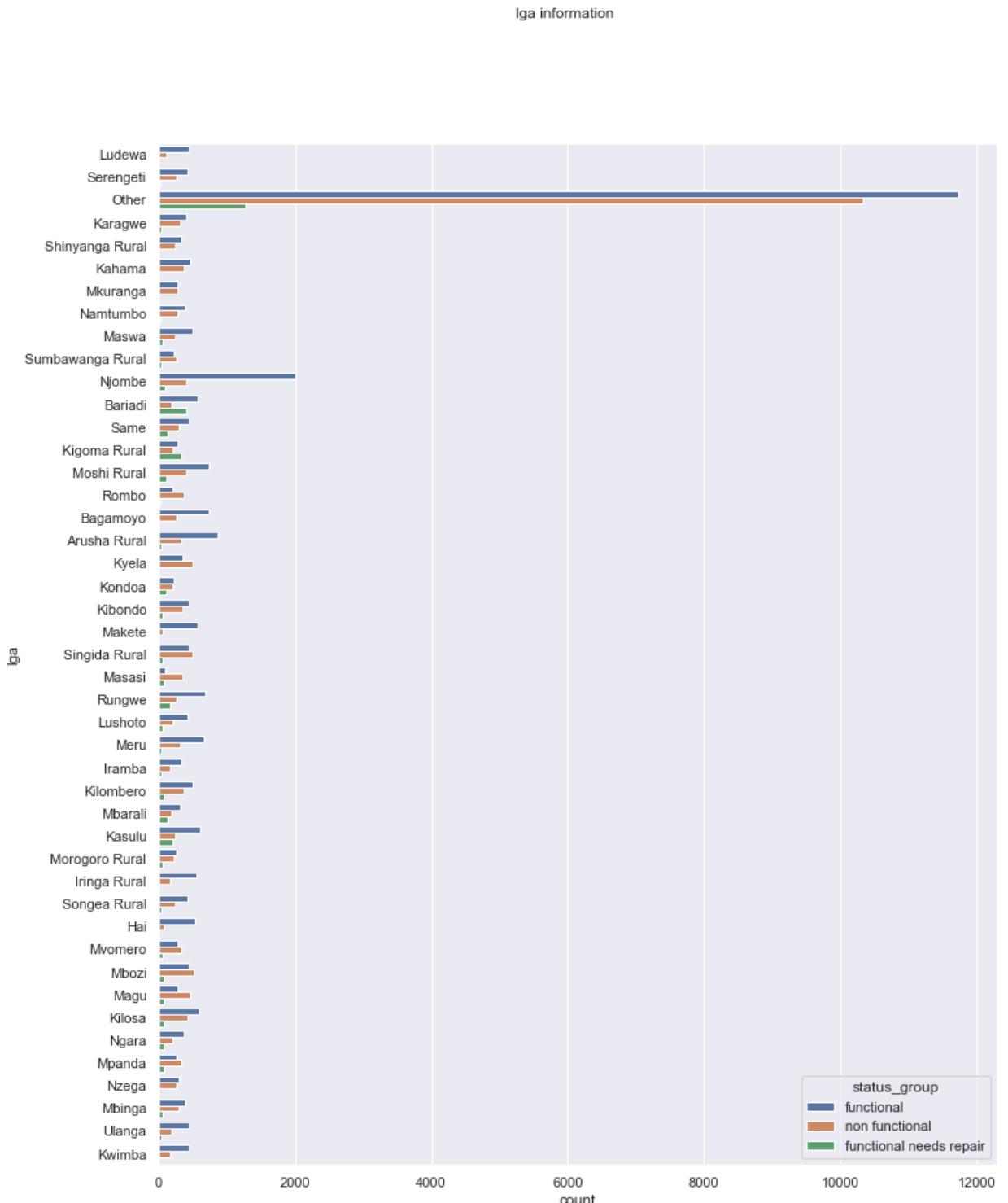
```
Out[69]: Other          23317
Njombe          2503
Arusha Rural    1252
Moshi Rural     1251
Bariadi          1177
Rungwe           1106
Kilosa            1094
Kasulu            1047
Mbozi             1034
Meru              1009
Bagamoyo          997
Singida Rural    995
Kilombero         959
Same              877
Kibondo           874
Kyela              859
Kahama             836
Magu               824
Kigoma Rural      824
Maswa              809
Karagwe            771
Mbinga             750
Iringa Rural       728
Serengeti          716
Namtumbo           694
Lushoto             694
Songea Rural       693
Mpanda              679
Mvomero             671
Ngara              669
Ulanga              665
Makete              630
Kwimba              627
Mbarali             626
Hai                 625
Rombo                594
Shinyanga Rural    588
Nzega                575
Ludewa                564
Mkuranga             560
Iramba                544
Masasi                528
Kondoa                523
Sumbawanga Rural    521
Morogoro Rural       521
Name: lga, dtype: int64
```

In [70]:

```

1 df_transformed['lga'] = transformed_lga
2
3 fig = plt.figure(figsize=(12,15))
4 fig.suptitle('lga information')
5 sns.countplot(y= 'lga', hue='status_group', data=df_transformed)
6 plt.show();

```



ward transformation

```
In [71]: 1 transformed_ward = cardinality_threshold(df3['ward'],
2                                         threshold=0.05,
3                                         return_categories_list=False)
4
5 transformed_ward.value_counts()
```

```
Out[71]: Other      56323
Igosi        307
Imalinyi     252
Siha Kati    232
Mdandu       231
Nduruma      217
Mishamo      203
Kitunda      203
Msindo       201
Chalinze     196
Maji ya Chai 190
Usuka        187
Ngarenanyuki 172
Chanika       171
Vikindu      162
Mtwango      153
Name: ward, dtype: int64
```

```
In [72]: 1 df_transformed['ward'] = transformed_ward
2
3 fig = plt.figure(figsize=(12,15))
4 fig.suptitle('ward information')
5 sns.countplot(y='ward', hue='status_group', data=df_transformed)
6 plt.show();
```

```
In [73]: 1 df3.columns
```

```
Out[73]: Index(['amount_tsh', 'funder', 'gps_height', 'installer', 'wpt_name', 'basin',
       'subvillage', 'region', 'region_code', 'district_code', 'lga', 'ward',
       'public_meeting', 'permit', 'construction_year',
       'extraction_type_class', 'management', 'payment', 'quality_group',
       'quantity', 'source_type', 'waterpoint_type', 'status_group',
       'near_river', 'ward_pop'],
      dtype='object')
```

zero skewed data

population

The column describing population is heavily skewed towards a value of zero, and it would seem very unlikely for so many wells to have a population of zero. Nearly 50% of the data is skewed towards a value of either zero or one. It is hard to trust the validity of this column. I decided to drop this column and bring in population information by ward collected by government census in 2012. This data has its own issues but I believe it to be a more complete dataset and give a better idea about the number of people who could benefit from the presence of a functional well in their area. Ward is a pretty high level division of geological boundaries for Tanzania. May have to revisit this population data at a later date. See beginning of notebook to see how population data was added by ward.

construction_year

This column is heavily skewed towards a value of zero. Zero seems to be a placeholder for unknown construction year. This column will be turned into a categorical column with binned decades and a column to replace zeroes with 'unknown'.

```
In [74]: 1 #with zeroes
2 df['construction_year'].describe()
```

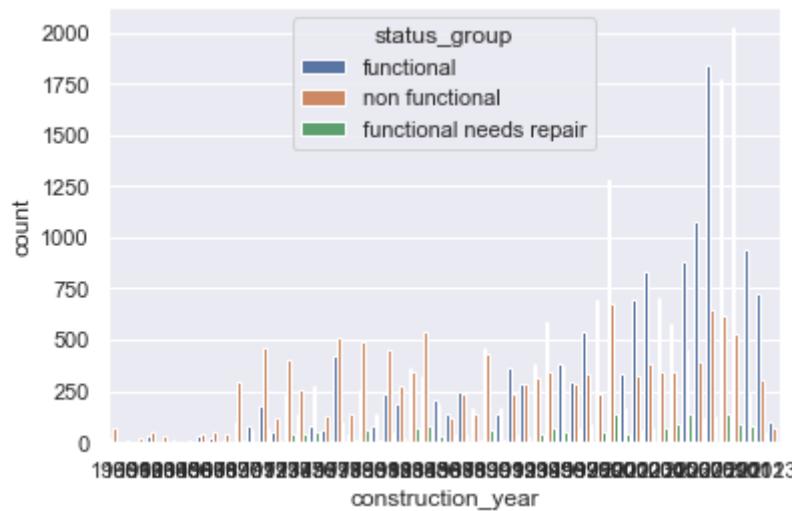
```
Out[74]: count    59400.000000
mean     1300.652475
std      951.620547
min      0.000000
25%     0.000000
50%     1986.000000
75%     2004.000000
max     2013.000000
Name: construction_year, dtype: float64
```

```
In [75]: 1 #without zeroes (placeholder values)
2 construction_zero_index = df3[df3['construction_year'] == 0].index
3 df_construction = df3.drop(construction_zero_index)
4 df_construction['construction_year'].describe()
```

```
Out[75]: count    38691.000000
mean     1996.814686
std      12.472045
min     1960.000000
25%    1987.000000
50%    2000.000000
75%    2008.000000
max     2013.000000
Name: construction_year, dtype: float64
```

```
In [76]: 1 sns.countplot(x='construction_year', hue='status_group', data = df_cons
```

```
Out[76]: <AxesSubplot:xlabel='construction_year', ylabel='count'>
```



In [77]:

```
1 #bin construction year into decades and 'unknown' for zeroes
2
3
4 def construction_wrangler(row):
5     if row['construction_year'] >= 1960 and row['construction_year'] <
6         return '60s'
7     elif row['construction_year'] >= 1970 and row['construction_year'] <
8         return '70s'
9     elif row['construction_year'] >= 1980 and row['construction_year'] <
10        return '80s'
11    elif row['construction_year'] >= 1990 and row['construction_year'] <
12        return '90s'
13    elif row['construction_year'] >= 2000 and row['construction_year'] <
14        return '00s'
15    elif row['construction_year'] >= 2010:
16        return '10s'
17    else:
18        return 'unknown'
19
20 df_transformed['construction_year'] = df_transformed.apply(lambda row:
```

In [78]:

```
1 df_transformed['construction_year'].value_counts()
```

Out[78]:

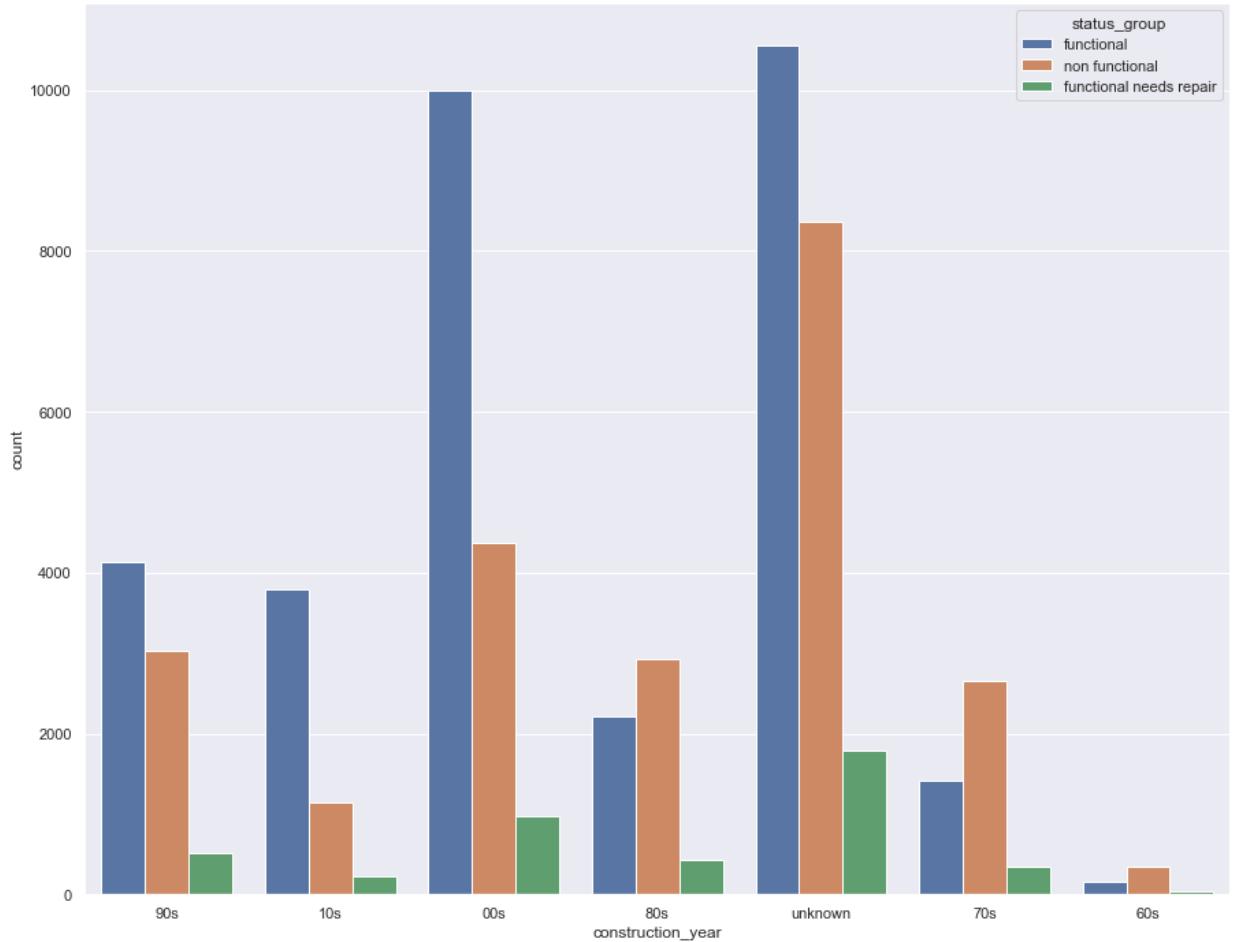
unknown	20709
00s	15330
90s	7678
80s	5578
10s	5161
70s	4406
60s	538

Name: construction_year, dtype: int64

In [79]:

```
1 fig = plt.figure(figsize=(15,12))
2 fig.suptitle('construction_year binned')
3 sns.countplot(x = 'construction_year', hue='status_group', data=df_tran
4 plt.show();
```

construction_year binned



amount_tsh

```
In [80]: 1 df_transformed['amount_tsh'].sort_values(ascending=False)[:50]
```

```
Out[80]: 10812    350000.0
22191    250000.0
22817    200000.0
45067    170000.0
9961     138000.0
9917     120000.0
45470    117000.0
57811    117000.0
54595    117000.0
5558     117000.0
37459    117000.0
44381    117000.0
10615    117000.0
3228     100000.0
38137    100000.0
6591     100000.0
21567    70000.0
53968    60000.0
50494    50000.0
12895    50000.0
51260    50000.0
47201    50000.0
26130    45000.0
39114    45000.0
11349    45000.0
59375    40000.0
44803    40000.0
7364     40000.0
3245     40000.0
41790    40000.0
31976    40000.0
543      40000.0
49244    38000.0
23252    30000.0
45421    30000.0
31013    30000.0
34335    30000.0
17794    30000.0
48611    30000.0
57770    30000.0
20500    30000.0
23087    26000.0
27536    26000.0
45540    25000.0
55981    25000.0
58566    25000.0
30955    25000.0
22487    25000.0
194      25000.0
43712    25000.0
Name: amount_tsh, dtype: float64
```

Amount_tsh

This column describes the total static head of each pump. It could be very useful in determining the amount of water available at each well. However, the data is extremely skewed towards the value of zero. There also seems to be extreme outliers in the other direction as well, with a value of 350000. Also, there are no dimensions in the data to know what the scale is. It could be feet or meters or something else.

Dealing with the large skew (over 70%) towards the value of zero is out of the scope for my project. I will drop this column for now and move on.

```
In [81]: 1 df_transformed.drop(columns=['amount_tsh'], axis=1, inplace=True)
```

GPS height

This column also has a skew towards zero. Nearly 35% of the values are zero. I am going to drop this column for now.

```
In [82]: 1 df_transformed.drop(columns=['gps_height'], axis=1, inplace=True)
```

```
In [83]: 1 df_transformed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59400 entries, 0 to 59399
Data columns (total 23 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   funder            59400 non-null   object  
 1   installer         59400 non-null   object  
 2   wpt_name          59400 non-null   object  
 3   basin             59400 non-null   object  
 4   subvillage        59400 non-null   object  
 5   region            59400 non-null   object  
 6   region_code       59400 non-null   int64  
 7   district_code     59400 non-null   int64  
 8   lga               59400 non-null   object  
 9   ward               59400 non-null   object  
 10  public_meeting    59400 non-null   object  
 11  permit             59400 non-null   object  
 12  construction_year 59400 non-null   object  
 13  extraction_type_class 59400 non-null   object  
 14  management         59400 non-null   object  
 15  payment            59400 non-null   object  
 16  quality_group      59400 non-null   object  
 17  quantity           59400 non-null   object  
 18  source_type        59400 non-null   object  
 19  waterpoint_type    59400 non-null   object  
 20  status_group       59400 non-null   object  
 21  near_river         59400 non-null   float64 
 22  ward_pop           59400 non-null   float64 

dtypes: float64(2), int64(2), object(19)
memory usage: 13.4+ MB
```

```
In [84]: 1 df_transformed.columns
```

```
Out[84]: Index(['funder', 'installer', 'wpt_name', 'basin', 'subvillage', 'region',
       'region_code', 'district_code', 'lga', 'ward', 'public_meeting',
       'permit', 'construction_year', 'extraction_type_class', 'management',
       'payment', 'quality_group', 'quantity', 'source_type',
       'waterpoint_type', 'status_group', 'near_river', 'ward_pop'],
      dtype='object')
```

```
In [85]: 1 df_transformed.drop(columns=['region_code', 'district_code'], axis=1, i
```

```
In [86]: 1 df_transformed.columns
```

```
Out[86]: Index(['funder', 'installer', 'wpt_name', 'basin', 'subvillage', 'region',
       'lga', 'ward', 'public_meeting', 'permit', 'construction_year',
       'extraction_type_class', 'management', 'payment', 'quality_group',
       'quantity', 'source_type', 'waterpoint_type', 'status_group',
       'near_river', 'ward_pop'],
      dtype='object')
```

Below is some additional geographical/census/population data that could prove useful in improving modeling in future work. Due to time constraints, the data was not thoroughly explored.

I kept it below for now as an example of ways to bring in more information from government resources in order to explore new ways to improve predictive capabilities of any future modeling.

```
In [87]: 1 # df1 = pd.read_csv('data/region_points.csv')
2
3 # df2 = pd.read_csv('data/ObservationData_eqpqcq.csv')
4 # region_list = list(train_features['region'].unique())
5 # df_2 = df2[df2['region'].isin(region_list)]
6
7 # df3 = pd.read_csv('data/ObservationData_mzztfvd.csv')
```

```
In [88]: 1 # len(df3[df3['location'].isin(region_list)]['location'].unique())
2 # df2[df2['indicator'] == 'Revenue Collection (Tshs)'].groupby('region')
3 # df_2.groupby(['region', 'indicator']).mean()
4 # df3[df3['variable'] == 'Mean Maximum Temperature (°C)'].groupby('location')
5 # dfT = pd.read_csv('data/ObservationData_eqpqcq.csv')
6 # dfT2 = pd.read_csv('data/ObservationData_mzztfvd.csv')
7
8 # dfT2.head()
```

End of Exploratory Notebook

This is the end of data exploration and of this jupyter notebook. Please feel free to reach me at the following email for any questions or comments. Please continue on to my modeling notebook which contains information about what classification metrics are most important to consider given my business problem, all data preprocessing steps, and all modeling steps. Thank you!

Author Name: Dylan Dey

Email: ddey2985@gmail.com (<mailto:ddey2985@gmail.com>)

Github: [Dddehy Github](https://github.com/ddey117/Tanzanian_Water_Pump_Classification) (https://github.com/ddey117/Tanzanian_Water_Pump_Classification)