CHARLES ODHIAMBO

Predicting Waterpoint Functionality in Tanzania

Business Understanding

Overview

After gaining independence, the Tanzanian government introduced a policy to provide free potable water to all rural residents by 1991. Formalized in 1971, this policy made the government responsible for developing, operating, and maintaining water supply systems without implementing cost recovery measures. During the 1970s, many projects were funded by donors, particularly from Sweden, leading to the construction of numerous waterpoints. By the time the dataset was collected, some of these waterpoints remained fully operational, others required repairs, and some had ceased functioning altogether.

Objectives

The main goal of this project is to develop a model capable of predicting whether a waterpoint is functional or non-functional based on a set of independent variables. This insight can assist the Tanzanian Government and other stakeholders in identifying waterpoints that may require repairs based on their specific characteristics.

Stakeholders

- Government Entities: The Tanzanian Ministry of Water, responsible for infrastructure management and policymaking.
- Non-Governmental Organizations (NGOs): Organizations working to improve access to safe water in underserved areas.
- Local Communities: Direct users who benefit from operational waterpoints.
- Funders and Donors: Investors focused on the impact and sustainability of water infrastructure projects.

Success criteria

- **Accuracy:** The model should have a high accuracy in predicting the status of waterpoints.

Constraints

- **Data Quality:** The accuracy of the model depends on the quality and completeness of thedata.
- **Resource Limitations:** Limited resources for maintenance and repairs may affec tth eimplementation of the model's recommendations.
- **Multinormial Classification:** The dataset is a multinormial classification problem but I will treat it as a binary classification problem

DATA SOURCE

DrivenData. (2015). Pump it Up: Data Mining the Water Table. Retrieved [December 3 2024] from https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table.

Data Understanding

In [ ]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import math
import numpy as np
import re
from scipy.stats import chi2_contingency
from sklearn.model_selection import train_test_split,GridSearchCV,cross_val_score
from sklearn.preprocessing import RobustScaler,StandardScaler,MinMaxScaler,OneHotEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score,classification_report
, confusion_matrix, ConfusionMatrixDisplay,roc_auc_score, roc_curve, auc, f1_score
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.compose import ColumnTransformer
import category_encoders as ce
from category_encoders import TargetEncoder
%matplotlib inline
```

In [14]:

```
pd.options.display.max_columns = None
```

In [15]:

```
#Test Data
test_set_values_df = pd.read_csv("data/Test_set_values.csv")
test_set_values_df
```

Out[15]:

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wpt_name	num_pri
0	50785	0.0	2013-02-04	Dmdd	1996	DMDD	35.290799	-4.059696	Dinamu Secondary School	
1	51630	0.0	2013-02-04	Government Of Tanzania	1569	DWE	36.656709	-3.309214	Kimnyak	
2	17168	0.0	2013-02-01	NaN	1567	NaN	34.767863	-5.004344	Puma Secondary	
3	45559	0.0	2013-01-22	Finn Water	267	FINN WATER	38.058046	-9.418672	Kwa Mzee Pange	
4	49871	500.0	2013-03-27	Bruder	1260	BRUDER	35.006123	-10.950412	Kwa Mzee Turuka	
...	...	...	...	...	...	...	...	...	...	
14845	39307	0.0	2011-02-24	Danida	34	Da	38.852669	-6.582841	Kwambwezi	
14846	18990	1000.0	2011-03-21	Hiap	0	HIAP	37.451633	-5.350428	Bonde La Mkondoa	
14847	28749	0.0	2013-03-04	NaN	1476	NaN	34.739804	-4.585587	Bwawani	
14848	33492	0.0	2013-02-18	Germany	998	DWE	35.432732	-10.584159	Kwa John	
				Government				-	Kwa Mzee	

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wpt_name	num_pri
14849	68707	0.0	2013-02-13	Government Of Tanzania	481	Government	34.765054	-11.226012	Kwa Mzee Chagala	

14850 rows × 40 columns

In [16]:

```
# Data to be used
Training_set_values_df = pd.read_csv("data/Training_set_values.csv")
Training_set_values_df
```

Out[16]:

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wpt_name	num_private
0	69572	6000.0	2011-03-14	Roman	1390	Roman	34.938093	-9.856322	none	0
1	8776	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.698766	-2.147466	Zahanati	0
2	34310	25.0	2013-02-25	Lottery Club	686	World vision	37.460664	-3.821329	Kwa Mahundi	0
3	67743	0.0	2013-01-28	Unicef	263	UNICEF	38.486161	-11.155298	Zahanati Ya Nanyumbu	0
4	19728	0.0	2011-07-13	Action In A	0	Artisan	31.130847	-1.825359	Shuleni	0
...	...	...	...	...	...	...	...	...	...	...
59395	60739	10.0	2013-05-03	Germany Republi	1210	CES	37.169807	-3.253847	Area Three Namba 27	0
59396	27263	4700.0	2011-05-07	Cefa-njombe	1212	Cefa	35.249991	-9.070629	Kwa Yahona Kuvala	0
59397	37057	0.0	2011-04-11	NaN	0	NaN	34.017087	-8.750434	Mashine	0
59398	31282	0.0	2011-03-08	Malec	0	Musa	35.861315	-6.378573	Mshoro	0
59399	26348	0.0	2011-03-23	World Bank	191	World	38.104048	-6.747464	Kwa Mzee Lugawa	0

59400 rows × 40 columns

In [17]:

```
# Predictor y label
Training_set_labels_df = pd.read_csv("data/Training_set_labels.csv")
Training_set_labels_df
```

Out[17]:

	id	status_group
0	69572	functional
1	8776	functional
2	34310	functional

	id	status_group
2	34310	functional
3	67743	non functional
4	19728	functional
...	...	...
59395	60739	functional
59396	27263	functional
59397	37057	functional
59398	31282	functional
59399	26348	functional

59400 rows × 2 columns

In [18]:

```
# Merge Dataset
# Merging the train x data to its y predictor
df_merge = pd.merge(Training_set_labels_df,Training_set_values_df,how= "inner", on = "id
")
```

In [19]:

```
df_merge.sample(20)
```

Out[19]:

	id	status_group	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wp
29614	23639	non functional	0.0	2012-11-13	Tlc	0	TLC	32.999352	-5.074673	Ran
21848	71945	functional	0.0	2011-03-23	Government Of Tanzania	0	DWE	36.825923	-6.311998	Kw
36417	67889	functional	20.0	2011-04-19	Dh	0	DH	37.422486	-6.447170	K
27880	6447	functional	0.0	2011-04-07	Government Of Tanzania	1510	DWE	38.288192	-4.782214	K
15642	21853	functional	0.0	2013-01-31	Rwssp	0	DWE	32.569792	-3.447407	Naml
5894	8215	functional	0.0	2012-10-13	Wateraid	0	DWE	33.187346	-3.963325	
12441	413	functional needs repair	0.0	2011-03-24	Kkkt	1247	KKKT	38.455788	-4.910434	Kw Shemn
1474	71889	non functional	0.0	2013-01-29	Finw	208	FinW	39.772936	-10.742649	
14373	3046	functional	0.0	2011-03-15	Government Of Tanzania	278	DWE	36.112651	-8.889769	Kwa N
3802	20492	non functional	0.0	2012-10-14	Total Land Care	0	Total land care	32.241347	-5.086573	:
51636	22526	non functional	0.0	2013-01-19	Jika	842	JIKA	35.071992	-5.851101	Kiosk

	id	status_group	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wp
23060	[illegible]	functional	6.0	[illegible]	Unicef	[illegible]	DWE	[illegible]	[illegible]	Mr
19162	25412	functional needs repair	0.0	2013-01-16	Lips	133	District Council	39.437293	-10.035545	F
499	32255	non functional	30.0	2011-03-24	African Relie	19	Af	38.984115	-6.553264	M
42301	36705	functional	1000.0	2011-04-04	Rc Church	2119	RC CHURCH	34.413233	-9.200537	Kwa C
18471	10000	non functional	0.0	2013-03-20	Tcrs	1543	TCRS	37.963686	-4.433325	Kwa F
35350	46398	non functional	1000.0	2011-04-12	Go	0	DW	37.229539	-6.045969	C
21351	70903	functional	0.0	2011-07-08	Hesawa	0	DWE	31.375467	-1.041825	H
20570	7809	functional needs repair	0.0	2013-03-13	Government Of Tanzania	1481	Government	34.919663	-11.113842	Kw
47518	39854	functional	50.0	2011-03-23	Parastatal	-13	Da	38.979195	-6.519993	M

In [20]:

```
# Getting basic information on the merged dataset
df_merge.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 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: 19.0+ MB
```

In [21]:

```
# Checking the columns available in our dataset
df_merge.columns
```

Out[21]:

```
Index(['id', 'status_group', 'amount_tsh', 'date_recorded', 'funder',
       'gps_height', 'installer', 'longitude', 'latitude', 'wpt_name',
       'num_private', 'basin', 'subvillage', 'region', 'region_code',
       'district_code', 'lga', 'ward', 'population', '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_type', 'source_class',
       'waterpoint_type', 'waterpoint_type_group'],
      dtype='object')
```

In [22]:

```
# finding the shape of our dataset
df_merge.shape
```

Out[22]:

```
(59400, 41)
```

In [23]:

```
# finding unique values in each column

object_columns = df_merge.select_dtypes(include=['object','int64','float64']).columns

object_column_unique = {col : df_merge[col].nunique() for col in object_columns}

columns_unique_df =pd.DataFrame(list(object_column_unique.items()), columns= ['column_na
me', 'number_of_unique_values'])
columns_unique_df
```

Out[23]:

	column_name	number_of_unique_values
0	id	59400
1	status_group	3
2	amount_tsh	98
3	date_recorded	356
4	funder	1897
5	gps_height	2428
6	installer	2145
7	longitude	57516
8	latitude	57517

	column_name	number_of_unique_values
9	wpt_name	37400
10	num_private	65
11	basin	9
12	subvillage	19287
13	region	21
14	region_code	27
15	district_code	20
16	lga	125
17	ward	2092
18	population	1049
19	public_meeting	2
20	recorded_by	1
21	scheme_management	12
22	scheme_name	2696
23	permit	2
24	construction_year	55
25	extraction_type	18
26	extraction_type_group	13
27	extraction_type_class	7
28	management	12
29	management_group	5
30	payment	7
31	payment_type	7
32	water_quality	8
33	quality_group	6
34	quantity	5
35	quantity_group	5
36	source	10
37	source_type	7
38	source_class	3
39	waterpoint_type	7
40	waterpoint_type_group	6

Basic understanding of each of the columns based on their cardinality

High Cardinality columns

- id (59400) = Unique values are equal to the total number of rows of dataframe indicating its the unique identifier of the dataset
- longitude and latitude = indicates that the wells are spread on a huge geographical region
- wpt_name = indicates a large variety on water point names
- sub_villages = indicates wells are located on a high number of unique sub-villages

Medium Cardinality columns

- funder = indicates a huge number of funders i.e people who funded the project
- gps_height = indicates a large number of wells were built along multiple altitude heights
- ward -Indicates wells are found in multiple different wards
- installer - indicates that there are multiple organizations that have constructed wells in Tanzania
- scheme_name - Indicates there are a lot of different water schemes in the area

- scheme_name - indicates there are a lot of different water schemes in the area
- amount_tsh (98)- this represents the total static head this means the elevation between the free level of water till the discharge point of the pump

Low Cardinality Columns

- status group = indicates the three types of status a well might be i.e functional, non-functional, functional but in need of repair
- public meeting (2) = Indicates a boolean value where its either a yes or no
- management_group (5) = Indicates there are 5 unique ways a well may be managed
- permit (2) = shows a well can be built having a permit or no permit
- source_class = shows the classification of the source of the water

numerical columns that are distinct

- id
- longitude and latitude
- region code
- district code
- construction year

In [24]:

```
df_merge.describe()
```

Out[24]:

	id	amount_tsh	gps_height	longitude	latitude	num_private	region_code	district_code	
count	59400.000000	59400.000000	59400.000000	59400.000000	5.940000e+04	59400.000000	59400.000000	59400.000000	59
mean	37115.131768	317.650385	668.297239	34.077427	-5.706033e+00	0.474141	15.297003	5.629747	
std	21453.128371	2997.574558	693.116350	6.567432	2.946019e+00	12.236230	17.587406	9.633649	
min	0.000000	0.000000	-90.000000	0.000000	-1.164944e+01	0.000000	1.000000	0.000000	
25%	18519.750000	0.000000	0.000000	33.090347	-8.540621e+00	0.000000	5.000000	2.000000	
50%	37061.500000	0.000000	369.000000	34.908743	-5.021597e+00	0.000000	12.000000	3.000000	
75%	55656.500000	20.000000	1319.250000	37.178387	-3.326156e+00	0.000000	17.000000	5.000000	
max	74247.000000	350000.000000	2770.000000	40.345193	-2.000000e-08	1776.000000	99.000000	80.000000	30

- amount_total_static_head = the min amount is 0, the mean value is 317.65. There is also an indication that majority of the values are 0 because the 1st quartile and the median being 0
- gps_height = min value is -90 might indicate the wells are located below sea level
- population = min amount is 0. This might be an error or indicates that the area where the wells are have no poulation while the maximum is 30500

Data Cleaning

In [25]:

```
# checking the number of null values in my dataset

df_merge.isna().sum()
```

Out[25]:

```
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
```

There are missing value in the following columns

- **funder**
- **installer**
- **sub_village**
- **public_meeting**
- **scheme_management**
- **scheme_name**
- **permit**

All this are categorical columns and missing values might be caused by missed information in the data collection process

In [26]:

```
# checking the information contained on the columns with the missing values

missing_cols = df_merge.columns[df_merge.isnull().any()]

df_merge.loc[: ,missing_cols].sample(20)
```

Out[26]:

	funder	installer	subvillage	public_meeting	scheme_management	scheme_name	permit
[illegible]	Government Of	Government	Busamwa	True	VWC	Nyamtukuza	True

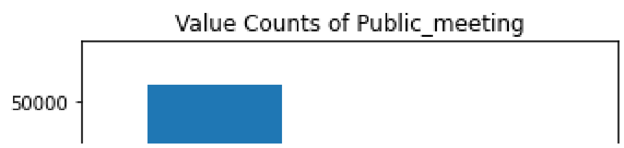
	funder	installer	subvillage	public_meeting	scheme_management	scheme_name	permit
30357	Tanzania	Government	Busemwa	True	VWC	Nyamtukuza	True
12590	Miziriol	Miziriol	Bare A	True	VWC	Endawasu	True
22060	Lwi	LWI	Mbiti A	True	WUG	NaN	False
23413	Unhcr	TWESA	Songambele	True	VWC	Kabingo/kiobela gravity water supply	False
26526	Unicef	DWE	Kati	True	WUA	wanging'ombe water supply s	True
10782	NaN	NaN	Itaba	True	VWC	NaN	False
32412	World Bank	District Council	Duma	True	VWC	Myombo Water Supply	True
38591	Netherlands	DWE	Mwang'Holo	NaN	WUG	NaN	False
52793	Water	Commu	Sokoine	True	VWC	NaN	False
30578	Norad	NORAD	Kamazi	True	Water authority	Nyafisi	True
9691	NaN	NaN	Darajani	True	Water authority	NaN	NaN
2770	NaN	NaN	Iwalanji	True	VWC	NaN	False
58263	Kanisa Katoliki Lolovoni	DWE	Olomatejo	True	Parastatal	Olikimo water project	True
41732	Go	DWE	Muungano	False	VWC	K	True
30243	Tcrs	TCRS	Mbuyuni	True	VWC	NaN	True
31609	NaN	NaN	Kijenge Juu	False	Private operator	A	NaN
10637	Danida	Central government	Mpunguti	True	VWC	ngamanga water supplied sch	True
27345	Germany Republi	CES	Mrimumu	True	Water Board	Losaa-Kia water supply	True
43597	Conce	DWE	L	True	WUA	Kit	True
37722	0	0	Mtaa Wa Kivule	NaN	Private operator	NaN	False

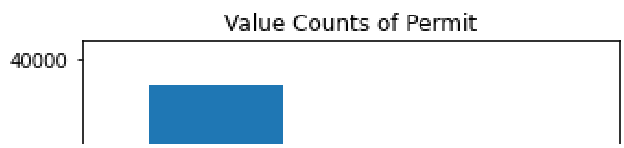
In [27]:

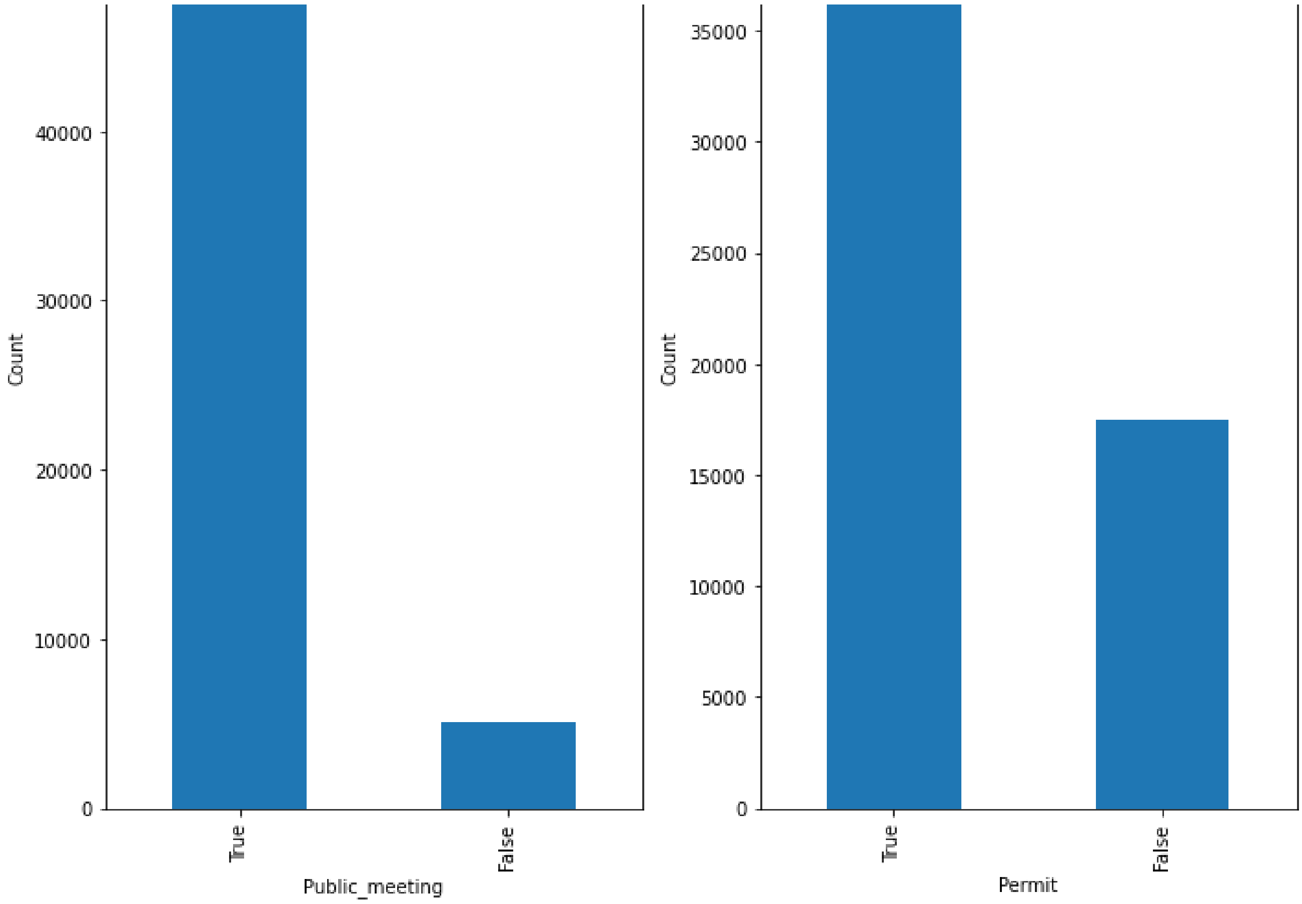
```
# for categorical columns that explain extra information on the well we can fill null values by the word unknown i.e funder, installer , subvillage, scheme_management, scheme_name

df_merge[['funder', 'installer', 'subvillage','scheme_management','scheme_name']] = df_merge[['funder', 'installer', 'subvillage','scheme_management','scheme_name']].fillna("unknown")
```

In [28]:

```
# for the remaining lets check the value counts as they are boolean values i.e between two choices

object_columns = df_merge.columns[df_merge.isna().any()]

fig,ax= plt.subplots(nrows= 1, ncols = 2 ,figsize=(10,8))

for i,col in enumerate(object_columns) :
    df_merge[col].value_counts().plot(kind= 'bar', ax= ax[i])
    ax[i].set_title(f'Value Counts of {col.capitalize()}')
    ax[i].set_xlabel(col.capitalize())
    ax[i].set_ylabel('Count')

plt.tight_layout()
plt.show()
```







- **We can see that most wells go through a public meeting before being built**
- **Most wells have permits before being built**

In [29]:

```
# filling missing values in both this fields by the modes of their columns

df_merge = df_merge.apply(lambda x: x.fillna(x.value_counts().index[0]))
```

In [30]:

```
df_merge.isna().sum()
```

Out[30]:

```
id                       0
status_group             0
amount_tsh               0
date_recorded            0
funder                   0
gps_height               0
installer                0
longitude                0
latitude                 0
wpt_name                 0
num_private              0
basin                    0
subvillage               0
region                   0
region_code              0
district_code            0
lga                      0
ward                     0
population               0
public_meeting           0
recorded_by              0
scheme_management        0
scheme_name              0
permit                   0
construction_year        0
extraction_type          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 [31]:

```python
# lets check for instances of duplication
# will check for duplicates in all columns except the first column i.e the primary key of this dataset

df_duplicated =df_merge[df_merge.iloc[:, 1:].duplicated(keep= False)]
```

In [32]:

```python
sorted_duplicates = df_duplicated.sort_values(by=df_merge.columns[1:].tolist())
sorted_duplicates.head(20)
```

Out[32]:

	id	status_group	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	w
17451	29553	functional	0.0	2011-07-13	He	0	HE	31.61953	-1.793342e+00	
39187	18713	functional	0.0	2011-07-13	He	0	HE	31.61953	-1.793342e+00	
326	7900	functional	0.0	2011-07-18	Government Of Tanzania	0	Government	0.00000	-2.000000e-08	
28518	68204	functional	0.0	2011-07-18	Government Of Tanzania	0	Government	0.00000	-2.000000e-08	
40696	28134	functional	0.0	2011-07-18	Government Of Tanzania	0	Government	0.00000	-2.000000e-08	
301	70379	functional	0.0	2011-07-18	Government Of Tanzania	0	Government	0.00000	-2.000000e-08	
56268	70312	functional	0.0	2011-07-18	Government Of Tanzania	0	Government	0.00000	-2.000000e-08	
15097	64405	functional	0.0	2011-07-19	Government Of Tanzania	0	Government	0.00000	-2.000000e-08	K/S
37439	56859	functional	0.0	2011-07-19	Government Of Tanzania	0	Government	0.00000	-2.000000e-08	K/S
19733	32781	functional	0.0	2011-07-19	Government Of Tanzania	0	Government	0.00000	-2.000000e-08	M
25928	11721	functional	0.0	2011-07-19	Government Of Tanzania	0	Government	0.00000	-2.000000e-08	M

	id	status_group	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	w
8460	61071	functional	0.0	2011-07-26	Hesawa	0	DWE	0.00000	-2.000000e-08	
37202	16464	functional	0.0	2011-07-26	Hesawa	0	DWE	0.00000	-2.000000e-08	
7907	63570	functional	0.0	2011-07-27	Hesawa	0	DWE	0.00000	-2.000000e-08	
25300	4532	functional	0.0	2011-07-27	Hesawa	0	DWE	0.00000	-2.000000e-08	
31558	17141	functional	0.0	2011-07-27	Hesawa	0	DWE	0.00000	-2.000000e-08	
24855	71964	functional	0.0	2012-10-25	Dwsp	0	DWE	0.00000	-2.000000e-08	
34465	16967	functional	0.0	2012-10-25	Dwsp	0	DWE	0.00000	-2.000000e-08	
9600	19126	functional	0.0	2012-10-25	Dwsp	0	DWE	0.00000	-2.000000e-08	
53441	21595	functional	0.0	2012-10-25	Dwsp	0	DWE	0.00000	-2.000000e-08	

In [33]:

```
# Dropping duplicated data

df_merge = df_merge.drop_duplicates(subset = df_merge.columns[1:], keep= 'first')
```

In [34]:

```
df_merge.shape
```

Out[34]:

```
(59364, 41)
```

In [35]:

```
df_cleaned = df_merge.copy()
```

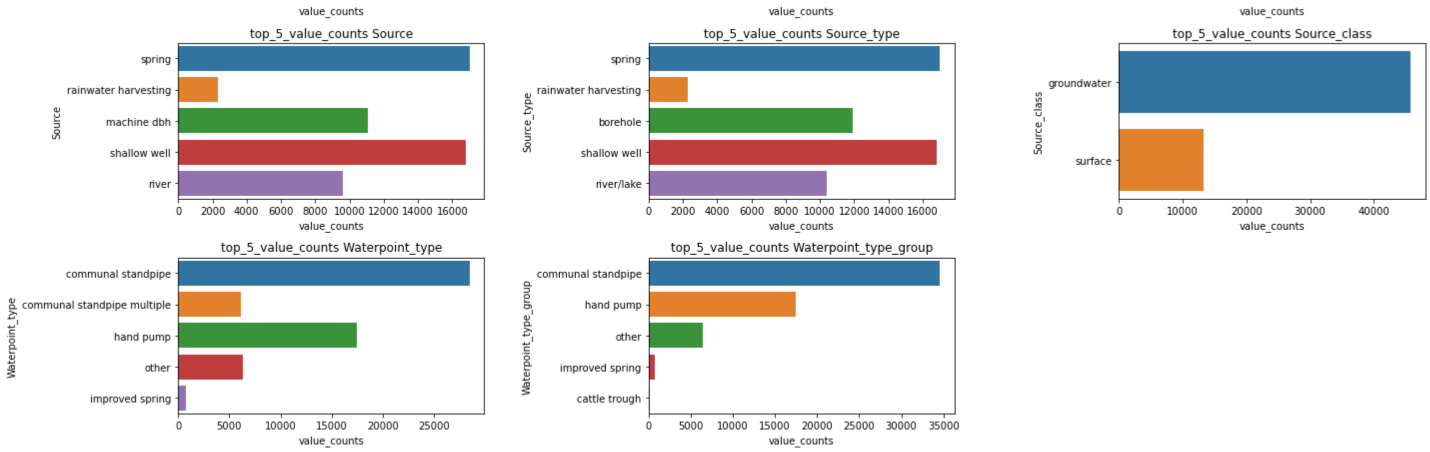
Univariate analysis

Categorical columns

In [36]:

```
# lets create visualisations to look at value counts of all data types that are object

object_columns = df_cleaned.select_dtypes(include = "object").columns

# initializing the columns to be on the figure
number_of_cols = 3
fig, axes = plt.subplots(nrows= math.ceil(len(object_columns)/ number_of_cols),ncols= num
ber_of_cols, figsize=(20,30))
# Flatten the axes for easier iteration
axes = axes.flatten()
```

```
for i, col in enumerate(object_columns):
    # Filter out 'unknown' values
    filtered_data = df_cleaned[df_cleaned[col] != 'unknown']
    top_values= filtered_data[col].value_counts().nlargest(5).index
    sns.countplot(y = col, data = filtered_data[filtered_data[col].isin(top_values)],ax=
axes[i])
    axes[i].set_title(f'top_5_value_counts {col.capitalize()}')
    axes[i].set_ylabel(col.capitalize())
    axes[i].set_xlabel("value_counts")

#removing any extra subplots that may form
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```



Insights

- **Status group** : most water points are considered to be functional, followed by non_functional and lastly functional but in need of repair
- **Recorded_by** :all records were recorded by the same company. These column can be removed as it wont provide any information for our model
- **funder** :govt of Tanzania has funded majority of the projects, followed by "Danida","Hesawa","Rwsp","World bank"
- **installer** : The leading installer is DWE(District Water Engineer). This indicates districts have hired water engineers to install the pumps followed by"Danida", "Commu","Government","RWE(Region Water Engineer)"
- **Source_class**:Majority of water is obtained from groundwater the rest is surface water
- **Basin** :majority of the walls points source their water from Lake Victoria, followed by lake Tanganyika,Lake Pangani and Lake Rufiji
- **Payment** : majority of the water points the population do not pay to access the water
- **Waterpoint type**In majority of the water points there is a communal standpipe
- **Water Quantity** :majority of the water points have water that is regarded enough by the residents in the area
- **Extraction type** :majority of the water points use gravity to extract water from its source
- **Extraction type class** : majority of the pumps are considered to be in the class of gravity
- **scheme_management** :majority of the water points are managed through vwc (Village Water Committee) followed by WUG (Water User Group), WUA (Water User Association), Water Board, Water Authority

Further Investigation

From the visualisions we discover that some columns contain similar information found in other columns example

- quantity and quantity_group
- extraction_type and extraction_type_class
- source and source_type
- payment and payment_type
- waterpoint_type and waterpoint_type_group
- management and management_group

Removing duplicated columns

In [37]:

```
def get_value_counts(df, col):
    return df[col].value_counts()
```

In [38]:

```
quantity= get_value_counts(df= df_cleaned,col ='quantity')
quantity_group = get_value_counts(df= df_cleaned,col= 'quantity_group')

quantity,quantity_group
```

Out[38]:

```
(enough          33165
 insufficient    15119
 dry              6243
 seasonal         4048
 unknown           789
 Name: quantity, dtype: int64,
 enough          33165
 insufficient    15119
 dry              6243
 seasonal         4048
 unknown           789
 Name: quantity_group, dtype: int64)
```

In [39]:

```
# we can remove one of these columns as they are duplicate of each other and also recorde
d by

df_cleaned.drop(columns=["quantity_group",'recorded_by'], inplace = True)
```

In [40]:

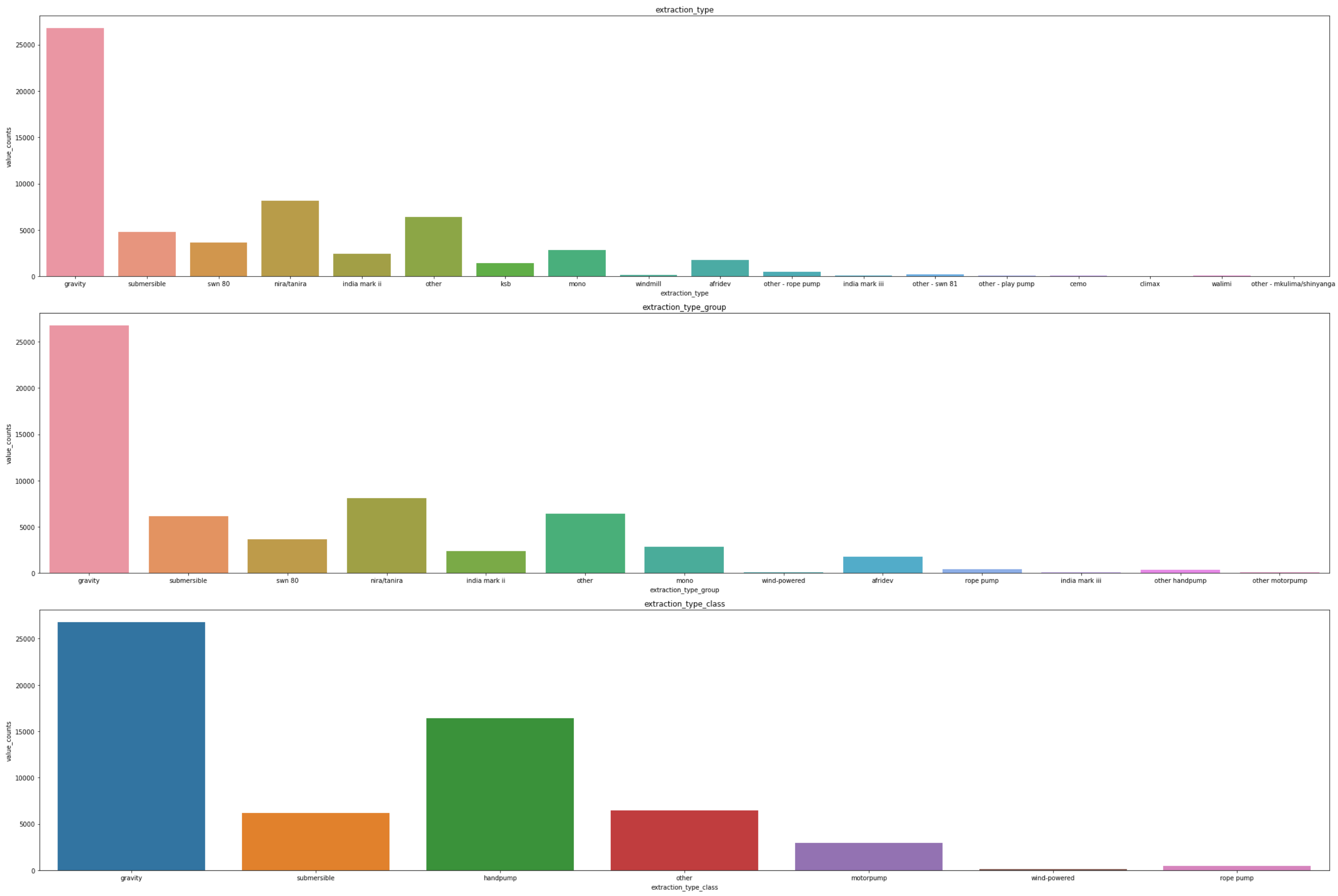
```
extraction_type = get_value_counts(df_cleaned,'extraction_type')
extraction_group = get_value_counts(df_cleaned,'extraction_type_group')
extraction_class = get_value_counts(df_cleaned,'extraction_type_class')

extraction_type,extraction_group,extraction_class
```

Out[40]:

```
(gravity                      26776
 nira/tanira                   8143
 other                         6427
 submersible                   4759
 swn 80                        3663
 mono                          2865
 india mark ii                 2398
 afridev                       1769
 ksb                           1413
 other - rope pump              451
 other - swn 81                 229
 windmill                       117
 india mark iii                  97
 cemo                            90
 other - play pump               85
 walimi                          48
 climax                          32
 other - mkulima/shinyanga        2
 Name: extraction_type, dtype: int64,
 gravity            26776
 nira/tanira         8143
 other               6427
 submersible         6172
 swn 80              3663
 mono                2865
 india mark ii       2398
 afridev             1769
 rope pump            451
 other handpump       364
 other motorpump      122
 wind-powered         117
 india mark iii        97
 Name: extraction_type_group, dtype: int64,
 gravity         26776
 handpump        16434
 other            6427
 submersible      6172
 motorpump        2987
 rope pump         451
 wind-powered      117
 Name: extraction_type_class, dtype: int64)
```

In [41]:

```
# checking extraction group value counts
extraction_columns= ['extraction_type','extraction_type_group','extraction_type_class']

fig,axes = plt.subplots( nrows= 3, ncols= 1, figsize = (30,20))

axes= axes.flatten()
for i,col in enumerate(extraction_columns):
    sns.countplot(x= col, data = df_cleaned, ax=axes[i])
    axes[i].set_title(col)
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('value_counts')

plt.tight_layout()
plt.show()
```



Extraction type class seems to group the type group data well so we are going to drop extraction_type_group and extraction_type

In [42]:

```
df_cleaned.drop(columns=['extraction_type_group','extraction_type'], inplace = True)
```

In [43]:

```
source= get_value_counts(df_cleaned,'source')
source_type= get_value_counts(df_cleaned,'source_type')

source,source_type
```

Out[43]:

```
(spring                  17020
 shallow well            16801
 machine dbh             11069
 river                    9612
 rainwater harvesting     2293
 hand dtw                  874
 lake                      763
```

```
 lake                      765
 dam                       655
 other                     211
 unknown                    66
 Name: source, dtype: int64,
 spring                  17020
 shallow well            16801
 borehole                11943
 river/lake              10375
 rainwater harvesting     2293
 dam                       655
 other                     277
 Name: source_type, dtype: int64)
```

In [44]:

```
# the columns again seem to hold similar information therefore we are going to drop sourc
e and remain with source type

df_cleaned.drop(columns="source", inplace = True)
```

In [45]:

```
waterpoint=get_value_counts(df_cleaned,'waterpoint_type')
waterpoint_group = get_value_counts(df_cleaned,'waterpoint_type_group')

waterpoint,waterpoint_group
```

Out[45]:

```
(communal standpipe             28516
 hand pump                      17466
 other                           6377
 communal standpipe multiple     6099
 improved spring                  783
 cattle trough                    116
 dam                                7
 Name: waterpoint_type, dtype: int64,
 communal standpipe    34615
 hand pump             17466
 other                  6377
 improved spring         783
 cattle trough           116
 dam                       7
 Name: waterpoint_type_group, dtype: int64)
```

In [46]:

```
# The columns have similar information we will again use the group option

df_cleaned.drop(columns="waterpoint_type", inplace = True)
```

In [47]:

```
payment= get_value_counts(df_cleaned,"payment")
payment_group = get_value_counts(df_cleaned,"payment_type")

payment,payment_group
```

Out[47]:

```
(never pay                25337
 pay per bucket            8984
 pay monthly               8300
 unknown                   8134
 pay when scheme fails     3914
 pay annually              3642
 other                     1053
 Name: payment, dtype: int64,
 never pay     25337
 per bucket     8984
 monthly        8300
 unknown        8134
```

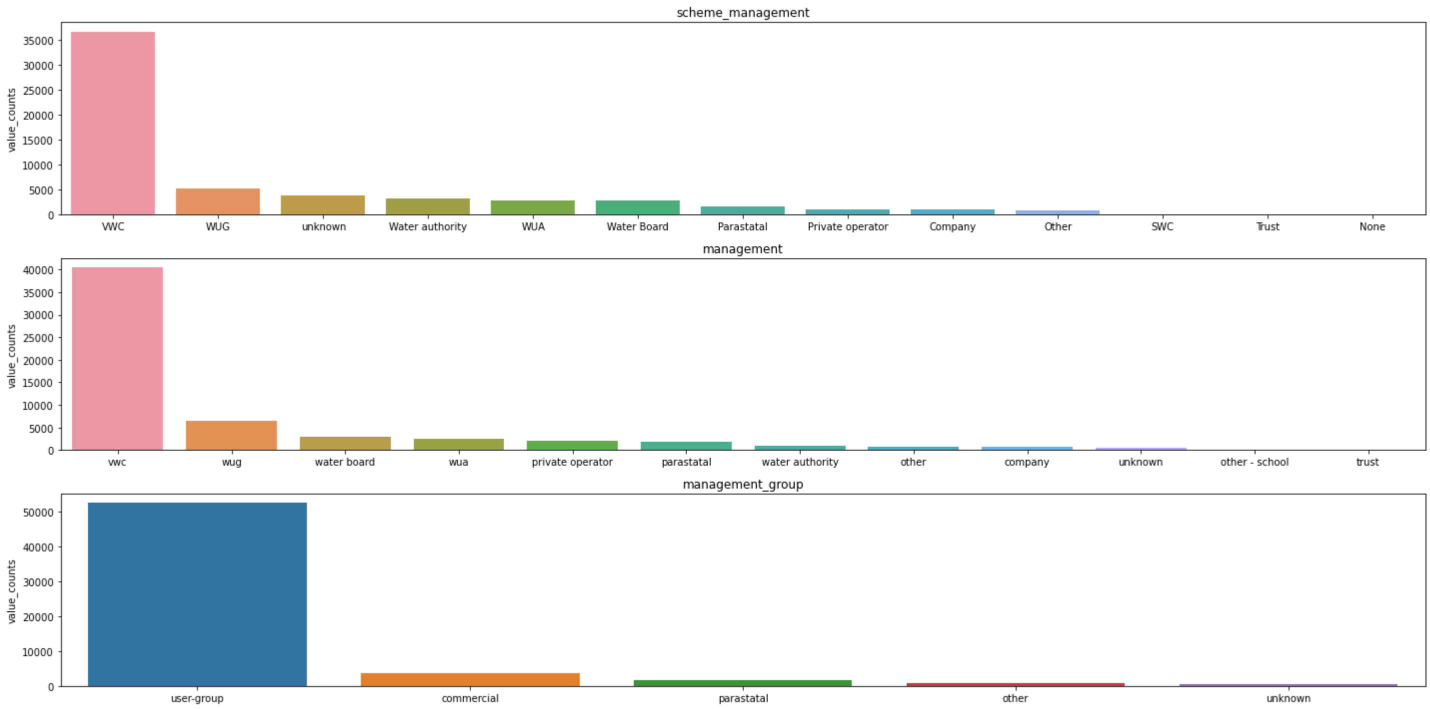
```
 on failure     3914
 annually       3642
 other          1053
 Name: payment_type, dtype: int64)
```

In [48]:

```
# The two columns are duplicate of each other

df_cleaned.drop(columns= "payment", inplace = True)
```

In [49]:

```
# plotting the three columns to get a better understanding on their contents
columns = ['scheme_management', 'management', 'management_group']
fig,axes = plt.subplots(nrows = 3, ncols= 1 , figsize = (20,10))

for i,col in enumerate(columns):
    sns.barplot(x= get_value_counts(df_cleaned,col).index, y=get_value_counts(df_cleaned
,col).values, ax= axes[i])
    axes[i].set_title(col)
    axes[i].set_ylabel("value_counts")

plt.tight_layout()
plt.show()
```



In [50]:

```
scheme_management= get_value_counts(df_cleaned,'scheme_management')
management= get_value_counts(df_cleaned,"management")
management_group = get_value_counts(df_cleaned,"management_group")

scheme_management,management,management_group
```

Out[50]:

```
(VWC                 36779
 WUG                  5186
 unknown              3877
 Water authority      3153
 WUA                  2883
 Water Board          2748
 Parastatal           1678
 Private operator     1063
 Company              1061
 Other                 766
 SWC                    97
 Trust                  72
```

```
None                         1
Name: scheme_management, dtype: int64,
vwc                     40493
wug                      6495
water board              2933
wua                      2535
private operator         1971
parastatal               1766
water authority           904
other                     844
company                   685
unknown                   561
other - school             99
trust                      78
Name: management, dtype: int64,
user-group    52456
commercial     3638
parastatal     1766
other           943
unknown         561
Name: management_group, dtype: int64)
```

It seems scheme managemnt is not similar to the management and management group

managemnt group is grouped in the following ways

- **user-group = vwc + wug+ water board + wua**
- **commercial = private operator + water authority + company + trust**
- **parastatal = parastatal**
- **other = other + other - school**
- **unknown = unknown**

with this understanding will take the management group column

In [51]:

```
df_cleaned.drop(columns= "management", inplace = True)
```

In [52]:

```
water_quality = get_value_counts(df_cleaned,'water_quality')
quality_group = get_value_counts(df_cleaned,'quality_group')

water_quality,quality_group
# the columns have similar information and we will use the group option
```

Out[52]:

```
(soft                      50785
 salty                      4856
 unknown                    1873
 milky                       804
 coloured                    490
 salty abandoned             339
 fluoride                    200
 fluoride abandoned           17
 Name: water_quality, dtype: int64,
 good        50785
 salty        5195
 unknown      1873
 milky         804
 colored       490
 fluoride      217
 Name: quality_group, dtype: int64)
```

In [53]:

```
df_cleaned.drop(columns= "water_quality", inplace = True)
```

In [54]:

```
df_cleaned.shape
```

Out[54]:

```
(59364, 32)
```

Numerical columns

In [55]:

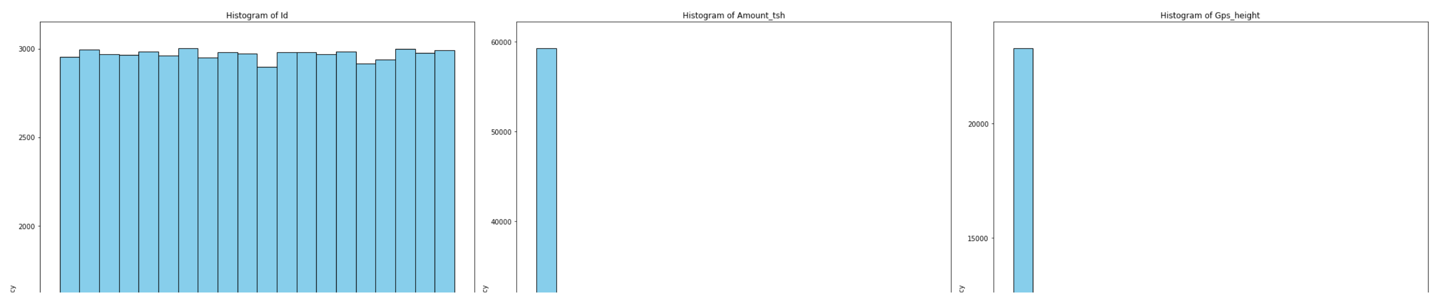
```
df_cleaned.describe()
```

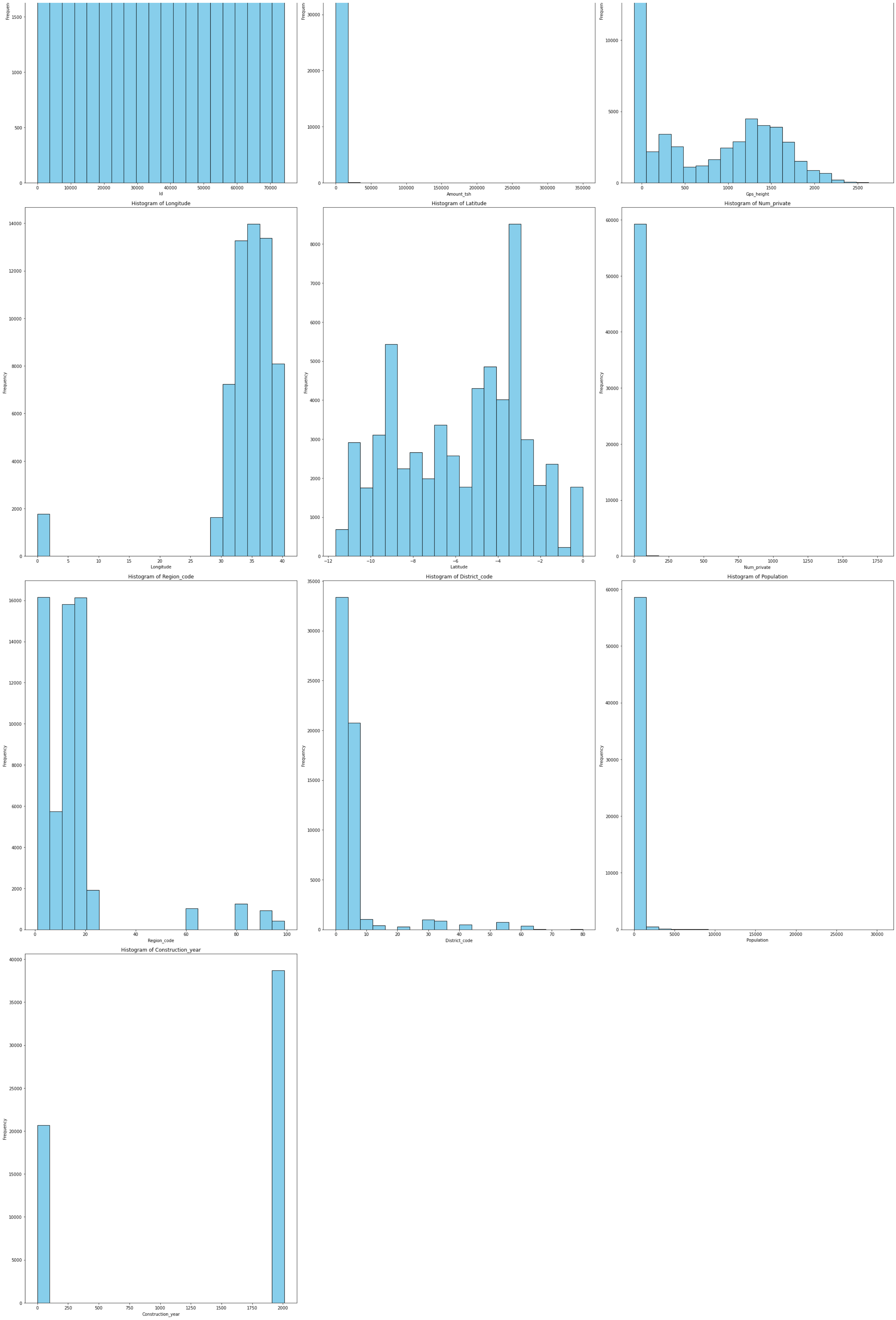
Out[55]:

	id	amount_tsh	gps_height	longitude	latitude	num_private	region_code	district_code	
count	59364.000000	59364.000000	59364.000000	59364.000000	5.936400e+04	59364.000000	59364.000000	59364.000000	59
mean	37117.957988	317.843017	668.702513	34.097560	-5.709463e+00	0.474429	15.295516	5.631494	
std	21451.843216	2998.473133	693.131013	6.517065	2.943608e+00	12.239934	17.592619	9.636138	
min	0.000000	0.000000	-90.000000	0.000000	-1.164944e+01	0.000000	1.000000	0.000000	
25%	18527.250000	0.000000	0.000000	33.095187	-8.541904e+00	0.000000	5.000000	2.000000	
50%	37063.500000	0.000000	370.000000	34.910318	-5.023822e+00	0.000000	12.000000	3.000000	
75%	55656.500000	20.000000	1320.000000	37.179490	-3.326918e+00	0.000000	17.000000	5.000000	
max	74247.000000	350000.000000	2770.000000	40.345193	-2.000000e-08	1776.000000	99.000000	80.000000	30

In [56]:

```
numerical_columns = df_cleaned.select_dtypes(include=['int64','float64']).columns

# Plotting histograms for numerical columns
fig, axes = plt.subplots(nrows=math.ceil(len(numerical_columns)/3), ncols=3, figsize=(30
, 5 * len(numerical_columns)))
axes = axes.flatten()
for i, col in enumerate(numerical_columns):
    df_cleaned[col].plot(kind='hist', ax=axes[i], bins=20, color='skyblue', edgecolor='b
lack')
    axes[i].set_title(f'Histogram of {col.capitalize()}')
    axes[i].set_xlabel(col.capitalize())
    axes[i].set_ylabel('Frequency')



#removing any extra subplots that may form
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])


plt.tight_layout()
plt.show()
```




Id
Amount_tsh
Gps_height
Histogram of Longitude
Longitude
Histogram of Latitude
Latitude
Histogram of Num_private
Num_private
Histogram of Region_code
Region_code
Histogram of District_code
District_code
Histogram of Population
Population
Histogram of Construction_year
Construction_year
Frequency

Insights

amount_tsh : majority of the total static head seem to lie between 0 and 5000 with high level at 0 this may indicate the vertical distance between the source and the extraction point is on the same level or there is no value for this data **gps_height** : gps height seems to be normally distributed but with a positive skew due to majority of the values lying at 0. There is also presence of negative values. This indicates that some of wells are located below sea level **longitudes** : the longitudes lie within the same area but there are longitudes that are 0. this indicates they are missing values in the longitudes field **latitudes** : the latitudes lie where you expect Tanzania to be located **Regioncode** : Region code shows that majority of the water point can be found in one region **Population** : Population shows that the populations where this water points are located have low population. This might be accurate as abundance of wells in African countries are found in areas that are sparsely populated **Construction_year** : The construction year shows that majority of the wells were built almost at the same time. The column has a lot of missing data

Further Investigation

- Missing values are located in longitudes and construction year
- num_private analyse what data is found on that column

In [57]:

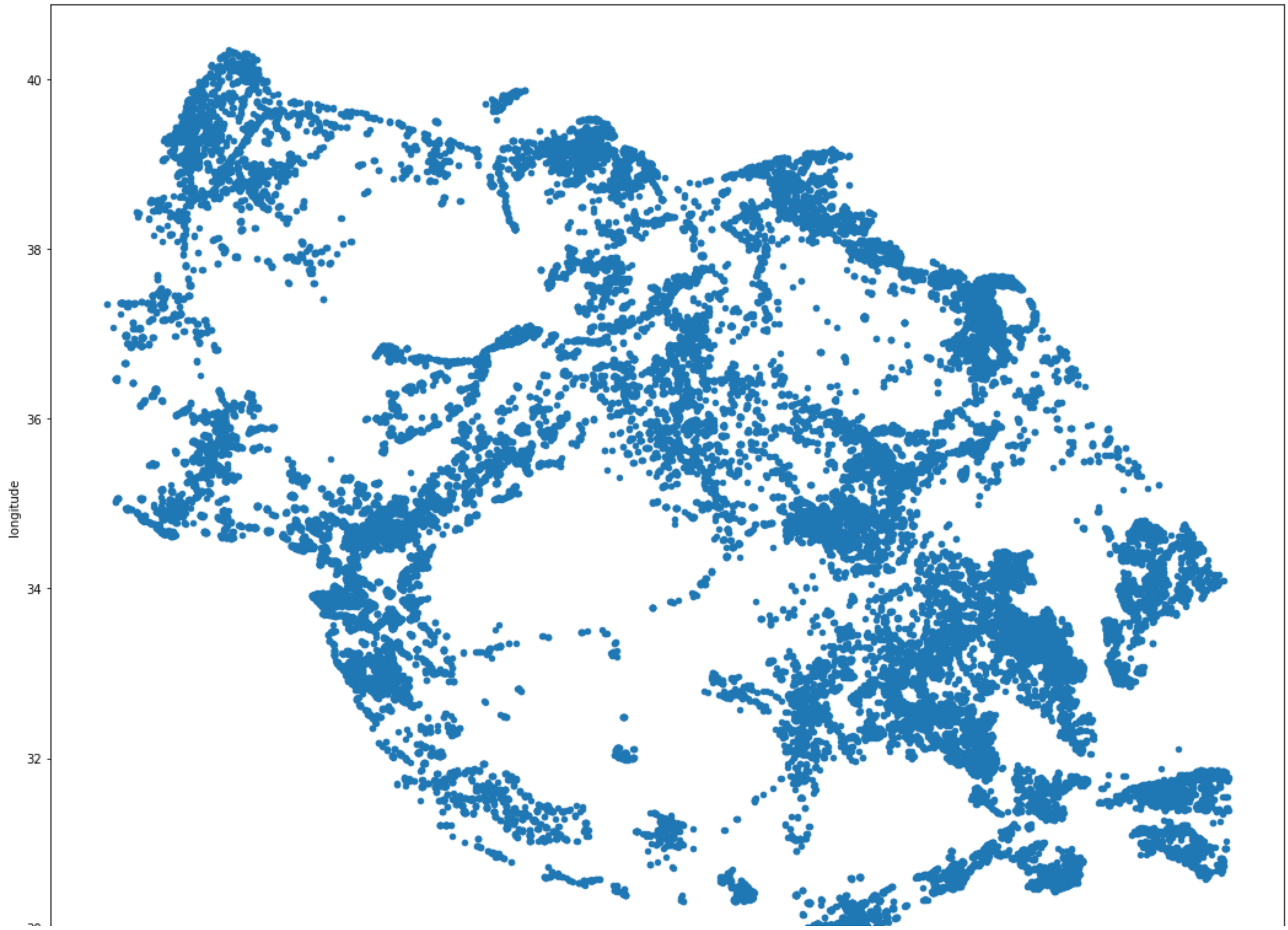
```
#filling longitude mising values by the median value and removing extreme values in the l
atitude

df_cleaned['longitude'].where(df_cleaned['longitude']> 25, df_cleaned['longitude'].media
n(),inplace= True)
```

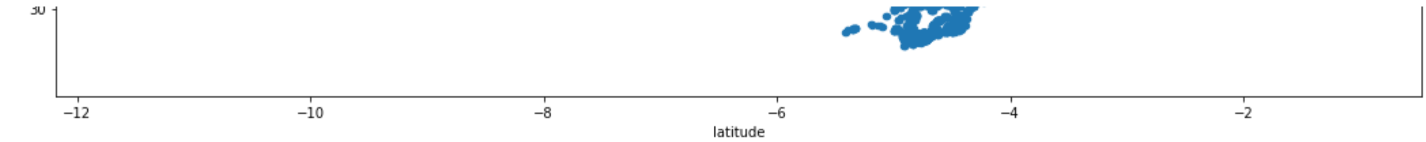
In [58]:

```
df_cleaned['latitude'].where(df_cleaned['latitude']<-1, df_cleaned['latitude'].median(),
inplace = True)
```

In [59]:

```
df_cleaned.plot(x='latitude',y='longitude',kind="scatter",figsize=(18,15))
plt.show();
```



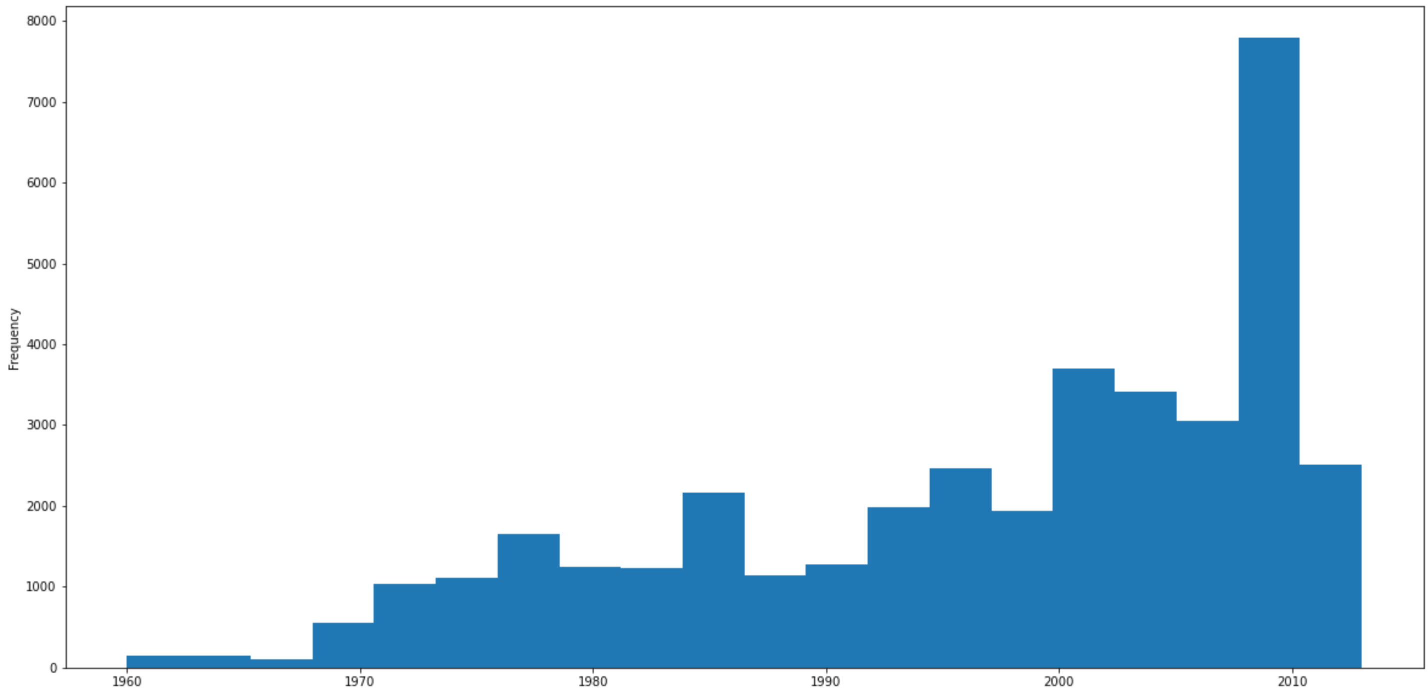


In [60]:

```
#plotting construnction year without the missing values
df_cleaned['construction_year'].where(df_cleaned['construction_year']>1750).plot(kind= 'h
ist', bins= 20, figsize= (20,10))
```

Out[60]:

```
<AxesSubplot:ylabel='Frequency'>
```



In [61]:

```
df_cleaned['construction_year'].describe()
```

Out[61]:

```
count    59364.000000
mean      1301.441227
std        951.369704
min          0.000000
25%          0.000000
50%       1986.000000
75%       2004.000000
max       2013.000000
Name: construction_year, dtype: float64
```

In [62]:

```
df_cleaned['construction_year'].value_counts()
```

Out[62]:

```
0       20673
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
```

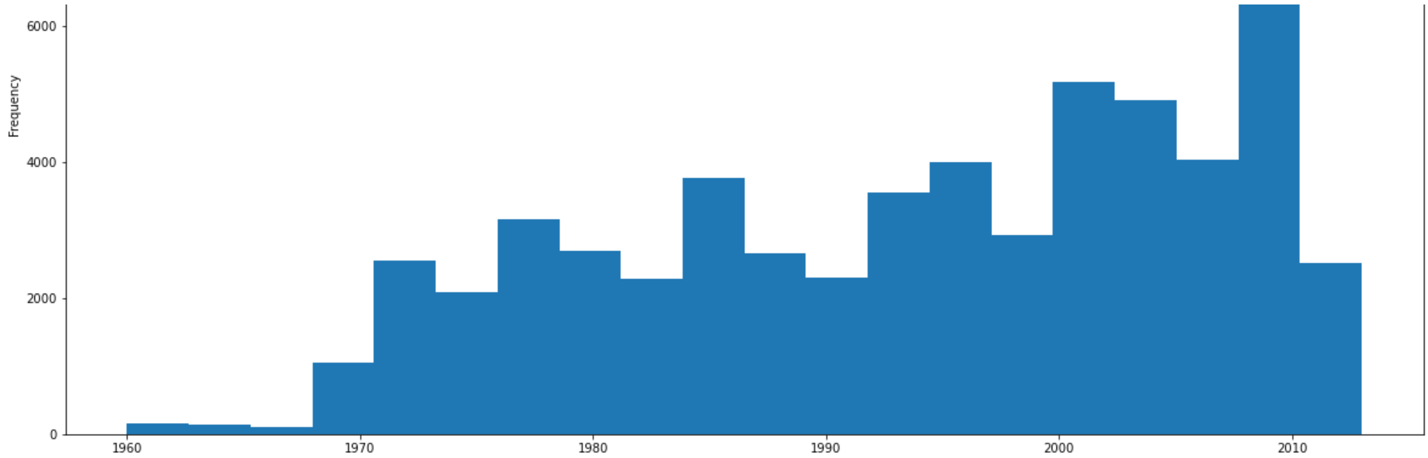
We can see that majority of the values lie after 1970. Due to the large number of missing values we will create random years that lie between 1970 and 2010

In [63]:

```
# Creating random construction years that lie inbetween 1970 and 2010
np.random.seed(seed= 42)
random_years = np.random.randint(1970,2010 +1, size= df_cleaned['construction_year'].val
ue_counts()[0])

#replacing random values
df_cleaned.loc[df_cleaned['construction_year']== 0, 'construction_year'] = random_years

df_cleaned['construction_year'].plot(kind= 'hist', bins= 20, figsize= (20,10))
```

Out[63]:

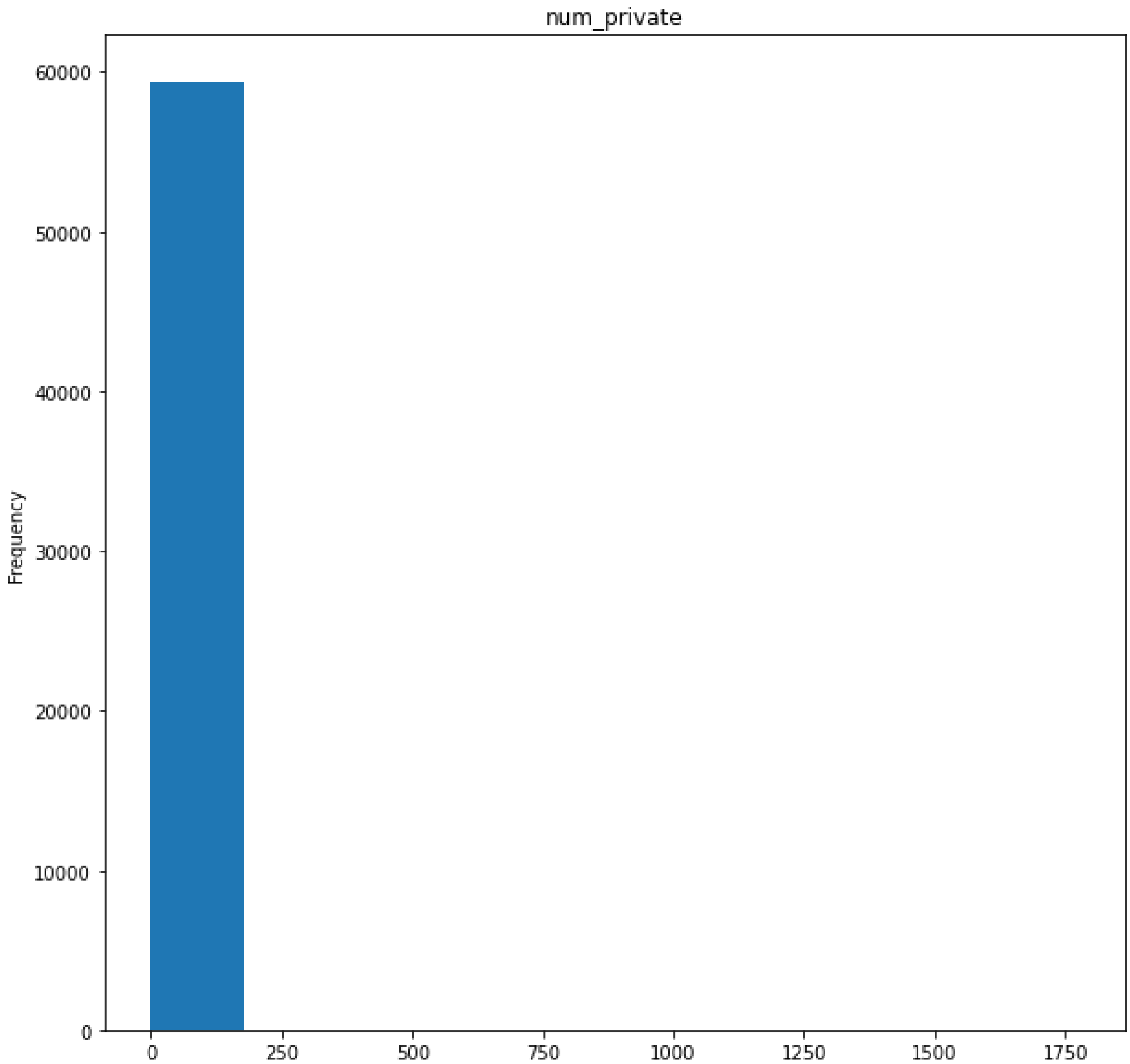
```
<AxesSubplot:ylabel='Frequency'>
```



This provides a better distribution of what we would expect from the previous observations

In [64]:

```
# num_private column

df_cleaned['num_private'].plot(kind= 'hist', bins= 10, figsize= (10,10))
plt.title("num_private")
plt.show()

# With lack of understanding of this column i will drop the column from the dataset
```



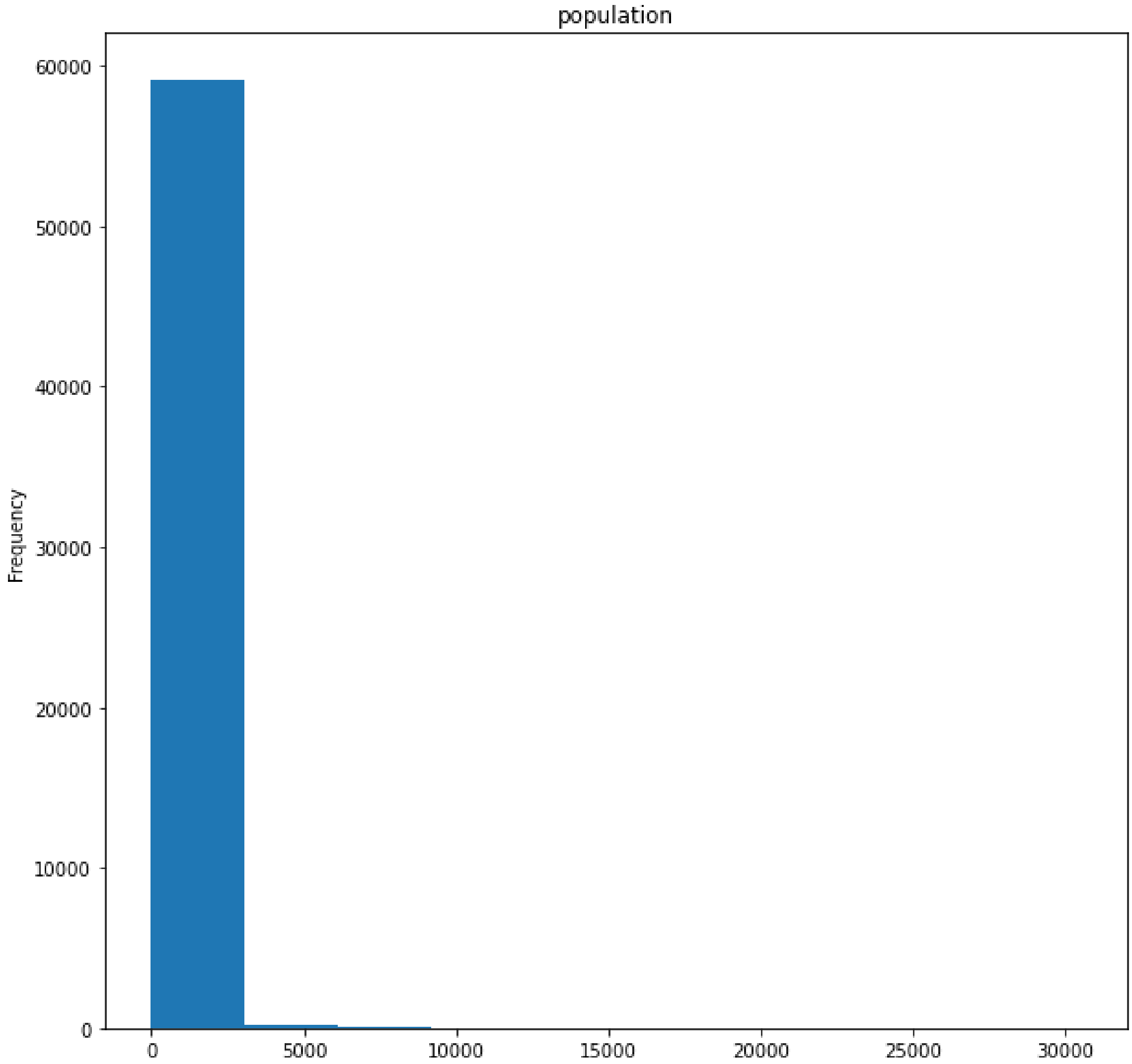
In [65]:

```
df_cleaned.drop(columns='num_private', inplace = True)
```

In [66]:

```
df_cleaned['population'].plot(kind= 'hist',bins= 10, figsize= (10,10))
```

```
plt.title("population")
plt.show()
```



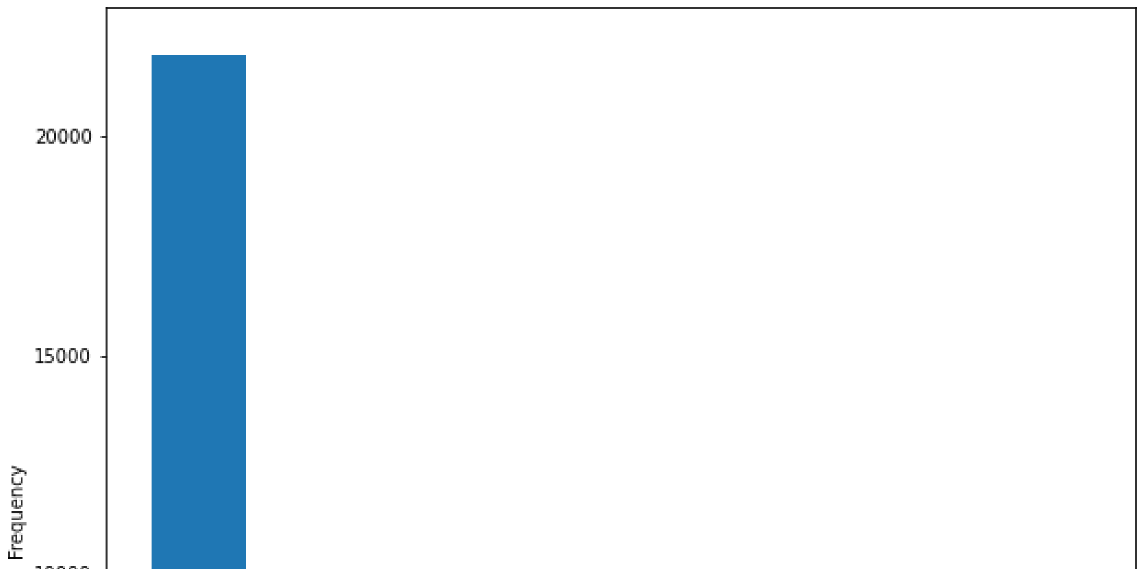
The above columns seem to contain a lot of missing values

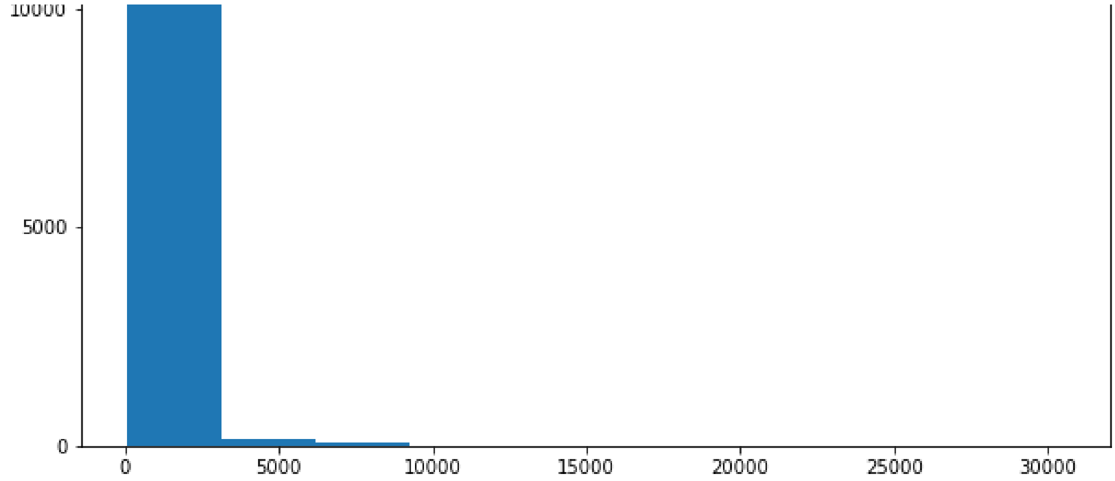
In [67]:

```
# checking how the datatset looks without the missing values plotted
df_cleaned['population'].where(df_cleaned['population']>100).plot(kind= 'hist',bins= 10,
figsize= (10,10))
```

Out[67]:

<AxesSubplot:ylabel='Frequency'>





The dataset seems to lean close to the 0 value. This can be because if a well dries up the people living in the location tend to move or settle in other locations or it might still be missing values

In [68]:

```
df_cleaned['population'].describe()
```

Out[68]:

```
count    59364.000000
mean       180.019086
std        471.604294
min          0.000000
25%          0.000000
50%         25.000000
75%        215.000000
max      30500.000000
Name: population, dtype: float64
```

In [69]:

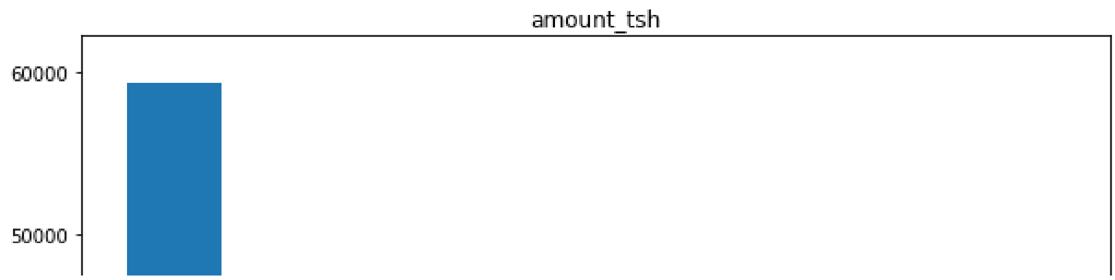
```
df_cleaned['population'].value_counts()
```

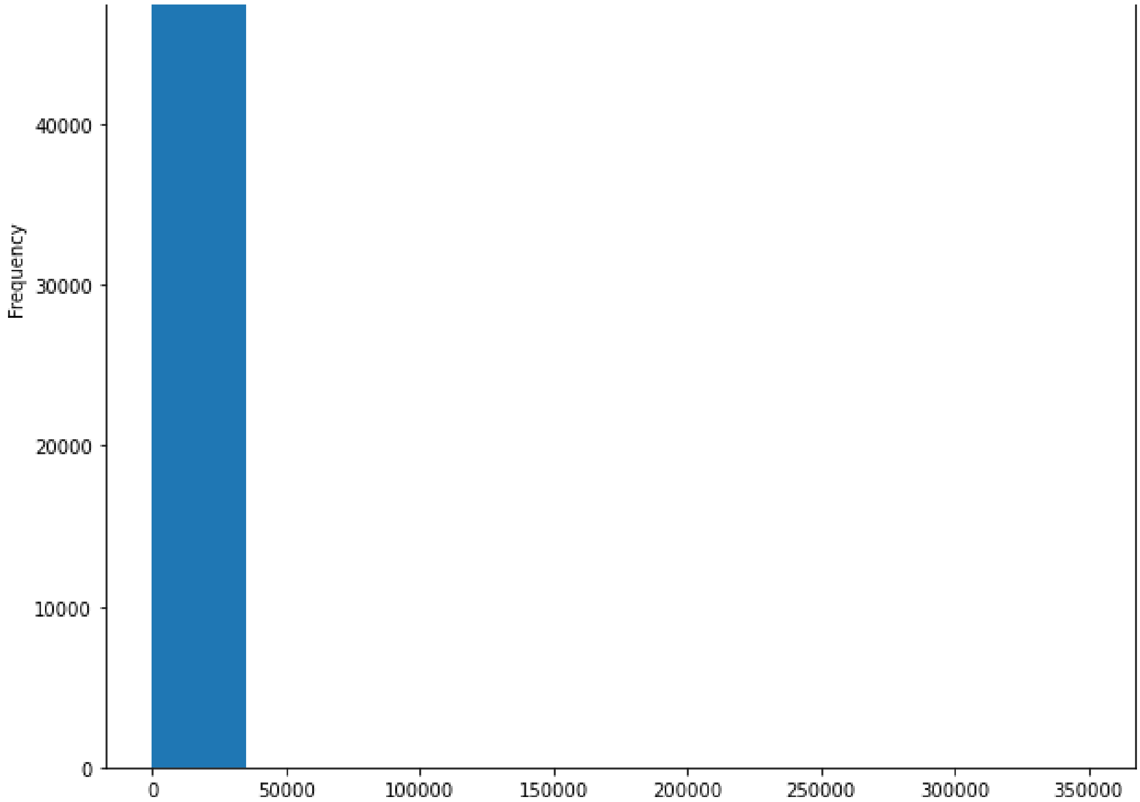
Out[69]:

```
0       21345
1        7025
200      1940
150      1892
250      1681
        ...  
3241        1
1960        1
1685        1
2248        1
1439        1
Name: population, Length: 1049, dtype: int64
```

In [70]:

```
df_cleaned['amount_tsh'].plot(kind= 'hist',bins= 10, figsize= (10,10))


plt.title("amount_tsh")
plt.show()
```



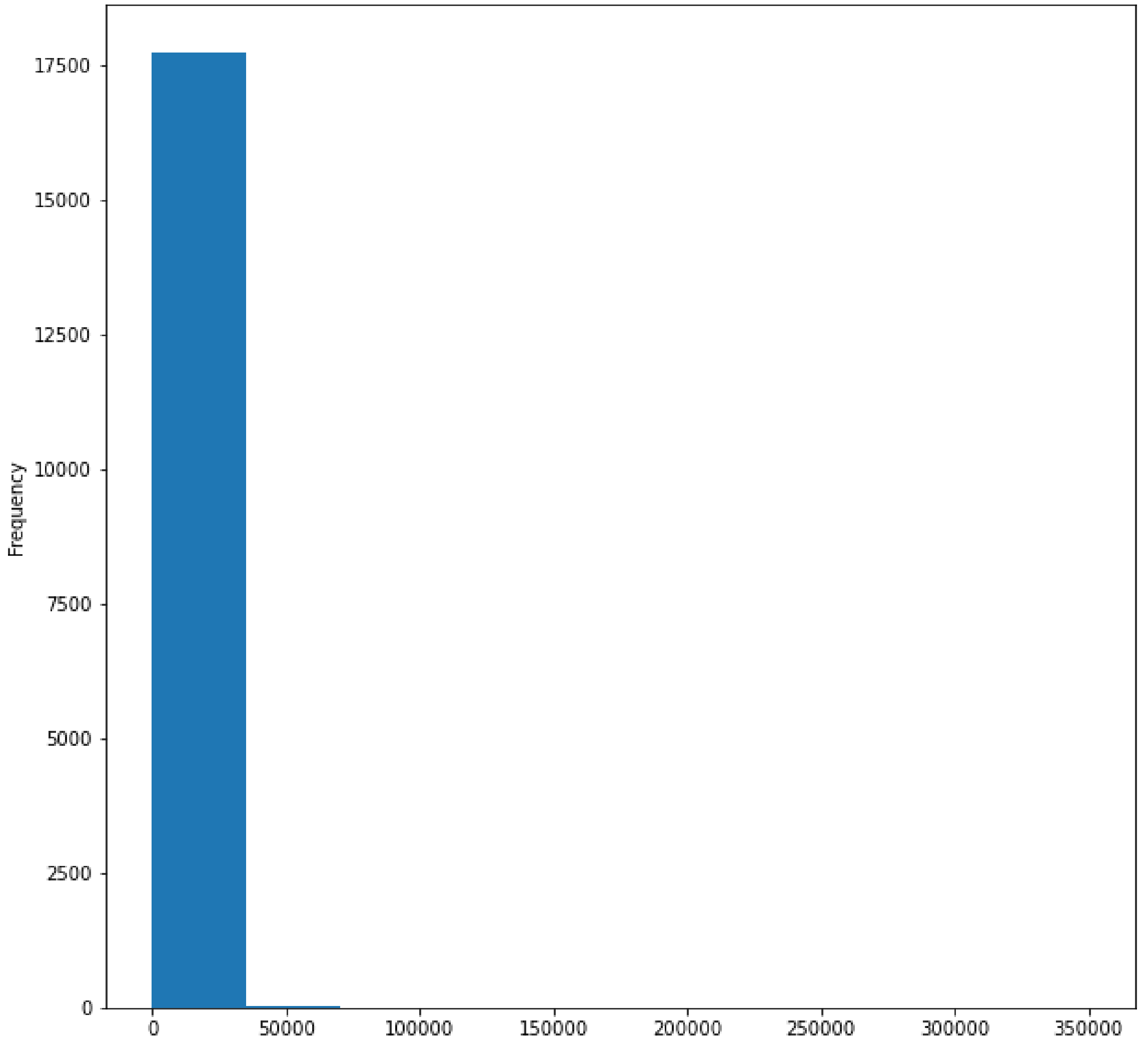


In [71]:

```
# checking how the datatset looks without the missing values plotted
df_cleaned['amount_tsh'].where(df_cleaned['amount_tsh']>0).plot(kind= 'hist',bins= 10, f
igsize= (10,10))
```

Out[71]:

```
<AxesSubplot:ylabel='Frequency'>
```



tsh starts for total static head which means the vertical distance between from the source to the discharge point. From the above visualisation it might mean that majority of the vertical distance is closer to zero or they can be missing values

In [72]:

```
df_cleaned['amount_tsh'].describe()
```

Out[72]:

```
count     59364.000000
mean        317.843017
std        2998.473133
min           0.000000
25%           0.000000
50%           0.000000
75%          20.000000
max      350000.000000
Name: amount_tsh, dtype: float64
```

In [73]:

```
df_bivariate = df_cleaned.copy()
```

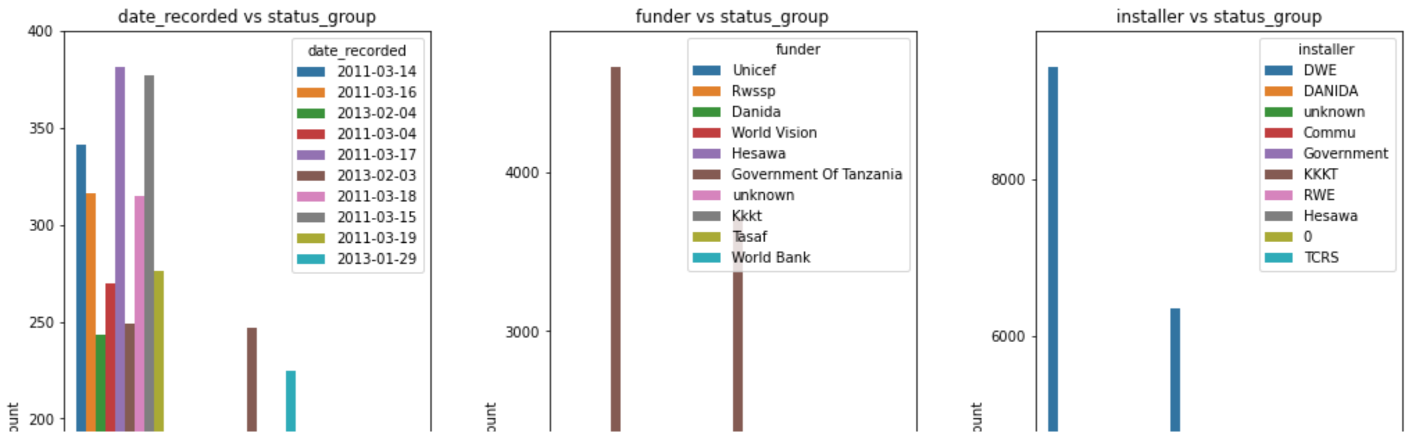
Bivariate analysis

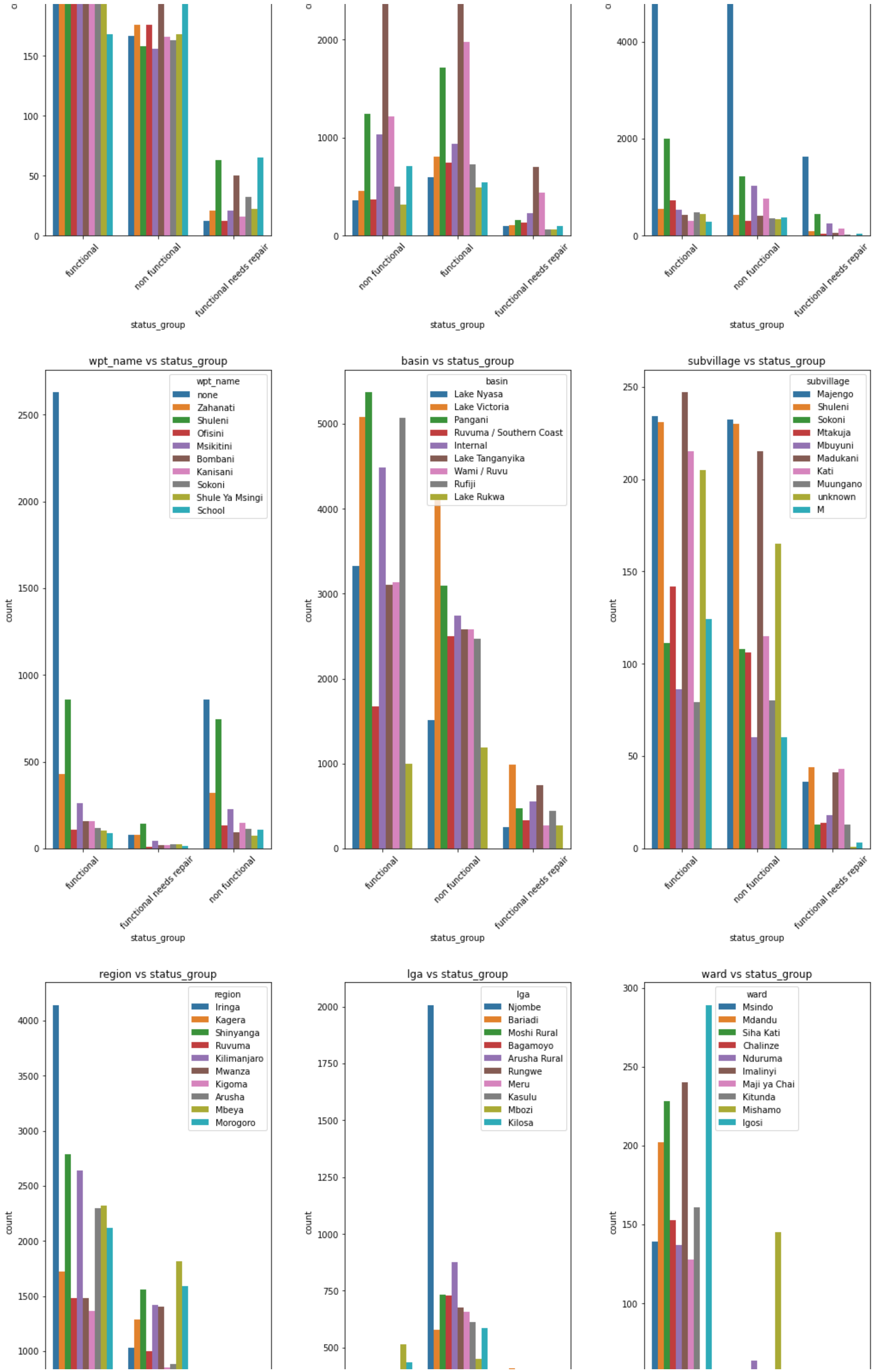
Categorical columns

stacked bar chart

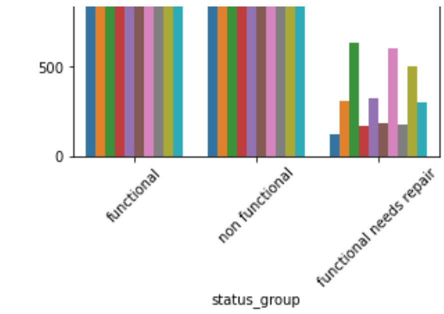
In [74]:

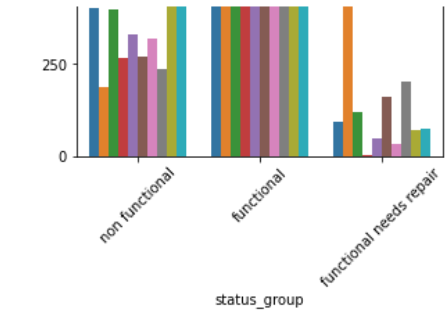
```
# Select categorical columns
categorical_columns = df_bivariate.select_dtypes(include=['object']).columns

# Plot bivariate analysis for categorical features
fig, axes = plt.subplots(nrows=math.ceil(len(categorical_columns)/3), ncols=3,figsize=(1
5, 70))
axes = axes.flatten()
for i, col in enumerate(categorical_columns[1:], start =1):
    if col != 'status_group':
        top_values = df_bivariate[col].value_counts().head(10).index
        sns.countplot(x='status_group', hue=col, data=df_bivariate[df_bivariate[col].isi
n(top_values)], ax=axes[i-1])
        axes[i-1].set_title(f'{col} vs status_group')
        axes[i-1].tick_params(axis='x', rotation=45) # Rotate x-axis labels

# Remove any unused subplots
for j in range(i , len(axes)):
    fig.delaxes(axes[j])
plt.tight_layout(pad=3.0)
plt.show()
```

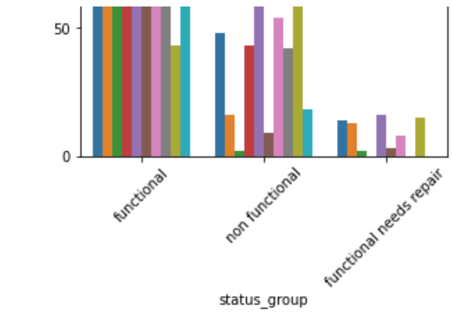


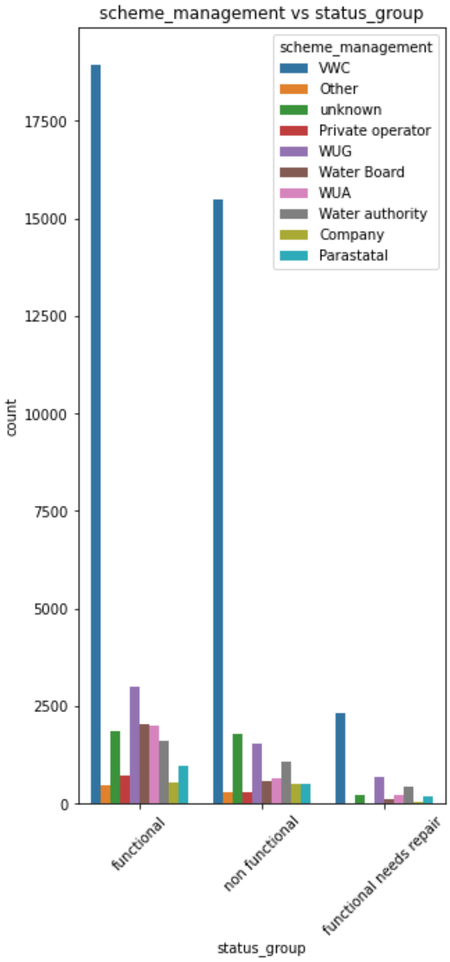


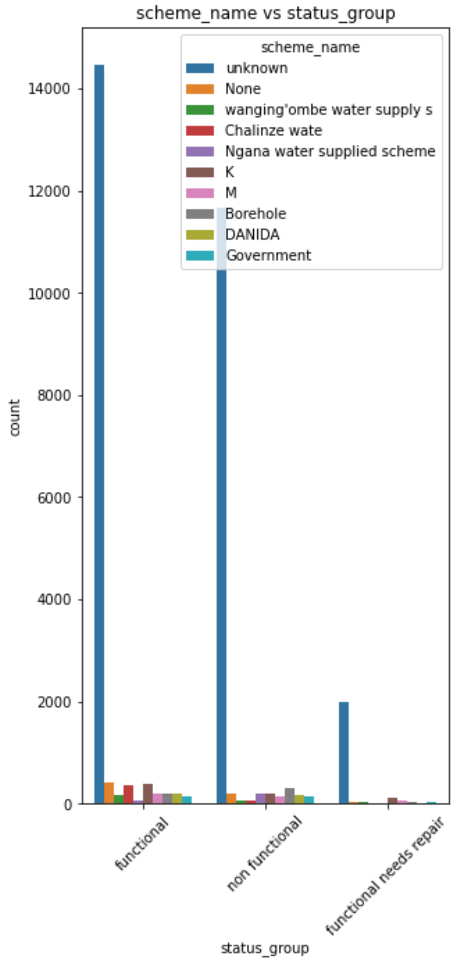
status_group
functional
non functional
functional needs repair
count
wpt_name vs status_group
wpt_name
none
Zahanati
Shuleni
Ofisini
Msikitini
Bombani
Kanisani
Sokoni
Shule Ya Msingi
School
basin vs status_group
basin
Lake Nyasa
Lake Victoria
Pangani
Ruvuma / Southern Coast
Internal
Lake Tanganyika
Wami / Ruvu
Rufiji
Lake Rukwa
subvillage vs status_group
subvillage
Majengo
Shuleni
Sokoni
Mtakuja
Mbuyuni
Madukani
Kati
Muungano
unknown
M
region vs status_group
region
Iringa
Kagera
Shinyanga
Ruvuma
Kilimanjaro
Mwanza
Kigoma
Arusha
Mbeya
Morogoro
lga vs status_group
lga
Njombe
Bariadi
Moshi Rural
Bagamoyo
Arusha Rural
Rungwe
Meru
Kasulu
Mbozi
Kilosa
ward vs status_group
ward
Msindo
Mdandu
Siha Kati
Chalinze
Nduruma
Imalinyi
Maji ya Chai
Kitunda
Mishamo
Igosi

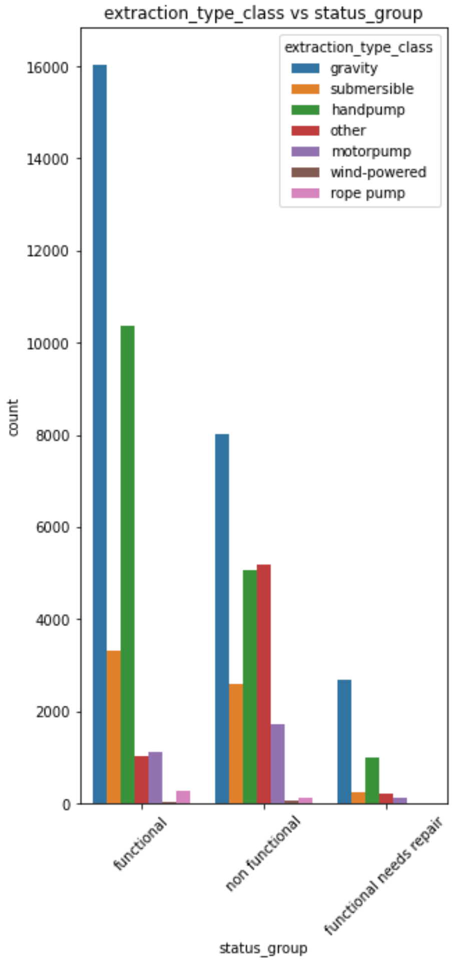

500
0
functional
non functional
functional needs repair
status_group

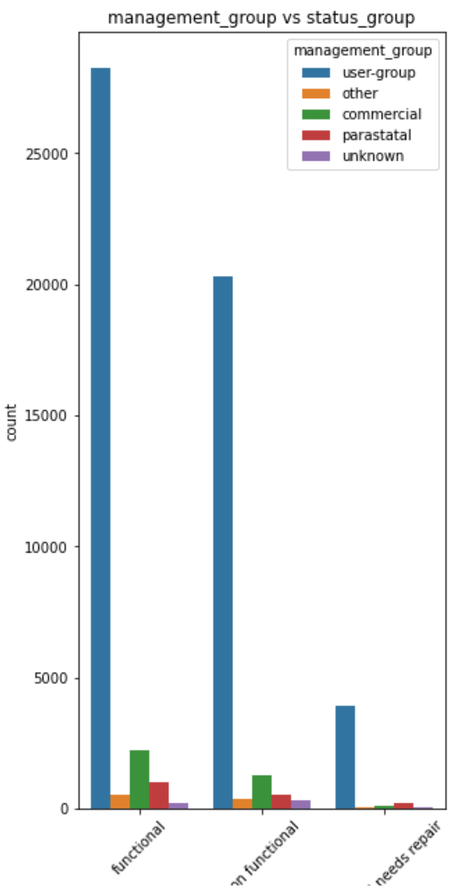

250
0
non functional
functional
functional needs repair
status_group

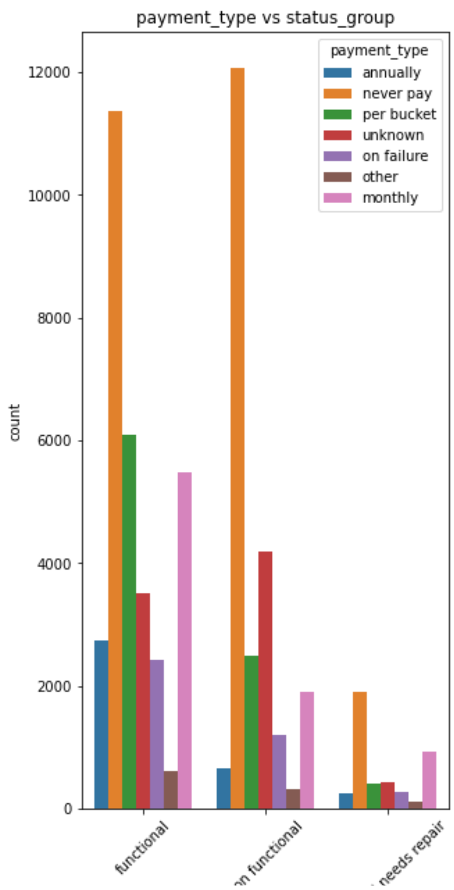

50
0
functional
non functional
functional needs repair
status_group

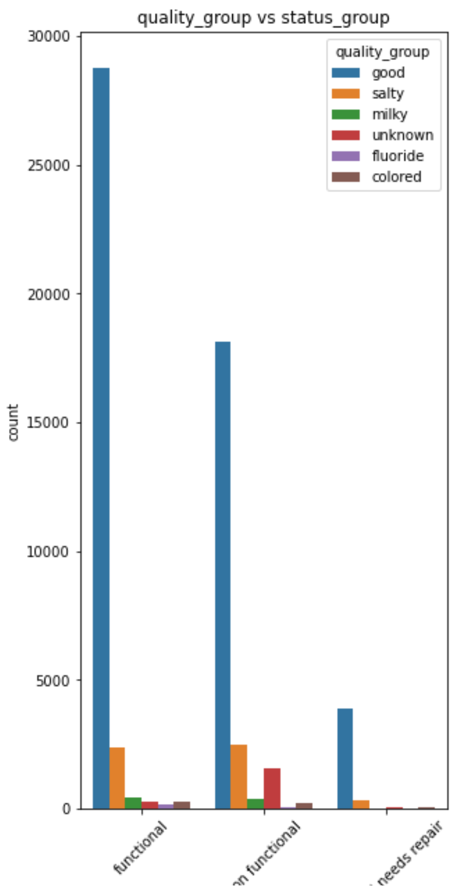

scheme_management vs status_group
scheme_management
VWC
Other
unknown
Private operator
WUG
Water Board
WUA
Water authority
Company
Parastatal
count
17500
15000
12500
10000
7500
5000
2500
0
functional
non functional
functional needs repair
status_group

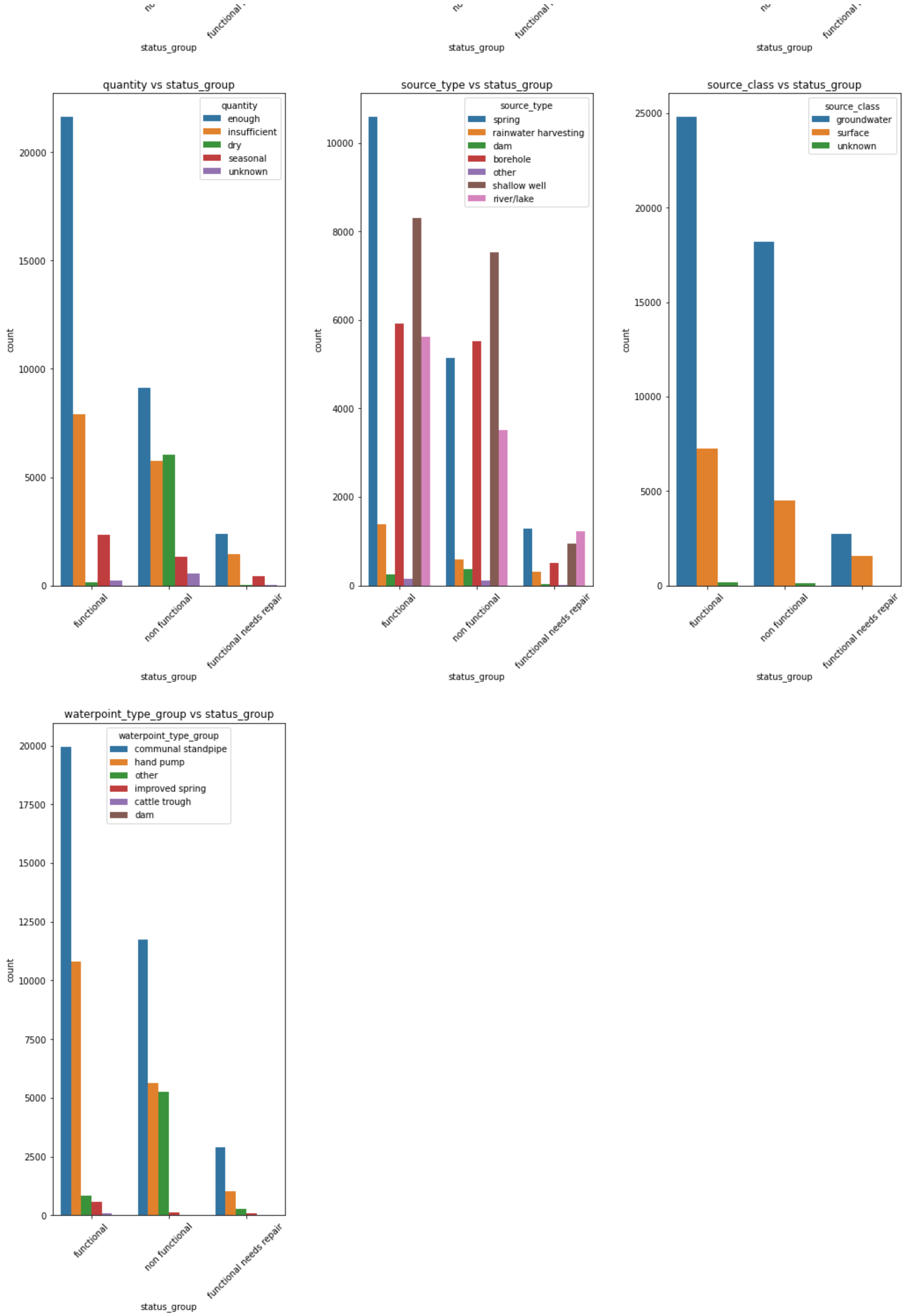

scheme_name vs status_group
scheme_name
unknown
None
wanging'ombe water supply s
Chalinze wate
Ngana water supplied scheme
K
M
Borehole
DANIDA
Government
count
14000
12000
10000
8000
6000
4000
2000
0
functional
non functional
functional needs repair
status_group


extraction_type_class vs status_group
extraction_type_class
gravity
submersible
handpump
other
motorpump
wind-powered
rope pump
count
16000
14000
12000
10000
8000
6000
4000
2000
0
functional
non functional
functional needs repair
status_group


management_group vs status_group
management_group
user-group
other
commercial
parastatal
unknown
count
25000
20000
15000
10000
5000
0
functional


payment_type vs status_group
payment_type
annually
never pay
per bucket
unknown
on failure
other
monthly
count
12000
10000
8000
6000
4000
2000
0
functional


quality_group vs status_group
quality_group
good
salty
milky
unknown
fluoride
colored
count
30000
25000
20000
15000
10000
5000
0
functional



Insights

- **funder and status group :** The government of Tanzania builds the most wells, this shows that majority of the

- **funder and status group** : The government of Tanzania builds the most wells, this shows that majority of the wells they build are also non functional, they also lead in the functional and need repairs categories
- **installer and status group** : District Water Engineer installed the most pumps in majority of the well so its safe to assume that they will be the most prevalent showcased on this chart
- **basin and status group** : In the Univariate analysis we noted that Lake Victoria was the source of the water for majority of the wells. But in the bivariate analysis we note that Lake Pangani wells are the most functional. Will be interesting to look
- **subvillage and statusgroup** : In the univariate analysis Majengo, shuleni and Madukani had the most well built in their locations. In the bivariate analysis we can see that madukani has the most functional wells from the other two while Majengo subvillage has almost no difference whether well located in their location will be functional or non functional
- **region and statusgroup** : In the univariate analysis we saw that wells were majority built in Iringa. In the bivariate analysis we notice that majority of well built in Iringa are functional while the region mbeya which was our third most built area where wells were built has the most non functional wells
- **iga nad status group** : In the univariate analysis we noted that Njombe had the most well built in their location. It is interesting to note that wells built in this region also leads in that most of their wells are considered functional
- **ward and status group** : In the univariate analysis we noted that Igosi had the most number of wells located in their region. In the bivariate analysis we note that wells built in Igosi are also considered to be functional while well built in Mishamo are considered to be non functional
- **scheme_mgt and status group** : Village Water Committee (VWC) managemntt type managed the most wells. Due to majority of the wells being managed by a VWC its safe to assume they will have managed a lot of functional and non functional wells
- **management_group and status group** :user group had the majority type of managemnt group by a huge amount. Therefore its safe to assume that they will lead on both having functional and non functional wells
- **payment type and status group** : Most well, required the users not to pay for the service therefore they would lead in majority of the classes but is interesting to see that most none functional wells users were not paying while where the users had to pay for the service had less none functioning wells. This may indicate when users were not paying when the pump breaks the managemt group lacked the finances to repair the well
- **quality group vs status group** : Majority of the wells had good quality of water so as expected they would lead in majority of occurences in regards to the target variable
- **quantity group vs status group** : Majority of the wells were regarded to have water quantity that was considered to be enough therefore it will lead in majority of the classes in regards to the target variable. However in this column there is presence where data that mean almost the same thing are separate i.e insufficient and seasonal
- **sourcetype vs status group** : Springs and shallow well led in the most occurence type of water source.Its interesting to note that water that come from springs functional while water sourced from shallow wells have the most non functioning wells
- **source class vs status group** : Groundwater was the most prevalent source class so it correlates that it will be the most prevalent in regards to our target variable
- **waterpoint type vs status group** : Communal standpipe was the most prevalent in terms of type of waterpoint type group.It correlates that it will lead in majority with regards to the target variable

Further Investigation

- We have a lot of data with regards to location.To improve the performance of the model we have to drop some of the columns so to get a generalised location.
- We have to also remove data that are in regards to the names as they dont correlate with our target variable
- To get better information we will extract the year from that date recorded. We will then create a new column called year in service that is the year in between when the well was constructed to the year recorded

In [75]:

```
df_bivariate.columns
```

Out[75]:

```
Index(['id', 'status_group', 'amount_tsh', 'date_recorded', 'funder',
       'gps_height', 'installer', 'longitude', 'latitude', 'wpt_name', 'basin',
       'subvillage', 'region', 'region_code', 'district_code', 'lga', 'ward',
       'population', 'public_meeting', 'scheme_management', 'scheme_name',
       'permit', 'construction_year', 'extraction_type_class',
```

```
        'management_group', 'payment_type', 'quality_group', 'quantity',
        'source_type', 'source_class', 'waterpoint_type_group'],
      dtype='object')
```

In [76]:

```
# Removing location based columns and unecessary columns

df_bivariate.drop(columns=['wpt_name','subvillage','lga','ward','scheme_name'], inplace=
True)
```

In [77]:

```
df_bivariate
```

Out[77]:

	id	status_group	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	basin	
0	69572	functional	6000.0	2011-03-14	Roman	1390	Roman	34.938093	-9.856322	Lake Nyasa	
1	8776	functional	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.698766	-2.147466	Lake Victoria	
2	34310	functional	25.0	2013-02-25	Lottery Club	686	World vision	37.460664	-3.821329	Pangani	
3	67743	non functional	0.0	2013-01-28	Unicef	263	UNICEF	38.486161	-11.155298	Ruvuma / Southern Coast	
4	19728	functional	0.0	2011-07-13	Action In A	0	Artisan	31.130847	-1.825359	Lake Victoria	
...	...	...	...	...	...	...	...	...	...	...	
59395	60739	functional	10.0	2013-05-03	Germany Republi	1210	CES	37.169807	-3.253847	Pangani	Ki
59396	27263	functional	4700.0	2011-05-07	Cefa-njombe	1212	Cefa	35.249991	-9.070629	Rufiji	
59397	37057	functional	0.0	2011-04-11	unknown	0	unknown	34.017087	-8.750434	Rufiji	
59398	31282	functional	0.0	2011-03-08	Malec	0	Musa	35.861315	-6.378573	Rufiji	
59399	26348	functional	0.0	2011-03-23	World Bank	191	World	38.104048	-6.747464	Wami / Ruvu	I

59364 rows × 26 columns

In [78]:

```
df_bivariate['date_recorded'].head()
```

Out[78]:

```
0    2011-03-14
1    2013-03-06
2    2013-02-25
3    2013-01-28
4    2011-07-13
Name: date_recorded, dtype: object
```

In [79]:

```
# Extracting year from date_recorded

df_bivariate['date_recorded_year'] = df_bivariate['date_recorded'].str.extract(r'(\d{4})
').astype(int)

df_bivariate.head()
```

Out[79]:

Out[79]:

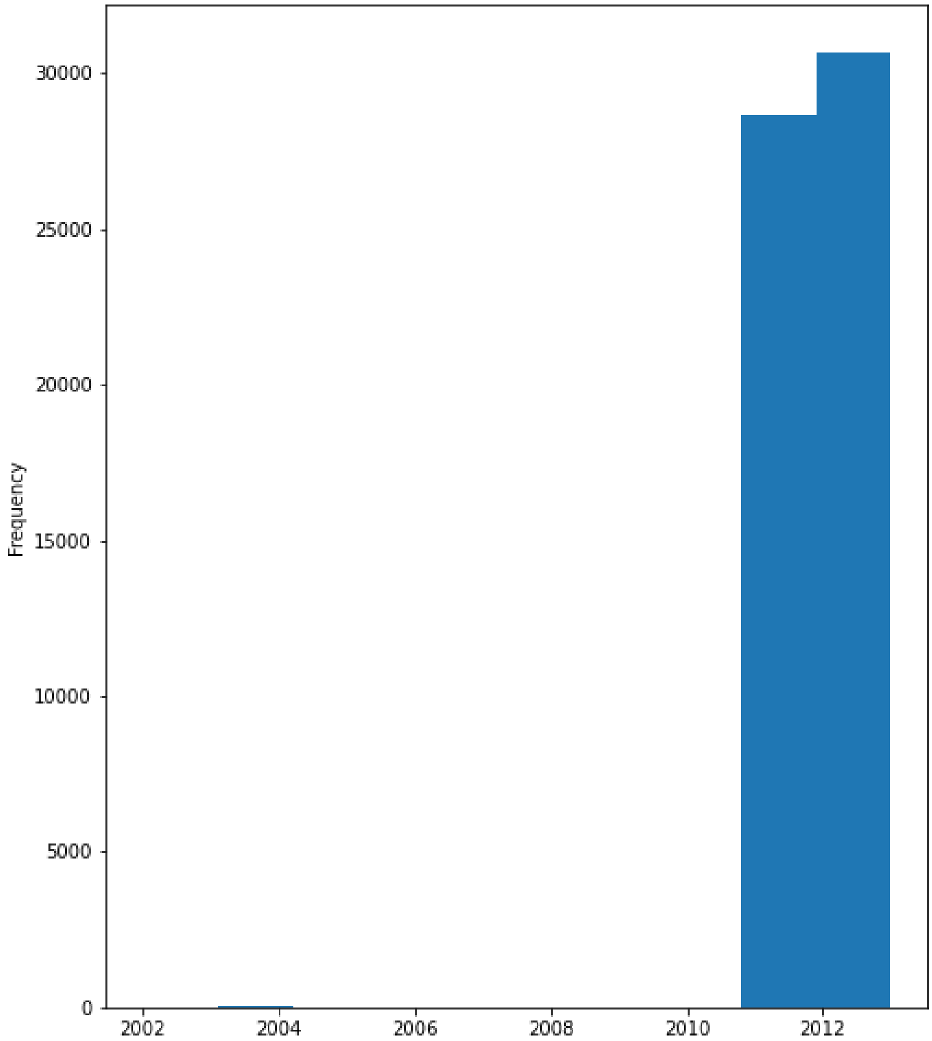
	id	status_group	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	basin	region
0	69572	functional	6000.0	2011-03-14	Roman	1390	Roman	34.938093	-9.856322	Lake Nyasa	Iringa
1	8776	functional	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.698766	-2.147466	Lake Victoria	Mara
2	34310	functional	25.0	2013-02-25	Lottery Club	686	World vision	37.460664	-3.821329	Pangani	Manyara
3	67743	non functional	0.0	2013-01-28	Unicef	263	UNICEF	38.486161	-11.155298	Ruvuma / Southern Coast	Mtwara
4	19728	functional	0.0	2011-07-13	Action In A	0	Artisan	31.130847	-1.825359	Lake Victoria	Kagera

In [80]:

```
df_bivariate['date_recorded_year'].plot(x="date_recorded_year", kind = 'hist',bins= 10,
figsize=(8,10))
```

Out[80]:

<AxesSubplot:ylabel='Frequency'>



In [81]:

```
# Lets subtract the recorded year from construction to discover how long the wells had op
erated before being recorded

df_bivariate['period']= df_bivariate['date_recorded_year'] - df_bivariate['construction_
year']
```

```
df_bivariate
```

Out[81]:

	id	status_group	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	basin	
0	69572	functional	6000.0	2011-03-14	Roman	1390	Roman	34.938093	-9.856322	Lake Nyasa	
1	8776	functional	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.698766	-2.147466	Lake Victoria	
2	34310	functional	25.0	2013-02-25	Lottery Club	686	World vision	37.460664	-3.821329	Pangani	
3	67743	non functional	0.0	2013-01-28	Unicef	263	UNICEF	38.486161	-11.155298	Ruvuma / Southern Coast	
4	19728	functional	0.0	2011-07-13	Action In A	0	Artisan	31.130847	-1.825359	Lake Victoria	
...	...	...	...	...	...	...	...	...	...	...	
59395	60739	functional	10.0	2013-05-03	Germany Republi	1210	CES	37.169807	-3.253847	Pangani	Ki
59396	27263	functional	4700.0	2011-05-07	Cefa-njombe	1212	Cefa	35.249991	-9.070629	Rufiji	
59397	37057	functional	0.0	2011-04-11	unknown	0	unknown	34.017087	-8.750434	Rufiji	
59398	31282	functional	0.0	2011-03-08	Malec	0	Musa	35.861315	-6.378573	Rufiji	
59399	26348	functional	0.0	2011-03-23	World Bank	191	World	38.104048	-6.747464	Wami / Ruvu	I

59364 rows × 28 columns

In [82]:

```
df_bivariate['period'].describe()
```

Out[82]:

```
count    59364.000000
mean        17.508524
std         12.592847
min         -7.000000
25%          6.000000
50%         15.000000
75%         28.000000
max         53.000000
Name: period, dtype: float64
```

In [83]:

```
#lets drop rows that have period with negative values
df_bivariate = df_bivariate[df_bivariate['period'] >= 0]
```

In [84]:

```
# lets drop the date recorded column and date_recorded_year

df_bivariate.drop(columns= ['date_recorded','date_recorded_year'], inplace = True)
```

```
c:\Users\USER\anaconda3\envs\learn-env\lib\site-packages\pandas\core\frame.py:4163: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

In [85]:

```
df_bivariate.columns
```

Out[85]:

```
Index(['id', 'status_group', 'amount_tsh', 'funder', 'gps_height', 'installer',
       'longitude', 'latitude', 'basin', 'region', 'region_code',
       'district_code', 'population', 'public_meeting', 'scheme_management',
       'permit', 'construction_year', 'extraction_type_class',
       'management_group', 'payment_type', 'quality_group', 'quantity',
       'source_type', 'source_class', 'waterpoint_type_group', 'period'],
      dtype='object')
```

Chi Square Test

In [86]:

```
#Define a function to perform a chi square tes
def chi_square_test(df, target, columns):
    results = {}
    for col in columns:
        contingency_table = pd.crosstab(df[target], df[col])
        chi2,p,dof,expected = chi2_contingency(contingency_table)
        results[col] = {'chi2':chi2,
                        'p-value': p}

    return results
```

In [87]:

```
# select target variable

target_variable = 'status_group'
# get thecategorical columns in a list
categorical_columns =df_bivariate.select_dtypes(include='object').columns.to_list()
#remove the status_group
categorical_columns.pop(0)

results = chi_square_test(df= df_bivariate, target= target_variable, columns = categoric
al_columns)
for col, result in results.items():
    print(f"Column: {col}")
    print(f"Chi2: {result['chi2']}, p-value: {result['p-value']}\n")
```

```
Column: funder
Chi2: 14171.889173609099, p-value: 0.0

Column: installer
Chi2: 14750.703708213303, p-value: 0.0

Column: basin
Chi2: 1923.5586810402722, p-value: 0.0

Column: region
Chi2: 4794.299507665105, p-value: 0.0

Column: scheme_management
Chi2: 1987.6020257769587, p-value: 0.0

Column: extraction_type_class
Chi2: 6921.90911467295, p-value: 0.0

Column: management_group
Chi2: 286.0752588490578, p-value: 3.774735857120648e-57

Column: payment_type
Chi2: 3966.0686916486497, p-value: 0.0

Column: quality_group
Chi2: 2094.1353027502073, p-value: 0.0
```

```
Column: quantity
Chi2: 11351.076397724744, p-value: 0.0

Column: source_type
Chi2: 1906.2723413088465, p-value: 0.0

Column: source_class
Chi2: 589.6893504275961, p-value: 2.6402697082320118e-126

Column: waterpoint_type_group
Chi2: 6106.112717515814, p-value: 0.0
```

- **Ho : categories in the column does not affect if a well will be functional or not**
- **H1 : categories in a column affects whether a well will be functional or not**
- **alpha : 0.05**
- **From the above the columns management_group and source_class have a pvalue greater than 0.5 this shows that the categories in the column makes us to accept the null hypothesis.We will drop those columns as they wont provide any relevant information to our models**
- **All other columns have a pvalue that is less than our alpha therefore we accept the alternate hypothesis**

In [88]:

```python
df_bivariate.drop(columns= ['management_group','source_class'], inplace = True)
```

```
c:\Users\USER\anaconda3\envs\learn-env\lib\site-packages\pandas\core\frame.py:4163: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

Numerical columns

Grouped by and summary statistics

In [89]:

```python
# Select numerical columns
numerical_columns = df_bivariate.select_dtypes(include=['number']).columns


summary_stats = df_bivariate.groupby('status_group')[numerical_columns]

summary_stats.describe()
```

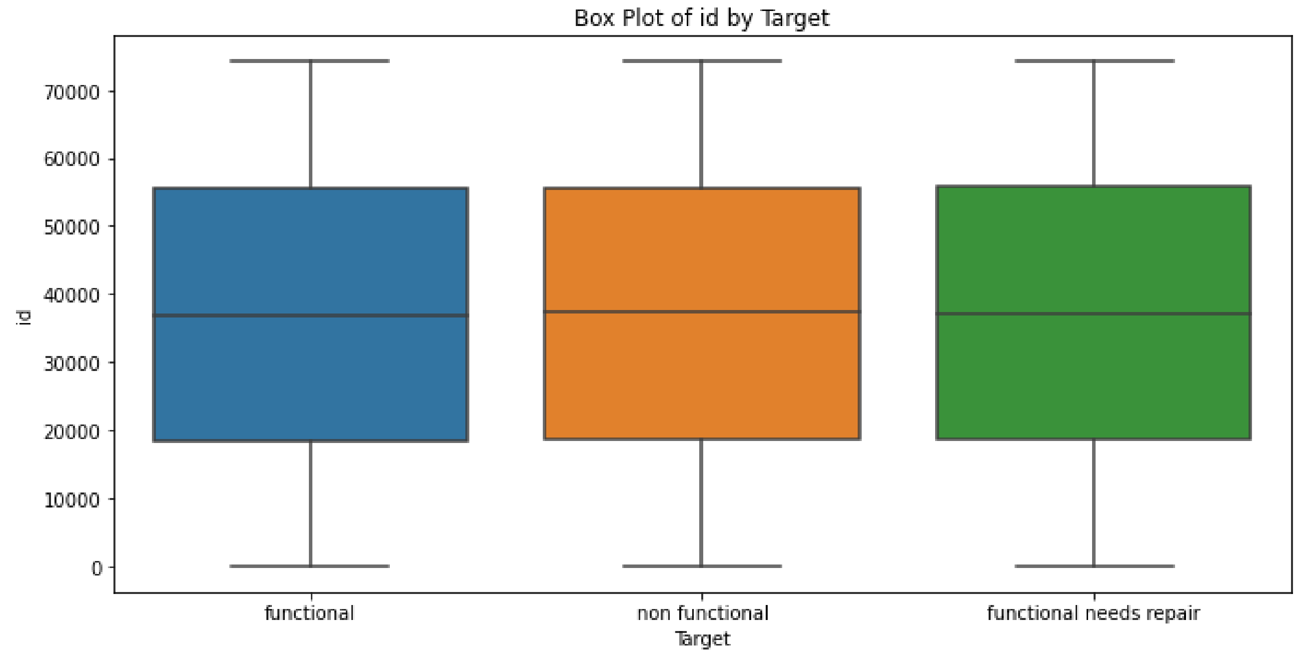
Out[89]:

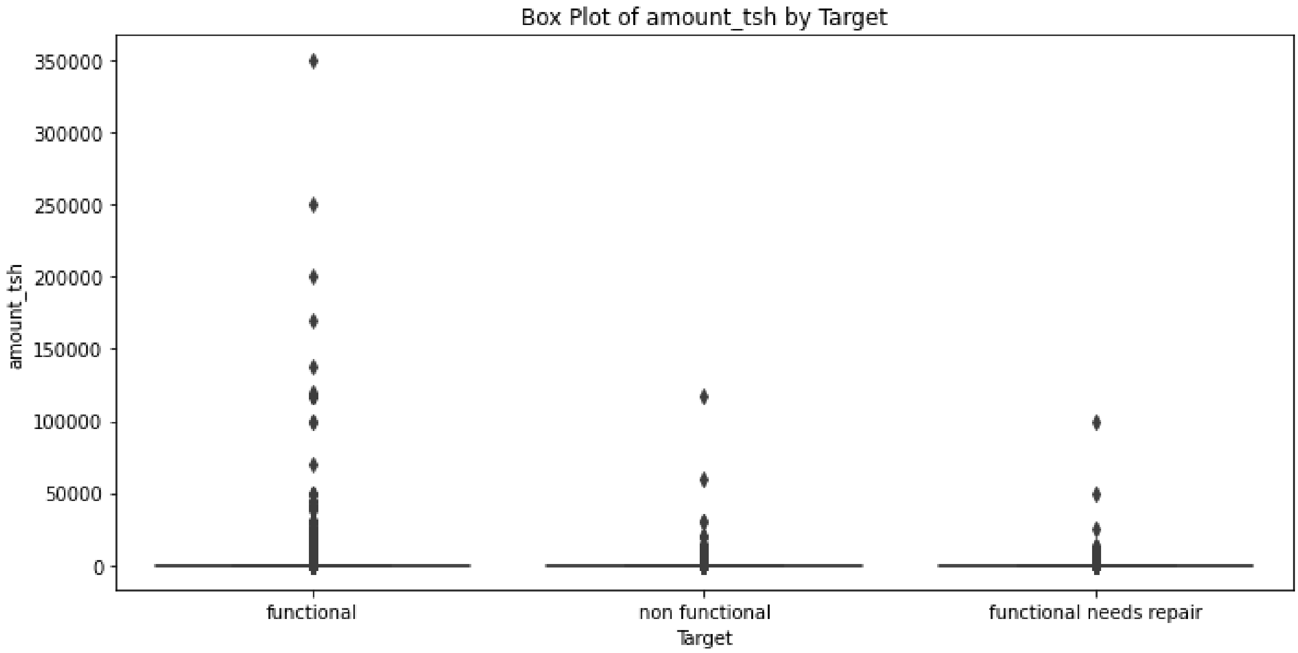
	id								amount_tsh		
	count	mean	std	min	25%	50%	75%	max	count	mean	std
status_group											
functional	32233.0	37039.804269	21488.220554	1.0	18329.00	36890.0	55683.00	74242.0	32233.0	462.150568	3891.283
functional needs repair	4314.0	37159.558646	21343.151470	20.0	18720.75	37196.0	55713.50	74233.0	4314.0	267.257302	1925.682
non functional	22806.0	37224.724678	21421.266125	0.0	18777.25	37295.0	55605.75	74247.0	22806.0	123.577813	1110.553

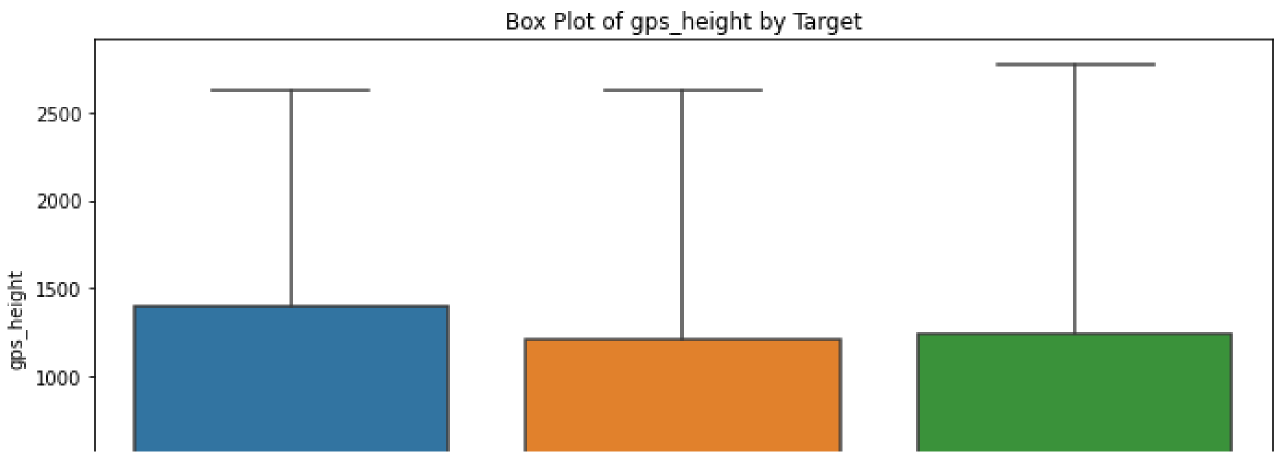
Boxplots

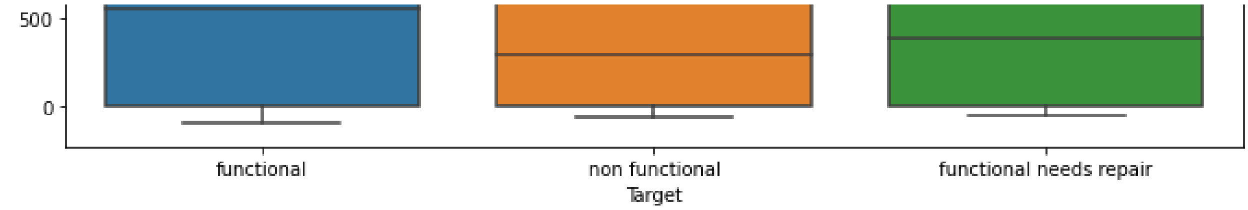
In [90]:

```
# Plot box plots for numerical columns
fig, axes = plt.subplots(nrows=len(numerical_columns), ncols=1, figsize=(10, 5 * len(num
erical_columns)))
for i, col in enumerate(numerical_columns):
    sns.boxplot(x='status_group', y=col, data=df_bivariate, ax=axes[i])
    axes[i].set_title(f'Box Plot of {col} by Target')
    axes[i].set_xlabel('Target')
    axes[i].set_ylabel(col)

plt.tight_layout()
plt.show()
```

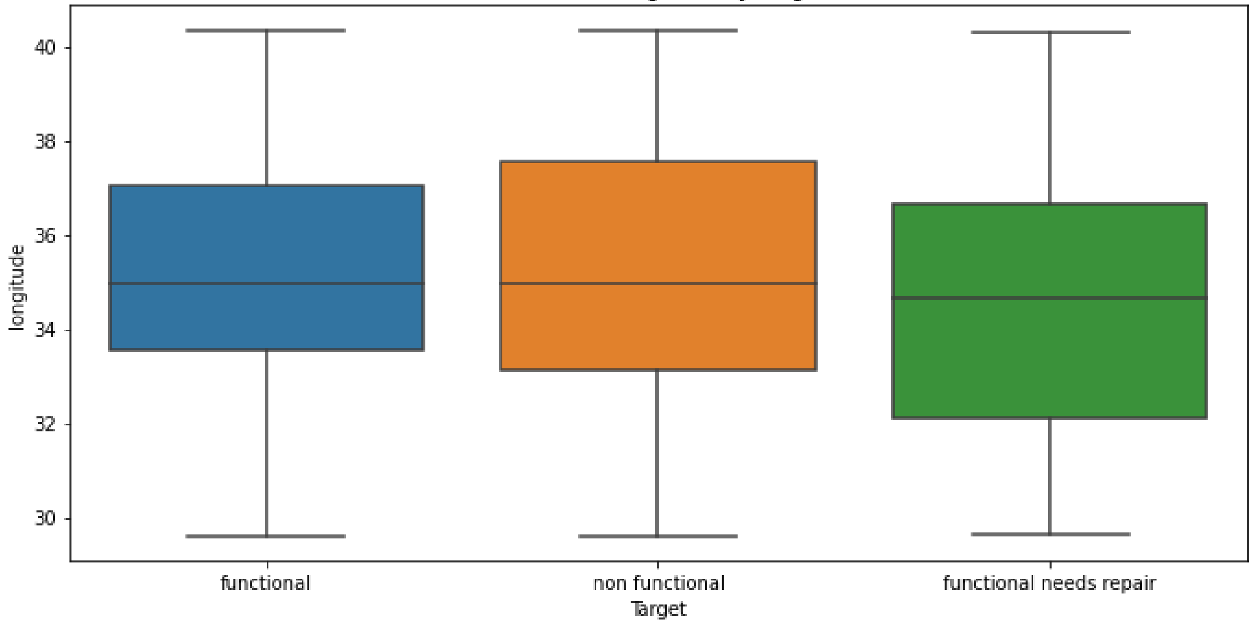




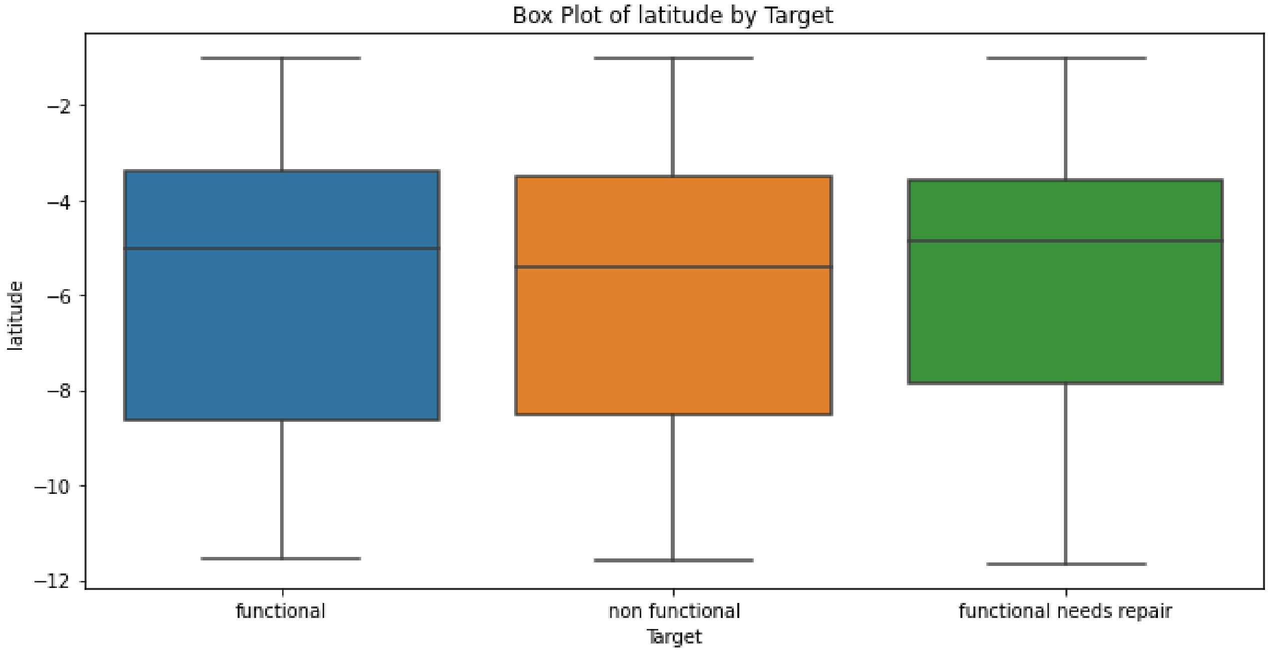




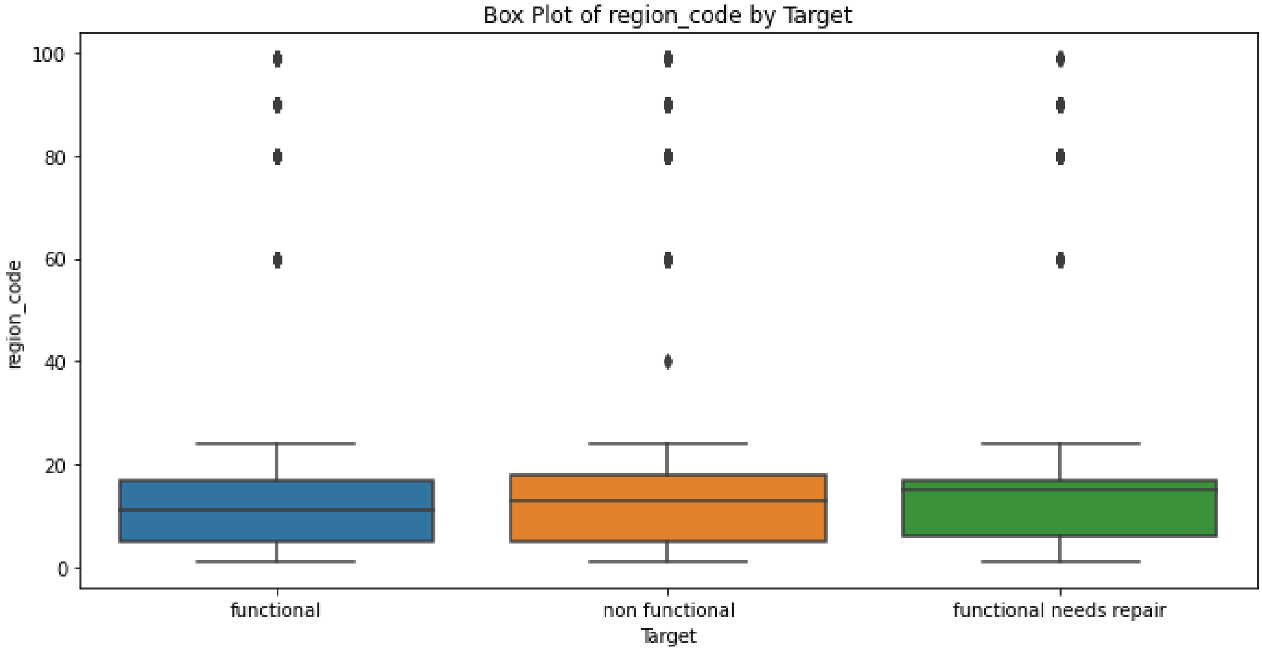
Box Plot of longitude by Target

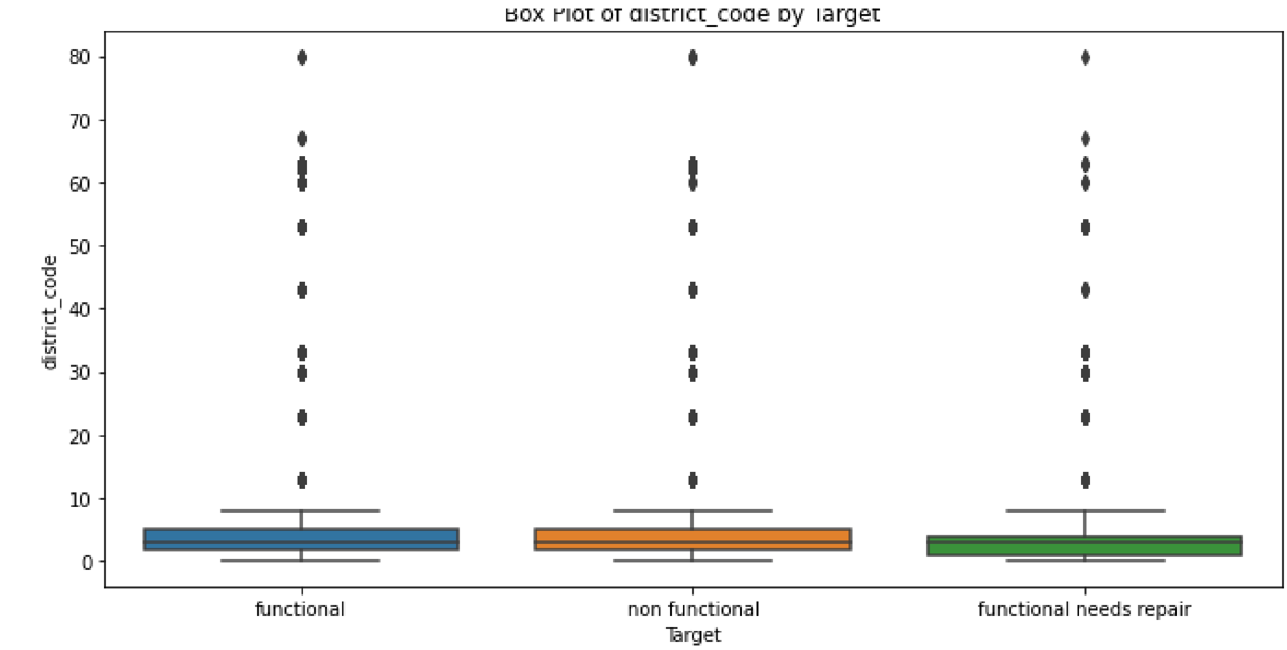


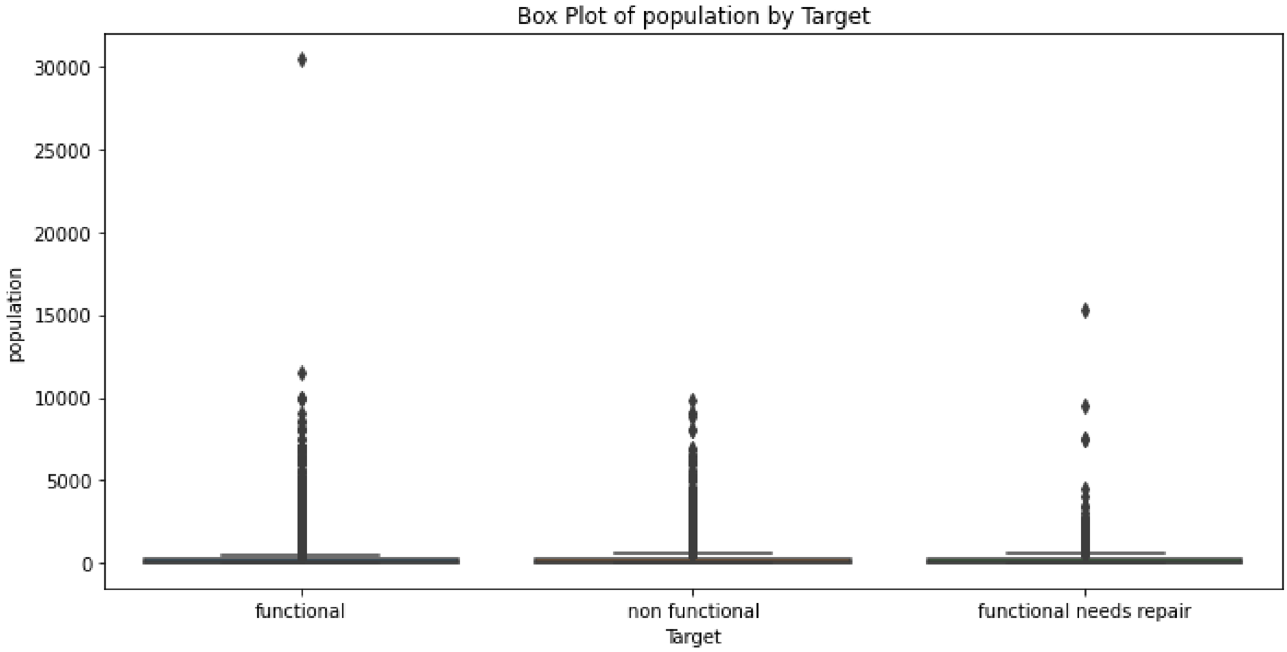
Box Plot of latitude by Target

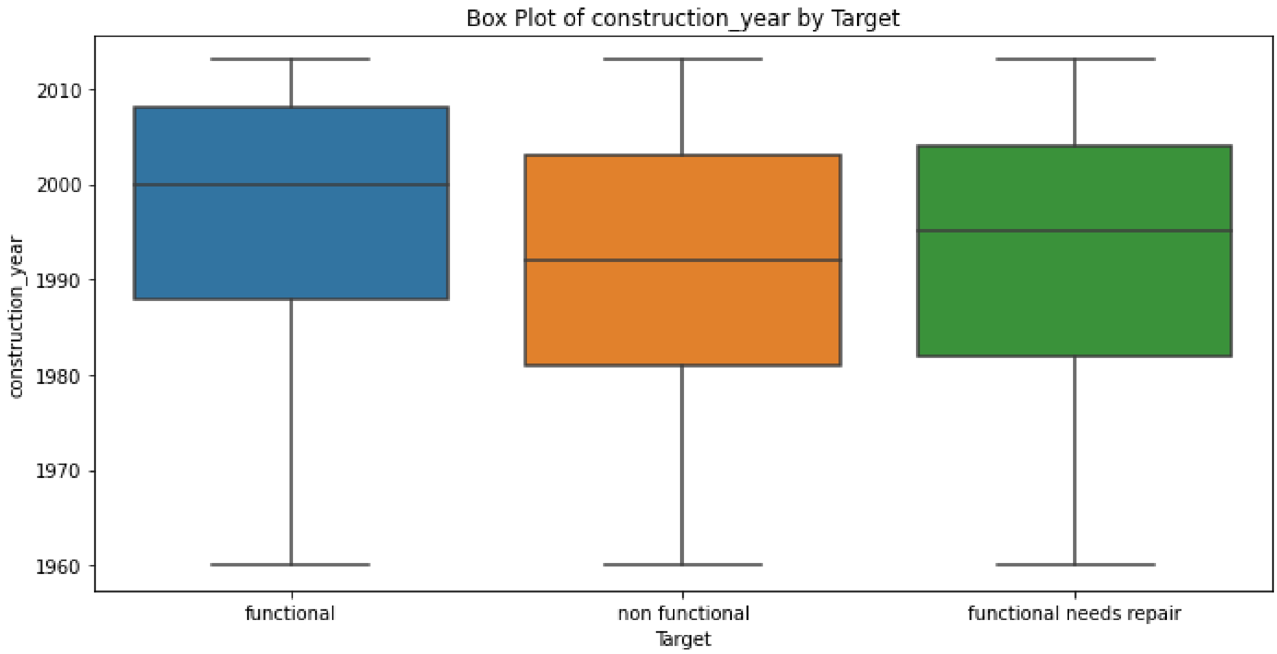


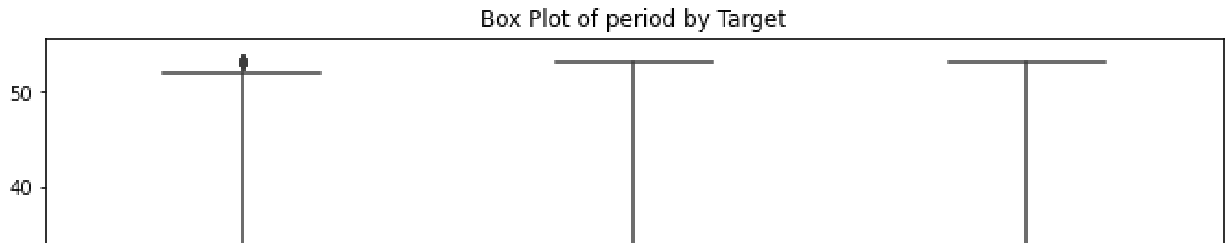
Box Plot of region_code by Target

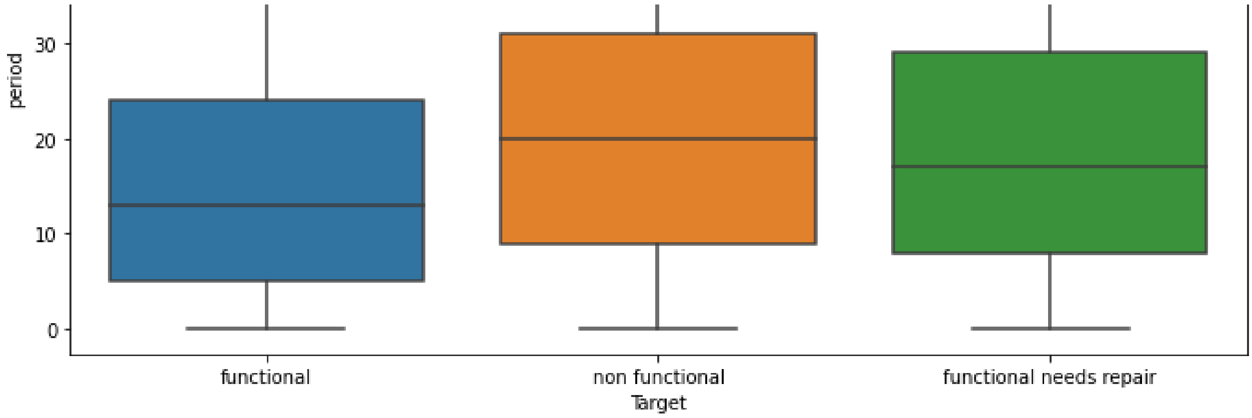



Box Plot of district_code by Target
district_code
0
10
20
30
40
50
60
70
80
functional
non functional
functional needs repair
Target


Box Plot of population by Target
population
0
5000
10000
15000
20000
25000
30000
functional
non functional
functional needs repair
Target


Box Plot of construction_year by Target
construction_year
1960
1970
1980
1990
2000
2010
functional
non functional
functional needs repair
Target


Box Plot of period by Target
40
50



insights

- **amount_tsh - the columns seems to have a lot of outliers. It will be advisable to drop the column because of the high number of outliers**
- **gps_height - wells that functional seem to be found at higher altitudes than the wells that are considered non functional**
- **population - the column has a lot of outliers but functional wells tend to have higher population**
- **construction year - functional columns tend to be built newer than non functional ones**
- **period - functional wells tend to be younger though there is presence of an outlier on the dataset**

In [91]:

```
# dropping unecessary columns

df_bivariate.drop(columns=["id","amount_tsh","district_code"],inplace = True)
```

```
c:\Users\USER\anaconda3\envs\learn-env\lib\site-packages\pandas\core\frame.py:4163: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

In [92]:

```
# checking the categories counts of region and the region code match
len(df_bivariate['region'].value_counts()) == len(df_bivariate['region_code'].value_count
s())
```

Out[92]:

```
False
```

In [93]:

```
region_code = df_bivariate.value_counts("region_code")
region = df_bivariate.value_counts("region")

region_code, region
```

Out[93]:

```
(region_code
 11     5299
 17     4989
 12     4639
 3      4379
 5      4040
 18     3323
 19     3033
 2      3024
 16     2816
 10     2640
 4      2510
```

```
1     2201
13    2093
14    1979
20    1968
15    1807
6     1608
21    1583
80    1238
60    1023
90     916
7      805
99     423
9      390
24     326
8      300
40       1
dtype: int64,
region
Iringa           5293
Shinyanga        4980
Mbeya            4639
Kilimanjaro      4379
Morogoro         4006
Arusha           3350
Kagera           3315
Mwanza           3068
Kigoma           2816
Ruvuma           2640
Pwani            2632
Tanga            2544
Dodoma           2201
Singida          2093
Mara             1968
Tabora           1959
Rukwa            1807
Mtwara           1729
Manyara          1583
Lindi            1546
Dar es Salaam     805
dtype: int64)
```

In [94]:

```
# From an onlline search Tanzania has 33 regions therefore this means that region code ha
s eroneous data. therefore we will drop the column

df_bivariate.drop(columns="region_code",inplace = True)
```

In [95]:

```
df_multivariate = df_bivariate.copy()
```

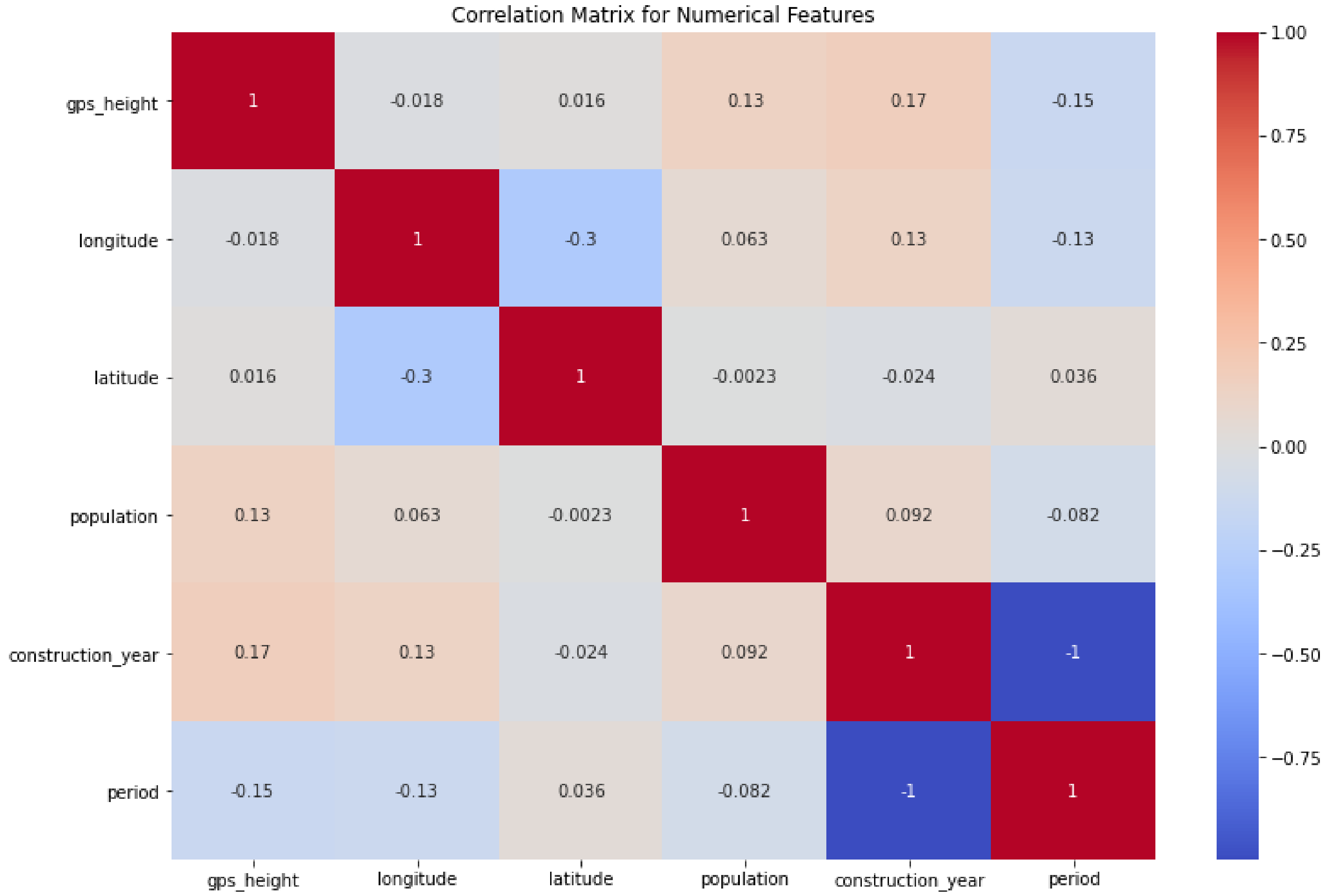
Multivariate analysis

Correlation matrix

In [96]:

```
numerical_columns = df_multivariate.select_dtypes(include=['number'])


# correlation matrix for numerical features

corr_matrix = numerical_columns.corr()

# Display heatmap of correlations
plt.figure(figsize=(13, 9))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
```

```
plt.title('Correlation Matrix for Numerical Features')
plt.show()
```



	gps_height	longitude	latitude	population	construction_year	period
gps_height	1	-0.018	0.016	0.13	0.17	-0.15
longitude	-0.018	1	-0.3	0.063	0.13	-0.13
latitude	0.016	-0.3	1	-0.0023	-0.024	0.036
population	0.13	0.063	-0.0023	1	0.092	-0.082
construction_year	0.17	0.13	-0.024	0.092	1	-1
period	-0.15	-0.13	0.036	-0.082	-1	1

Insights

The coorreltion analysis shows there is no presence of a strong correlation between any of the numerical X variables. This is true except for the period in which we feature engineer as an additional information for our model

In [97]:

```
df_model = df_multivariate.copy()
```

Modelling

- **Target variable = status group**

This is a classification problem. The aim of this project in accordance with our business problem is to predict whether given certain features if we can predict whether a well will be functional or not functional

feature engineering

In [98]:

```
df_model.shape
```

Out[98]:

```
(59353, 20)
```

In [99]:

```
df_model.info()
```

```
&lt;class 'pandas.core.frame.DataFrame'&gt;
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59353 entries, 0 to 59399
Data columns (total 20 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   status_group           59353 non-null  object 
 1   funder                 59353 non-null  object 
 2   gps_height             59353 non-null  int64  
 3   installer              59353 non-null  object 
 4   longitude              59353 non-null  float64
 5   latitude               59353 non-null  float64
 6   basin                  59353 non-null  object 
 7   region                 59353 non-null  object 
 8   population             59353 non-null  int64  
 9   public_meeting         59353 non-null  bool   
 10  scheme_management      59353 non-null  object 
 11  permit                 59353 non-null  bool   
 12  construction_year      59353 non-null  int64  
 13  extraction_type_class  59353 non-null  object 
 14  payment_type           59353 non-null  object 
 15  quality_group          59353 non-null  object 
 16  quantity               59353 non-null  object 
 17  source_type            59353 non-null  object 
 18  waterpoint_type_group  59353 non-null  object 
 19  period                 59353 non-null  int64  
dtypes: bool(2), float64(2), int64(4), object(12)
memory usage: 11.2+ MB
```

target variable

In [100]:

```
target_variable = df_model.value_counts("status_group")
target_variable
```

Out[100]:

```
status_group
functional                 32233
non functional             22806
functional needs repair     4314
dtype: int64
```

A logistic model is a binary model so we have to engineer our class into a binary classification i.e

- **both functional and functional and need repairs will be a 1**
- **non functional will be a 0**

In [101]:

```
df_model["status_group"] = df_model['status_group'].map({"functional": 1, "functional needs repair": 1, "non functional": 0})
```

In [102]:

```
target_variable = df_model.value_counts("status_group")
target_variable
```

Out[102]:

```
status_group
1    36547
0    22806
dtype: int64
```

Independent variables

funder

In [103]:

```
# lets check number of unique values that are in each of our catehorical columns.

object_columns = df_model.select_dtypes(include=['object']).columns

object_column_unique = {col : df_model[col].nunique() for col in object_columns}

columns_unique_df =pd.DataFrame(list(object_column_unique.items()), columns= ['column_name', 'number_of_unique_values'])
columns_unique_df
```

Out[103]:

	column_name	number_of_unique_values
0	funder	1897
1	installer	2143
2	basin	9
3	region	21
4	scheme_management	13
5	extraction_type_class	7
6	payment_type	7
7	quality_group	6
8	quantity	5
9	source_type	7
10	waterpoint_type_group	6

From the above we can see that funder and installer columns have a lot of unique values. To aid in our model performance we need to reduce its dimensionality

In [104]:

```
# We will filter our dataset to contain funders who hav funded over 500 projects
funder =df_model['funder'].value_counts()

funder_filtered_index = funder[funder> 500].index
```

In [105]:

```
#lets filter our dataframe with our filtered index
funder_filtered_columns =funder_filtered_index.to_list()

df_model_filtered = df_model[df_model['funder'].isin(funder_filtered_columns)].reset_index()
```

In [106]:

```
object_columns = df_model_filtered.select_dtypes(include=['object']).columns

object_column_unique = {col : df_model_filtered[col].nunique() for col in object_columns}

columns_unique_df =pd.DataFrame(list(object_column_unique.items()), columns= ['column_name', 'number_of_unique_values'])
columns_unique_df
```

Out[106]:

	column_name	number_of_unique_values
0	funder	20
1	installer	546

	column_name	number_of_unique_values
1	installer	546
2	basin	9
3	region	21
4	scheme_management	13
5	extraction_type_class	7
6	payment_type	7
7	quality_group	6
8	quantity	5
9	source_type	7
10	waterpoint_type_group	6

In [107]:

```
df_model_filtered.drop(columns="index", inplace= True)
```

In [108]:

```
df_model_filtered.shape
```

Out[108]:

```
(32433, 20)
```

installer

In [109]:

```
installer = df_model_filtered["installer"].value_counts()
installer
```

Out[109]:

```
DWE                 10375
unknown              3618
Government           1722
DANIDA               1047
RWE                   995
                    ...
Word bank               1
KOYI                    1
Jaica                   1
VITECOS INVEST          1
ICF/TWESA               1
Name: installer, Length: 546, dtype: int64
```

The installer dataset has a lot of inconsistent data. We will create a dictionary so to have a more consistent data on that column

In [110]:

```
# Create a function to remove any special character found in a column

def remove_special_characters(text):
    return re.sub(r'[^A-Za-z0-9\s]', '', text)
```

In [111]:

```
df_model_filtered["installer"] = df_model_filtered["installer"].apply(remove_special_cha
racters)
df_model_filtered["installer"] = df_model_filtered["installer"].str.capitalize()
```

In [112]:

```
installer = df_model_filtered["installer"].value_counts()
```

```
installer
```

Out[112]:

```
Dwe                    10389
Unknown                 3618
Government              1757
Danida                  1047
Rwe                      995
                        ...
Filber                     1
Hamis makombo              1
Kisiriri adp               1
Spar drilling              1
Masele nzengula            1
Name: installer, Length: 482, dtype: int64
```

In [113]:

```
installer =df_model_filtered["installer"].value_counts()
installer
```

Out[113]:

```
Dwe                    10389
Unknown                 3618
Government              1757
Danida                  1047
Rwe                      995
                        ...
Filber                     1
Hamis makombo              1
Kisiriri adp               1
Spar drilling              1
Masele nzengula            1
Name: installer, Length: 482, dtype: int64
```

In [114]:

```
#we will create a dictionary to correct spelling error that might were caused in data col
lection for the main installers
replacements = {
    "Dwe": "District Water Engineer","Government": "Central Government","Rwe":"Region Wa
ter Engineer","District council":"District Water Engineer",
    "0":"Unknown","Central government":"Central Government","Commu":"Community","Danid":
"Danida","Gover":"Central Government","Centr":"Central Government","Idara ya maji":"Centr
al Government","District water department":"District Water Engineer","Gove":"Central Gove
rnment",
    "Central govt":"Central Government","Unisef":"UNICEF","Unicef":"UNICEF","Wizara ya ma
ji":"Central Government","Region water department":"Region Water Engineer","Sengerema wat
er department":"District Water Engineer","Distri":"District Water Engineer","Kkt":"KKKT",
"Kkkt":"KKKT","Tanzania government":"Central Government","Govern":"Central Government","M
inistry of water":"Central Government","Would bank":"World Bank","World bank":"World Ban
k","District counci":"District Water Engineer",
    "RWe community":"Region Water Engineer","World vission":"World Vision","Ministry of w
ater enginneer":"Central Government","Regional water":"Regional Water Engineer","Not know
n":"Unknown","Cental government":"Central Government","Cebtral governemnt":"Central Gover
nment","Tanzanian governemt":"Central Government","District water depar":"District Water
Engineer", "Local":"Community",
    "World vision":"World Vision","Rwe community":"Region Water Engineer","Ministry of wa
ter engineer":"Central Government","Regional Water Engineer":"Region Water Engineer","Ceb
tral government":"Central Government","Kkt c":"KKKT","World":"World Bank","Villagers":"Co
mmunity","Dwsp":"District Water Engineer","Rwssp":"Region Water Engineer"

}
```

In [115]:

```
df_model_filtered["installer"] = df_model_filtered["installer"].replace(replacements)
```

In [116]:

```
# Get value counts
value_counts = df_model_filtered['installer'].value_counts()
# Identify categories with counts less than 100
categories_to_replace = value_counts[value_counts < 100].index
# Replace these categories with 'Other'
df_model_filtered['installer'] = df_model_filtered['installer'].apply(lambda x: 'Other'
if x in categories_to_replace else x)
```

In [117]:

```
installer =df_model_filtered["installer"].value_counts()
installer
```

Out[117]:

```
District Water Engineer    11769
Unknown                     4404
Central Government          3766
Other                       2205
Danida                      1670
Region Water Engineer       1235
Community                   1080
KKKT                         945
Hesawa                       913
World Vision                 692
Tcrs                         658
Ces                          610
Tasaf                        412
Norad                        365
UNICEF                       330
Wedeco                       309
World Bank                   289
Da                           281
Wu                           167
Dmdd                         121
Handeni trunk main           111
Consulting engineer          101
Name: installer, dtype: int64
```

In [118]:

```
object_columns = df_model_filtered.select_dtypes(include=['object']).columns

object_column_unique = {col : df_model_filtered[col].nunique() for col in object_columns
}

columns_unique_df =pd.DataFrame(list(object_column_unique.items()), columns= ['column_na
me', 'number_of_unique_values'])
columns_unique_df
```

Out[118]:

	column_name	number_of_unique_values
0	funder	20
1	installer	22
2	basin	9
3	region	21
4	scheme_management	13
5	extraction_type_class	7
6	payment_type	7
7	quality_group	6
8	quantity	5
9	source_type	7
10	waterpoint_type_group	6

In [119]:

```
df_model_filtered.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32433 entries, 0 to 32432
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   status_group            32433 non-null  int64  
 1   funder                  32433 non-null  object 
 2   gps_height              32433 non-null  int64  
 3   installer               32433 non-null  object 
 4   longitude               32433 non-null  float64
 5   latitude                32433 non-null  float64
 6   basin                   32433 non-null  object 
 7   region                  32433 non-null  object 
 8   population              32433 non-null  int64  
 9   public_meeting          32433 non-null  bool   
 10  scheme_management       32433 non-null  object 
 11  permit                  32433 non-null  bool   
 12  construction_year       32433 non-null  int64  
 13  extraction_type_class   32433 non-null  object 
 14  payment_type            32433 non-null  object 
 15  quality_group           32433 non-null  object 
 16  quantity                32433 non-null  object 
 17  source_type             32433 non-null  object 
 18  waterpoint_type_group   32433 non-null  object 
 19  period                  32433 non-null  int64  
dtypes: bool(2), float64(2), int64(5), object(11)
memory usage: 4.5+ MB
```

In [120]:

```
# Lets change construction year to a discrete variable. To allow for better interpration.
 # creating new columns
df_model_filtered['decade'] = df_model_filtered['construction_year']
# Create a dictionary to map years to decades
decade_mapping = {
**dict.fromkeys(range(1960, 1970), '60s'),
**dict.fromkeys(range(1970, 1980), '70s'),
**dict.fromkeys(range(1980, 1990), '80s'),
**dict.fromkeys(range(1990, 2000), '90s'),
**dict.fromkeys(range(2000, 2010), '00s'),
**dict.fromkeys(range(2010, 2014), '10s')
}
# Apply the mapping to the 'decade' column
df_model_filtered['decade'] = df_model_filtered['decade'].map(decade_mapping)
df_model_filtered['decade'].value_counts()
```

Out[120]:

```
00s    8610
90s    7826
70s    6710
80s    6489
10s    2357
60s     441
Name: decade, dtype: int64
```

In [121]:

```
# lets drop construction year
# We will also drop longitudes and latitudes for its complexity

df_model_filtered.drop(columns= ["longitude","latitude","construction_year"], inplace= T
rue)
df_model_filtered.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32433 entries, 0 to 32432
Data columns (total 18 columns):
 #   Column                  Non-Null Count  Dtype
```

```
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   status_group           32433 non-null  int64
 1   funder                 32433 non-null  object
 2   gps_height             32433 non-null  int64
 3   installer              32433 non-null  object
 4   basin                  32433 non-null  object
 5   region                 32433 non-null  object
 6   population             32433 non-null  int64
 7   public_meeting         32433 non-null  bool
 8   scheme_management      32433 non-null  object
 9   permit                 32433 non-null  bool
 10  extraction_type_class  32433 non-null  object
 11  payment_type           32433 non-null  object
 12  quality_group          32433 non-null  object
 13  quantity               32433 non-null  object
 14  source_type            32433 non-null  object
 15  waterpoint_type_group  32433 non-null  object
 16  period                 32433 non-null  int64
 17  decade                 32433 non-null  object
dtypes: bool(2), int64(4), object(12)
memory usage: 4.0+ MB
```

In [122]:

```
#handling boolean values

boolean_columns = df_model_filtered.select_dtypes(include="bool").columns.to_list()
df_model_filtered[boolean_columns]= df_model_filtered[boolean_columns].astype(int)
```

In [123]:

```
df_model_filtered.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32433 entries, 0 to 32432
Data columns (total 18 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   status_group           32433 non-null  int64
 1   funder                 32433 non-null  object
 2   gps_height             32433 non-null  int64
 3   installer              32433 non-null  object
 4   basin                  32433 non-null  object
 5   region                 32433 non-null  object
 6   population             32433 non-null  int64
 7   public_meeting         32433 non-null  int32
 8   scheme_management      32433 non-null  object
 9   permit                 32433 non-null  int32
 10  extraction_type_class  32433 non-null  object
 11  payment_type           32433 non-null  object
 12  quality_group          32433 non-null  object
 13  quantity               32433 non-null  object
 14  source_type            32433 non-null  object
 15  waterpoint_type_group  32433 non-null  object
 16  period                 32433 non-null  int64
 17  decade                 32433 non-null  object
dtypes: int32(2), int64(4), object(12)
memory usage: 4.2+ MB
```

In [124]:

```
#Exporting to tableau

#df_model_filtered.to_csv("data/clean_data.csv")
```

Models

Logistic Regression

In [125]:

```
df_mdl = df_model_filtered.copy()
```

In [126]:

```
#Assigning target variables and independent variables

y = df_mdl["status_group"]
X= df_mdl.drop(columns="status_group")
```

In [127]:

```
# Splitting the data into train and test data
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size= 0.2, random_state= 4
2)
```

In [128]:

```
# Lets create a dataframe to store our results

df_results = pd.DataFrame(columns=["Model","Scaler","Encoder","Mean F1 Score","roc_auc sc
ore mean","roc_auc score std"])
```

In [129]:

```
categorical_columns= ['funder',
 'installer',
 'basin',
 'region',
 'scheme_management',
 'extraction_type_class',
 'payment_type',
 'quality_group',
 'quantity',
 'source_type',
 'waterpoint_type_group',
 'decade',
 'public_meeting',
 'permit']
```

In [130]:

```
numerical_columns = ['gps_height','population','period']
```

Standard Scaler , One hot enconder logistic regression

In [131]:

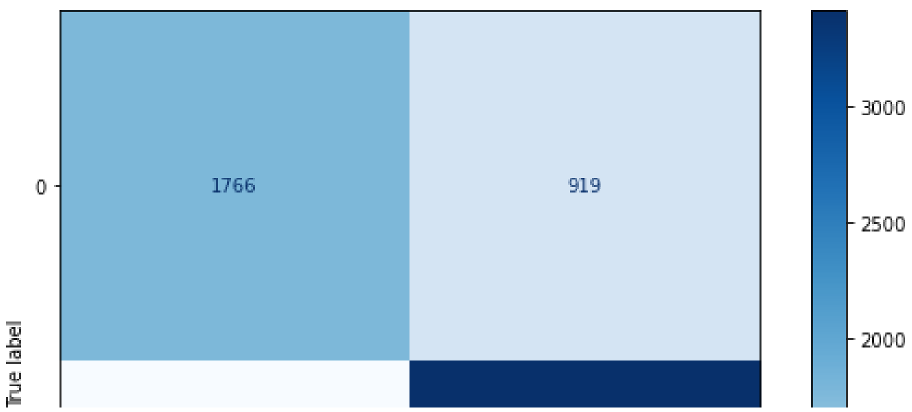
```
scaler = StandardScaler(with_std=True)
encoder = OneHotEncoder(sparse = False,handle_unknown="ignore")

#create pipeline for categorical transformation

cat_transformer = make_pipeline(encoder)
num_transformer = make_pipeline(scaler)

#create a preprocessor

preprocessor = ColumnTransformer(
    transformers= [
        ('num', num_transformer, numerical_columns),
        ('cat',cat_transformer,categorical_columns)
    ]
)
#Fit the preprocessor on the training data
preprocessor.fit(X_train)
```

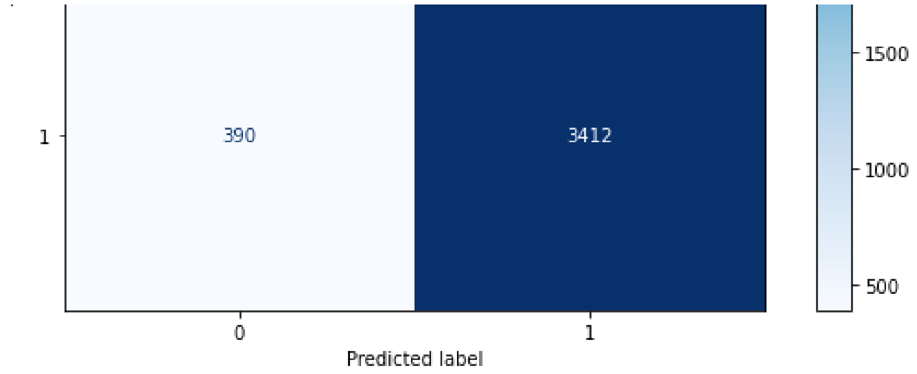
```
logisticreg = LogisticRegression(solver = 'lbfgs',random_state= 42,max_iter=1000)

#Creating the full pipeline with preprocessing and model
pipe = make_pipeline(preprocessor, logisticreg)


# Fit thepipeline to the training data
pipe.fit(X_train, y_train)

#Make predictions on the training set

y_pred_train = pipe.predict(X_train)

# Make predictions on test set

y_pred_test = pipe.predict(X_test)

#print accuracy results

print("Accuracy")
print("="*len("Accuracy"))
print(f"TRAIN:{accuracy_score(y_train, y_pred_train)}")
print(f"TEST: {accuracy_score(y_test, y_pred_test)}")
print()

#print balanced accuracy results

print("Balanced Results")
print("="*len("Balanced Results"))
print(f"TRAIN:{balanced_accuracy_score(y_train,y_pred_train)}")
print(f"TEST:{balanced_accuracy_score(y_test,y_pred_test)}")
```

```
Accuracy
========
TRAIN:0.791836892006475
TEST: 0.7982118082318483

Balanced Results
================
TRAIN:0.7718956631233738
TEST:0.7775752642194591
```

In [132]:

```
# Computing the confusion matrix

cm = confusion_matrix(y_test, y_pred_test)

# plotting the confusion matrix

disp = ConfusionMatrixDisplay(confusion_matrix=cm)

# Increase the size of the display
fig, ax = plt.subplots(figsize=(10, 7))
disp.plot(cmap=plt.cm.Blues,ax=ax)
# Show the plot plt.show()

plt.show()
```





Insights

From the confusion matrix we get the following insights

- **True Positives = 3411 functional wells were correctly predicted as non functional**
- **False Positives = 920 non functional wells were predicted functional while they were not**
- **False Negatives = 391 functional wells were predicted as non functional while they were functional**
- **True Negatives = 1765 non functional were correctly predicted as non functional wells**

In [133]:

```
f1 = f1_score(y_true=y_test, y_pred= y_pred_test)
print(f"F1_SCORE : {f1}")
```

```
F1_SCORE : 0.8390507807697036
```

This shows that model is doing very well and that both precision metric and recall metrics are high

We will calculate the ROC AUC score using Logistic Regression with cross validation to compare with other models.We will use the mean and standard of the scores for better understanding . We will use a cv = 5 to obtain 5 different results for each trial and use their mean . This approac provides more accurate results compared to a single train-test split

In [134]:

```
# Calculate cross-validated ROC AUC scores
scores = cross_val_score(pipe, X, y, cv=5, scoring='roc_auc')
roc_auc_score_mean = round(scores.mean(),4)
roc_auc_score_std = round(scores.std(),4)

# Print the mean and standard deviation of the scores
print(f"Mean ROC AUC: {roc_auc_score_mean} +/- {roc_auc_score_std}")
```

```
Mean ROC AUC: 0.8474 +/- 0.0029
```

In [135]:

```
# Calculate cross-validated ROC AUC scores
scores = cross_val_score(pipe, X, y, cv=5, scoring='f1')
f1_score_mean = round(scores.mean(),4)


# Print the mean and standard deviation of the scores
print(f"Mean f1 score: {f1_score_mean}")
```

```
Mean f1 score: 0.8336
```

We achieved better result with cross-validation compared to a simple train test split indicating strong baseline performance. The standard deviation is also low, reinforcing the reliability of our results

In [136]:

```
#Posting our results
```

```
new_results = pd.DataFrame([{
    "Model": "LogReg",
    "Scale": "Standard Scaler",
    "Encoder": "One Hot Encoding",
    "Mean F1 Score": f1_score_mean,
    "roc_auc score mean": roc_auc_score_mean,
    "roc_auc score std" : roc_auc_score_std
}]
)

# Check if df_results is empty or all-NA
if df_results.empty or df_results.isna().all().all():
    df_results = new_results
else:
# Concatenate the new results with the existing df_results
    df_results = pd.concat([df_results, new_results], ignore_index=True)
```

In [137]:

```
df_results
```

Out[137]:

	Model	Scale	Encoder	Mean F1 Score	roc_auc score mean	roc_auc score std
0	LogReg	Standard Scaler	One Hot Encoding	0.8336	0.8474	0.0029

Min Max Scaler, One Hot Encoding

In [138]:

```
# Initialize our scalers and encoders
scaler = MinMaxScaler()
encoder = OneHotEncoder(handle_unknown="ignore", sparse= False)

# Creating pipelines for numeric and categorical transformers
cat_transformer = make_pipeline(encoder)
num_transformer = make_pipeline(scaler)

# combining into a preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('num',num_transformer,numerical_columns),
        ("cat",cat_transformer,categorical_columns)
    ]
)

#Initialize the logistic regression
logisticreg =  LogisticRegression(class_weight="balanced", solver="lbfgs", max_iter=1000
, random_state= 42)

# making the pipeline
pipe = make_pipeline(preprocessor, logisticreg)

#Calculate cross validated F1 scores

scores = cross_val_score(pipe, X, y, cv=5, scoring='f1')
f1_score_mean = round(scores.mean(), 4)

#Calculate cross validated ROC AUC

scores = cross_val_score(pipe, X, y, cv=5, scoring='roc_auc')
roc_auc_score_mean = round(scores.mean(),4)
roc_auc_score_std = round(scores.std(),4)

#print the mean F1 score
print(f"F1 SCORE : {f1_score_mean}")
```

```
print()
# Print the mean and standard deviation of the scores
print(f"Mean ROC AUC: {roc_auc_score_mean} +/- {roc_auc_score_std}")
```

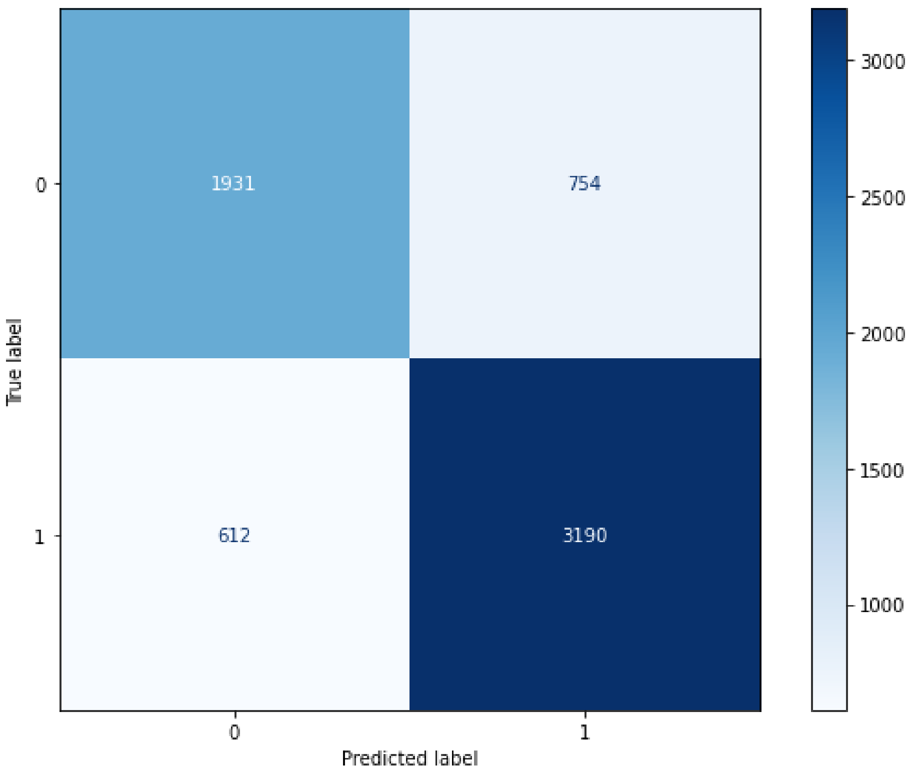
```
F1 SCORE : 0.8195

Mean ROC AUC: 0.8478 +/- 0.0027
```

Insights

Compared to ou previous model using the min max scaler produces no significant changes

In [139]:

```
# Confusion matrix

# Fit thepipeline to the training data
pipe.fit(X_train, y_train)

#Make predictions on the training set

y_pred_train = pipe.predict(X_train)

# Make predictions on test set

y_pred_test = pipe.predict(X_test)


cm = confusion_matrix(y_test, y_pred_test)

# plotting the confusion matrix

disp = ConfusionMatrixDisplay(confusion_matrix=cm)

# Increase the size of the display
fig, ax = plt.subplots(figsize=(10, 7))
disp.plot(cmap=plt.cm.Blues,ax=ax)
# Show the plot plt.show()

plt.show()
```



Insights

- **True Positives = 3190 true functional wells were predicted**

- **True Positives = [illegible] true functional wells were predicted**
- **False Positives = 754 true non functional wells were wrongly predicted**
- **False Negatives = 612 true functional wells were wrongly predicted**
- **True Negatives = 1931 true non functioning wells were predicted**

In [140]:

```
new_results = pd.DataFrame([{
    "Model": 'LogReg',
    "Scale": "Min Max",
    "Encoder": "One Hot Encoding",
    "Mean F1 Score": f1_score_mean,
    "roc_auc score mean": roc_auc_score_mean,
    "roc_auc score std" : roc_auc_score_std
}])

# Concatenate the new results with the existing df_results
df_results = pd.concat([df_results, new_results], ignore_index=True)
```

In [141]:

```
df_results
```

Out[141]:

	Model	Scale	Encoder	Mean F1 Score	roc_auc score mean	roc_auc score std
0	LogReg	Standard Scaler	One Hot Encoding	0.8336	0.8474	0.0029
1	LogReg	Min Max	One Hot Encoding	0.8195	0.8478	0.0027

Robust Scaler, Target Encoder Logistic Regression

The robust scaler uses the interquartile range to scale its numericals Target encoder works by providing changing categories to numericals according to the mean of the target variable

In [142]:

```
# Initialize our scalers and encoders
scaler = RobustScaler()
encoder = TargetEncoder(cols = categorical_columns)

# Creating pipelines for numeric and categorical transformers
cat_transformer = make_pipeline(encoder)
num_transformer = make_pipeline(scaler)

# combining into a preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('num',num_transformer,numerical_columns),
        ("cat",cat_transformer,categorical_columns)
    ]
)

#Initialize the logistic regression
logisticreg =  LogisticRegression(class_weight="balanced", solver="lbfgs", max_iter=1000
, random_state= 42)

# making the pipeline
pipe = make_pipeline(preprocessor, logisticreg)

#Calculate cross validated F1 scores

scores = cross_val_score(pipe, X, y, cv=5, scoring='f1')
f1_score_mean = round(scores.mean(), 4)
```

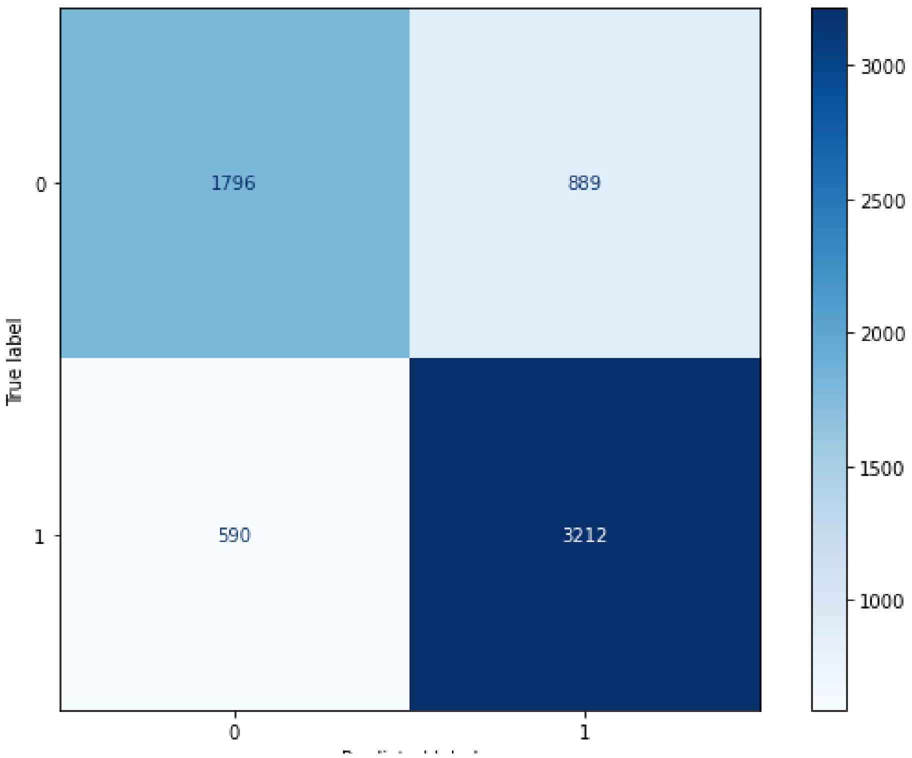
```
#Calculate cross validated ROC AUC

scores = cross_val_score(pipe, X, y, cv=5, scoring='roc_auc')
roc_auc_score_mean = round(scores.mean(),4)
roc_auc_score_std = round(scores.std(),4)

#print the mean F1 score
print(f"F1 SCORE : {f1_score_mean}")
print()
# Print the mean and standard deviation of the scores
print(f"Mean ROC AUC: {roc_auc_score_mean} +/- {roc_auc_score_std}")
```

```
F1 SCORE : 0.8078

Mean ROC AUC: 0.825 +/- 0.0043
```

The models performs faster but has a lower F1 score and ROC AUC

In [143]:

```
# Confusion matrix

# Fit thepipeline to the training data
pipe.fit(X_train, y_train)

#Make predictions on the training set

y_pred_train = pipe.predict(X_train)

# Make predictions on test set

y_pred_test = pipe.predict(X_test)


cm = confusion_matrix(y_test, y_pred_test)

# plotting the confusion matrix

disp = ConfusionMatrixDisplay(confusion_matrix=cm)

# Increase the size of the display
fig, ax = plt.subplots(figsize=(10, 7))
disp.plot(cmap=plt.cm.Blues,ax=ax)
# Show the plot plt.show()

plt.show()
```



Predicted label

Insights

- **True Positives = 3213 true functional wells were predicted**
- **False Positives = 891 true non functional wells were wrongly predicted**
- **False Negatives = 589 true functional wells were wrongly predicted**
- **True Negatives = 1794 true non functioning wells were predicted**

In [144]:

```
new_results = pd.DataFrame([{
    "Model": 'LogReg',
    "Scale": "Robust",
    "Encoder": "Target",
    "Mean F1 Score": f1_score_mean,
    "roc_auc score mean": roc_auc_score_mean,
    "roc_auc score std" : roc_auc_score_std
}])

# Concatenate the new results with the existing df_results
df_results = pd.concat([df_results, new_results], ignore_index=True)
```

In [145]:

```
df_results
```

Out[145]:

	Model	Scale	Encoder	Mean F1 Score	roc_auc score mean	roc_auc score std
0	LogReg	Standard Scaler	One Hot Encoding	0.8336	0.8474	0.0029
1	LogReg	Min Max	One Hot Encoding	0.8195	0.8478	0.0027
2	LogReg	Robust	Target	0.8078	0.8250	0.0043

Robust Scaling, One Hot Encoding

In [146]:

```
# Initialize our scalers and encoders
scaler = RobustScaler()
encoder = OneHotEncoder(handle_unknown="ignore", sparse = "False")

# Creating pipelines for numeric and categorical transformers
cat_transformer = make_pipeline(encoder)
num_transformer = make_pipeline(scaler)

# combining into a preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('num',num_transformer,numerical_columns),
        ("cat",cat_transformer,categorical_columns)
    ]
)

#Initialize the logistic regression
logisticreg =  LogisticRegression(class_weight="balanced", solver="lbfgs", max_iter=1000
, random_state= 42)

# making the pipeline
pipe = make_pipeline(preprocessor, logisticreg)

#Calculate cross validated F1 scores
```

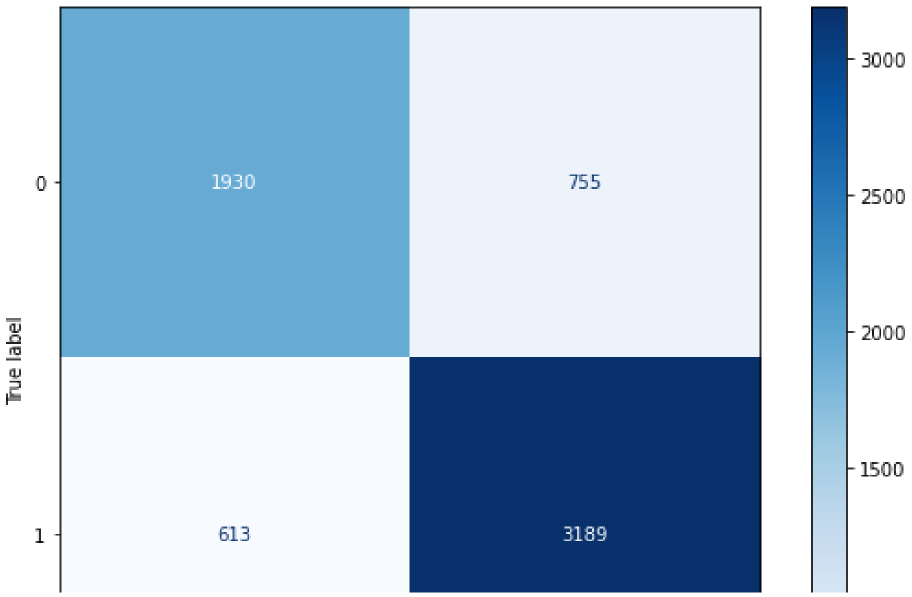
```
scores = cross_val_score(pipe, X, y, cv=5, scoring='f1')
f1_score_mean = round(scores.mean(), 4)

#Calculate cross validated ROC AUC

scores = cross_val_score(pipe, X, y, cv=5, scoring='roc_auc')
roc_auc_score_mean = round(scores.mean(),4)
roc_auc_score_std = round(scores.std(),4)

#print the mean F1 score
print(f"F1 SCORE : {f1_score_mean}")
print()
# Print the mean and standard deviation of the scores
print(f"Mean ROC AUC: {roc_auc_score_mean} +/- {roc_auc_score_std}")
```

```
F1 SCORE : 0.8194

Mean ROC AUC: 0.8479 +/- 0.0028
```

Performing a one hot encoding makes our models perform better than using a target encoder. This moves our model closer to our previous models

In [147]:

```
# Confusion matrix

# Fit thepipeline to the training data
pipe.fit(X_train, y_train)

#Make predictions on the training set

y_pred_train = pipe.predict(X_train)

# Make predictions on test set

y_pred_test = pipe.predict(X_test)


cm = confusion_matrix(y_test, y_pred_test)

# plotting the confusion matrix

disp = ConfusionMatrixDisplay(confusion_matrix=cm)

# Increase the size of the display
fig, ax = plt.subplots(figsize=(10, 7))
disp.plot(cmap=plt.cm.Blues,ax=ax)
# Show the plot plt.show()

plt.show()
```



Insights

- **True Positives = 3189 true functional wells were predicted**
- **False Positives = 755 true non functional wells were wrongly predicted**
- **False Negatives = 613 true functional wells were wrongly predicted**
- **True Negatives = 1930 true non functioning wells were predicted**

In [148]:

```
new_results = pd.DataFrame([{
    "Model": 'LogReg',
    "Scale": "Robust",
    "Encoder": "One Hot Encoding",
    "Mean F1 Score": f1_score_mean,
    "roc_auc score mean": roc_auc_score_mean,
    "roc_auc score std" : roc_auc_score_std
}])

# Concatenate the new results with the existing df_results
df_results = pd.concat([df_results, new_results], ignore_index=True)
```

In [149]:

```
df_results
```

Out[149]:

	Model	Scale	Encoder	Mean F1 Score	roc_auc score mean	roc_auc score std
0	LogReg	Standard Scaler	One Hot Encoding	0.8336	0.8474	0.0029
1	LogReg	Min Max	One Hot Encoding	0.8195	0.8478	0.0027
2	LogReg	Robust	Target	0.8078	0.8250	0.0043
3	LogReg	Robust	One Hot Encoding	0.8194	0.8479	0.0028

Analysis of Logistic Regression

1. **LogReg with Standard Scaler and One Hot Encoding**

- **F1 Score:** 0.8195
- **ROC AUC Mean:** 0.8474
- **ROC AUC std:** 0.0029
- **Summary:** This model showed good performance with a stable ROC AUC score showing consistent results across different folds

1. **LogReg with Min Max Scaler and One Hot Encoding**

- **F1 Score:** 0.8195
- **ROC AUC Mean:** 0.8478
- **ROC AUC std:** 0.0027
- **Summary:** This model showed a similar performance with the previous with almost identical ROC AUC scores but was more consistent across different folds

1. **LogReg with Robust Scaler and Target Encoding**

- **F1 Score:** 0.8077
- **ROC AUC Mean:** 0.8250
- **ROC AUC std:** 0.0043
- **Summary:** This model performed worse than the previous models with a wore F1 score and ROC mean. It showed that target encoding made the model performed worse than One Hot encoding. It also was more inconsistent in performance with a worse ROC AUC std

1. **LogReg with Robust Scaler and One Hot Encoding**

- **F1 Score:** 0.8196
- **ROC AUC Mean:** 0.8479
- **ROC AUC std:** 0.0028
- **Summary:** This model showed the best performance further proving that one hot encoding shows the better performance compared to target encoding

Decision Tree

In [150]:

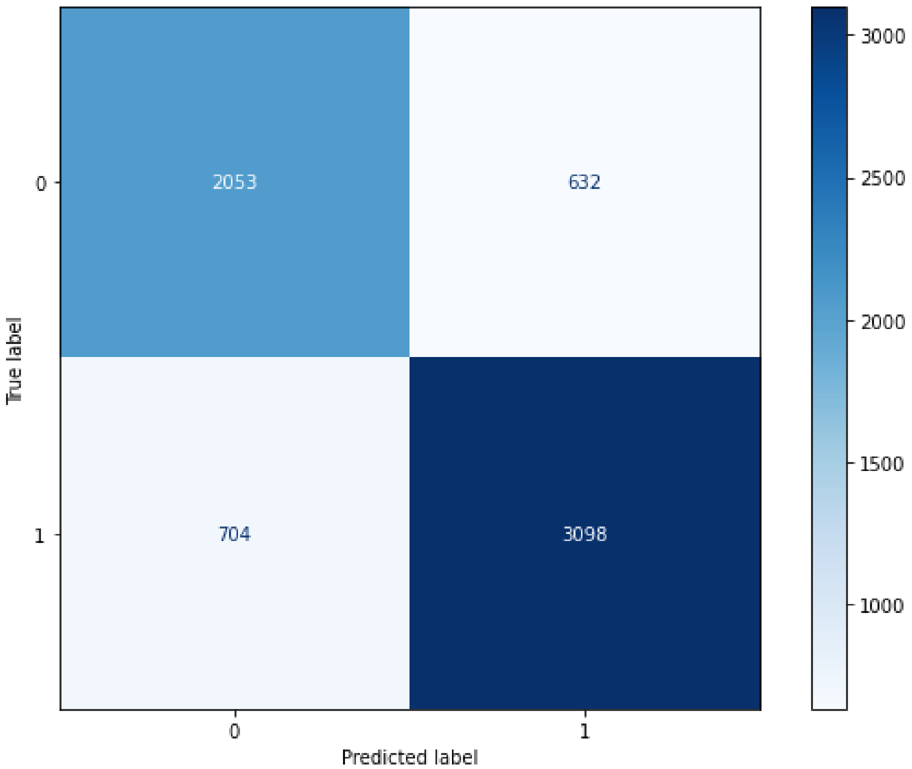
```
scaler = RobustScaler()
encoder = OneHotEncoder(handle_unknown= "ignore", sparse= False)

# Creating pipelines for numeric and categorical transformers
cat_transformer = make_pipeline(encoder)
num_transformer = make_pipeline(scaler)

# combining into a preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('num',num_transformer,numerical_columns),
        ("cat",cat_transformer,categorical_columns)
    ]
)

dt_model = DecisionTreeClassifier(class_weight= "balanced", random_state= 42)

pipe = make_pipeline(preprocessor, dt_model)

#Calculate cross validated F1 scores

scores = cross_val_score(pipe, X, y, cv=5, scoring='f1')
f1_score_mean = round(scores.mean(), 4)

#Calculate cross validated ROC AUC

scores = cross_val_score(pipe, X, y, cv=5, scoring='roc_auc')
roc_auc_score_mean = round(scores.mean(),4)
roc_auc_score_std = round(scores.std(),4)

#print the mean F1 score
print(f"F1 SCORE : {f1_score_mean}")
print()
# Print the mean and standard deviation of the scores
print(f"Mean ROC AUC: {roc_auc_score_mean} +/- {roc_auc_score_std}")
```

```
F1 SCORE : 0.8215

Mean ROC AUC: 0.7961 +/- 0.0048
```

In [151]:

```
# Confusion matrix

# Fit the pipeline to the training data
pipe.fit(X_train, y_train)

#Make predictions on the training set
```

```
y_pred_train = pipe.predict(X_train)

# Make predictions on test set

y_pred_test = pipe.predict(X_test)


cm = confusion_matrix(y_test, y_pred_test)

# plotting the confusion matrix

disp = ConfusionMatrixDisplay(confusion_matrix=cm)

# Increase the size of the display
fig, ax = plt.subplots(figsize=(10, 7))
disp.plot(cmap=plt.cm.Blues,ax=ax)
# Show the plot plt.show()

plt.show()
```



insights

- **True Positives = 3098 true functional wells were predicted**
- **False Positives = 632 true non functional wells were wrongly predicted**
- **False Negatives = 704 true functional wells were wrongly predicted**
- **True Negatives = 2053 true non functioning wells were predicted**

Finding optimum max depth for decision tree

In [152]:

```
train_accuracy = []
test_accuracy = []

for depth in range(1,30):
    dt_model = DecisionTreeClassifier(max_depth= depth, random_state= 42)
    pipe = make_pipeline(preprocessor,dt_model)
    pipe.fit(X_train,y_train)
    y_pred_train = pipe.predict(X_train)
    y_pred_test = pipe.predict(X_test)
    train_accuracy.append(balanced_accuracy_score(y_train,y_pred_train))
```

```
    test_accuracy.append(balanced_accuracy_score(y_test,y_pred_test))
```

In [153]:

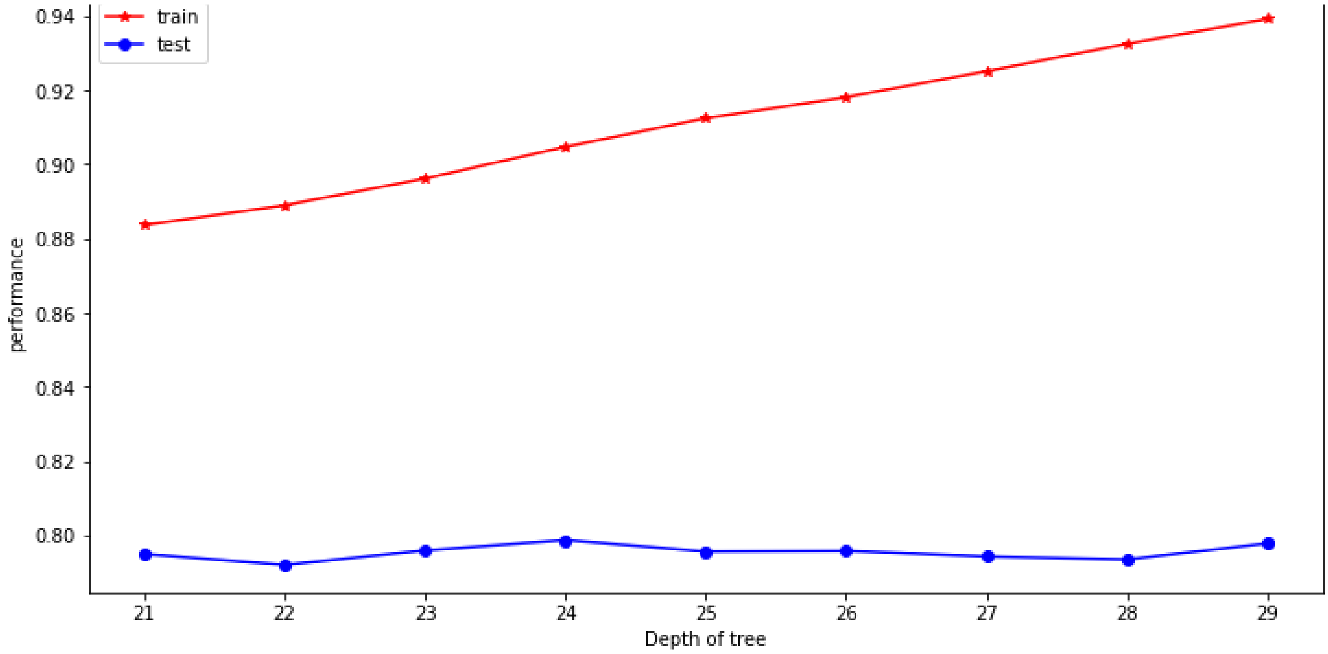
```
frame = pd.DataFrame({'max_depth':range(1,30), 'train_acc':train_accuracy, 'test_acc':te
st_accuracy})
frame.head(30)
```

Out[153]:

	max_depth	train_acc	test_acc
0	1	0.628812	0.627220
1	2	0.697815	0.700573
2	3	0.697815	0.700573
3	4	0.715364	0.712643
4	5	0.728617	0.727212
5	6	0.744282	0.744058
6	7	0.753837	0.747419
7	8	0.769298	0.761379
8	9	0.776790	0.767224
9	10	0.785950	0.774178
10	11	0.792657	0.774430
11	12	0.800122	0.775350
12	13	0.809153	0.778558
13	14	0.820151	0.779531
14	15	0.826046	0.781056
15	16	0.838594	0.781555
16	17	0.849855	0.789738
17	18	0.858210	0.790736
18	19	0.867477	0.796102
19	20	0.876091	0.796626
20	21	0.883640	0.794905
21	22	0.888991	0.791990
22	23	0.896211	0.795888
23	24	0.904762	0.798660
24	25	0.912474	0.795612
25	26	0.918153	0.795765
26	27	0.925149	0.794251
27	28	0.932536	0.793449
28	29	0.939236	0.797776

In [154]:

```
plt.figure(figsize=(12,6))
plt.plot(frame['max_depth'][20:], frame['train_acc'][20:], marker='*',color="red",label=
"train")
plt.plot(frame['max_depth'][20:], frame['test_acc'][20:], marker='o',color="blue", label
="test")
plt.xlabel('Depth of tree')
plt.ylabel('performance')
plt.legend()
plt.show();
```



From the above graph we can see that 24 will be our most ideal depth in building our model

In [155]:

```
scaler = RobustScaler()
encoder = OneHotEncoder(handle_unknown= "ignore", sparse= False)

# Creating pipelines for numeric and categorical transformers
cat_transformer = make_pipeline(encoder)
num_transformer = make_pipeline(scaler)

# combining into a preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('num',num_transformer,numerical_columns),
        ("cat",cat_transformer,categorical_columns)
    ]
)

dt_model = DecisionTreeClassifier(class_weight= "balanced",max_depth=24,min_samples_leaf
=5,max_leaf_nodes=25, random_state= 42)

pipe = make_pipeline(preprocessor, dt_model)

#Calculate cross validated F1 scores

scores = cross_val_score(pipe, X, y, cv=5, scoring='f1')
f1_score_mean = round(scores.mean(), 4)

#Calculate cross validated ROC AUC

scores = cross_val_score(pipe, X, y, cv=5, scoring='roc_auc')
roc_auc_score_mean = round(scores.mean(),4)
roc_auc_score_std = round(scores.std(),4)

#print the mean F1 score
print(f"F1 SCORE : {f1_score_mean}")
print()
# Print the mean and standard deviation of the scores
print(f"Mean ROC AUC: {roc_auc_score_mean} +/- {roc_auc_score_std}")
```

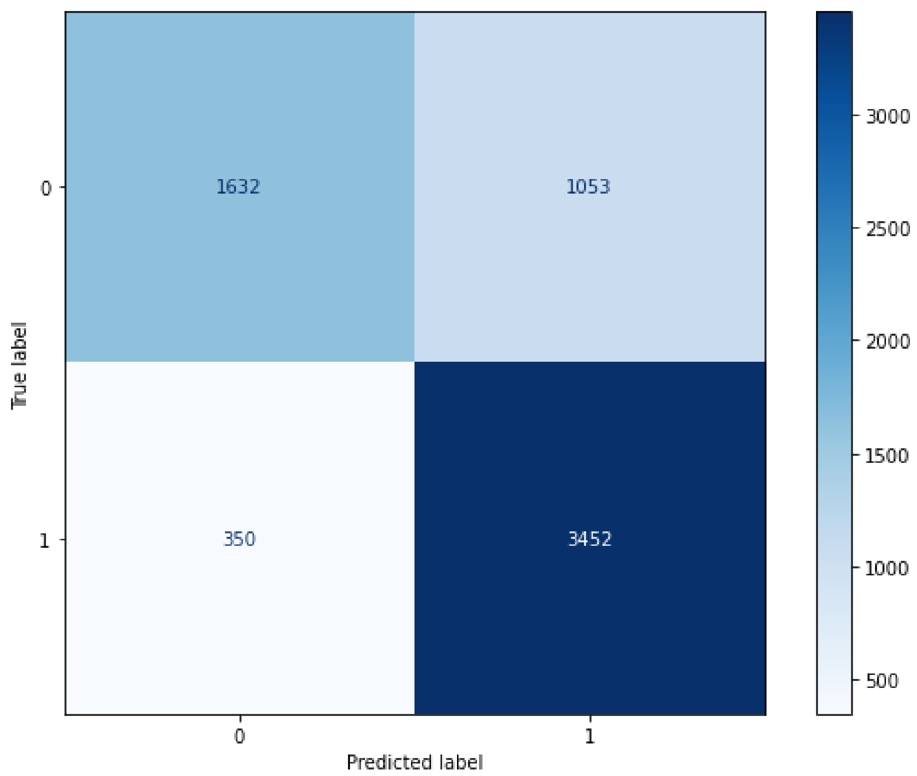
```
F1 SCORE : 0.8218

Mean ROC AUC: 0.8179 +/- 0.0051
```

insights

There is an improvement from our initial decision tree model in the mean ROC AUC

In [156]:

```
# Confusion matrix

# Fit the pipeline to the training data
pipe.fit(X_train, y_train)

#Make predictions on the training set

y_pred_train = pipe.predict(X_train)

# Make predictions on test set

y_pred_test = pipe.predict(X_test)


cm = confusion_matrix(y_test, y_pred_test)

# plotting the confusion matrix

disp = ConfusionMatrixDisplay(confusion_matrix=cm)

# Increase the size of the display
fig, ax = plt.subplots(figsize=(10, 7))
disp.plot(cmap=plt.cm.Blues,ax=ax)
# Show the plot plt.show()

plt.show()
```



insights

- **True Positives = 3452 true functional wells were predicted**
- **False Positives = 1053 true non functional wells were wrongly predicted**
- **False Negatives = 350 true functional wells were wrongly predicted**
- **True Negatives = 1632 true non functioning wells were predicted**

In [157]:

```
new_results = pd.DataFrame([{
    "Model": 'Decision Tree',
    "Scale": "Robust",
    "Encoder": "One Hot Encoding",
```

```
    "Mean F1 Score": f1_score_mean,
    "roc_auc score mean": roc_auc_score_mean,
    "roc_auc score std" : roc_auc_score_std
}])

# Concatenate the new results with the existing df_results
df_results = pd.concat([df_results, new_results], ignore_index=True)
```

In [158]:

```
df_results
```

Out[158]:

	Model	Scale	Encoder	Mean F1 Score	roc_auc score mean	roc_auc score std
0	LogReg	Standard Scaler	One Hot Encoding	0.8336	0.8474	0.0029
1	LogReg	Min Max	One Hot Encoding	0.8195	0.8478	0.0027
2	LogReg	Robust	Target	0.8078	0.8250	0.0043
3	LogReg	Robust	One Hot Encoding	0.8194	0.8479	0.0028
4	Decision Tree	Robust	One Hot Encoding	0.8218	0.8179	0.0051

Model Analysis

Analysis of all models

1. **LogReg with Standard Scaler and One Hot Encoding**

- **F1 Score:** 0.8195
- **ROC AUC Mean:** 0.8474
- **ROC AUC std:** 0.0029
- **Summary:** This model showed good performance with a stable ROC AUC score showing consistent results across different folds

1. **LogReg with Min Max Scaler and One Hot Encoding**

- **F1 Score:** 0.8195
- **ROC AUC Mean:** 0.8478
- **ROC AUC std:** 0.0027
- **Summary:** This model showed a similar performance with the previous with almost identical ROC AUC scores but was more consistent across different folds

1. **LogReg with Robust Scaler and Target Encoding**

- **F1 Score:** 0.8077
- **ROC AUC Mean:** 0.8250
- **ROC AUC std:** 0.0043
- **Summary:** This model performed worse than the previous models with a wore F1 score and ROC mean. It showed that target encoding made the model performed worse than One Hot encoding. It also was more inconsistent in performance with a worse ROC AUC std

1. **LogReg with Robust Scaler and One Hot Encoding**

- **F1 Score:** 0.8196
- **ROC AUC Mean:** 0.8479
- **ROC AUC std:** 0.0028
- **Summary:** This model showed similar performance with other linear regression models with one hot encoding further showing the strength of using one hot encoding for our categorical performance

1. **Decision Tree with Robust Scaler and One Hot Encoding**

- **F1 Score:** 0.8218
- **ROC AUC Mean:** 0.8179
- **ROC AUC std:** 0.0051
- **Summary:** This model showed the worst performance among our other models with the worse ROC AUC scores compared to other. The model also showed the least consistency on each fold

Model choice

From the analysis of model performance, logistic regression outperformed the decision tree model. One-Hot Encoding emerged as the preferred method for handling categorical columns, significantly improving model performance. While the choice of scaling had no impact on model metrics, robust scaling proved beneficial by improving the model's runtime efficiency.

Based on this analysis, the recommended approach is to use logistic regression combined with Robust Scaling and One-Hot Encoding.

Results

1. **Simple Pumps and Extraction Type:** From the bivariate analysis we did on the dataset I noticed that simple pumps, with simple extractions type were the most functional as compared to other pumps. These might because the need little to no maintenance
2. **Source Type:** From the bivariate analysis we noticed that wells that sourced their water from spring retained their functionality than other water sources. This might because water from springs get replenished therefore are least affected from climate of an area
3. **Water point age:** From the analysis of the datasest it showed that older water points were mi=ore likely to be non functional than younger water points. These might because older pumps tend to break more often or older wells tends to have less water than newer wells
4. **Payment:** Wells that require no payment were most likely to be functioning. This might because without any form of transactions it becomes difficult to maintain the pumps.
5. **Predictive Model Performance** The model chosen demostrated high accuracy in predicting the operational status of wells. This suggest that the model succeeds its intended purpose and can be used to predict operational status of wells in Tannzania

Recommendations

1. **Payment** Based on the findings it is recommended to priotise building wells that have a payment transaction. This ensures that scheme managers have money they can use to maintain wells in Tanzania
2. **Improved data collection.** The dataset had a lot errors that were noticed in data cleaning. This can affect the model accuracy so improvement in data collection might improve the accuracy of the model
3. **Integration of external factors** : Further information such as climate of the area would have been useful in our analysis

Next Steps

1. **Longitudes and Latitudes** : The model did not use the longitudes and latitudes which could have provided better insights on the regions where the wells are located

Thank You