MINISTRY OF WATER AND IRRIGATION TANZANIA

Overview

Water is a basic need for humans for health and sanitation. In remote areas, water wells become the source of this requirement. For a population of over 65 million people in Tanzania, being able to service and maintain this wells becomes paramount and therefore fulfils the sustainable development goal number 6: Clean Water and Sanitation

Business Understanding

MWN Consultancy has been tasked to predict the condition of a well through the data provided. Through modeling, we should come up with the best model that classifies pumps into 3 categories; functional, non-functional, requires repair.

This model should be able to categorise the current installed base but also declare combined characteristics and features of attributes that are precursors to the conditions of the wells

Objectives

1. Identify and present the best model that classifies the state of the well with highest accuracy

Data understanding

The data seems to have been split between train and test data

```
#importing libraries to support in data understanding and cleaning
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings('ignore')
```

```
df_values= pd.read_csv('Training set values.csv')
df_values.head()
```

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wpt_name	num_private	...	payment_type	wate
0	69572	6000.0	2011-03-14	Roman	1390	Roman	34.938093	-9.856322	none	0	...	annually	
1	8776	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.698766	-2.147466	Zahanati	0	...	never pay	
2	34310	25.0	2013-02-25	Lottery Club	686	World vision	37.460664	-3.821329	Kwa Mahundi	0	...	per bucket	
3	67743	0.0	2013-01-28	Unicef	263	UNICEF	38.486161	-11.155298	Zahanati Ya Nanyumbu	0	...	never pay	
4	19728	0.0	2011-07-13	Action In A	0	Artisan	31.130847	-1.825359	Shuleni	0	...	never pay	

5 rows × 40 columns

```
df_labels= pd.read_csv('Training set labels.csv')
df_labels.head()
```

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

Next steps: View recommended plots New interactive sheet

```
df_test= pd.read_csv('Test set Values.csv')
df_test.head()
```

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wpt_name	num_private	...	payment_type
0	50785	0.0	2013-02-04	Dmdd	1996	DMDD	35.290799	-4.059696	Dinamu Secondary School	0	...	never pay
1	51630	0.0	2013-02-04	Government Of Tanzania	1569	DWE	36.656709	-3.309214	Kimnyak	0	...	never pay
2	17168	0.0	2013-02-01	NaN	1567	NaN	34.767863	-5.004344	Puma Secondary	0	...	never pay
3	45559	0.0	2013-01-22	Finn Water	267	FINN WATER	38.058046	-9.418672	Kwa Mzee Pange	0	...	unknown
4	49871	500.0	2013-03-27	Bruder	1260	BRUDER	35.006123	-10.950412	Kwa Mzee Turuka	0	...	monthly

5 rows × 40 columns

```
df_pred= pd.read_csv('SubmissionFormat.csv')
df_pred.head()
```

	id	status_group
0	50785	predicted label
1	51630	predicted label
2	17168	predicted label
3	45559	predicted label
4	49871	predicted label

Next steps: View recommended plots New interactive sheet

```
df_merge = df_values.merge(df_labels, on="id")
df_merge
```

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

59400 rows × 41 columns

```
df_merge.shape
```

```
(59400, 41)
```

```
df_merge.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59400 entries, 0 to 59399
Data columns (total 41 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     59400 non-null  int64  
 1   amount_tsh             59400 non-null  float64
 2   date_recorded          59400 non-null  object 
 3   funder                 55763 non-null  object 
 4   gps_height             59400 non-null  int64  
 5   installer              55745 non-null  object 
 6   longitude              59400 non-null  float64
 7   latitude               59400 non-null  float64
 8   wpt_name               59398 non-null  object 
 9   num_private            59400 non-null  int64  
 10  basin                  59400 non-null  object 
 11  subvillage             59029 non-null  object 
 12  region                 59400 non-null  object 
 13  region_code            59400 non-null  int64  
 14  district_code          59400 non-null  int64  
 15  lga                    59400 non-null  object 
 16  ward                   59400 non-null  object 
 17  population             59400 non-null  int64  
 18  public_meeting         56066 non-null  object 
 19  recorded_by            59400 non-null  object 
 20  scheme_management      55522 non-null  object 
 21  scheme_name            30590 non-null  object 
 22  permit                 56344 non-null  object 
 23  construction_year      59400 non-null  int64  
 24  extraction_type        59400 non-null  object 
 25  extraction_type_group  59400 non-null  object 
 26  extraction_type_class  59400 non-null  object 
 27  management             59400 non-null  object 
 28  management_group       59400 non-null  object 
 29  payment                59400 non-null  object 
 30  payment_type           59400 non-null  object 
 31  water_quality          59400 non-null  object 
```

```
 32  quality_group          59400 non-null  object
 33  quantity               59400 non-null  object
 34  quantity_group         59400 non-null  object
 35  source                 59400 non-null  object
 36  source_type            59400 non-null  object
 37  source_class           59400 non-null  object
 38  waterpoint_type        59400 non-null  object
 39  waterpoint_type_group  59400 non-null  object
 40  status_group           59400 non-null  object
dtypes: float64(3), int64(7), object(31)
memory usage: 18.6+ MB
```

```
df_merge.describe()
```

	id	amount_tsh	gps_height	longitude	latitude	num_private	region_code	district_code	population	cons
count	59400.000000	59400.000000	59400.000000	59400.000000	5.940000e+04	59400.000000	59400.000000	59400.000000	59400.000000	
mean	37115.131768	317.650385	668.297239	34.077427	-5.706033e+00	0.474141	15.297003	5.629747	179.909983	
std	21453.128371	2997.574558	693.116350	6.567432	2.946019e+00	12.236230	17.587406	9.633649	471.482176	
min	0.000000	0.000000	-90.000000	0.000000	-1.164944e+01	0.000000	1.000000	0.000000	0.000000	
25%	18519.750000	0.000000	0.000000	33.090347	-8.540621e+00	0.000000	5.000000	2.000000	0.000000	
50%	37061.500000	0.000000	369.000000	34.908743	-5.021597e+00	0.000000	12.000000	3.000000	25.000000	
75%	55656.500000	20.000000	1319.250000	37.178387	-3.326156e+00	0.000000	17.000000	5.000000	215.000000	
max	74247.000000	350000.000000	2770.000000	40.345193	-2.000000e-08	1776.000000	99.000000	80.000000	30500.000000	

```
df_merge.columns
```

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

```
df_merge.dtypes
```

	0
id	int64
amount_tsh	float64
date_recorded	object
funder	object
gps_height	int64
installer	object
longitude	float64
latitude	float64
wpt_name	object
num_private	int64
basin	object
subvillage	object
region	object
region_code	int64
district_code	int64
lga	object
ward	object
population	int64
public_meeting	object
recorded_by	object
scheme_management	object
scheme_name	object
permit	object
construction_year	int64
extraction_type	object
extraction_type_group	object
extraction_type_class	object
management	object
management_group	object
payment	object
payment_type	object
water_quality	object
quality_group	object
quantity	object
quantity_group	object
source	object
source_type	object
source_class	object
waterpoint_type	object
waterpoint_type_group	object
status_group	object

dtype: object

```
#checking for total count of the 3 categories
df_merge['status_group'].value_counts()
```

status_group	count
functional	32259
non functional	22824
functional needs repair	4317

dtype: int64

Data Cleaning

Cleaning Training Data

Handling null values

Dropping duplicates

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

	0
id	0
amount_tsh	0
date_recorded	0
funder	3637
gps_height	0
installer	3655
longitude	0
latitude	0
wpt_name	2
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	3878
scheme_name	28810
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
status_group	0

dtype: int64

```
df_merge_null = df_merge.columns[df_merge.isna().sum() > 0]
df_merge_null
```

```
Index(['funder', 'installer', 'wpt_name', 'subvillage', 'public_meeting',
       'scheme_management', 'scheme_name', 'permit'],
      dtype='object')
```

```
#Since scheme_name = scheme_management accordng to data description, we drop the data with the most null values
df_merge_drop = df_merge.drop(columns=['scheme_name'])
df_merge_drop.shape
```

```
(59400, 40)
```

```
#viewing data in columns with null values
df_merge[['funder', 'installer', 'wpt_name', 'subvillage', 'public_meeting',
       'scheme_management', 'scheme_name', 'permit']].head()
```

	funder	installer	wpt_name	subvillage	public_meeting	scheme_management	scheme_name	permit
0	Roman	Roman	none	Mnyusi B	True	VWC	Roman	False
1	Grumeti	GRUMETI	Zahanati	Nyamara	NaN	Other	NaN	True
2	Lottery Club	World vision	Kwa Mahundi	Majengo	True	VWC	Nyumba ya mungu pipe scheme	True
3	Unicef	UNICEF	Zahanati Ya Nanyumbu	Mahakamani	True	VWC	NaN	True
4	Action In A	Artisan	Shuleni	Kyanyamisa	True	NaN	NaN	True

```
#viewing the difference between region and subvillage based on data description
df_merge[['region', 'subvillage']].nunique()
```

	0
region	21
subvillage	19287

dtype: int64

```
df_merge_clean=df_merge_drop.dropna()
```

```
df_merge_clean.duplicated().sum()
```

```
0
```

```
non_unique_cols = [col for col in df_merge_clean if df_merge_clean[col].nunique() == 1]

if non_unique_cols:
  print(non_unique_cols)
```

```
['recorded_by']
```

```
df_merge_clean = df_merge_clean.drop(columns='recorded_by')
```

```
df_merge_clean['date_recorded']=pd.to_datetime(df_merge_clean['date_recorded'])
```

```
df_merge_clean['construction_year']=pd.to_datetime(df_merge_clean['construction_year']).dt.year
```

```
df_merge_clean.dtypes
```

	0
id	int64
amount_tsh	float64
date_recorded	datetime64[ns]
funder	object
gps_height	int64
installer	object
longitude	float64
latitude	float64
wpt_name	object
num_private	int64
basin	object
subvillage	object
region	object
region_code	int64
district_code	int64
lga	object
ward	object
population	int64
public_meeting	object
scheme_management	object
permit	object
construction_year	int32
extraction_type	object
extraction_type_group	object
extraction_type_class	object
management	object
management_group	object
payment	object
payment_type	object
water_quality	object
quality_group	object
quantity	object
quantity_group	object
source	object
source_type	object
source_class	object
waterpoint_type	object
waterpoint_type_group	object
status_group	object

dtype: object

```
(df_merge_clean['extraction_type'] == df_merge_clean['extraction_type_group']).all()
```

```
False
```

```
df_merge_clean.nunique()
```

	0
id	48285
amount_tsh	91
date_recorded	324
funder	1586
gps_height	2426
installer	1787
longitude	46913
latitude	46915
wpt_name	31029
num_private	58
basin	9
subvillage	16183
region	21
region_code	27
district_code	18
lga	117
ward	1862
population	991
public_meeting	2
scheme_management	11
permit	2
construction_year	1
extraction_type	18
extraction_type_group	13
extraction_type_class	7
management	12
management_group	5
payment	7
payment_type	7
water_quality	8
quality_group	6
quantity	5
quantity_group	5
source	10
source_type	7
source_class	3
waterpoint_type	7
waterpoint_type_group	6
status_group	3

dtype: int64

```
#extraction_type, extraction_type_group, extraction_type_class all have the same column description
#keeping column with the highest number of unique values
df_merge_clean = df_merge_clean.drop(columns=['extraction_type_group','extraction_type_class'])
```

```
#payment, payment_type,  all have the same column description
#keeping column with the least words
df_merge_clean = df_merge_clean.drop(columns=['payment'])
```

```
#water_quality, quality_group,  all have the same column description
#keeping column that is most descriptive
df_merge_clean = df_merge_clean.drop(columns=['quality_group'])
```

```
(df_merge_clean['quantity'] == df_merge_clean['quantity_group']).all()
```

```
True
```

```
df_merge_clean = df_merge_clean.drop(columns=['quantity'])
```

```
#source, source_type, source_class all have the same column description
#keeping column with the highest number of unique values
df_merge_clean = df_merge_clean.drop(columns=['source_type','source_class'])
```

```
#waterpoint_type, waterpoint_type_group, all have the same column description
#keeping column with the highest number of unique values
df_merge_clean = df_merge_clean.drop(columns=['waterpoint_type_group'])
```

```
df_merge_clean.shape
```

```
(48285, 31)
```

Cleaning Testing Data

Handling null values

Dropping duplicates

```
df_test.isna().sum()
```

	0
id	0
amount_tsh	0
date_recorded	0
funder	870
gps_height	0
installer	877
longitude	0
latitude	0
wpt_name	0
num_private	0
basin	0
subvillage	99
region	0
region_code	0
district_code	0
lga	0
ward	0
population	0
public_meeting	821
recorded_by	0
scheme_management	969
scheme_name	7242
permit	737
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

```
#viewing data in columns with null values
df_test[['funder', 'installer', 'wpt_name', 'subvillage', 'public_meeting',
       'scheme_management', 'scheme_name', 'permit']].head()
```

	funder	installer	wpt_name	subvillage	public_meeting	scheme_management	scheme_name	permit
0	Dmdd	DMDD	Dinamu Secondary School	Magoma	True	Parastatal	NaN	True
1	Government Of Tanzania	DWE	Kimnyak	Kimnyak	True	VWC	TPRI pipe line	True
2	NaN	NaN	Puma Secondary	Msatu	True	VWC	P	NaN
3	Finn Water	FINN WATER	Kwa Mzee Pange	Kipindimbi	NaN	VWC	NaN	True
4	Bruder	BRUDER	Kwa Mzee Turuka	Losonga	NaN	Water Board	BRUDER	True

```
dropped_columns = list(set(df_merge.columns) - set(df_merge_clean.columns))
dropped_columns
```

```
['source_class',
 'payment',
 'extraction_type_class',
 'waterpoint_type_group',
 'scheme_name',
 'quality_group',
 'quantity',
 'source_type',
 'extraction_type_group',
 'recorded_by']
```

```
#dropping all columns dropped in the training data
df_test_clean=df_test.copy()
df_test_clean = df_test_clean.drop(columns=['quantity',
 'payment',
 'quality_group',
 'source_type',
 'source_class',
 'extraction_type_class',
 'extraction_type_group','scheme_name',
 'recorded_by',
 'waterpoint_type_group'])
```

```
df_test_clean.shape
```

```
(14850, 30)
```

```
df_test_clean['date_recorded']=pd.to_datetime(df_test_clean['date_recorded'])
```

```
df_test_clean.isna().sum()
```

	0
id	0
amount_tsh	0
date_recorded	0
funder	870
gps_height	0
installer	877
longitude	0
latitude	0
wpt_name	0
num_private	0
basin	0
subvillage	99
region	0
region_code	0
district_code	0
lga	0
ward	0
population	0
public_meeting	821
scheme_management	969
permit	737
construction_year	0
extraction_type	0
management	0
management_group	0
payment_type	0
water_quality	0
quantity_group	0
source	0
waterpoint_type	0

dtype: int64

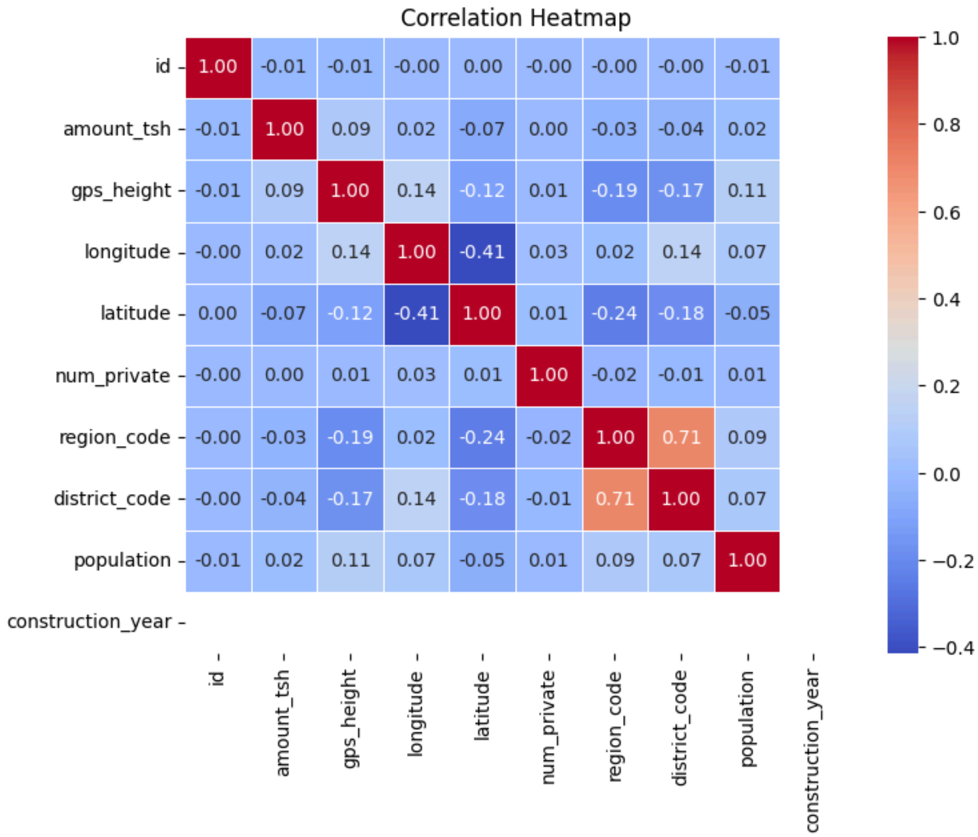
```
df_test_clean = df_test_clean.dropna()
df_test_clean.shape
```

```
(12097, 30)
```

Exploratory Data Analysis

Correlation Heatmap

```
#plotting a correlation map
corr_matrix = df_merge_clean.select_dtypes(include=['int','float']).corr()
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()
```




```
unique_values = df_merge_clean['status_group'].value_counts()
unique_values
```

status_group	count
functional	26516
non functional	18271
functional needs repair	3498

dtype: int64

Preprocessing

Encoding

```
#identifying the columns and the unique features
df_merge_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 48285 entries, 0 to 59399
Data columns (total 31 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id                  48285 non-null  int64
 1   amount_tsh          48285 non-null  float64
 2   date_recorded       48285 non-null  datetime64[ns]
 3   funder              48285 non-null  object
 4   gps_height          48285 non-null  int64
 5   installer           48285 non-null  object
 6   longitude           48285 non-null  float64
 7   latitude            48285 non-null  float64
 8   wpt_name            48285 non-null  object
 9   num_private         48285 non-null  int64
 10  basin               48285 non-null  object
```

```
 11  subvillage         48285 non-null  object
 12  region             48285 non-null  object
 13  region_code        48285 non-null  int64
 14  district_code      48285 non-null  int64
 15  lga                48285 non-null  object
 16  ward               48285 non-null  object
 17  population         48285 non-null  int64
 18  public_meeting     48285 non-null  object
 19  scheme_management  48285 non-null  object
 20  permit             48285 non-null  object
 21  construction_year  48285 non-null  int32
 22  extraction_type    48285 non-null  object
 23  management         48285 non-null  object
 24  management_group   48285 non-null  object
 25  payment_type       48285 non-null  object
 26  water_quality      48285 non-null  object
 27  quantity_group     48285 non-null  object
 28  source             48285 non-null  object
 29  waterpoint_type    48285 non-null  object
 30  status_group       48285 non-null  object
dtypes: datetime64[ns](1), float64(3), int32(1), int64(6), object(20)
memory usage: 11.6+ MB
```

```
# getting the numeric cols and the categorical cols
num_original_columns = df_merge_clean.select_dtypes(include=np.number).columns.tolist()
categorical_cols = df_merge_clean.select_dtypes(exclude=np.number).columns.tolist()

print("Categorical columns:", categorical_cols)
num_original_columns
```

```
Categorical columns: ['date_recorded', 'funder', 'installer', 'wpt_name', 'basin', 'subvillage', 'region', 'lga', 'ward', 'public_meetin
['id',
 'amount_tsh',
 'gps_height',
 'longitude',
 'latitude',
 'num_private',
 'region_code',
 'district_code',
 'population',
 'construction_year']
```

```
df_merge_clean[categorical_cols].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 48285 entries, 0 to 59399
Data columns (total 21 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   date_recorded      48285 non-null  datetime64[ns]
 1   funder             48285 non-null  object
 2   installer          48285 non-null  object
 3   wpt_name           48285 non-null  object
 4   basin              48285 non-null  object
 5   subvillage         48285 non-null  object
 6   region             48285 non-null  object
 7   lga                48285 non-null  object
 8   ward               48285 non-null  object
 9   public_meeting     48285 non-null  object
 10  scheme_management  48285 non-null  object
 11  permit             48285 non-null  object
 12  extraction_type    48285 non-null  object
 13  management         48285 non-null  object
 14  management_group   48285 non-null  object
 15  payment_type       48285 non-null  object
 16  water_quality      48285 non-null  object
 17  quantity_group     48285 non-null  object
 18  source             48285 non-null  object
 19  waterpoint_type    48285 non-null  object
 20  status_group       48285 non-null  object
dtypes: datetime64[ns](1), object(20)
memory usage: 8.1+ MB
```

```
# Converting 'year' to integer
df_merge_encoded=df_merge_clean.copy()
df_merge_encoded['date_recorded'] = df_merge_encoded['date_recorded'].dt.year
df_merge_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 48285 entries, 0 to 59399
Data columns (total 31 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 48285 non-null  int64  
 1   amount_tsh         48285 non-null  float64
 2   date_recorded      48285 non-null  int32  
 3   funder             48285 non-null  object 
 4   gps_height         48285 non-null  int64  
 5   installer          48285 non-null  object 
 6   longitude          48285 non-null  float64
 7   latitude           48285 non-null  float64
 8   wpt_name           48285 non-null  object 
 9   num_private        48285 non-null  int64  
 10  basin              48285 non-null  object 
 11  subvillage         48285 non-null  object 
 12  region             48285 non-null  object 
 13  region_code        48285 non-null  int64  
 14  district_code      48285 non-null  int64  
 15  lga                48285 non-null  object 
 16  ward               48285 non-null  object 
 17  population         48285 non-null  int64  
 18  public_meeting     48285 non-null  object 
 19  scheme_management  48285 non-null  object 
 20  permit             48285 non-null  object 
 21  construction_year  48285 non-null  int32  
 22  extraction_type    48285 non-null  object 
 23  management         48285 non-null  object 
 24  management_group   48285 non-null  object 
 25  payment_type       48285 non-null  object 
 26  water_quality      48285 non-null  object 
 27  quantity_group     48285 non-null  object 
 28  source             48285 non-null  object 
 29  waterpoint_type    48285 non-null  object 
 30  status_group       48285 non-null  object 
dtypes: float64(3), int32(2), int64(6), object(20)
memory usage: 11.4+ MB
```

```
# Converting 'year' to integer
df_test_encoded = df_test_clean.copy()
df_test_encoded['date_recorded'] = df_test_encoded['date_recorded'].dt.year
df_test_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 12097 entries, 0 to 14849
Data columns (total 30 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 12097 non-null  int64  
 1   amount_tsh         12097 non-null  float64
 2   date_recorded      12097 non-null  int32  
 3   funder             12097 non-null  object 
 4   gps_height         12097 non-null  int64  
 5   installer          12097 non-null  object 
 6   longitude          12097 non-null  float64
 7   latitude           12097 non-null  float64
 8   wpt_name           12097 non-null  object 
 9   num_private        12097 non-null  int64  
 10  basin              12097 non-null  object 
 11  subvillage         12097 non-null  object 
 12  region             12097 non-null  object 
 13  region_code        12097 non-null  int64  
 14  district_code      12097 non-null  int64  
 15  lga                12097 non-null  object 
 16  ward               12097 non-null  object 
 17  population         12097 non-null  int64  
 18  public_meeting     12097 non-null  object 
 19  scheme_management  12097 non-null  object 
 20  permit             12097 non-null  object 
 21  construction_year  12097 non-null  int64  
 22  extraction_type    12097 non-null  object 
 23  management         12097 non-null  object 
 24  management_group   12097 non-null  object 
 25  payment_type       12097 non-null  object 
 26  water_quality      12097 non-null  object 
 27  quantity_group     12097 non-null  object 
 28  source             12097 non-null  object 
 29  waterpoint_type    12097 non-null  object 
dtypes: float64(3), int32(1), int64(7), object(19)
memory usage: 2.8+ MB
```

```
#having an overview of unique values in the categorical columns to determine type of encoding
for i in categorical_cols:
  print(f'The variable "{i}" has {df_merge_encoded[i].nunique()} variables: {df_merge_encoded[i].unique()} \n')
```

```
The variable "region" has 21 variables: ['Iringa' 'Manyara' 'Mtwara' 'Tanga' 'Shinyanga' 'Tabora' 'Pwani' 'Ruvuma'
 'Kilimanjaro' 'Rukwa' 'Kigoma' 'Lindi' 'Dodoma' 'Mbeya' 'Arusha' 'Mwanza'
 'Kagera' 'Singida' 'Morogoro' 'Mara' 'Dar es Salaam']

The variable "lga" has 117 variables: ['Ludewa' 'Simanjiro' 'Nanyumbu' 'Mkinga' 'Shinyanga Rural' 'Tabora Urban'
 'Mkuranga' 'Namtumbo' 'Maswa' 'Siha' 'Meatu' 'Sumbawanga Rural' 'Njombe'
 'Same' 'Kigoma Rural' 'Moshi Rural' 'Lindi Rural' 'Rombo' 'Chamwino'
 'Bagamoyo' 'Kyela' 'Kondoa' 'Kilolo' 'Kibondo' 'Makete' 'Arusha Rural'
 'Masasi' 'Moshi Urban' 'Geita' 'Bukoba Rural' 'Muheza' 'Lushoto' 'Meru'
 'Iramba' 'Karagwe' 'Kasulu' 'Korogwe' 'Bukombe' 'Morogoro Rural'
 'Kishapu' 'Sengerema' 'Iringa Rural' 'Dodoma Urban' 'Ruangwa' 'Hanang'
 'Misenyi' 'Missungwi' 'Songea Rural' 'Tanga' 'Tunduru' 'Hai' 'Mwanga'
 'Chato' 'Biharamulo' 'Ileje' 'Mpwapwa' 'Mvomero' 'Bunda' 'Kiteto'
 'Urambo' 'Mbozi' 'Sikonge' 'Muleba' 'Temeke' 'Mbeya Rural' 'Magu'
 'Manyoni' 'Igunga' 'Bariadi' 'Kilosa' 'Babati' 'Chunya' 'Mufindi'
 'Mtwara Rural' 'Ngara' 'Karatu' 'Mpanda' 'Kibaha' 'Ukerewe' 'Newala'
 'Nzega' 'Bahi' 'Ulanga' 'Nkasi' 'Sumbawanga Urban' 'Morogoro Urban'
 'Tandahimba' 'Kisarawe' 'Mbinga' 'Liwale' 'Longido' 'Kilombero' 'Uyui'
 'Rufiji' 'Kwimba' 'Ilala' 'Shinyanga Urban' 'Ngorongoro' 'Handeni'
 'Mtwara Urban' 'Rorya' 'Pangani' 'Nachingwea' 'Kilwa' 'Serengeti'
 'Musoma Rural' 'Mbulu' 'Kinondoni' 'Kahama' 'Kigoma Urban' 'Tarime'
 'Ilemela' 'Singida Urban' 'Kilindi' 'Songea Urban' 'Singida Rural'
 'Nyamagana']

The variable "ward" has 1862 variables: ['Mundindi' 'Ngorika' 'Nanyumbu' ... 'Mbinga Urban' 'Jana' 'Ngaya']

The variable "public_meeting" has 2 variables: [True False]

The variable "scheme_management" has 11 variables: ['VWC' 'Private operator' 'WUG' 'Water Board' 'WUA' 'Water authority'
 'Company' 'Other' 'Parastatal' 'Trust' 'SWC']

The variable "permit" has 2 variables: [False True]

The variable "extraction_type" has 18 variables: ['gravity' 'submersible' 'swn 80' 'india mark ii' 'nira/tanira' 'ksb'
 'windmill' 'other' 'afridev' 'other - rope pump' 'mono' 'india mark iii'
 'other - swn 81' 'other - play pump' 'cemo' 'climax' 'walimi'
 'other - mkulima/shinyanga']

The variable "management" has 12 variables: ['vwc' 'private operator' 'wug' 'water board' 'wua' 'company' 'other'
 'water authority' 'parastatal' 'other - school' 'unknown' 'trust']

The variable "management_group" has 5 variables: ['user-group' 'commercial' 'other' 'parastatal' 'unknown']

The variable "payment_type" has 7 variables: ['annually' 'per bucket' 'never pay' 'on failure' 'other' 'monthly'
 'unknown']

The variable "water_quality" has 8 variables: ['soft' 'salty' 'unknown' 'milky' 'fluoride' 'coloured' 'salty abandoned'
 'fluoride abandoned']

The variable "quantity_group" has 5 variables: ['enough' 'dry' 'seasonal' 'insufficient' 'unknown']

The variable "source" has 10 variables: ['spring' 'dam' 'machine dbh' 'other' 'shallow well' 'river' 'hand dtw'
 'rainwater harvesting' 'lake' 'unknown']

The variable "waterpoint_type" has 7 variables: ['communal standpipe' 'communal standpipe multiple' 'hand pump' 'other'
 'improved spring' 'cattle trough' 'dam']

The variable "status_group" has 3 variables: ['functional' 'non functional' 'functional needs repair']
```

Label Encoding

Selecting public_meeting and permit independent variables that are binary

```
#Label encoding for train x data
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
df_merge_encoded['public_meeting_le']=encoder.fit_transform(df_merge_encoded['public_meeting'])
df_merge_encoded['permit_le']=encoder.fit_transform(df_merge_encoded['permit'])
df_merge_encoded['status_group_e']=encoder.fit_transform(df_merge_encoded['status_group'])

df_merge_encoded.drop(columns=['public_meeting','permit','status_group'], inplace=True)


#Label encoding for test x data

encoder = LabelEncoder()
df_test_encoded['public_meeting_le']=encoder.fit_transform(df_test_encoded['public_meeting'])
```

```
df_test_encoded['permit_le']=encoder.fit_transform(df_test_encoded['permit'])

df_test_encoded.drop(columns=['public_meeting','permit'], inplace=True)
```

Frequency Encoding

Selecting region,iga, ward,funder,installer,wpt_name, subvillage and region code independent variables due to high cardinality

```
#frequency encoding for train data
fe_columns =['region','lga','ward','funder','installer','wpt_name','subvillage','region_code','district_code']

for col in fe_columns:
  freq_map = df_merge_encoded[col].value_counts(normalize=True)
  df_merge_encoded[col + '_fe'] = df_merge_encoded[col].map(freq_map)

df_merge_encoded.drop(columns=['region','lga','ward','funder','installer','wpt_name','subvillage','region_code','district_code'], inplace=Tr
```

```
#frequency encoding for test data
fe_columns =['region','lga','ward','funder','installer','wpt_name','subvillage','region_code','district_code']

for col in fe_columns:
  freq_map = df_test_encoded[col].value_counts(normalize=True)
  df_test_encoded[col + '_fe'] = df_test_encoded[col].map(freq_map)

df_test_encoded.drop(columns=['region','lga','ward','funder','installer','wpt_name','subvillage','region_code','district_code'], inplace=Tru
```

One Hot Encoding

Selecting scheme_management,extraction_type, management, management_group, payment_type, water_quality, quantity_group, source,basin and waterpoint_type independent variables due to their high categorical nature

```
#One Hot encoding for train x data
ohe_columns = ['scheme_management','extraction_type', 'management', 'management_group',
                  'payment_type', 'water_quality', 'quantity_group', 'source', 'waterpoint_type','basin']

df_merge_encoded = pd.get_dummies(df_merge_encoded,columns=ohe_columns,drop_first=True)
```

```
#One Hot encoding for test x data
ohe_columns = ['scheme_management','extraction_type', 'management', 'management_group',
                  'payment_type', 'water_quality', 'quantity_group', 'source', 'waterpoint_type','basin']

df_test_encoded = pd.get_dummies(df_test_encoded,columns=ohe_columns,drop_first=True)
```

```
df_merge_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 48285 entries, 0 to 59399
Columns: 103 entries, id to basin_Wami / Ruvu
dtypes: bool(82), float64(12), int32(2), int64(7)
memory usage: 11.5 MB
```

```
df_test_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 12097 entries, 0 to 14849
Columns: 101 entries, id to basin_Wami / Ruvu
dtypes: bool(81), float64(12), int32(1), int64(7)
memory usage: 2.8 MB
```

```
missing_in_dftest = set(df_merge_encoded.columns) - set(df_test_encoded.columns)
missing_in_dftest
```

```
{'extraction_type_other - mkulima/shinyanga', 'status_group_e'}
```

```
df_test_encoded = df_test_encoded.reindex(columns=df_merge_encoded.columns, fill_value=0)
df_test_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 12097 entries, 0 to 14849
Columns: 103 entries, id to basin_Wami / Ruvu
```

```
dtypes: bool(81), float64(12), int32(1), int64(9)
memory usage: 3.0 MB
```

```
column_name = "extraction_type_other - mkulima/shinyanga"

if column_name in df_test_encoded.columns:
    print(f"'{column_name}' exists in the DataFrame.")
else:
    print(f" '{column_name}' is missing from the DataFrame.")
```

```
'extraction_type_other - mkulima/shinyanga' exists in the DataFrame.
```

Scaling

1. Identify what type of scaling to be done based on the type of model
2. Identify distribution of data, check for outliers, check for skewness

Models selected are;

1. Logistic Regression
2. Gradient Boost
3. Random Forest
4. Decision Trees
5. Support Vector Machine
6. K Nearest Neighbour

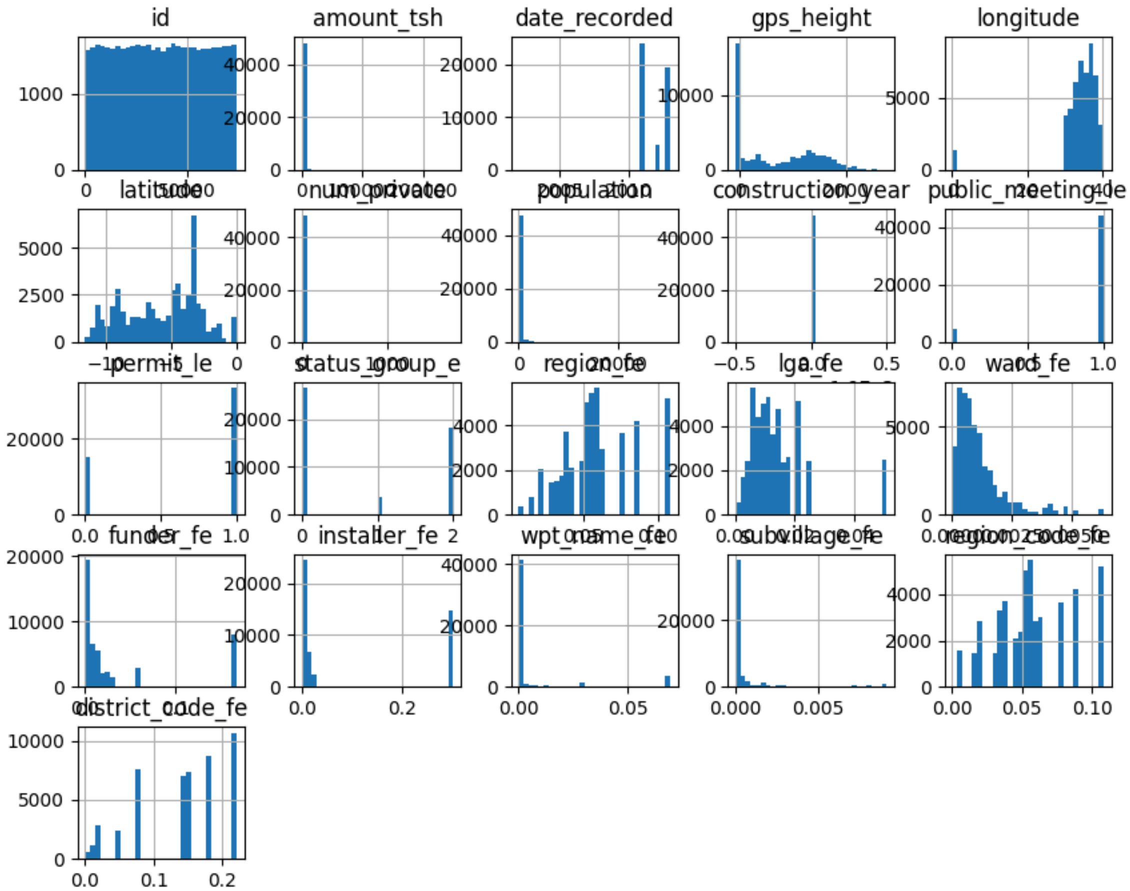
Identifying data properties

i.e. outliers, skewness, distribution

```
#checking for outliers
import numpy as np

df_numeric = df_merge_encoded.select_dtypes(include=['number'])

Q1 = df_numeric.quantile(0.25)
Q3 = df_numeric.quantile(0.75)
IQR = Q3 - Q1

outliers = ((df_numeric < (Q1 - 1.5 * IQR)) | (df_numeric > (Q3 + 1.5 * IQR))).sum()
print(outliers.sort_values(ascending=False))
```

```
wpt_name_fe          10116
amount_tsh            9535
funder_fe             8080
subvillage_fe         7683
public_meeting_le     4348
population            3617
ward_fe               3232
lga_fe                2455
longitude             1326
num_private            721
date_recorded           21
id                       0
region_code_fe           0
installer_fe             0
permit_le                0
region_fe                0
status_group_e           0
construction_year        0
latitude                 0
gps_height               0
district_code_fe         0
dtype: int64
```

```
#checking distribution of data
df_merge_encoded.iloc[:, :500].hist(figsize=(10, 8), bins=30)  # Checking first 5 columns
plt.show()
```




```
from scipy.stats import shapiro, normaltest
def check_normality(df, alpha=0.05):
    normal_cols = []
    non_normal_cols = []

    for col in df_merge_encoded.select_dtypes(include=['float64', 'int64']).columns:
        data = df_merge_encoded[col].dropna()


        stat, p = normaltest(data)

        if p > alpha:
            normal_cols.append(col)
        else:
            non_normal_cols.append(col)

    return normal_cols, non_normal_cols


normal_cols, non_normal_cols = check_normality(df_merge_encoded)


print("Normally Distributed Columns:\n", normal_cols)
print("\nNon-Normally Distributed Columns:\n", non_normal_cols)
```

```
Normally Distributed Columns:
 []

Non-Normally Distributed Columns:
 ['id', 'amount_tsh', 'gps_height', 'longitude', 'latitude', 'num_private', 'population', 'public_meeting_le', 'permit_le', 'status_grou
```

```
def categorize_skewness(df):
    skew_categories = {
        "Right Skewed": [],
        "Moderate Right Skewed": [],
        "Normal": [],
        "Moderate Left Skewed": [],
        "Left Skewed": []
```

```
    }


    excluded_suffixes = ('_fe', '_le')
    numeric_cols = [col for col in df_merge_encoded.select_dtypes(include=['number']).columns if not col.endswith(excluded_suffixes)]


    skewness_values = df_merge_encoded[numeric_cols].skew()

    for col, skew in skewness_values.items():
        if skew > 1.5:
            skew_categories["Right Skewed"].append(col)
        elif 1 < skew <= 1.5:
            skew_categories["Moderate Right Skewed"].append(col)
        elif -1 <= skew <= 1:
            skew_categories["Normal"].append(col)
        elif -1.5 <= skew < -1:
            skew_categories["Moderate Left Skewed"].append(col)
        elif skew < -1.5:
            skew_categories["Left Skewed"].append(col)

    return skew_categories


skew_results = categorize_skewness(df_merge_encoded)
for category, cols in skew_results.items():
    print(f"{category}: {cols}")
```

```
Right Skewed: ['amount_tsh', 'num_private', 'population']
Moderate Right Skewed: []
Normal: ['id', 'date_recorded', 'gps_height', 'latitude', 'construction_year', 'status_group_e']
Moderate Left Skewed: []
Left Skewed: ['longitude']
```

```
#skewness for test x data
def categorize_skewness(df):
    skew_categories = {
        "Right Skewed": [],
        "Moderate Right Skewed": [],
        "Normal": [],
        "Moderate Left Skewed": [],
        "Left Skewed": []
    }


    excluded_suffixes = ('_fe', '_le')
    numeric_cols = [col for col in df_test_encoded.select_dtypes(include=['number']).columns if not col.endswith(excluded_suffixes)]

# Calculate skewness for each column
    skewness_values = df_test_encoded[numeric_cols].skew()

    for col, skew in skewness_values.items():
        if skew > 1.5:
            skew_categories["Right Skewed"].append(col)
        elif 1 < skew <= 1.5:
            skew_categories["Moderate Right Skewed"].append(col)
        elif -1 <= skew <= 1:
            skew_categories["Normal"].append(col)
        elif -1.5 <= skew < -1:
            skew_categories["Moderate Left Skewed"].append(col)
        elif skew < -1.5:
            skew_categories["Left Skewed"].append(col)

    return skew_categories


skew_results = categorize_skewness(df_test_encoded)
for category, cols in skew_results.items():
    print(f"{category}: {cols}")
```

```
Right Skewed: ['amount_tsh', 'num_private', 'population']
Moderate Right Skewed: []
Normal: ['id', 'date_recorded', 'gps_height', 'latitude', 'construction_year', 'status_group_e', 'extraction_type_other - mkulima/shinya
Moderate Left Skewed: []
Left Skewed: ['longitude']
```

columns with frequency encoding have introduced skewness however we can just apply Standard Scaler directly

```
#finding columns with high outliers train x data
def find_outlier_columns(df_merge_encoded, threshold=0.05):


    outlier_cols = []
    num_rows = df_merge_encoded.shape[0]

    for col in df_merge_encoded.select_dtypes(include=['number']).columns:
        Q1 = df_merge_encoded[col].quantile(0.25)
        Q3 = df_merge_encoded[col].quantile(0.75)
        IQR = Q3 - Q1


        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR


        outliers = ((df_merge_encoded[col] < lower_bound) | (df_merge_encoded[col] > upper_bound)).sum()
        outlier_fraction = outliers / num_rows

        if outlier_fraction > threshold:
            outlier_cols.append(col)

    return outlier_cols


outlier_columns = find_outlier_columns(df_merge_encoded, threshold=0.05)
print("Columns with many outliers:", outlier_columns)
```

```
Columns with many outliers: ['amount_tsh', 'population', 'public_meeting_le', 'lga_fe', 'ward_fe', 'funder_fe', 'wpt_name_fe', 'subvilla
```

```
#finding columns with high outliers test x data
def find_outlier_columns(df_test_encoded, threshold=0.05):


    outlier_cols = []
    num_rows = df_test_encoded.shape[0]

    for col in df_test_encoded.select_dtypes(include=['number']).columns:
        Q1 = df_test_encoded[col].quantile(0.25)
        Q3 = df_test_encoded[col].quantile(0.75)
        IQR = Q3 - Q1


        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR


        outliers = ((df_test_encoded[col] < lower_bound) | (df_test_encoded[col] > upper_bound)).sum()
        outlier_fraction = outliers / num_rows

        if outlier_fraction > threshold:
            outlier_cols.append(col)

    return outlier_cols


outlier_columns = find_outlier_columns(df_test_encoded, threshold=0.05)
print("Columns with many outliers:", outlier_columns)
```

```
Columns with many outliers: ['amount_tsh', 'population', 'public_meeting_le', 'lga_fe', 'funder_fe', 'wpt_name_fe', 'subvillage_fe']
```

Due to large number of outliers, robust scaler is recommended for columns that have not been encoded i.e. amount_tsh and population

Robust and Standard Scaling

```
#scaling train x data
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.model_selection import train_test_split
```

```
df_merge_scaled = df_merge_encoded.copy()

X =df_merge_scaled.drop(columns = ['id','date_recorded','status_group_e'])
y = df_merge_scaled['status_group_e']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

robust_cols = ['amount_tsh', 'population','num_private']
standard_cols = [col for col in X_train.columns if col not in robust_cols]

# Applying RobustScaler to selected columns
robust_scaler = RobustScaler()
X_train_robust = robust_scaler.fit_transform(X_train[robust_cols])
X_test_robust= robust_scaler.fit_transform(X_test[robust_cols])

# Applying StandardScaler to the remaining columns
standard_scaler = StandardScaler()
X_train_standard= standard_scaler.fit_transform(X_train[standard_cols])
X_test_standard= robust_scaler.fit_transform(X_test[standard_cols])

X_train_scaled = pd.DataFrame(np.hstack((X_train_robust, X_train_standard)),
                              columns=robust_cols + standard_cols)

X_test_scaled = pd.DataFrame(np.hstack((X_test_robust, X_test_standard)),
                             columns=robust_cols + standard_cols)
```

```
X_train_scaled
```

	amount_tsh	population	num_private	gps_height	longitude	latitude	construction_year	public_meeting_le	permit_le	region_fe
0	0.0	-0.150	0.0	-1.004609	-0.344520	0.277306	0.0	0.312667	0.670090	-0.856663
1	1.0	20.890	0.0	-0.868622	0.791893	-0.492238	0.0	-3.198292	-1.492336	-0.316043
2	20.0	-0.145	0.0	-0.118546	0.421787	-1.932584	0.0	0.312667	-1.492336	-0.638865
3	0.0	0.025	0.0	0.340946	0.501852	0.828550	0.0	-3.198292	0.670090	1.160333
4	10.0	2.350	0.0	0.864854	-0.035164	0.927622	0.0	0.312667	-1.492336	0.707520
...	...	...	...	...	...	...	...	...	...	...
38623	0.0	1.100	0.0	0.975075	-0.538832	0.943789	0.0	0.312667	-1.492336	-0.071558
38624	0.0	-0.150	0.0	-1.004609	-0.196283	-0.982713	0.0	0.312667	0.670090	-0.139566
38625	10.0	0.130	0.0	1.179771	0.082701	-1.099101	0.0	0.312667	0.670090	2.034106
38626	20.0	1.350	0.0	-0.562293	0.338323	-1.073837	0.0	0.312667	0.670090	0.082536
38627	0.0	-0.150	0.0	-1.004609	-0.427130	1.324159	0.0	0.312667	-1.492336	-0.010437

38628 rows × 100 columns

```
X_test_scaled
```

	amount_tsh	population	num_private	gps_height	longitude	latitude	construction_year	public_meeting_le	permit_le	region_fe
0	0.0	-0.125	0.0	-0.350376	-0.177683	0.302437	0.0	0.0	0.0	0.578645
1	0.0	-0.125	0.0	-0.350376	-0.463677	-0.838118	0.0	0.0	-1.0	-0.050512
2	0.0	0.875	0.0	0.899248	0.358518	0.350763	0.0	0.0	0.0	0.039642
3	0.0	-0.125	0.0	-0.350376	-0.428500	0.049607	0.0	-1.0	-1.0	-0.583120
4	0.0	0.375	0.0	0.887970	0.534666	0.350981	0.0	0.0	0.0	0.914962
...	...	...	...	...	...	...	...	...	...	...
9652	0.0	-0.125	0.0	-0.350376	0.271587	-0.327866	0.0	0.0	0.0	-0.603581
9653	0.0	6.375	0.0	0.824812	-1.230782	0.090189	0.0	0.0	0.0	0.000000
9654	0.0	-0.125	0.0	-0.350376	0.236939	-0.241967	0.0	0.0	0.0	-0.603581
9655	0.0	19.875	0.0	-0.300000	0.945177	-0.415377	0.0	0.0	-1.0	-0.181586
9656	0.0	0.875	0.0	-0.299248	0.885701	-0.476808	0.0	0.0	-1.0	-0.181586

9657 rows × 100 columns

Modeling

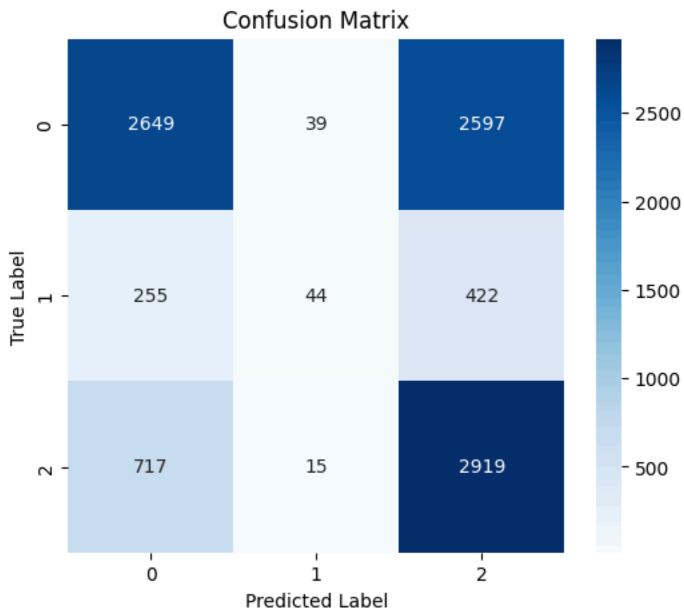
Classification

Logistic Regression

Base Model

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,precision_score,recall_score,f1_score


log_reg = LogisticRegression()
log_reg.fit(X_train_scaled, y_train)

y_pred = log_reg.predict(X_test_scaled)



accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
recall = recall_score(y_test, y_pred,average='macro')
precision = precision_score(y_test, y_pred,average='macro')
f1 = f1_score(y_test, y_pred,average='macro')


print(f"Accuracy: {accuracy:.4f}")
print(f"Recall: {recall}")
print(f"Precision: {precision}")
print(f"F1-score: {f1}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))

conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6,5))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.5811
Recall: 0.45392107753607
Precision: 0.557375038023217
F1-score: 0.43705019098817627

Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.50      0.59      5285
           1       0.45      0.06      0.11       721
           2       0.49      0.80      0.61      3651

    accuracy                           0.58      9657
   macro avg       0.56      0.45      0.44      9657
weighted avg       0.62      0.58      0.56      9657


Confusion Matrix:
 [[2649   39 2597]
 [ 255   44  422]
 [ 717   15 2919]]
```



The base model has performed poorly with an average overall performance of 61% accuracy, precision of 46% , recall of 49% and f1-score of 47%. This below half which means the modeling is not accurate at all.

From the classification report we see that class 0 having the highest weight has better predictability than class 1 and 2. Class 1 has very low performance due to class imbalance, therefore some feature selection or class balancing should be performed

From the confusion matrix, the class 0 is being confused with class 2 due to closeness in weights.

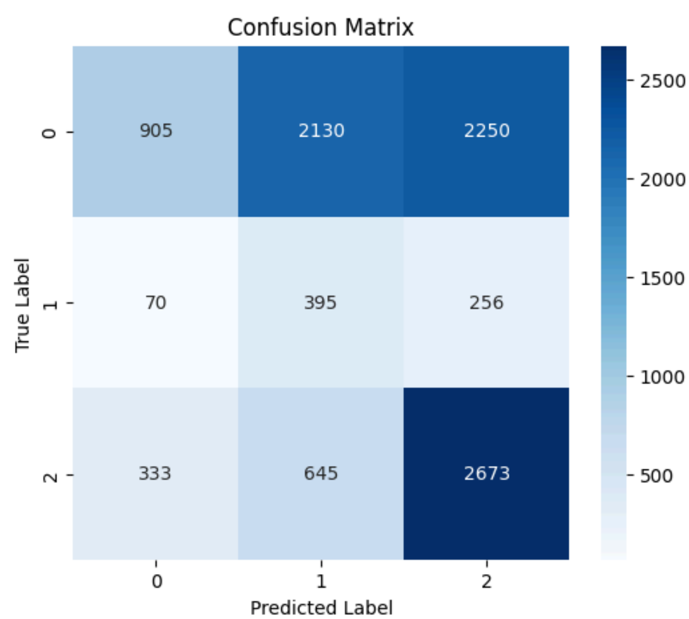
In general this model is not reliable and needs tuning

Tuned Model

```
#Applying SMOTE to tune the model
from imblearn.over_sampling import SMOTE
from sklearn.datasets import make_classification

X, y = make_classification(n_classes=3, weights=[0.1, 0.9, 0.2],
                           n_samples=5000, n_features=100,  # Keep all 100 features
                           n_informative=6, random_state=42)


smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_train)


log_reg = LogisticRegression(max_iter=500, solver='lbfgs', multi_class='multinomial', random_state=42)
log_reg.fit(X_train_resampled, y_train_resampled)
```

```
y_pred = log_reg.predict(X_test_scaled)


accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
recall = recall_score(y_test, y_pred,average='macro')
precision = precision_score(y_test, y_pred,average='macro')
f1 = f1_score(y_test, y_pred,average='macro')


print(f"Accuracy: {accuracy:.4f}")
print(f"Recall: {recall}")
print(f"Precision: {precision}")
print(f"F1-score: {f1}")
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)


plt.figure(figsize=(6,5))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.4114
Recall: 0.48373924959112163
Precision: 0.44420816877610486
F1-score: 0.36100074975092705
Confusion Matrix:
 [[ 905 2130 2250]
 [  70  395  256]
 [ 333  645 2673]]
Classification Report:
               precision    recall  f1-score   support

           0       0.69      0.17      0.27      5285
           1       0.12      0.55      0.20       721
           2       0.52      0.73      0.61      3651

    accuracy                           0.41      9657
   macro avg       0.44      0.48      0.36      9657
weighted avg       0.58      0.41      0.39      9657
```

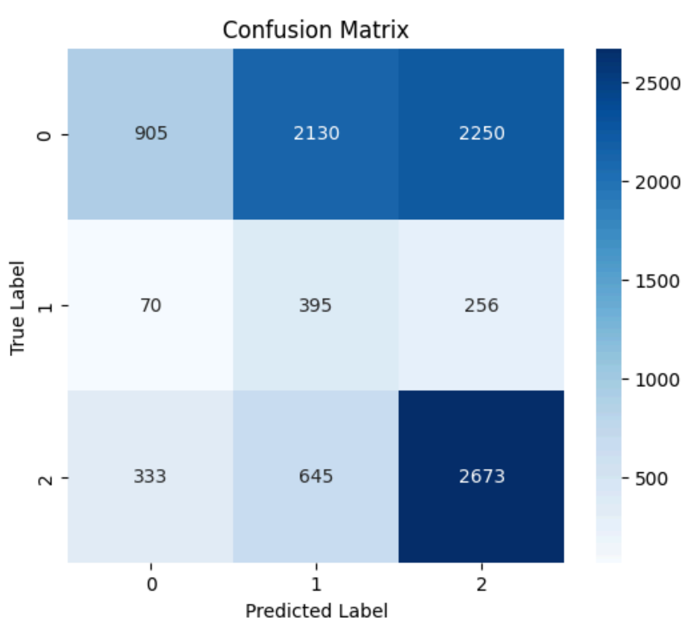


The SMOTE method does not seem to improve the model has lower performance on all metrics Feature selection may help improve the model

```
#Feature selction using SelectK Best Anova Method
from sklearn.feature_selection import SelectKBest, f_classif,RFE
```

```
anova_selector = SelectKBest(score_func=f_classif, k=10)
X_train_anova = anova_selector.fit_transform(X_train_scaled, y_train)
X_test_anova = anova_selector.transform(X_test_scaled)

anova_columns =  X_train_scaled.columns[anova_selector.get_support()]
print("ANOVA Selected Features:", anova_columns.tolist())

log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train_anova, y_train)
y_pred_anova = log_reg.predict(X_test_anova)

recall = recall_score(y_test, y_pred,average='macro')
precision = precision_score(y_test, y_pred,average='macro')
f1 = f1_score(y_test, y_pred,average='macro')

print("ANOVA Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_anova))
print(f"Recall: {recall}")
print(f"Precision: {precision}")
print(f"F1-score: {f1}")
print(classification_report(y_test, y_pred_anova))

plt.figure(figsize=(6,5))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
ANOVA Selected Features: ['region_fe', 'lga_fe', 'ward_fe', 'region_code_fe', 'extraction_type_gravity', 'extraction_type_other', 'payme
ANOVA Logistic Regression Accuracy: 0.6293880086983535
Recall: 0.48373924959112163
Precision: 0.44420816877610486
F1-score: 0.36100074975092705
              precision    recall  f1-score   support

           0       0.68      0.70      0.69      5285
           1       0.00      0.00      0.00       721
           2       0.56      0.65      0.61      3651

    accuracy                           0.63      9657
   macro avg       0.41      0.45      0.43      9657
weighted avg       0.59      0.63      0.61      9657
```



ANOVA seems to have a slightly improved metrics especially on accuracy but this does not seem like a good measure to focus on since the precision, accuracy and recall are still below 50% however it performs way better than the SMOTE

Therefore Logistic Regression does not seem to be a good model fit for this dataset. Reason being it is used to work on simpler datasets with fewer features

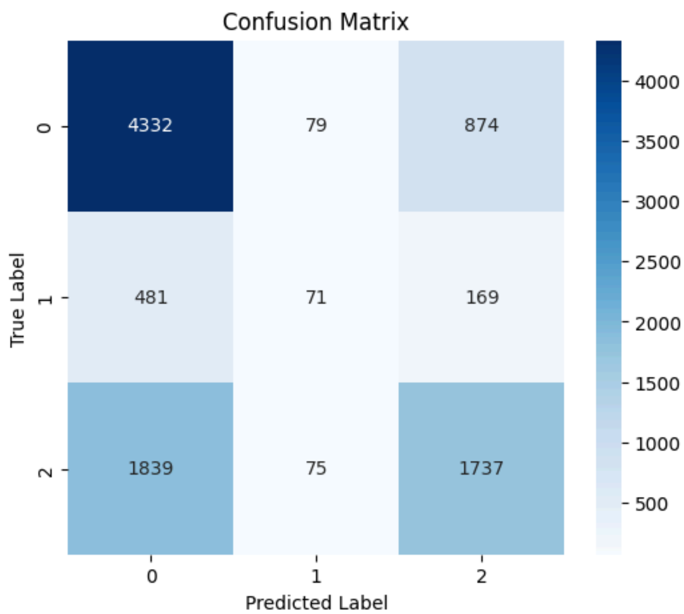
Tune the logistic model

K-NN model

Base model

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=21)


knn.fit(X_train_scaled, y_train)


y_pred = knn.predict(X_test_scaled)


print("KNN Accuracy:", accuracy_score(y_test, y_pred))
print(f"Recall: {recall}")
print(f"Precision: {precision}")
print(f"F1-score: {f1}")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))


conf_matrix = confusion_matrix(y_test, y_pred)
recall = recall_score(y_test, y_pred,average='macro')
precision = precision_score(y_test, y_pred,average='macro')
f1 = f1_score(y_test, y_pred,average='macro')


plt.figure(figsize=(6,5))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
KNN Accuracy: 0.6358082220151186
Recall: 0.48373924959112163
Precision: 0.44420816877610486
F1-score: 0.36100074975092705
Confusion Matrix:
 [[4332   79  874]
 [ 481   71  169]
 [1839   75 1737]]
Classification Report:
               precision    recall  f1-score   support

           0       0.65      0.82      0.73      5285
           1       0.32      0.10      0.15       721
           2       0.62      0.48      0.54      3651

    accuracy                           0.64      9657
   macro avg       0.53      0.46      0.47      9657
weighted avg       0.62      0.64      0.61      9657
```



The KNN model seems to be performing similar to the Logistic Regression Model with feature selection from ANOVA. The values of accuracy being 63%, precision 46%, recall 51%, f1 score 47% on an average basis

In terms of the individual performance we see an improvement on the class 1 performance with atleast a 15% performance as compared to the ANOVA Logistic Regression which was 0% however this is still not satisfactory

In terms of the confusion matrix, the model still confuses a lot of class 2 and class 0 predictions but has much better performance than Logistic Regression. This may be due to the use of a higher K which improves the classification

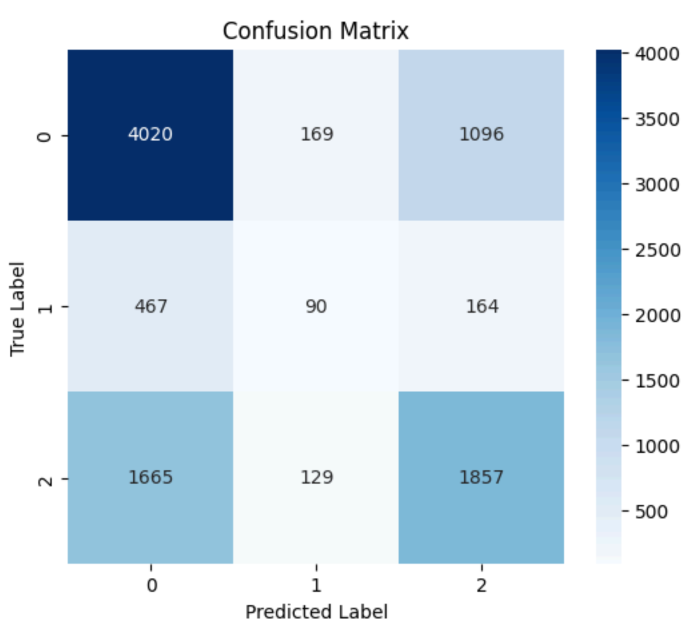
Tuned model

```
#selecting best K through cross validation
from sklearn.model_selection import cross_val_score
import numpy as np
from sklearn.neighbors import KNeighborsClassifier


k_values = range(1, 50, 4)
cv_scores = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train_scaled, y_train, cv=5)  # 5-fold cross-validation
    cv_scores.append(scores.mean())

best_k = k_values[np.argmax(cv_scores)]
print(f"Best K: {best_k}")
```

```
Best K: 5
```

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5)


knn.fit(X_train_scaled, y_train)


y_pred = knn.predict(X_test_scaled)


print("KNN Accuracy:", accuracy_score(y_test, y_pred))
print(f"Recall: {recall}")
print(f"Precision: {precision}")
print(f"F1-score: {f1}")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))


conf_matrix = confusion_matrix(y_test, y_pred)
recall = recall_score(y_test, y_pred,average='macro')
precision = precision_score(y_test, y_pred,average='macro')
f1 = f1_score(y_test, y_pred,average='macro')


plt.figure(figsize=(6,5))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
KNN Accuracy: 0.6178937558247903
Recall: 0.4646375806127752
Precision: 0.5305361371355911
F1-score: 0.4720373797936029
Confusion Matrix:
 [[4020  169 1096]
 [ 467   90  164]
 [1665  129 1857]]
Classification Report:
               precision    recall  f1-score   support

           0       0.65      0.76      0.70      5285
           1       0.23      0.12      0.16       721
           2       0.60      0.51      0.55      3651

    accuracy                           0.62      9657
   macro avg       0.49      0.46      0.47      9657
weighted avg       0.60      0.62      0.60      9657
```

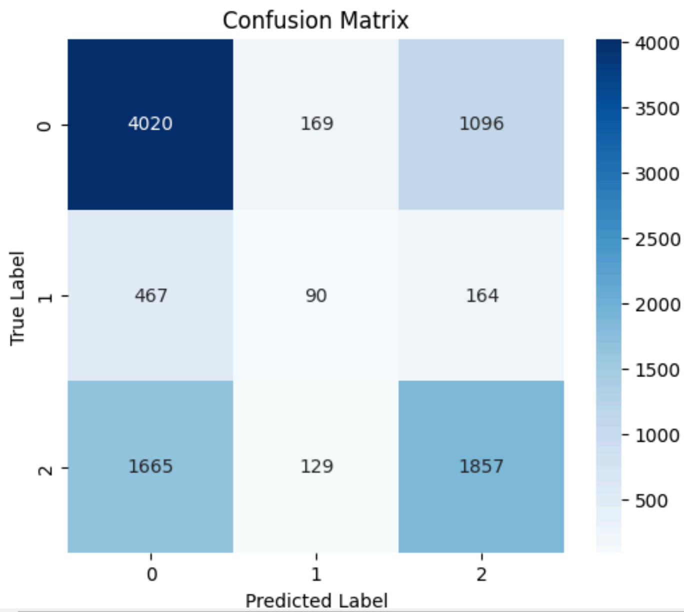


K=5 seems to be performing worse than K = 21 but it may be the best K which avoids over or under fitting and gives the balance between variance and bias.

We may need to do feature selection to improve the model

```
#Using ANOVA feature selection due to its better performance in Logistic Regression
from sklearn.neighbors import KNeighborsClassifier


k =  7
anova_selector = SelectKBest(score_func=f_classif, k=k)
X_train_selected = anova_selector.fit_transform(X_train_scaled, y_train)
X_test_selected = anova_selector.transform(X_test_scaled)


knn = KNeighborsClassifier(n_neighbors=5, weights='uniform', metric='euclidean')
knn.fit(X_train_scaled, y_train)


y_pred = knn.predict(X_test_scaled)


accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred,average='macro')
precision = precision_score(y_test, y_pred,average='macro')
f1 = f1_score(y_test, y_pred,average='macro')
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.4f}")
print(f"Recall: {recall}")
print(f"Precision: {precision}")
print(f"F1-score: {f1}")
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)


selected_features = X_train_scaled.columns[anova_selector.get_support()]
print("Selected Features:", selected_features.tolist())

plt.figure(figsize=(6,5))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.6179
Recall: 0.46469924435785703
Precision: 0.49372331850109524
F1-score: 0.4713496007625198
Confusion Matrix:
 [[4020  169 1096]
 [ 467   90  164]
 [1665  129 1857]]
Classification Report:
               precision    recall  f1-score   support

           0       0.65      0.76      0.70      5285
           1       0.23      0.12      0.16       721
           2       0.60      0.51      0.55      3651

    accuracy                           0.62      9657
   macro avg       0.49      0.46      0.47      9657
weighted avg       0.60      0.62      0.60      9657

Selected Features: ['region_fe', 'region_code_fe', 'extraction_type_gravity', 'extraction_type_other', 'payment_type_never pay', 'quanti
```



Anova feature selection has no difference from the base model even with fewer best features therefore there is no effect

KNN may not be the best model. it still needs to work with a smaller dataset with fewer features

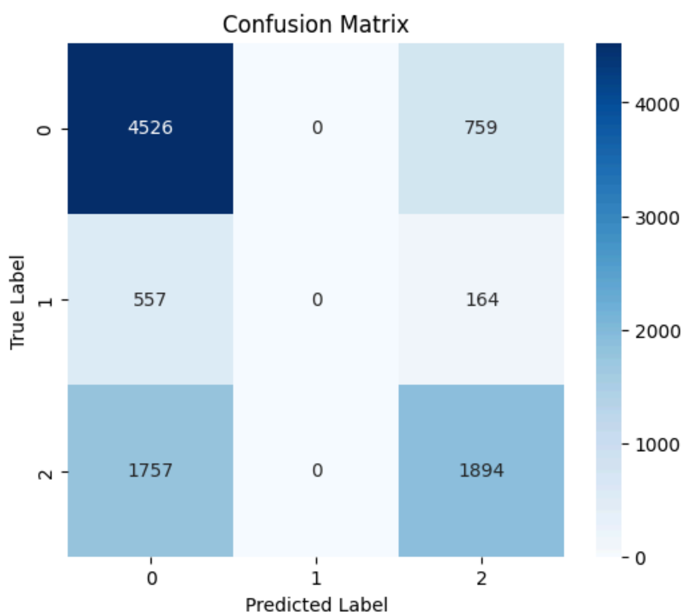
Support Vector Machine Model

Base Model

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


svm = SVC(kernel='rbf', C=1.0, gamma='scale')
svm.fit(X_train_scaled, y_train)


y_pred = svm.predict(X_test_scaled)


accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
recall = recall_score(y_test, y_pred,average='macro')
precision = precision_score(y_test, y_pred,average='macro')
f1 = f1_score(y_test, y_pred,average='macro')

print(f"Accuracy: {accuracy:.4f}")
print(f"Recall: {recall}")
```

```
print(f"Precision: {precision}")
print(f"F1-score: {f1}")
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)

plt.figure(figsize=(6,5))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.6648
Recall: 0.45838266037540115
Precision: 0.44468079143545763
F1-score: 0.4440697146087426
Confusion Matrix:
 [[4526    0  759]
 [ 557    0  164]
 [1757    0 1894]]
Classification Report:
               precision    recall  f1-score   support

           0       0.66      0.86      0.75      5285
           1       0.00      0.00      0.00       721
           2       0.67      0.52      0.59      3651

    accuracy                           0.66      9657
   macro avg       0.44      0.46      0.44      9657
weighted avg       0.62      0.66      0.63      9657
```



Support Vector Machine seems to be performing better than kNN with a better accuracy and with better values on the confusion matrix

Has performed poorly on class 1 and worse on the metrics on precision, recall and f1-score.

Parameter tuning and weights could help improve the model.

Tuned Model

```
from sklearn.feature_selection import SelectKBest, chi2, f_classif,RFE
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

k = 10
anova_selector = SelectKBest(score_func=f_classif, k=k)
X_train_selected = anova_selector.fit_transform(X_train_scaled, y_train)
X_test_selected = anova_selector.transform(X_test_scaled)


svm = SVC(kernel='rbf', C=1.0, gamma='scale')
```

```
svm.fit(X_train_selected, y_train)


y_pred = svm.predict(X_test_selected)


accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
recall = recall_score(y_test, y_pred,average='macro')
precision = precision_score(y_test, y_pred,average='macro')
f1 = f1_score(y_test, y_pred,average='macro')

print(f"Accuracy: {accuracy:.4f}")
print(f"Recall: {recall}")
print(f"Precision: {precision}")
print(f"F1-score: {f1}")
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)


selected_features = X_train_scaled.columns[anova_selector.get_support()]
print("Selected Features:", selected_features.tolist())
```

```
Accuracy: 0.6428
Recall: 0.4472696921852646
Precision: 0.42510239408500156
F1-score: 0.4327036808980605
Confusion Matrix:
 [[4234    0 1051]
 [ 575    0  146]
 [1677    0 1974]]
Classification Report:
               precision    recall  f1-score   support

           0       0.65      0.80      0.72      5285
           1       0.00      0.00      0.00       721
           2       0.62      0.54      0.58      3651

    accuracy                           0.64      9657
   macro avg       0.43      0.45      0.43      9657
weighted avg       0.59      0.64      0.61      9657

Selected Features: ['region_fe', 'lga_fe', 'ward_fe', 'region_code_fe', 'extraction_type_gravity', 'extraction_type_other', 'payment_typ
```

ANOVA does not seem to improve the model therefore the base model is better in accuracy terms only compared to other previous models

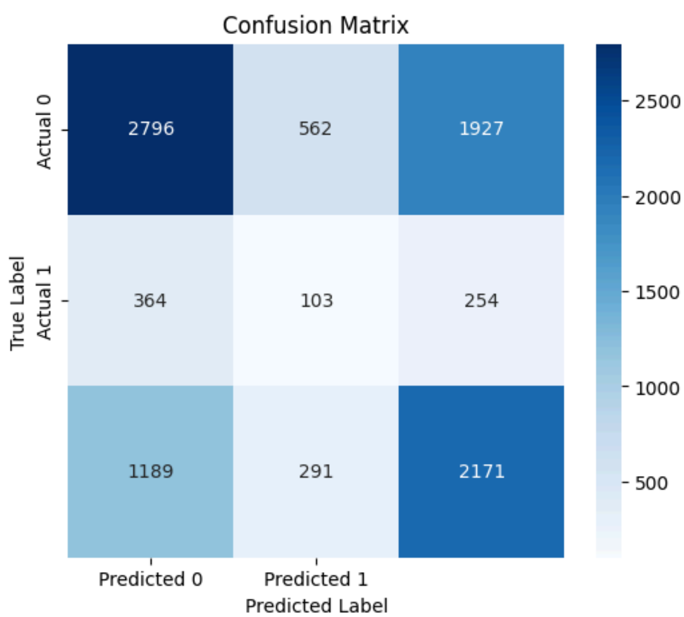
Decision tree

Base Model

```
#Decision Trees model with scaled data

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,recall_score,precision_score,f1_score


dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train_scaled, y_train)


y_pred = dt_model.predict(X_test_scaled)




accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred,average='macro')
precision = precision_score(y_test, y_pred,average='macro')
f1 = f1_score(y_test, y_pred,average='macro')

print(f"Accuracy: {accuracy}")
print(f"Recall: {recall}")
```

```
print(f"Precision: {precision}")
print(f"F1-score: {f1}")


print(classification_report(y_test, y_pred))


cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Accuracy: 0.5250077663870767
Recall: 0.4221777387013801
Precision: 0.4164993679943709
F1-score: 0.415276402288862
              precision    recall  f1-score   support

           0       0.64      0.53      0.58      5285
           1       0.11      0.14      0.12       721
           2       0.50      0.59      0.54      3651

    accuracy                           0.53      9657
   macro avg       0.42      0.42      0.42      9657
weighted avg       0.55      0.53      0.53      9657
```

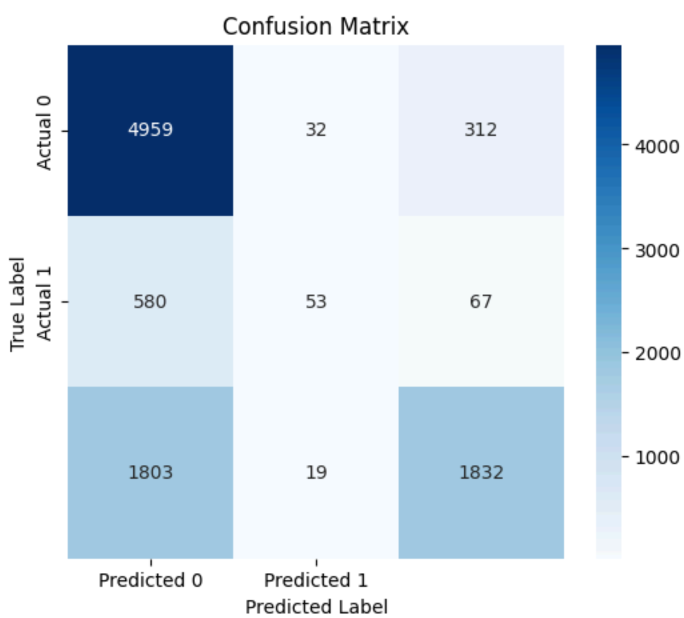


Deision trees performs poorly with scaled data and worse on all other metrics

```
#Decision trees without scaled data
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


X = df_merge_encoded.drop(columns=['id','date_recorded','status_group_e'])
y = df_merge_encoded["status_group_e"]


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)


dt_model = DecisionTreeClassifier(criterion="gini", max_depth=5, random_state=42)
dt_model.fit(X_train, y_train)


y_pred = dt_model.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
recall = recall_score(y_test, y_pred,average='macro')
precision = precision_score(y_test, y_pred,average='macro')
f1 = f1_score(y_test, y_pred,average='macro')


print(f"Accuracy: {accuracy:.4f}")
print(f"Recall: {recall}")
print(f"Precision: {precision}")
print(f"F1-score: {f1}")
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Accuracy: 0.7087
Recall: 0.5040712356811247
Precision: 0.6712095913323161
F1-score: 0.513635121893966
Confusion Matrix:
 [[4959   32  312]
 [ 580   53   67]
 [1803   19 1832]]
Classification Report:
               precision    recall  f1-score   support

           0       0.68      0.94      0.78      5303
           1       0.51      0.08      0.13       700
           2       0.83      0.50      0.62      3654

    accuracy                           0.71      9657
   macro avg       0.67      0.50      0.51      9657
weighted avg       0.72      0.71      0.68      9657
```



Model performs significantly better without scaled data with an accuracy of 70%. So far the best performing model

Precision, recall are above 50% which is more improved however still poor, with precision having a higher percentage

Class 1 prediction is significantly improved in this model compared to the others

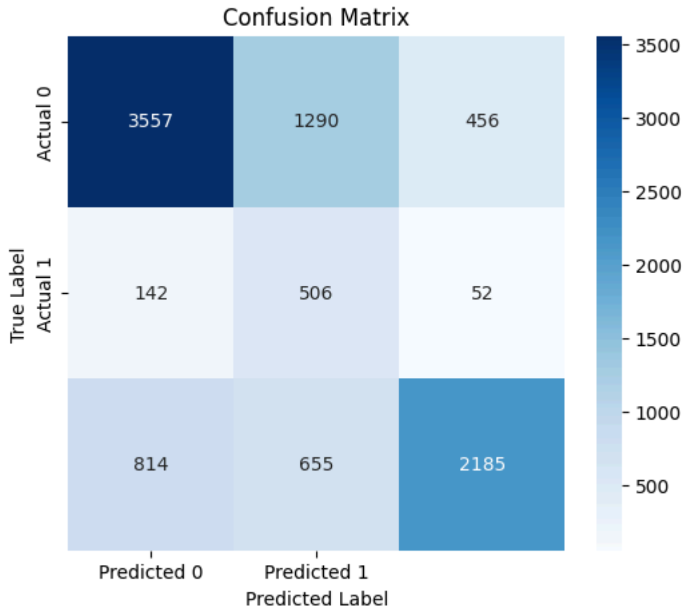
Lesser confusion between class 0 and 2

Hyper parameter tuning may improve the model

Tuned Model

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    "criterion": ["gini", "entropy"],
    "max_depth": [3, 5, 10, None],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 5]
}


dt_model_tuned = DecisionTreeClassifier(random_state=42)


grid_search = GridSearchCV(dt_model_tuned, param_grid, cv=5, scoring="accuracy", n_jobs=-1)
grid_search.fit(X_train, y_train)


best_dt_model = grid_search.best_estimator_


y_pred_dt_tuned = best_dt_model.predict(X_test)


accuracy_dtt = accuracy_score(y_test, y_pred)
conf_matrix_dtt = confusion_matrix(y_test, y_pred)
class_report_dtt= classification_report(y_test, y_pred)
recall = recall_score(y_test, y_pred,average='macro')
precision = precision_score(y_test, y_pred,average='macro')
f1 = f1_score(y_test, y_pred,average='macro')


print("Best Parameters:", grid_search.best_params_)
print(f"Accuracy: {accuracy:.4f}")
print(f"Recall: {recall}")
print(f"Precision: {precision}")
print(f"F1-score: {f1}")
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Best Parameters: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 5, 'min_samples_split': 2}
Accuracy: 0.7087
Recall: 0.6638614564231164
Precision: 0.6019922189729064
F1-score: 0.5781390894136491
Confusion Matrix:
 [[4959   32  312]
 [ 580   53   67]
 [1803   19 1832]]
Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.94      0.78      5303
           1       0.51      0.08      0.13       700
           2       0.83      0.50      0.62      3654

    accuracy                           0.71      9657
   macro avg       0.67      0.50      0.51      9657
weighted avg       0.72      0.71      0.68      9657
```



Through hyper parameter tuning we have identified that gini is the best criterion to be used to classify the decision trees. No particular max depth and min sample split of 5 and leaves of 2.

By specifying the tree depth, minimum sample split and minimum leaves, the accuracy seems to have dropped but the most important parameters being precision recall and f1 score have significantly improved to an average of 57% considering f1 score

The confusion of class 2 and 0 has significantly reduced

This model can be considered however still needs large improvement to reach desirable values

Model performance worse after tuning, the data maybe overfitting due to many columns

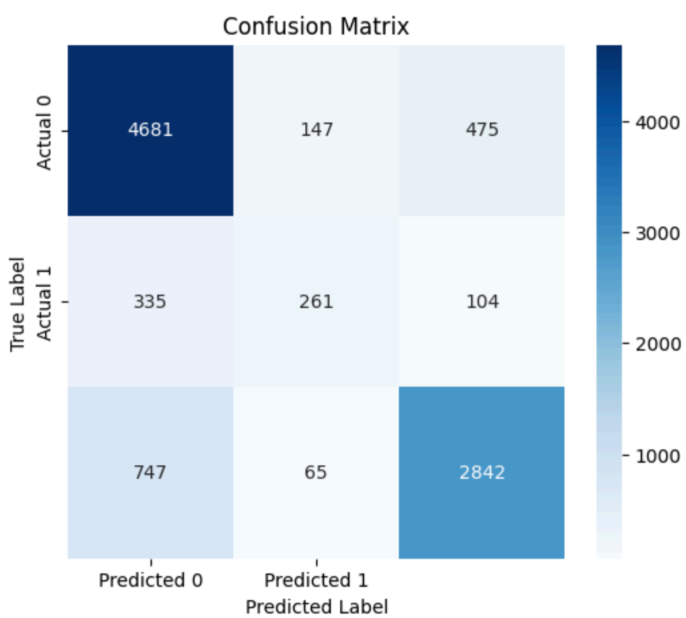
Random Forest

Base Model

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)


y_pred = rf_model.predict(X_test)

recall = recall_score(y_test, y_pred,average='macro')
precision = precision_score(y_test, y_pred,average='macro')
f1 = f1_score(y_test, y_pred,average='macro')
```

```
print("Accuracy:", accuracy_score(y_test, y_pred))
print(f"Recall: {recall}")
print(f"Precision: {precision}")
print(f"F1-score: {f1}")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'])
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Accuracy: 0.806047426737082
Recall: 0.6777809406076424
Precision: 0.7315996154794431
F1-score: 0.6981399446806948
Confusion Matrix:
 [[4681  147  475]
 [ 335  261  104]
 [ 747   65 2842]]
Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.88      0.85      5303
           1       0.55      0.37      0.45       700
           2       0.83      0.78      0.80      3654

    accuracy                           0.81      9657
   macro avg       0.73      0.68      0.70      9657
weighted avg       0.80      0.81      0.80      9657
```



This is the best performing model with an accuracy of 80% and metrics of precision, recall and f1 score all above 67% which is highly desirable

Class 2 still has more confusion with class 0 but in terms of prediction class 0 is performing better

Class 1 has an overall f1 score of 45% which is better than all other models

Tuned Model

```
from sklearn.metrics import classification_report, confusion_matrix


smote = SMOTE(sampling_strategy={1: int(np.sum(y == 1) * 2)}, random_state=42)
```

```
    X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)


    rf = RandomForestClassifier(
        n_estimators=300,
        max_depth=12,
        min_samples_split=5,
        min_samples_leaf=2,
        max_features='sqrt',
```