In [1]:
```python
from custom_class import CustomCrossValidator
from tools import *

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression, Lasso
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE, VarianceThreshold
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.tree import DecisionTreeClassifier
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import recall_score, confusion_matrix

import folium
from folium.plugins import MarkerCluster
```

In [2]:
```python
# Import the split data set
target_df = pd.read_csv("data/target_columns.csv")
independent_df = pd.read_csv("data/independent_columns.csv")
```

In [3]:
```python
# Joining the data sets
wells_df = pd.merge(target_df, independent_df, on='id', how='inner')
```

In [4]:
```python
# Standardize categorical variables
wells_df = wells_df.applymap(lambda x: x.lower() if isinstance(x, str) else x)
```

In [5]:
```python
# Convert boolean values to strings
wells_df = wells_df.applymap(lambda x: str(x) if isinstance(x, bool) else x)
```

In [6]:
```python
# Standardize missing values to NaN
missing_values = ['', ' ', 'na', 'n/a', 'unknown', 'other', 'none']
wells_df.replace(missing_values, np.nan, inplace=True)
```

In [7]:
```python
# Set 'id' as the index of the DataFrame
wells_df.set_index('id', inplace=True)
```

In [8]:
```python
wells_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59400 entries, 69572 to 26348
Data columns (total 40 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   status_group           59400 non-null  object 
 1   amount_tsh             59400 non-null  float64
 2   date_recorded          59400 non-null  object 
 3   funder                 55759 non-null  object 
 4   gps_height             59400 non-null  int64  
 5   installer              55741 non-null  object 
 6   longitude              59400 non-null  float64
 7   latitude               59400 non-null  float64
 8   wpt_name               55832 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      54756 non-null  object 
 21  scheme_name            30565 non-null  object 
 22  permit                 56344 non-null  object 
 23  construction_year      59400 non-null  int64  
 24  extraction_type        52970 non-null  object 
 25  extraction_type_group  52970 non-null  object 
 26  extraction_type_class  52970 non-null  object 
 27  management             57995 non-null  object 
 28  management_group       57896 non-null  object 
 29  payment                50189 non-null  object 
 30  payment_type           50189 non-null  object 
 31  water_quality          57524 non-null  object 
 32  quality_group          57524 non-null  object 
 33  quantity               58611 non-null  object 
 34  quantity_group         58611 non-null  object 
 35  source                 59122 non-null  object 
 36  source_type            59122 non-null  object 
 37  source_class           59122 non-null  object 
 38  waterpoint_type        53020 non-null  object 
 39  waterpoint_type_group  53020 non-null  object 
dtypes: float64(3), int64(6), object(31)
memory usage: 18.6+ MB
```

Target Variable Investigation

In [9]:
```
wells_df['status_group'].value_counts(normalize=True)
```

Out[9]:
```
functional                 0.543081
non functional             0.384242
functional needs repair    0.072677
Name: status_group, dtype: float64
```

There is a class imbalance between the three classes inside of the target variable. We will need to handle this through by either oversampling or SMOTE'ing to bring the minority class in proportion with the majority class.

Given the small proportion of the 'function needs repair' class we will combine the 'functional needs repair' and 'non functional' class together to create the new class 'needs attention'. Then we will cast the 'functional' class of wells as 'does not need attention'.

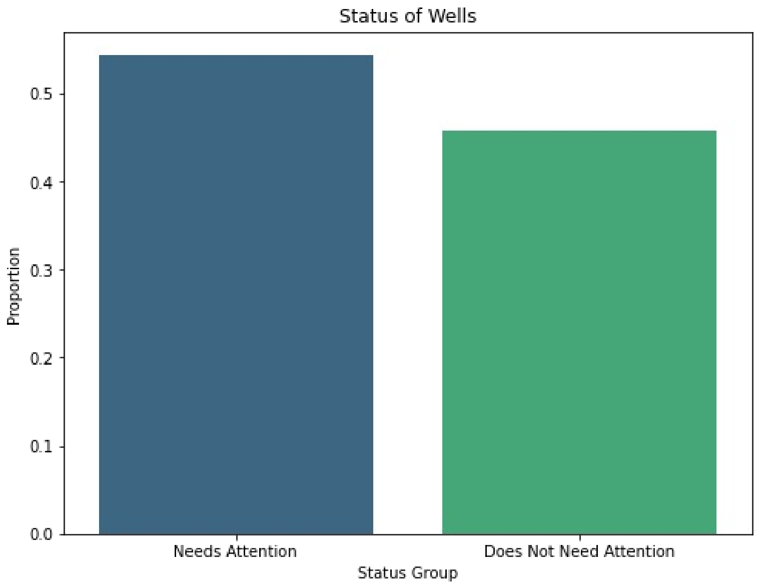
In [10]:
```
# Define the mapping of the new labels
label_mapping = {
    'functional': 0,
    'non functional': 1,
    'functional needs repair': 1
}

# Replace the values in the 'status_group' column
wells_df['status_group'] = wells_df['status_group'].replace(label_mapping)
```

In [11]:
```
status_counts_normalized = wells_df['status_group'].value_counts(normalize=True)
status_counts_normalized
```

Out[11]:
```
0    0.543081
1    0.456919
Name: status_group, dtype: float64
```

In [12]:
```
# Plotting
```

```
plt.figure(figsize=(8, 6))
sns.barplot(x=status_counts_normalized.index, y=status_counts_normalized.values, palette='viridis')
plt.xlabel('Status Group')
plt.ylabel('Proportion')
plt.title('Status of Wells')
plt.xticks([0, 1], ['Needs Attention', 'Does Not Need Attention'])
plt.show()
```



Data Set Visualizations

Map Visualization for Wells Status

In [13]:
```
# Create a map centered around the mean latitude and longitude
map_center = [wells_df['latitude'].mean(), wells_df['longitude'].mean()]
mymap = folium.Map(location=map_center, zoom_start=6)
```

In [14]:
```
# Define a function to determine marker color based on status_group
def get_color(status):
    return 'green' if status == 0 else 'red'
```

In [15]:
```
# Create markers for each row in the DataFrame
for index, row in wells_df.iterrows():
    color = get_color(row['status_group'])

    folium.CircleMarker(
        location=[row['latitude'], row['longitude']],
        radius=5,
        color=color,
        fill=True,
        fill_color=color
    ).add_to(mymap)
```

In [16]:
```
# Display the map
mymap
```

Out[16]:

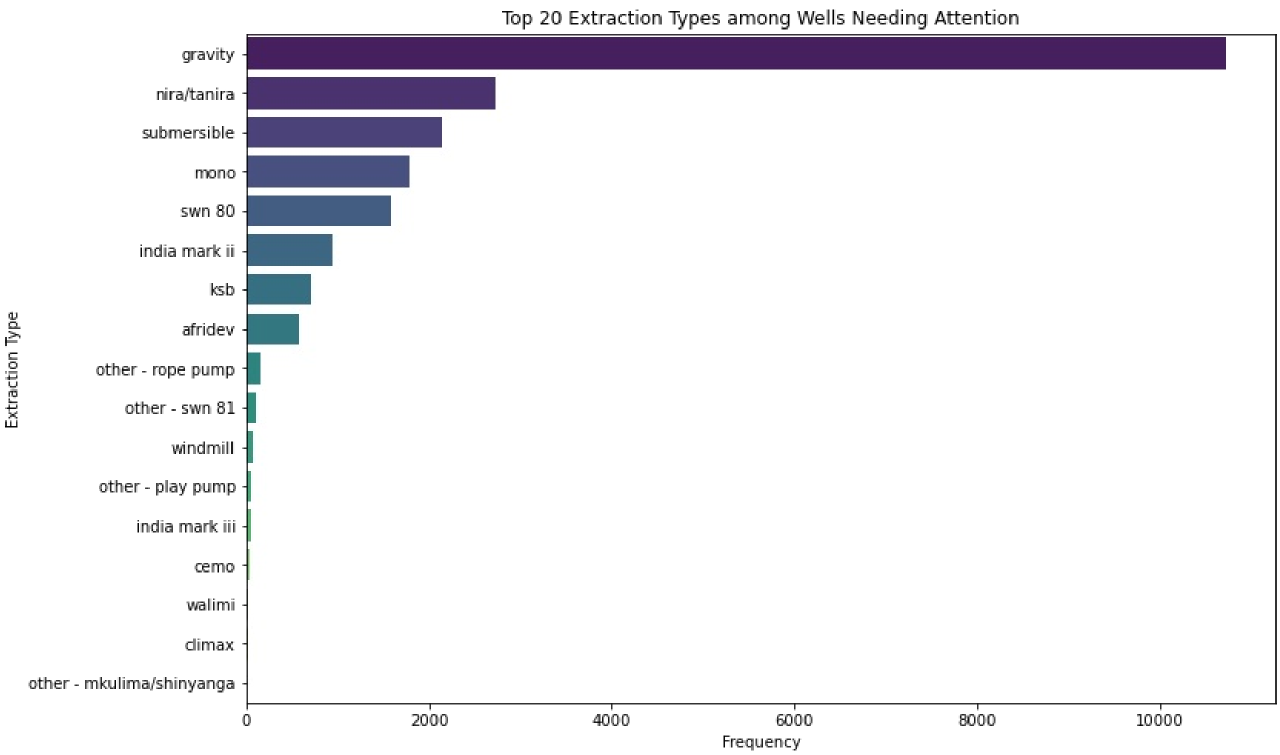
Extraction Type Investigation for Wells Needing Attention

In [17]:

```python
# Counting the frequency of each extraction_type for wells needing attention (status_group == 1)
attention_wells = wells_df[wells_df['status_group'] == 1]
extraction_type_counts = attention_wells['extraction_type'].value_counts()

# Selecting top 20 categories
top_20_extraction_types = extraction_type_counts.head(20)
```

In [18]:

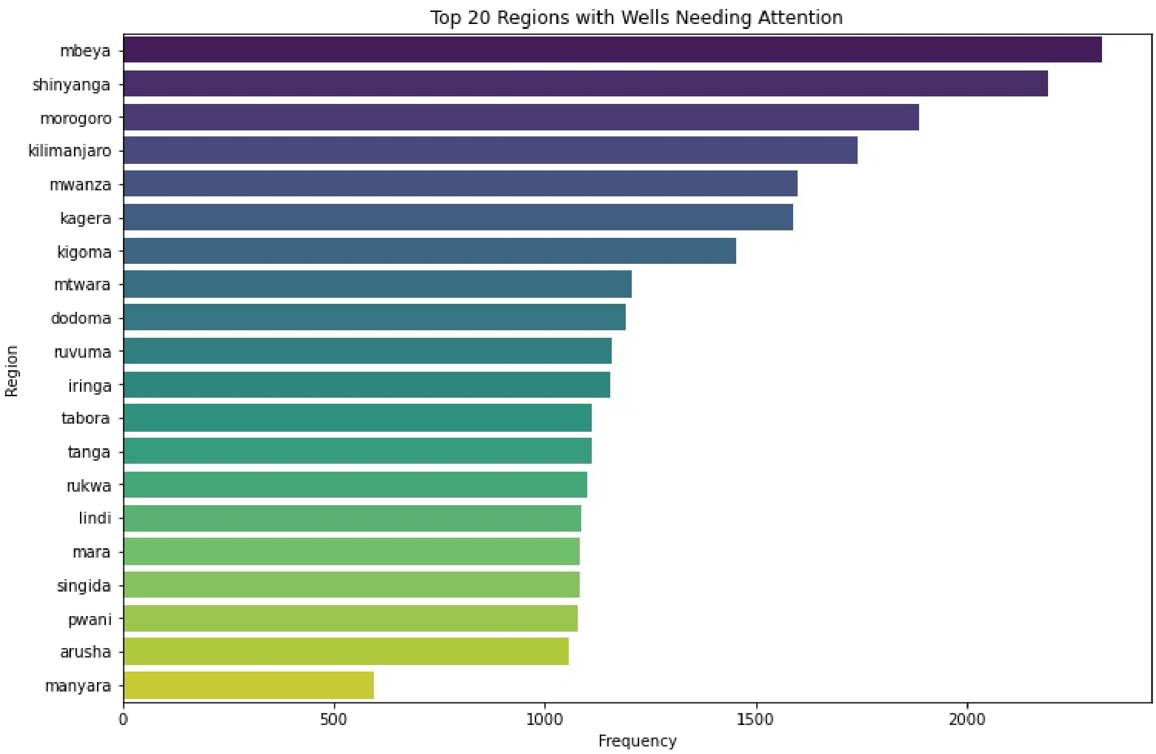
```python
# Plotting
plt.figure(figsize=(12, 8))
sns.barplot(x=top_20_extraction_types.values, y=top_20_extraction_types.index, palette='viridis')
plt.xlabel('Frequency')
plt.ylabel('Extraction Type')
plt.title('Top 20 Extraction Types among Wells Needing Attention')
plt.show()
```



Region Invetigation for Wells Needing Attention

In [19]:
```
# Counting the frequency of each region for wells needing attention (status_group == 1)
region_counts = attention_wells['region'].value_counts()

# Selecting top 20 regions
top_20_regions = region_counts.head(20)
```

In [20]:
```
# Plotting
plt.figure(figsize=(12, 8))
sns.barplot(x=top_20_regions.values, y=top_20_regions.index, palette='viridis')
plt.xlabel('Frequency')
plt.ylabel('Region')
plt.title('Top 20 Regions with Wells Needing Attention')
plt.show()
```



Data Cleaning

Amount TSH: Illogical Values

In [21]:
```
filtered_df = wells_df[wells_df['amount_tsh'] >= 50000]

filtered_df['amount_tsh'].value_counts()
```

Out[21]:
```
117000.0    7
50000.0     4
100000.0    3
250000.0    1
138000.0    1
70000.0     1
350000.0    1
60000.0     1
170000.0    1
200000.0    1
120000.0    1
Name: amount_tsh, dtype: int64
```

Most total static head measurements are in the range of hundreds to tens of thousands.

Depths exceeding 50,000 units (e.g., feet or meters) are rare or unusual so we will filter these out as they are outliers.

Most residential wells range from a few meters to around 100 meters (10 to 300 feet).

Wells used for industrial or agricultural purposes may have deeper total static heads, often ranging from tens to several hundred meters (hundreds to thousands of feet).

In [22]:
```
wells_df = wells_df[wells_df['amount_tsh'] <= 50000]
```

Longitude and Latitude: Zero Value Coordinates

In [23]:
```
wells_df['longitude'].value_counts()
```

Out[23]:
```
0.000000     1812
37.252194       2
37.297680       2
33.010510       2
39.093484       2
             ...
36.871976       1
37.579803       1
33.196490       1
34.017119       1
30.163579       1
Name: longitude, Length: 57498, dtype: int64
```

In [24]:
```
wells_df['latitude'].value_counts()
```

Out[24]:
```
-2.000000e-08    1812
-7.104923e+00       2
-6.964258e+00       2
-6.963565e+00       2
-7.056372e+00       2
                 ...
-3.411358e+00       1
-8.958207e+00       1
-3.261113e+00       1
-5.435762e+00       1
-2.598965e+00       1
Name: latitude, Length: 57499, dtype: int64
```

In [25]:
```
# Filtering to see if any other rows are affected
filtered_df = wells_df[wells_df['longitude'] == 0]

print(len(filtered_df))

filtered_df.head()
```

1812

Out[25]:

id	status_group	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wpt_name	num_priv
6091	0	0.0	2013-02-10	dwsp	0	dwe	0.0	-2.000000e-08	muungano	
32376	1	0.0	2011-08-01	government of tanzania	0	government	0.0	-2.000000e-08	polisi	
72678	0	0.0	2013-01-30	wvt	0	wvt	0.0	-2.000000e-08	wvt tanzania	
56725	1	0.0	2013-01-17	netherlands	0	dwe	0.0	-2.000000e-08	kikundi cha wakina mama	
13042	1	0.0	2012-10-29	hesawa	0	dwe	0.0	-2.000000e-08	kwakisusi	

5 rows × 40 columns

Since these wells have longitude and latiude of 0 it is safe to say that these wells while they were recorded either have an unknown location or do not have a location inside of Tanzania and as such are not reliable data point.

We will be dropping all of these rows from the dataset.

In [26]:
```
wells_df = wells_df[wells_df['longitude'] != 0]
```

Installer Column: Typos

In [27]:
```
wells_df['installer'].nunique()
```

Out[27]:
```
1903
```

In [28]:
```
installer_replacement_dict = {
    'central government': 'central government',
    'tanzania government': 'central government',
    'cental government': 'central government',
    'cebtral government': 'central government',
    'centra government': 'central government',
```

```
    'central govt': 'central government',
    'centr': 'central government',
    'centra govt': 'central government',
    'tanzanian government': 'central government',
    'tanzania': 'central government',

    'district council': 'district council',
    'counc': 'district council',
    'district counci': 'district council',
    'council': 'district council',
    'coun': 'district council',
    'distri': 'district council',
    'district  council': 'district council',

    'villigers': 'villagers',
    'villager': 'villagers',
    'villa': 'villagers',
    'village': 'villagers',
    'villi': 'villagers',
    'village council': 'villagers',
    'village counil': 'villagers',
    'villages': 'villagers',
    'vill': 'villagers',
    'village community': 'villagers',
    'villaers': 'villagers',
    'villag': 'villagers',
    'villege council': 'villagers',
    'villagerd': 'villagers',
    'village technician': 'villagers',
    'village office': 'villagers',
    'village community members': 'villagers',
    'village government': 'villagers',
    'village govt': 'villagers',

    'district water department': 'district water department',
    'district water depar': 'district water department',
    'distric water department': 'district water department',

    'finw': 'fini water',
    'fini water': 'fini water',
    'fin water': 'fini water',
    'finwater': 'fini water',
    'finn water': 'fini water',
    'fw': 'fini water',

    'rc church': 'rc church',
    'rc churc': 'rc church',
    'rc': 'rc church',
    'rc ch': 'rc church',
    'rc c': 'rc church',
    'rc cathoric': 'rc church',
    'ch': 'rc church',

    'world vision': 'world vision',
    'world division': 'world vision',
    'world vission': 'world vision',

    'unisef': 'unicef',
    'unicef': 'unicef',

    'danid': 'danida',

    'commu': 'community',
    'communit': 'community',
    'adra /community': 'community',
    'adra/community': 'community',
    'adra/ community': 'community',

    'government': 'government',
    'gover': 'government',
    'governme': 'government',
    'goverm': 'government',
    'govern': 'government',
    'gove': 'government',
    'governmen': 'government',

    'hesawa': 'hesawa',

    'jaica': 'jaica',
    'jica': 'jaica',
    'jeica': 'jaica',
    'jaica co': 'jaica',
```

```
    'kkkt _ konde and dwe': 'kkkt',
    'kkt': 'kkkt',
    'kkkt church': 'kkkt'
}
```

In [29]:
```
wells_df['installer'] = wells_df['installer'].replace(installer_replacement_dict)
```

In [30]:
```
wells_df['installer'].nunique()
```

Out[30]:
```
1835
```

Formatting Cleanup

In [31]:
```
# Function to replace "other - " prefix
def replace_other_prefix(value):
    if isinstance(value, str) and value.startswith('other -'):
        return value.split('other -', 1)[1].strip()
    return value

# Apply the function to the entire DataFrame
wells_df = wells_df.applymap(replace_other_prefix)
```

Missing Values

Categorical columns with 0's

In [32]:
```
# Replace all 0's with NaNs
for col in wells_df.columns:
    if wells_df[col].dtype == 'O':
        wells_df[col] = wells_df[col].replace('0', np.nan)
```

Applying the correct data types

The columns listed below act more as lables than as integers. They identify a specific aspect of about the row entry and as such should be considered categorical variables.

In [33]:
```
wells_df['region_code'] = wells_df['region_code'].astype(str)
wells_df['district_code'] = wells_df['district_code'].astype(str)
```

Missing values

In [34]:
```
# Assuming you already have 'expected_length' and 'wells_df' defined
expected_length = 59400
columns_with_missing_values = {}

# Iterate over columns to find those not meeting expected length
for col in wells_df.columns:
    if wells_df[col].count() != expected_length:
        columns_with_missing_values[col] = {
            'missing_count': expected_length - wells_df[col].count(),
            'unique_count': len(wells_df[col].value_counts()),
            'dtype': wells_df[col].dtype  # Add data type information
        }

# Print the dictionary
for col, info in columns_with_missing_values.items():
    print(f"{col}': Missing Values: {info['missing_count']}, Unique Values: {info['unique_count']}, Dtype: {inf
```

```
status_group': Missing Values: 1830, Unique Values: 2, Dtype: int64
amount_tsh': Missing Values: 1830, Unique Values: 88, Dtype: float64
date_recorded': Missing Values: 1830, Unique Values: 353, Dtype: object
funder': Missing Values: 6234, Unique Values: 1854, Dtype: object
gps_height': Missing Values: 1830, Unique Values: 2428, Dtype: int64
installer': Missing Values: 6246, Unique Values: 1834, Dtype: object
longitude': Missing Values: 1830, Unique Values: 57497, Dtype: float64
latitude': Missing Values: 1830, Unique Values: 57498, Dtype: float64
wpt_name': Missing Values: 5327, Unique Values: 36707, Dtype: object
num_private': Missing Values: 1830, Unique Values: 65, Dtype: int64
basin': Missing Values: 1830, Unique Values: 9, Dtype: object
subvillage': Missing Values: 2201, Unique Values: 18566, Dtype: object
region': Missing Values: 1830, Unique Values: 21, Dtype: object
region_code': Missing Values: 1830, Unique Values: 27, Dtype: object
district_code': Missing Values: 1830, Unique Values: 20, Dtype: object
lga': Missing Values: 1830, Unique Values: 124, Dtype: object
ward': Missing Values: 1830, Unique Values: 2033, Dtype: object
population': Missing Values: 1830, Unique Values: 1047, Dtype: int64
public_meeting': Missing Values: 4806, Unique Values: 2, Dtype: object
recorded_by': Missing Values: 1830, Unique Values: 1, Dtype: object
scheme_management': Missing Values: 6346, Unique Values: 10, Dtype: object
scheme_name': Missing Values: 29189, Unique Values: 2536, Dtype: object
permit': Missing Values: 4884, Unique Values: 2, Dtype: object
construction_year': Missing Values: 1830, Unique Values: 55, Dtype: int64
extraction_type': Missing Values: 7990, Unique Values: 17, Dtype: object
extraction_type_group': Missing Values: 7990, Unique Values: 12, Dtype: object
extraction_type_class': Missing Values: 7990, Unique Values: 6, Dtype: object
management': Missing Values: 3221, Unique Values: 10, Dtype: object
management_group': Missing Values: 3320, Unique Values: 3, Dtype: object
payment': Missing Values: 10386, Unique Values: 5, Dtype: object
payment_type': Missing Values: 10386, Unique Values: 5, Dtype: object
water_quality': Missing Values: 3491, Unique Values: 7, Dtype: object
quality_group': Missing Values: 3491, Unique Values: 5, Dtype: object
quantity': Missing Values: 2603, Unique Values: 4, Dtype: object
quantity_group': Missing Values: 2603, Unique Values: 4, Dtype: object
source': Missing Values: 2096, Unique Values: 8, Dtype: object
source_type': Missing Values: 2096, Unique Values: 6, Dtype: object
source_class': Missing Values: 2096, Unique Values: 2, Dtype: object
waterpoint_type': Missing Values: 7997, Unique Values: 6, Dtype: object
waterpoint_type_group': Missing Values: 7997, Unique Values: 5, Dtype: object
```

Filling Missing Values

Since all of the columns above are categorical columns, we will be filling them with the string 'unknown' so that we can maintain as much data as possible.

In [35]:
```
wells_df.fillna('unknown', inplace=True)
```

In [36]:
```
wells_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 57570 entries, 69572 to 26348
Data columns (total 40 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   status_group           57570 non-null  int64  
 1   amount_tsh             57570 non-null  float64
 2   date_recorded          57570 non-null  object 
 3   funder                 57570 non-null  object 
 4   gps_height             57570 non-null  int64  
 5   installer              57570 non-null  object 
 6   longitude              57570 non-null  float64
 7   latitude               57570 non-null  float64
 8   wpt_name               57570 non-null  object 
 9   num_private            57570 non-null  int64  
 10  basin                  57570 non-null  object 
 11  subvillage             57570 non-null  object 
 12  region                 57570 non-null  object 
 13  region_code            57570 non-null  object 
 14  district_code          57570 non-null  object 
 15  lga                    57570 non-null  object 
 16  ward                   57570 non-null  object 
 17  population             57570 non-null  int64  
 18  public_meeting         57570 non-null  object 
 19  recorded_by            57570 non-null  object 
 20  scheme_management      57570 non-null  object 
 21  scheme_name            57570 non-null  object 
 22  permit                 57570 non-null  object 
 23  construction_year      57570 non-null  int64  
 24  extraction_type        57570 non-null  object 
 25  extraction_type_group  57570 non-null  object 
 26  extraction_type_class  57570 non-null  object 
 27  management             57570 non-null  object 
 28  management_group       57570 non-null  object 
 29  payment                57570 non-null  object 
 30  payment_type           57570 non-null  object 
 31  water_quality          57570 non-null  object 
 32  quality_group          57570 non-null  object 
 33  quantity               57570 non-null  object 
 34  quantity_group         57570 non-null  object 
 35  source                 57570 non-null  object 
 36  source_type            57570 non-null  object 
 37  source_class           57570 non-null  object 
 38  waterpoint_type        57570 non-null  object 
 39  waterpoint_type_group  57570 non-null  object 
dtypes: float64(3), int64(5), object(32)
memory usage: 18.0+ MB
```

Dropping Columns

In [37]:
```
are_identical = wells_df['quantity'].equals(wells_df['quantity_group'])
print('Are they identical?', are_identical)
```

```
Are they identical? True
```

We are going to drop the following columns:

- 'recorded_by'
  - All records were recorded by the same person. No unique data is provided.
- 'quantity_group'
  - This column is identical to 'quantity'
- 'date_recorded'
  - The date that the information was recorded does not hold valuable information for determining the status of the well.
- 'num_private'
  - There is no description for this column inside of the data table's documentation.
  - https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/page/25/

In [38]:
```
# Drop columns
wells_df.drop(['recorded_by', 'quantity_group', 'date_recorded', 'num_private'], axis=1, inplace=True)
```

Baseline Feature Importance

Baseline Feature Importance: Decision Tree

In [39]:
```
baseline_dt_df = wells_df.copy()
```

In [40]:
```
# Encode categorical variables using LabelEncoder
le = LabelEncoder()
for col in baseline_dt_df.select_dtypes(include='object').columns:
    baseline_dt_df[col] = le.fit_transform(baseline_dt_df[col])
```

In [41]:
```
X_train, X_test, y_train, y_test = split_data(baseline_dt_df, 'status_group')
```

In [42]:
```
# Train a simple decision tree classifier
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
```

Out[42]:
```
DecisionTreeClassifier(random_state=42)
```

In [43]:
```
# Get feature importances
importances = dt.feature_importances_
feature_names = X_train.columns
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
```

In [44]:
```
# Sort by importance
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)
feature_importance_df
```

Out[44]:

	Feature	Importance
29	quantity	0.143414
4	longitude	0.109715
5	latitude	0.095917
34	waterpoint_type_group	0.074964
6	wpt_name	0.072433
8	subvillage	0.063031
2	gps_height	0.047830
19	construction_year	0.038912
13	ward	0.037973
14	population	0.035885
1	funder	0.030737
3	installer	0.024856
0	amount_tsh	0.024518
12	lga	0.023856
17	scheme_name	0.022924
9	region	0.019552
22	extraction_type_class	0.014009
23	management	0.010207
20	extraction_type	0.009441
16	scheme_management	0.008787
18	permit	0.008369
25	payment	0.008288
11	district_code	0.008110
30	source	0.007853
31	source_type	0.007844
33	waterpoint_type	0.007246
7	basin	0.006857
21	extraction_type_group	0.006657
28	quality_group	0.005540
10	region_code	0.005316
26	payment_type	0.004768
27	water_quality	0.004146
24	management_group	0.004025
15	public_meeting	0.003425
32	source_class	0.002597

Baseline Feature Importance: RFE

In [45]:
```
# Initialize RFE
rfe = RFE(estimator=dt, n_features_to_select=10)
```

In [46]:
```
# Fit RFE
rfe.fit(X_train, y_train)
```

Out[46]:
```
RFE(estimator=DecisionTreeClassifier(random_state=42), n_features_to_select=10)
```

In [47]:
```
# Get the selected features
selected_features = X_train.columns[rfe.support_]
print("Selected Features:")
print(selected_features)
```

```
Selected Features:
Index(['funder', 'gps_height', 'longitude', 'latitude', 'wpt_name',
       'subvillage', 'ward', 'construction_year', 'quantity',
       'waterpoint_type'],
      dtype='object')
```

In [48]:
```
# Get feature ranking
ranking = rfe.ranking_
feature_ranking = pd.DataFrame({'Feature': X_train.columns, 'Ranking': ranking})

# Sort by ranking
feature_ranking = feature_ranking.sort_values(by='Ranking')
print("Features ranked by importance:")
print(feature_ranking)
```

```
Features ranked by importance:
                  Feature  Ranking
1                  funder        1
2              gps_height        1
4               longitude        1
5                latitude        1
6                wpt_name        1
29               quantity        1
8              subvillage        1
19      construction_year        1
13                   ward        1
33        waterpoint_type        1
14             population        2
12                    lga        3
0              amount_tsh        4
3               installer        5
22  extraction_type_class        6
9                  region        7
17            scheme_name        8
23             management        9
30                 source       10
25                payment       11
20        extraction_type       12
16      scheme_management       13
18                 permit       14
28          quality_group       15
11          district_code       16
7                   basin       17
21  extraction_type_group       18
31            source_type       19
10            region_code       20
26           payment_type       21
15         public_meeting       22
27          water_quality       23
24       management_group       24
34  waterpoint_type_group       25
32           source_class       26
```

Baseline Feature Importance: Analysis

In [49]:
```
# Merge feature importance and ranking DataFrames on feature names
baseline_feature_analysis_df = pd.merge(
feature_importance_df, feature_ranking, on='Feature', suffixes=('_importance', '_ranking')
)
```

Now that we have combined the feature selections together we can filter the dataframe so we can see the featuresthat are less likely to contribute to our models performance.

We will want to look at any feature that matches both of the criteria listed below and consider dropping those features.

For RFE (Recursive Feature Elimination) features with higher ranks are considered less important by RFE so we will look at features with a rank below 20.

For the decision tree feature importance, we will look at features that have a gini impurity below .01.

In [50]:
```
# Filter the DataFrame
features_to_drop = baseline_feature_analysis_df[
    (baseline_feature_analysis_df['Importance'] < 0.01) &
    (baseline_feature_analysis_df['Ranking'] >= 20)
]['Feature']
```

In [51]:
```
features_to_drop
```

Out[51]:
```
29         region_code
30        payment_type
31       water_quality
32    management_group
33      public_meeting
34        source_class
Name: Feature, dtype: object
```

In [52]:
```
columns_to_drop = {}

for col in features_to_drop:
    columns_to_drop[col] = {
        'dtype' : wells_df[col].dtype,
        'unique_values' : wells_df[col].nunique()
    }

columns_to_drop
```

Out[52]:
```
{'region_code': {'dtype': dtype('O'), 'unique_values': 27},
 'payment_type': {'dtype': dtype('O'), 'unique_values': 6},
 'water_quality': {'dtype': dtype('O'), 'unique_values': 8},
 'management_group': {'dtype': dtype('O'), 'unique_values': 4},
 'public_meeting': {'dtype': dtype('O'), 'unique_values': 3},
 'source_class': {'dtype': dtype('O'), 'unique_values': 3}}
```

'source_class' and 'source_type'

In [53]:
```
wells_df['source_class'].value_counts()
```

Out[53]:
```
groundwater    44201
surface        13103
unknown          266
Name: source_class, dtype: int64
```

In [54]:
```
wells_df['source_type'].value_counts()
```

Out[54]:
```
spring                  17006
shallow well            15498
borehole                11697
river/lake              10236
rainwater harvesting     2218
dam                       649
unknown                   266
Name: source_type, dtype: int64
```

Even though this a low scoring column, the water source of the well is most likely a good indicator of the status of the well itself. We will want to retain as much information as we can that can lead us to determining the functional status of a well.

'payment_type'

In [55]:
```
wells_df['payment_type'].value_counts()
```

Out[55]:
```
never pay     24380
per bucket     8953
unknown        8556
monthly        8216
on failure     3841
annually       3624
Name: payment_type, dtype: int64
```

In [56]:
```
wells_df['payment'].value_counts()
```

Out[56]:
```
never pay                24380
pay per bucket            8953
unknown                   8556
pay monthly               8216
pay when scheme fails     3841
pay annually              3624
Name: payment, dtype: int64
```

This column is essentially the same as 'payment' as such we can drop this column.

'public_meeting'

In [57]:
```
wells_df['public_meeting'].value_counts()
```

Out[57]:
```
True       49722
False       4872
unknown     2976
Name: public_meeting, dtype: int64
```

Given its low scores and that there is no further information given in the documentation aside from True/False, we will drop this column.

'management_group'

In [58]:
```
wells_df['management_group'].value_counts()
```

Out[58]:
```
user-group    50755
commercial     3634
parastatal     1691
unknown        1490
Name: management_group, dtype: int64
```

- How the waterpoint is managed?

This could indicate how different management practices that affect the functionality of the well. As such, we will want to keep this column.

'water_quality'

In [59]:
```
wells_df['water_quality'].value_counts()
```

Out[59]:
```
soft                  49413
salty                  4772
unknown                1661
milky                   803
coloured                479
salty abandoned         228
fluoride                199
fluoride abandoned       15
Name: water_quality, dtype: int64
```

In [60]:
```
wells_df['quality_group'].value_counts()
```

Out[60]:
```
good        49413
salty        5000
unknown      1661
milky         803
colored       479
fluoride      214
Name: quality_group, dtype: int64
```

While there is less crossover inside of water_quality and quality_group we will want to keep both.

The water quality of the well is most likely one of the best indicators of the status of the well.

We will want to retain as much information as we can that can lead us to determining the functional status of a well.

Dropping columns

In [61]:
```
wells_df = wells_df.drop(['payment_type', 'public_meeting'], axis=1)
```

Columns with too many unique values

In [62]:
```
wells_df.nunique()
```

Out[62]:
```
status_group                2
amount_tsh                 88
funder                   1855
gps_height               2428
installer                1835
longitude               57497
latitude                57498
wpt_name                36708
basin                       9
subvillage              18567
region                     21
region_code                27
district_code              20
lga                       124
ward                     2033
population               1047
scheme_management          11
scheme_name              2537
permit                      3
construction_year          55
extraction_type            18
extraction_type_group      13
extraction_type_class       7
management                 11
management_group            4
payment                     6
water_quality               8
quality_group               6
quantity                    5
source                      9
source_type                 7
source_class                3
waterpoint_type             7
waterpoint_type_group       6
dtype: int64
```

Given the high amount of unique values that exist inside of various columns, we will bucket the values that fall under 1% of the columns total values.

We will only be bucketing values for columns that have over 30 unique values so that we maintain all of the unique values in columns that have smaller amount of unique values.

In [63]:
```
too_many_unique_columns = []

for col in wells_df.columns:
    if wells_df[col].dtype == 'object' and wells_df[col].nunique() > 30:
        too_many_unique_columns.append(col)

too_many_unique_columns
```

Out[63]:
```
['funder', 'installer', 'wpt_name', 'subvillage', 'lga', 'ward', 'scheme_name']
```

In [64]:
```
for col in too_many_unique_columns:
    if wells_df[col].dtype == 'object':
        freq = wells_df[col].value_counts(normalize=True)
        mask = wells_df[col].isin(freq[freq >= 0.01].index)
        wells_df.loc[~mask, col] = 'other'
```

In [65]:
```
wells_df.nunique()
```

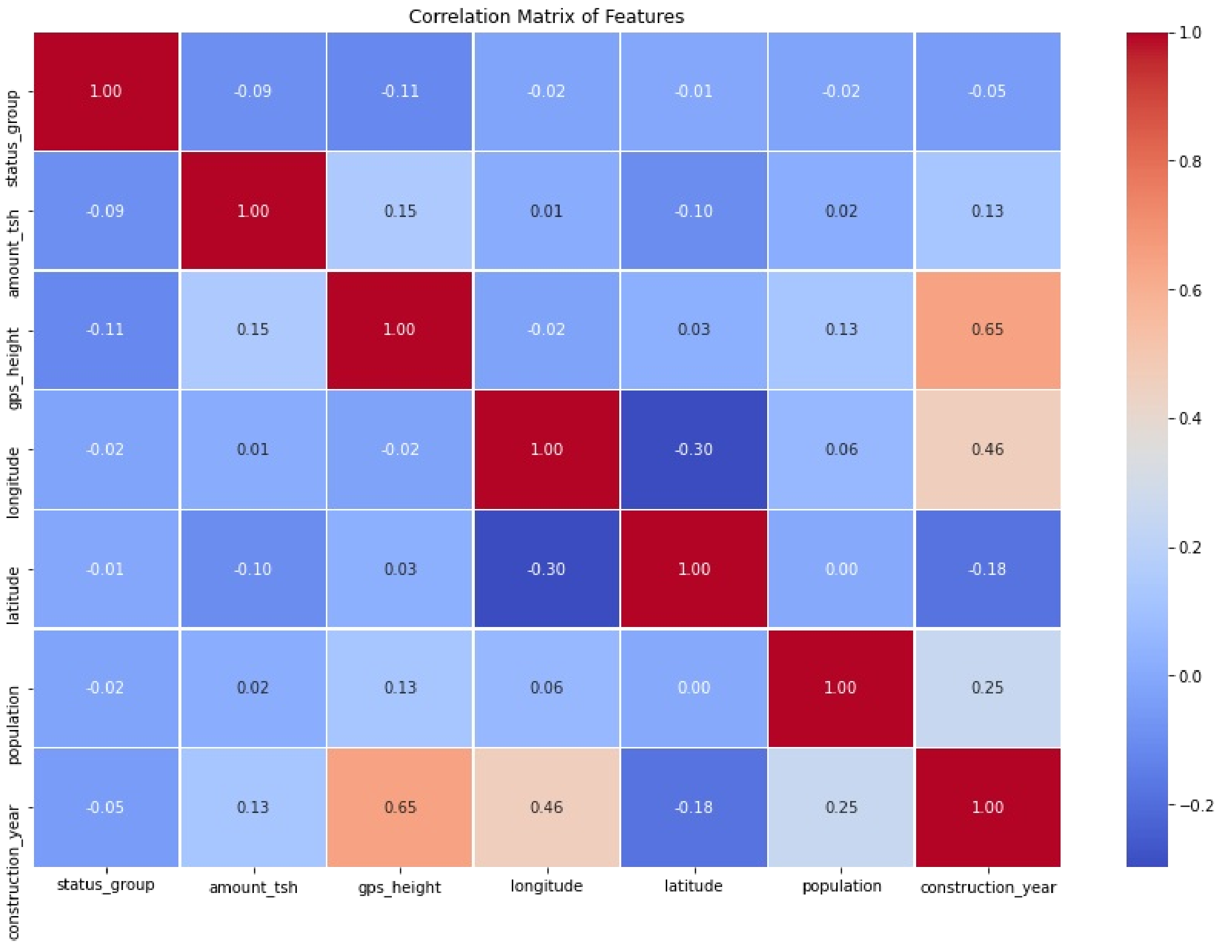
Out[65]:
```
status_group                 2
amount_tsh                  88
funder                      19
gps_height                2428
installer                   15
longitude                57497
latitude                 57498
wpt_name                     4
basin                        9
subvillage                   1
region                      21
region_code                 27
district_code               20
lga                         35
ward                         1
population                1047
scheme_management           11
scheme_name                  3
permit                       3
construction_year           55
extraction_type             18
extraction_type_group       13
extraction_type_class        7
management                  11
management_group             4
payment                      6
water_quality                8
quality_group                6
quantity                     5
source                       9
source_type                  7
source_class                 3
waterpoint_type              7
waterpoint_type_group        6
dtype: int64
```

Column Correlation

Numeric Columns

In [66]:
```
correlation_matrix = wells_df.corr()
```

In [67]:
```
# Set up the matplotlib figure
plt.figure(figsize=(15, 10))

# Draw the heatmap with the correlation matrix
sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='coolwarm', linewidths=0.5)

# Add title and labels
plt.title('Correlation Matrix of Features')
plt.show()
```



While some columns border on being too correlated with each other (~0.80), none of columns exceed this threshold. If a column had multiple high correlations (either positive or negative) with other columns we would drop that column, but this is not the case.

Scoring Metric

Inside of all the models listed below, we are using recall as our scoring metric.

In the target variable, we started with three classes:

- Functional
- Non-Functional
- Functional needs repair

We combined the non-functional and the functional needs repair classes due to the heavy class imbalance inside of the functional needs repair class. Since functional needs repair and non-functional wells could both be classified into a needs attention class we combined them. Our updated classes are:

- Does not need attention (for functional wells)
- Needs attention (for non-functional and functionally needs repair wells)

We chose to use recall as our primary scoring metric because it measures the proportion of wells that need attention that are correctly predicted as being in need of attention.

Sampling Methods

In [68]:
```
sampling_methods = ['oversample', 'undersample', 'smote']
```

Inside of each model we will be performing these three sampling methods to determine which method performs the best.

Base Logistic Regression Models

In [69]:
```
base_log_model_df = wells_df.copy()
```

In [70]:
```
X_train, X_test, y_train, y_test = split_data(base_log_model_df, 'status_group')
```

In [71]:
```
base_logistic_model = LogisticRegression(random_state=42, max_iter=2000)
```

In [72]:
```
base_log_cv = CustomCrossValidator(base_logistic_model, X_train, y_train)
```

In [73]:
```
base_log_cv.run_sampling_methods(sampling_methods, features=None, features_from='base', folds=5)
```

```
Running cross-validation with sampling method: oversample

Running cross-validation with sampling method: undersample

Running cross-validation with sampling method: smote

Log: {('LogisticRegression', 'base', 'oversample'): 0.7043794944655426, ('LogisticRegression', 'base', 'undersam
ple'): 0.7032904517477562, ('LogisticRegression', 'base', 'smote'): 0.7011659423295937}
```

Base Decision Tree Models

In [74]:
```
base_dt_model_df = wells_df.copy()
```

In [75]:
```
X_train, X_test, y_train, y_test = split_data(base_dt_model_df, 'status_group')
```

In [76]:
```
base_dt_model = DecisionTreeClassifier(random_state=42, max_depth=5, min_samples_split=20, min_samples_leaf=10)
```

In [77]:
```
base_dt_cv = CustomCrossValidator(base_dt_model, X_train, y_train)
```

In [78]:
```
base_dt_cv.run_sampling_methods(sampling_methods, features=None, features_from='base', folds=5)
```

```
Running cross-validation with sampling method: oversample

Running cross-validation with sampling method: undersample

Running cross-validation with sampling method: smote

Log: {('LogisticRegression', 'base', 'oversample'): 0.7043794944655426, ('LogisticRegression', 'base', 'undersam
ple'): 0.7032904517477562, ('LogisticRegression', 'base', 'smote'): 0.7011659423295937, ('DecisionTreeClassifier
', 'base', 'oversample'): 0.5150329468026201, ('DecisionTreeClassifier', 'base', 'undersample'): 0.5310489564026
828, ('DecisionTreeClassifier', 'base', 'smote'): 0.5393838624465799}
```

L1 Penalty Logistic Regression: Feature Selection

In [79]:
```
l1_penalty_log_regression_df = wells_df.copy()
```

Feature Selection

In [80]:
```
# Split the data
X_train, X_test, y_train, y_test = split_data(l1_penalty_log_regression_df, target='status_group')

# Scale numeric features
X_train_numeric_scaled, X_test_numeric_scaled = scale_numeric_features(X_train, X_test)

# Encode categorical features
X_train_encoded, X_test_encoded = encode_categorical_features(X_train, X_test)

# Combine numeric and categorical features
X_train_preprocessed = combine_features(X_train_numeric_scaled, X_train_encoded)
X_test_preprocessed = combine_features(X_test_numeric_scaled, X_test_encoded)
```

In [81]:
```
# Instantiate the model
l1_penalty_log_regression_model = LogisticRegression(random_state=42, penalty='l1', solver='liblinear')
```

In [82]:
```
# Fit the model
l1_penalty_log_regression_model.fit(X_train_preprocessed, y_train)
```

Out[82]:
```
LogisticRegression(penalty='l1', random_state=42, solver='liblinear')
```

In [83]:
```
# Extract feature importance (non-zero coefficients)
importance = l1_penalty_log_regression_model.coef_[0]

l1_penalty_selected_features = X_train_preprocessed.columns[importance != 0]
```

In [84]:
```
print(f"Starting Features: {X_train_preprocessed.shape[1]}")
print()
print(f"Selected Features: {len(l1_penalty_selected_features)}")
```

```
Starting Features: 284

Selected Features: 224
```

The L1 penalty on the Logistic Regression set the coefficient of 60 features to 0 indicating that these features are not considered important for predicting the target variable.

L1 Penalty Features: Logistic Regression

In [85]:
```
l1_features_l2_logistic_model_df = wells_df.copy()
```

In [86]:
```
# Split the data
X_train, X_test, y_train, y_test = split_data(l1_features_l2_logistic_model_df, target='status_group')
```

In [87]:
```
# Instantiate the model
l1_features_l2_logistic_model = LogisticRegression(random_state=42, max_iter=2000)
```

In [88]:
```
l1_features_l2_logistic_cv = CustomCrossValidator(
    model=l1_features_l2_logistic_model,
    X=X_train,
    y=y_train,
)
```

In [89]:
```
l1_features_l2_logistic_cv.run_sampling_methods(
    sampling_methods,
    features=l1_penalty_selected_features,
    features_from='L1_Penalty'
)
```

```
Running cross-validation with sampling method: oversample

Running cross-validation with sampling method: undersample

Running cross-validation with sampling method: smote

Log: {('LogisticRegression', 'base', 'oversample'): 0.7043794944655426, ('LogisticRegression', 'base', 'undersam
ple'): 0.7032904517477562, ('LogisticRegression', 'base', 'smote'): 0.7011659423295937, ('DecisionTreeClassifier
', 'base', 'oversample'): 0.5150329468026201, ('DecisionTreeClassifier', 'base', 'undersample'): 0.5310489564026
828, ('DecisionTreeClassifier', 'base', 'smote'): 0.5393838624465799, ('LogisticRegression', 'L1_Penalty', 'over
sample'): 0.7048153281712818, ('LogisticRegression', 'L1_Penalty', 'undersample'): 0.7038352475891534, ('Logisti
cRegression', 'L1_Penalty', 'smote'): 0.7014929177579201}
```

L1 Penalty Features: Decision Tree

In [90]:
```
l1_features_dt_model_df = wells_df.copy()

X_train, X_test, y_train, y_test = split_data(l1_features_dt_model_df, 'status_group')
```

In [91]:
```
l1_features_dt_model = DecisionTreeClassifier(random_state=42, max_depth=5, min_samples_split=20, min_samples_le
```

In [92]:
```
l1_features_dt_cv = CustomCrossValidator(
    model=l1_features_dt_model,
    X=X_train,
    y=y_train,
)
```

In [93]:
```
l1_features_dt_cv.run_sampling_methods(
    sampling_methods,
    features=l1_penalty_selected_features,
    folds=5,
    features_from='L1_Penalty'
)
```

```
Running cross-validation with sampling method: oversample

Running cross-validation with sampling method: undersample

Running cross-validation with sampling method: smote

Log: {('LogisticRegression', 'base', 'oversample'): 0.7043794944655426, ('LogisticRegression', 'base', 'undersam
ple'): 0.7032904517477562, ('LogisticRegression', 'base', 'smote'): 0.7011659423295937, ('DecisionTreeClassifier
', 'base', 'oversample'): 0.5150329468026201, ('DecisionTreeClassifier', 'base', 'undersample'): 0.5310489564026
828, ('DecisionTreeClassifier', 'base', 'smote'): 0.5393838624465799, ('LogisticRegression', 'L1_Penalty', 'over
sample'): 0.7048153281712818, ('LogisticRegression', 'L1_Penalty', 'undersample'): 0.7038352475891534, ('Logisti
cRegression', 'L1_Penalty', 'smote'): 0.7014929177579201, ('DecisionTreeClassifier', 'L1_Penalty', 'oversample')
: 0.5151418792644937, ('DecisionTreeClassifier', 'L1_Penalty', 'undersample'): 0.5311034374705116, ('DecisionTre
eClassifier', 'L1_Penalty', 'smote'): 0.539274915147814}
```

VIF: Feature Selection

In [94]:
```
vif_df = wells_df.copy()
```

Feature Selection

In [95]:
```
# Split the data
X_train, X_test, y_train, y_test = split_data(vif_df, target='status_group')

# Scale numeric features
X_train_numeric_scaled, X_test_numeric_scaled = scale_numeric_features(X_train, X_test)

# Encode categorical features
X_train_encoded, X_test_encoded = encode_categorical_features(X_train, X_test)

# Combine numeric and categorical features
X_train_preprocessed = combine_features(X_train_numeric_scaled, X_train_encoded)
X_test_preprocessed = combine_features(X_test_numeric_scaled, X_test_encoded)
```

In [96]:
```
# Remove features with variance below 1%
selector = VarianceThreshold(threshold=0.01)
X_train_reduced_df = selector.fit_transform(X_train_preprocessed)

# Get the selected feature names
selected_features = X_train_preprocessed.columns[selector.get_support()]

# Create a new DataFrame with selected features
X_train_reduced_df = pd.DataFrame(X_train_reduced_df, columns=selected_features)
```

In [97]:
```
X_train_reduced_df.shape
```

Out[97]:
```
(40299, 234)
```

In [98]:
```
# Function to calculate VIF
def calculate_vif(df):
    vif_data = pd.DataFrame()
    vif_data['feature'] = df.columns
    vif_data['VIF'] = [variance_inflation_factor(df.values, i) for i in range(df.shape[1])]
    return vif_data

# Apply VIF calculation on reduced DataFrame
vif_data_reduced = calculate_vif(X_train_reduced_df)

vif_data_reduced
```

```
/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/statsmodels/stats/outliers_influence.py:193: RuntimeWa
rning: divide by zero encountered in double_scalars
  vif = 1. / (1. - r_squared_i)
```

Out[98]:

	feature	VIF
0	amount_tsh	1.326252
1	gps_height	11.058491
2	longitude	90.910726
3	latitude	94.372529
4	population	1.224808
...	...	...
229	waterpoint_type_unknown	inf
230	waterpoint_type_group_communal standpipe	inf
231	waterpoint_type_group_hand pump	inf
232	waterpoint_type_group_improved spring	inf
233	waterpoint_type_group_unknown	inf

234 rows × 2 columns

In [99]:
```
vif_under_10 = vif_data_reduced[vif_data_reduced['VIF'] < 10]
vif_under_10
```

Out[99]:

	feature	VIF
0	amount_tsh	1.326252
4	population	1.224808
7	funder_dhv	3.541435
8	funder_district council	3.381726
11	funder_hesawa	6.251303
12	funder_kkkt	6.220374
13	funder_norad	3.205758
15	funder_private individual	3.597767
16	funder_rwssp	4.516637
17	funder_tasaf	2.994374
18	funder_tcrs	4.749076
19	funder_unicef	3.548683
21	funder_water	2.764913
22	funder_world bank	4.081632
23	funder_world vision	5.146454
99	district_code_33	4.070209
102	district_code_53	1.898749
106	lga_arusha rural	9.185187
107	lga_bagamoyo	8.076241
109	lga_iringa rural	4.935398
110	lga_kahama	4.505269
111	lga_karagwe	4.632777
115	lga_kilombero	7.291015
116	lga_kilosa	7.363698
117	lga_kwimba	4.234895
118	lga_kyela	6.953799
119	lga_lushoto	4.525396
120	lga_makete	4.769707
121	lga_maswa	3.617241
122	lga_mbarali	5.927934
123	lga_mbinga	6.360571
124	lga_mbozi	9.059500
125	lga_meru	9.479916
126	lga_moshi rural	7.421927
127	lga_mpanda	5.476681
128	lga_mvomero	5.587916
129	lga_namtumbo	6.492646
133	lga_rombo	7.111824
135	lga_same	6.307670
136	lga_serengeti	5.374103
137	lga_singida rural	9.735145
138	lga_songea rural	6.203831
139	lga_ulanga	5.873941

In [100…
```
vif_selected_features = list(vif_under_10['feature'])

len(vif_selected_features)
```

Out[100… 43

VIF Features: Logistic Regression

In [101…
```python
vif_logistic_model_df = wells_df.copy()

X_train, X_test, y_train, y_test = split_data(vif_logistic_model_df, 'status_group')
```

In [102…
```python
vif_logistic_model = LogisticRegression(random_state=42, max_iter=2000)
```

In [103…
```python
vif_logistic_cv = CustomCrossValidator(
    model=vif_logistic_model,
    X=X_train,
    y=y_train,
)
```

In [104…
```python
vif_logistic_cv.run_sampling_methods(
    sampling_methods,
    features=vif_selected_features,
    folds=5,
    features_from='VIF'
)
```

```
Running cross-validation with sampling method: oversample

Running cross-validation with sampling method: undersample

Running cross-validation with sampling method: smote

Log: {('LogisticRegression', 'base', 'oversample'): 0.7043794944655426, ('LogisticRegression', 'base', 'undersam
ple'): 0.7032904517477562, ('LogisticRegression', 'base', 'smote'): 0.7011659423295937, ('DecisionTreeClassifier
', 'base', 'oversample'): 0.5150329468026201, ('DecisionTreeClassifier', 'base', 'undersample'): 0.5310489564026
828, ('DecisionTreeClassifier', 'base', 'smote'): 0.5393838624465799, ('LogisticRegression', 'L1_Penalty', 'over
sample'): 0.7048153281712818, ('LogisticRegression', 'L1_Penalty', 'undersample'): 0.7038352475891534, ('Logisti
cRegression', 'L1_Penalty', 'smote'): 0.7014929177579201, ('DecisionTreeClassifier', 'L1_Penalty', 'oversample')
: 0.5151418792644937, ('DecisionTreeClassifier', 'L1_Penalty', 'undersample'): 0.5311034374705116, ('DecisionTre
eClassifier', 'L1_Penalty', 'smote'): 0.539274915147814, ('LogisticRegression', 'VIF', 'oversample'): 0.68063043
73500361, ('LogisticRegression', 'VIF', 'undersample'): 0.6832991788076954, ('LogisticRegression', 'VIF', 'smote
'): 0.6713702433665739}
```

VIF Features: Decision Tree

In [105…
```python
vif_dt_model_df = wells_df.copy()

X_train, X_test, y_train, y_test = split_data(vif_dt_model_df, 'status_group')
```

In [106…
```python
vif_dt_model = DecisionTreeClassifier(random_state=42, max_depth=5, min_samples_split=20, min_samples_leaf=10)
```

In [107…
```python
vif_dt_cv = CustomCrossValidator(
    model=vif_dt_model,
    X=X_train,
    y=y_train,
)
```

In [108…
```python
vif_dt_cv.run_sampling_methods(
    sampling_methods,
    features=vif_selected_features,
    folds=5,
    features_from='VIF'
)
```

```
Running cross-validation with sampling method: oversample

Running cross-validation with sampling method: undersample

Running cross-validation with sampling method: smote

Log: {('LogisticRegression', 'base', 'oversample'): 0.7043794944655426, ('LogisticRegression', 'base', 'undersam
ple'): 0.7032904517477562, ('LogisticRegression', 'base', 'smote'): 0.7011659423295937, ('DecisionTreeClassifier
', 'base', 'oversample'): 0.5150329468026201, ('DecisionTreeClassifier', 'base', 'undersample'): 0.5310489564026
828, ('DecisionTreeClassifier', 'base', 'smote'): 0.5393838624465799, ('LogisticRegression', 'L1_Penalty', 'over
sample'): 0.7048153281712818, ('LogisticRegression', 'L1_Penalty', 'undersample'): 0.7038352475891534, ('Logisti
cRegression', 'L1_Penalty', 'smote'): 0.7014929177579201, ('DecisionTreeClassifier', 'L1_Penalty', 'oversample')
: 0.5151418792644937, ('DecisionTreeClassifier', 'L1_Penalty', 'undersample'): 0.5311034374705116, ('DecisionTre
eClassifier', 'L1_Penalty', 'smote'): 0.539274915147814, ('LogisticRegression', 'VIF', 'oversample'): 0.68063043
73500361, ('LogisticRegression', 'VIF', 'undersample'): 0.6832991788076954, ('LogisticRegression', 'VIF', 'smote
'): 0.6713702433665739, ('DecisionTreeClassifier', 'VIF', 'oversample'): 0.7701797459805376, ('DecisionTreeClass
ifier', 'VIF', 'undersample'): 0.767510559416115, ('DecisionTreeClassifier', 'VIF', 'smote'): 0.768382701608141}
```

Current Model Results

In [109…
```python
def get_log_results_df():
    # Extracting the log dictionary
```

```
    log_dict = vif_dt_cv.log

    # Initialize an empty list to store the rows
    log_list = []

    # Iterate over the log dictionary and append each entry as a tuple to the list
    for key, value in log_dict.items():
        model, feature_selection, sampling = key
        score = value
        log_list.append((model, feature_selection, sampling, score))

    # Convert the list to a DataFrame
    log_df = pd.DataFrame(log_list, columns=['Model', 'Feature Selection', 'Sampling Technique', 'Recall-Score'

    return log_df
```

In [110…

```
log_df = get_log_results_df()
log_df.sort_values(by='Recall-Score', ascending=False)
```

Out[110…

	Model	Feature Selection	Sampling Technique	Recall-Score
15	DecisionTreeClassifier	VIF	oversample	0.770180
17	DecisionTreeClassifier	VIF	smote	0.768383
16	DecisionTreeClassifier	VIF	undersample	0.767511
6	LogisticRegression	L1_Penalty	oversample	0.704815
0	LogisticRegression	base	oversample	0.704379
7	LogisticRegression	L1_Penalty	undersample	0.703835
1	LogisticRegression	base	undersample	0.703290
8	LogisticRegression	L1_Penalty	smote	0.701493
2	LogisticRegression	base	smote	0.701166
13	LogisticRegression	VIF	undersample	0.683299
12	LogisticRegression	VIF	oversample	0.680630
14	LogisticRegression	VIF	smote	0.671370
5	DecisionTreeClassifier	base	smote	0.539384
11	DecisionTreeClassifier	L1_Penalty	smote	0.539275
10	DecisionTreeClassifier	L1_Penalty	undersample	0.531103
4	DecisionTreeClassifier	base	undersample	0.531049
9	DecisionTreeClassifier	L1_Penalty	oversample	0.515142
3	DecisionTreeClassifier	base	oversample	0.515033

Given the success of the base logistic regression model, we want to perform some dimensionality reduction to see if we can get higher recall scores from our model.

We will perform dimensionality reduction on columns that share similar values. We kept these columns in previously because we believed that while they were similar, they did have differences and those differences could lead to a better model performance.

Dimensionality Reduction

Various columns contain similar information. While these columns are not a equal in every regard, they are similar enough as to where our models may perform better if we removed these features. We had originally kept these features as we thought having the extra data points inside of these features would be helpful for classification.

If the columns contain over 80% of shared row values, we will drop a column.

We will be comparing the following features:

- 'scheme_management' and 'management'
- 'region' and 'region_code'
- 'extraction_type' and 'extraction_type_group'
- 'water_quality' and 'quality_group'
- 'source' and 'source_type'
- 'waterpoint_type' and 'waterpoint_type_group'

In [111…

```
def get_shared_values_percentage(df, pairs):
    """
    Calculates the percentage of shared values between pairs of columns in a DataFrame.
```

```
    Return a dictionary with the shared values percentage.
    ---
    Input:
    df: Pandas DataFrame
    pairs : A list of tuples
    ---
    Output:
    dict: A dictionary
    """
    result = {}

    for column1, column2 in pairs:
        # Create a new DataFrame with only the two specified columns
        df_subset = df[[column1, column2]].copy()

        # Create a new column that indicates if the values in the two columns are the same
        df_subset['is_same'] = df_subset[column1] == df_subset[column2]

        # Count the number of shared values (where the values are the same in both columns)
        shared_values_count = df_subset['is_same'].sum()

        # Calculate the percentage of shared values
        shared_values_percentage = (shared_values_count / len(df_subset)) * 100

        # Store the result in the dictionary
        result[(column1, column2)] = shared_values_percentage

    return result
```

In [112…

```
# Define the list of column pairs to compare
pairs = [
    ('scheme_management', 'management'),
    ('extraction_type', 'extraction_type_group'),
    ('water_quality', 'quality_group'),
    ('source', 'source_type'),
    ('waterpoint_type', 'waterpoint_type_group')
]
```

In [113…

```
# Calculate shared values percentages for each pair
results = get_shared_values_percentage(wells_df, pairs)
results
```

Out[113…

```
{('scheme_management', 'management'): 83.8909154073302,
 ('extraction_type', 'extraction_type_group'): 96.64929650859823,
 ('water_quality', 'quality_group'): 12.914712523883967,
 ('source', 'source_type'): 61.902032308494,
 ('waterpoint_type', 'waterpoint_type_group'): 89.65085982282439}
```

Given the results above, we will take a closer look at the following pairs:

- 'scheme_management' and 'management'
- 'extraction_type' and 'extraction_type_group'
- 'waterpoint_type' and 'waterpoint_type_group'

We will also take a look at 'region' and 'region_code' seperately as while the values are not the same, they contain similar information.

'scheme_management' and 'management'

In [114…

```
wells_df['scheme_management'].value_counts()
```

Out[114…

```
vwc                  36140
unknown               4516
wug                   4248
water authority       3150
wua                   2872
water board           2747
parastatal            1605
private operator      1063
company               1060
swc                     97
trust                   72
Name: scheme_management, dtype: int64
```

In [115…

```
wells_df['management'].value_counts()
```

Out[115…
```
vwc                  39743
wug                   5554
water board           2932
wua                   2526
private operator      1969
parastatal            1691
unknown               1391
water authority        902
company                685
school                  99
trust                   78
Name: management, dtype: int64
```

Since management has fewer unknown values, we will drop the 'scheme_management' column from the DataFrame.

'extraction_type' and 'extraction_type_group'

In [116…
```
wells_df['extraction_type'].value_counts()
```

Out[116…
```
gravity               26687
nira/tanira            7361
unknown                6160
submersible            4687
swn 80                 3447
mono                   2815
india mark ii          2283
afridev                1659
ksb                    1354
rope pump               451
swn 81                  229
windmill                117
india mark iii           91
cemo                     90
play pump                85
climax                   32
walimi                   20
mkulima/shinyanga         2
Name: extraction_type, dtype: int64
```

In [117…
```
wells_df['extraction_type_group'].value_counts()
```

Out[117…
```
gravity             26687
nira/tanira          7361
unknown              6160
submersible          6041
swn 80               3447
mono                 2815
india mark ii        2283
afridev              1659
rope pump             451
other handpump        336
other motorpump       122
wind-powered          117
india mark iii         91
Name: extraction_type_group, dtype: int64
```

Since the 'extraction_type' column is more specific than 'extraction_type_group', we will be dropping 'extraction_type_group' from the DataFrame.

'waterpoint_type' and 'waterpoint_type_group'

In [118…
```
wells_df['waterpoint_type'].value_counts()
```

Out[118…
```
communal standpipe             28360
hand pump                      16179
unknown                         6167
communal standpipe multiple     5958
improved spring                  783
cattle trough                    116
dam                                7
Name: waterpoint_type, dtype: int64
```

In [119…
```
wells_df['waterpoint_type_group'].value_counts()
```

Out[119…
```
communal standpipe    34318
hand pump             16179
unknown                6167
improved spring         783
cattle trough           116
dam                       7
Name: waterpoint_type_group, dtype: int64
```

Since the 'waterpoint_type' column is more specific than 'waterpoint_type_group', we will be dropping 'waterpoint_type_group' from the DataFrame.

'region' and 'region_code'

In [120…
```
len(wells_df['region'].value_counts())
```

Out[120…
```
21
```

In [121…
```
len(wells_df['region_code'].value_counts())
```

Out[121…
```
27
```

Since the 'region_code' contains more values than the region, we will be dropping 'region' from the DataFrame.

Dropping columns

In [122…
```
dimensionality_reduction_df = wells_df.copy()
```

In [123…
```
dimensionality_reduction_df.drop(['scheme_management','extraction_type_group', 'waterpoint_type_group', 'region
```

In [124…
```
dimensionality_reduction_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 57570 entries, 69572 to 26348
Data columns (total 30 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   status_group           57570 non-null  int64  
 1   amount_tsh             57570 non-null  float64
 2   funder                 57570 non-null  object 
 3   gps_height             57570 non-null  int64  
 4   installer              57570 non-null  object 
 5   longitude              57570 non-null  float64
 6   latitude               57570 non-null  float64
 7   wpt_name               57570 non-null  object 
 8   basin                  57570 non-null  object 
 9   subvillage             57570 non-null  object 
 10  region_code            57570 non-null  object 
 11  district_code          57570 non-null  object 
 12  lga                    57570 non-null  object 
 13  ward                   57570 non-null  object 
 14  population             57570 non-null  int64  
 15  scheme_name            57570 non-null  object 
 16  permit                 57570 non-null  object 
 17  construction_year      57570 non-null  int64  
 18  extraction_type        57570 non-null  object 
 19  extraction_type_class  57570 non-null  object 
 20  management             57570 non-null  object 
 21  management_group       57570 non-null  object 
 22  payment                57570 non-null  object 
 23  water_quality          57570 non-null  object 
 24  quality_group          57570 non-null  object 
 25  quantity               57570 non-null  object 
 26  source                 57570 non-null  object 
 27  source_type            57570 non-null  object 
 28  source_class           57570 non-null  object 
 29  waterpoint_type        57570 non-null  object 
dtypes: float64(3), int64(4), object(23)
memory usage: 13.6+ MB
```

Dimensionality Reduction: Logistic Model

In [125…
```
dim_reduct_logistic_model_df = dimensionality_reduction_df.copy()
```

In [126…
```
X_train, X_test, y_train, y_test = split_data(dim_reduct_logistic_model_df, 'status_group')
```

In [127…
```
dim_reduct_logistic_model = LogisticRegression(random_state=42, max_iter=2000)
```

In [128…
```
dim_reduct_log_cv = CustomCrossValidator(dim_reduct_logistic_model, X_train, y_train)
```

In [129…
```
dim_reduct_log_cv.run_sampling_methods(sampling_methods, features=None, features_from='dimensionality_reduction
```

```
Running cross-validation with sampling method: oversample

Running cross-validation with sampling method: undersample

Running cross-validation with sampling method: smote

Log: {('LogisticRegression', 'base', 'oversample'): 0.7043794944655426, ('LogisticRegression', 'base', 'undersam
ple'): 0.7032904517477562, ('LogisticRegression', 'base', 'smote'): 0.7011659423295937, ('DecisionTreeClassifier
', 'base', 'oversample'): 0.5150329468026201, ('DecisionTreeClassifier', 'base', 'undersample'): 0.5310489564026
828, ('DecisionTreeClassifier', 'base', 'smote'): 0.5393838624465799, ('LogisticRegression', 'L1_Penalty', 'over
sample'): 0.7048153281712818, ('LogisticRegression', 'L1_Penalty', 'undersample'): 0.7038352475891534, ('Logisti
cRegression', 'L1_Penalty', 'smote'): 0.7014929177579201, ('DecisionTreeClassifier', 'L1_Penalty', 'oversample')
: 0.5151418792644937, ('DecisionTreeClassifier', 'L1_Penalty', 'undersample'): 0.5311034374705116, ('DecisionTre
eClassifier', 'L1_Penalty', 'smote'): 0.539274915147814, ('LogisticRegression', 'VIF', 'oversample'): 0.68063043
73500361, ('LogisticRegression', 'VIF', 'undersample'): 0.6832991788076954, ('LogisticRegression', 'VIF', 'smote
'): 0.6713702433665739, ('DecisionTreeClassifier', 'VIF', 'oversample'): 0.7701797459805376, ('DecisionTreeClass
ifier', 'VIF', 'undersample'): 0.767510559416115, ('DecisionTreeClassifier', 'VIF', 'smote'): 0.768382701608141,
('LogisticRegression', 'dimensionality_reduction', 'oversample'): 0.7030721120434614, ('LogisticRegression', 'di
mensionality_reduction', 'undersample'): 0.7016560790604568, ('LogisticRegression', 'dimensionality_reduction',
'smote'): 0.6995859765256629}
```

Dimensionality Reduction: Decision Tree

In [130…
```
dim_reduct_dt_model_df = dimensionality_reduction_df.copy()
```

In [131…
```
X_train, X_test, y_train, y_test = split_data(dim_reduct_dt_model_df, 'status_group')
```

In [132…
```
dim_reduct_dt_model = DecisionTreeClassifier(random_state=42, max_depth=5, min_samples_split=20, min_samples_le
```

In [133…
```
dim_reduct_dt_cv = CustomCrossValidator(dim_reduct_dt_model, X_train, y_train)
```

In [134…
```
dim_reduct_dt_cv.run_sampling_methods(sampling_methods, features=None, features_from='dimensionality_reduction'
```

```
Running cross-validation with sampling method: oversample

Running cross-validation with sampling method: undersample

Running cross-validation with sampling method: smote

Log: {('LogisticRegression', 'base', 'oversample'): 0.7043794944655426, ('LogisticRegression', 'base', 'undersam
ple'): 0.7032904517477562, ('LogisticRegression', 'base', 'smote'): 0.7011659423295937, ('DecisionTreeClassifier
', 'base', 'oversample'): 0.5150329468026201, ('DecisionTreeClassifier', 'base', 'undersample'): 0.5310489564026
828, ('DecisionTreeClassifier', 'base', 'smote'): 0.5393838624465799, ('LogisticRegression', 'L1_Penalty', 'over
sample'): 0.7048153281712818, ('LogisticRegression', 'L1_Penalty', 'undersample'): 0.7038352475891534, ('Logisti
cRegression', 'L1_Penalty', 'smote'): 0.7014929177579201, ('DecisionTreeClassifier', 'L1_Penalty', 'oversample')
: 0.5151418792644937, ('DecisionTreeClassifier', 'L1_Penalty', 'undersample'): 0.5311034374705116, ('DecisionTre
eClassifier', 'L1_Penalty', 'smote'): 0.539274915147814, ('LogisticRegression', 'VIF', 'oversample'): 0.68063043
73500361, ('LogisticRegression', 'VIF', 'undersample'): 0.6832991788076954, ('LogisticRegression', 'VIF', 'smote
'): 0.6713702433665739, ('DecisionTreeClassifier', 'VIF', 'oversample'): 0.7701797459805376, ('DecisionTreeClass
ifier', 'VIF', 'undersample'): 0.767510559416115, ('DecisionTreeClassifier', 'VIF', 'smote'): 0.768382701608141,
('LogisticRegression', 'dimensionality_reduction', 'oversample'): 0.7030721120434614, ('LogisticRegression', 'di
mensionality_reduction', 'undersample'): 0.7016560790604568, ('LogisticRegression', 'dimensionality_reduction',
'smote'): 0.6995859765256629, ('DecisionTreeClassifier', 'dimensionality_reduction', 'oversample'): 0.5150329468
026201, ('DecisionTreeClassifier', 'dimensionality_reduction', 'undersample'): 0.5311034374705116, ('DecisionTre
eClassifier', 'dimensionality_reduction', 'smote'): 0.5393293813787509}
```

Review Model Results

In [135…
```
log_df = get_log_results_df()
log_df = log_df.sort_values(by='Recall-Score', ascending=False)
log_df
```

Out[135...

	Model	Feature Selection	Sampling Technique	Recall-Score
15	DecisionTreeClassifier	VIF	oversample	0.770180
17	DecisionTreeClassifier	VIF	smote	0.768383
16	DecisionTreeClassifier	VIF	undersample	0.767511
6	LogisticRegression	L1_Penalty	oversample	0.704815
0	LogisticRegression	base	oversample	0.704379
7	LogisticRegression	L1_Penalty	undersample	0.703835
1	LogisticRegression	base	undersample	0.703290
18	LogisticRegression	dimensionality_reduction	oversample	0.703072
19	LogisticRegression	dimensionality_reduction	undersample	0.701656
8	LogisticRegression	L1_Penalty	smote	0.701493
2	LogisticRegression	base	smote	0.701166
20	LogisticRegression	dimensionality_reduction	smote	0.699586
13	LogisticRegression	VIF	undersample	0.683299
12	LogisticRegression	VIF	oversample	0.680630
14	LogisticRegression	VIF	smote	0.671370
5	DecisionTreeClassifier	base	smote	0.539384
23	DecisionTreeClassifier	dimensionality_reduction	smote	0.539329
11	DecisionTreeClassifier	L1_Penalty	smote	0.539275
10	DecisionTreeClassifier	L1_Penalty	undersample	0.531103
22	DecisionTreeClassifier	dimensionality_reduction	undersample	0.531103
4	DecisionTreeClassifier	base	undersample	0.531049
9	DecisionTreeClassifier	L1_Penalty	oversample	0.515142
3	DecisionTreeClassifier	base	oversample	0.515033
21	DecisionTreeClassifier	dimensionality_reduction	oversample	0.515033

Our best models are the VIF Decision Tree models.

We will hyper-parameter tune all of these models to determine which model performs best of the validation fold.

Then we will use that model on the true hold out test.

Hyperparameter Tuning

In [136...

```
decision_tree_param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [5, 10, 15],
    'min_samples_split': [5, 10, 15],
    'min_samples_leaf': [4, 5, 6],
    'max_features': [None, 'sqrt', 'log2']
}
```

VIF Oversampling Decision Tree Tuning

In [137...

```
tuned_vif_oversample_dt_model_df = wells_df.copy()

X_train, X_test, y_train, y_test = split_data(tuned_vif_oversample_dt_model_df, 'status_group')
```

In [138...

```
tuned_vif_oversample_dt_model = DecisionTreeClassifier(random_state=42)
```

In [139...

```
# Create an instance of CustomCrossValidator
tuned_vif_oversample_dt_cv = CustomCrossValidator(
    model=tuned_vif_oversample_dt_model,
    X=X_train,
    y=y_train,
    sampling_method='oversample'
)
```

In [140...

```
vif_oversample_dt_grid_search = tuned_vif_oversample_dt_cv.run_grid_search(
    param_grid=decision_tree_param_grid,
    features=vif_selected_features,
```

```
    cv=StratifiedKFold(n_splits=5)
)
```

```
Fitting 5 folds for each of 162 candidates, totalling 810 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    5.7s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   15.7s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   30.2s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:   55.7s
[Parallel(n_jobs=-1)]: Done 810 out of 810 | elapsed:   56.7s finished
Fitting 5 folds for each of 162 candidates, totalling 810 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    3.0s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   11.3s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   28.0s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:   56.7s
[Parallel(n_jobs=-1)]: Done 810 out of 810 | elapsed:   58.0s finished
Fitting 5 folds for each of 162 candidates, totalling 810 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    3.5s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   14.2s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   30.7s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:   56.4s
[Parallel(n_jobs=-1)]: Done 810 out of 810 | elapsed:   57.5s finished
Fitting 5 folds for each of 162 candidates, totalling 810 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    3.3s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   12.9s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   30.9s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:   59.1s
[Parallel(n_jobs=-1)]: Done 810 out of 810 | elapsed:  1.0min finished
Fitting 5 folds for each of 162 candidates, totalling 810 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    3.3s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   12.0s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   25.9s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:   48.8s
[Parallel(n_jobs=-1)]: Done 810 out of 810 | elapsed:   49.9s finished
Mean Recall Score: 0.7284020548502097
Best parameters: {'criterion': 'gini', 'max_depth': 5, 'max_features': 'log2', 'min_samples_leaf': 4, 'min_sampl
es_split': 5}
Best score: 0.7414135468421996
```

In [141…

```
vif_oversample_dt_grid_search_best_params = vif_oversample_dt_grid_search.best_params_
```

VIF Undersampling Decision Tree Tuning

In [142…

```
tuned_vif_undersample_dt_model_df = wells_df.copy()

X_train, X_test, y_train, y_test = split_data(tuned_vif_undersample_dt_model_df, 'status_group')
```

In [143…

```
tuned_vif_undersample_dt_model = DecisionTreeClassifier(random_state=42)
```

In [144…

```
# Create an instance of CustomCrossValidator
tuned_vif_undersample_dt_cv = CustomCrossValidator(
    model=tuned_vif_undersample_dt_model,
    X=X_train,
    y=y_train,
    sampling_method='undersample'
)
```

In [145…

```
vif_undersample_dt_grid_search = tuned_vif_undersample_dt_cv.run_grid_search(
    param_grid=decision_tree_param_grid,
    features=vif_selected_features,
    cv=StratifiedKFold(n_splits=5)
)
```

```
Fitting 5 folds for each of 162 candidates, totalling 810 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    2.8s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   10.0s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   22.6s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:   40.3s
[Parallel(n_jobs=-1)]: Done 810 out of 810 | elapsed:   41.0s finished
Fitting 5 folds for each of 162 candidates, totalling 810 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    3.2s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   15.4s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   29.2s

[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:   44.4s
[Parallel(n_jobs=-1)]: Done 810 out of 810 | elapsed:   44.9s finished
Fitting 5 folds for each of 162 candidates, totalling 810 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    2.3s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:    9.5s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   22.5s
[Parallel(n_jobs=-1)]: Done 810 out of 810 | elapsed:   38.9s finished
Fitting 5 folds for each of 162 candidates, totalling 810 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    2.7s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   10.6s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   22.3s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:   45.0s
[Parallel(n_jobs=-1)]: Done 810 out of 810 | elapsed:   45.7s finished
Fitting 5 folds for each of 162 candidates, totalling 810 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    2.4s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:    9.1s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   22.9s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:   43.6s
[Parallel(n_jobs=-1)]: Done 810 out of 810 | elapsed:   44.2s finished
Mean Recall Score: 0.7104807212391298
Best parameters: {'criterion': 'gini', 'max_depth': 10, 'max_features': 'log2', 'min_samples_leaf': 5, 'min_samp
les_split': 15}
Best score: 0.7490241520767522
```

In [146…
```
vif_undersample_dt_grid_search_best_params = vif_undersample_dt_grid_search.best_params_
```

VIF SMOTE Decision Tree Tuning

In [147…
```
tuned_vif_smote_dt_model_df = wells_df.copy()

X_train, X_test, y_train, y_test = split_data(tuned_vif_smote_dt_model_df, 'status_group')
```

In [148…
```
tuned_vif_smote_dt_model = DecisionTreeClassifier(random_state=42)
```

In [149…
```
# Create an instance of CustomCrossValidator
tuned_vif_smote_dt_cv = CustomCrossValidator(
    model=tuned_vif_smote_dt_model,
    X=X_train,
    y=y_train,
    sampling_method='smote'
)
```

In [150…
```
vif_smote_dt_grid_search = tuned_vif_smote_dt_cv.run_grid_search(
    param_grid=decision_tree_param_grid,
    features=vif_selected_features,
    cv=StratifiedKFold(n_splits=5)
)
```

```
Fitting 5 folds for each of 162 candidates, totalling 810 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    3.4s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   13.8s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   34.5s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 810 out of 810 | elapsed:  1.1min finished
Fitting 5 folds for each of 162 candidates, totalling 810 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    3.3s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   12.9s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   30.8s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:  1.0min
[Parallel(n_jobs=-1)]: Done 810 out of 810 | elapsed:  1.0min finished
Fitting 5 folds for each of 162 candidates, totalling 810 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    4.7s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   15.8s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   33.8s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:   56.7s
[Parallel(n_jobs=-1)]: Done 810 out of 810 | elapsed:   58.1s finished
Fitting 5 folds for each of 162 candidates, totalling 810 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    3.7s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   13.2s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   35.3s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 810 out of 810 | elapsed:  1.1min finished
Fitting 5 folds for each of 162 candidates, totalling 810 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    3.6s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   14.6s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   31.4s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:   52.4s
[Parallel(n_jobs=-1)]: Done 810 out of 810 | elapsed:   53.2s finished
Mean Recall Score: 0.7214271280109246
Best parameters: {'criterion': 'entropy', 'max_depth': 10, 'max_features': 'log2', 'min_samples_leaf': 4, 'min_s
amples_split': 10}
Best score: 0.7406721731700371
```

In [151…

```
vif_smote_dt_grid_search_best_params = vif_smote_dt_grid_search.best_params_
```

Oversampling yielded a better scoring model without hyperparameter tuning and had the highest mean recall score amongst all validation folds, the undersampling approach produced the best individual model on the validation set.

Given our business problem, we want to choose the model that performs consistently over a one-time better test.

As such, we have choosen the oversampling model as our best model.

Fitting the Tuned Best Model

Fit and Feature Importance

In [173…

```
final_model_df = wells_df.copy()

X_train, X_test, y_train, y_test = split_data(final_model_df, 'status_group')
```

In [174…

```
final_model = DecisionTreeClassifier(
    random_state=42,
    **vif_oversample_dt_grid_search_best_params
)
```

In [175…

```
X_train_numeric_scaled_df, X_test_numeric_scaled_df = scale_numeric_features(X_train, X_test)
```

In [176…

```
encoded_train_df, encoded_test_df = encode_categorical_features(X_train, X_test)
```

In [177…

```
x_train_preprocessed_ = combine_features(X_train_numeric_scaled_df, encoded_train_df)
x_test_preprocessed = combine_features(X_test_numeric_scaled_df, encoded_test_df)
```

In [178…

```
# Fit the model on the preprocessed training data
final_model.fit(X_train_preprocessed, y_train)
```

Out[178…

```
DecisionTreeClassifier(max_depth=5, max_features='log2', min_samples_leaf=4,
                       min_samples_split=5, random_state=42)
```

In [179…

```
# Get feature importances
importances = final_model.feature_importances_
```

In [180…

```
# Get column names after preprocessing
preprocessed_columns = X_train_preprocessed.columns
```

In [181…

```
# Create a DataFrame to hold feature importances with corresponding names
feature_importances = pd.DataFrame({'feature': preprocessed_columns, 'importance': importances})
```

In [182…

```
# Sort features by importance in descending order
feature_importances = feature_importances.sort_values(by='importance', ascending=False)
```

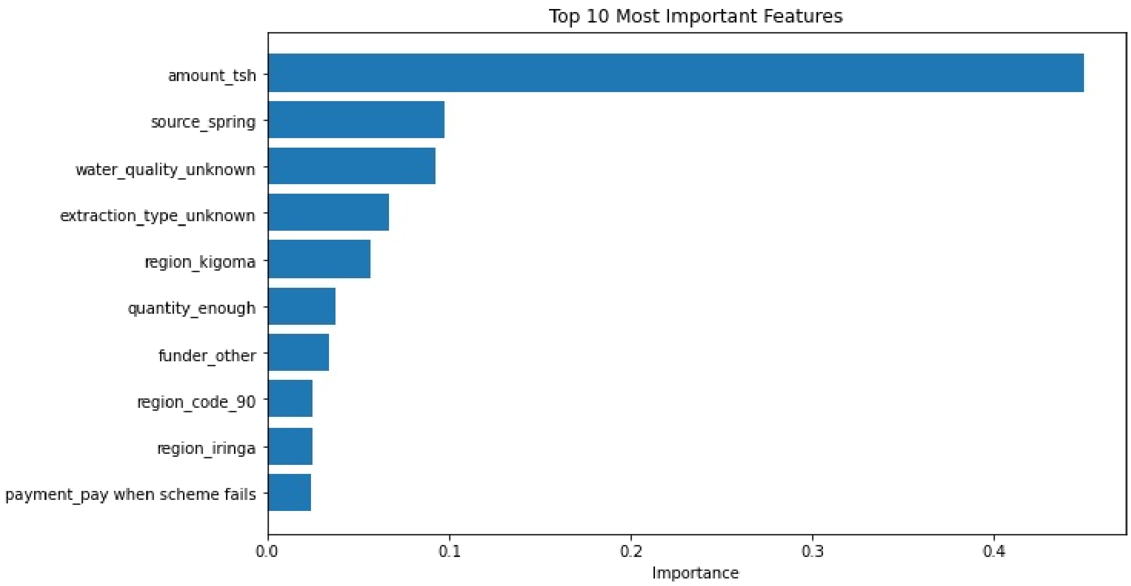
In [183…

```
# Print or visualize the top features
print("Top 10 Most Important Features:")
print(feature_importances.head(10))
```

```
Top 10 Most Important Features:
                        feature  importance
0                    amount_tsh    0.449971
259               source_spring    0.097126
240       water_quality_unknown    0.092763
189     extraction_type_unknown    0.066650
59                region_kigoma    0.057295
248             quantity_enough    0.037538
15                 funder_other    0.033881
99               region_code_90    0.024713
57                region_iringa    0.024641
231  payment_pay when scheme fails    0.024184
```

In [184…

```python
plt.figure(figsize=(10, 6))
plt.barh(feature_importances['feature'][:10], feature_importances['importance'][:10], align='center')
plt.xlabel('Importance')
plt.title('Top 10 Most Important Features')
plt.gca().invert_yaxis()  # Invert y-axis to have the most important feature at the top
plt.show()
```



Predicting Using the Hold Out Test

In [185…

```python
# Predict on the preprocessed test data
y_pred = final_model.predict(X_test_preprocessed)
```

In [186…

```python
final_model_recall_score = recall_score(y_test, y_pred)
print(f"Recall on test set: {final_model_recall_score:.4f}")
```

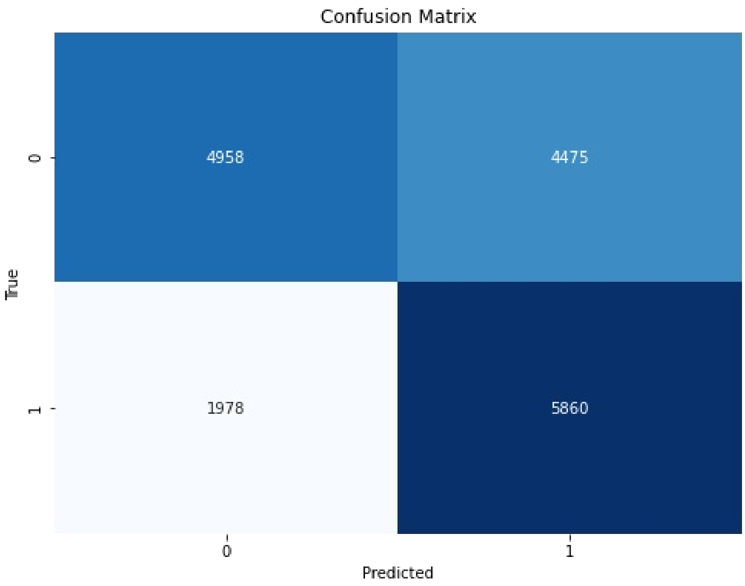
```
Recall on test set: 0.7476
```

In [187…

```python
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
```

In [188…

```python
# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



Confusion Matrix Breakdown:

- True Positive (TP): Correctly predicting a well as "Does not need attention" (0).
- False Positive (FP): Incorrectly predicting a well as "Needs attention" (1) when it actually does not need attention.
- True Negative (TN): Correctly predicting a well as "Needs attention" (1).
- False Negative (FN): Incorrectly predicting a well as "Does not need attention" (0) when it actually needs attention.

There is a slight improvement in recall score on the test data compared to the training data when using the oversampling model with VIF-selected features.

This suggests that our model has generalized well to unseen data. Which given the drop in recall score from the train to the hyperparameter tuned train alleviates some of the fears around overfitting.

The recall score on the training data was 0.7414, while the test data scored 0.7476, indicating that the model can effectively identify wells needing attention.

Recommendations

Business Recommendations

Prioritize wells in the northwest and southeast of the country

Out of the top twenty regions that have wells in need of attention, these regions are in the top 20.

If you could only prioritize one region, start in the northwest of the country.

Northwest:

- Kagera
- Kigoma
- Shinyanga
- Mwanza
- Mara

Southeast:

- Lindi
- Mtwara
- Iringa

For new wells, look to improve or use the latest version of gravity extraction pumps

Gravity extraction pumps are the most prevalent extraction type used in wells that need attention.

This is counterintuitive as gravity powered well pump should not be as susceptible to wear and tear as a motorized or centrifugal pumps.

It appears as though the construction quality and materials used could be playing a role here.

Predicitive Recommendation

All data provided was surveyed by GeoData Consultants Ltd. Our prediction model can be used in place of consultants, cutting costs.

Benefit of the model

- Can assist in efficient resource allocation and proactive maintenance planning.
    - **Note**
        - The model is based on historic results, will not be apt for use during natural disasters but can be used before-hand to identify high priority regions.

```
In [ ]:
```