Detection of Non-functional Water Wells Using Machine Learning Algorithms

Tanzanian Water Wells

Overview

Tanzania, as a developing country, struggles with providing clean water to its population of over 57,000,000. There are many wells (water points) already established in the country, but some are in need of repair while others have failed altogether. The Government of Tanzania is looking to find patterns in non-functional wells to influence how new wells are built.

Business Problem

Tanzania faces challenges in ensuring access to clean water for its population due to non-functional water wells. We will be looking to predict patterns in non-functional wells to inform more robust construction methods for new wells.

Data Understanding

Dataset

Driven Data - Tanzanian Water Wells

- Labels
- Values

Target

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

Features

- amount_tsh : Total static head (amount water available to well)
- date_recorded : The date the row was entered
- funder : Who funded the well
- gps_height : Altitude of the well
- installer : Organization that installed the well
- longitude : GPS coordinate
- latitude : GPS coordinate
- wpt_name : Name of the well if there is one

- num_private :Private use or not
- basin : Geographic water basin
- subvillage : Geographic location
- region : Geographic location
- region_code : Geographic location (coded)
- district_code : Geographic location (coded)
- lga : Geographic location
- ward : Geographic location
- population : Population around the well
- public_meeting : True/False
- recorded_by : Group entering this row of data
- scheme_management : Who operates the well
- scheme_name : Who operates the well
- permit : If the well is permitted
- construction_year : Year the well was constructed
- extraction_type : The kind of extraction the well uses
- extraction_type_group : The kind of extraction the well uses
- extraction_type_class : The kind of extraction the well uses
- management : How the well is managed
- management_group : How the well is managed
- payment : What the water costs
- payment_type : What the water costs
- water_quality : The quality of the water
- quality_group : The quality of the water
- quantity : The quantity of water
- quantity_group : The quantity of water
- source : The source of the water
- source_type : The source of the water
- source_class : The source of the water
- waterpoint_type : The kind of well
- waterpoint_type_group : The kind of well

Limitations

- Do not know what types of repairs are needed
- Not able to use the well age, due to missing construction years
- Without full population information, we do not know the supply needs

Data Preparation

Import and Read Datasets

In [3]:
```
# Import standard packages
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import datetime

# Model Selection
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_sc

# Classification Models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.dummy import DummyClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import plot_confusion_matrix, accuracy_score, f1_score, pre
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import roc_curve, auc, roc_auc_score

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

# Categorical Create Dummies
from sklearn.preprocessing import OneHotEncoder

# Column Transformer
from sklearn.compose import ColumnTransformer

# Pipeline
from sklearn.pipeline import Pipeline

# Base
from sklearn.base import BaseEstimator, TransformerMixin
```

In [4]:

```
# Load the datasets
features_data = 'data/WellWaterData.csv'
target_data = 'data/TargetData.csv'

features = pd.read_csv(features_data)
target = pd.read_csv(target_data)

# Display the contents of the datasets
features.head(), target.head()
```

Out[4]:

```
(      id  amount_tsh date_recorded        funder  gps_height     installer  \
 0  69572      6000.0    2011-03-14         Roman        1390         Roman
 1   8776         0.0    2013-03-06       Grumeti        1399       GRUMETI
 2  34310        25.0    2013-02-25  Lottery Club         686  World vision
 3  67743         0.0    2013-01-28        Unicef         263        UNICEF
 4  19728         0.0    2011-07-13   Action In A           0       Artisan

    longitude   latitude              wpt_name  num_private  ... payment_type  \
 0  34.938093  -9.856322                  none            0  ...     annually
 1  34.698766  -2.147466              Zahanati            0  ...    never pay
 2  37.460664  -3.821329           Kwa Mahundi            0  ...   per bucket
 3  38.486161 -11.155298  Zahanati Ya Nanyumbu            0  ...    never pay
 4  31.130847  -1.825359               Shuleni            0  ...    never pay

   water_quality quality_group      quantity  quantity_group  \
 0          soft          good        enough          enough
 1          soft          good  insufficient    insufficient
```

```
2          soft          good        enough          enough
3          soft          good           dry             dry
4          soft          good      seasonal        seasonal

                source           source_type  source_class  \
0               spring                spring   groundwater
1  rainwater harvesting  rainwater harvesting       surface
2                  dam                   dam       surface
3          machine dbh              borehole   groundwater
4  rainwater harvesting  rainwater harvesting       surface

              waterpoint_type waterpoint_type_group
0          communal standpipe    communal standpipe
1          communal standpipe    communal standpipe
2 communal standpipe multiple    communal standpipe
3 communal standpipe multiple    communal standpipe
4          communal standpipe    communal standpipe

[5 rows x 40 columns],
      id    status_group
0  69572      functional
1   8776      functional
2  34310      functional
3  67743  non functional
4  19728      functional)
```

Merging Datasets

In [5]:
```
# Checking for unique IDs in both datasets to ensure they match
unique_ids_features = features['id'].nunique()
unique_ids_target = target['id'].nunique()

unique_ids_features, unique_ids_target
```

Out[5]:
```
(59400, 59400)
```

In [6]:
```
# Merging the datasets on the 'id' column
merged_data = pd.merge(features, target, on='id')

# Displaying the first few rows of the merged dataset
merged_data.head()
```

Out[6]:

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wpt
0	69572	6000.0	2011-03-14	Roman	1390	Roman	34.938093	-9.856322	
1	8776	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.698766	-2.147466	Z
2	34310	25.0	2013-02-25	Lottery Club	686	World vision	37.460664	-3.821329	M
3	67743	0.0	2013-01-28	Unicef	263	UNICEF	38.486161	-11.155298	Z Nar
4	19728	0.0	2011-07-13	Action In A	0	Artisan	31.130847	-1.825359	

5 rows × 41 columns

In [7]:

```
merged_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59400 entries, 0 to 59399
Data columns (total 41 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     59400 non-null  int64  
 1   amount_tsh             59400 non-null  float64
 2   date_recorded          59400 non-null  object 
 3   funder                 55765 non-null  object 
 4   gps_height             59400 non-null  int64  
 5   installer              55745 non-null  object 
 6   longitude              59400 non-null  float64
 7   latitude               59400 non-null  float64
 8   wpt_name               59400 non-null  object 
 9   num_private            59400 non-null  int64  
 10  basin                  59400 non-null  object 
 11  subvillage             59029 non-null  object 
 12  region                 59400 non-null  object 
 13  region_code            59400 non-null  int64  
 14  district_code          59400 non-null  int64  
 15  lga                    59400 non-null  object 
 16  ward                   59400 non-null  object 
 17  population             59400 non-null  int64  
 18  public_meeting         56066 non-null  object 
 19  recorded_by            59400 non-null  object 
 20  scheme_management      55523 non-null  object 
 21  scheme_name            31234 non-null  object 
 22  permit                 56344 non-null  object 
 23  construction_year      59400 non-null  int64  
 24  extraction_type        59400 non-null  object 
 25  extraction_type_group  59400 non-null  object 
 26  extraction_type_class  59400 non-null  object 
 27  management             59400 non-null  object 
 28  management_group       59400 non-null  object 
 29  payment                59400 non-null  object 
 30  payment_type           59400 non-null  object 
 31  water_quality          59400 non-null  object 
 32  quality_group          59400 non-null  object 
 33  quantity               59400 non-null  object 
 34  quantity_group         59400 non-null  object 
 35  source                 59400 non-null  object 
```

```
 36  source_type            59400 non-null  object
 37  source_class           59400 non-null  object
 38  waterpoint_type        59400 non-null  object
 39  waterpoint_type_group  59400 non-null  object
 40  status_group           59400 non-null  object
dtypes: float64(3), int64(7), object(31)
memory usage: 19.0+ MB
```

In [8]:

```
merged_data.value_counts()
```

Out[8]:

```
id     amount_tsh  date_recorded  funder                  gps_height  installer
longitude  latitude   wpt_name          num_private  basin                    s
ubvillage  region       region_code  district_code  lga           ward
population  public_meeting  recorded_by              scheme_management  scheme_n
ame                   permit  construction_year  extraction_type  extraction_typ
e_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  s
tatus_group
74247  50.0        2013-02-16     Mission                 965         DWE
35.432998  -10.639270  Kwa Mapunda       0           Ruvuma / Southern Coast  M
pakani     Ruvuma       10           2             Songea Rural  Maposeni
900        True            GeoData Consultants Ltd  VWC                Mradi wa
maji wa peramiho     True    2009               other            other
other                  vwc         user-group        pay per bucket         per
bucket    soft           good           dry           dry             river
river/lake   surface       communal standpipe           communal standpipe     n
on functional    1
24588  0.0         2013-03-23     Government Of Tanzania  1344        DWE
37.544739  -3.291398   Kwa Bariki Kombe  0           Pangani                  B
arazani    Kilimanjaro  3            4             Moshi Rural   Mamba Kusini
1          False           GeoData Consultants Ltd  VWC                Una mkol
owoni                 True    1972               gravity          gravity
gravity                vwc         user-group        never pay              neve
r pay     soft           good           insufficient  insufficient    spring
spring       groundwater   other                        other                 n
on functional    1
24558  0.0         2011-07-24     Wananchi                0           wananchi
33.814988  -9.490739   Kwa Asukile       0           Lake Nyasa               B
ugoba      Mbeya        12           3             Kyela         Ipande
0          True            GeoData Consultants Ltd  VWC                Sinyanga
water supplied sc   True    0                gravity          gravity
gravity                vwc         user-group        never pay              neve
r pay     soft           good           dry           dry             spring
spring       groundwater   communal standpipe           communal standpipe     n
on functional    1
24563  0.0         2011-03-14     Go                      526         Go
36.990775  -7.400210   Bustanini         0           Rufiji                   M
jini       Morogoro     5            1             Kilosa        Mikumi
250        True            GeoData Consultants Ltd  Company             Mi
True    1975               gravity          gravity                gravity
company     commercial        never pay              never pay     soft
good           enough        enough         river         river/lake   surface
communal standpipe          communal standpipe     functional       1
24564  0.0         2013-07-03     Government Of Tanzania  1232        RWE
36.874949  -3.343532   Aminieli Nanyaru  0           Pangani                  K
iwawa      Arusha       2            7             Meru          Maji ya Chai
120        True            GeoData Consultants Ltd  VWC                 Tuvaila
gravity water supply  True    1968               gravity          gravity
gravity                wug         user-group        unknown                unkn
own       soft           good           enough        enough          river
river/lake   surface       communal standpipe           communal standpipe     f
unctional       1

..
```

```
49306  500.0       2013-02-13     Kiwanda Cha Tangawizi   1215        District C
ouncil  37.989865  -4.390224   Kwa Mzee Mkota    0            Pangani
Goha        Kilimanjaro  3          3             Same          Mnyamba
38          True            GeoData Consultants Ltd  VWC                Tangawiz
i Water Supply        True    2012              gravity          gravity
gravity               vwc         user-group        pay when scheme fails  on f
ailure    soft           good          enough        enough          spring
spring       groundwater   communal standpipe         communal standpipe     f
unctional        1
49308  0.0         2011-04-17     Water                   0           DWE
35.915891  -4.695893   Ikova Mwisho      0           Internal                I
kova       Dodoma       1            1             Kondoa        Pahi
0           True            GeoData Consultants Ltd  VWC                Pahi
False   0                  gravity          gravity                 gravity
vwc         user-group        never pay              never pay     soft
good           enough         enough          spring        spring       groundwat
er   communal standpipe          communal standpipe     non functional    1
49310  0.0         2011-04-05     Government Of Tanzania  1488        DWE
38.286117  -4.822377   Kwa Mzee Hoza     0           Pangani                 N
yankei     Tanga        4            1             Lushoto       Ubiri
1           True            GeoData Consultants Ltd  VWC                Ilente s
treem                 True    1981              gravity          gravity
gravity               vwc         user-group        never pay               neve
r pay     soft           good          enough        enough          spring
spring       groundwater   communal standpipe         communal standpipe     n
on functional    1
49311  0.0         2011-03-16     Government Of Tanzania  1793        DWE
34.828328  -9.016824   Kwa Esau Lulambo  0           Rufiji                  L
yalamo     Iringa       11           4             Njombe        Mtwango
35          True            GeoData Consultants Ltd  VWC                Ilunda p
umping scheme         False   1976              gravity          gravity
gravity               vwc         user-group        never pay               neve
r pay     soft           good          enough        enough          spring
spring       groundwater   communal standpipe         communal standpipe     n
on functional    1
2      0.0         2011-03-27     Lvia                    0           LVIA
36.115056  -6.279268   Bombani           0           Wami / Ruvu             S
ongambele  Dodoma       1            4             Chamwino      Msamalo
0           True            GeoData Consultants Ltd  VWC                Mgun
True    0                  mono             mono                    motorpump
vwc         user-group        pay per bucket         per bucket    soft
good           insufficient  insufficient    machine dbh  borehole       groundwat
er   communal standpipe multiple  communal standpipe     functional        1
Length: 27813, dtype: int64
```

Clean

Preparing the merged dataset for feature exploration and how they relate to the 'status_group' target variable.

In [9]:
```
# Converting 'date_recorded' to datetime
merged_data['date_recorded'] = pd.to_datetime(merged_data['date_recorded'])
```

Addressing Categorical Features with Parent and Subgroup Columns

In [10]:
```
grouped = merged_data.groupby(['extraction_type_class', 'extraction_type_group',
print(grouped)
```

```
extraction_type_class  extraction_type_group  extraction_type
gravity                gravity                gravity                       26780
handpump               afridev                afridev                        1770
```

```
                       india mark ii         india mark ii                2400
                       india mark iii        india mark iii                 98
                       nira/tanira           nira/tanira                  8154
                       other handpump        other - mkulima/shinyanga       2
                                             other - play pump              85
                                             other - swn 81                229
                                             walimi                         48
                       swn 80                swn 80                       3670
motorpump              mono                  mono                         2865
                       other motorpump       cemo                           90
                                             climax                         32
other                  other                 other                        6430
rope pump              rope pump             other - rope pump             451
submersible            submersible           ksb                          1415
                                             submersible                  4764
wind-powered           wind-powered          windmill                      117
dtype: int64
```

In [11]:

```
grouped = merged_data.groupby(['management_group', 'management']).size()
print(grouped)
```

```
management_group  management
commercial        company                685
                  private operator      1971
                  trust                   78
                  water authority        904
other             other                  844
                  other - school          99
parastatal        parastatal            1768
unknown           unknown                561
user-group        vwc                  40507
                  water board           2933
                  wua                   2535
                  wug                   6515
dtype: int64
```

In [12]:

```
grouped = merged_data.groupby(['waterpoint_type_group', 'waterpoint_type']).size
print(grouped)
```

```
waterpoint_type_group  waterpoint_type
cattle trough          cattle trough                    116
communal standpipe     communal standpipe             28522
                       communal standpipe multiple     6103
dam                    dam                                7
hand pump              hand pump                      17488
improved spring        improved spring                  784
other                  other                           6380
dtype: int64
```

Handling Missing Values

In [13]:

```
# Calculating the percentage of zero values for each column
zero_value_percentages = {}
for column in merged_data.columns:
    zero_count = (merged_data[column] == 0).sum()
    zero_value_percentages[column] = (zero_count / len(merged_data)) * 100


zero_value_percentages
```

Out[13]:

```
{'id': 0.0016835016835016834,
 'amount_tsh': 70.09932659932659,
 'date_recorded': 0.0,
```

```
 'funder': 0.0,
 'gps_height': 34.40740740740741,
 'installer': 0.0,
 'longitude': 3.0505050505050505,
 'latitude': 0.0,
 'wpt_name': 0.0,
 'num_private': 98.72558922558923,
 'basin': 0.0,
 'subvillage': 0.0,
 'region': 0.0,
 'region_code': 0.0,
 'district_code': 0.038720538720538725,
 'lga': 0.0,
 'ward': 0.0,
 'population': 35.994949494949495,
 'public_meeting': 8.51010101010101,
 'recorded_by': 0.0,
 'scheme_management': 0.0,
 'scheme_name': 0.0,
 'permit': 29.447811447811445,
 'construction_year': 34.86363636363636,
 'extraction_type': 0.0,
 'extraction_type_group': 0.0,
 'extraction_type_class': 0.0,
 'management': 0.0,
 'management_group': 0.0,
 'payment': 0.0,
 'payment_type': 0.0,
 'water_quality': 0.0,
 'quality_group': 0.0,
 'quantity': 0.0,
 'quantity_group': 0.0,
 'source': 0.0,
 'source_type': 0.0,
 'source_class': 0.0,
 'waterpoint_type': 0.0,
 'waterpoint_type_group': 0.0,
 'status_group': 0.0}
```

In [14]:
```
# Calculating the percentage of missing values in each column
missing_values = merged_data.isnull().mean() * 100
missing_values = missing_values[missing_values > 0].sort_values(ascending=False)

missing_values
```

Out[14]:
```
scheme_name          47.417508
scheme_management     6.526936
installer             6.153199
funder                6.119529
public_meeting        5.612795
permit                5.144781
subvillage            0.624579
dtype: float64
```

Since 'scheme_management', 'installer', 'funder', and 'permit' have less than 7% missing values and are potentially relevant, replacing them is a good option.

In [15]:
```
# Replacing the missing values
for column in ['scheme_management', 'installer', 'funder', 'permit']:
    merged_data[column].fillna('Unknown', inplace=True)

# Sanity check on missing values
```

```
remaining_missing_values = merged_data.isnull().sum()
remaining_missing_values[remaining_missing_values > 0]
```

Out[15]:
```
subvillage          371
public_meeting     3334
scheme_name       28166
dtype: int64
```

Dropping Columns

In [16]:
```
# Dropping additional group columns
# Dropping irrelvant columns
# Dropping features with missing values over 45%
features_dropped = merged_data.drop(columns=['quantity_group', 'extraction_type_
                                             'waterpoint_type_group', 'managemer
                                             'payment', 'water_quality', 'source
                                             'region_code', 'public_meeting', 'r
                                             'wpt_name', 'district_code', 'id',
                                             'ward', 'subvillage', 'extraction_t


features_dropped.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59400 entries, 0 to 59399
Data columns (total 21 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   amount_tsh             59400 non-null  float64
 1   date_recorded          59400 non-null  datetime64[ns]
 2   funder                 59400 non-null  object
 3   gps_height             59400 non-null  int64
 4   installer              59400 non-null  object
 5   longitude              59400 non-null  float64
 6   latitude               59400 non-null  float64
 7   basin                  59400 non-null  object
 8   region                 59400 non-null  object
 9   population             59400 non-null  int64
 10  scheme_management      59400 non-null  object
 11  permit                 59400 non-null  object
 12  construction_year      59400 non-null  int64
 13  extraction_type_class  59400 non-null  object
 14  management             59400 non-null  object
 15  payment_type           59400 non-null  object
 16  quality_group          59400 non-null  object
 17  quantity               59400 non-null  object
 18  source_type            59400 non-null  object
 19  waterpoint_type        59400 non-null  object
 20  status_group           59400 non-null  object
dtypes: datetime64[ns](1), float64(3), int64(3), object(14)
memory usage: 10.0+ MB
```

Removing Duplicates

In [17]:
```
features_dropped.duplicated().sum()
```

Out[17]:
```
685
```

In [18]:
```
data_dedup = features_dropped.drop_duplicates()

# Rechecking for duplicates
```

```
new_duplicate_count = data_dedup.duplicated().sum()
new_duplicate_count
```

Out[18]: 0

Feature Engineering

In [19]:
```
# Extracting year and month from 'date_recorded'
data_dedup['year_recorded'] = data_dedup['date_recorded'].dt.year
data_dedup['month_recorded'] = data_dedup['date_recorded'].dt.month

data_dedup[['longitude', 'gps_height', 'construction_year', 'year_recorded', 'mo
```

```
<ipython-input-19-a25e6a562f69>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_dedup['year_recorded'] = data_dedup['date_recorded'].dt.year
<ipython-input-19-a25e6a562f69>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_dedup['month_recorded'] = data_dedup['date_recorded'].dt.month
```

Out[19]:

	longitude	gps_height	construction_year	year_recorded	month_recorded
0	34.938093	1390	1999	2011	3
1	34.698766	1399	2010	2013	3
2	37.460664	686	2009	2013	2
3	38.486161	263	1986	2013	1
4	31.130847	0	0	2011	7

In [20]:
```
# Adding 'well_age' feature
data_dedup['well_age'] = data_dedup.apply(lambda row: 0 if row['construction_yea
                                           else row['year_recorded'] - row['const

# Displaying the first 50 rows to check the 'construction_year' and 'well_age'
data_dedup[['year_recorded', 'construction_year', 'well_age']].value_counts()
```

```
<ipython-input-20-aef931d43ca2>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_dedup['well_age'] = data_dedup.apply(lambda row: 0 if row['construction_y
ear'] == 0
```

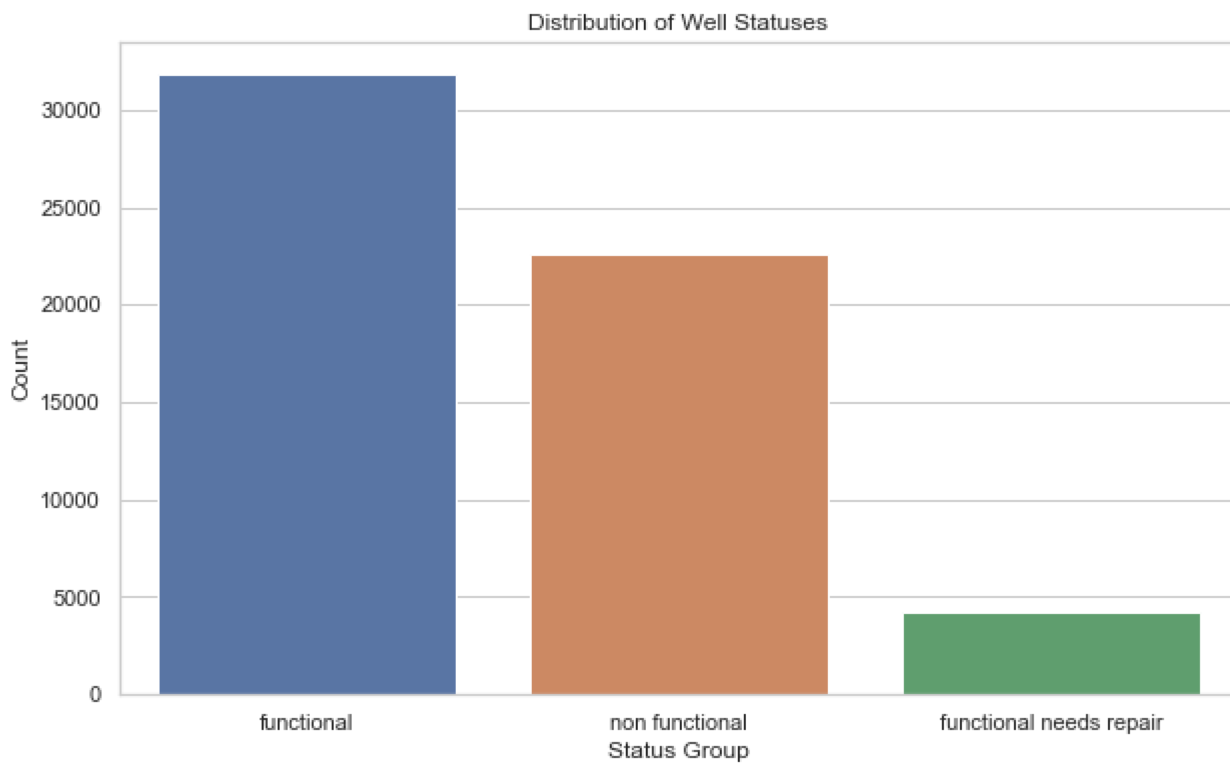
Out[20]:
```
year_recorded  construction_year  well_age
2011           0                  0           13104
2012           0                  0            5000
2013           0                  0            1906
               2000               13           1506
               2010               3            1407
                                                ...
```

```
2004           2005           -1            1
               2006           -2            1
               2007           -3            1
2012           1984            28           1
2002           0               0            1
Length: 168, dtype: int64
```

Creating Binary Target Column

First, we'll need to understand the distribution of the target variable and visualize the proportion of functional vs. non-functional wells.

In [21]:
```
sns.set(style="whitegrid")

# Plotting the distribution of well statuses
plt.figure(figsize=(10, 6))
sns.countplot(x='status_group', data=data_dedup)
plt.title('Distribution of Well Statuses')
plt.ylabel('Count')
plt.xlabel('Status Group')
plt.show()
```



To address differing opinions on how to condense our target into a binary column, we will create two separte binary target columns and assess the better performer on our baseline model.

status_binary:

- Class 0 = non-functional & functional needs repair
- Class 1 = functional

status_binary_reversed:

- Class 0 = non-functional

- Class 1 = functional & functional needs repair

In [22]:
```
# Binary encoding of the 'status_group' column

# 'functional' is assigned 1 and 'non functional' or 'functional needs repair' a
data_dedup['status_binary'] = data_dedup['status_group'].apply(lambda x: 1 if x 

# 'non-functional' is assigned 0 and 'functional' or 'functional needs repair' a
data_dedup['status_binary_reversed'] = data_dedup['status_group'].apply(lambda x

data_dedup[['status_group', 'status_binary', 'status_binary_reversed']].head(20)
```

```
<ipython-input-22-e016e00ec615>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_dedup['status_binary'] = data_dedup['status_group'].apply(lambda x: 1 if
x == 'functional' else 0)
<ipython-input-22-e016e00ec615>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  data_dedup['status_binary_reversed'] = data_dedup['status_group'].apply(lambda
x: 0 if x == 'non functional' else 1)
```

Out[22]:

	status_group	status_binary	status_binary_reversed
0	functional	1	1
1	functional	1	1
2	functional	1	1
3	non functional	0	0
4	functional	1	1
5	functional	1	1
6	non functional	0	0
7	non functional	0	0
8	non functional	0	0
9	functional	1	1
10	functional	1	1
11	functional	1	1
12	functional	1	1
13	functional	1	1
14	functional	1	1
15	functional	1	1
16	non functional	0	0
17	non functional	0	0

	status_group	status_binary	status_binary_reversed
18	functional needs repair	0	1
19	functional	1	1

Creating Master Dataset

In [23]:
```
# Creating master dataset with all values of 'well_age' greater than or equal to
# to eliminate negative values where 'recorded_year' was likely listed inaccurat
master_data = data_dedup[data_dedup['well_age'] >= 0]
master_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 58706 entries, 0 to 59399
Data columns (total 26 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   amount_tsh              58706 non-null  float64
 1   date_recorded           58706 non-null  datetime64[ns]
 2   funder                  58706 non-null  object
 3   gps_height              58706 non-null  int64
 4   installer               58706 non-null  object
 5   longitude               58706 non-null  float64
 6   latitude                58706 non-null  float64
 7   basin                   58706 non-null  object
 8   region                  58706 non-null  object
 9   population              58706 non-null  int64
 10  scheme_management       58706 non-null  object
 11  permit                  58706 non-null  object
 12  construction_year       58706 non-null  int64
 13  extraction_type_class   58706 non-null  object
 14  management              58706 non-null  object
 15  payment_type            58706 non-null  object
 16  quality_group           58706 non-null  object
 17  quantity                58706 non-null  object
 18  source_type             58706 non-null  object
 19  waterpoint_type         58706 non-null  object
 20  status_group            58706 non-null  object
 21  year_recorded           58706 non-null  int64
 22  month_recorded          58706 non-null  int64
 23  well_age                58706 non-null  int64
 24  status_binary           58706 non-null  int64
 25  status_binary_reversed  58706 non-null  int64
dtypes: datetime64[ns](1), float64(3), int64(8), object(14)
memory usage: 12.1+ MB
```

In [24]:
```
# Saving master dataset to csv
master_data.to_csv('data/master_data.csv', index=False)
```

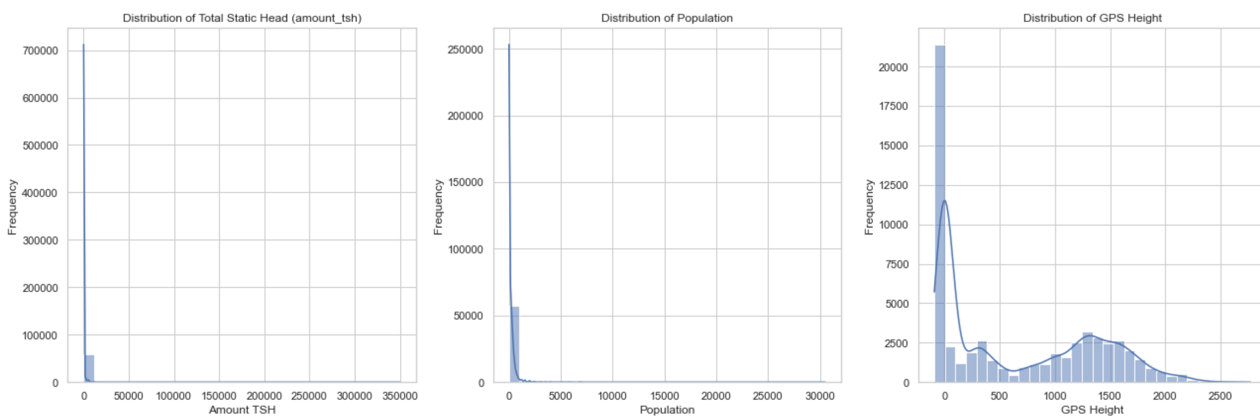
EDA

Data Distribution Visualizations

Using histograms, we'll plot the distribution of key numeric variables like amount_tsh, population, and gps_height.

In [25]:
```
sns.set_style("whitegrid")

# Creating histograms for 'amount_tsh', 'population', and 'gps_height'
```

```
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Plot for 'amount_tsh'
sns.histplot(master_data['amount_tsh'], bins=30, ax=axes[0], kde=True)
axes[0].set_title('Distribution of Total Static Head (amount_tsh)')
axes[0].set_xlabel('Amount TSH')
axes[0].set_ylabel('Frequency')

# Plot for 'population'
sns.histplot(master_data['population'], bins=30, ax=axes[1], kde=True)
axes[1].set_title('Distribution of Population')
axes[1].set_xlabel('Population')
axes[1].set_ylabel('Frequency')

# Plot for 'gps_height'
sns.histplot(master_data['gps_height'], bins=30, ax=axes[2], kde=True)
axes[2].set_title('Distribution of GPS Height')
axes[2].set_xlabel('GPS Height')
axes[2].set_ylabel('Frequency')

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



Observations

These distributions reflect the high number of zero values in our dataset for these features.

Total Static Head (amount_tsh): The distribution appears to be highly skewed to the right, indicating that most wells have a low static head value.

Population: This distribution is also right-skewed, showing that most wells serve a relatively small population, with fewer points serving larger populations.

GPS Height: The distribution is more varied, indicating a range of elevations at which wells are located.

Geographical Analysis

We'll create a geographical plot using latitude and longitude to see if there is any geographical pattern in the status of wells.

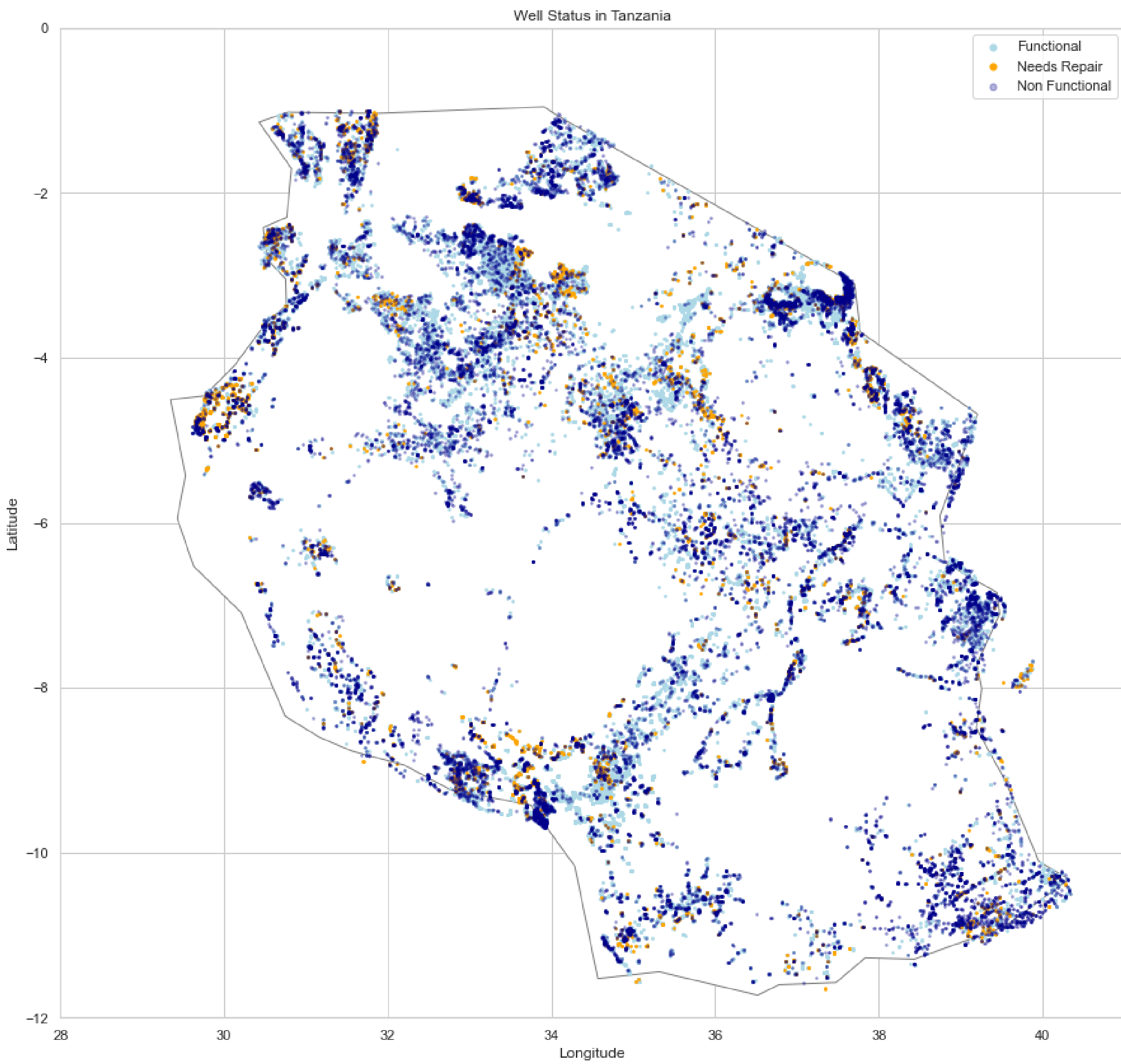
In [26]:
```
!pip install geopandas
```

```
Requirement already satisfied: geopandas in /Users/kariprimiano/anaconda3/envs/l
earn-env/lib/python3.8/site-packages (0.13.2)
Requirement already satisfied: pyproj>=3.0.1 in /Users/kariprimiano/anaconda3/en
vs/learn-env/lib/python3.8/site-packages (from geopandas) (3.5.0)
Requirement already satisfied: fiona>=1.8.19 in /Users/kariprimiano/anaconda3/en
vs/learn-env/lib/python3.8/site-packages (from geopandas) (1.9.5)
Requirement already satisfied: pandas>=1.1.0 in /Users/kariprimiano/anaconda3/en
vs/learn-env/lib/python3.8/site-packages (from geopandas) (1.1.3)
Requirement already satisfied: shapely>=1.7.1 in /Users/kariprimiano/anaconda3/e
nvs/learn-env/lib/python3.8/site-packages (from geopandas) (2.0.2)
Requirement already satisfied: packaging in /Users/kariprimiano/anaconda3/envs/l
earn-env/lib/python3.8/site-packages (from geopandas) (20.4)
Requirement already satisfied: certifi in /Users/kariprimiano/anaconda3/envs/lea
rn-env/lib/python3.8/site-packages (from pyproj>=3.0.1->geopandas) (2023.7.22)
Requirement already satisfied: setuptools in /Users/kariprimiano/anaconda3/envs/
learn-env/lib/python3.8/site-packages (from fiona>=1.8.19->geopandas) (50.3.0.po
st20201103)
Requirement already satisfied: six in /Users/kariprimiano/anaconda3/envs/learn-e
nv/lib/python3.8/site-packages (from fiona>=1.8.19->geopandas) (1.15.0)
Requirement already satisfied: click~=8.0 in /Users/kariprimiano/anaconda3/envs/
learn-env/lib/python3.8/site-packages (from fiona>=1.8.19->geopandas) (8.1.7)
Requirement already satisfied: click-plugins>=1.0 in /Users/kariprimiano/anacond
a3/envs/learn-env/lib/python3.8/site-packages (from fiona>=1.8.19->geopandas)
(1.1.1)
Requirement already satisfied: cligj>=0.5 in /Users/kariprimiano/anaconda3/envs/
learn-env/lib/python3.8/site-packages (from fiona>=1.8.19->geopandas) (0.7.2)
Requirement already satisfied: importlib-metadata; python_version < "3.10" in /U
sers/kariprimiano/anaconda3/envs/learn-env/lib/python3.8/site-packages (from fio
na>=1.8.19->geopandas) (2.0.0)
Requirement already satisfied: attrs>=19.2.0 in /Users/kariprimiano/anaconda3/en
vs/learn-env/lib/python3.8/site-packages (from fiona>=1.8.19->geopandas) (20.2.
0)
Requirement already satisfied: python-dateutil>=2.7.3 in /Users/kariprimiano/ana
conda3/envs/learn-env/lib/python3.8/site-packages (from pandas>=1.1.0->geopanda
s) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in /Users/kariprimiano/anaconda3/env
s/learn-env/lib/python3.8/site-packages (from pandas>=1.1.0->geopandas) (2020.1)
Requirement already satisfied: numpy>=1.15.4 in /Users/kariprimiano/anaconda3/en
vs/learn-env/lib/python3.8/site-packages (from pandas>=1.1.0->geopandas) (1.18.
5)
Requirement already satisfied: pyparsing>=2.0.2 in /Users/kariprimiano/anaconda
3/envs/learn-env/lib/python3.8/site-packages (from packaging->geopandas) (2.4.7)
Requirement already satisfied: zipp>=0.5 in /Users/kariprimiano/anaconda3/envs/l
earn-env/lib/python3.8/site-packages (from importlib-metadata; python_version <
"3.10"->fiona>=1.8.19->geopandas) (3.3.0)
```

In [30]:

```python
import geopandas
import matplotlib.pyplot as plt

# Create GeoDataFrame
gdf = geopandas.GeoDataFrame(
    master_data, geometry=geopandas.points_from_xy(master_data.longitude, master

# Assigning 'status_group'
functional = gdf[gdf['status_group'] == 'functional']
repair = gdf[gdf['status_group'] == 'functional needs repair']
non_functional = gdf[gdf['status_group'] == 'non functional']

# Load world shapefile
world_shapefile_path = 'data/ne_110m_admin_0_countries/ne_110m_admin_0_countries
world = geopandas.read_file(world_shapefile_path)

# Filter for Tanzania
```

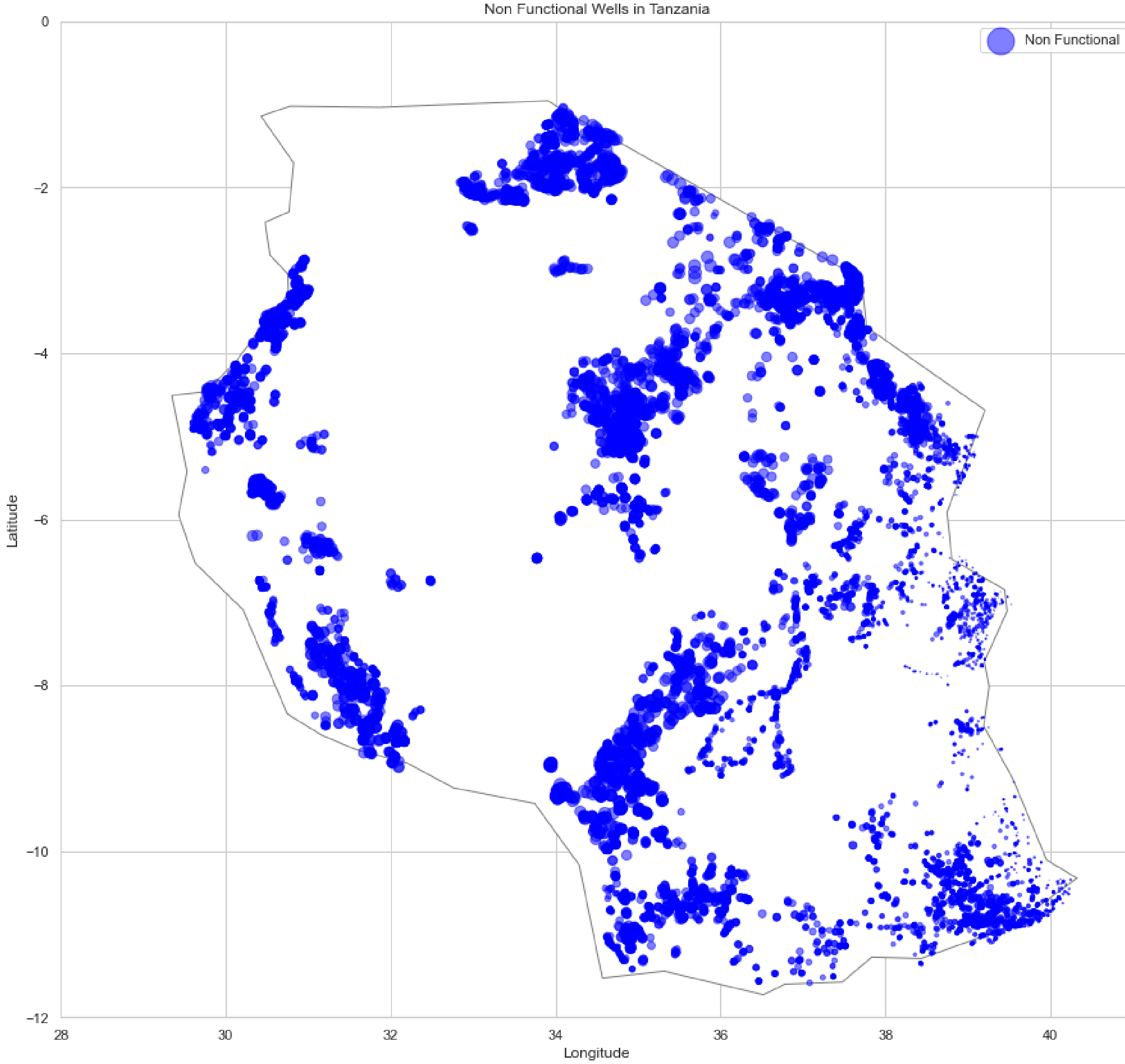
```
fig, ax = plt.subplots(figsize=(15, 15))
base = world[world.ADMIN == 'United Republic of Tanzania'].plot(color='white', e

# Scatter plots for each category
ax.scatter(functional['longitude'], functional['latitude'], c='lightblue', alpha
ax.scatter(repair['longitude'], repair['latitude'], c='orange', alpha=1, s=3, la
ax.scatter(non_functional['longitude'], non_functional['latitude'], c='darkblue'

# Limiting the display area
ax.set_ylim(-12, 0)
ax.set_xlim(28, 41)

# Adding labels and title
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
ax.set_title('Well Status in Tanzania')

# Adding legend
ax.legend(markerscale=3, loc='upper right')

plt.show()
```



To further identify geographic feature relationships, we'll plot only the non-functional wells and adjust the marker size by 'gps_height'.

In [32]:
```
# Filter for 'non functional' wells
non_functional = gdf[gdf['status_group'] == 'non functional']

# Filter for Tanzania and plot
fig, ax = plt.subplots(figsize=(15, 15))
base = world[world.ADMIN == 'United Republic of Tanzania'].plot(color='white', e

# Plot for 'non functional' wells with marker size based on 'gps_height'
# Normalize 'gps_height' for visualization
max_height = non_functional['gps_height'].max()
marker_size = (non_functional['gps_height'] / max_height) * 100

ax.scatter(non_functional['longitude'], non_functional['latitude'], c='blue', al

# Limiting the display area
ax.set_ylim(-12, 0)
ax.set_xlim(28, 41)

# Adding labels and title
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
ax.set_title('Non Functional Wells in Tanzania')

# Adding legend
ax.legend(markerscale=3, loc='upper right')

plt.show()
```

```
/Users/kariprimiano/anaconda3/envs/learn-env/lib/python3.8/site-packages/matplot
lib/collections.py:922: RuntimeWarning: invalid value encountered in sqrt
  scale = np.sqrt(self._sizes) * dpi / 72.0 * self._factor
```



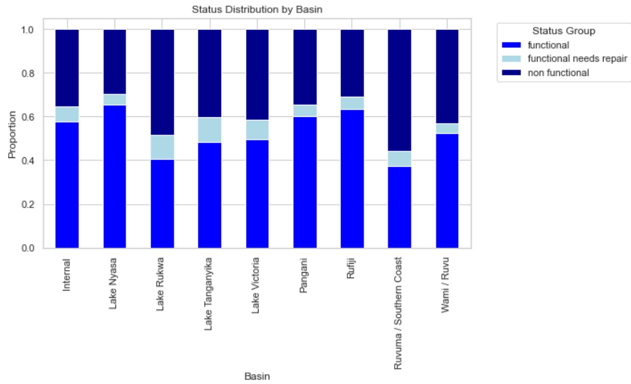
The majority of 'non functional' wells seem to be located at higher altitudes.

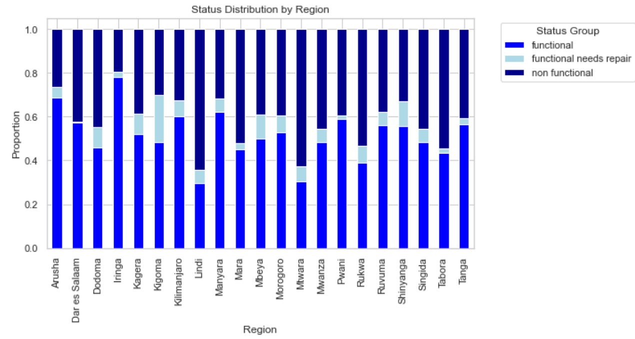
Categorical to Target Relationships

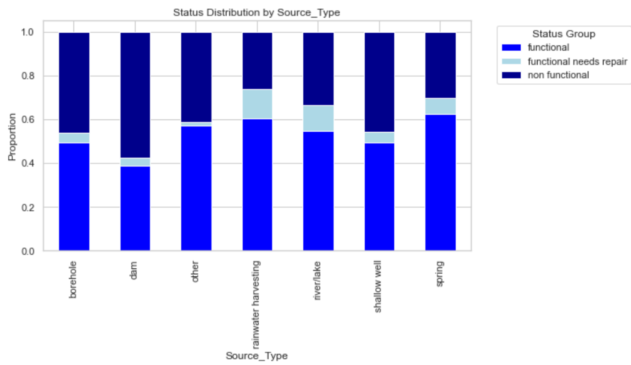
We'll explore the relationships between categorical variables and the target variable 'status_group'.

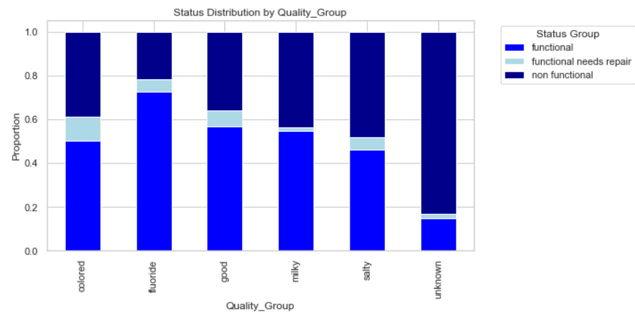
In [36]:
```
import math

categorical_vars = ['basin', 'region', 'source_type', 'quality_group', 'extracti

# Define custom colors for each status
color_map = {'functional': 'blue', 'non functional': 'darkblue', 'functional nee

# Determine the number of rows needed for the subplot (2 plots per row)
num_vars = len(categorical_vars)
num_rows = math.ceil(num_vars / 2)

# Creating subplots for each categorical variable
fig, axes = plt.subplots(nrows=num_rows, ncols=2, figsize=(20, num_rows * 6))
axes = axes.flatten()  # Flatten the axes array for easy iteration
```

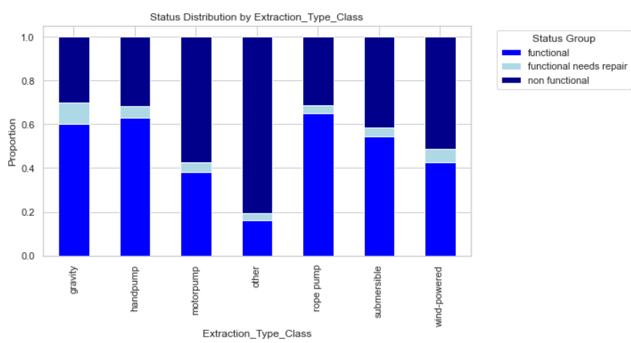
```
# Looping through the variables and creating a stacked bar plot for each
for i, var in enumerate(categorical_vars):
    # Creating a crosstab for the variable and status_group
    crosstab = pd.crosstab(master_data[var], master_data['status_group'], normal

    # Creating a stacked bar plot with custom colors
    crosstab.plot(kind='bar', stacked=True, color=[color_map[status] for status
    axes[i].set_title(f'Status Distribution by {var.title()}')
    axes[i].set_xlabel(var.title())
    axes[i].set_ylabel('Proportion')
    axes[i].legend(title='Status Group', bbox_to_anchor=(1.05, 1), loc='upper le

# Adjust the layout
plt.tight_layout()
plt.show()
```

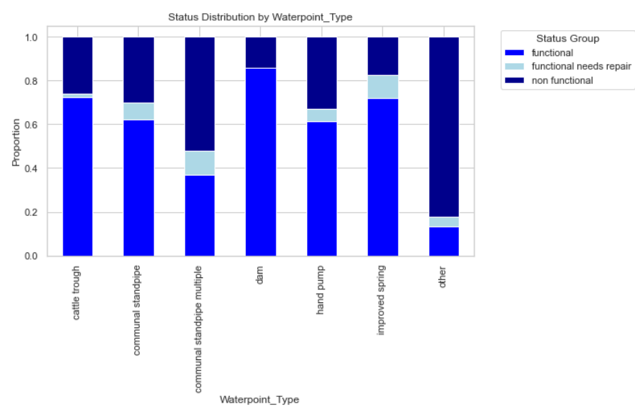


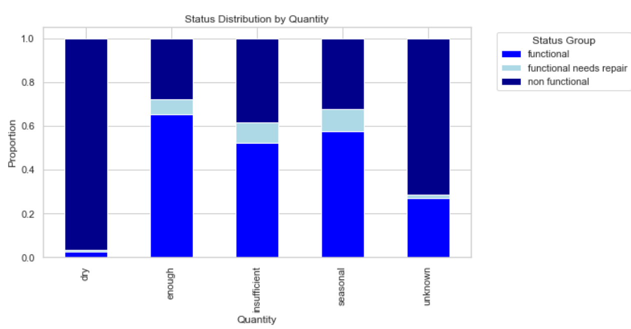


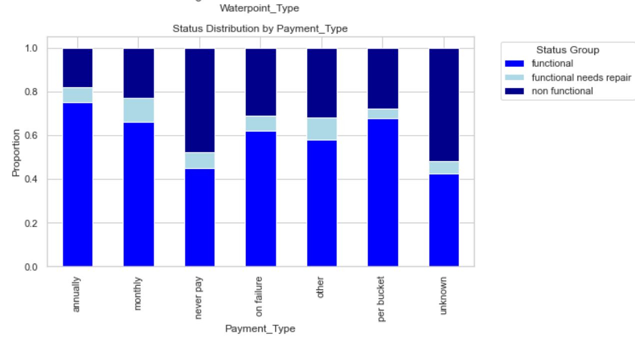












Observations

Region: Similar to basins, each region has a unique distribution of well statuses. This seems to be an indicator for well status.

Payment Type: Whether a well is paid seems to be a crucial factor. Wells that are not paid have a high number of non-functional wells.

Waterpoint Type: The method used for the population to access the water from the wells is another crucial factor. Similarly to extraction methods, waterpoint types might be more robust and less prone to failure, while others could be more complex and require frequent repairs.

Machine Learning

Baseline Model #1: Binary Target Column

Class 0 = non-functional/needs repair

Class 1 = functional

In the context of non-functional wells, focusing on recall (false negative) may be more important to ensure that most of the non-functional wells are correctly identified. We will test two separate baseline models, each with a different binary target column to see which performs best on recall.

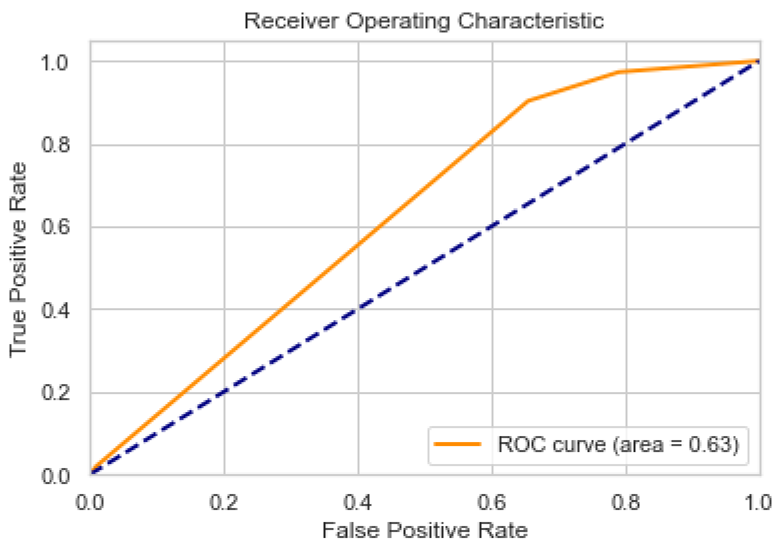
In [37]:
```
# Intitating train_test_split
X = master_data[['waterpoint_type']]
y = master_data['status_binary']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random

# Encoding categorical variable
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder()
ohe.fit(X_train)
X_train_encoded = ohe.transform(X_train)
X_test_encoded = ohe.transform(X_test)

# Plotting Log Reg transform
logreg = LogisticRegression(random_state=42)
logreg.fit(X_train_encoded, y_train)

# Checking if the target is balanced
y_test.value_counts(normalize=True)

y_pred = logreg.predict(X_test_encoded)

print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.75      0.35      0.47      8894
           1       0.62      0.90      0.73     10479

    accuracy                           0.65     19373
   macro avg       0.69      0.62      0.60     19373
weighted avg       0.68      0.65      0.61     19373

Confusion Matrix:
 [[3070 5824]
 [1015 9464]]
```

Evaluating with ROC Curve

In [38]:
```
# Predict probas for the positive class
y_pred_proba = logreg.predict_proba(X_test_encoded)[:, 1]
```

```
# Compute AUC-ROC
roc_auc = roc_auc_score(y_test, y_pred_proba)

# Compute ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)

# Plotting ROC Curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



Baseline Model #2: Reverse Binary Target Column

Class 0 = non-functional

Class 1 = functional/needs repair

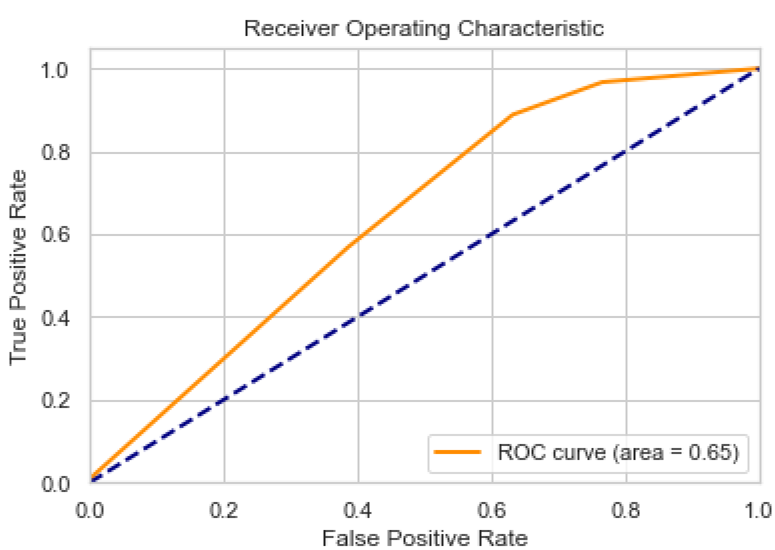
In [39]:
```
# Intitating train_test_split
X = master_data[['waterpoint_type']]
y = master_data['status_binary_reversed']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random

# Encoding categorical variable
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder()
ohe.fit(X_train)
X_train_encoded = ohe.transform(X_train)
X_test_encoded = ohe.transform(X_test)

# Plotting Log Reg transform
logreg = LogisticRegression(random_state=42)
logreg.fit(X_train_encoded, y_train)

# Checking if the target is balanced
y_test.value_counts(normalize=True)
```

```
# Predicting and evaluating the model
y_pred = logreg.predict(X_test_encoded)
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.68      0.37      0.48      7490
           1       0.69      0.89      0.78     11883

    accuracy                           0.69     19373
   macro avg       0.68      0.63      0.63     19373
weighted avg       0.68      0.69      0.66     19373

Confusion Matrix:
 [[ 2758  4732]
 [ 1327 10556]]
```

Evaluating with ROC Curve

In [40]:
```
# Predict probas for the positive class
y_pred_proba = logreg.predict_proba(X_test_encoded)[:, 1]

# Compute AUC-ROC
roc_auc = roc_auc_score(y_test, y_pred_proba)

# Compute ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)

# Plotting ROC Curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



Baseline Comparison

Baseline 1

Precision:

- Class 0: 0.75 (Higher precision for class 0)
- Class 1: 0.62

Recall:

- Class 0: 0.35 (Lower recall for class 0)
- Class 1: 0.90 (Higher recall for class 1)

F1-Score:

- Class 0: 0.47 (Lower F1-score for class 0)
- Class 1: 0.73

Accuracy: 65%

Baseline 2

Precision:

- Class 0: 0.68 (Lower precision for class 0)
- Class 1: 0.69

Recall:

- Class 0: 0.37 (Slightly higher recall for class 0)
- Class 1: 0.89 (Slightly lower recall for class 1)

F1-Score:

- Class 0: 0.48 (Slightly higher F1-score for class 0)
- Class 1: 0.78

Accuracy: 69% (Higher)

Analysis

Baseline 2 shows improved overall performance, with better accuracy and a better balance in precision and recall for both classes. However, it is more prone to falsely identifying class 0 (non-functional) instances as class 1 (functional).

Baseline 1 while having higher precision for class 0 (non-functional), falls short in accurately identifying class 0 (non-functional) instances (lower recall).

Since we are more concerned with better recall, we will continue our modeling with **Baseline 2**.

Random Forest Classifier

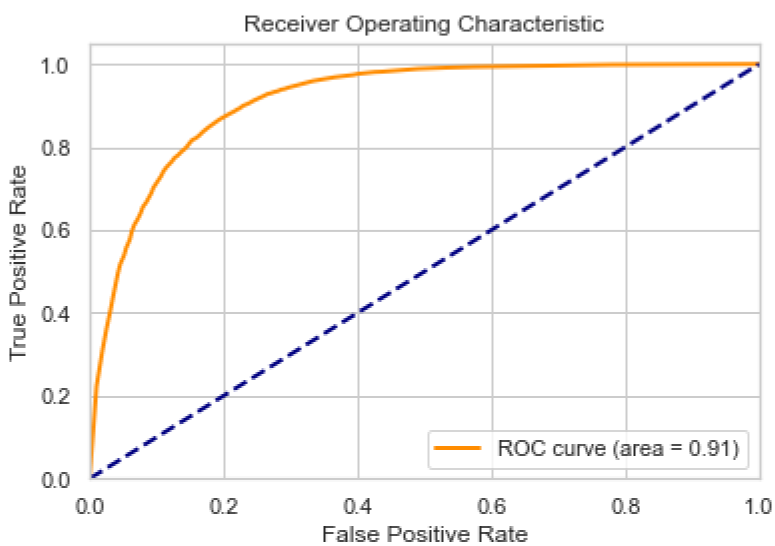
This model can provide insights into the importance of various features in predicting well functionality. It's less likely to overfit than individual decision trees and doesn't require feature scaling.

In [45]:
```
X = master_data.drop(['status_binary', 'status_binary_reversed', 'status_group',
                      'installer', 'permit', 'date_recorded', 'construction_year
y = master_data['status_binary_reversed']

# Initializing train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_

# Identifying categoricals
categorical_cols = X_train.select_dtypes(include=['object', 'category']).columns

# Creating a column transformer with OneHotEncoder for categoricals
column_transformer = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
    ],
    remainder='passthrough'
)

# Applying the column transformer
X_train_encoded = column_transformer.fit_transform(X_train)
X_test_encoded = column_transformer.transform(X_test)

# Creating and training the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_encoded, y_train)

# Extracting feature names manually for older versions of scikit-learn
onehot_features = column_transformer.named_transformers_['cat'].get_feature_name
other_features = [col for col in X_train.columns if col not in categorical_cols]
feature_names = np.concatenate([onehot_features, other_features])

# Predicting and evaluating the model
y_pred = rf_model.predict(X_test_encoded)
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.77      0.80      6856
           1       0.86      0.90      0.88     10756

    accuracy                           0.85     17612
   macro avg       0.85      0.84      0.84     17612
weighted avg       0.85      0.85      0.85     17612

Confusion Matrix:
 [[5290 1566]
 [1087 9669]]
```

Evaluating with ROC Curve

In [42]:
```
# Predict probas for the positive class
y_pred_proba = rf_model.predict_proba(X_test_encoded)[:, 1]

# Compute AUC-ROC
roc_auc = roc_auc_score(y_test, y_pred_proba)

# Compute ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)

# Plotting ROC Curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



Evaluating Feature Importance

In [43]:
```
# Extracting and displaying feature importances
importances = rf_model.feature_importances_
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importance
top_features = importance_df.sort_values(by='Importance', ascending=False).head(
print(top_features)
```

```
                          Feature  Importance
84                       latitude    0.153992
83                      longitude    0.152580
62                   quantity_dry    0.101336
82                     gps_height    0.077973
85                     population    0.054906
80           waterpoint_type_other    0.040619
81                     amount_tsh    0.030302
33     extraction_type_class_other    0.030182
63                quantity_enough    0.028917
87                 month_recorded    0.026894
64           quantity_insufficient    0.015844
51           payment_type_never pay    0.013426
```

```
75  waterpoint_type_communal standpipe    0.011889
30       extraction_type_class_gravity    0.010953
44                      management_vwc    0.010815
```

Tuning Random Forest Classifier

- SMOTE for oversampling the minority class or adjusting class weights in the model.
- Hyperparameter tuning of the Random Forest model
- RandomizedSearchCV will randomly sample 10 combos of parameters and use 3-fold cross-validation. This will reduce run time compared to GridSearchCV

In [44]:

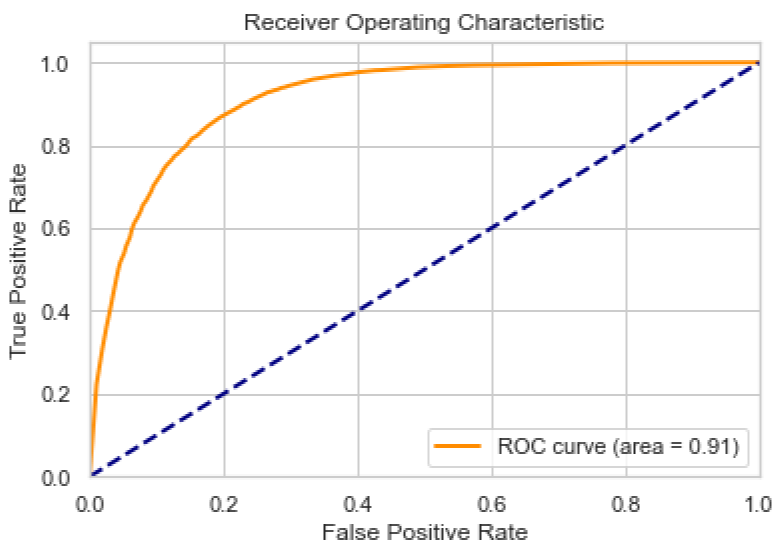
```python
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import RandomizedSearchCV

# Handling class imbalance with SMOTE
smote = SMOTE()
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_encoded, y_tra

# Define the hyperparameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']
}

# Hyperparameter tuning with Randomized Search
random_search = RandomizedSearchCV(
    RandomForestClassifier(random_state=42),
    param_grid,
    n_iter=10,
    cv=3,
    scoring='recall',
    n_jobs=-1
)

random_search.fit(X_train_resampled, y_train_resampled)

# Get the best model
best_model = random_search.best_estimator_

# Re-train and evaluate the model with the best params
best_model.fit(X_train_resampled, y_train_resampled)
y_pred = best_model.predict(X_test_encoded)

# Evaluate the model
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.84      0.62      0.71      6856
           1       0.79      0.93      0.85     10756

    accuracy                           0.81     17612
```

```
   macro avg       0.82      0.77      0.78     17612
weighted avg       0.81      0.81      0.80     17612

Confusion Matrix:
 [[4227 2629]
 [ 787 9969]]
```

Evaluating with ROC Curve

In [46]:
```
# Predict probas for the positive class
y_pred_proba = rf_model.predict_proba(X_test_encoded)[:, 1]

# Compute AUC-ROC
roc_auc = roc_auc_score(y_test, y_pred_proba)

# Compute ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)

# Plotting ROC Curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



Comparison Interpretations of Random Forest Models

Class 0 = non-functional/needs repair

Class 1 = functional

Precision:

- **Original Model:** Precision is 0.83 for both classes.
- **Tuned Model:** Precision increased to 0.85 for class 0 but decreased to 0.79 for class 1.

Recall:

- **Original Model:** Recall is 0.77 for class 0 and 0.90 for class 1.
- **Tuned Model:** Recall decreased to 0.62 for class 0 but increased to 0.93 for class 1.

F1-Score:

- **Original Model:** F1-scores are 0.80 (class 0) and 0.88 (class 1).
- **Tuned Model:** F1-scores are 0.72 (class 0) and 0.86 (class 1).

Accuracy:

- **Original Model:** Overall accuracy is 0.85.
- **Tuned Model:** Overall accuracy decreased to 0.81.

Macro and Weighted Averages:

- **Original Model:** Both macro and weighted averages are around 0.85.
- **Tuned Model:** Both macro and weighted averages are around 0.77 - 0.81.

AUC-ROC Score:

- **Original Model:** The area under the curve is 0.91, which is high. This means the model can effectively distinguish between the positive class (class 1) and the negative class (class 0).
- **Tuned Model:** The area under the curve is also 0.91.

Analysis of Comparison:

- Tuning the model appears to have made it more biased towards class 1, improving its ability to detect class 1 instances but worsening its performance for class 0 (higher false positives).
- The original model is more balanced in terms of precision and recall across both classes.
- The tuned model has a lower overall accuracy compared to the original model.

Logistic Regression Model: Most Important Features

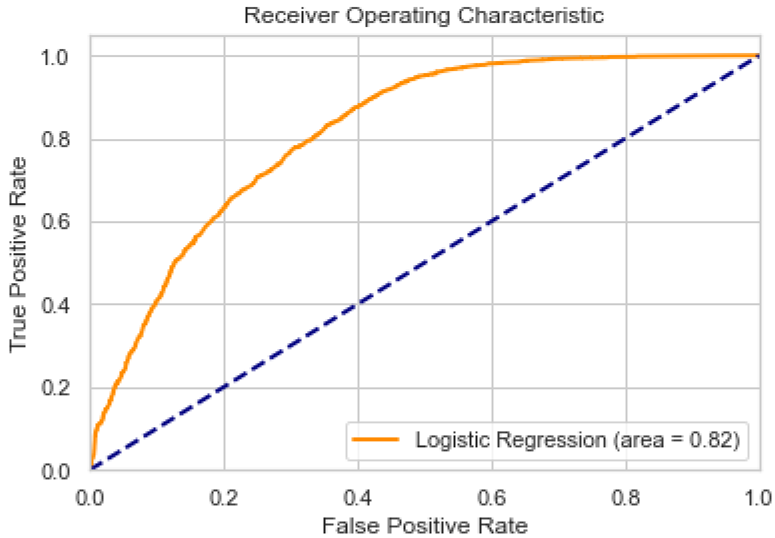
In [47]:
```
# Selecting most important featues
features = ['region', 'quantity', 'gps_height', 'waterpoint_type', 'extraction_t
X = master_data[features]
y = master_data['status_binary_reversed']

# Initializing train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_

# Identifying categorical and numerical
```

```
categorical_cols = ['quantity', 'region', 'waterpoint_type', 'extraction_type_cl
numerical_cols = ['gps_height']

# Creating a column transformer with OneHotEncoder and StandardScaler
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(), categorical_cols)
    ]
)

# Creating a pipeline with preprocessing and log reg model
model_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(solver='saga', max_iter=1000, random_state
])

# Training the model
model_pipeline.fit(X_train, y_train)

# Predicting and evaluating the model
y_pred = model_pipeline.predict(X_test)
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.54      0.65      6856
           1       0.76      0.93      0.84     10756

    accuracy                           0.78     17612
   macro avg       0.79      0.73      0.74     17612
weighted avg       0.79      0.78      0.76     17612

Confusion Matrix:
 [[3694 3162]
 [ 768 9988]]
```

In [48]:

```
# Calculate probas, ROC curve, and AUC for log reg
logreg_probs = model_pipeline.predict_proba(X_test)[:, 1]
fpr_logreg, tpr_logreg, _ = roc_curve(y_test, logreg_probs)
roc_auc_logreg = auc(fpr_logreg, tpr_logreg)


# Plotting ROC Curve
plt.figure()
plt.plot(fpr_logreg, tpr_logreg, color='darkorange', lw=2, label='Logistic Regre
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



Interpretation

Precision:

- For class 0: 83% precision means that out of all instances predicted as class 0, 83% were actually class 0.
- For class 1: 76% precision indicates that out of all instances predicted as class 1, 76% were actually class 1.

Recall:

- For class 0: The recall of 54% is moderate, meaning the model correctly identifies 54% of the actual class 0 instances.
- For class 1: A high recall of 93% indicates the model is very effective at identifying class 1 instances.

F1-Score:

- For class 0: The F1-score of 0.65 suggests a balance between precision and recall for class 0, but more weighted towards precision.
- For class 1: The F1-score of 0.84 indicates a strong balance between precision and recall for class 1, favoring recall.

Accuracy:

- The overall accuracy of 78% indicates that the model correctly predicts the class for 78% of all instances.

Macro and Weighted Averages:

- Macro average treats both classes equally, showing an average precision of 79%, recall of 73%, and F1-score of 74%.

- Weighted average considers class imbalance, showing slightly higher precision and recall, indicating better performance on the more prevalent class 1.

ROC Score:

- An ROC score of 0.82 suggests a good ability of the model to distinguish between the two classes. It indicates a favorable balance between the true positive rate and false positive rate across different thresholds.

Insights:

- The model performs well overall, especially in predicting class 1, which is indicated by the high recall and F1-score for class 1.
- Model is less effective in correctly identifying class 0 instances, as evidenced by the lower recall for class 0.
- The relatively high number of false positives for class 0 (3162) indicates that the model often misclassifies class 1 instances as class 0.

Key Findings

- **Geographic Indicators:** Including region and altitude, geographic features are 21% MORE influential in identifying non-functional wells than other features.
- **Region:** Mbeya, Morogoro, and Kilimanjaro have the highest rates of non-functional wells. Altitude may play an important role in water source access.
- **Type of Wells:** Communal Standpipe wells are most likely to be a functional well. Other well types have the highest percentage of non-functional wells at 81.38%. Multi Communal Handpipe wells have the second highest percentage of non-functional wells at 53.85%.
- **Payment Type:** Whether a well is paid seems to be a crucial factor. Wells that are not paid have a high number of non-functional wells.
- **Random Forest Classifier:** Our best performing model gave us actionable insights into feature importance and effectively minimized false-negatives.

Conclusion

In this project, we have unearthed critical insights to steer the strategic decision-making of the Tanzanian Government. This includes pinpointing the precise types of wells that warrant prioritized construction efforts, as well as identifying the specific regions that should receive initial focus and substantial investment in well infrastructure.

Next Steps

- **Investigate Additional Features:** Concentrating on geographical indicators like climate, population, and amount of water available in the area.
- **Time-Series Analysis:** Further consideration of the well ages should be analyzed to predict the average lifetime of more robust well structures.
- **Repairs:** Local governments should look at what type of water wells are needing repairs, and the severity of those repairs, to fine-tune non-functional indicators.

Sources

- Driven Data - Tanzanian Water Wells
  - Labels
  - Values
- The World Bank
- Groundwater Wells
- Detection of Non-Function Bore Wells Using Maching Learning Algorithms

About Us

Goknur Kaya

- Github Lead
- Email: goknurkaya@gmail.com
- github.com/goknurk

Kari Primiano

- Tech Lead
- Email: kkprim@gmail.com
- github.com/kkprim

Mytreyi Abburu

- Presentation Lead
- Email: abburumk@gmail.com
- github.com/myt-hue