Phase-3-Project

TANZANIAN WATER WELLS

OVERVIEW

This project is based on data on the on the condition of wells within Tanzania. The data is used to create a predictive model and provide insights into whether a well is functional or not. This can aid the Tanzanian Government in deciding which wells neeed to be visited first in order to repair and aid in the crisis the country is facing.

BUSINESS UNDERSTANDING

Tanzania is a country in East Africa known for its national parks and wild animals. The World Bank estimates its population at 65 million as of 2022 and its land size is about 947,303 km2. Like many other sub-Saharan African countries, Tanzania is a developing country struggling to provide adequate clean water for its growing population. The main problem being faced is that there are a number of water wells within Tanzania and many are in need of repairs. This is an issue beacuse wells are the main water source therefore, lacking functional wells is a severe threat to livelihood.

DATA

Column Names and Descriptions for Tanzanian Water Wells Data Set

The training dataset contains 59,400 waterpoints in Tanzania and the following 39 features:

- **amount_tsh** - Total static head (amount water available to waterpoint)
- **date_recorded** - Date on which the row was recorded
- **price** - Individual or organization that funded installation of the well
- **gps_height** - The altitude at which the water pump is located
- **installer** - Individual or organization that installed the well
- **longitude** - Longitude coordinates of the water point
- **latitude** - Latitude coordinates of the water point
- **wpt_name** - Name of the waterpoint if there is one
- **num_private** - Information about this feature is unavailable
- **basin** - Name of the geographic water basin
- **subvillage** - Geographic location

- **region** - Geographic location
- **region_code** - Coded geographic location
- **district_code** - Coded geographic location
- **lga** - Geographic location
- **ward** - Geographic location
- **population** - Population around the well
- **public_meeting** - Boolean data whether public meeting for the water pump was conducted
- **recorded_by** - Name of agency which recorded the water pump data
- **scheme_management** - Name of scheme that manages the waterpoint
- **scheme_name** - Name of scheme under which the water point was established
- **permit** - Boolean data describing whether the water point has permit available or not
- **contruction_year** - Year in which the water point was constructed
- **extraction_type** - The kind of extraction the waterpoint uses
- **extraction_type_group** - The kind of extraction the waterpoint uses
- **extraction_type_class** - The kind of extraction the waterpoint uses
- **management** - Name of organization/authority that manages the water point
- **management_group** - Category of organization/authority that manages the water point
- **payment** - Describes how residents pay for water
- **payment_type** - Describes how residents pay for water
- **water_quality** - The quality of water
- **quality_group** - The quality of water
- **quantity** - The quantity of water
- **quantity_group** - The quantity of water
- **source** - The source of water
- **source_type** - The source of water
- **source_class** - The source of water
- **waterpoint_type** - The nature of water point
- **waterpoint_type_group** - The nature of water point

METHODOLOGY

Due to the many categorical features that can influence functionality, this project investigates features and their effects through two types of models in an attempt to best predict the well condition. This is done through the use of logistic regression and decision trees.

IMPORT LIBRARIES & DATA

```
# importing the neccessary libraries
# Import relevant Python modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split, GridSearchCV, cross_validate
from sklearn.model_selection import cross_val_predict, cross_val_score, RepeatedStratifiedKFold
from sklearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE

# Classification Models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.dummy import DummyClassifier
from sklearn.preprocessing import StandardScaler, OneHotEncoder, FunctionTransformer
from sklearn.impute import SimpleImputer
```

```
from sklearn.compose import ColumnTransformer

from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, precision_score, recall_score
from sklearn.metrics import ConfusionMatrixDisplay, classification_report
from sklearn.metrics import roc_curve, auc, roc_auc_score

# Scalers
from sklearn.impute import SimpleImputer

from sklearn.preprocessing import StandardScaler, LabelBinarizer, label_binarize

from sklearn.preprocessing import OneHotEncoder
```

```
# importing the csv files
labels = pd.read_csv('data\Training_set_labels.csv')

df = pd.read_csv('data/Training_set_values.csv')
```

```
#checking the shape of the dataframes
print('df: ', df.shape)
print('labels: ',labels.shape)
```

```
df:  (59400, 40)
labels:  (59400, 2)
```

Based on the two shapes, it is likely that the two datasets have the same data. Therefore, they are suitable to merge.

```
df.head()
```

DATA CLEANING

```
df_1['scheme_management'].value_counts()
```

```
scheme_management
VWC                 36793
WUG                  5206
Water authority      3153
WUA                  2883
Water Board          2748
Parastatal           1680
Private operator     1063
Company              1061
Other                 766
SWC                    97
Trust                  72
Name: count, dtype: int64
```

```
df_1.head(10)
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wpt_name	num_
0	69572	6000.0	2011-03-14	Roman	1390	Roman	34.938093	-9.856322	none	0
1	8776	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.698766	-2.147466	Zahanati	0
2	34310	25.0	2013-02-25	Lottery Club	686	World vision	37.460664	-3.821329	Kwa Mahundi	0
3	67743	0.0	2013-01-28	Unicef	263	UNICEF	38.486161	-11.155298	Zahanati Ya Nanyumbu	0
4	19728	0.0	2011-07-13	Action In A	0	Artisan	31.130847	-1.825359	Shuleni	0
5	9944	20.0	2011-03-13	Mkinga Distric Coun	0	DWE	39.172796	-4.765587	Tajiri	0
6	19816	0.0	2012-10-01	Dwsp	0	DWSP	33.362410	-3.766365	Kwa Ngomho	0
7	54551	0.0	2012-10-09	Rwssp	0	DWE	32.620617	-4.226198	Tushirikiane	0
8	53934	0.0	2012-11-03	Wateraid	0	Water Aid	32.711100	-5.146712	Kwa Ramadhan Musa	0
9	46144	0.0	2011-08-03	Isingiro Ho	0	Artisan	30.626991	-1.257051	Kwapeto	0

10 rows × 43 columns

```
# checking the datatypes of the columns
df_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59400 entries, 0 to 59399
Data columns (total 43 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     59400 non-null  int64  
 1   amount_tsh             59400 non-null  float64
 2   date_recorded          59400 non-null  object 
 3   funder                 55763 non-null  object 
 4   gps_height             59400 non-null  int64  
 5   installer              55745 non-null  object 
 6   longitude              59400 non-null  float64
 7   latitude               59400 non-null  float64
 8   wpt_name               59398 non-null  object 
 9   num_private            59400 non-null  int64  
 10  basin                  59400 non-null  object 
 11  subvillage             59029 non-null  object 
 12  region                 59400 non-null  object 
 13  region_code            59400 non-null  int64  
 14  district_code          59400 non-null  int64  
 15  lga                    59400 non-null  object 
 16  ward                   59400 non-null  object 
 17  population             59400 non-null  int64  
 18  public_meeting         56066 non-null  object 
 19  recorded_by            59400 non-null  object 
 20  scheme_management      55522 non-null  object 
 21  scheme_name            30590 non-null  object 
```

```
 22  permit                 56344 non-null  object
 23  construction_year      59400 non-null  int64
 24  extraction_type        59400 non-null  object
 25  extraction_type_group  59400 non-null  object
 26  extraction_type_class  59400 non-null  object
 27  management             59400 non-null  object
 28  management_group       59400 non-null  object
 29  payment                59400 non-null  object
 30  payment_type           59400 non-null  object
 31  water_quality          59400 non-null  object
 32  quality_group          59400 non-null  object
 33  quantity               59400 non-null  object
 34  quantity_group         59400 non-null  object
 35  source                 59400 non-null  object
 36  source_type            59400 non-null  object
 37  source_class           59400 non-null  object
 38  waterpoint_type        59400 non-null  object
 39  waterpoint_type_group  59400 non-null  object
 40  status_group           59400 non-null  object
 41  status                 59400 non-null  object
 42  binary_status          59400 non-null  int32
dtypes: float64(3), int32(1), int64(7), object(32)
memory usage: 19.3+ MB
```

Noting that there are many integer columns that should be string, change the datatypes of these columns

```
#changing the datatypes of some columns
df_1['region_code']= df_1['region_code'].astype(str)
df_1['district_code'] = df_1['district_code'].astype(str)
df_1['construction_year'] = df_1['construction_year'].astype(str)
df_1['amount_tsh'] = df_1['amount_tsh'].astype(int)
df_1['permit'] = np.where(df_1['permit'] == True, 1, df_1['permit'])
df_1['permit'] = np.where(df_1['permit'] == False, 0, df_1['permit'])
df_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59400 entries, 0 to 59399
Data columns (total 43 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   id                     59400 non-null  int64
 1   amount_tsh             59400 non-null  int32
 2   date_recorded          59400 non-null  object
 3   funder                 55763 non-null  object
 4   gps_height             59400 non-null  int64
 5   installer              55745 non-null  object
 6   longitude              59400 non-null  float64
 7   latitude               59400 non-null  float64
 8   wpt_name               59398 non-null  object
 9   num_private            59400 non-null  int64
 10  basin                  59400 non-null  object
 11  subvillage             59029 non-null  object
 12  region                 59400 non-null  object
 13  region_code            59400 non-null  object
 14  district_code          59400 non-null  object
 15  lga                    59400 non-null  object
 16  ward                   59400 non-null  object
 17  population             59400 non-null  int64
 18  public_meeting         56066 non-null  object
 19  recorded_by            59400 non-null  object
 20  scheme_management      55522 non-null  object
 21  scheme_name            30590 non-null  object
 22  permit                 56344 non-null  object
 23  construction_year      59400 non-null  object
 24  extraction_type        59400 non-null  object
 25  extraction_type_group  59400 non-null  object
 26  extraction_type_class  59400 non-null  object
 27  management             59400 non-null  object
 28  management_group       59400 non-null  object
 29  payment                59400 non-null  object
```

```
 30  payment_type           59400 non-null  object
 31  water_quality          59400 non-null  object
 32  quality_group          59400 non-null  object
 33  quantity               59400 non-null  object
 34  quantity_group         59400 non-null  object
 35  source                 59400 non-null  object
 36  source_type            59400 non-null  object
 37  source_class           59400 non-null  object
 38  waterpoint_type        59400 non-null  object
 39  waterpoint_type_group  59400 non-null  object
 40  status_group           59400 non-null  object
 41  status                 59400 non-null  object
 42  binary_status          59400 non-null  int32
dtypes: float64(2), int32(2), int64(4), object(35)
memory usage: 19.0+ MB
```

```
df_1.loc[((df_1['permit'] != 0) &
             (df_1['permit'] != 1))] #locating Null
df_1.drop(df_1[(df_1['permit'] != 0) &
                        (df_1['permit'] != 1)].index, inplace=True)
df_1['permit'] = df_1['permit'].astype(int)
```

```
df_1['source_type'].value_counts()
```

```
source_type
shallow well            16253
spring                  15981
borehole                11162
river/lake              10013
rainwater harvesting     2039
dam                       630
other                     266
Name: count, dtype: int64
```

```
df_1['extraction_type_class'].value_counts()
```

```
extraction_type_class
gravity         25234
handpump        16048
other            6050
submersible      5854
motorpump        2704
rope pump         349
wind-powered      105
Name: count, dtype: int64
```

```
df_1['funder'].value_counts()
```

```
funder
Government Of Tanzania    9043
Danida                    3112
Hesawa                    2027
Rwssp                     1372
World Bank                1345
                          ...
Comune Di Roma               1
Swifti                       1
Area                         1
Rwi                          1
Samlo                        1
Name: count, Length: 1834, dtype: int64
```

```
df_1['installer'].value_counts()
```

```
installer
DWE              17361
Government        1788
RWE               1203
Commu             1060
DANIDA            1049
                 ...
B.A.P                1
R                    1
Nasan workers        1
TWESS                1
SELEPTA              1
Name: count, Length: 2056, dtype: int64
```

```
df_1.loc[df_1['installer'] == '-']
# there's construction year 0
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wpt_name	n
10217	42616	0	2011-08-03	Kalebejo Parish	0	-	32.356645	-2.499427	Kalebejo Parish	0
20968	10873	0	2011-07-26	Government Of Tanzania	0	-	32.677150	-2.508912	Kwa Madebele	0
25769	21336	0	2011-07-26	Government Of Tanzania	0	-	32.674665	-2.506721	Health Center	0

3 rows × 43 columns

```
df_1['construction_year'].value_counts()
#dropping this column
```

```
construction_year
0       19580
2008     2576
2009     2491
2010     2430
2000     1566
2007     1559
2006     1447
2003     1276
2011     1211
2004     1109
2002     1065
1978     1027
2012     1026
2005      985
1995      979
1999      954
```

```
1985      943
1998      924
1984      779
1996      768
1982      741
1972      705
1994      703
1974      676
1990      667
1980      647
1992      632
1997      613
1993      595
2001      533
1988      521
1983      487
1975      437
1986      433
1976      411
1991      323
1989      316
1970      310
1987      301
1981      238
1977      199
1979      192
1973      183
2013      173
1971      145
1967       86
1963       85
1968       68
1969       59
1960       45
1964       40
1962       29
1961       20
1965       19
1966       17
Name: count, dtype: int64
```

```
df_1['management'].value_counts()
```

```
management
vwc                 38296
wug                  6340
water board          2830
wua                  2468
private operator     1893
parastatal           1595
water authority       825
other                 744
company               658
unknown               519
other - school         99
trust                  77
Name: count, dtype: int64
```

```
df_1['quantity_group'].value_counts()
```

```
quantity_group
enough          31979
insufficient    13934
dry              5836
seasonal         3901
```

```
unknown           694
Name: count, dtype: int64
```

```
df_1['quality_group'].value_counts()
```

```
quality_group
good        48416
salty        5035
unknown      1399
milky         801
colored       490
fluoride      203
Name: count, dtype: int64
```

```
#dropping some specific wells
df_1.drop(df_1[(df_1['quantity_group'] == 'unknown') |
             (df_1['quality_group'] == 'unknown')].index, inplace=True)
df_1.shape
```

```
(54744, 43)
```

```
df_1['quantity_group'].value_counts()
```

```
quantity_group
enough          31851
insufficient    13830
dry              5202
seasonal         3861
Name: count, dtype: int64
```

```
df_1['quality_group'].value_counts()
```

```
quality_group
good        48246
salty        5007
milky         799
colored       489
fluoride      203
Name: count, dtype: int64
```

```
df_1['payment'].value_counts()
```

```
payment
never pay                23132
pay per bucket            8591
pay monthly               8167
unknown                   6628
pay when scheme fails     3728
pay annually              3533
other                      965
Name: count, dtype: int64
```

```
df_1.drop(['payment'], axis=1, inplace = True)
#Dropping the payment column because it was initially thought to represent the cost of water,
# but it actually indicates the payment method, which is not as useful for the goals of this project
```

```
df_1['amount_tsh'].value_counts()
```

```
amount_tsh
0         37625
500        3046
50         2310
1000       1431
20         1393
          ...
53            1
138000        1
306           1
6300          1
59            1
Name: count, Length: 92, dtype: int64
```

```
columns_to_drop = ['date_recorded', 'funder', 'wpt_name', 'subvillage', 'lga',
 'ward', 'recorded_by', 'scheme_name', 'extraction_type',
 'extraction_type_group', 'management','quality_group',
 'quantity', 'source', 'source_type', 'waterpoint_type', 'num_private',
 'region_code', 'district_code']

# Drop the columns from dataset
df_1 = df_1.drop(columns_to_drop, axis=1)
```

The columns above have been dropped because there are some column pairs that have duplicate information, such as a region and region code, which makes them collinear. Some columns have too many unique values or the same single value, making them irrelevant for modelling.

```
# Check columns with missing values
df_1.isna().sum()
```

```
id                          0
amount_tsh                  0
gps_height                  0
installer                1117
longitude                   0
latitude                    0
basin                       0
region                      0
population                  0
public_meeting           2730
scheme_management        3487
permit                      0
construction_year           0
extraction_type_class       0
management_group            0
payment_type                0
water_quality               0
quantity_group              0
source_class                0
waterpoint_type_group       0
status_group                0
status                      0
binary_status               0
dtype: int64
```

```
# Create a list of missing-value columns
missing_value_columns = ['installer', 'public_meeting', 'scheme_management', 'permit']

# Check the value counts
for col in missing_value_columns:
    print(df_1[col].value_counts())
```

```
installer
DWE                     16899
Government               1674
RWE                      1151
DANIDA                   1041
Commu                    1025
                        ...
harison                     1
MSIGWA                      1
Singida yetu                1
MINISTRY OF EDUCATION       1
SELEPTA                     1
Name: count, Length: 2011, dtype: int64
public_meeting
True     47455
False     4559
Name: count, dtype: int64
scheme_management
VWC                 33712
WUG                  4943
Water authority      2898
WUA                  2747
Water Board          2640
Parastatal           1495
Company              1030
Private operator     1016
Other                 608
SWC                    97
Trust                  71
Name: count, dtype: int64
permit
1    37948
0    16796
Name: count, dtype: int64
```

```
df_1.info() #checking cleaned dataset
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 54744 entries, 0 to 59399
Data columns (total 23 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     54744 non-null  int64  
 1   amount_tsh             54744 non-null  int32  
 2   gps_height             54744 non-null  int64  
 3   installer              53627 non-null  object 
 4   longitude              54744 non-null  float64
 5   latitude               54744 non-null  float64
 6   basin                  54744 non-null  object 
 7   region                 54744 non-null  object 
 8   population             54744 non-null  int64  
 9   public_meeting         52014 non-null  object 
 10  scheme_management      51257 non-null  object 
 11  permit                 54744 non-null  int32  
 12  construction_year      54744 non-null  object 
 13  extraction_type_class  54744 non-null  object 
 14  management_group       54744 non-null  object 
 15  payment_type           54744 non-null  object 
 16  water_quality          54744 non-null  object 
 17  quantity_group         54744 non-null  object 
 18  source_class           54744 non-null  object 
 19  waterpoint_type_group  54744 non-null  object 
 20  status_group           54744 non-null  object 
 21  status                 54744 non-null  object 
 22  binary_status          54744 non-null  int32  
dtypes: float64(2), int32(3), int64(3), object(15)
memory usage: 9.4+ MB
```

```
# Drop rows with missing values in 'installer' and 'scheme_management' columns
df_1.dropna(subset=['installer', 'scheme_management'], axis=0, inplace=True)
```

```
# Fill missing values in funder and installer and scheme_management columns with 'Other'
for col in ['public_meeting', 'permit']:
    df_1[col] = df[col].fillna(True)
```

```
# Confirm there are no more missing values
df_1.isna().sum()
```

```
id                       0
amount_tsh               0
gps_height               0
installer                0
longitude                0
latitude                 0
basin                    0
region                   0
population               0
public_meeting           0
scheme_management        0
permit                   0
construction_year        0
extraction_type_class    0
management_group         0
payment_type             0
water_quality            0
quantity_group           0
source_class             0
waterpoint_type_group    0
status_group             0
status                   0
binary_status            0
dtype: int64
```

```
df_1.shape
```

```
(50166, 23)
```

```
df_1.describe() #looking for outliers
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	id	amount_tsh	gps_height	longitude	latitude	population	binary_status
count	50166.000000	50166.000000	50166.000000	50166.000000	5.016600e+04	50166.000000	50166.000000
mean	37140.276343	348.388670	697.901248	34.160264	-5.680130e+00	176.615576	0.627915
std	21443.386912	2793.108425	697.281175	6.473404	2.909040e+00	467.986126	0.483366
min	0.000000	0.000000	-90.000000	0.000000	-1.164944e+01	0.000000	0.000000

	id	amount_tsh	gps_height	longitude	latitude	population	binary_status
25%	18569.500000	0.000000	0.000000	33.067725	-8.226623e+00	0.000000	0.000000
50%	37095.500000	0.000000	463.000000	35.096511	-4.982342e+00	35.000000	1.000000
75%	55674.750000	40.000000	1332.000000	37.300843	-3.324285e+00	200.000000	1.000000
max	74247.000000	250000.000000	2770.000000	40.323402	-2.000000e-08	30500.000000	1.000000

```
df_1['population'].hist()
```

```
<Axes: >
```

[png]

```
df_1['amount_tsh'].hist()
```

```
<Axes: >
```

[png]

MODELLING

DATA PROCESSING

```
# Assign status_group column to y series
y = df_1['status_group']

# Drop status_group to create X dataframe
X = df_1.drop('status_group', axis=1)

# Print first 5 rows of X
X.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	id	amount_tsh	gps_height	installer	longitude	latitude	basin	region	population	public_meet
0	69572	6000	1390	Roman	34.938093	-9.856322	Lake Nyasa	Iringa	109	True
1	8776	0	1399	GRUMETI	34.698766	-2.147466	Lake Victoria	Mara	280	True
2	34310	25	686	World vision	37.460664	-3.821329	Pangani	Manyara	250	True
3	67743	0	263	UNICEF	38.486161	-11.155298	Ruvuma / Southern Coast	Mtwara	58	True

	id	amount_tsh	gps_height	installer	longitude	latitude	basin	region	population	public_meet
5	9944	20	0	DWE	39.172796	-4.765587	Pangani	Tanga	1	True

5 rows × 22 columns

```
X.dtypes #looking into data types of X
```

```
id                          int64
amount_tsh                  int32
gps_height                  int64
installer                  object
longitude                 float64
latitude                  float64
basin                      object
region                     object
population                  int64
public_meeting               bool
scheme_management          object
permit                       bool
construction_year          object
extraction_type_class      object
management_group           object
payment_type               object
water_quality              object
quantity_group             object
source_class               object
waterpoint_type_group      object
status                     object
binary_status               int32
dtype: object
```

```
#One-hot encoding of categorical features
# Create lists of categorical, continuous, and binary columns
cat_col = ['installer', 'basin', 'region', 'scheme_management',
           'extraction_type_class', 'management_group', 'payment_type',
           'water_quality', 'quantity_group', 'source_class','status',
           'waterpoint_type_group']

cont_col = ['amount_tsh','gps_height','longitude','latitude','population','construction_year']

binary_col = ['public_meeting', 'permit']
```

```
# Transformer for numeric and categorical features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), cont_col),
        ('cat', OneHotEncoder(), cat_col)
    ])

# Create a pipeline with preprocessing and logistic regression
pipe_logistic = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression())
])
```

```
#Create dummies
X = pd.get_dummies(X, columns=cat_col)

# Print X
X
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	id	amount_tsh	gps_height	longitude	latitude	population	public_meeting	permit	construction_ye
0	69572	6000	1390	34.938093	-9.856322	109	True	False	1999
1	8776	0	1399	34.698766	-2.147466	280	True	True	2010
2	34310	25	686	37.460664	-3.821329	250	True	True	2009
3	67743	0	263	38.486161	-11.155298	58	True	True	1986
5	9944	20	0	39.172796	-4.765587	1	True	True	2009
...	...	...	...	...	...	...	...	...	...
59394	11164	500	351	37.634053	-6.124830	89	True	True	2007
59395	60739	10	1210	37.169807	-3.253847	125	True	True	1999
59396	27263	4700	1212	35.249991	-9.070629	56	True	True	1996
59398	31282	0	0	35.861315	-6.378573	0	True	True	0
59399	26348	0	191	38.104048	-6.747464	150	True	True	2002

50166 rows × 1980 columns

Train - Test split

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size= 0.2, random_state=42)
```

Baseline Machine Learning Model

```
# Initiate metrics list
metrics_list=[]

class Metrics:
    def model_score(self, model, y_true, y_pred):
        # Print classification report, accuracy, precision, recall, f1_score
        print(classification_report(y_true, y_pred))
        print("Overall accuracy score", accuracy_score(y_true, y_pred))
        print("Overall precision score", precision_score(y_true, y_pred, average='weighted'))
        print("Overall recall score", recall_score(y_true, y_pred, average='weighted'))
        print("Overall F1-score", f1_score(y_true, y_pred, average='weighted'))

        # Print a confusion matrix
        cnf_matrix = confusion_matrix(y_true, y_pred)
        disp = ConfusionMatrixDisplay(confusion_matrix=cnf_matrix, display_labels=model.classes_)
        print('\nConfusion Matrix')
        return disp.plot()

    # Create a list of model metrics
    def get_metrics(self, model_name, model, y_true, y_pred):  #y_test, X_test, model
        metrics = {}
        metrics['model_name'] = model_name
        metrics['accuracy'] = accuracy_score(y_true, y_pred)
        metrics['f1 score'] = f1_score(y_true, y_pred, average='weighted')
        metrics['precision'] = precision_score(y_true, y_pred, average='weighted')
```

```
        metrics['recall'] = recall_score(y_true, y_pred, average='weighted')
        metrics_list.append(metrics)
        return metrics_list
```

```
# Baseline model pipeline
# pipeline for baseline logistic regression
pipe_logistic = Pipeline([('ss', StandardScaler()),
                    ('lr', LogisticRegression(random_state=42))])

# pipeline for baseline decision tree classification
pipe_decision_tree = Pipeline([('ss', StandardScaler()),
                    ('tree', DecisionTreeClassifier(random_state=42))])
```

LOGISTIC REGRESSION MODEL

```
# Fit the logistic regression pipeline to the training data
log_model = pipe_logistic.fit(X_train, y_train)
# Print the accuracy on test set
pipe_logistic.score(X_test, y_test)
```

```
0.9364161849710982
```

```
# Print model metrics
# Predict target
y_pred = log_model.predict(X_test)

# Create metrics object
score_metrics = Metrics()

# Print classification report, scores, and confusion matrix
score_metrics.model_score(log_model, y_test, y_pred)
```

```
                         precision    recall  f1-score   support

            functional       0.91      0.98      0.95      5579
functional needs repair       0.61      0.20      0.31       689
        non functional       1.00      1.00      1.00      3766

              accuracy                           0.94     10034
             macro avg       0.84      0.73      0.75     10034
          weighted avg       0.92      0.94      0.92     10034

Overall accuracy score 0.9364161849710982
Overall precision score 0.9227758355018202
Overall recall score 0.9364161849710982
Overall F1-score 0.9218428472967931

Confusion Matrix
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1896fca2790>
```

png

```
# Fit a logistic regression model
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)

# Retrieve the coefficients for each feature and class
coefficients = logistic_model.coef_
```

```python
# Calculate feature importances based on the absolute values of coefficients
importances = np.abs(coefficients)

# Normalize the importances within each class
importances /= np.sum(importances, axis=1)[:, np.newaxis]

# Aggregate importances across classes for an overall importance measure
# Mean importance across classes
overall_importance = np.mean(importances, axis=0)

# Sort and plot feature importances
sorted_indices = overall_importance.argsort()
sorted_feature_names = X_train.columns[sorted_indices]

plt.figure(figsize=(12, 30))

# Plot of only the top N features (top 20)
N = 20
top_indices = sorted_indices[-N:]

plt.barh(range(N), overall_importance[top_indices], align='center')
plt.yticks(range(N), sorted_feature_names[top_indices], fontsize=8)
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Top Feature Importances in Order')
plt.show()
```

[png]

```python
from sklearn.model_selection import cross_validate

def cross_val_test(K_folds, model_instance, X, y):

    # Lists to store train and test accuracy scores
    train_accuracy = []
    test_accuracy = []

    for fold in K_folds:
        # Perform cross-validation on full dataset to evaluate model performance
        cv_results = cross_validate(model_instance, X, y, cv=fold, return_train_score=True, n_jobs=-1)

        # Append the mean scores for each fold
        train_accuracy.append(np.mean(cv_results['train_score']))
        test_accuracy.append(np.mean(cv_results['test_score']))

    # Convert lists to numpy arrays for more efficient plotting
    train_accuracy = np.array(train_accuracy)
    test_accuracy = np.array(test_accuracy)

    # Plot the mean train and test scores against K folds
    plt.figure(figsize=(10, 6))
    plt.plot(K_folds, train_accuracy, label='Train Accuracy', marker='o')
    plt.plot(K_folds, test_accuracy, label='Test Accuracy', marker='o')
    plt.xlabel('Number of Folds (K)')
    plt.ylabel('Accuracy Score')
    plt.title(f'Cross-Validation Train vs Test Accuracy for {model_instance.__class__.__name__}')
    plt.legend()
    plt.grid(True)
    plt.show()

    return f"Cross-validation test completed for {model_instance.__class__.__name__}"

cross_val_test(range(3, 6), LogisticRegression(random_state=42), X, y)
```

[png]

```
'Cross-validation test completed for LogisticRegression'
```

DECISION TREE CLASSIFIER

```
# Fit the decision tree classifier to training data
dt_model = pipe_decision_tree.fit(X_train, y_train)

# Print the accuracy on test set
pipe_decision_tree.score(X_test, y_test)
```

```
0.9139924257524417
```

```
# Predict target
y_pred = dt_model.predict(X_test)

# Create metrics object
score_metrics = Metrics()

# Print classification report, scores, and confusion matrix
score_metrics.model_score(dt_model, y_test, y_pred)
```

```
                         precision    recall  f1-score   support

             functional       0.93      0.92      0.92      5579
functional needs repair       0.39      0.43      0.41       689
         non functional       1.00      1.00      1.00      3766

               accuracy                           0.91     10034
              macro avg       0.77      0.78      0.78     10034
           weighted avg       0.92      0.91      0.92     10034

Overall accuracy score 0.9139924257524417
Overall precision score 0.9182597840637288
Overall recall score 0.9139924257524417
Overall F1-score 0.9160174910575444

Confusion Matrix
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x18970341c50>
```

png

```
# Fit the decision tree classifier to training data
dt_model_f = pipe_decision_tree.named_steps['tree'].fit(X_train, y_train)

# Get feature importances from the decision tree
feature_importances = dt_model_f.feature_importances_

# Continue with the rest of your code
n_features = X_train.shape[1]
sorted_indices = feature_importances.argsort()
# Sort feature names based on importance order
sorted_feature_names = X.columns[sorted_indices]

plt.figure(figsize=(12, 30))

# Plot of only the top N features (top 20)
N = 20
top_indices = sorted_indices[-N:]
plt.barh(range(N), overall_importance[top_indices], align='center')
plt.yticks(range(N), sorted_feature_names[top_indices], fontsize=8)
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Feature Importances in Decision Tree Model');
```

png

DECISION TREE CLASSIFIER WITH CROSS VALIDATION

```
# Distribution of train and test accuracies over a range of K folds
cross_val_test(range(3, 6), DecisionTreeClassifier(random_state=42), X, y)
```

[png]

```
'Cross-validation test completed for DecisionTreeClassifier'
```

HANDLING THE IMBALANCE

```
# Previous original class distribution
print(y_train.value_counts())

# Fit SMOTE to training data
X_train_resampled, y_train_resampled = SMOTE().fit_resample(X_train, y_train)

# Preview synthetic sample class distribution
print('\n')
print(pd.Series(y_train_resampled).value_counts())
```

```
status_group
functional                 22309
non functional             14900
functional needs repair     2923
Name: count, dtype: int64


status_group
functional                 22309
non functional             22309
functional needs repair    22309
Name: count, dtype: int64
```

```
# Fit logistic regression, decision tree, and KNN classifiers
# resampled data

# Print the baseline logistic accuracy and f1 score on test set
y_pred = log_model.predict(X_test)
print("Baseline Logistic Regression Accuracy", log_model.score(X_test, y_test))
print("Baseline Logistic Regression F1-score", f1_score(y_test, y_pred, average='weighted'))

# Print the baseline decision tree accuracy and f1 score on test set
y_pred = dt_model.predict(X_test)
print("Baseline Decision Tree Accuracy", pipe_decision_tree.score(X_test, y_test))
print("Baseline Decision Tree F1-score", f1_score(y_test, y_pred, average='weighted'))

# Fit the logistic regression model on resampled data
log_model_resampled = pipe_logistic.fit(X_train_resampled, y_train_resampled)
y_pred_log = log_model_resampled.predict(X_test)

# Fit the decision tree classifier on resampled data
dt_model_resampled = pipe_decision_tree.fit(X_train_resampled, y_train_resampled)
y_pred_dt = dt_model_resampled.predict(X_test)
# Print the accuracy and f1 scores on SMOTE sample
print("Logistic Regression Accuracy with SMOTE", log_model_resampled.score(X_test, y_test))
print("Logistic Regression F1-score with SMOTE", f1_score(y_test, y_pred_log,
                                                          average='weighted'))
print("Decision Tree Accuracy with SMOTE", dt_model_resampled.score(X_test, y_test))
print("Decision Tree F1-score with SMOTE", f1_score(y_test, y_pred_dt,
                                                          average='weighted'))
```

```
Baseline Logistic Regression Accuracy 0.9364161849710982
Baseline Logistic Regression F1-score 0.9218428472967931
Baseline Decision Tree Accuracy 0.5605939804664142
```

```
Baseline Decision Tree F1-score 0.6002894371513324
Logistic Regression Accuracy with SMOTE 0.9365158461231812
Logistic Regression F1-score with SMOTE 0.9245022712645242
Decision Tree Accuracy with SMOTE 0.9168825991628463
Decision Tree F1-score with SMOTE 0.9186762932212614
```

CONCLUSION

From the results above:

1. Baseline Logistic Regression:

Accuracy: 93.64% F1-score: 92.18% The Logistic Regression model shows strong performance with high accuracy and F1-score, indicating it is correctly predicting both classes and maintaining a good balance between precision and recall.

2. Baseline Decision Tree:

Accuracy: 56.06% F1-score: 60.03% The Decision Tree model performs poorly in comparison, with significantly lower accuracy and F1-score. This suggests that the model may be overfitting the training data or failing to generalize well to the test set.

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

- **Jupyter Notebook** 100.0%