

1. Overview

In this Python Script, we will apply all the data transformations that were done in the 01_data_preprocessing python script on the test dataset. Moreover, we will also generate predictions of the test dataset with the best decision tree classifier model that was trained in the 02_model_creation. With all this, we will be able to obtain the predicted values and determine whether a pump will be functional or non-functional

2. Data Understanding

2.1 Data Description

This notebook will use the test dataset given to us from DrivenData called: Test_set_values

2.2 Import Necessary Libraries

```
In [1]: import pickle
import pandas as pd
import re # Import regular expressions library
```

2.3 Define global variables

```
In [2]: INPUT_PATH_Test_set_values = "../Data/Test_set_values.csv"
```

2.4 Functions

```
In [3]: def categorize_funder(funder):  
        """  
        Categorizes a funder name into specific groups based on keywords.  
  
        Args:  
        funder (str): A string representing the name of the funder to categorize.  
  
        Returns:  
        str: A category name representing the type of organization the funder belongs to.  
  
        This function takes a funder name, converts it to lowercase, removes leading and trailing spaces,  
        and categorizes it into predefined groups like 'Government', 'Religious Organizations',  
        'International Aid', 'Private Companies', or 'Individual/Other' based on keywords.  
        """  
        funder = funder.lower().strip() # convert to lowercase and strip whitespace  
        if any(x in funder for x in ['government', 'ministry', 'gov', 'minis']):  
            return 'Government'  
        elif any(x in funder for x in ['church', 'muslim', 'mus', 'islamic', 'islam']):  
            return 'Religious Organizations'  
        elif any(x in funder for x in ['ngo', 'foundation', 'fund', 'trust', 'social']):  
            return 'NGO'  
        elif any(x in funder for x in ['international', 'internatio', 'un', 'world']):  
            return 'International Aid'  
        elif any(x in funder for x in ['ltd', 'company', 'compa', 'group', 'enterprise']):  
            return 'Private Companies'  
        else:  
            return 'Individual/Other'
```

```
In [4]: def categorize_installer(installer):
        """
        Categorizes an installer name into specific groups based on keywords.

        Args:
        installer (str): A string representing the name of the installer to categorize.

        Returns:
        str: A category name representing the type of entity the installer belongs to.

        This function processes an installer name by converting it to lowercase and stripping any leading/trailing whitespace. It categorizes the name into predefined groups: 'DWE', 'Government', 'Community', 'NGO', 'Private Company', 'Institutional' based on specific keywords present in the installer's name. This helps in organizing installer data for better analysis and insight extraction.
        """
        installer = installer.lower().strip() # convert to lowercase and strip whitespace
        if 'dw' in installer:
            return 'DWE'
        elif any(x in installer for x in ['government', 'govt', 'gove']):
            return 'Government'
        elif any(x in installer for x in ['resource']):
            return 'Other'
        elif any(x in installer for x in ['community', 'villagers', 'village', 'community']):
            return 'Community'
        elif any(x in installer for x in ['ngo', 'unicef', 'foundat']):
            return 'NGO'
        elif 'company' in installer or 'contractor' in installer:
            return 'Private Company'
        elif any(x in installer for x in ['school', 'school', 'church', 'rc']):
            return 'Institutional'
        else:
            return 'Other'
```

```
In [5]: def group_scheme_management(value):
        """
        Categorizes scheme management types into broader, more generalized groups.

        Args:
        value (str): A string representing the scheme management type to categoriz

        Returns:
        str: A generalized category name representing the type of scheme managemen

        This function takes a specific scheme management type and categorizes it i
        more generalized groups such as 'Government', 'Community', 'Private Sector
        'Water Board', or 'Other'. This categorization aids in simplifying the ana
        and understanding of the data by reducing the number of distinct categorie
        making trends and patterns more discernible.
        """
        if value in ['VWC', 'Water authority', 'Parastatal']:
            return 'Government'
        elif value in ['WUG', 'WUA']:
            return 'Community'
        elif value in ['Company', 'Private operator']:
            return 'Private Sector'
        elif value == 'Water Board':
            return 'Water Board' # Retain this as a separate category if distinct
        else:
            return 'Other'
```

```
In [6]: def clean_text(text):
        """
        Cleans a text string by converting to lowercase, removing non-alphanumeric
        and replacing multiple spaces with a single space. If the input is solely

        Args:
        text (str or NaN): The text to be cleaned; can be a string, numeric, or Na

        Returns:
        str or NaN: The cleaned text, with all characters in lowercase, non-alphan
                    and multiple spaces collapsed to a single space, or the origin

        This function standardizes a text string by making it lowercase, stripping
        and then replacing sequences of spaces with a single space, facilitating u
        is numeric, it is assumed to be standardized already and is returned witho
        """
        if pd.isna(text):
            return text
        if isinstance(text, (int, float)): # Check if the input is numeric
            return text
        text = text.lower() # Convert to lowercase
        text = ''.join(char for char in text if char.isalpha() or char.isspace())
        text = re.sub(r'\s+', ' ', text) # Replace multiple spaces with a single
        return text
```

3. Code

3.1 Import the dataset

```
In [7]: df_predict = pd.read_csv(INPUT_PATH_Test_set_values)
df_predict.head()
```

Out[7]:

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude
0	50785	0.0	2013-02-04	Dmdd	1996	DMDD	35.290799	-4.059696
1	51630	0.0	2013-02-04	Government Of Tanzania	1569	DWE	36.656709	-3.309214
2	17168	0.0	2013-02-01	NaN	1567	NaN	34.767863	-5.004344
3	45559	0.0	2013-01-22	Finn Water	267	FINN WATER	38.058046	-9.418672
4	49871	500.0	2013-03-27	Bruder	1260	BRUDER	35.006123	-10.950412

5 rows × 40 columns

3.2 Apply the same data transformations on df_predict as the ones done in 00_data_understanding

3.2.1 Applying transformation functions

Column 'funder'

```
In [8]: # Handling NaN values with a filler string like 'Unknown'
df_predict['funder'] = df_predict['funder'].fillna('Unknown').astype(str)

# Apply the mapping function to the 'funder' column
df_predict['funder_type'] = df_predict['funder'].apply(categorize_funder)

# Check the categorized data
print(df_predict['funder_type'].value_counts())
```

```
Individual/Other      9955
Government            2438
International Aid     2093
Religious Organizations  329
NGO                   29
Private Companies      6
Name: funder_type, dtype: int64
```

Column 'installer'

```
In [9]: # Handling NaN values with a filler string like 'Unknown'
df_predict['installer'] = df_predict['installer'].fillna('Unknown').astype(str)

# Apply the mapping function to the 'installer' column
df_predict['installer_type'] = df_predict['installer'].apply(categorize_instal

# Now you can check your categorized data
print(df_predict['installer_type'].value_counts())
```

```
Other      8480
DWE        4537
Government  926
Community  599
Institutional 185
NGO         93
Private Company 30
Name: installer_type, dtype: int64
```

Column 'scheme_management_grouped'

```
In [10]: # Apply the grouping function to the 'scheme_management' column
df_predict['scheme_management_grouped'] = df_predict['scheme_management'].appl

# Check the new value counts to see the grouped data
print(df_predict['scheme_management_grouped'].value_counts(normalize=True))
```

```
Government      0.699663
Community       0.131852
Other           0.083838
Water Board     0.048081
Private Sector  0.036566
Name: scheme_management_grouped, dtype: float64
```

3.2.2 Converting data types

```
In [11]: # Converting 'construction_year' to object
df_predict['construction_year'] = df_predict['construction_year'].astype('object')
```

```
In [12]: df_predict.columns
```

```
Out[12]: Index(['id', 'amount_tsh', 'date_recorded', 'funder', 'gps_height',
               'installer', 'longitude', 'latitude', 'wpt_name', 'num_private',
               'basin', 'subvillage', 'region', 'region_code', 'district_code', 'lg
               a',
               'ward', 'population', 'public_meeting', 'recorded_by',
               'scheme_management', 'scheme_name', 'permit', 'construction_year',
               'extraction_type', 'extraction_type_group', 'extraction_type_class',
               'management', 'management_group', 'payment', 'payment_type',
               'water_quality', 'quality_group', 'quantity', 'quantity_group',
               'source', 'source_type', 'source_class', 'waterpoint_type',
               'waterpoint_type_group', 'funder_type', 'installer_type',
               'scheme_management_grouped'],
              dtype='object')
```

3.2.3 Drop unnecessary columns

```
In [13]: drop_column_list = ['scheme_name', 'num_private', 'wpt_name', 'subvillage', 'l
               'extraction_type', 'management', 'payment', 'water_quality
               'waterpoint_type_group', 'date_recorded', 'funder', 'install
               'longitude', 'latitude', 'region_code', 'district_code', 'cons
```

```
In [14]: df_predict = df_predict.drop(drop_column_list, axis=1)
```

3.2.3 Cleaning the data set

```
In [15]: # Apply the cleaning function to each object-type column in the DataFrame
for col in df_predict.select_dtypes(include='object').columns:
    df_predict[col] = df_predict[col].apply(clean_text)
```

3.2 Fillna with the modes calculated in 01_data_preprocessing

```
In [16]: (df_predict.isna().sum()/len(df_predict))*100
```

```
Out[16]: id                0.000000
amount_tsh              0.000000
gps_height              0.000000
basin                  0.000000
region                 0.000000
population             0.000000
public_meeting         5.528620
permit                4.962963
extraction_type_class  0.000000
management_group       0.000000
payment_type           0.000000
quality_group          0.000000
quantity_group         0.000000
source_type            0.000000
waterpoint_type        0.000000
funder_type            0.000000
installer_type         0.000000
scheme_management_grouped 0.000000
dtype: float64
```

From the python script 01_data_preprocessing we know that public_meeting_mode is 1.0 and the permit_mode is 1.0. So we are going to directly fill the NaNs of public_meeting and of permit with the value 1.0

Fillna in column 'public_meeting'

```
In [17]: df_predict['public_meeting'].fillna(1.0, inplace=True)
```

Fillna in column 'permit'

```
In [18]: df_predict['permit'].fillna(1.0, inplace=True)
```

Let's check that there are no more null-values left


```
In [19]: (df_predict.isna().sum()/len(df_predict))*100
```

```
Out[19]: id                0.0
amount_tsh              0.0
gps_height              0.0
basin                  0.0
region                 0.0
population              0.0
public_meeting         0.0
permit                 0.0
extraction_type_class  0.0
management_group       0.0
payment_type           0.0
quality_group          0.0
quantity_group         0.0
source_type            0.0
waterpoint_type        0.0
funder_type            0.0
installer_type         0.0
scheme_management_grouped 0.0
dtype: float64
```

3.3 Doing target encoder on the categorical columns

Let's apply a one hot encoder for the categorical columns that have 6 or less categories

```
In [20]: # Capture categorical columns from X_train for encoding
categorical_columns = df_predict.select_dtypes(include=['object', 'category'])

# Encoding the categorical columns in df_predict
for col in categorical_columns:
    if df_predict[col].nunique() <= 6:
        # Apply OneHotEncoder for columns with 6 or fewer unique values
        df_predict = pd.get_dummies(df_predict, columns=[col], drop_first=True)
```

Let's call in the saved fits (for the categorical columns that have more than 6 categories) applied to the categorical columns in the 01_data_preprocessing script

In [21]: `df_predict.columns`

```
Out[21]: Index(['id', 'amount_tsh', 'gps_height', 'basin', 'region', 'population',
               'extraction_type_class', 'payment_type', 'source_type',
               'waterpoint_type', 'installer_type', 'public_meeting_True',
               'permit_True', 'management_group_other', 'management_group_parastata
1',
               'management_group_unknown', 'management_group_usergroup',
               'quality_group_fluoride', 'quality_group_good', 'quality_group_milky',
               'quality_group_salty', 'quality_group_unknown', 'quantity_group_enoug
h',
               'quantity_group_insufficient', 'quantity_group_seasonal',
               'quantity_group_unknown', 'funder_type_individualother',
               'funder_type_international aid', 'funder_type_ngo',
               'funder_type_private companies', 'funder_type_religious organization
s',
               'scheme_management_grouped_government',
               'scheme_management_grouped_other',
               'scheme_management_grouped_private sector',
               'scheme_management_grouped_water board'],
              dtype='object')
```

```
In [22]: # Column 'basin'
basin_pickle = pickle.load(open('model_objects/basin_target_encoder.pickle', '
df_predict['basin'] = basin_pickle.transform(df_predict['basin'])

# Column 'extraction_type_class'
extraction_type_class_pickle = pickle.load(open('model_objects/extraction_type
df_predict['extraction_type_class'] = extraction_type_class_pickle.transform(d

# Column 'installer_type'
installer_type_pickle = pickle.load(open('model_objects/installer_type_target_
df_predict['installer_type'] = installer_type_pickle.transform(df_predict['ins

# Column 'payment_type'
payment_type_pickle = pickle.load(open('model_objects/payment_type_target_enco
df_predict['payment_type'] = payment_type_pickle.transform(df_predict['payment

# Column 'region_target'
region_target_pickle = pickle.load(open('model_objects/region_target_encoder.p
df_predict['region'] = region_target_pickle.transform(df_predict['region'])

# Column 'source_type'
source_type_pickle = pickle.load(open('model_objects/source_type_target_encode
df_predict['source_type'] = source_type_pickle.transform(df_predict['source_ty

# Column 'waterpoint_type'
waterpoint_type_pickle = pickle.load(open('model_objects/waterpoint_type_targe
df_predict['waterpoint_type'] = waterpoint_type_pickle.transform(df_predict['w
```

3.4 Dealing with numerical columns

Let's call in the saved fits applied to the numerical columns in the 01_data_preprocessing script

```
In [23]: # Capture numerical columns
numerical_columns = df_predict.select_dtypes(include=['int64', 'float64']).col

# Let's also drop column 'id' from the numerical_columns as they don't serve f
numerical_columns = numerical_columns.drop('id')

# Numerical Columns
numerical_columns_pickle = pickle.load(open('model_objects/numerical_columns_s
df_predict[numerical_columns] = numerical_columns_pickle.transform(df_predict[
```

3.5 Apply the Decision Tree Classifier created in 02_model_creation

```
In [24]: df_predict
```

Out[24]:

	id	amount_tsh	gps_height	basin	region	population	extraction_type_class
0	50785	-0.100621	1.915327	-0.379005	-0.984626	2979.061988	2.617222
1	51630	-0.100621	1.299135	-0.379010	-1.835764	2783.936606	-0.521411
2	17168	-0.100621	1.296248	-0.379005	1.032355	4642.273578	2.617222
3	45559	-0.100621	-0.579749	-0.378561	3.744407	2319.352363	2.617222
4	49871	0.055600	0.853225	-0.378561	-0.024106	553.932240	-0.521411
...
14845	39307	-0.100621	-0.915985	-0.378824	0.161577	182.264846	1.165688
14846	18990	0.211821	-0.965049	-0.379010	0.365118	27499.818332	-0.463637
14847	28749	-0.100621	1.164929	-0.379005	1.032355	1854.768120	-0.521411
14848	33492	-0.100621	0.475139	-0.379106	-0.024106	1390.183877	-0.521411
14849	68707	-0.100621	-0.270931	-0.379106	-0.024106	368.098543	-0.521411

14850 rows × 35 columns

```
In [25]: # Loading the pickle for the best Decision Tree Classifier
best_tree_pickle = pickle.load(open('model_objects/best_tree.pickle', 'rb'))

# Let's drop the 'id' column
df_predict_copy = df_predict.drop('id', axis=1)
```

```
In [26]: # Decision Tree Classifier
df_predict['status_group'] = best_tree_pickle.predict_proba(df_predict_copy)[:
```

```
In [27]: # Apply a threshold to the probabilities of status_group to determine to which  
df_predict['status_group_class'] = df_predict['status_group'].map(lambda x: 'N
```

```
In [28]: df_predict[['id', 'status_group', 'status_group_class']]
```

Out[28]:

	id	status_group	status_group_class
0	50785	0.884615	Non-functional
1	51630	0.163102	Functional
2	17168	0.666667	Non-functional
3	45559	0.988601	Non-functional
4	49871	0.576923	Non-functional
...
14845	39307	0.816399	Non-functional
14846	18990	0.154597	Functional
14847	28749	0.255814	Functional
14848	33492	0.255814	Functional
14849	68707	0.984899	Non-functional

14850 rows × 3 columns

4. Export the data

```
In [29]: df_predict[['id', 'status_group_class']].to_excel('Final_results.xlsx', index=
```