1. Overview

This competition, "Pump it Up: Data Mining the Water Table," hosted on DrivenData, challenges participants to predict the functional status of water pumps across Tanzania using a provided dataset. The contest spans from 2024 and aims to enhance access to clean, potable water by identifying malfunctioning water pumps. Participants are supplied with extensive data on various characteristics of the water points, from construction year to water quality. The primary goal is to classify each water point into one of three categories: functional, functional needs repair, and non-functional. This analysis could guide strategic decisions for improving water access and infrastructure investments in developing regions.

2. Business Understanding

The core objective of the "Pump it Up: Data Mining the Water Table" competition is to enable the identification of water pumps in Tanzania that are functional, require repairs, or are non-functional. The insights derived from this analysis will directly influence decisions regarding maintenance, investments, and resource allocation in the water infrastructure sector. Stakeholders, including government agencies and NGOs, will use these findings to prioritize and streamline efforts towards ensuring reliable water access. By effectively categorizing water points, the project aims to enhance operational efficiencies and reduce downtime due to pump failures. The ultimate goal is to support sustainable water management practices that can significantly impact public health and economic development in Tanzania.

Primary stakeholders for this project are the Tanzanian government and international development organizations focused on improving water access in the region.

3. Data Understanding

3.1 Data Description

Drawing from a comprehensive dataset provided by the "Pump it Up: Data Mining the Water Table" competition on DrivenData, our analysis is centered around extensive information regarding water points across Tanzania. This dataset includes:

- Geographic data such as location coordinates, altitude, and administrative divisions (region, district, and ward).
- Water point specifics such as the type, construction year, funding organization, and managing entity.
- Operational data including the water source, extraction type, water quality, and current functional status of each water pump.

Our investigation targets three key objectives: identifying patterns of pump functionality, understanding factors leading to pump failures or repairs, and assessing the impacts of management practices on pump operability. By analyzing these elements, we aim to derive actionable insights that can guide infrastructural improvements and strategic investments in water resource management. The outcome of this analysis will inform decision-making processes for stakeholders involved in Tanzanian water supply, optimizing interventions for enhanced water accessibility and reliability. This focused approach empowers our stakeholders to efficiently address the most critical needs, leveraging data-driven strategies to improve public health and community resilience.

3.2 Code

The intention of this notebook is to show the general procedure of the whole project. In each one of the sections, before showing the results, we will provide a link to the notebooks that include a step by step description of the procedure.

All of the following analysis of section 3.2 was performed on the train dataset provided by DrivenData.

3.2.1 Descriptive and Exploratory Analysis

To have the detailed step by step results of the exploratory analysis, please see the data understanding notebook through this link [Go to Notebook 00_data_understanding.ipynb (00_data_understanding.ipynb)](00_data_understanding.ipynb)

In [1]:
```
# # Reading the dataset
# df_train = pd.read_csv(INPUT_PATH_Training_set_values)
# df_train.head()
```

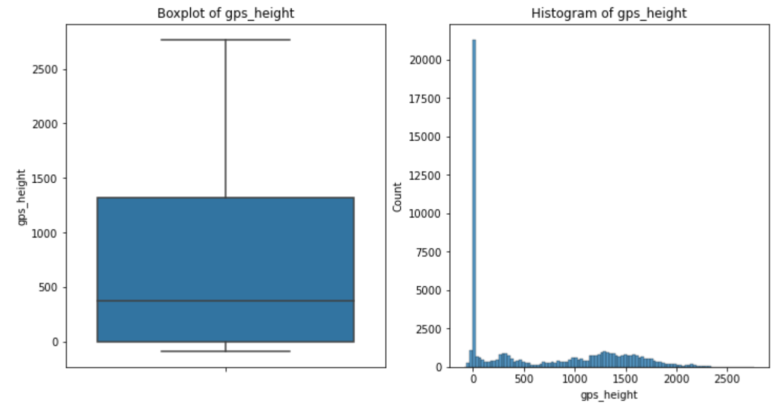
	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wpt_name	num_private	...	payment_type	water_quality	quality_grou
0	69572	6000.0	2011-03-14	Roman	1390	Roman	34.938093	-9.856322	none	0	...	annually	soft	goo
1	8776	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.698766	-2.147466	Zahanati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

5 rows × 40 columns

3.2.1.1 Univaried Analysis

```
Stats for gps_height:
Max: 2770
Min: -90
Mean: 668.297239057239
Median: 369.0
Standard Deviation: 693.11635032505
```

```
Coefficient of Variation: 1.037137833013979
Skewness: 0.4624020849809572
Kurtosis: -1.2924401348688863
25th percentile (Q1): 0.0
50th percentile (Median): 369.0
75th percentile (Q3): 1319.25
```



3.2.1.2 Multivaried Analysis

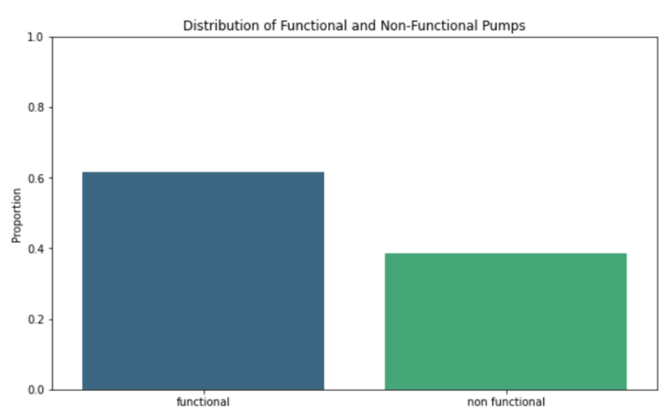
```
Table for payment_type:
```

payment_type	annually	monthly	never pay	on failure	other	per bucket	unknown
status_group							
functional	8.49	16.99	35.27	7.53	1.89	18.88	10.94
functional needs repair	5.72	21.47	44.17	6.42	2.73	9.47	10.01
non functional	2.87	8.29	52.85	5.29	1.42	10.89	18.39

3.2.2 Data preprocessing

To have the detailed step by step results of the data preprocessing, please see the data preprocessing notebook through this link [Go to Notebook 01_data_preprocessing.ipynb (01_data_preprocessing.ipynb)](01_data_preprocessing.ipynb)

3.2.2.1 Imbalance check



Status Group

3.2.2.2 Categorical encoding

In [2]:
```
# # Identifying categorical columns
# categorical_columns = X_train.select_dtypes(include=['object', 'category']).

# # Printing the list of categorical columns
# print("Categorical columns in X_train:")
# print(categorical_columns)
```

Let's do a code to apply one hot enconder on the columns that have less than 6 variables and a target enconder on the columns that have more than 6 variables. The reason why we decide to not apply target encoding to all the columns directly is to avoid overfitting

In [3]:
```
# # Check if 'y_train' and 'y_test' need to be converted to a numeric type
# if y_train.dtype == 'object':
#     y_train = y_train.astype('category').cat.codes
# if y_test.dtype == 'object':
#     y_test = y_test.astype('category').cat.codes

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

# # Initialize encoders
# target_encoder = TargetEncoder()

# # Encoding the categorical columns in X_train and X_test
# for col in categorical_columns:
#     if X_train[col].nunique() <= 6:
#         # Apply OneHotEncoder for columns with 6 or fewer unique values
#         X_train = pd.get_dummies(X_train, columns=[col], drop_first=True)
#         X_test = pd.get_dummies(X_test, columns=[col], drop_first=True)
#     else:
#         # Apply TargetEncoder for columns with more than 6 unique values
#         X_train[col] = target_encoder.fit_transform(X_train[col], y_train)
#         X_test[col] = target_encoder.transform(X_test[col])
#         pickle.dump(target_encoder, open(f"model_objects/{col}_target_encode

# # Display the DataFrame to check the results
# X_train.head()
```

	amount_tsh	gps_height	population	basin	region	extraction_type_class	payment_type	source_type	waterpoint_type	installer_type	...	quantity_grou
3607	50.0	2092	160	0.346722	0.315956	0.300187	0.277862	0.301175	0.298881	0.383794	...	
50870	0.0	0	0	0.346722	0.443875	0.309484	0.475440	0.447489	0.324167	0.570368	...	
20413	0.0	0	0	0.485901	0.398196	0.805243	0.475440	0.447489	0.821499	0.383794	...	
52806	0.0	0	0	0.311216	0.398196	0.300187	0.226650	0.343784	0.298881	0.383794	...	
50091	300.0	1023	120	0.432348	0.398697	0.805243	0.308180	0.447489	0.821499	0.383794	...	

5 rows × 34 columns

3.2.2.3 Numerical encoding

In [4]:
```
# X_test[numerical_columns] = scaler.transform(X_test[numerical_columns])

# # Display the DataFrame to check the results
# X_test.head()
```

	amount_tsh	gps_height	population	basin	region	extraction_type_class	payment_type	source_type	waterpoint_type	installer_type	...	quantity_gr
2980	-0.100621	-0.965049	-0.379739	0.205860	-0.699807	2.617222	1.090170	0.850673	2.622191	-0.281827	...	
5246	-0.100621	-0.965049	-0.379739	0.205860	1.453840	-0.463637	0.771866	0.850673	-0.359301	-0.005208	...	
22659	-0.097497	1.452101	-0.066689	-0.540016	-0.633090	-0.521411	-0.897587	-1.112570	-0.510890	-0.281827	...	
39888	-0.100621	-0.965049	-0.379739	1.471270	0.131062	-0.463637	0.771866	0.850673	-0.359301	-0.005208	...	
13361	-0.084999	0.635320	0.117334	-0.540016	0.663779	1.165688	-0.897587	1.017142	0.869823	-0.005208	...	

5 rows × 34 columns

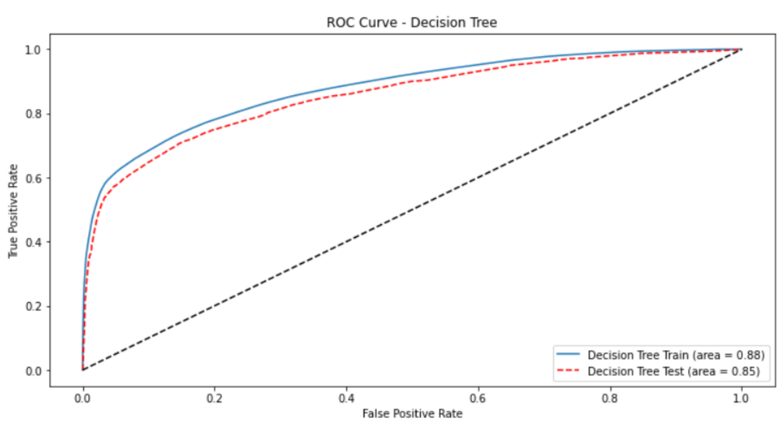
3.2.3 Model Creation

To have the detailed step by step results of the data preprocessing, please see the data preprocessing notebook through this link Go to Notebook 02_model_creation.ipynb (02_model_creation.ipynb)

3.2.3.1 Hypertuning Decision Tree Classifier model

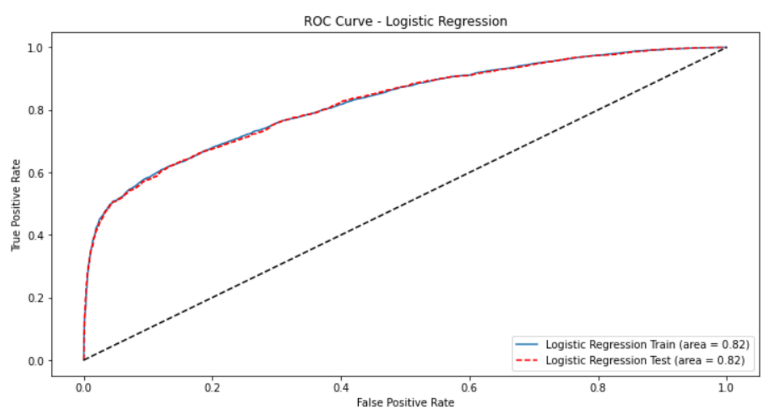
We created a baseline Decision Tree Classifier model and then by using the function GridSearchCV, we hypertuned the best combinations of parameters to improve the performance of the model.

In [5]:
```
# # Initialize the Decision Tree model
# decision_tree = DecisionTreeClassifier()

# # Define the parameter grid to search
# param_grid = {
#     'max_depth': range(5, 10),  # Explore depths from 1 to 20
#     'min_samples_split': range(5, 15, 2),  # Minimum number of samples requi
#     'min_samples_leaf': range(5, 10),  # Minimum number of samples required
#     'max_features': ['auto', 'log2', None]  # Number of features to consider
# }

# # Define the scoring function using AUC
# scorer = make_scorer(roc_auc_score, needs_proba=True)

# # Setup the grid search with cross-validation
# grid_search = GridSearchCV(estimator=decision_tree, param_grid=param_grid, s

# # Fit grid search on the training data
# grid_search.fit(X_train, y_train)

# # Find the best model
# best_tree = grid_search.best_estimator_

# # Save the best_tree
# pickle.dump(best_tree, open(f"model_objects/best_tree.pickle", 'wb'))

# # Output the best parameter combination and the corresponding score
# print("Best parameters found:", grid_search.best_params_)
# print("Best AUC achieved:", grid_search.best_score_)

# # Optional: Evaluate the best model on the test set
# y_pred_proba_best_tree = best_tree.predict_proba(X_test)[:, 1]
# test_auc = roc_auc_score(y_test, y_pred_proba_best_tree)
# print("Test AUC of best model:", test_auc)
```



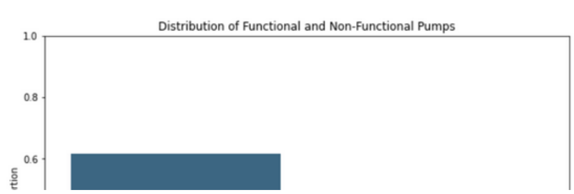
3.2.3.2 Hypertuning Logistic Regression model

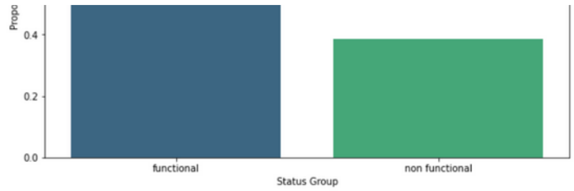
In [6]:
```
# # Initialize the Logistic Regression model
# logistic_regression = LogisticRegression()

# # Define the parameter grid to search
# param_grid = {
#     'C': [0.001, 0.01, 0.1, 1, 10, 100],  # Inverse of regularization streng
#     'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],  # Algorit
#     'max_iter': [100, 200, 300],  # Maximum number of iterations taken for t
# }

# # Define the scoring function using AUC
# scorer = make_scorer(roc_auc_score, needs_proba=True)

# # Setup the grid search with cross-validation
# grid_search = GridSearchCV(estimator=logistic_regression, param_grid=param_g

# # Fit grid search on the training data
# grid_search.fit(X_train, y_train)

# # Find the best model
# best_log_reg = grid_search.best_estimator_

# # Output the best parameter combination and the corresponding score
# print("Best parameters found:", grid_search.best_params_)
# print("Best AUC achieved:", grid_search.best_score_)

# # Optional: Evaluate the best model on the test set
# y_pred_proba_best_log_reg = best_log_reg.predict_proba(X_test)[:, 1]
# test_auc = roc_auc_score(y_test, y_pred_proba_best_log_reg)
# print("Test AUC of best model:", test_auc)
```



3.2.3.3 Conclusions

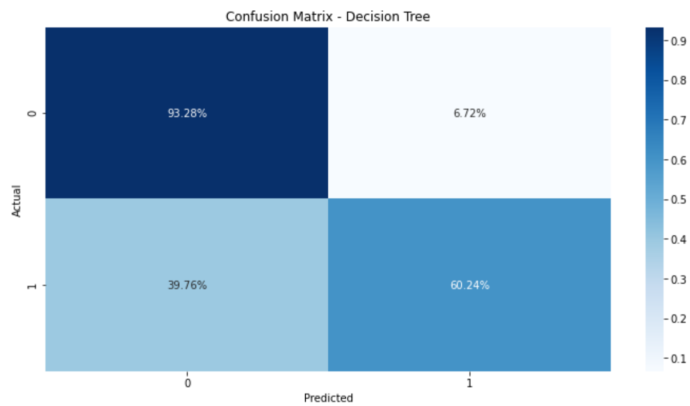
Considering the distribution of the dependent variable





As we can see there is not an imbalance problem even though the majority of pumps are functional.

Diving into the model results, let's begin by looking into the confusion matrix



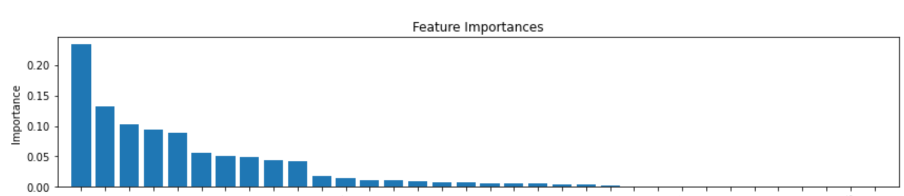
The confusion matrix indicates that the model has a high rate of false negatives (39.76%), which can significantly impact the business by failing to identify non-functional pumps that need repairs. This can lead to prolonged downtimes and negatively affect customer satisfaction. The false positive rate (6.72%) is relatively low, meaning fewer resources will be wasted on unnecessary maintenance. However, the primary concern should be reducing the false negative rate to ensure that non-functional pumps are correctly identified and repaired promptly.

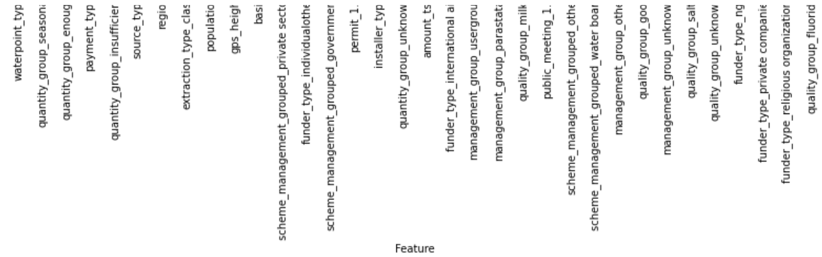
Based on the metrics, the best AUC and confusion matrix is obtained with a Decision Tree Classifier. As is observable, the AUC is of 0.85 for the test. In the case of the Logistic Regression model, an AUC of 0.82 is obtained for the test.

The variables that are most important and that permit us to best descriminate are:

1. waterpoint_type
2. quantity_group_seasonal
3. quantity_group_enough
4. payment_type
5. quantity_group_insufficient

We are interested in these 5 variables because they are the ones that have the most influence when determining whether a pump is functional or non-functional.





Considering that we used a one-hot encoder and that the categories for each variable were treated as independent variables, the three variables that contribute the most to the model are:

1. waterpoint_type
2. quantity_group
3. payment_type

Here we will show the contingency tables for each variable divided into functional, functional with repairs, and non functional pumps:

```
Table for payment_type:
```

payment_type status_group	annually	monthly	never pay	on failure	other	per bucket	unknown
functional	8.49	16.99	35.27	7.53	1.89	18.88	10.94
functional needs repair	5.72	21.47	44.17	6.42	2.73	9.47	10.01
non functional	2.87	8.29	52.85	5.29	1.42	10.89	18.39

```
Table for quantity_group:
```

quantity_group status_group	dry	enough	insufficient	seasonal	unknown
functional	0.49	67.11	24.54	7.21	0.66
functional needs repair	0.86	55.59	33.59	9.64	0.32
non functional	26.52	40.04	25.25	5.74	2.46

```
Table for waterpoint_type:
```

waterpoint_type status_group	cattle trough	communal standpipe	communal standpipe multiple	dam	hand pump	improved spring	other
functional	0.26	54.95	6.93	0.02	33.49	1.75	2.60
functional needs repair	0.05	52.35	15.01	0.00	23.84	1.97	6.79
non functional	0.13	37.40	14.11	0.00	24.77	0.60	22.99

3.2.3.4 Recommendations

1. Considering that most of the functional pumps have monthly payment plans or a per bucket, the Tanzanian government can consider modifying the existing payment plans of those pumps where the payments are different from those payment types, so that the chance of the pump being functional can be increased.

2. Considering that almost none of the functional pumps are dry, it is possible to verify which pumps are dry as a proxy variable to know if they are functional or not and thus focus efforts on repairing them.

3. Considering that non-functional pumps have in most cases a waterpoint_type different from cattle trough, communal standpipe, communal standpipe multiple, dam, hand pump and improved spring, it is possible to verify which pumps do not have these waterpoint_types as a proxy variable to know if they are functional or not and thus focus efforts on repairing them.

3.3 Predictions in Test_set_values dataset

To have the detailed step by step results of the exploratory analysis, please see the data understanding notebook through this link [Go to Notebook 03_predict.ipynb (03_predict.ipynb)](03_predict.ipynb)

Based on the selected final model and the conclusions gained from the training data analysis, in this section we use the model to predict whether or not the pump is functional on the test dataset provided by DrivenData.

Out[27]:

	id	status_group	status_group_class
0	50785	0.180556	Functional
1	51630	0.000000	Functional
2	17168	0.180505	Functional
3	45559	0.777778	Non-functional
4	49871	0.404255	Functional
...	...	...	...
14845	39307	0.888889	Non-functional
14846	18990	0.666667	Non-functional
14847	28749	0.777778	Non-functional
14848	33492	0.000000	Functional
14849	68707	0.882353	Non-functional

14850 rows × 3 columns

It is important to mention that it is not possible to know how well the predictions perform because we don't have the real labels as this was a competition