Predicting the Condition of Tanzanian Water Wells: A Machine Learning Approach to Maintenance and Sustainability

By

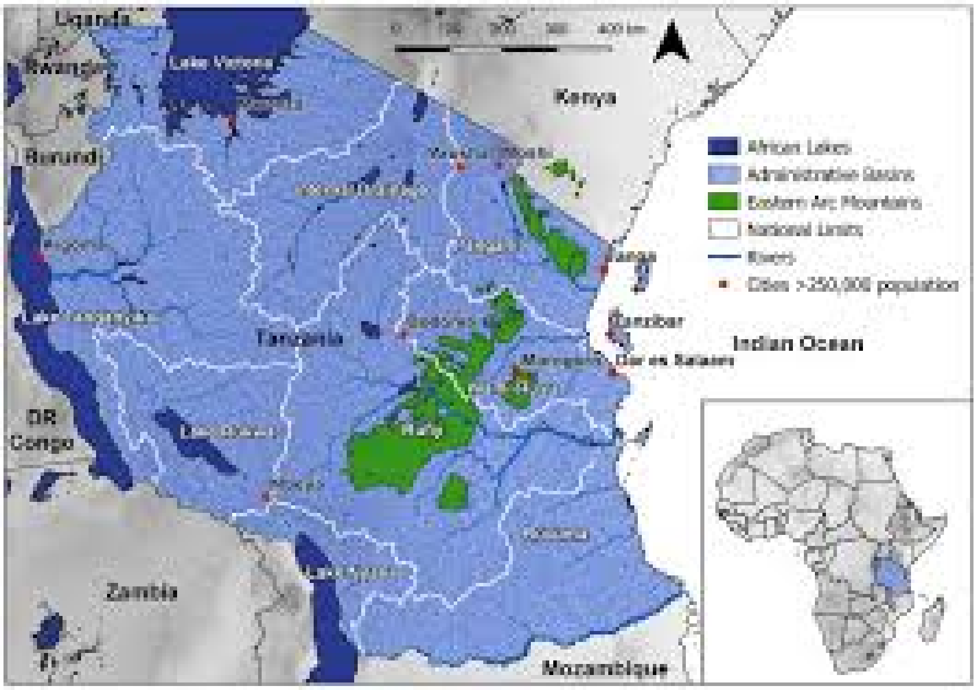
Knight Mbithe Wambua

Business Understanding

Project Overview

Tanzania, with over 57 million people, faces challenges in providing reliable clean water, especially in rural areas where 70% of the population lives. Over 40% of water wells are non-functional and require repair. Current manual inspections are inefficient, with only about 10,000 wells being checked annually out of the 40,000 needing attention.

A predictive model using data such as pump type, installation year, maintenance history, and location could help prioritize repairs and ensure continued access to clean water for communities.



A map of Tanzania

Problem Statement

Tanzania's water wells face frequent failures, with 40% of wells non-functional. Manual inspections are slow, leaving many communities without safe water. Each failed well can affect 500-1,000 people. A predictive model is needed to identify at-risk wells and enable timely interventions, ensuring continued access to clean water.

non-functional well that needs repair

Objectives

Major Objective

To develop a machine learning classifier that predicts the condition of water wells in Tanzania based on available features such as pump type, installation year, maintenance history, and geographical location.

Specific Objectives

1. To preprocess the provided historical data related to water wells, ensuring it is clean and ready for model training.
2. To analyze the dataset to identify important features and understand the relationships between variables that influence well conditions.
3. To train and evaluate at least three classification models (Logistic Regression as a baseline, Decision Tree before tuning, and Hyper-tuned Decision Trees) to determine the best-performing model for predicting well conditions.
4. To optimize the selected model's performance by tuning hyperparameters and improving classification metrics such as accuracy, precision, recall, and F1 score.

Success Criteria

- Accuracy: The model should achieve an accuracy score of 75% or higher, meaning it correctly predicts the condition of water wells in at least 75% of cases.
- Precision: The model should aim for a precision score of 75%, ensuring that when it predicts a well needs repair, it is correct 75% of the time.
- Recall: The model should aim for a recall score of 75%, ensuring that it identifies 75% of wells in need of repair.
- F1 Score: A balanced F1 score above 0.75 is expected, reflecting a good balance between precision and recall in identifying wells needing repair. ll in identifying wells needing repair.

Data Understanding

This section outlines the structure, characteristics, and quality of the provided datasets. It includes an overview of basic properties, statistical summaries, and missing values analysis, along with additional observations to guide preprocessing and analysis.

1. Loading and the Basic Properties of the DataFrames

Train Values

- Rows: 59,400
- Columns: 40
- Categorical Columns: 34
- Numerical Columns: 6
- Boolean Columns: 2 (public_meeting, permit)
- Target Variable: Not present in this dataset.

Train Labels

- Rows: 59,400 (aligned with Train Values)
- Columns: 2 (id, status_group)
- Target Variable: status_group (classification task with three categories: functional, functional needs repair, non-functional).

Test Values

- Rows: 14,850
- Columns: 40
- Categorical Columns: 34 (same as Train Values)
- Numerical Columns: 6
- Boolean Columns: 2
- Target Variable: Not present.

2. Statistical Summary

Numerical Columns (Train Values)

- Amount (amount_tsh): Mean = 318, with high variability (std = 2998), ranging from 0 to 350,000, mostly concentrated at 0.
- GPS Height: Mean = 668 meters, ranging from -90 to 2770 meters, indicating diverse elevations.
- Longitude & Latitude: Longitude ranges from 0 to 40.35° and Latitude from -11.65° to -0.02°, covering southern Tanzania.
- Private Connections (num_private): Mean = 0.47, with most values being 0, a few locations have up to 1776 private connections.
- Region & District Code: Region code ranges from 1 to 99; District code ranges from 0 to 80.
- Population: Mean = 180, with a large spread (0 to 30,500), many locations have 0 population.
- Construction Year: Mean = 1301, with a wide range from 0 to 2013, suggesting old and newer wells.

Numerical Columns (Train Labels)

- The ID column is just a sequential identifier for the records, showing the total number of entries along with the status_group the target variable which is categorical.

Numerical Columns (Test Values)

- ID: 14,850 entries, with a mean of 37,162 and a range from 10 to 74,249.
- Amount (amount_tsh): Mean = 323, with many zero values and a max of 200,000.
- GPS Height: Mean = 655 meters, range from -57 to 2777 meters.
- Longitude & Latitude: Longitude (0 to 40.33°), Latitude (-11.56° to -0.02°).
- Private Connections (num_private): Mean = 0.42, with a few values up to 669.
- Region & District Code: Region (1 to 99), District (0 to 80).
- Population: Mean = 184, with a range from 0 to 11,469.

- Construction Year: Mean = 1,290, ranging from 0 to 2013.

The dataset shows similar patterns to the training set, with many zero values in certain fields

3. Missing Values

Train Values

- scheme_name: 48.5% missing
- scheme_management: 6.5% missing
- installer: 6.2% missing
- funder: 6.1% missing
- public_meeting: 5.6% missing
- permit: 5.1% missing
- sub_village: 0.6% missing
- wpt_name: 0.003% missing

scheme_name has the most missing values.

Train Labels

- The Train Labels dataset contains no missing values, making it clean and ready for model training without the need for imputation.

Test Values

- scheme_name: 48.77% missing
- scheme_management: 6.53% missing
- installer: 5.91% missing
- funder: 5.86% missing
- public_meeting: 5.53% missing
- permit: 4.96% missing
- sub_village: 0.67% missing

scheme_name has the highest percentage of missing values.

4. Outlier Detection

Outliers can significantly affect the model's performance and may need to be handled appropriately. Below are the outliers identified in the datasets:

Outliers in Train Values:

- amount_tsh: 11161
- gps_height: 0
- longitude: 1812
- latitude: 0
- num_private: 757
- region_code: 3604
- district_code: 4188
- population: 4383
- construction_year: 0

Outliers in Train Labels:

- id: 0

Outliers in Test Values:

- amount_tsh: 2806
- gps_height: 0
- longitude: 457
- latitude: 0
- num_private: 194
- region_code: 876
- district_code: 1049
- population: 1070
- construction_year: 0

Outliers like negative or extreme values in gps_height, amount_tsh, and construction_year need to be reviewed further to determine if they should be removed or adjusted based on the context of the data.

5. Duplicate Entries

Duplicate entries can introduce bias and distort results. Here's the summary of duplicate rows found in each dataset:

Duplicate Rows in Train Values: 0
Duplicate Rows in Train Labels: 0
Duplicate Rows in Test Values: 0

No duplicates were found in any of the datasets, ensuring that the datasets are unique and no data redundancy exists.

Data Preprocessing and Data Analysis

1. Data Cleaning

In this stage, we combine train_values and train_labels to facilitate efficient data preparation while addressing key issues like missing values, ensuring consistency in the values and the columns, and dealing with outliers irrelevant columns, and columns with constant or similar values. All the data cleaning steps done to the train dataset will also be done to the test dataset to ensure consistency.

1.1 Dealing with Missing Values

Missing values are handled systematically to ensure data integrity and prevent bias in modeling. The following strategy is adopted:

- Categorical Columns: Missing values in categorical columns like funder, installer, public_meeting, scheme_management, permit, and wpt_name are filled with their mode (most frequently occurring value).

- Subvillage Column: Missing values in the subvillage column are filled with the string "Unknown" since there is no definite most occurring value and probably the missing values are missing since it was not recorded.
- Scheme Name Column: The scheme_name column is dropped due to the high percentage of missing values of 48%

1.2 Dropping Irrelevant or Similar Columns

Redundant columns that either contain constant values or are highly similar to other columns are dropped to simplify the dataset and ensure the model performs well. The following changes are made:

- Columns with Constant Values: The recorded_by column, which has a constant value across all rows, is dropped as it does not contribute to the analysis.
- Geographic Classification Columns: region, region_code, and district_code columns are related, but region_code and district_code are redundant because the region column already provides sufficient geographic data. These two columns are dropped.
- Management Columns: The scheme_management and management columns are very similar, so the management column is dropped.
- Extraction Method Columns: The extraction_type, extraction_type_group, and extraction_type_class columns describe the extraction method for water points. Since extraction_type is the most detailed and useful, the other two columns are dropped.
- Water Point Type Columns: The waterpoint_type and waterpoint_type_group columns are highly similar, so waterpoint_type_group is dropped.
- Water Source Columns: The source and source_type columns describe the water source. Since the source contains sufficient detail, source_type is dropped.
- Payment Model Columns: The payment and payment_type columns are similar. Since payment_type is more standardized, payment is dropped.
- Water Quantity Columns: The quantity and quantity_group columns are similar, so quantity_group is dropped.
- Water Quality Columns: The water_quality and quality_group columns are similar, so quality_group is dropped.

1.3 Convert date_recorded to Datetime

The date_recorded column is converted to a datetime format. Additionally, the year, month, and day are extracted as separate columns, and the original date_recorded column is dropped.

1.4 Convert Categorical Column Values to Title Case

This step ensures consistency by applying title case to all string values in categorical columns.

1.5 Standardize Column Names to Title Case

The column names are converted to title case for consistency across the dataset.

1.6 Remove Trailing/Leading Whitespace

The strip_whitespace function is applied to all string columns (both object and category types) to remove any leading or trailing spaces, ensuring clean and uniform data.

2. Encoding Categorical Columns

2.1 Encoding Categorical Predictors

Categorical columns are encoded to numerical formats to make them suitable for machine learning models. Two encoding techniques are used:

- Frequency Encoding: This technique replaces each category with the proportion of rows that belong to that category. It is applied to categorical columns with high cardinality (many unique categories) to avoid creating too many new features.
- Label Encoding: For columns with lower cardinality (fewer unique categories), label encoding assigns a unique integer to each category.

2.2 Encoding the Target Column (Status_Group)

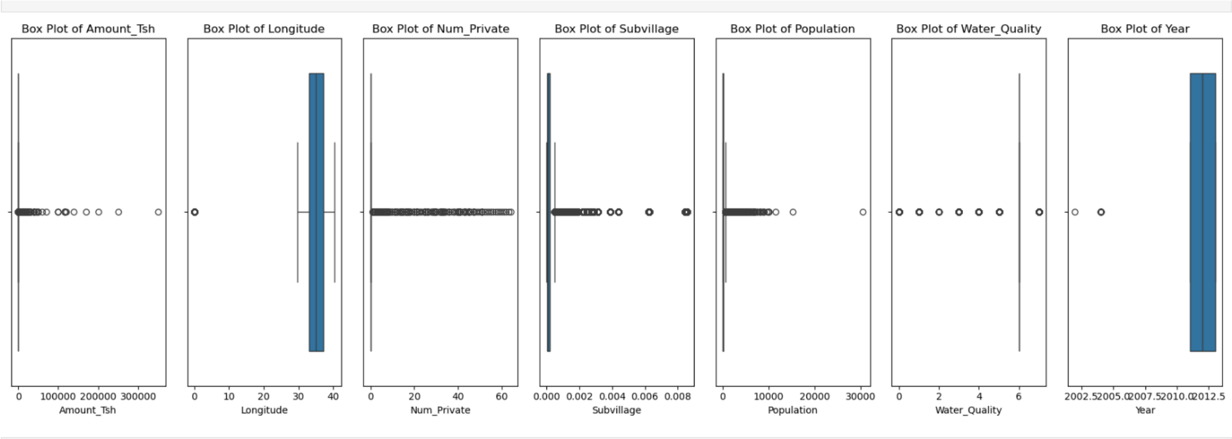
The target column Status_Group is originally multi-class (3 classes). To simplify for binary classification, the two classes Non Functional and Functional Needs Repair are combined into a new class called Needs Repair. The Ordinal Encoding method is then applied to convert this into binary (0 for 'Functional' and 1 for 'Needs Repair').

3. Converting Booleans to Numeric

The Boolean columns (e.g., permit, public_meeting) need to be converted to numeric values (1 for True and 0 for False). This transformation is necessary because machine learning models require numerical inputs.

4. Outliers Detection and Treatment

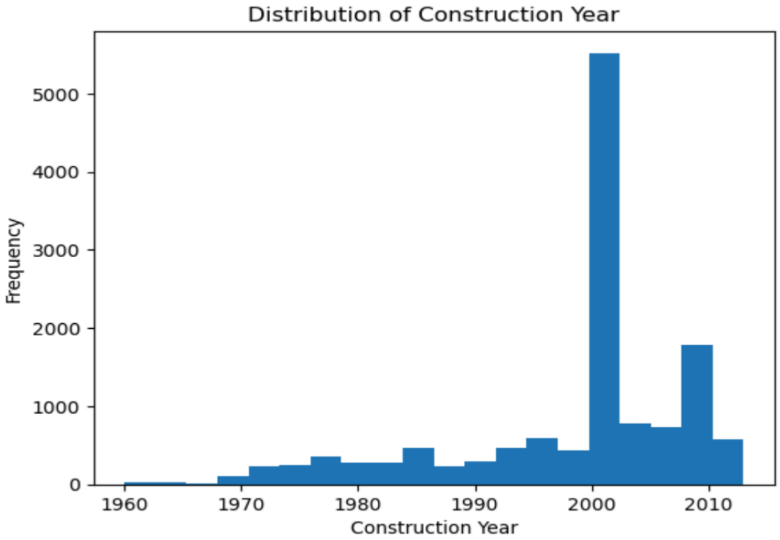
Outliers are detected and treated using the Interquartile Range (IQR) method. The following columns are checked for outliers: Amount_Tsh, Longitude, Num_Private, Subvillage, Population, Water_Quality, and Year. However, the interquartiles used were 95% and 5% to ensure not much data is lost when removing the outliers.



Visual representation of the columns with outliers before removing them using box plots. After removing the outliers, a loss of approximately 10.4% of the data occurs. This is a standard procedure to mitigate the influence of extreme values on the model.

5. Handling Abnormal Zeros in Construction_Year

The Construction_Year column contains abnormal zeros that are unrealistic (a construction year of zero). These entries are removed to ensure the data is valid and meaningful and is replaced with the median

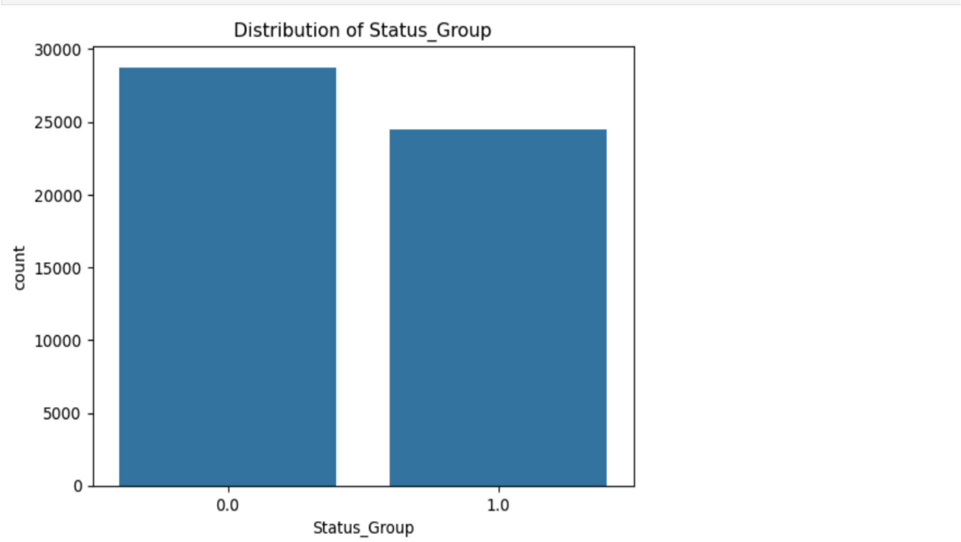


The histogram above shows the distribution of the Construction_Year values, excluding any entries where the year was incorrectly recorded as zero. This provides insight into the actual range of well construction years, helping to understand the age distribution of the wells.

6. Exploratory Data Analysis (EDA)

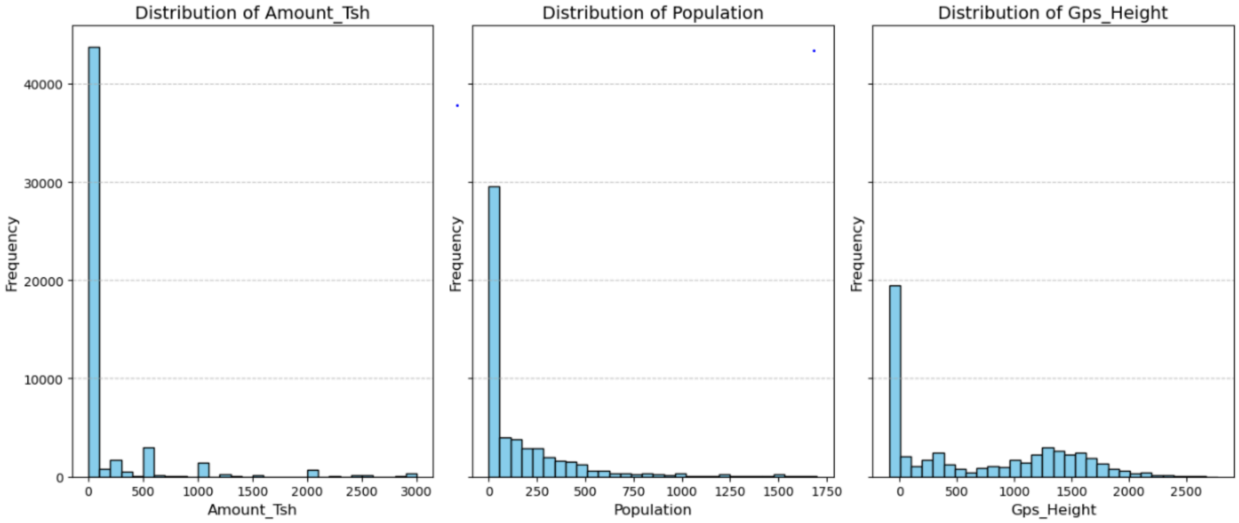
6.1 Univariate Analysis

- 6.1.1 Counterplot for the Target Variable: It is used to visualize the distribution of values in the Status_Group target variable.



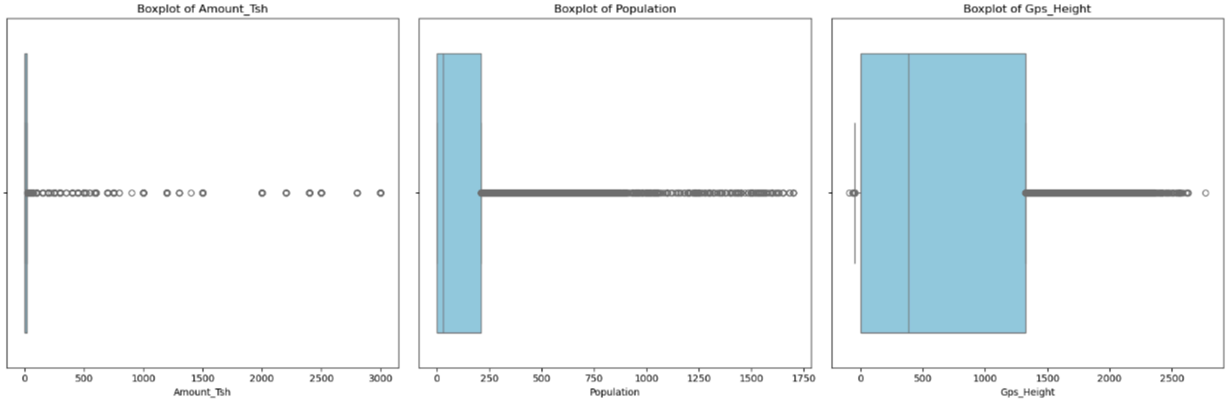
The distribution of the classes is slightly imbalanced, and techniques such as oversampling, undersampling, or class weighting may be needed to solve the imbalance before the modeling phase.

- 6.1.2 Distribution of Amount_Tsh, Gps_Height, and Population: The distributions of these columns are skewed, even after outlier removal. These columns may require scaling (e.g., Min-Max scaling or standardscaler) before modeling.



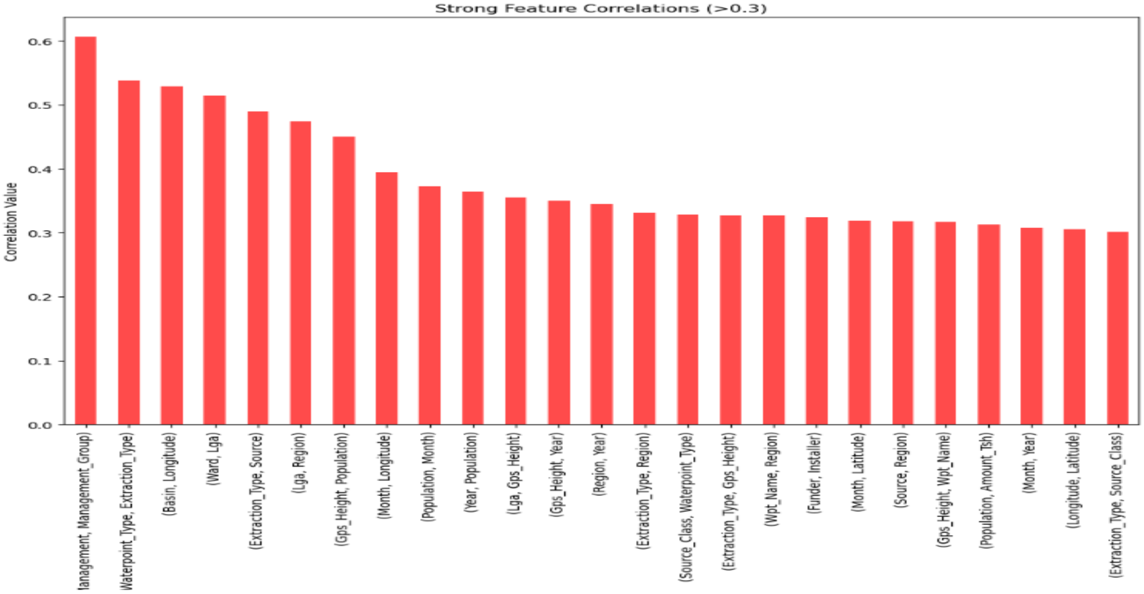
-

- 6.1.3 Outliers in Gps_Height, Amounts_Tsh, and Population: Though outliers had been already been removed, but not entirely therefore scaling will be required to ensure the impact of the outliers is minimized.

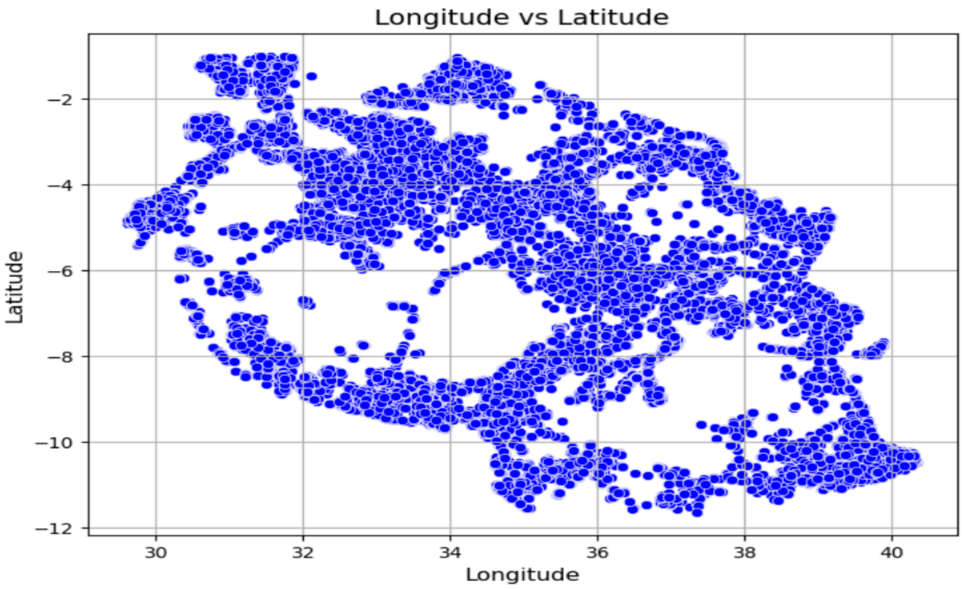


6.2 Bivariate Analysis

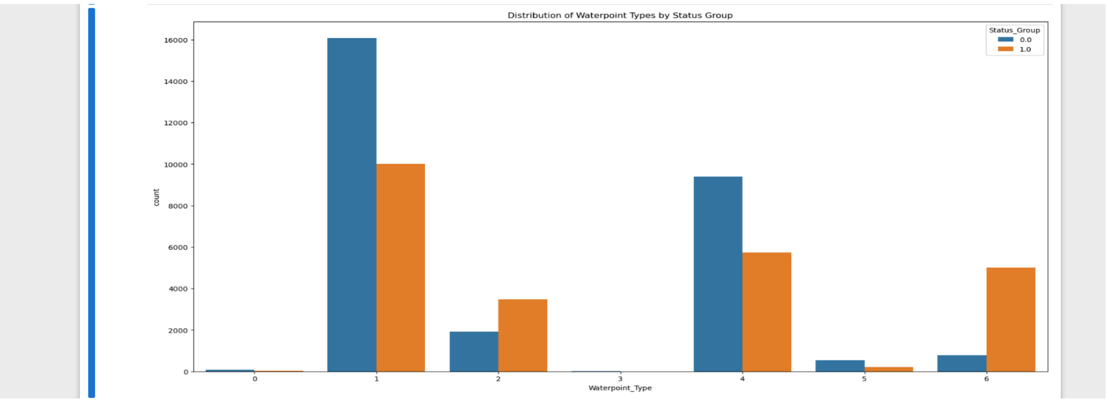
- 6.2.1 Pairwise Correlation Values Using Barplot Since the columns were many: It is used to examine the correlation between different features eliminating features with the very same values.



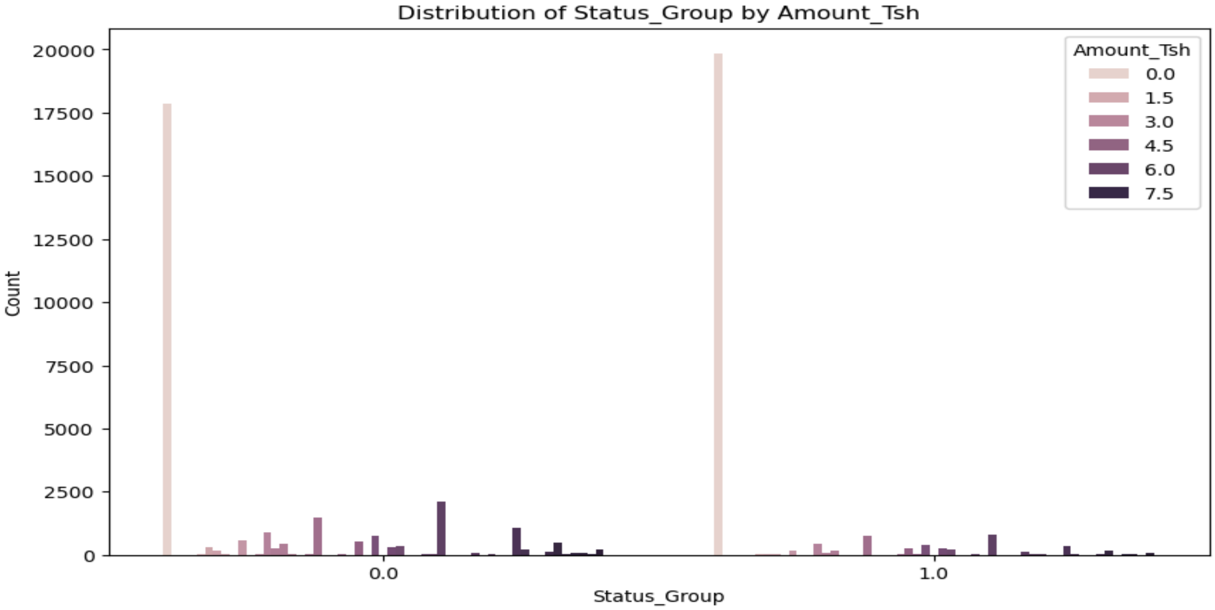
- 6.2.2 Relationship Between Longitude and Latitude Using Scatter Plot: A scatter plot of Longitude and Latitude helps reveal any geographic patterns or clusters, which could inform regional analysis for further investigation.



- 6.2.3 Distribution for Waterpoint_Type Based on Status_Group: A count plot or bar plot is created to visualize the distribution of Waterpoint_Type based on the target variable (Status_Group). The most common water point types, such as Communal Standpipe and Communal Standpipe Multiple, are found in both functional and non-functional wells.

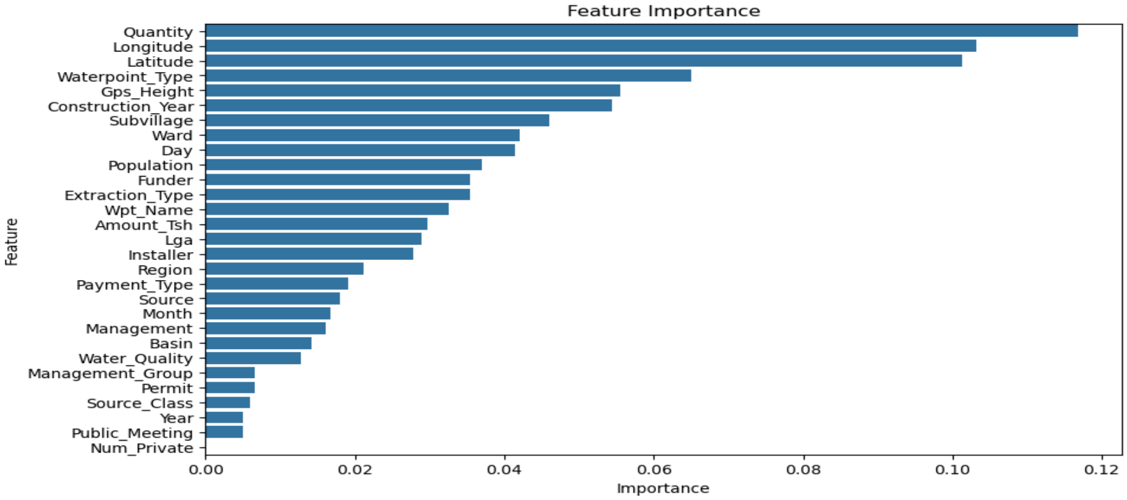


- 6.2.4 Counterplot to visualize how 'Amount_Tsh' is distributed by 'Status_Group'
This plot helps to observe the distribution of construction costs across these two conditions, potentially revealing trends such as whether wells in need of repair tend to have higher or lower construction costs compared to functional wells.



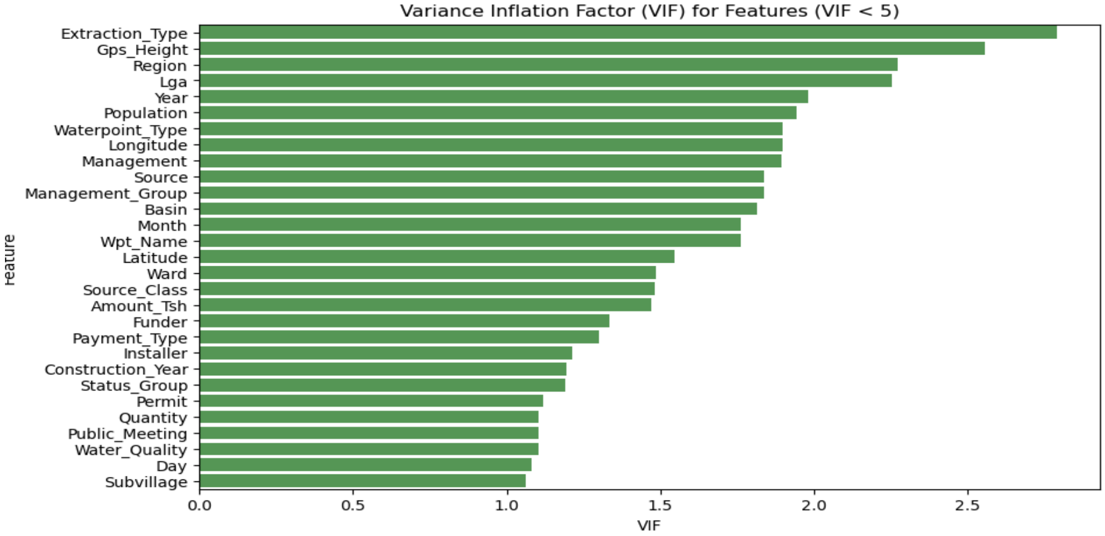
Even though there are high counts of the wells with amounts recorded zeros, generally there is a trend of more expensive wells being functional than the cheaper ones.

- 6.2.5 Feature Importance using Random Forest classifiers: Feature importance helps identify the most influential features in the dataset, providing insights into which factors most significantly affect the model's predictions.



6.3 Multivariate Analysis

- 6.3.1 Checking Multicollinearity through VIF (Variance Inflation Factor): The VIF for most features is between 1 and 2, indicating low multicollinearity. This is ideal for model training, as features are not highly correlated with each other.

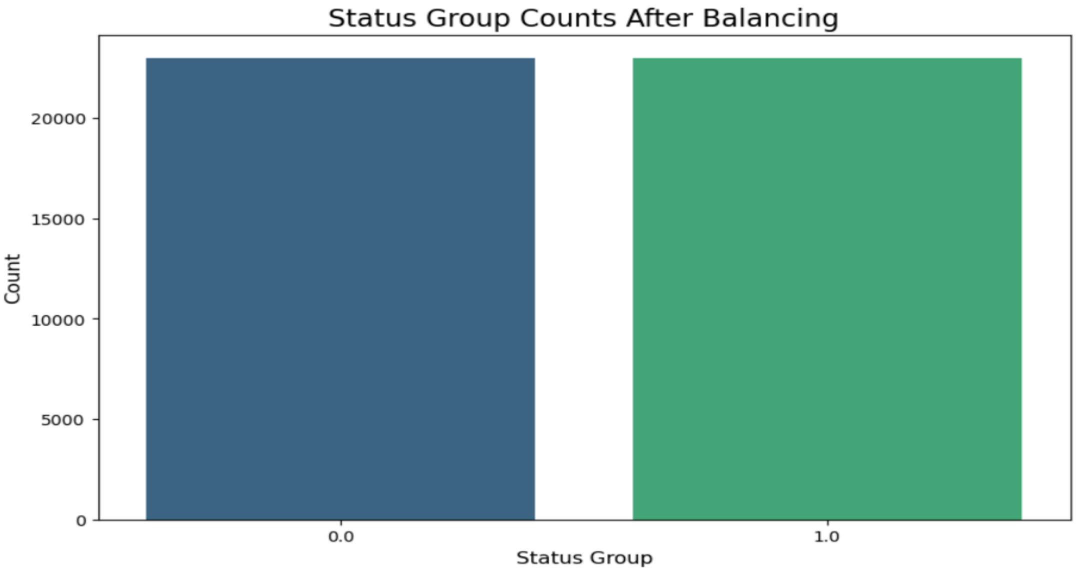


7. Feature Engineering

interaction terms were created by multiplying features such as Population and Management_Group to capture their combined effect, and Years_Since_Construction was calculated to represent the age of the waterpoint. Additionally, Construction_Age_Group was created by binning the Years_Since_Construction into categorical age groups, and this categorical feature was encoded numerically using label encoding for use in machine learning models.

8. Solving Class Imbalance

To address class imbalance in the dataset, the training data is split further into train and test sets. Techniques to handle class imbalance was SMOTE(Synthetic Minority Oversampling Technique) is a technique used to address class imbalance in datasets. It works by generating synthetic samples for the minority class rather than duplicating existing samples.



The target variable is now balanced, which is a crucial step before modeling. This process reduces the model's bias toward predicting the majority class, improving its ability to accurately classify both the functional and the needs repair categories. Addressing class imbalance ensures that the model is not overly influenced by the more frequent class, leading to a fairer and more reliable prediction of well conditions.

9. Standardizing the Data Using StandardScaler

To ensure features with different scales (such as Amount_Tsh, Population, and Gps_Height) are comparable, the data is standardized using StandardScaler. This transformation makes the mean 0 and standard deviation 1, improving model performance. The data is then output to the folder called Preprocessed Data ready for modeling.

Modeling Stage

In this phase of the CRISP-DM process, three machine learning models were developed and evaluated to predict the target variable effectively. Below is a summary of the modeling approach and key findings for each model:

Model 1: Logistic Regression (Baseline Model)

- Description: A Logistic Regression model was selected as the baseline to establish a standard for comparison with other models. Logistic Regression is a simple, interpretable model commonly used for binary classification tasks.
- Training Process: The model was trained on the resampled training dataset to address potential class imbalance issues.
- Evaluation: Predictions were made on the training data to assess performance.
- Purpose: This model serves as a benchmark to evaluate the performance improvements achieved by more complex algorithms.

Model 2: Decision Tree Classifier (Untuned)

- Description: A Decision Tree Classifier was implemented as a step toward more flexible and interpretable models.
- Training Process: The model was trained on the same dataset as Model 1, cross validated the data but without any hyperparameter optimization.
- Evaluation: Predictions on the training data revealed the model's capability to capture nonlinear relationships.

- Purpose: The untuned model allows a direct comparison with the hyperparameter-tuned Decision Tree model to highlight the benefits of tuning.

Model 3: Decision Tree Classifier (Tuned with GridSearchCV)

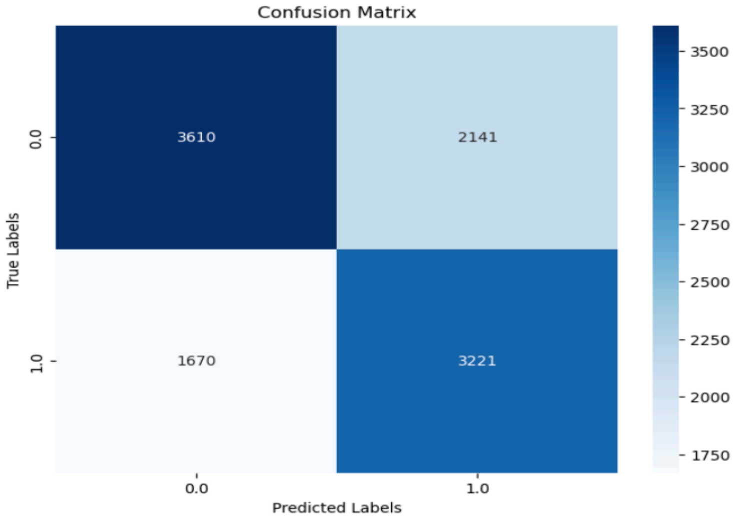
- Description: This model represents an optimized version of the Decision Tree Classifier. GridSearchCV was used to systematically explore a range of hyperparameters to identify the optimal combination.
- Hyperparameter Tuning:
  - Explored parameters include max_depth, min_samples_split, min_samples_leaf, max_features, and criterion.
  - The model underwent 5-fold cross-validation to ensure robust parameter selection.
- Best Parameters: The hyperparameter tuning process identified the optimal settings for the Decision Tree, which were applied to train the final model.
- Evaluation: Predictions on the training data confirmed the model's improved performance compared to both the baseline and untuned Decision Tree models.
- Purpose: This model demonstrates the impact of hyperparameter tuning in improving predictive performance.

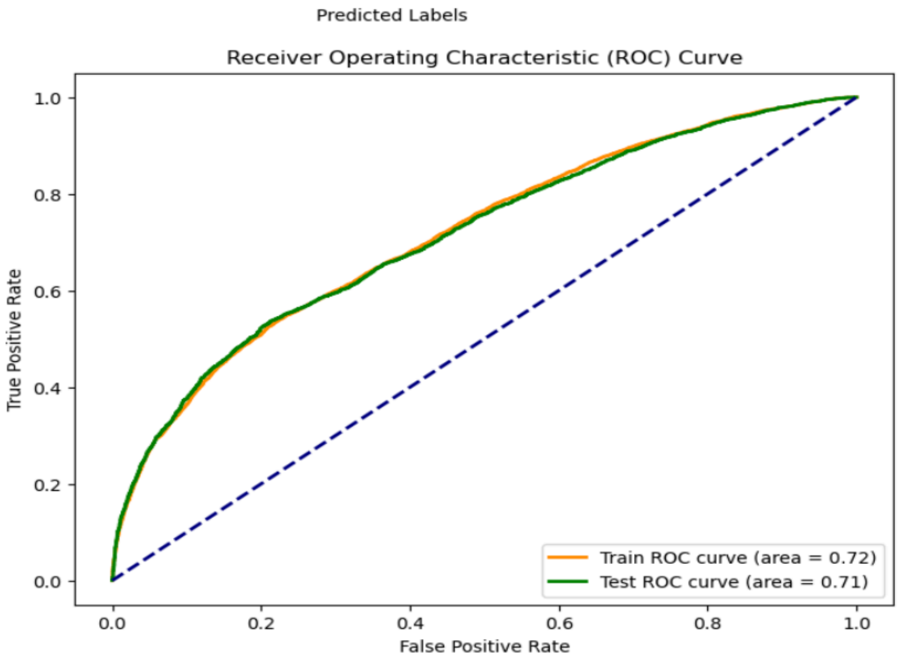
Evaluation Stage

The evaluation of the three models was conducted to assess their performance on both the training and test datasets. Below is a summary of the evaluation results, including key insights derived from the confusion matrices, classification reports, and overall accuracy.

Model 1: Logistic Regression (Baseline Model)

- Training Accuracy: 64.46%
  Functional Wells (0): Precision 64%, Recall 63%.
  Wells Needing Repair (1): Precision 64%, Recall 66%.
  Misclassification rate: ~35%.
- Test Accuracy: 64.19%
  Functional Wells (0): Precision 68%, Recall 63%.
  Wells Needing Repair (1): Precision 60%, Recall 66%.
  Misclassification rate: ~36%.
- Insights:
  The model performs slightly better in identifying functional wells (0) than those needing repair (1), with higher precision for class 0. Performance is fairly consistent across both training and test data, indicating minimal overfitting. However, precision for wells needing repair (1) drops on the test data, suggesting room for improvement. Optimizing the model, perhaps through feature tuning or regularization, could lead to more accurate predictions for wells needing repair.





Model 2: Untuned Decision Tree

- Training Accuracy: 100%

  Functional Wells (0): Precision 100%, Recall 100%.

  Wells Needing Repair (1): Precision 100%, Recall 100%.

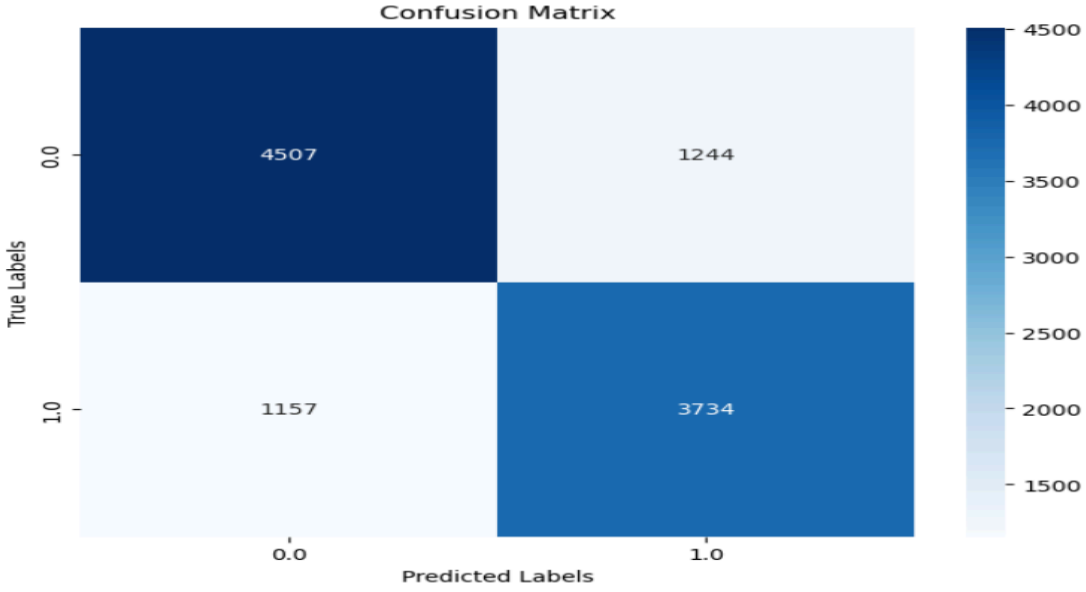
  Misclassification rate: 0%.
- Test Accuracy: 77.44%

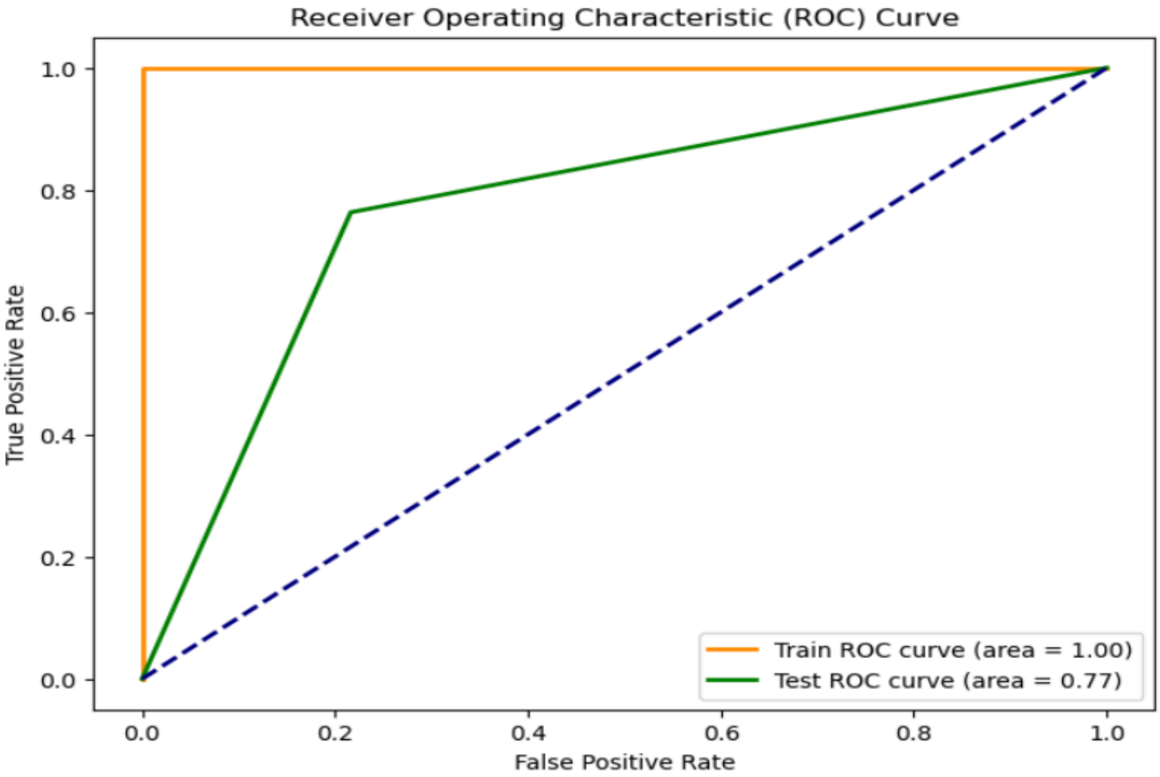
  Functional Wells (0): Precision 80%, Recall 78%.

  Wells Needing Repair (1): Precision 75%, Recall 76%.

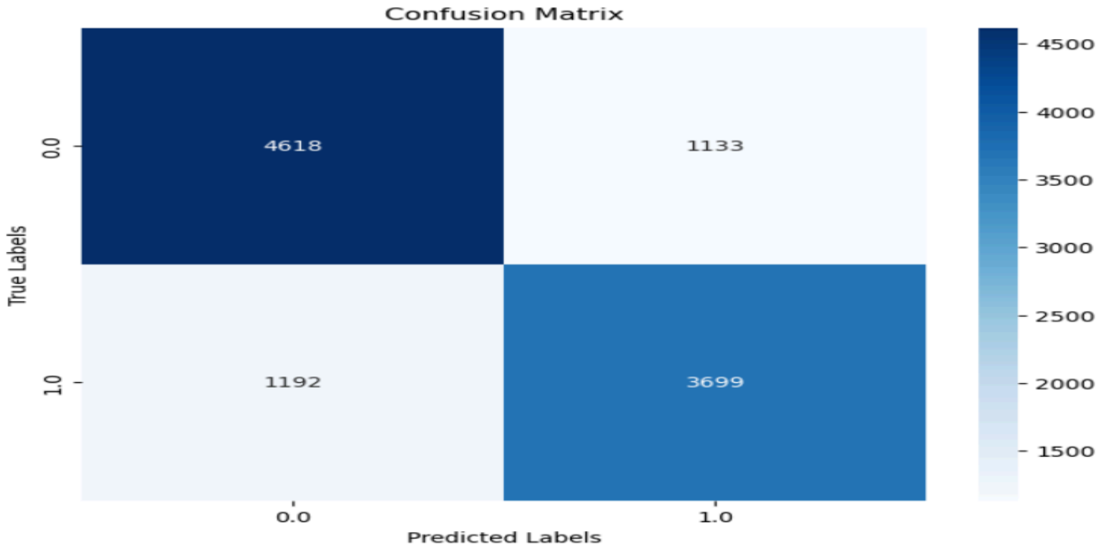
  Misclassification rate: ~23%.
- Insights:

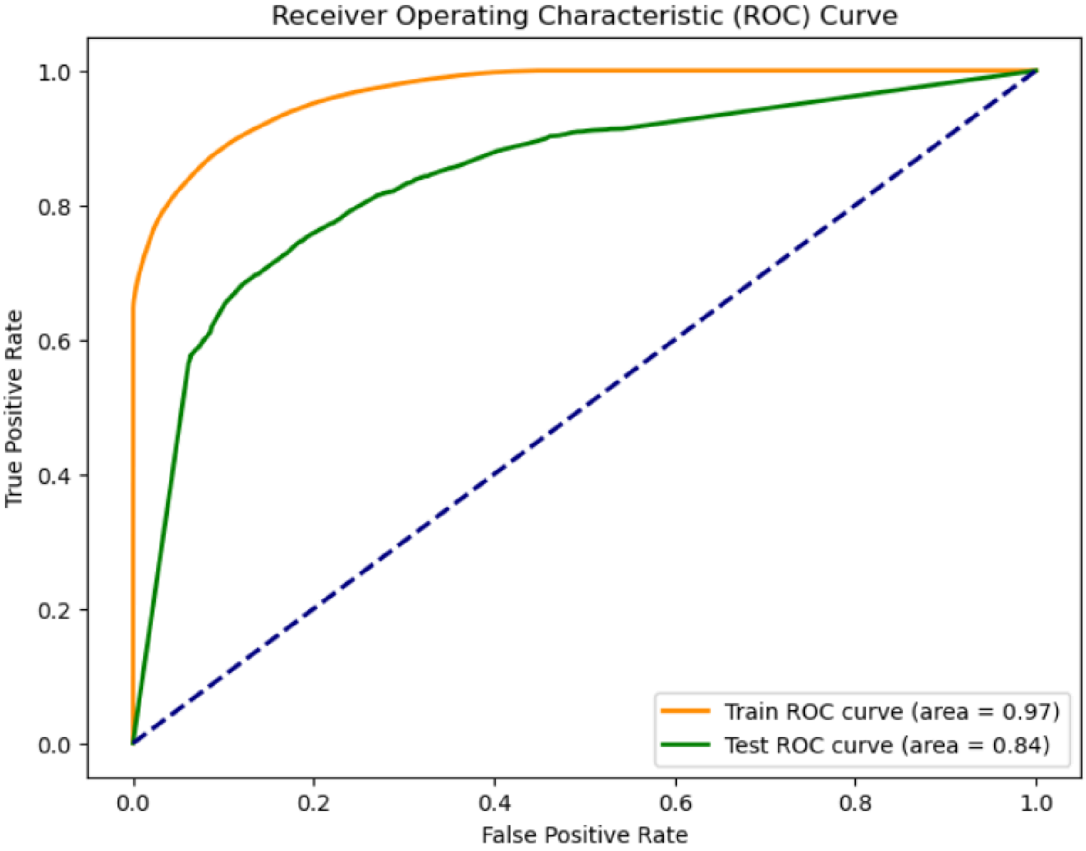
  The untuned decision tree model achieves perfect performance on the training data, with 100% precision and recall for both classes. However, there is a notable drop in performance on the test data, with accuracy at 77.44%, which indicates that the model has overfitted to the training data. It performs better at identifying functional wells (0) with higher precision and recall compared to wells needing repair (1), where performance is slightly lower. Further model tuning, such as adjusting hyperparameters or implementing regularization, would likely improve the generalization ability and balance the precision and recall across both classes.





Model 3: Tuned Decision Tree

- 



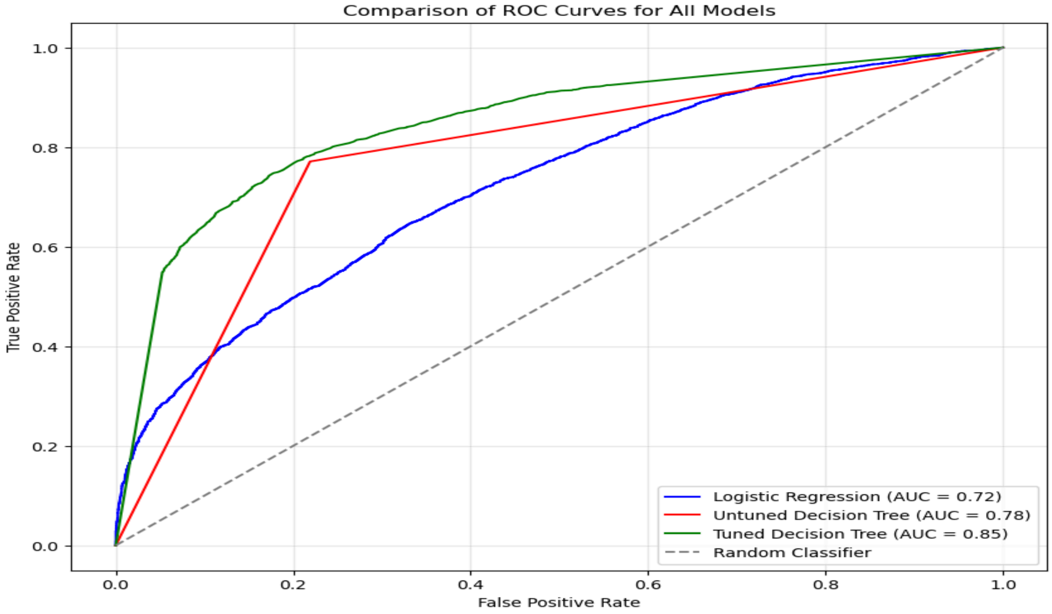
- Training Accuracy: 89.27%

- Functional Wells (0): Precision 88%, Recall 91%.
- Wells Needing Repair (1): Precision 91%, Recall 87%.
- Misclassification rate: ~10%.
-
- Test Accuracy: 78.15%
- Functional Wells (0): Precision 80%, Recall 80%.
- Wells Needing Repair (1): Precision 77%, Recall 76%.
- Misclassification rate: ~22%.

- Insights:

The tuned decision tree model performs well with high accuracy on both the training and test data, achieving a substantial reduction in misclassification compared to the untuned version. The model's performance on the training data is impressive, with high precision and recall for both classes. However, on the test data, the accuracy drops to 78.15%, reflecting some overfitting, though not as severe as the untuned model. The model performs slightly better at identifying functional wells (0), with precision and recall higher than for wells needing repair (1). Fine-tuning the model further could improve performance, especially in balancing precision and recall across both classes.

Conclusions



1. Model Performance:
    a. Model 1 (Logistic Regression): The model achieved moderate accuracy but faced challenges with false positives and false negatives. However, the lack of test labels for validation hindered a proper assessment of the model's performance on the test data.
    b. Model 2 (Untuned Decision Tree): The decision tree model overfitted the training data and performed poorly on the test set. The absence of test labels impacted the evaluation, making it difficult to assess its real-world effectiveness.
    c. Model 3 (Tuned Decision Tree): This model demonstrated an improvement in generalization over the untuned model, with better handling of class imbalance and overfitting. However, without test_labels, it was not possible to fully evaluate its performance in a test scenario.
2. Class Imbalance:

    Class imbalance was managed through techniques like SMOTE and class weights, which helped balance the influence of the two classes (functional wells vs. wells needing repair). Despite this, the lack of test labels undermined the ability to evaluate the model's performance accurately, particularly on the test set.

3. Test Labels Issue:

    The absence of test labels for the test set significantly impacted the ability to reliably assess model performance. To ensure robust and actionable insights, it is crucial that test labels are available for all model evaluations moving forward.

Recommendations

1. Feature Engineering:
    - Continue to refine and engineer features to better capture relevant patterns. However, make sure test_labels are available for comprehensive validation of how these new features influence model performance.
2. Additional Data Collection:
    - Ensure that future datasets include test_labels, which are essential for validating model predictions and evaluating performance metrics accurately.
3. Model Enhancements:
    - Explore more advanced models, such as Random Forests or Gradient Boosting Machines, to further improve predictive accuracy. Ensure proper validation by using datasets with test labels for robust evaluation.

4. Evaluation Improvements:
    - Going forward, ensure test_labels are available for all test sets. This will enable precise computation of key performance metrics (accuracy, precision, recall, etc.) and lead to better-informed decisions regarding model improvements.

Next Steps

1. Deployment:
    - Prepare to deploy the tuned decision tree model once test_labels are included in the testing phase. This will ensure that the model's performance is accurately evaluated and ready for real-world implementation.
2. Iterative Data Collection and Testing:
    - Update datasets with test_labels to ensure that all future models are validated against reliable test data. This will allow for ongoing model tuning and performance enhancement.
3. Future Exploration:
    - Explore ensemble methods such as Random Forest and Gradient Boosting to improve model performance. Ensure that test_labels are always included in future datasets to provide robust and consistent evaluation of model effectiveness.