Water Pumps

Capstone Project Two
Springboard - School of Data

David Clark
April, 2021

1. Introduction

Tanzania is a developing country and access to water is very important for the health of the population. For this reason, it is vital that all water pumps are properly working. Currently, the only way to monitor pump working status is by physically visiting the site. This is time consuming and costly. Therefore, a more intelligent solution to monitor water pump status is desirable.

In this project I addressed the following question: How can the government of Tanzania improve water pump maintenance by knowing the pump functional status in advance?

The data was provided by the Water District of Tanzania. It consisted of demographic, geographic, and maintenance data for each pump in the country. I defined a response variable as the functional status, which consisted of three class: *functional*, *functional needs repair*, and *non-functional*. Therefore, this was a classification problem.

The modeling objective of this project was to create a classification algorithm that optimized for a high recall value for the minority class. I was able to achieve a recall score of 0.80 on the minority class using a random forest model.

2. Approach

The data was acquired from Taarifa and the Tanzanian Ministry of Water. The dataset consisted of 59,400 rows and 39 columns. The data types were strings, numerical, boolean, and dates. The majority of features were categorical.

During the data wrangling phase I dealt with missing values, data and variable types, and duplicate values.

2.1.1 Missing Values

I handled missing values using two methods. I either removed them, or I imputed their value from additional information.

There were three columns that contained 50% of missing values. These were: scheme name, amount_tsh, and num_private. Therefore, I removed these columns from the dataset.

Over 30% of the rows in the dataset contained missing population values. In this project I chose to remove these rows. But, in a future project, I will consider methods to derive missing population values, such as using Tanzanian census data.

I also found that 3% of the rows contained missing geographical locations. But, after performing the data cleaning discussed above, these rows dropped out as well.

For additional columns in the dataset I was able to impute the missing values. In the case of the boolean columns *public_meeting* and *permit*, only 5% of the values were missing. The majority of the values in these columns were True. Thus, I chose to set the missing values to True.

There were four more columns with missing values. These included: *subvillage*, *funder*, *installer*, and *scheme_management*. All these columns contained the value None. Therefore, I chose to use this value to set any missing values observed in these columns. In addition, the columns *funder* and *installer* include the string value 0, which I interpreted as designating a missing value. Therefore, I chose to set all 0 values to None.

2.1.2 Duplicate Values

I did observe duplicated location information and recording date. But, these dropped out after I performed data cleaning, as discussed above.

2.1.3 Redundant Features

The majority of features were categorical. Several of these features referred to related quantities. For example, *management* and *management group*, referred to the same quantity. In other cases, certain features were subsets of other features, such as *extraction_type_class*, which was a subset of *extraction_type*. For these features, I selected the variable that was most general in order to keep the model simple. Even so, there were a few more granular columns that I decided to keep as I believed they would be more informative: *source_type*, waterpoint_type, and *region*.

2.1.4 Uninformative Columns

Three of the features did not provide any useful information for the project objective. These were: *funder*, *installer*, and *wpt_name*. So, I decided to drop these columns.

2.2 Story Telling and Inferential Statistics

2.2.1 Numerical Variables

After completing data cleaning, I was left with the following numerical variables: *gps_height*, *longitude*, *latitude*, and *population*. I began exploring these variables by plotting empirical cumulative distribution functions for each variable (see Figure 1.) As can be seen in Figure 1, the plots for gps_height, longitude, and latitude follow a more-or-less linear trend. This indicates a uniform distribution in values for each.

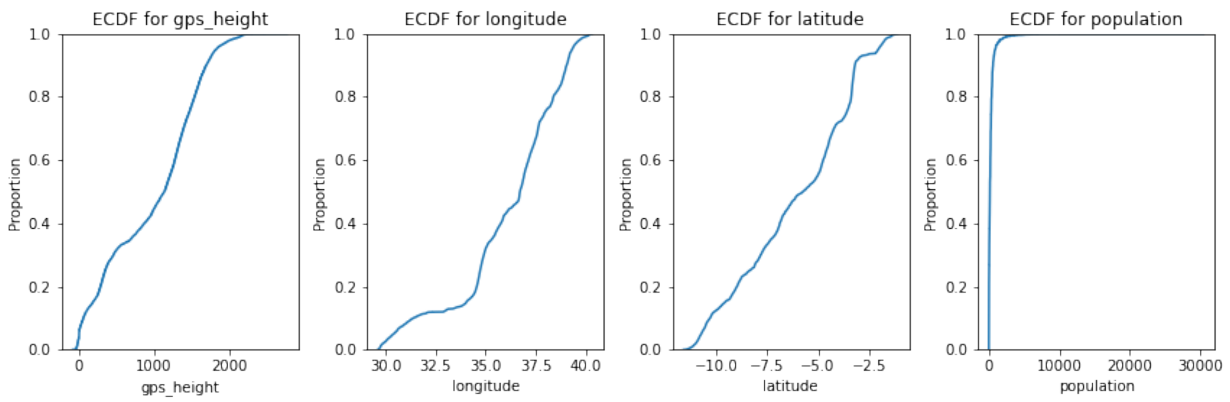


Figure 1: Empirical cumulative distribution functions for each of the numerical variables.

On the other hand, the ECDF for population shows a sharp rise in values, followed by a plateau. This indicates many small values for population, but only a few values for large populations. Due to the heavily skewed distribution in population, a better way to analyze this variable would be to converting into log space. I performed this transformation and show the updated ECDFs in Figure 2.

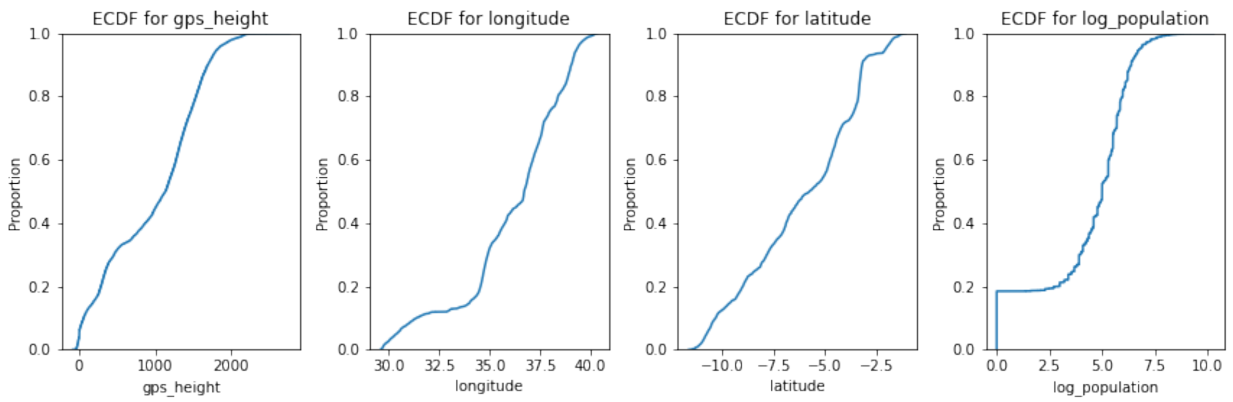


Figure 2: Same plot as Figure 1, but population has now been converted into log space.

Another way to visualize relationships between numerical variables is using a pair plot (see Figure 3). Notice, there is no relationship between *log_population* and the additional variables. In fact, there is no obvious linear relationship between any of the variables. However, the plots comparing longitude and latitude show a nice outline of the country. This indicates good geographical coverage for this dataset.

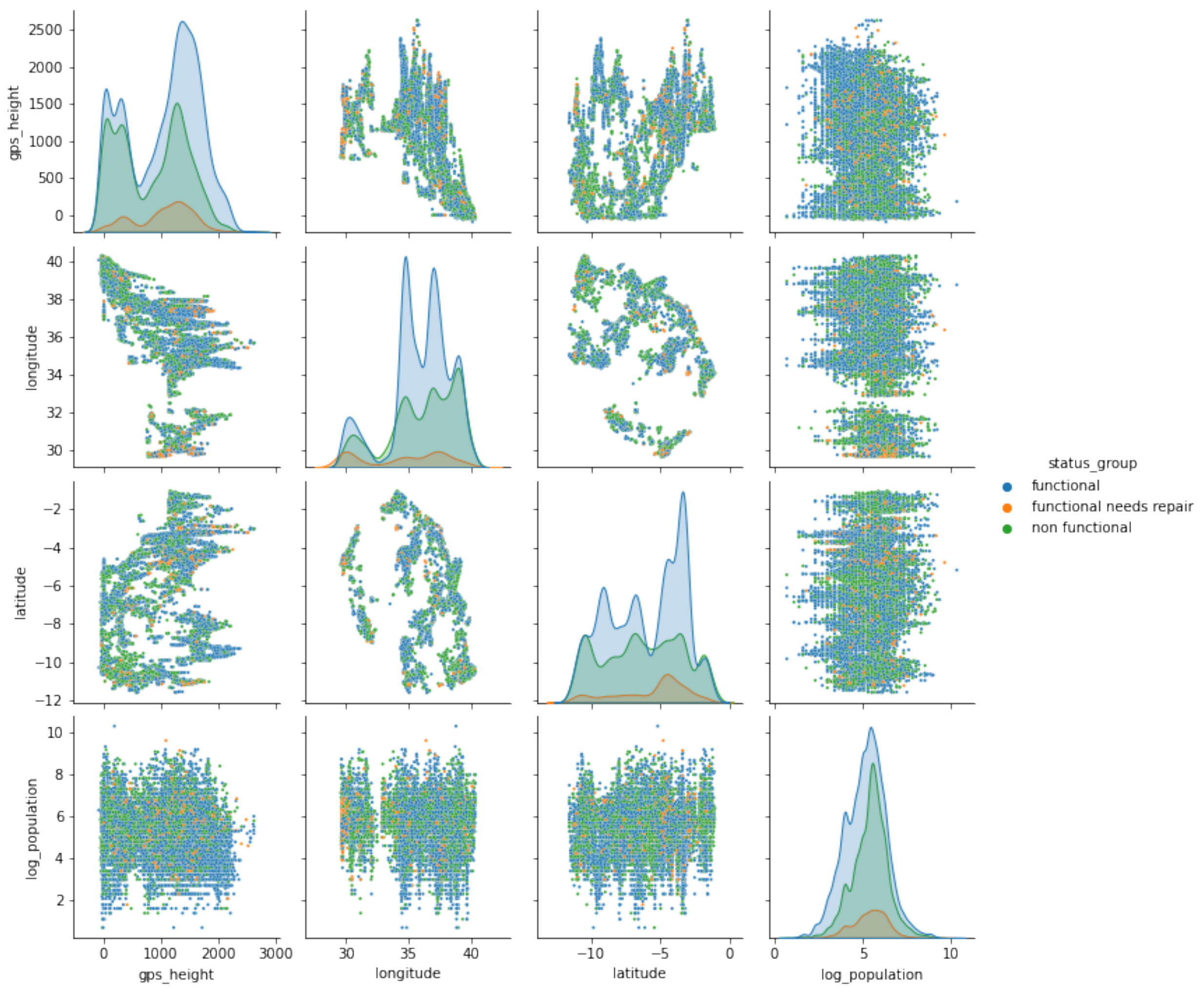


Figure 3: Pair plot comparing relationships between each numerical variable.

I explored the relationship between geographical location and population in more detail in Figure 4. In this figure I also included the pump functional status. There are a couple of things to point out. First, functional pumps seem to concentrate along the middle of the country, while non-functional pumps are to the East and West. Second, there is no obvious trend in population size and functional status.

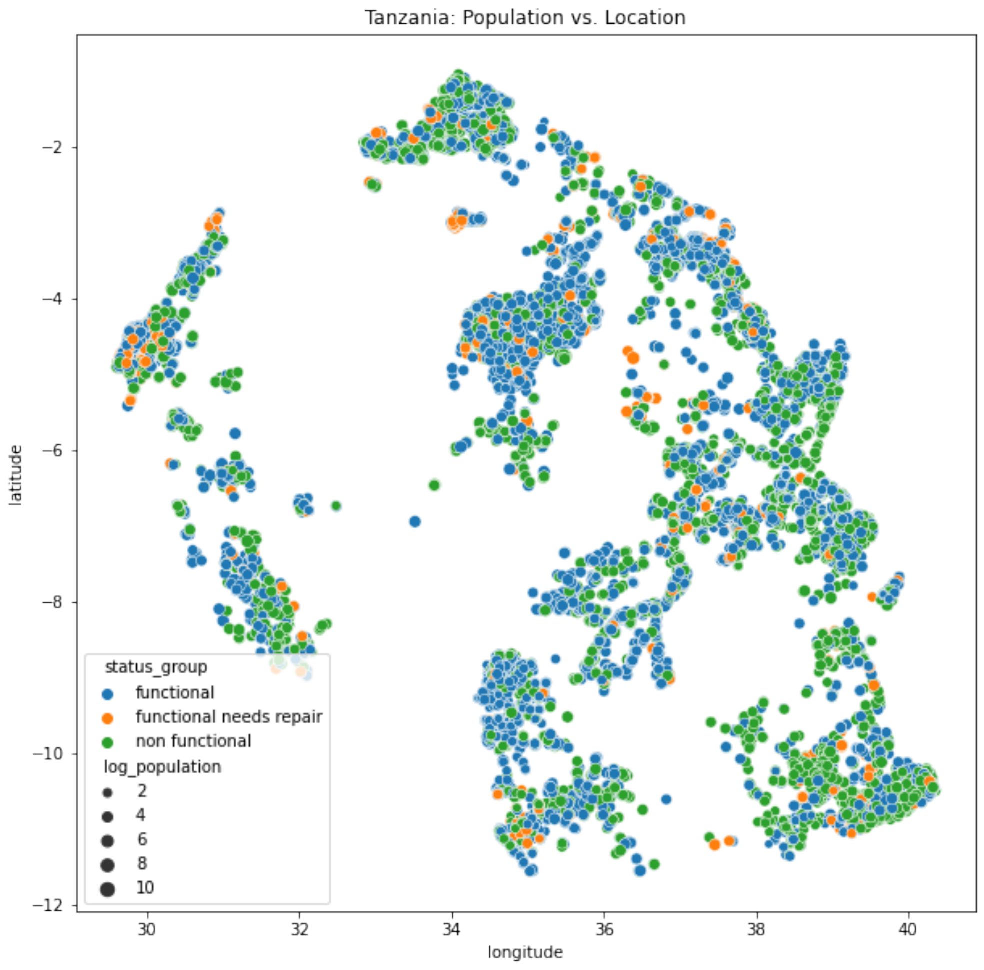


Figure 5: Geographical location for each pump in Tanzania. The colors designate the pump functional status, while the size of the dots correspond to the population size.

Lastly, I compared KDEs for each numerical variable in Figure 5. These plots provide more detail than the ECDFs above. Notice that gps_height and longitude appear double peaked, with a smaller peak at lower values. Latitude appears more uniform, with a large peak at larger values. The variable log_population shows a normal distribution.

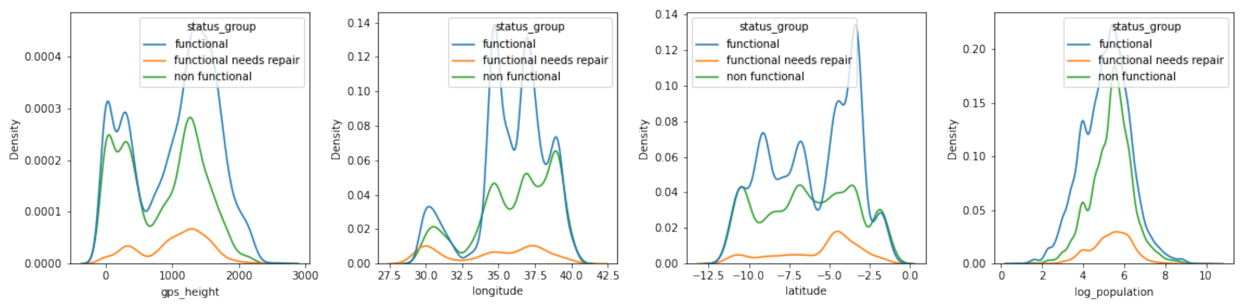


Figure 5: KDE plots for each numerical variable, broken down by *status_group*.

2.2.2 Categorical Variables

I compared the categorical variables using stacked bar plots, labeled by the response variable *status_group* (See Figure 6). Each bar represents a different category type for each categorical variable. Notice the wide range in categories. The variable *construction_year* shows the largest number of categories, 54, while *public_meeting* and *permit* have boolean categories, True or False.

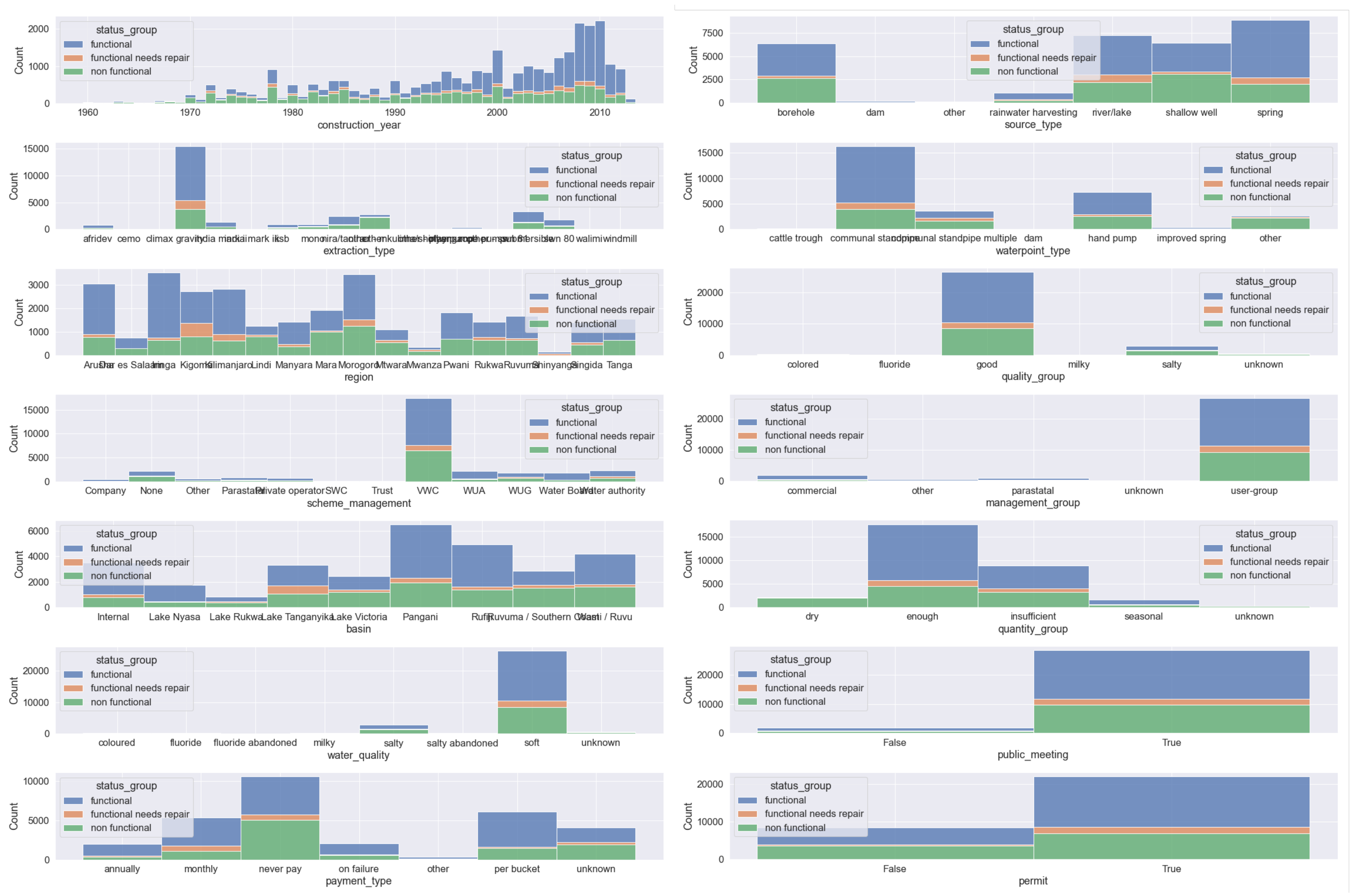


Figure 6: Bar plots for each categorical variable in the data set. The x-axis labels correspond to each category in the corresponding categorical variable. The colors designate the water pump functional status.

For most of the variables there does not appear to be any relationship between category type and *status_group*. The only variable that does show a relationship is *payment*. For those pumps lacking payment, there was a larger fraction of pumps that were not functional.

Before progressing to the modeling phase, I converted each categorical variable to a one-hot vector. This produced a wide dataset with 30,423 rows and 232 columns.

2.3 Baseline Model

I split the data into a training set and a test set. I chose to reserve 30% for testing, and the rest of the data for training. Given the imbalance in the response variable, I used stratification when performing the train-test split to ensure equal representation of classes between each data set.

I scaled the numerical data to values between 0 and 1. Given that the categorical variables are one-hot-encoded, I chose this scaling strategy so that all variables would be in the same numerical range.

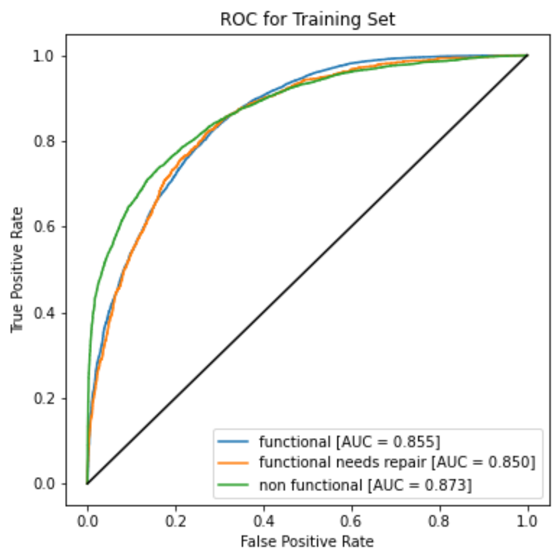
The goal of this project is to classify water pump status. Therefore, I chose to use a logistic regression model as a base line. I found the best model used a one-vs-rest strategy, meaning that each class was evaluated with-respect-to the other two classes. This produced the following results when fit to the training and test sets:

Table 1: Baseline Model Results

	Train			Test		
	precision	**recall**	**support**	**precision**	**recall**	**support**
functional	0.76	0.91	12482	0.77	0.91	5349
functional needs repair	0.63	0.15	1505	0.63	0.15	645
non functional	0.78	0.65	7309	0.79	0.66	3133

Given the similarity in metrics between the training and test sets, I determined that overfitting was not a problem. Hence, I did not considering performing any regularization.

Finally, the performance of the model on the test set can be analyzed visually using an ROC plot (see Figure 7). The steep curves and large values of AUC for each class indicates that the model is doing a reasonable job of predicting each class. Nevertheless, there is still room for improvement.



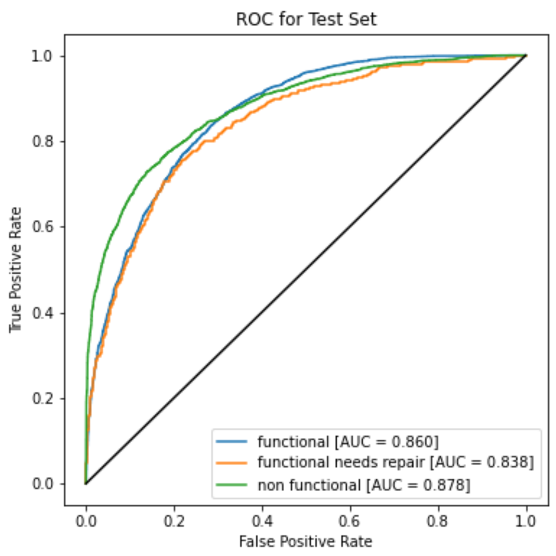


Figure 7: ROC plots for the training set (left) and test set (right). Colors indicate the ROC for each of three water pump status groups.

After speaking with the client, I determined that they would like to optimize the model for recall for the minority classes. This will reduce type II error. Meaning there would be less errors classifying pumps as working when in fact they are not working.

In the next section, I will go into a detailed exploration of models and discuss the final results.

2.4 Extended Modeling

While the baseline model produced satisfactory results, there was room for improvement. One main issue for this dataset was the imbalanced classes. I decided to address this by transforming the data in two ways, over sampling and under sampling. The transformations were only done on the training data. The model fit was then evaluated on the test set.

I chose to evaluate three models, logistic regression with regularization, random forest, and XGBoost. Random forest and XGBoost are great at performing classification tasks, especially with class imbalanced datasets, which is why I chose them.

For each model I identified the optimal values of parameters by carrying out a random grid search. I defined the search space using suggested values from the documentation and blog posts. I then created a final model for each algorithm using the best hyperparameter values. This yielded six models, three algorithms each of which was run on an oversampled and under sampled data set. In the table below, I summarize the hyperparameter values I used for each of the final models.

Table 2: Hyperparameters for Best Fit Models

Logistic Regresion							
sampling	**num_classes**	**penalty**	**C**				
under	3	l2	2.2				
over	3	l1	2.2				
Random Forest							
sampling	**num_classes**	**n_estimators**	**min_samples_split**	**min_samples_ leaf**	**max_features**	**max_depth**	**bootstrap**
under	3	1600	3	1	auto	90	TRUE
over	3	1600	3	1	auto	90	TRUE
XGBoost							
sampling	**num_classes**	**min_child_weight**	**max_depth**	**gamma**	**n_estimators**		
under	3	1	5	1	1000		
over	3	1	5	0	1000		

In the case of random forest and XGBoost, one of the hyperparameters was the number of trees. After identifying the optimal tree count, I doubled it for the final model, to improved model performance.

2.5 Findings

The below table summarizes the results of fitting six models to the training data and comparing it with the baseline model. The table includes the metrics precision and recall. The final column in the table shows the class count for each of the three classes in the response variable.

Table 3: Final Results

	base_line		logreg_over		logreg_under		rf_over		rf_under		xgb_over		xgb_under		
	precision	recall	precision	recall	precision	recall	precision	recall	precision	recall	precision	recall	precision	recall	class count
functional	0.768	0.911	0.846	0.652	0.839	0.638	0.835	0.867	0.851	0.649	0.825	0.867	0.853	0.646	5349
functional needs repair	0.632	0.152	0.226	0.741	0.218	0.726	0.467	0.416	0.260	0.757	0.463	0.388	0.236	0.758	645
non functional	0.788	0.659	0.733	0.674	0.718	0.668	0.813	0.778	0.716	0.725	0.803	0.759	0.738	0.706	3133

I found that all models perform better in recall than the baseline model for the minority class, functional needs repair. The highest recall score of 0.758 was found for the XGBoost model using under sampling. The minority class in the test set consisted of 645 values.

The best recall score the second, non-majority class, non-functional, was 0.777. This was found using a random forest model with over sampling. The class count in the test set for non-functional was 3133.

I investigated whether the recall score could be improved by reframing the project as a binary classification problem. I combined the two minority classes non-functional and functional needs repair into the single class faulty. The test set now had 3778 values for the faulty class and 5349 for the functional class. In Table 4 I compare the model results using two classes.

Table 4: Final Results using Two Classes

	logreg_over		logreg_under		rf_over		rf_under		xgb_over		xgb_under		
	precision	recall	precision	recall	precision	recall	precision	recall	precision	recall	precision	recall	class count
faulty	0.732	0.737	0.727	0.742	0.806	0.764	0.765	0.799	0.795	0.747	0.752	0.785	3778
functional	0.813	0.809	0.815	0.803	0.839	0.870	0.853	0.827	0.829	0.864	0.843	0.817	5349

While I did see an improvement in recall for the minority class, it was not significantly better as compared with the three-class problem above. The highest recall score for the minority class was 0.80, found using a random forest classifier with under sampling. This means that the Tanzanian government can be 80% sure they can identify faulty pumps, while 20% of those predicted to be faulty are actually functional.

4. Conclusions and Future Work

I was able to achieve a recall score of 0.80 on the minority class (faulty). The best algorithm was random forest, using an under sampled training set to balance the classes. Also, the two minority classes, non functional and functional needs repair were combined into a single class, faulty.

The choice to optimize on recall was to reduce Type II error. With regards to the results, this means that 20% of the pumps predicted to be faulty will actually be functional. This is a much better way to err since it confirms faulty pumps are working, but doesn't miss faulty pumps that were labeled as working.

While I have achieved an acceptable model for the Tanzanian government, there are a few areas I would like to improve on during updated versions of the project. This includes:

- Try additional sampling methods to counter class imbalance, aside from the two I tried here.
- Perform a more detailed exploration of hyperparameter values. A more granular search could lead to more refined models with better results.
- Try a different train/test split. A smaller split could give the models more data to train on, which could possibly improve the predictions.

5. Recommendation for the Client

Here are the following recommendations I would like to make to the Tanzanian government:

- Run this model on a quarterly basis to identify those pumps that are predicted to be faulty. The maintenance staff can then visit these pumps to make repairs or confirm that they are working.
- Gather missing population and GPS height information for the pumps. Almost 30% of these features were missing values. Including it could improve the model performance. Once the information has been gathered, I would be happy to return and create a new version of the model with improved performance.

6. Consulted Resources

- *Fine-tuning XGBoost in Python like a boss* [https://towardsdatascience.com/fine-tuning-xgboost-in-python-like-a-boss-b4543ed8b1e]
- *Ensemble methods (Scikit-Learn)* [https://scikit-learn.org/stable/modules/ensemble.html]

- *Complete Guide to Parameter Tuning in XGBoost with codes in Python* [https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/]