Pump it Up

Data Mining approach to tackle Tanzanian Water Crisis

by

Sakshi Sharma

A report presented for the degree of
Masters of Science in Data Engineering

JACOBS
UNIVERSITY

Prof. Dr. Adalbert F. X. Wilhelm
Jacobs University Bremen
Prof. Dr. Stefan Kettemann
Jacobs University Bremen

Contents

1 Executive Summary

1.1 Task Outline and Dataset

The task of this project is to predict which water pumps throughout Tanzania are functional, which of those need repairs, and which do not work at all. The prediction is based on a number of variables about what kind of pump is operating, when it was installed, and how is it managed. We are comparing the performance of a couple of classifiers .We used Naive Bayes classifier as baseline performance and then we performed random forest followed by the gradient boosting classifier.The ultimate goal would be to find the best classifier and suggest improvements. Another aim of this project was to be able to get familiar with popular machine learning libraries like scikit-learn, Gradient boosting and matplotlib.

We are using the data from Taarifa and Tanzanian Ministry of Water to classify the water pumps. The data was collected using handheld sensor, paper reports, and user feedback via cellular phones. The dataset has features such as the location of the pump, water quality, source type, extraction technique used, and population demographics of pump location.The data for this comes from the Taarifa waterpoints dashboard, which aggregates data from the Tanzania Ministry of Water.The training set has 59,401 rows and 40 features including an output column. The output column specifies the status of the water pump in the category of functional, functional needs repairs, or non functional. Out of the 40 features in the data we have:

- 31 categorical variables
- 7 numerical variables
- 2 date variable.

1.2 Method of approach

With regards to the objective, data mining is performed on the raw dataset in order to create a comprehensive and tidy dataset.The processed dataset is used to build a baseline model, using the technique of Naive Bayes classifier. Naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature. Even with such strong assumptions, the baseline model is able to give us a maximum testing accuracy of 71%. To further improve the testing accuracy, the dataset is used to train ensemble models. First a Random forest classifier is built which has a testing accuracy of 78% and then further improved by dealing with the issue of class imbalance which gave the accuracy of 76%. Finally a Gradient Boosting tree ensemble is built which gave the testing accuracy of 79%.

2 Introduction

Water is critical to a country's development, as it is not only used in agriculture but also for industrial development. Though Tanzania has access to a lot of water, the country still faces the dilemmas of many African countries where many areas have no reliable access to portable water. This project uses the dataset of water pumps in Tanzania to predict the operating condition of each water point. By finding which water pumps that are either functional, functional need repair, or non functional, the Tanzanian Ministry of water aims to improve the maintenance operations of the water pumps and make sure that clean, potable water is available to communities across Tanzania. Data driven challenges hosted this competition to help Tanzania ministry with classifying the water pumps.[1]

The dataset for this project is taken from this challenge.The project is focused on exploration and primarily uses visualization and statistical techniques to develop comprehensive insights from the available data. Data visualisations is used to convey these observations because graphical representations are an effective way to provide a clear understanding of the data, due to the predominantly visual nature of human perceptual abilities. The software used for this project is Python. All the machine learning packages used in the project are present in the open-source sklearn package available for python. Following this idea, initially, this report will describe the existing data in order to familiarise readers with the available variables. Subsequently, this report will address the data preprocessing methods used to create enriched tidy data-sets for analysis. Furthermore, the report will then explore the process and choices of various classifiers to know how well we can predict the conditions of waterpump.

3 The tanzania waterpump Dataset

The dataset is the collection of all the water pumps across Tanzania.The source of this data is collected by the government officials of Tanzania from the residents who complained about the waterpump via cellular phones or paper reports.A description about the water point that are reported by residents of a particular region in Tanzania, are then displayed over the taarifa dashboard.The below image represents the taarifa dashboard:

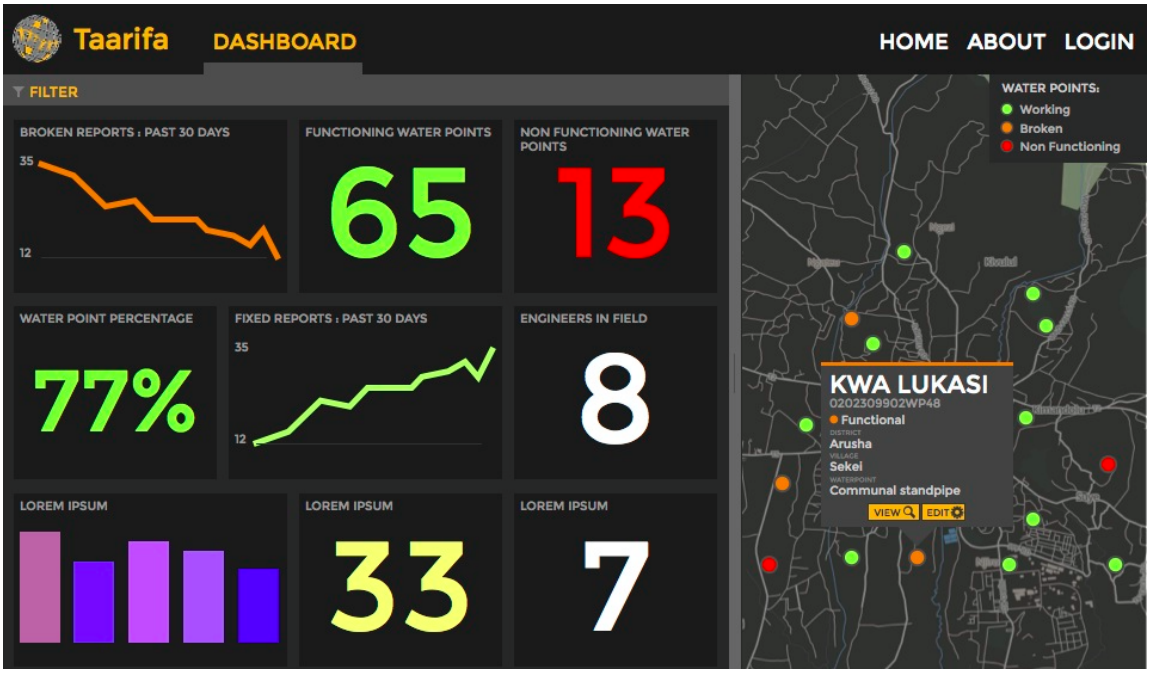


Figure 1

The dataset provide by the driven data challenge was already spilt into training values and training labels.We have no clue about the criteria for this spilt. The Tanzania waterpump data obtained for this project included two files:

- Training set values.csv
- Training set labels.csv

The Training set values data file contains the following set of information about the waterpoints. The training set has 59,401 rows and 40 features. Out of these 40 features in our dataset we haveThe below figure shows first few columns of features:

	amount_tsh	funder	installer	basin	population	public_meeting	scheme_management	permit	construction_year	extraction_type_class	payment_type
0	6000.0	other	other	Lake Nyasa	109	True	vwc	False	90s	gravity	annually
1	0.0	other	other	Lake Victoria	280	Unknown	other	True	10s	gravity	never pay
2	25.0	other	other	Pangani	250	True	vwc	True	00s	gravity	per bucket
3	0.0	Unicef	other	Ruvuma / Southern Coast	58	True	vwc	True	80s	submersible	never pay
4	0.0	other	other	Lake Victoria	0	True	other	True	unknown	gravity	never pay

Figure 2: Snapshot of the original dataset with the first five rows shown

Further details about the column are given in appendix.
Training set labels includes the distribution of Labels The labels in this dataset are simple. There are three possible values:

- functional - the waterpoint is operational and there are no repairs needed
- functional needs repair - the waterpoint is operational, but needs repairs
- non functional - the waterpoint is not operational

4 Dataset Exploration and Pre-processing

While exploring the raw dataset we faced many challenges.Here are the risks and challenges we faced in our project:

4.1 Deal with columns containing null values

The dataset contain features like funders,installers, scheme management that have large number of null values. So, to deal with such features for example 'Funder'. I created a function to reduce the amount of dummy columns by keeping the top 10 funder values and set the rest to 'other'.Then, added a column name 'Status_group_values' it is column which contain class labels and used this column as a pivot table to check the differences.Further,calculated the percentage of funder per status group. Below figure shows the percentage of well by functionality type for top 10 funders.

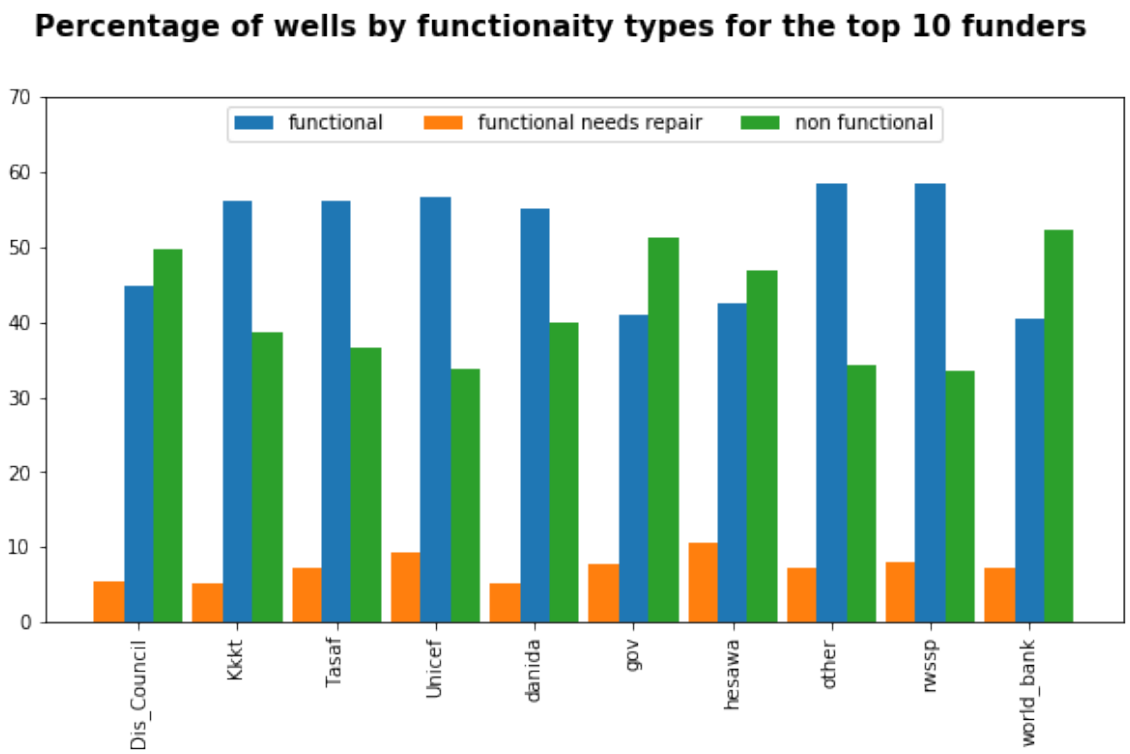


Figure 3: Percentage of funder per status group.

From the figure we can see that there are large number of non-functional pumps which are funded by dis_council, goverment and world_bank in comparision with the other funders.It can also be seen that the precentage of pumps that needs repair is much less than the pumps that are functional or non- functional.

There were column which have only two values: True and false for example Public meeting had 3334 null values.To deal with columns like this we filled the null values with 'unknown'.The figure shown below represents the number of counts of true, false and unknown for the feature permit.

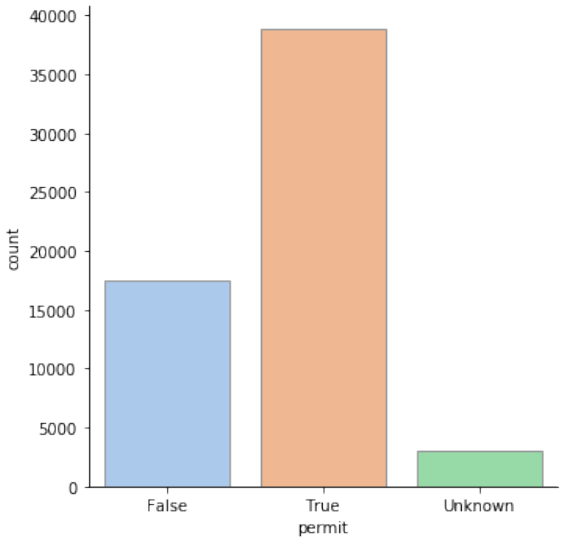


Figure 4: Number of counts of public meeting feature .

4.2 Repeated Values

Our dataset contains many features that contain similar representation of data presented in different grains. The group of features of (extraction_type, extraction_type_group, extraction_type_class), (payment, payment_type), (water_quality, quality_group), (source, source_class), (subvillage, region, region_code, district_code, lga, ward), and (waterpoint_type, waterpoint_type_group) all contain similar representation of data in different grains. Hence, we risk overfitting our data during training by including all the features in our analysis. We tried to avoid this risk by identifying features in each group that contained the finer grain which held more information in the analysis.

4.3 Adding a new feature for the waterpump

The name of the columns is operational_years. Here I create a feature that indicates how many years the pump has been around since construction to the date the record was recorded.The idea being that more recently recorded pumps might be more likely to be functional than non-functional. The figure below represents the number of wells based on their functionality for 25 operational years.

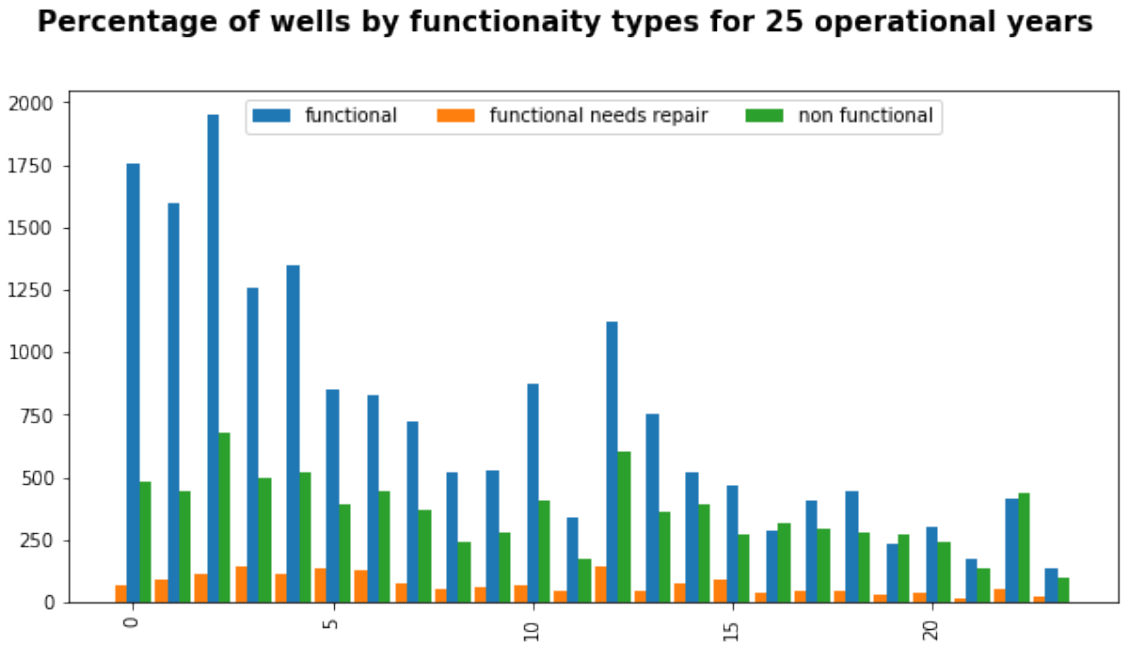


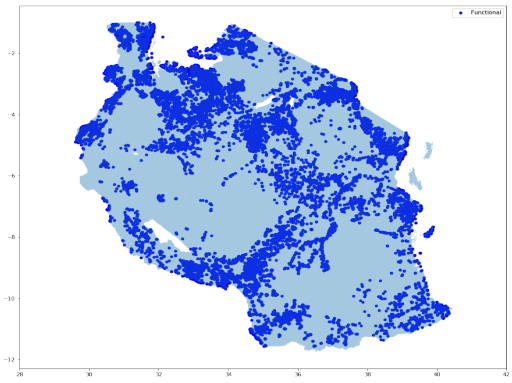
Figure 5: The number of wells based on their functionality for 25 operational years

From the figure it is clearly seen that the pumps that were recently recorded are more likely to be functional than the pumps whose operational year is more than 10 years.We can see a another peak in between 10 to 15 years the reason behind this peak is that there were more number of pumps built in this year in comparison with others.

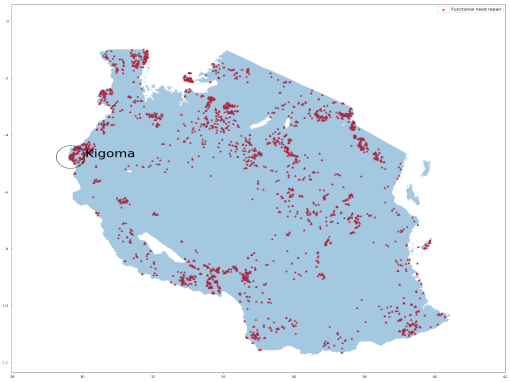
4.4 Analyzing columns with longitude and latitudes

Since our dataset contained latitude and longitude for each water pumps, we wanted to identify any regions that contain high concentration of functional; functional needs repair, and non-functional water pumps. Below is our geographic map of Tanzania that contains all the data points of water pumps and the regional concentration of functional, functional needs repair, and non-functional water pumps.

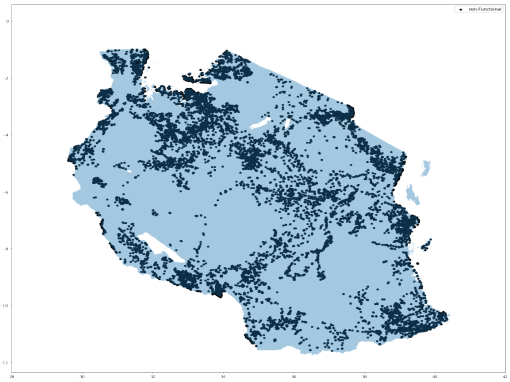
(a) Geographical map of tanzania



(b) Functional water pump map



(c) Functional needs repair water pump map



(d) Non-Functional water pump map

Though most of the water pumps status is concentrated in the same regions most of the time, we can notice two distinct patterns from the maps. Figures b, c and d suggest, the concentration of water pumps that are functional and those that are non functional are more spread out throughout Tanzania. The water pumps that are functional but need repair are more concentrated among the rural areas, especially areas near protected national reserves and forests. We can see these plots are in line with the reported statistics by WHO/UNICEF, only 44% of the rural areas in Tanzania get proper supply of water compared to 77.9% of the urban areas.
The second pattern we see from figure c and d is the the high concentration of water pumps that are functional needs repair and non-functional in the Kigoma region. Kigoma city is one of the developing cities in Tanzania that get the poorest supply of water in Tanzania.
The third pattern is if you look at the geographical map of tanzania it can be seen that most of the water pumps are located near rivers/lakes.

4.5 Deal with categorical Features

Our dataset contains many categorical features. There are 31 categorical features in our dataset. These features include funders, installer, water point names, sub village, ward, and scheme name. These features contain over 1000 different values that causes huge memory overhead when running algorithms such as Random Forest on our machines. We used dimension reduction techniques

such as Multiple Correspondence Analysis(MCA)[2] Figure below shows the MCA plot on all the categorical features and labels present in our dataset

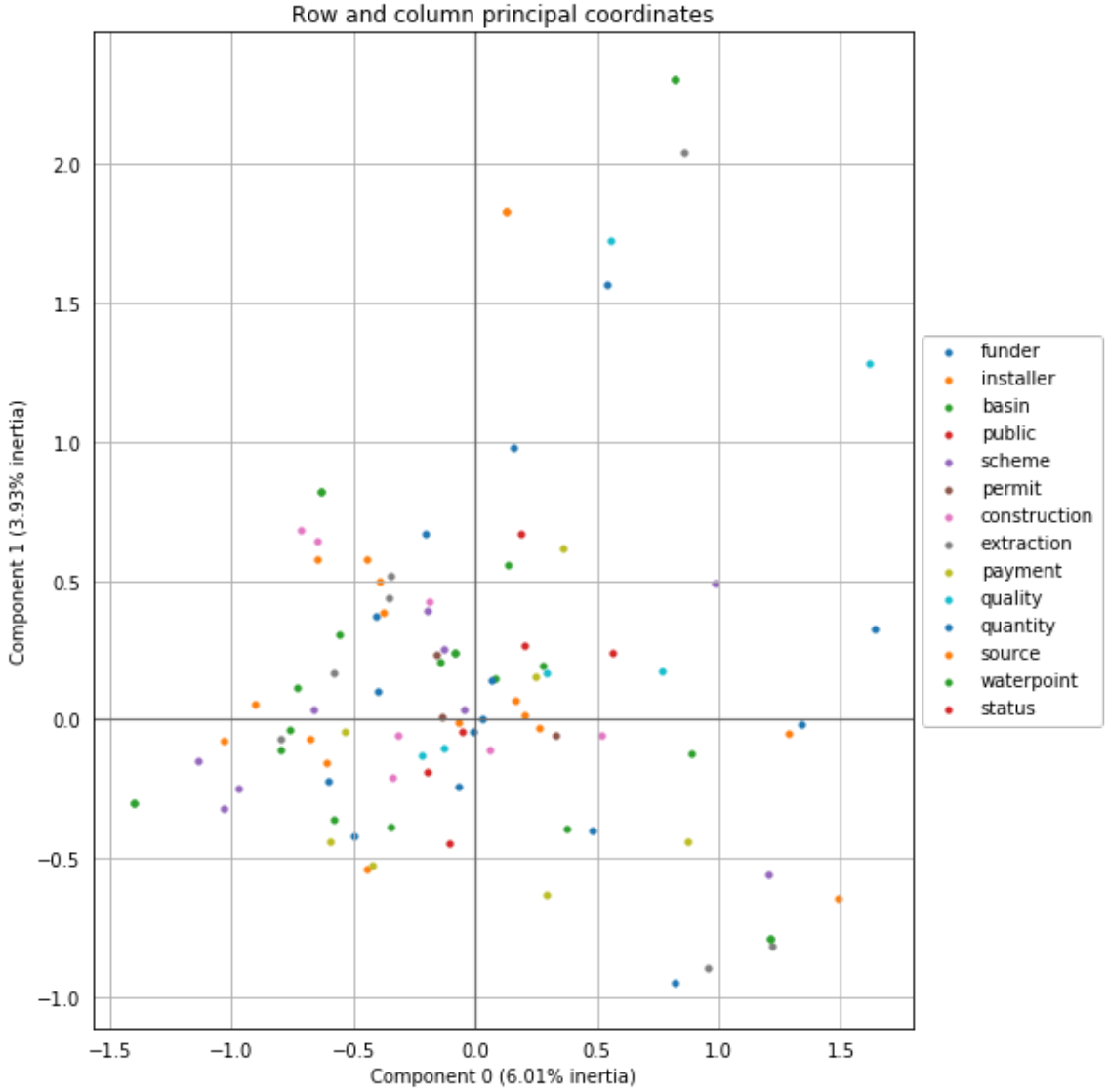


Figure 7: Multiple Correspondance Analysis(MCA) of categorical features between functional, functional needs repair and non-functional responses

From the above plot it is really difficult to understand which features and their categories, have strong relationship with the categories in the labels. So, to simplify the feature selection we used chi-square test.In Chi-Square goodness of fit test, categorical data is divided into intervals. Then the numbers of points that fall into the interval are compared, with the expected numbers of points

in each interval. Figure below shows best 25 categorical features that are best fit for classifying between 3 labels.

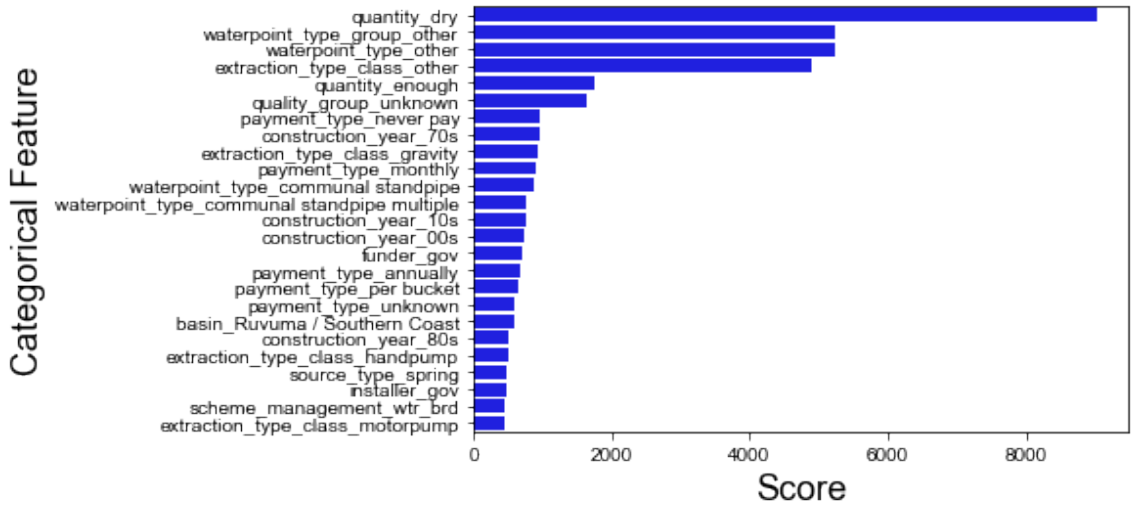


Figure 8: chi2 scores of best 25 categorical feature categories

The way to interpret the above chi2 scores is that categorical features with the highest values for the chi-squared stat indicate higher relevance and importance in predicting labels and may be included in a predictive model development.For this dataset we used chi-square test for to understand which categorical features is important.

4.6 Class Imbalance

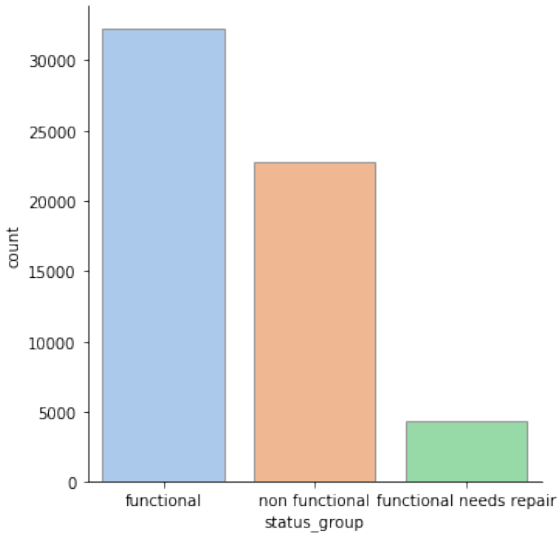


Figure 9: Class Imbalance

Our dataset has severe class imbalance, with 32,259 data points for functional water pumps, 4,317 data points for functional water pumps but needs repair, and 22,824 data points for non functional water pumps as seen in Figure above. The goal is to reduce the false positives i.e. misclassification of functional or nonfunctional pumps into functional needs repair. To mitigate the issue of class imbalance, we used sampling methods like oversampling the under-represented class. One of the most popular oversampling methods is SMOTE(Synthetic Minority Oversampling Technique). This method creates synthetic samples of the under-represented class by finding nearest neighbors and making minor random perturbations.[3] while training the random forest algorithm in this project, the training has been done once on the imbalanced training dataset and once on the oversampled training dataset, through which we also compare the performance of the model when dealing with imbalanced dataset.

4.7 Analyze a few other attributes with respect to the class labels

Plotting some of the relevant predictors determined from chi-square test will helps us understand how the class labels are distributed among the categories of the corresponding attributes.

- Quantity v/s Status Group

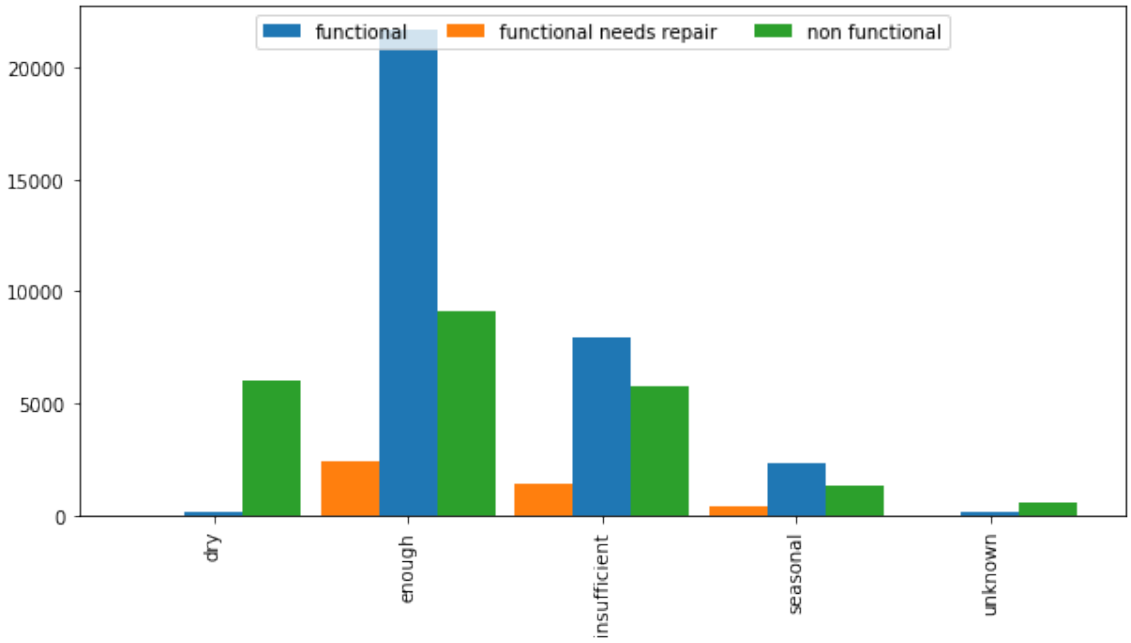


Figure 10: Quantity of the water pump with respect to operating condition of a water point

From this plot we can deduce that most of the dry pumps are not functional and the pumps with enough water are functional, while the classes are more evenly distributed across the other variables.

- Quality_group v/s Status Group

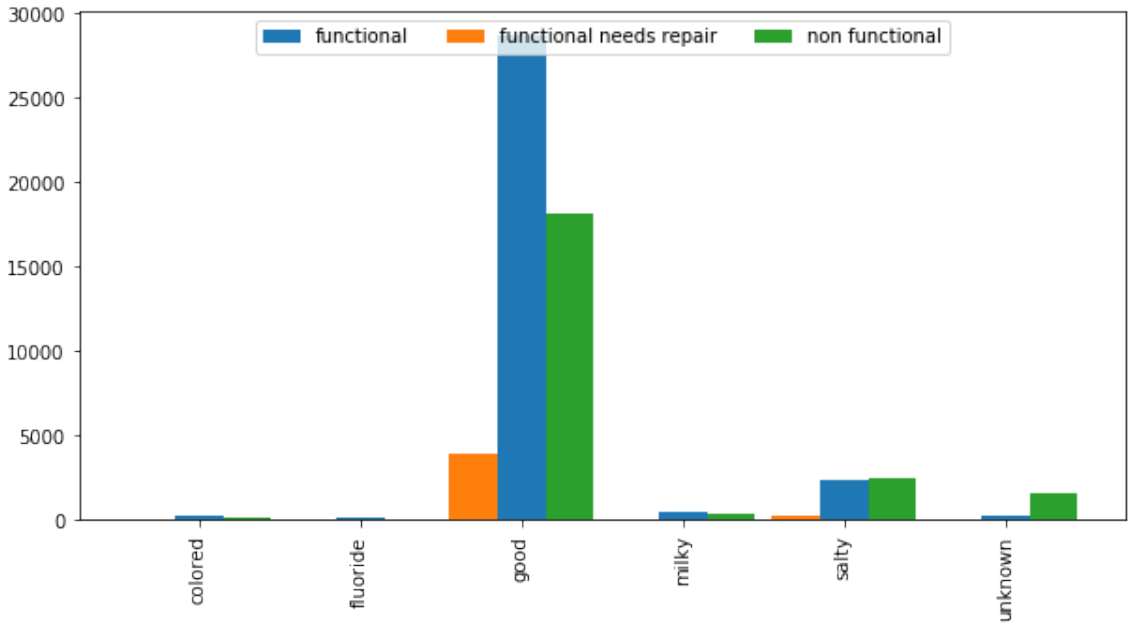


Figure 11: Quality_group of the water pump with respect to status group

Could clearly see that water is good then there is a very high probability of the water point being functional, while if it is salty then there are almost equal probability of functional and non-functional. Most pumps with an unknown quality_group are non functional, so are the colored ones.

- Source_type v/s Status Group

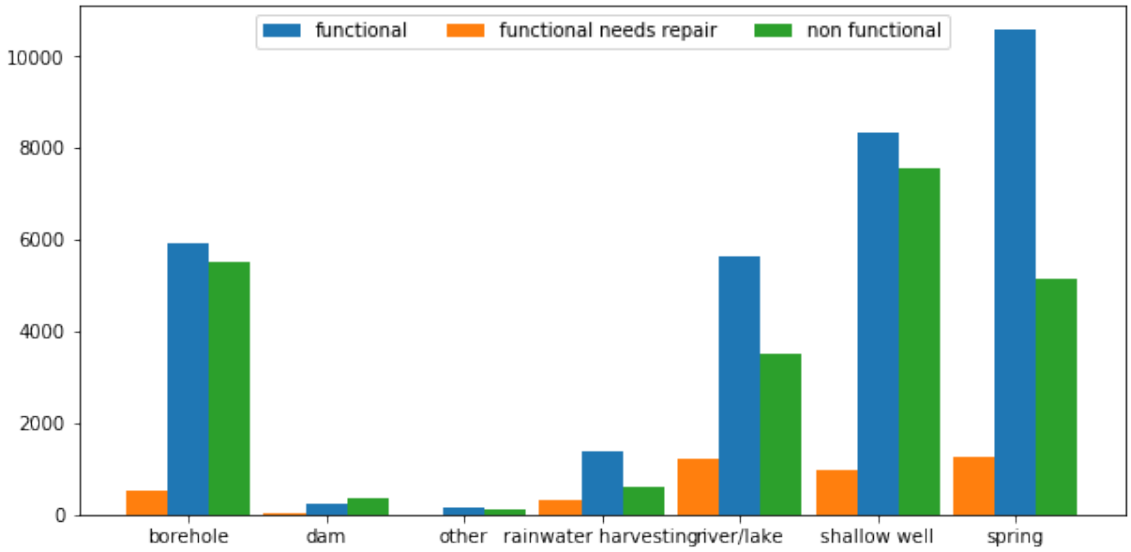


Figure 12: Source_type of the water pump with respect to status group

From this plot we can deduce that waterpumps which have the source type spring or river/lake have high probability of being functional,while the classes are more evenly distributed across the other variables.

5 Analysis and Exploration

5.1 Naive Bayes Classifier for classifying the waterpumps

Classification is a supervised learning task, where the dataset used must be represented as a labelled pair:

$$(x_i, y_j) \tag{1}$$

where x_i are the different features that represent a case in the dataset that has been labelled using a class y_j. In the Tanzania dataset we have each waterpump represented by features (the waterpump characteristics, collected from the resident feedback) and a class label represented by the status of the waterpump i.e functional,functional need repair,and non functional. A classification algorithm is also an estimator, which takes as input a dataset and produces an class estimate for objects in the dataset. Using classification algorithm to classify objects in a dataset requires two steps, the first step which we classically call the "training" step and the second step the "testing" step. The training step is where we input a subset of our dataset as training-data into the classifier and allow the algorithm to "learn" from the training data. The objective of the learning process is to obtain a "well learnt" algorithm that when fed with the test dataset generalizes in a meaningful way what it has "learnt" from the training sample.[4] In this way the "learning" task looked upon mathematically, is trying to estimate the conditional distribution:

$$P(Y = y_j \mid X = x_i) \tag{2}$$

5.2 Naive Bayes Classifier algorithm

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. They are extremely fast and simple classification algorithms that are often suitable for very high-dimensional datasets.

5.2.1 Naive Bayes Explanation and Theory

As discussed above, the objective of the learning process of an classification task looked upon mathematically, is to find an estimate of the conditional distribution:

$$P(Y = y_j \mid X = x_i) \tag{3}$$

One way to estimate the conditional distribution $P(Y = y_j \mid X = x_i)$, is to utilize Bayes' theorem, mathematically is written as:

$$P(Y = y_j \mid X = x_i) = \frac{P(Y = y_j)P(X = x_i \mid Y = y_j)}{\sum_j P(Y = y_j)P(X = x_i \mid Y = y_j)} \tag{4}$$

The denominator can be removed and a proportionality can be introduced.

$$P(Y = y_j \mid X = x_i) \propto P(Y = y_j) \prod_{i=1}^{n} P(X = x_i \mid Y = y_j) \tag{5}$$

Finally, the class label predicted is the class for which the estimated value of $P(Y = y_j \mid X = x_i)$ is maximum, i.e.

$$Y_{pred} = \ argmax_Y P(Y) \prod_{i=1}^{n} P(x_i \mid Y) \tag{6}$$

The classifier can estimate the values for each $P(Y = y_j)$ and $P(X = x_i \mid Y = y_j)$ from the dataset during the training process. The value of $P(Y = y_j)$ is easily estimated by computing the ratio of data points that fall in each class y_j. Further, the modeller also needs to make important assumptions regarding the conditional distribution $P(X = x_i \mid Y = y_j)$. according to the dataset, the modeller can assume any distribution, and compute the distribution parameters from the dataset. Using the Naive Bayes classifier function from Skit learn, allows us to choose between three different widely used distributions. These are

- Normal (Gaussian) distribution
- Multivariate distribution
- Bernoulli distribution

We do not assume the conditional distribution $P(X = x_i \mid Y = y_j)$ to be Gaussian distribution because random variables that have a Gaussian distribution must take up a continuous data value space. In the data set we use, there are only three features that are continuous. Instead, before running the model we will drop these three variables. Similarly, if we do not assume that the conditional distribution $P(X = x_i \mid Y = y_j)$ is a Multivariate distribution because multinomial distribution describes the probability of observing counts among a number of categories, and thus multinomial naive Bayes is most appropriate for features that represent counts or count rates. In the data set we use, there are no such features that represent counts. Therefore,We will build our classifier on the assumption that the conditional distribution $P(X = x_i \mid Y = y_j)$ is a Bernoulli distribution.Bernoulli implements the naive Bayes training and classification algorithms according to multivariate Bernoulli distributions; i.e., there may be multiple features but each one is assumed to be a binary-valued (Bernoulli, boolean) variable. As, this method requires samples to be represented as binary-valued feature vector we have created dummy variable for all the categorical features present in our cleaned processed dataset. This final dataset has a total of 99 variables, each taking a boolean data value space. Figure below shows a snapshot of the first five rows of the final dataset with the dummy features.

	funder_Dis_Council	funder_Kkkt	funder_Tasaf	funder_Unicef	funder_danida	funder_gov	funder_hesawa	funder_other	funder_rwssp	funder_world_ba
40630	0	0	0	0	0	1	0	0	0	
36440	0	0	0	0	0	0	0	1	0	
25799	0	0	0	0	0	0	0	1	0	
24265	0	0	0	0	0	0	0	1	0	
1462	0	0	0	0	0	0	0	1	0	

5 rows × 96 columns

Figure 13: dummy variable dataset

5.3 Results

After creating the dummy variable for our features, the dataset has total 99 features. Using all the 99 features might lead to the over fitting of the data. So, to avoid the over fitting we need to figure out what are the best number of features to be used. Therefore, we first split the data set into training data and the test data using train_test_split function from sklearn package and run

the different models for different number of feature ranging from 5 to 99. Below plot shows the testing accuracy of Bernoulli Naive Bayes for features ranging from 5 to 99.

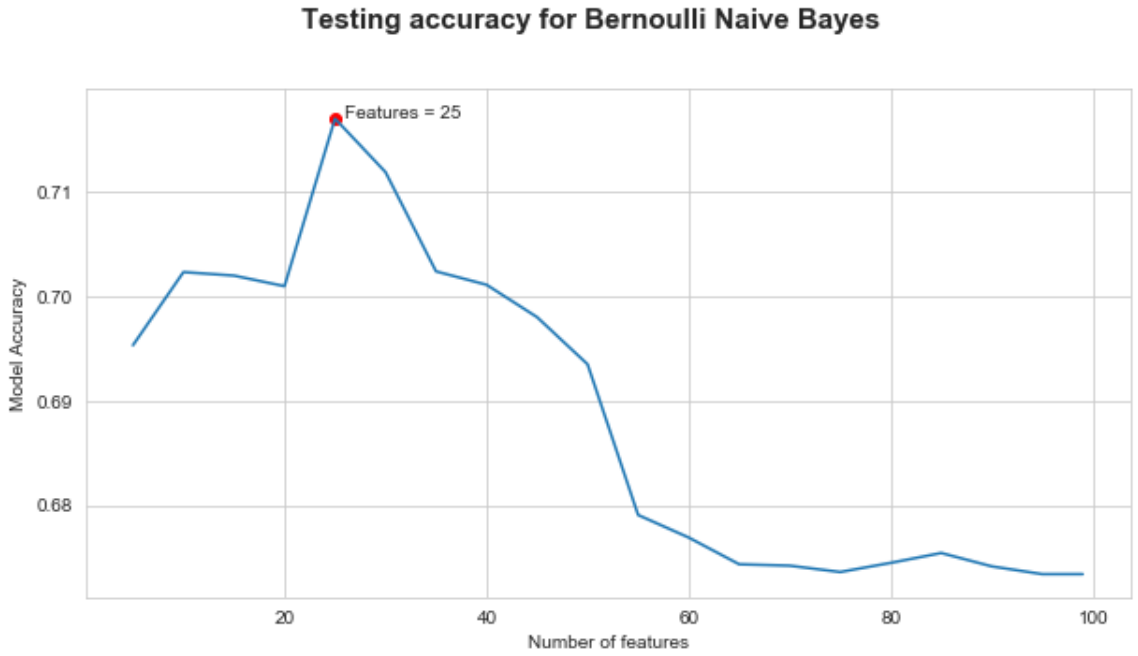


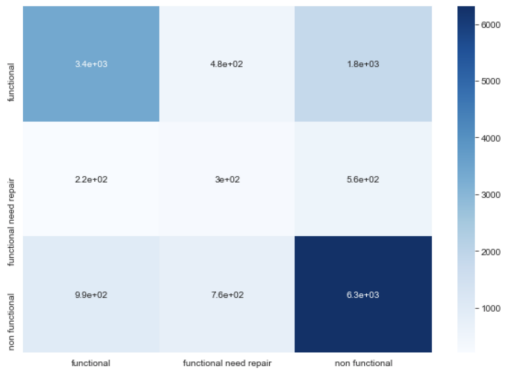
Figure 14: Bernoulli Naive Bayes Model Accuracy for different number of variables

From the plot, we can determine that the model with 25 features has the maximum accuracy of 71%. So, as the final model, I build a model with 25 features. The figure below shows the evaluation metrics and confusion metrics for this model.

Classification Performance:

	Value
accuracy	0.715017
precision	0.701890
recall	0.715017

(a) Model Evaluation Metrics



(b) Confusion Metrics

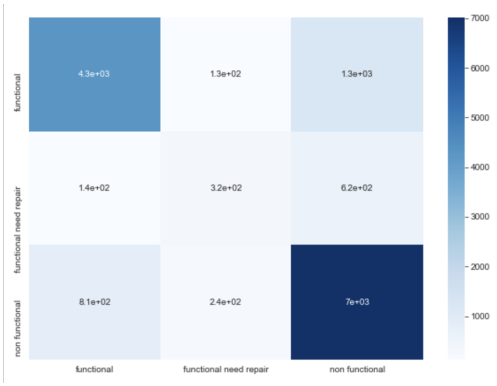
These metrics show that the model is able to accurately classify 75% of water pumps present in the test dataset. This performance of the models can be further improved dealing with class imbalance and choosing other classifiers such as Random Forest. This optimization task for this project is further explored in this report.

6 Random Forest Classifier

In comparision to the Naive Bayes classifier, which was a generative model, that learns the joint distribution $P(X = x_i, Y = y_j)$ and tried to estimate the conditional distribution $P(Y = y_j \mid X = x_i)$, Random forest is an ensemble tree-based learning algorithm. Tree based learning algorithms are discriminative models,that instead of estimating distributions, estimate the decision boundaries between the classes. The Random Forest Classifier is a set of decision trees from randomly selected subset of training set. It aggregates the votes from different decision trees to decide the final class of the test object.[5]. Classification trees have a built-in method for feature selection, which chooses the feature with the most impact at each split. Thus, adding more features should generally increase predictive accuracy.Therefore,the training set includes all 99 features for the model to randomy select the subset and build the ensembled trees.The random forest algorithm in sklearn package of python can be passed other parameters. The random forest model built for this project includes following parameters: n_estimators=200,max_depth=50. The figure below shows the evaluation metrics for this model.

	Value
accuracy	0.783502
precision	0.775930
recall	0.783502

(a) Random Forest Model Evaluation Metrics



(b) Confusion Metrics

These metrics show that the model is able to accurately classify 78% of water pumps present in the test dataset.All the metrices are higher with random forest model in comparision with Bernoulli Naive Bayes Model.To further improve our classification performance, I explore the issue of class imbalance by using the oversampling method of SOMTE present in sklearn library. The figure below shows the count of the data points in each class after balancing the dataset.

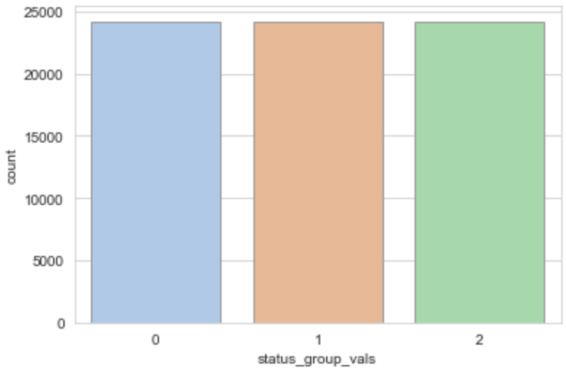


Figure 17: count of the balanced labels

The final training dataset includes 99 features with the balanced labels, I ran the random forest model again to see if there is improvement in the performance of the model. The below figure shows the performance of the classifier with the balanced labels:

	Value
accuracy	0.762626
precision	0.771804
recall	0.762626

Figure 18: Random Forest Model Evaluation Metrics

From the evaluation metrics its clearly seen that balancing the labels does not have much effect on the classification performance, in fact the accuracy reduced by 2% and rest of the metrices are almost same.So instead of further exploring the other techniques of mitigating with imbalanced dataset,I decided to perform another ensemble classifier called Gradient boosting classifier.

7 Gradient boosting classifier

Similar to Random Forests, Gradient Boosting is an ensemble learner. Gradient Boosting is a technique for performing supervised machine learning tasks, like classification and regression. The idea behind using the gradient boosting after random forest is that in gradient boosting trees are built sequentially i.e. each tree is grown using information from previously grown trees unlike in bagging where we create multiple copies of original training data and fit separate decision tree on each. [6][7]
The gradient boosting algorithm in sklearn package of python can be passed other parameters. Here are the best ones that I have chosen, learning_rate, max_depth ,min_samples_leaf,max_features and the n_estimators. The max_depth and n_estimators are also the same parameters we chose in a random forest. Here we are taking an extra that is the learning_rate, min_samples_leaf and max_features. The figure below shows the evaluation metrics for this model:

```
acurracy for gradient boosting classifier 0.7901683501683502
```

Figure 19: Gradient boosting model Evaluation

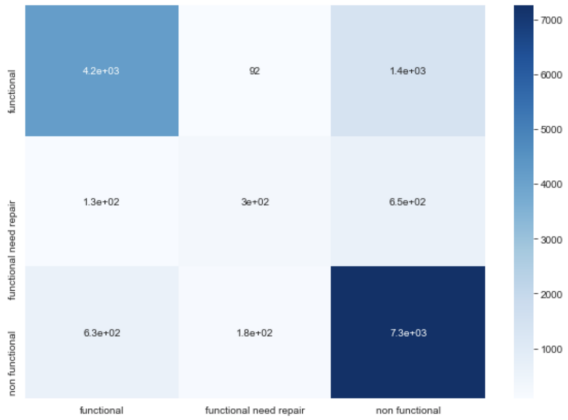


Figure 20: Confusion Metrics

8 Conclusion

It is intuitive to think about how the cost included in repairing the pumps will be less than fixing the non-functional pumps.The cost of a standard pump ranges from $100-2000$ []5, and installing this pump further requires drilling which can be cost aroud $1000-3000$. On the other hand maintaining and repairing existing pumps would only cost tens of dollars [][6]. Thus,we focus only on identifying the pumps that needs repair in comparison with non- functional pumps, due to cost reasons. Optimally for this, we want to a classifier which has the high true positive rates, and low the false positive rate for the 'functional needs repair class.

When we look at the the confusion matrix for the Bernoulli naive bayes classifier,*fig 15(b) we see that for the 'functional needs repair' class, out-of 1080 pumps in the test data-set falling in this class, the model correctly classifies 300 pumps but, miss-classifies 560 pumps as non-functional and 220 as functional which shows that this model does a good job in identifying functional needs repair from functional pumps, but is not able to identify functional needs repair from non-functional pumps.
Next,when we look at the the confusion matrix for the random forest classifier,*fig 16(b) we see that for the 'functional needs repair' class, out-of 1080 pumps in the test data-set falling in this class, the model correctly classifies 320 pumps but, miss-classifies 620 pumps as non-functional and 140 as functional which shows that this model does a better job in identifying functional needs repair from functional pumps, but there is not much improvement seen in identify functional needs repair from non-functional pumps.
Looking at the confusion matrix for the Gradient Boosting classifier,*fig 18(b) we see that for the 'functional needs repair' class, out-of 1080 pumps in the test data-set falling in this class, the model correctly classifies 300 pumps but, miss-classifies 650 pumps as non-functional and 130 as functional which shows that this model dis not perform well in identifying functional needs repair from functional pumps,neither there is improvement seen in identify functional needs repair from non-functional pumps.

After looking at the confusion matrix of all three classifiers we can say that it difficult to choose one classifier from the above which correctly classified the pumps which needs repair but if we just concerned about the prediction in terms of accuracy than random forest performs better than others.

To further improve our performance we can use sparse classification method. Since our dataset contains 40 features with 31 categorical features, we believe that using sparse classification models will help determining important features that will significantly speed up our training time for models such as Random Forest. Additionally, we would like to try out few other methods to reduce the arity of the features such as clustering.
The another scope to improve performance is to identify the lifetime of the waterpumps and see how it correlates across different population and number of pumps across the area.Assuming that the pump will be more prone to being non-functional and functional needs repairs in areas where there is dense population and not enough coverage by the functional pumps.

References

[1] Drivendata, *pump-it-up-data-mining-the-water-table*, 2020. https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/page/25/.

[2] A. I. 'D'Enza' and M. Greenacre', *Multiple correspondence analysis for the quantification and visualization of large categorical data sets*. 2012.

[3] W.-Y. W. 'Han', 'Hui' and B.-H. Mao', *Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning*. 2005.

[4] J. F. "Trevor hastie", "Robert Tibshirani", *The Elements of Statistical Learning: Data Mining, Inference and Prediction*.

[5] A. 'Liaw' and M. Wiener', *Classification and regression by randomForest*. 2002.

[6] Z. 'Xu', *Gradient boosted feature selection*. 2019.

[7] kharshit, *Gradient Boosting vs Random Forest*, 2018 (accessed February 23, 2018). https://kharshit.github.io/blog/2018/02/23/gradient-boosted-trees-better-than-random-forest.

[8] H. Jaeger, *Machine Learning lecture notes Spring 2019*.

A column description

- amount_tsh - Total static head (amount water available to waterpoint)
- date_recorded - The date the row was entered
- funder - Who funded the well
- gps_height - Altitude of the well
- installer - Organization that installed the well
- longitude - GPS coordinate
- latitude - GPS coordinate
- wpt_name - Name of the waterpoint if there is one
- basin - Geographic water basin
- subvillage - Geographic location
- region - Geographic location
- region_code - Geographic location (coded)
- district_code - Geographic location (coded)
- lga - Geographic location
- ward - Geographic location
- population - Population around the well
- public_meeting - True/False
- recorded_by - Group entering this row of data
- scheme_management - Who operates the waterpoint
- scheme_name - Who operates the waterpoint
- permit - If the waterpoint is permitted
- construction_year - Year the waterpoint 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 - How the waterpoint is managed
- management_group - How the waterpoint is managed

- payment - What the water costs
- payment_type - What the water costs
- water_quality - The quality of the water
- quality_group - The quality of the water
- quantity - The quantity of water
- quantity_group - The quantity of water
- source - The source of the water
- source_type - The source of the water
- source_class - The source of the water
- waterpoint_type - The kind of waterpoint
- waterpoint_type_group - The kind of waterpoint

B Code

[11pt]article
[breakable]tcolorbox parskip
iftex [T1]fontenc mathpazo
graphicx caption nocaption
[Export]adjustbox max size=0.90.9 float xcolor enumerate geometry amsmath amssymb textcomp upquote eurosym [mathletters]ucs fancyvrb grffile
hyperref titling longtable booktabs [inline]enumitem [normalem]ulem mathrsfs
HighlightingVerbatimcommandchars=
{}
verbose,tmargin=1in,bmargin=1in,lmargin=1in,rmargin=1in

pump-it-up_advanceproject

August 10, 2020

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

[commandchars=
{},codes*=] import numpy as np import pandas as pd import time
import matplotlib.pyplot as plt import seaborn as sns import seaborn as sn
from sklearn.model_selection import train_test_split from sklearn.naive_bayes import GaussianNB from sklearn import metrics import prince from sklearn.feature_selection import SelectKBest, chi2 # for chi-squared feature selection
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder from sklearn.naive_bayes import MultinomialNB

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

[commandchars=
{},codes*=] train_data = pd.read_csv('training set values.csv') train_target = pd.read_csv('training set labels.csv')

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

[commandchars=
{},codes*=] # Merge data and labels together in one dataframe. train_data = pd.merge(train_data, train_target, on='id') del train_target

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

[commandchars=
{},codes*=] # Explore data set.
train_data.info()

[commandchars=
{},codes*=] ¡class 'pandas.core.frame.DataFrame'¿ Int64Index: 59400 entries, 0 to 59399 Data columns (total 41 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 55765 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 59400 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 55523 non-null object 21 scheme_name 31234 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 dtypes: float64(3), int64(7), object(31) memory usage: 19.0+ MB

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

{},codes*=] sns.catplot(x="status_group", kind="count",palette="pastel", edgecolor=".6",data=train_data)

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:

{},codes*=] <seaborn.axisgrid.FacetGrid at 0x1a46aa5890>

max size=0.90.9output_41.*png*

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

{},codes*=] train_data.head(5)

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:

{},codes*=] id amount_tsh date_recorded funder gps_height installer \ 0 69572 6000.0 2011-03-14 Roman 1390 Roman 1 8776 0.0 2013-03-06 Grumeti 1399 GRUMETI 2 34310 25.0 2013-02-25 Lottery Club 686 World vision 3 67743 0.0 2013-01-28 Unicef 263 UNICEF 4 19728 0.0 2011-07-13 Action In A 0 Artisan

longitude latitude wpt_name num_private ... water_quality \ 0 34.938093 -9.856322 none 0 ... soft 1 34.698766 -2.147466 Zahanati 0 ... soft 2 37.460664 -3.821329 Kwa Mahundi 0 ... soft 3 38.486161 -11.155298 Zahanati Ya Nanyumbu 0 ... soft 4 31.130847 -1.825359 Shuleni 0 ... soft

quality_group quantity quantity_group source \ 0 good enough enough spring 1 good insufficient insufficient rainwater harvesting 2 good enough enough dam 3 good dry dry machine dbh 4 good seasonal seasonal rainwater harvesting

source_type source_class waterpoint_type \ 0 spring groundwater communal standpipe 1 rainwater harvesting surface communal standpipe 2 dam surface communal standpipe multiple 3 borehole groundwater communal standpipe multiple 4 rainwater harvesting surface communal standpipe

waterpoint_type_group status_group 0 communal standpipe functional 1 communal standpipe functional 2 communal standpipe functional 3 communal standpipe non functional 4 communal standpipe functional

[5 rows x 41 columns]

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

{},codes*=] *# Check for nulls.*

train_data.apply(lambda x: sum(x.isnull()))

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:

{},codes*=] id 0 amount_tsh 0 date_recorded 0 funder 3635 gps_height 0 installer 3655 longitude 0 latitude 0 wpt_name 0 num_private 0 basin 0 subvillage 371 region 0 region_code 0 district_code 0 lga 0 ward 0 population 0 public_meeting

```
3334 recorded_by 0 scheme_management 3877 scheme_name 28166 permit 3056
construction_year 0 extraction_type 0 extraction_type_group 0 extraction_type_class
0 management 0 management_group 0 payment 0 payment_type 0 water_quality 0
quality_group 0 quantity 0 quantity_group 0 source 0 source_type 0 source_class
0 waterpoint_type 0 waterpoint_type_group 0 status_group 0 dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, frame hidden, borders= {},codes*=]

```
# Deal with the columns containing null values one by one. Start with 'funder'.
train_data.funder.value_counts()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0.5]:

```
Government Of Tanzania 9084 Danida 3114 Hesawa 2202 Rwssp 1374 World
Bank 1349 ...  Rc Missi 1 Tgrs 1 Dhv\swis 1 Rotary Club Of Chico And Moshi 1
Unice/ Cspd 1 Name: funder, Length: 1897, dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, frame hidden, borders= {},codes*=]

```
# Create a function to reduce the amount of dummy columns needed
whilst maintaining the  # information contained in the column.
def funder_wrangler(row): '''Keep top 10 values and set the rest to 'other''' 
if row['funder']=='Government Of Tanzania': return 'gov' elif
row['funder']=='Danida': return 'danida' elif row['funder']=='Hesawa':
 ↪ return 'hesawa' elif row['funder']=='Rwssp': return 'rwssp' elif
row['funder']=='World Bank': return 'world_bank' elif row['funder']=='Kkkt':
 ↪ return 'Kkkt' elif row['funder']=='World Vision ': return 'World_Vision '
elif row['funder']=='Unicef': return 'Unicef' elif row['funder']=='Tasaf': return
'Tasaf' elif row['funder']=='District Council': return 'Dis_Council' else: return
'other'
train_data['funder'] = train_data.apply(lambda row: funder_wrangler(row),
axis=1)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, frame hidden, borders= {},codes*=]

```
# Add column named 'status_group_vals' to allow the use of a pivot
table to check differences # between the different funders.
vals_to_replace = {'functional':2, 'functional needs repair':1, 'non
functional':0}
train_data['status_group_vals'] = train_data.status_group.
 ↪replace(vals_to_replace)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, frame hidden, borders= {},codes*=]

```
per = train_data.groupby(['funder','status_group'])['status_group_vals']\
.count() \ .unstack() \ .reset_index() \ .fillna(0) \ .set_index('funder') per.
 ↪head(10)
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:

```
status_group functional functional needs repair non functional funder
Dis_Council 378 45 420 Kkkt 723 66 498 Tasaf 493 64 320 Unicef 600 99 358 danida
```

```
1713 159 1242 gov 3720 701 4663 hesawa 936 232 1034 other 22346 2745 13122 rwssp
805 109 460 world_bank 545 97 707
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] per['total'] = per.sum(axis=1)
```

```
per
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
{},codes*=] status_group functional functional needs repair non functional total
funder Dis_Council 378 45 420 843 Kkkt 723 66 498 1287 Tasaf 493 64 320 877 Unicef
600 99 358 1057 danida 1713 159 1242 3114 gov 3720 701 4663 9084 hesawa 936 232
1034 2202 other 22346 2745 13122 38213 rwssp 805 109 460 1374 world_bank 545 97
707 1349
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] def percentage_share(df): for i in df.columns: df[i] = df.apply(lambda
row: (row[i]/row['total']*100), axis=1) df.drop('total',inplace=True,axis=1)
return df
```

```
per_df = percentage_share(per)
```

```
per_df
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
{},codes*=] status_group functional functional needs repair non functional funder
Dis_Council 44.839858 5.338078 49.822064 Kkkt 56.177156 5.128205 38.694639 Tasaf
56.214367 7.297605 36.488027 Unicef 56.764428 9.366131 33.869442 danida 55.009634
5.105973 39.884393 gov 40.951123 7.716865 51.332012 hesawa 42.506812 10.535876
46.957312 other 58.477481 7.183419 34.339099 rwssp 58.588064 7.933042 33.478894
world_bank 40.400297 7.190511 52.409192
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] funder = [0,1,2,3,4,5,6,7,8,9] functional = per_df['functional']
nr_functional = per_df['functional needs repair'] n_functional = per_df['non
functional'] header = per_df.columns.values #an array of columns headers
   fig = plt.figure() ax = plt.subplot()
   #Grouped bar chart ind = np.arange(len(funder)) #no of x ticks; width = 0.3
#width of the bar
   #ax.bar(position of the bar wrt the x-ticks, data, width of bar, label)
ax.bar(ind , functional, width,label='functional') ax.bar(ind- width ,
 ↪ nr_functional, width,label='functional needs repair') ax.bar(ind + width ,
n_functional, width,label='non functional') ax.legend(loc=9,ncol=3)
   ax.set_xticks([0,1,2,3,4,5,6,7,8,9]) ax.set_ylim(0,70) #create a list of
variable names to use as labels for ticks in hor axis xlab=[item.get_text() for
item in ax.get_xticklabels()] # list with empty strings. list length determined
my the ticks seen in the plots # replace empty strings at even list indexes by
variable names in the same order as they are present in the dataframe column
names xlab[0]='Dis_Council' xlab[1]='Kkkt' xlab[2]='Tasaf' xlab[3]='Unicef'
xlab[4]='danida' xlab[5]='gov' xlab[6] ='hesawa' xlab[7]='other' xlab[8]='rwssp'
xlab[9] ='world_bank'
```

```
   #set the vertical axis tick labels using the list created above ax.
↪set_xticklabels(xlab)
   #for i,v in enumerate(functional): #.text(i-.25,v+10,functional[i],fomtsize=8)
   for tick in ax.get_xticklabels(): tick.set_rotation(90)
   #Set the figure size so that all four panel graphs are clearly visible fig.
↪set_size_inches(10,5) #Set a title for the figure fig.suptitle(' Percentage
of wells by functionaity types for the top 10 funders '\ ,y=1,fontsize=15,
↪fontweight='bold') plt.show()
```

max size=0.90.9output$_1$3$_0$.*png*

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] # There are some clear differences here that will hopefully improve
the model. The next feature # to inspect is 'installer'.
   train_data.installer.value_counts()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
{},codes*=] DWE 17402 Government 1825 RWE 1206 Commu 1060 DANIDA 1050 ...  TINA/
↪Africare 1 Tanz/Egypt technical coopera 1 AGRICAN 1 George mtoto company 1
Birage 1 Name: installer, Length: 2145, dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] # Create a function to reduce the amount of dummy columns needed
whilst maintaining the  # information contained in the column.
   def installer_wrangler(row): '''Keep top 7 values and set the
rest to 'other''' if row['installer']=='DWE': return 'dwe' elif
row['installer']=='Government': return 'gov' elif row['installer']=='RWE':
↪ return 'rwe' elif row['installer']=='Commu': return 'commu' elif
row['installer']=='DANIDA': return 'danida' elif row['installer']=='KKKT ':
return 'KKKT ' elif row['installer']=='Hesawa': return 'Hesawa'
   else: return 'other'
   train_data['installer'] = train_data.apply(lambda row: installer_wrangler(row),
axis=1)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] per_inst = train_data.groupby(['installer','status_group'])['status_group_vals']\
.count() \ .unstack() \ .reset_index() \ .fillna(0) \ .set_index('installer')
per_inst.head(7)
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
{},codes*=] status_group functional functional needs repair non functional
installer Hesawa 475 17 348 commu 724 32 304 danida 542 83 425 dwe 9433 1622 6347
gov 535 256 1034 other 20246 2170 13601 rwe 304 137 765
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] per_inst['total'] = per_inst.sum(axis=1)
   per_inst
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:
{},codes*=] status_group functional functional needs repair non functional total
installer Hesawa 475 17 348 840 commu 724 32 304 1060 danida 542 83 425 1050 dwe
9433 1622 6347 17402 gov 535 256 1034 1825 other 20246 2170 13601 36017 rwe 304
137 765 1206

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]:
{},codes*=] perinst_df = percentage_share(per_inst)

perinst_df

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:
{},codes*=] status_group functional functional needs repair non functional
installer Hesawa 56.547619 2.023810 41.428571 commu 68.301887 3.018868 28.679245
danida 51.619048 7.904762 40.476190 dwe 54.206413 9.320768 36.472819 gov 29.
↪315068 14.027397 56.657534 other 56.212344 6.024933 37.762723 rwe 25.207297
11.359867 63.432836

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]:
{},codes*=] installer = [0,1,2,3,4,5,6] functional = perinst_df['functional']
nr_functional = perinst_df['functional needs repair'] n_functional =
perinst_df['non functional'] header = perinst_df.columns.values #an array of
columns headers

fig = plt.figure() ax = plt.subplot()

#Grouped bar chart ind = np.arange(len(installer)) #no of x ticks; width = 0.3
#width of the bar

#ax.bar(position of the bar wrt the x-ticks, data, width of bar, label)
ax.bar(ind , functional, width,label='functional') ax.bar(ind- width ,
↪ nr_functional, width,label='functional needs repair') ax.bar(ind + width ,
n_functional, width,label='non functional') ax.legend(loc=9,ncol=3)

ax.set_xticks([0,1,2,3,4,5,6]) ax.set_ylim(0,80) #create a list of variable
names to use as labels for ticks in hor axis xlab=[item.get_text() for item in
ax.get_xticklabels()] # list with empty strings. list length determined my the
ticks seen in the plots # replace empty strings at even list indexes by variable
names in the same order as they are present in the dataframe column names
xlab[0]='Hesawa' xlab[1]='commu' xlab[2]='danida' xlab[3]='dwe' xlab[4]='gov'
xlab[5]='other' xlab[6] ='rwe'

#set the vertical axis tick labels using the list created above ax.
↪set_xticklabels(xlab)

for tick in ax.get_xticklabels(): tick.set_rotation(90)

#Set the figure size so that all four panel graphs are clearly visible fig.
↪set_size_inches(10,5) #Set a title for the figure fig.suptitle(' Percentage
of wells by functionaity types for the top 7 installers '\ ,y=1,fontsize=15,
↪fontweight='bold') plt.show()

max size=0.90.9output$_1$9$_0$.*png*

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]:

```
[commandchars=
{},codes*=] # As was the case with 'funder' there are some clear differences
here that will hopefully # improve the model. The next feature to inspect is
'subvillage'.
print(train_data.subvillage.value_counts())
```

```
[commandchars=
{},codes*=] Madukani 508 Shuleni 506 Majengo 502 Kati 373 Mtakuja 262 ... Kwasekibaja 1
Nyagudi 1 Itembe 1 Daghaseta 1 Mwamazengo 1 Name: subvillage, Length: 19287, dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
[commandchars=
{},codes*=] print('Number of villages: ', len(train_data.subvillage.
↪value_counts()))
```

```
[commandchars=
{},codes*=] Number of villages: 19287
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
[commandchars=
{},codes*=] # 19287 unique values! The largest values don't dominate as was
the case with installer and # funder. It's probably not worth creating dummy
variables for the top 5. I'll drop this one but # feel free to experiment here.
train_data.drop('subvillage',inplace = True, axis=1)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
[commandchars=
{},codes*=] # Let's investigate the next column containg null data:
'public_meeting'.
train_data.public_meeting.value_counts()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0.2]:

```
[commandchars=
{},codes*=] True 51011 False 5055 Name: public_meeting, dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
[commandchars=
{},codes*=] # We only have two values here: true and false. This one can stay but
we'll have to replace  # the unknown data with a string value.
train_data.public_meeting = train_data.public_meeting.fillna('Unknown')
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
[commandchars=
{},codes*=] train_data.public_meeting.head(20)
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0.2]:

```
[commandchars=
{},codes*=] 0 True 1 Unknown 2 True 3 True 4 True 5 True 6 True 7 True 8 True 9
True 10 True 11 True 12 True 13 True 14 True 15 True 16 True 17 True 18 Unknown
19 True Name: public_meeting, dtype: object
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
[commandchars=
{},codes*=] train_data.scheme_management.value_counts()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0.2]:

```
[commandchars=
{},codes*=] 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 None
1 Name: scheme_management, dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] # Create a function to reduce the amount of dummy columns needed whilst maintaining the # information contained in the column.
def scheme_wrangler(row): '''Keep top 8 values and set the rest to 'other'. ''' if row['scheme_management']=='VWC': return 'vwc' elif row['scheme_management']=='WUG': return 'wug' elif row['scheme_management']=='Water authority': return 'wtr_auth' elif row['scheme_management']=='WUA': return 'wua' elif row['scheme_management']=='Water Board': return 'wtr_brd' elif row['scheme_management']=='Parastatal': return 'Parastatal' elif row['scheme_management']=='Private operator': return 'pri_opr' elif row['scheme_management']=='Company': return 'comp' else: return 'other'
train_data['scheme_management'] = train_data.apply(lambda row: scheme_wrangler(row), axis=1)
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] per_sch_mgmt = train_data.groupby(['scheme_management', ↪'status_group'])['status_group_vals']\ .count() \ .unstack() \ .reset_index() \ .fillna(0) \ .set_index('scheme_management') per_sch_mgmt.head(8)
[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:
{},codes*=] status_group functional functional needs repair non functional scheme_management Parastatal 966 202 512 comp 534 37 490 other 2398 251 2164 pri_opr 729 23 311 vwc 18960 2334 15499 wtr_auth 1618 448 1087 wtr_brd 2053 111 584 wua 1995 239 649
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] per_sch_mgmt['total'] = per_sch_mgmt.sum(axis=1)
per_sch_mgmt
[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:
{},codes*=] status_group functional functional needs repair non functional total scheme_management Parastatal 966 202 512 1680 comp 534 37 490 1061 other 2398 251 2164 4813 pri_opr 729 23 311 1063 vwc 18960 2334 15499 36793 wtr_auth 1618 448 1087 3153 wtr_brd 2053 111 584 2748 wua 1995 239 649 2883 wug 3006 672 1528 5206
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] #calculating the percentage per_sch_mgmt_df = percentage_share(per_sch_mgmt)
per_sch_mgmt_df
[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:
{},codes*=] status_group functional functional needs repair non functional scheme_management Parastatal 57.500000 12.023810 30.476190 comp 50.329877 3.487276 46.182846 other 49.823395 5.215043 44.961562 pri_opr 68.579492 2.163688 29.256820 vwc 51.531541 6.343598 42.124861 wtr_auth 51.316207 14.208690 34.475103 wtr_brd 74.708879 4.039301 21.251820 wua 69.198751 8.289976 22.511273 wug 57.741068 12. ↪908183 29.350749

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] scheme_management = [0,1,2,3,4,5,6,7,8] functional = per_sch_mgmt_df['functional'] nr_functional = per_sch_mgmt_df['functional needs repair'] n_functional = per_sch_mgmt_df['non functional'] header = per_sch_mgmt_df.columns.values #an array of columns headers
fig = plt.figure() ax = plt.subplot()
#Grouped bar chart ind = np.arange(len(scheme_management)) #no of x ticks; width = 0.3 #width of the bar
#ax.bar(position of the bar wrt the x-ticks, data, width of bar, label)
ax.bar(ind , functional, width,label='functional') ax.bar(ind- width , nr_functional, width,label='functional needs repair') ax.bar(ind + width , n_functional, width,label='non functional') ax.legend(loc=9,ncol=3)
ax.set_xticks([0,1,2,3,4,5,6,7,8]) ax.set_ylim(0,80) #create a list of variable names to use as labels for ticks in hor axis xlab=[item.get_text() for item in ax.get_xticklabels()] # list with empty strings. list length determined my the ticks seen in the plots # replace empty strings at even list indexes by variable names in the same order as they are present in the dataframe column names xlab[0]='Parastatal' xlab[1]='comp' xlab[2]='other' xlab[3]='pri_opr' xlab[4]='vwc' xlab[5]='wtr_auth' xlab[6] ='wtr_brd' xlab[7] ='wua' xlab[8] ='wug'
#set the vertical axis tick labels using the list created above ax.set_xticklabels(xlab)
#for i,v in enumerate(functional): #.text(i-.25,v+10,functional[i],fomtsize=8)
for tick in ax.get_xticklabels(): tick.set_rotation(90)
#Set the figure size so that all four panel graphs are clearly visible fig.set_size_inches(10,5) #Set a title for the figure fig.suptitle(' Percentage of wells by functionaity types for the top 9 Schmeme Manager '\ ,y=1,fontsize=15,fontweight='bold') plt.show()

max size=0.90.9output$_3$1$_0$.*png*

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] train_data.scheme_name.value_counts()
[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:
{},codes*=] K 682 None 644 Borehole 546 Chalinze wate 405 M 400 ... Michee Borehole Scheme 1 Mirumba 1 Hivuga Water Supply 1 Kayugi spring source 1 Mugoma spring source 1 Name: scheme_name, Length: 2696, dtype: int64
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] len(train_data.scheme_name.unique())
[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:
{},codes*=] 2697
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] # Lots of factors and the top 5 or so only represent a fraction of the total values. Probably # safe to drop this column.

```
train_data.drop('scheme_name',inplace = True , axis=1)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] # The final column containing nulls is 'permit'.
train_data.permit.value_counts()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
{},codes*=] True 38852 False 17492 Name: permit, dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] # We only have two values here: true and false. This one can stay but
we'll have to replace  # the unknown data with a string value.
train_data.permit = train_data.permit.fillna('Unknown')
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] sns.catplot(x="permit", kind="count",palette="pastel", edgecolor=".6",
↪data=train_data)
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
{},codes*=] <seaborn.axisgrid.FacetGrid at 0x1a1c078550>
```

max size=0.90.9output$_3$7$_1$.*png*

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] # Excellent! Now there are no nulls in the data set. We can move on to
look at columns with  # string values and modify or remove them as we see fit.
str_cols = train_data.select_dtypes(include = ['object']) str_cols.apply(lambda
x: len(x.unique()))
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
{},codes*=] date_recorded 356 funder 10 installer 7 wpt_name 37400 basin 9 region
21 lga 125 ward 2092 public_meeting 3 recorded_by 1 scheme_management 9 permit
3 extraction_type 18 extraction_type_group 13 extraction_type_class 7 management
12 management_group 5 payment 7 payment_type 7 water_quality 8 quality_group 6
quantity 5 quantity_group 5 source 10 source_type 7 source_class 3 waterpoint_type
7 waterpoint_type_group 6 status_group 3 dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] train_data.apply(lambda x: sum(x.isnull()))
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
{},codes*=] id 0 amount_tsh 0 date_recorded 0 funder 0 gps_height 0 installer
0 longitude 0 latitude 0 wpt_name 0 num_private 0 basin 0 region 0 region_code
0 district_code 0 lga 0 ward 0 population 0 public_meeting 0 recorded_by
0 scheme_management 0 permit 0 construction_year 0 extraction_type 0
extraction_type_group 0 extraction_type_class 0 management 0 management_group 0
payment 0 payment_type 0 water_quality 0 quality_group 0 quantity 0 quantity_group
0 source 0 source_type 0 source_class 0 waterpoint_type 0 waterpoint_type_group 0
status_group 0 status_group_vals 0 dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] # 'Date recorded'
train_data.date_recorded.describe()
[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:
{},codes*=] count 59400 unique 356 top 2011-03-15 freq 572 Name: date_recorded, dtype: object
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] import datetime
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] train_data['year'] = pd.DatetimeIndex(train_data['date_recorded']).year
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] op_years = list(train_data.year-train_data.construction_year)
operational_years = [] for i in op_years: if (i > 500) or (i < 0): operational_years.append(0) else: operational_years.append(i)
train_data['operational_years'] = operational_years
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] train_data.operational_years
[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:
{},codes*=] 0 12 1 3 2 4 3 27 4 0 .. 59395 14 59396 15 59397 0 59398 0 59399 9
Name: operational_years, Length: 59400, dtype: int64
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] piv_table = pd.pivot_table(train_data, index=['operational_years', 'status_group'], values=['status_group_vals'], aggfunc='count') piv_table
[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:
{},codes*=] status_group_vals operational_years status_group 0 functional 10972
functional needs repair 1800 non functional 8534 1 functional 1754 functional
needs repair 66 ... ... 52 functional needs repair 3 non functional 7 53
functional 23 functional needs repair 6 non functional 62
[160 rows x 1 columns]
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] yr = [] functional = [] nr_functional = [] n_functional = []
for i in range(1,25): yr.append(i) functional.append(piv_table.loc[i].loc['functional'][0]) nr_functional.append(piv_table.loc[i].loc['functional needs repair'][0]) n_functional.append(piv_table.loc[i].loc['non functional'][0])
header = per_sch_mgmt_df.columns.values #an array of columns headers
fig = plt.figure() ax = plt.subplot()
#Grouped bar chart ind = np.arange(len(yr)) #no of x ticks; width = 0.3 #width of the bar

```
   #ax.bar(position of the bar wrt the x-ticks, data, width of bar, label)
ax.bar(ind , functional, width,label='functional') ax.bar(ind- width ,
 ↪ nr_functional, width,label='functional needs repair') ax.bar(ind + width ,
n_functional, width,label='non functional') ax.legend(loc=9,ncol=3)
   for tick in ax.get_xticklabels(): tick.set_rotation(90)
   #Set the figure size so that all four panel graphs are clearly visible fig.
 ↪set_size_inches(10,5) #Set a title for the figure fig.suptitle(' Percentage
of wells by functionaity types for 25 operational years '\ ,y=1,fontsize=15,
 ↪fontweight='bold') plt.show()
```

max size=0.90.9output$_4$6$_0$.*png*

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] train_data.drop('date_recorded',inplace = True , axis=1)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] train_data.drop('year',inplace = True , axis=1)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] # There's a wide range of data here hopefully it will help improve the
predictive power of our # models. Next up for inspection is 'wpt_name' (Name of
the waterpoint if there is one).
   train_data.wpt_name.value_counts()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
{},codes*=] none 3563 Shuleni 1748 Zahanati 830 Msikitini 535 Kanisani 323 ...
Kwa Keneth Mlimbila 1 Mchinga Ii Primary School 1 Ipoja 1 Arkaria Primary School
1 Kwa Fiderick Ipanta 1 Name: wpt_name, Length: 37400, dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] # Due to the huge number of factors and the lack of a clear dominating
value I'll drop this. # I may come back and include the top 5 later. Next up is
'basin'.
   train_data.drop('wpt_name',inplace = True , axis=1)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] train_data.basin.value_counts()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
{},codes*=] Lake Victoria 10248 Pangani 8940 Rufiji 7976 Internal 7785 Lake
Tanganyika 6432 Wami / Ruvu 5987 Lake Nyasa 5085 Ruvuma / Southern Coast 4493
Lake Rukwa 2454 Name: basin, dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] piv_table = pd.pivot_table(train_data, index=['basin',
'status_group'], values=['status_group_vals'], aggfunc='count') piv_table
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
[commandchars=
{},codes*=] status_group_vals basin status_group Internal functional 4482
functional needs repair 557 non functional 2746 Lake Nyasa functional 3324
functional needs repair 250 non functional 1511 Lake Rukwa functional 1000
functional needs repair 270 non functional 1184 Lake Tanganyika functional
3107 functional needs repair 742 non functional 2583 Lake Victoria functional
5100 functional needs repair 989 non functional 4159 Pangani functional 5372
functional needs repair 477 non functional 3091 Rufiji functional 5068 functional
needs repair 437 non functional 2471 Ruvuma / Southern Coast functional 1670
functional needs repair 326 non functional 2497 Wami / Ruvu functional 3136
functional needs repair 269 non functional 2582
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
[commandchars=
{},codes*=] # Most basins have have more functional than non-functional pumps.
Lake Rukwa # and Ruvuma don't. All the values are over 2000 so this looks like a
good feature to keep. # Region will be considered next.
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
[commandchars=
{},codes*=] train_data.region.value_counts()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
[commandchars=
{},codes*=] Iringa 5294 Shinyanga 4982 Mbeya 4639 Kilimanjaro 4379 Morogoro 4006
Arusha 3350 Kagera 3316 Mwanza 3102 Kigoma 2816 Ruvuma 2640 Pwani 2635 Tanga 2547
Dodoma 2201 Singida 2093 Mara 1969 Tabora 1959 Rukwa 1808 Mtwara 1730 Manyara
1583 Lindi 1546 Dar es Salaam 805 Name: region, dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
[commandchars=
{},codes*=] # Both basin, lga, ward and region contain geographical information so
there is a risk of them being # highly correlated with each other. I'll drop then
for now. # They could be be worth including though, so I may come back to them.
train_data.drop(['region', 'lga', 'ward'],inplace = True, axis=1)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
[commandchars=
{},codes*=] train_data.recorded_by.value_counts().describe()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
[commandchars=
{},codes*=] count 1.0 mean 59400.0 std NaN min 59400.0 25% 59400.0 50% 59400.0 75%
59400.0 max 59400.0 Name: recorded_by, dtype: float64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
[commandchars=
{},codes*=] # All data points have the same value so this offers no information
that would help build our # model. train_data.drop('recorded_by',inplace = True,
axis=1)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
[commandchars=
{},codes*=] # extraction_type, extraction_type_group and extraction_type_class
appear to contain very similar # data. I'll drop the first two and keep the last
one.
```

```
train_data.drop(['extraction_type', 'extraction_type_group'],inplace = True,
axis=1)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] train_data.management.value_counts()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:

```
{},codes*=] vwc 40507 wug 6515 water board 2933 wua 2535 private operator 1971
parastatal 1768 water authority 904 other 844 company 685 unknown 561 other -
school 99 trust 78 Name: management, dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] # This appears to be almost identical to 'scheme_management'. I'll
drop it.
train_data.drop('management',inplace = True, axis=1)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] train_data.management_group.value_counts()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:

```
{},codes*=] user-group 52490 commercial 3638 parastatal 1768 other 943 unknown 561
Name: management_group, dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] train_data.scheme_management.unique()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:

```
{},codes*=] array(['vwc', 'other', 'pri_opr', 'wug', 'wtr_brd', 'wua', 'wtr_auth',
'comp', 'Parastatal'], dtype=object)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] # Appears to offer no new info and is likely to overlap with
'scheme_management'.
train_data.drop('management_group',inplace = True, axis=1)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] train_data.payment.value_counts()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:

```
{},codes*=] never pay 25348 pay per bucket 8985 pay monthly 8300 unknown 8157 pay
when scheme fails 3914 pay annually 3642 other 1054 Name: payment, dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] train_data.payment_type.value_counts()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:

```
{},codes*=] never pay 25348 per bucket 8985 monthly 8300 unknown 8157 on failure
3914 annually 3642 other 1054 Name: payment_type, dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]:

[commandchars=
{},codes*=] # Payment and payment_type contain identical data. Remove one and keep the other.
train_data.drop('payment',inplace = True, axis=1)
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
[commandchars=
{},codes*=] train_data.water_quality.value_counts()
[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:
{},codes*=] soft 50818 salty 4856 unknown 1876 milky 804 coloured 490 salty abandoned 339 fluoride 200 fluoride abandoned 17 Name: water_quality, dtype: int64
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
[commandchars=
{},codes*=] train_data.quality_group.value_counts()
[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:
{},codes*=] good 50818 salty 5195 unknown 1876 milky 804 colored 490 fluoride 217 Name: quality_group, dtype: int64
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
[commandchars=
{},codes*=] sns.catplot(y="quality_group", kind="count",palette="pastel", edgecolor=".6",data=train_data)
[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:
{},codes*=] <seaborn.axisgrid.FacetGrid at 0x1a1b05d690>

max size=0.90.9output_6 9_1.*png*

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
[commandchars=
{},codes*=] piv_table = pd.pivot_table(train_data, index=['quality_group', 'status_group'], values=['status_group_vals'], aggfunc='count')
piv_table
[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:
{},codes*=] status_group_vals quality_group status_group colored functional 246 functional needs repair 54 non functional 190 fluoride functional 157 functional needs repair 13 non functional 47 good functional 28760 functional needs repair 3904 non functional 18154 milky functional 438 functional needs repair 14 non functional 352 salty functional 2394 functional needs repair 297 non functional 2504 unknown functional 264 functional needs repair 35 non functional 1577
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
[commandchars=
{},codes*=] quality_group = [0,1,2,3,4,5] functional = [246,157,28760,438,2394, ↪264] nr_functional = [54,13,3904,14,297,35] n_functional = [190,47,18154,352, ↪2504,1577]
fig = plt.figure() ax = plt.subplot()
#Grouped bar chart ind = np.arange(len(quality_group)) #no of x ticks; width = 0.3 #width of the bar

```
   #ax.bar(position of the bar wrt the x-ticks, data, width of bar, label)
ax.bar(ind , functional, width,label='functional') ax.bar(ind- width ,
 ↪ nr_functional, width,label='functional needs repair') ax.bar(ind + width ,
n_functional, width,label='non functional') ax.legend(loc=9,ncol=3)
   #create a list of variable names to use as labels for ticks in hor axis
xlab=[item.get_text() for item in ax.get_xticklabels()] # list with empty
strings. list length determined my the ticks seen in the plots # replace empty
strings at even list indexes by variable names in the same order as they are
present in the dataframe column names xlab[1]='colored' xlab[2]='fluoride'
xlab[3]='good' xlab[4]='milky' xlab[5]='salty' xlab[6]='unknown' ax.
 ↪set_xticklabels(xlab)
   for tick in ax.get_xticklabels(): tick.set_rotation(90)
   #Set the figure size so that all four panel graphs are clearly visible fig.
 ↪set_size_inches(10,5) #Set a title for the figure
   plt.show()
```

max size=0.90.9output$_7$1$_0$.*png*

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] # Water_quality and quality_group contain identical data. Remove one
and keep the other.
   train_data.drop('water_quality',inplace = True, axis = 1)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] train_data.quantity.value_counts()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
{},codes*=] enough 33186 insufficient 15129 dry 6246 seasonal 4050 unknown 789
Name: quantity, dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] piv_table = pd.pivot_table(train_data, index=['quantity',
'status_group'], values=['status_group_vals'], aggfunc='count')
   piv_table
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
{},codes*=] status_group_vals quantity status_group dry functional 157 functional
needs repair 37 non functional 6052 enough functional 21648 functional needs
repair 2400 non functional 9138 insufficient functional 7916 functional needs
repair 1450 non functional 5763 seasonal functional 2325 functional needs repair
416 non functional 1309 unknown functional 213 functional needs repair 14 non
functional 562
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] quantity = [0,1,2,3,4] functional = [157,21648,7916,2325,213]
nr_functional = [37,2400,1450,416,14] n_functional = [6052,9138,5763,1309,562]
   fig = plt.figure() ax = plt.subplot()
```

```
   #Grouped bar chart ind = np.arange(len(quantity)) #no of x ticks; width = 0.3
#width of the bar
   #ax.bar(position of the bar wrt the x-ticks, data, width of bar, label)
ax.bar(ind , functional, width,label='functional') ax.bar(ind- width ,
 ↪ nr_functional, width,label='functional needs repair') ax.bar(ind + width ,
n_functional, width,label='non functional') ax.legend(loc=9,ncol=3)
   #create a list of variable names to use as labels for ticks in hor
axis xlab=[item.get_text() for item in ax.get_xticklabels()] # list with
empty strings. list length determined my the ticks seen in the plots #
replace empty strings at even list indexes by variable names in the same
order as they are present in the dataframe column names xlab[1]='dry'
xlab[2]='enough' xlab[3]='insufficient' xlab[4]='seasonal' xlab[5]='unknown' ax.
 ↪set_xticklabels(xlab)
   for tick in ax.get_xticklabels(): tick.set_rotation(90)
   #Set the figure size so that all four panel graphs are clearly visible fig.
 ↪set_size_inches(10,5) #Set a title for the figure
   plt.show()
```

max size=0.90.9output$_7$5$_0$.*png*

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] train_data.quantity_group.value_counts()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
{},codes*=] enough 33186 insufficient 15129 dry 6246 seasonal 4050 unknown 789
Name: quantity_group, dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] # Quantity and quantity_group contain identical data. Remove one and
keep the other.
   train_data.drop('quantity_group',inplace = True,axis = 1)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] train_data.source.value_counts()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
{},codes*=] spring 17021 shallow well 16824 machine dbh 11075 river 9612 rainwater
harvesting 2295 hand dtw 874 lake 765 dam 656 other 212 unknown 66 Name: source,
dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] train_data.source_class.value_counts()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
{},codes*=] groundwater 45794 surface 13328 unknown 278 Name: source_class, dtype:
int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]:

[commandchars=
{},codes*=] train_data.source_type.value_counts()

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:

{},codes*=] spring 17021 shallow well 16824 borehole 11949 river/lake 10377
rainwater harvesting 2295 dam 656 other 278 Name: source_type, dtype: int64

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]:

{},codes*=] piv_table = pd.pivot_table(train_data, index=['source_type',
'status_group'], values=['status_group_vals'], aggfunc='count')

piv_table

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:

{},codes*=] status_group_vals source_type status_group borehole functional 5919
functional needs repair 508 non functional 5522 dam functional 253 functional
needs repair 24 non functional 379 other functional 158 functional needs repair
5 non functional 115 rainwater harvesting functional 1386 functional needs repair
314 non functional 595 river/lake functional 5627 functional needs repair 1233
non functional 3517 shallow well functional 8324 functional needs repair 957
non functional 7543 spring functional 10592 functional needs repair 1276 non
functional 5153

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]:

{},codes*=] source_type = [0,1,2,3,4,5,6] functional = [5919,253,158,1386,5627,
↪8324,10592] nr_functional = [508,24,5,314,1233,957,1276] n_functional = [5522,
↪379,115,595,3517,7543,5153]

fig = plt.figure() ax = plt.subplot()

#Grouped bar chart ind = np.arange(len(source_type)) *#no of x ticks;* width =
0.3 *#width of the bar*

#ax.bar(position of the bar wrt the x-ticks, data, width of bar, label)
ax.bar(ind , functional, width,label='functional') ax.bar(ind- width ,
↪ nr_functional, width,label='functional needs repair') ax.bar(ind + width ,
n_functional, width,label='non functional') ax.legend(loc=9,ncol=3)

#create a list of variable names to use as labels for ticks in hor
axis xlab=[item.get_text() for item in ax.get_xticklabels()] *# list with*
empty strings. list length determined my the ticks seen in the plots #
replace empty strings at even list indexes by variable names in the same
order as they are present in the dataframe column names xlab[1]='borehole'
xlab[2]='dam' xlab[3]='other' xlab[4]='rainwater harvesting' xlab[5]='river/lake'
xlab[6]='shallow well' xlab[7]='spring' ax.set_xticklabels(xlab)

#for tick in ax.get_xticklabels(): #tick.set_rotation(90)

#Set the figure size so that all four panel graphs are clearly visible fig.
↪set_size_inches(10,5) *#Set a title for the figure*

plt.show()

max size=0.90.9output$_8$2$_0$.*png*

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]:

```
[commandchars=
{},codes*=] # Source and source_type contain very similar information. Remove one
and keep the other.
train_data.drop('source',inplace = True,axis = 1)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
[commandchars=
{},codes*=] # gps_height, longitude, latitude, region_code and district_code are
all geographic info which # is unlikely to add any predictive power to the model
given that there are other variables # containing geographic data. 'num_private'
hasn't been given a discription on Driven Data, # it appears to be superflous. We
expect id to not contain any useful information so that gets # dropped too.
train_data.drop(['gps_height','longitude','latitude','region_code',
'district_code', 'num_private', 'id'], inplace = True,axis=1)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
[commandchars=
{},codes*=] train_data.describe()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0.2]:

```
[commandchars=
{},codes*=] amount_tsh population construction_year status_group_vals \ count
59400.000000 59400.000000 59400.000000 59400.000000 mean 317.650385 179.909983
1300.652475 1.158838 std 2997.574558 471.482176 951.620547 0.949794 min 0.000000
0.000000 0.000000 0.000000 25% 0.000000 0.000000 0.000000 0.000000 50% 0.000000
25.000000 1986.000000 2.000000 75% 20.000000 215.000000 2004.000000 2.000000 max
350000.000000 30500.000000 2013.000000 2.000000
operational_years count 59400.000000 mean 10.002778 std 12.457472 min 0.000000
25% 0.000000 50% 4.000000 75% 16.000000 max 53.000000
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
[commandchars=
{},codes*=] train_data.apply(lambda x: len(x.unique()))
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0.2]:

```
[commandchars=
{},codes*=] amount_tsh 98 funder 10 installer 7 basin 9 population
1049 public_meeting 3 scheme_management 9 permit 3 construction_year 55
extraction_type_class 7 payment_type 7 quality_group 6 quantity 5 source_type
7 source_class 3 waterpoint_type 7 waterpoint_type_group 6 status_group 3
status_group_vals 3 operational_years 54 dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
[commandchars=
{},codes*=] train_data.construction_year.value_counts()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0.2]:

```
[commandchars=
{},codes*=] 0 20709 2010 2645 2008 2613 2009 2533 2000 2091 2007 1587 2006 1471
2003 1286 2011 1256 2004 1123 2012 1084 2002 1075 1978 1037 1995 1014 2005 1011
1999 979 1998 966 1990 954 1985 945 1980 811 1996 811 1984 779 1982 744 1994 738
1972 708 1974 676 1997 644 1992 640 1993 608 2001 540 1988 521 1983 488 1975 437
1986 434 1976 414 1970 411 1991 324 1989 316 1987 302 1981 238 1977 202 1979 192
1973 184 2013 176 1971 145 1960 102 1967 88 1963 85 1968 77 1969 59 1964 40 1962
30 1961 21 1965 19 1966 17 Name: construction_year, dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] # Turn construction_year into a categorical column containing the following values: '60s', '70s', # '80s', '90s, '00s', '10s', 'unknown'.
def construction_wrangler(row): if row['construction_year'] >= 1960 and row['construction_year'] < 1970: return '60s' elif row['construction_year'] >= 1970 and row['construction_year'] < 1980: return '70s' elif row['construction_year'] >= 1980 and row['construction_year'] < 1990: return '80s' elif row['construction_year'] >= 1990 and row['construction_year'] < 2000: return '90s' elif row['construction_year'] >= 2000 and row['construction_year'] < 2010: return '00s' elif row['construction_year'] >= 2010: return '10s' else: return 'unknown'
train_data['construction_year'] = train_data.apply(lambda row: construction_wrangler(row), axis=1)
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] per_con_yr = train_data.groupby(['construction_year', ↪'status_group'])['status_group_vals']\ .count() \ .unstack() \ .reset_index() \ .fillna(0) \ .set_index('construction_year') per_con_yr.head(8)
[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:
{},codes*=] status_group functional functional needs repair non functional construction_year 00s 9989 977 4364 10s 3794 220 1147 60s 156 42 340 70s 1406 348 2652 80s 2220 423 2935 90s 4139 518 3021 unknown 10555 1789 8365
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] per_con_yr['total'] = per_con_yr.sum(axis=1)
per_con_yr
[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:
{},codes*=] status_group functional functional needs repair non functional total construction_year 00s 9989 977 4364 15330 10s 3794 220 1147 5161 60s 156 42 340 538 70s 1406 348 2652 4406 80s 2220 423 2935 5578 90s 4139 518 3021 7678 unknown 10555 1789 8365 20709
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] per_con_yr_df = percentage_share(per_con_yr)
per_con_yr_df
[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityback=0]:
{},codes*=] status_group functional functional needs repair non functional construction_year 00s 65.159817 6.373125 28.467058 10s 73.512885 4.262740 22. ↪224375 60s 28.996283 7.806691 63.197026 70s 31.911030 7.898320 60.190649 80s 39.799211 7.583363 52.617426 90s 53.907268 6.746549 39.346184 unknown 50.968178 8.638756 40.393066
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] construction_year = [0,1,2,3,4,5,6] functional = per_con_yr_df['functional'] nr_functional = per_con_yr_df['functional needs

```
repair'] n_functional = per_con_yr_df['non functional'] header = per_con_yr_df.
↪columns.values #an array of columns headers
   fig = plt.figure() ax = plt.subplot()
   #Grouped bar chart ind = np.arange(len(construction_year)) #no of x ticks;
width = 0.3 #width of the bar
   #ax.bar(position of the bar wrt the x-ticks, data, width of bar, label)
ax.bar(ind , functional, width,label='functional') ax.bar(ind- width ,
↪ nr_functional, width,label='functional needs repair') ax.bar(ind + width ,
n_functional, width,label='non functional') ax.legend(loc=9,ncol=3)
   ax.set_xticks([0,1,2,3,4,5,6]) ax.set_ylim(0,80) #create a list of variable
names to use as labels for ticks in hor axis xlab=[item.get_text() for item
in ax.get_xticklabels()] # list with empty strings. list length determined my
the ticks seen in the plots # replace empty strings at even list indexes by
variable names in the same order as they are present in the dataframe column
names xlab[0]='00s' xlab[1]='10s' xlab[2]='60s' xlab[3]='70s' xlab[4]='80s'
xlab[5]='90s' xlab[6] ='unknown'
   #set the vertical axis tick labels using the list created above ax.
↪set_xticklabels(xlab)
   for tick in ax.get_xticklabels(): tick.set_rotation(90)
   #Set the figure size so that all four panel graphs are clearly visible fig.
↪set_size_inches(10,5)
   plt.show()
```

max size=0.90.9output$_9$2$_0$.*png*

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] [341]: [commandchars=
{},codes*=]

```
sns.distplot(train_data.population, bins = 40) plt.show()
```

max size=0.90.9output$_9$3$_0$.*png*

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] [342]: [commandchars=
{},codes*=]

```
# This plot measures the amount of water available at the pump. It
looks a lot like the   # population graph which makes sense.
   train_data.population.describe()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0] [342]: [commandchars=
{},codes*=]

```
count 59400.000000 mean 179.909983 std 471.482176 min 0.000000 25%
0.000000 50% 25.000000 75% 215.000000 max 30500.000000 Name: population, dtype:
float64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder] [343]: [commandchars=
{},codes*=]

```
train_data.amount_tsh.describe()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0] [343]: [commandchars=
{},codes*=]

```
count 59400.000000 mean 317.650385 std 2997.574558 min 0.000000 25%
0.000000 50% 0.000000 75% 20.000000 max 350000.000000 Name: amount_tsh, dtype:
float64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] train_data.info()
[commandchars=
{},codes*=] ¡class 'pandas.core.frame.DataFrame'¿ Int64Index: 59400 entries, 0 to 59399 Data columns (total 20 columns): # Column Non-Null Count Dtype — —— ————— —— 0 amount_tsh 59400 non-null float64 1 funder 59400 non-null object 2 installer 59400 non-null object 3 basin 59400 non-null object 4 population 59400 non-null int64 5 public_meeting 59400 non-null object 6 scheme_management 59400 non-null object 7 permit 59400 non-null object 8 construction_year 59400 non-null object 9 extraction_type_class 59400 non-null object 10 payment_type 59400 non-null object 11 quality_group 59400 non-null object 12 quantity 59400 non-null object 13 source_type 59400 non-null object 14 source_class 59400 non-null object 15 waterpoint_type 59400 non-null object 16 waterpoint_type_group 59400 non-null object 17 status_group 59400 non-null object 18 status_group_vals 59400 non-null int64 19 operational_years 59400 non-null int64 dtypes: float64(1), int64(3), object(16) memory usage: 12.0+ MB
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] train_data.shape
[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:
{},codes*=] (59400, 20)
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] cat_df = train_data.select_dtypes(include='object')
cat_df.info()
[commandchars=
{},codes*=] ¡class 'pandas.core.frame.DataFrame'¿ Int64Index: 59400 entries, 0 to 59399 Data columns (total 16 columns): # Column Non-Null Count Dtype — —— ————— —— 0 funder 59400 non-null object 1 installer 59400 non-null object 2 basin 59400 non-null object 3 public_meeting 59400 non-null object 4 scheme_management 59400 non-null object 5 permit 59400 non-null object 6 construction_year 59400 non-null object 7 extraction_type_class 59400 non-null object 8 payment_type 59400 non-null object 9 quality_group 59400 non-null object 10 quantity 59400 non-null object 11 source_type 59400 non-null object 12 source_class 59400 non-null object 13 waterpoint_type 59400 non-null object 14 waterpoint_type_group 59400 non-null object 15 status_group 59400 non-null object dtypes: object(16) memory usage: 10.2+ MB
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] mca = prince.MCA(n_components=2, n_iter=3, copy=True, check_input=True, engine='auto', random_state=42) churn_mca = mca.fit(cat_df)
ax = churn_mca.plot_coordinates(X=cat_df, ax=None, figsize=(8, 10), ↪ show_row_points=False, row_points_size=0, show_row_labels=False, ↪ show_column_points=True, column_points_size=10, show_column_labels=False, legend_n_cols=1).legend(loc='center left', bbox_to_anchor=(1, 0.5))

max size=0.90.9output_99_0.*png*

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]:

[commandchars=
{},codes*=] churn2 = pd.get_dummies(cat_df, drop_first=False)
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-
fram[commandchars=
{},codes*=] print(churn2.shape)
[commandchars=
{},codes*=] (59400, 99)
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-
fram[commandchars=
{},codes*=] print("The data set contains: {} rows and {} columns".format(churn2.
↪shape[0], churn2.shape[1])) print("Features after get_dummies:\n", list(churn2.
↪columns))
[commandchars=
{},codes*=] The data set contains: 59400 rows and 99 columns Features after get_dummies:
['funder_Dis_Council', 'funder_Kkkt', 'funder_Tasaf', 'funder_Unicef', 'funder_danida', 'funder_gov',
↪ 'funder_hesawa', 'funder_other', 'funder_rwssp', 'funder_world_bank', 'installer_Hesawa',
↪ 'installer_commu', 'installer_danida', 'installer_dwe', 'installer_gov', 'installer_other', 'in-
staller_rwe', 'basin_Internal', 'basin_Lake Nyasa', 'basin_Lake Rukwa', 'basin_Lake Tanganyika',
↪ 'basin_Lake Victoria', 'basin_Pangani', 'basin_Rufiji', 'basin_Ruvuma / Southern Coast',
'basin_Wami / Ruvu', 'public_meeting_False', 'public_meeting_True', 'public_meeting_Unknown',
↪ 'scheme_management_Parastatal', 'scheme_management_comp', 'scheme_management_other',
'scheme_management_pri_opr', 'scheme_management_vwc', 'scheme_management_wtr_auth',
↪ 'scheme_management_wtr_brd', 'scheme_management_wua', 'scheme_management_wug',
↪ 'permit_False', 'permit_True', 'permit_Unknown', 'construction_year_00s', 'construc-
tion_year_10s', 'construction_year_60s', 'construction_year_70s', 'construction_year_80s', 'con-
struction_year_90s', 'construction_year_unknown', 'extraction_type_class_gravity', 'extrac-
tion_type_class_handpump', 'extraction_type_class_motorpump', 'extraction_type_class_other',
'extraction_type_class_rope pump', 'extraction_type_class_submersible', 'extraction_type_class_wind-
powered', 'payment_type_annually', 'payment_type_monthly', 'payment_type_never pay',
↪ 'payment_type_on failure', 'payment_type_other', 'payment_type_per bucket', 'pay-
ment_type_unknown', 'quality_group_colored', 'quality_group_fluoride', 'quality_group_good', 'qual-
ity_group_milky', 'quality_group_salty', 'quality_group_unknown', 'quantity_dry', 'quantity_enough',
↪ 'quantity_insufficient', 'quantity_seasonal', 'quantity_unknown', 'source_type_borehole',
'source_type_dam', 'source_type_other', 'source_type_rainwater harvesting', 'source_type_river/lake',
'source_type_shallow well', 'source_type_spring', 'source_class_groundwater', 'source_class_surface',
↪ 'source_class_unknown', 'waterpoint_type_cattle trough', 'waterpoint_type_communal
standpipe', 'waterpoint_type_communal standpipe multiple', 'waterpoint_type_dam', 'water-
point_type_hand pump', 'waterpoint_type_improved spring', 'waterpoint_type_other', 'wa-
terpoint_type_group_cattle trough', 'waterpoint_type_group_communal standpipe', 'water-
point_type_group_dam', 'waterpoint_type_group_hand pump', 'waterpoint_type_group_improved
spring', 'waterpoint_type_group_other', 'status_group_functional', 'status_group_functional needs
repair', 'status_group_non functional']
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-
fram[commandchars=
{},codes*=] y = churn2[['status_group_functional','status_group_functional
needs repair', 'status_group_non functional']] # target variable X = churn2.

```
↪drop(['status_group_functional','status_group_functional needs repair',
'status_group_non functional'],axis=1) # input categorical features
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-
fram[commandchars=
{},codes*=]
```
{},codes*=] # categorical feature selection sf = SelectKBest(chi2, k='all')
sf_fit = sf.fit(X, y) # print feature scores for i in range(len(sf_fit.scores_)):
print(' %s: %f' % (X.columns[i], sf_fit.scores_[i]))
```

[commandchars=
{},codes*=] funder_Dis_Council: 46.735754 funder_Kkkt: 8.958172 funder_Tasaf: 1.443570 funder_Unicef: 13.285733 funder_danida: 22.032436 funder_gov: 694.831216 funder_hesawa: 130.
↪558801 funder_other: 288.658764 funder_rwssp: 14.216822 funder_world_bank: 116.721629 installer_Hesawa: 34.531385 installer_commu: 90.749346 installer_danida: 3.135005 installer_dwe:
↪ 118.209377 installer_gov: 482.555852 installer_other: 104.688615 installer_rwe: 412.146674 basin_Internal: 35.530079 basin_Lake Nyasa: 253.612197 basin_Lake Rukwa: 191.795267 basin_Lake Tanganyika: 208.953087 basin_Lake Victoria: 131.441654 basin_Pangani: 135.416944 basin_Rufiji:
↪ 275.321373 basin_Ruvuma / Southern Coast: 586.991201 basin_Wami / Ruvu: 101.972423 public_meeting_False: 261.956384 public_meeting_True: 44.171404 public_meeting_Unknown:
↪ 77.871905 scheme_management_Parastatal: 83.062192 scheme_management_comp:
↪ 40.577562 scheme_management_other: 99.258515 scheme_management_pri_opr: 101.
↪218791 scheme_management_vwc: 226.590779 scheme_management_wtr_auth: 227.
↪007774 scheme_management_wtr_brd: 460.901752 scheme_management_wua: 311.849334 scheme_management_wug: 350.742176 permit_False: 48.124954 permit_True: 21.535402 permit_Unknown: 34.520204 construction_year_00s: 744.848831 construction_year_10s: 767.
↪116997 construction_year_60s: 149.610783 construction_year_70s: 952.646615 construction_year_80s: 509.414919 construction_year_90s: 4.794637 construction_year_unknown: 116.
↪993879 extraction_type_class_gravity: 944.183000 extraction_type_class_handpump: 507.
↪205685 extraction_type_class_motorpump: 454.509639 extraction_type_class_other: 4887.
↪351064 extraction_type_class_rope pump: 23.046671 extraction_type_class_submersible: 106.
↪720369 extraction_type_class_wind-powered: 8.185213 payment_type_annually: 690.811137 payment_type_monthly: 912.896916 payment_type_never pay: 969.864077 payment_type_on failure: 101.724691 payment_type_other: 40.775107 payment_type_per bucket: 661.952180 payment_type_unknown: 587.595296 quality_group_colored: 11.030404 quality_group_fluoride: 29.
↪367244 quality_group_good: 157.385495 quality_group_milky: 39.795102 quality_group_salty: 211.116173 quality_group_unknown: 1651.442169 quantity_dry: 9025.547856 quantity_enough: 1753.276043 quantity_insufficient: 123.120145 quantity_seasonal: 96.710532 quantity_unknown: 362.113320 source_type_borehole: 388.352642 source_type_dam: 105.611828 source_type_other:
↪ 12.394751 source_type_rainwater harvesting: 238.862746 source_type_river/lake: 359.500214 source_type_shallow well: 309.980683 source_type_spring: 492.094549 source_class_groundwater: 125.
↪623827 source_class_surface: 452.244709 source_class_unknown: 12.394751 waterpoint_type_cattle trough: 16.671130 waterpoint_type_communal standpipe: 875.596693 waterpoint_type_communal standpipe multiple: 772.248103 waterpoint_type_dam: 2.841572 waterpoint_type_hand pump: 395.091117 waterpoint_type_improved spring: 148.651601 waterpoint_type_other: 5239.216380 waterpoint_type_group_cattle trough: 16.671130 waterpoint_type_group_communal standpipe: 312.302418 waterpoint_type_group_dam: 2.841572 waterpoint_type_group_hand pump: 395.091117 waterpoint_type_group_improved spring: 148.651601 waterpoint_type_group_other: 5239.216380

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-

```
{},codes*=] # plot the scores datset = pd.DataFrame() datset['feature'] = X.
 ↪columns[ range(len(sf_fit.scores_))] datset['scores'] = sf_fit.scores_ datset =
datset.sort_values(by='scores', ascending=False) sns.barplot(datset['scores'][0:
 ↪25], datset['feature'][0:25], color='blue') sns.set_style('whitegrid') plt.
 ↪ylabel('Categorical Feature', fontsize=18) plt.xlabel('Score', fontsize=18)
plt.show()
```

max size=0.90.9output$_1$05$_0$.*png*

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-

```
{},codes*=] feature_list = [i for i in datset['feature'][0:25]] #num_cols =
['amount_tsh', 'days_since_recorded','population']
   #for j in num_cols: #feature_list.append(j)
   #feature_list
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-

```
{},codes*=] datset['feature'][0:25]
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
{},codes*=] 68 quantity_dry 95 waterpoint_type_group_other 89 waterpoint_type_other
51 extraction_type_class_other 69 quantity_enough 67 quality_group_unknown 57
payment_type_never pay 44 construction_year_70s 48 extraction_type_class_gravity
56 payment_type_monthly 84 waterpoint_type_communal standpipe 85
waterpoint_type_communal standpipe multiple 42 construction_year_10s 41
construction_year_00s 5 funder_gov 55 payment_type_annually 60 payment_type_per
bucket 61 payment_type_unknown 24 basin_Ruvuma / Southern Coast 45
construction_year_80s 49 extraction_type_class_handpump 79 source_type_spring 14
installer_gov 35 scheme_management_wtr_brd 50 extraction_type_class_motorpump Name:
feature, dtype: object
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-

```
{},codes*=] train_data['status_group_vals'].value_counts()
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
{},codes*=] 2 32259 0 22824 1 4317 Name: status_group_vals, dtype: int64
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-

```
{},codes*=] train_data.drop('status_group',inplace=True,axis=1)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-

```
{},codes*=] data_train, data_test = train_test_split(train_data, test_size=0.25,
 ↪random_state=1443,stratify=train_data.status_group_vals)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-

```
{},codes*=] X_train1 = data_train y_train = data_train['status_group_vals']
   X_test1 = data_test y_test = data_test['status_group_vals']
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] X_train = pd.get_dummies(X_train1) X_train.drop('status_group_vals',
↪inplace=True,axis=1) X_test = pd.get_dummies(X_test1) X_test.
↪drop('status_group_vals',inplace=True,axis=1)

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] X_train.head(5)

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:
{},codes*=] amount_tsh population operational_years funder_Dis_Council \ 40630 0.0
96 50 0 36440 0.0 0 0 0 25799 0.0 0 0 0 24265 0.0 1 13 0 1462 0.0 0 0 0
funder_Kkkt funder_Tasaf funder_Unicef funder_danida funder_gov \ 40630 0 0 0 0
1 36440 0 0 0 0 0 25799 0 0 0 0 0 24265 0 0 0 0 0 1462 0 0 0 0 0
funder_hesawa ... waterpoint_type_dam waterpoint_type_hand pump \ 40630 0 ...
0 0 36440 0 ... 0 0 25799 0 ... 0 1 24265 0 ... 0 0 1462 0 ... 0 0
waterpoint_type_improved spring waterpoint_type_other \ 40630 0 0 36440 1 0
25799 0 0 24265 0 0 1462 0 1
waterpoint_type_group_cattle trough \ 40630 0 36440 0 25799 0 24265 0 1462 0
waterpoint_type_group_communal standpipe waterpoint_type_group_dam \ 40630 1 0
36440 0 0 25799 0 0 24265 1 0 1462 0 0
waterpoint_type_group_hand pump waterpoint_type_group_improved spring \ 40630 0
0 36440 0 1 25799 1 0 24265 0 0 1462 0 0
waterpoint_type_group_other 40630 0 36440 0 25799 0 24265 0 1462 1
[5 rows x 99 columns]

gaussian model

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] gnb = GaussianNB()

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] gnb.fit(X_train, y_train)

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:
{},codes*=] GaussianNB()

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] y_pred = gnb.predict(X_test)

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

[commandchars=
{},codes*=] Accuracy: 0.6270707070707071

Multinomial

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
{},codes*=] clf = MultinomialNB()

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
[commandchars=
{},codes*=] clf.fit(X_train, y_train)
[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:
[commandchars=
{},codes*=] MultinomialNB()
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
[commandchars=
{},codes*=] y_pred = clf.predict(X_test)
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
[commandchars=
{},codes*=] print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
[commandchars=
{},codes*=] Accuracy: 0.4606734006734007
Random Forest
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
[commandchars=
{},codes*=] from sklearn.ensemble import RandomForestClassifier
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
[commandchars=
{},codes*=] model = RandomForestClassifier(n_estimators=200,max_depth=50,
random_state=14400)
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
[commandchars=
{},codes*=] model.fit(X_train, y_train)
[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:
[commandchars=
{},codes*=] RandomForestClassifier(max_depth=50, n_estimators=200,
random_state=14400)
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
[commandchars=
{},codes*=] y_pred = model.predict(X_test)
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
[commandchars=
{},codes*=] print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
[commandchars=
{},codes*=] Accuracy: 0.7835016835016835
[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]
[commandchars=
{},codes*=] from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix,precision_score,recall_score
def calculate_metrics(y_test, y_pred, y_prob=None, sample_weights=None):
"""Cacluate model performance metrics"""
Dictionary of metrics to calculate metrics = {} metrics['confusion_matrix']
= confusion_matrix(y_test, y_pred, sample_weight=sample_weights)
metrics['accuracy'] = accuracy_score(y_test, y_pred, sample_weight=sample_weights)
metrics['precision'] = precision_score(y_test, y_pred, sample_weight=sample_weights,

```
↪average='weighted') metrics['recall'] = recall_score(y_test, y_pred,
sample_weight=sample_weights,average='weighted')
   return metrics
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] def conf_mat(metrics): '''plot and display the confusion matrix'''
array = metrics['confusion_matrix'] classes=['functional','functional need
repair','non functional'] df_cm = pd.DataFrame(array, index = [i for i in
classes], columns = [i for i in classes]) plt.figure(figsize = (10,7)) sn.
↪heatmap(df_cm, annot=True,cmap="Blues") plt.show()
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] metrics = calculate_metrics(y_test, y_pred) conf_mat(metrics)
```

max size=0.90.9output$_1$32$_0$.*png*

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] def pltcolor(lst): cols=[] for l in lst: if l==0: cols.append('green')
elif l==1: cols.append('yellow') elif l==2: cols.append('red')
   return cols # Create the colors list using the function above
cols_pred=pltcolor(y_pred) cols_true=pltcolor(y_test)
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] ax1=plt.subplot(1,2,1) ax1.scatter(X_test1['amount_tsh'],
↪X_test1['population'],c = cols_pred,s=30) ax1.set_xlim(0,1200) ax1.
↪set_ylim(0,12000) ax2=plt.subplot(1,2,2) ax2.scatter(X_test1['amount_tsh'],
↪X_test1['population'],c = cols_true,s=30) ax2.set_xlim(0,1200) ax2.set_ylim(0,
↪12000)
```

[breakable, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:

```
{},codes*=] (0, 12000)
```

max size=0.90.9output$_1$34$_1$.*png*

Naive bayes classifier assuming its a Bernaoulli distribution

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] from sklearn.naive_bayes import BernoulliNB
```

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] clf1 = BernoulliNB() test_error = [] feature_num = [5,10,
↪15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,99] data_train,
↪ data_test = train_test_split(train_data, test_size=0.25,random_state=1443,
↪stratify=train_data.status_group_vals) X_train1 = data_train
y_train = data_train['status_group_vals'] X_test1 = data_test y_test =
```

```
data_test['status_group_vals'] X_train1.drop('status_group_vals',inplace=True,
 ↪axis=1) X_test1.drop('status_group_vals',inplace=True,axis=1) #X_train1.
 ↪drop(['amount_tsh','operational_years','population'], inplace = True,axis=1)
#X_test1.drop(['amount_tsh','operational_years','population'], inplace = True,
 ↪axis=1)
   for i in feature_num: feature_list = [k for k in datset['feature'][0:i]]
#num_cols = ['amount_tsh','operational_years','population'] #for j in num_cols:
#feature_list.append(j)
   # selecting best catagories X_train = pd.get_dummies(X_train1) X_train =
X_train[feature_list]
   X_test = pd.get_dummies(X_test1) X_test = X_test[feature_list]
   clf1.fit(X_train, y_train)
   y_pred = clf1.predict(X_test) test_error.append(metrics.accuracy_score(y_test,
y_pred))
   fig = plt.figure() ax = plt.subplot() ax.plot(feature_num,test_error) ax.
 ↪scatter(25, 0.7169696969696969,c='red') ax.text(26,0.7169696969696969,'Features
= 25') ax.set_xlabel('Number of features') ax.set_ylabel('Model Accuracy')
#Set the figure size so that all four panel graphs are clearly visible fig.
 ↪set_size_inches(10,5) #Set a title for the figure fig.suptitle('Testing
accuracy for Bernoulli Naive Bayes'\ ,y=1,fontsize=15,fontweight='bold') plt.
 ↪show()
```

[commandchars= {},codes*=] /opt/anaconda3/lib/python3.7/site-packages/pandas/core/frame.py:3997: SettingWithCopyWarning: A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas- docs/stable/ ↪user_guide/indexing.html#returning-a-view-versus-a-copy errors=errors,

max size=0.90.9output$_1$37$_1$*.png*

selecting cotegorical feautres

errors for training and testing

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col- fram[commandchars= {},codes*=]

```
model = RandomForestClassifier(n_estimators=200,max_depth=50,
 ↪ random_state=14400) test_error = [] feature_num = [5,10,15,20,25,
 ↪30,35,40,45,50,55,60,65,70,75,80,85,90,95,99] data_train, data_test
= train_test_split(train_data, test_size=0.25,random_state=1443,
 ↪stratify=train_data.status_group_vals) X_train1 = data_train
y_train = data_train['status_group_vals'] X_test1 = data_test y_test =
data_test['status_group_vals'] X_train1.drop('status_group_vals',inplace=True,
 ↪axis=1) X_test1.drop('status_group_vals',inplace=True,axis=1)
   for i in feature_num: feature_list = [k for k in datset['feature'][0:i]]
num_cols = ['amount_tsh', 'operational_years','population'] for j in num_cols:
feature_list.append(j)
   # selecting best catagories X_train = pd.get_dummies(X_train1) X_train =
X_train[feature_list]
```

```
   X_test = pd.get_dummies(X_test1) X_test = X_test[feature_list]
   model.fit(X_train, y_train)
   y_pred = model.predict(X_test) test_error.append(metrics.accuracy_score(y_test,
y_pred))
   plt.plot(feature_num,test_error)
   [commandchars=
{},codes*=]              /opt/anaconda3/lib/python3.7/site-packages/pandas/core/frame.py:3997:
SettingWithCopyWarning: A value is trying to be set on a copy of a slice from a DataFrame
   See the caveats in the documentation: https://pandas.pydata.org/pandas- docs/stable/
↪user_guide/indexing.html#returning-a-view-versus-a-copy errors=errors,
   [commandchars=, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:
{},codes*=] [<matplotlib.lines.Line2D at 0x1a1d5558d0>]
```

max size=0.90.9output$_1$40$_2$.*png*

```
   [breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-
fra[commandchars=
{},codes*=] clf = MultinomialNB() test_error = [] feature_num = [5,10,
↪15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,99] data_train,
↪ data_test = train_test_split(train_data, test_size=0.25,random_state=1443,
↪stratify=train_data.status_group_vals) X_train1 = data_train
y_train = data_train['status_group_vals'] X_test1 = data_test y_test =
data_test['status_group_vals'] X_train1.drop('status_group_vals',inplace=True,
↪axis=1) X_test1.drop('status_group_vals',inplace=True,axis=1)
   for i in feature_num: feature_list = [k for k in datset['feature'][0:i]]
num_cols = ['amount_tsh', 'days_since_recorded','population'] for j in num_cols:
feature_list.append(j)
   # selecting best catagories X_train = pd.get_dummies(X_train1) X_train =
X_train[feature_list]
   X_test = pd.get_dummies(X_test1) X_test = X_test[feature_list]
   clf.fit(X_train, y_train)
   y_pred = clf.predict(X_test) test_error.append(metrics.accuracy_score(y_test,
y_pred))
   plt.plot(feature_num,test_error)
   [commandchars=
{},codes*=]              /opt/anaconda3/lib/python3.7/site-packages/pandas/core/frame.py:3997:
SettingWithCopyWarning: A value is trying to be set on a copy of a slice from a DataFrame
   See the caveats in the documentation: https://pandas.pydata.org/pandas- docs/stable/
↪user_guide/indexing.html#returning-a-view-versus-a-copy errors=errors,
   [commandchars=, size=fbox, boxrule=.5pt, pad at break*=1mm, opacityfill=0]:
{},codes*=] [<matplotlib.lines.Line2D at 0x1a18cecd10>]
```

max size=0.90.9output$_1$41$_2$.*png*

```
   [breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-
fra[commandchars=
{},codes*=] from sklearn.model_selection import train_test_split from
```

```
sklearn.model_selection import GridSearchCV from sklearn.ensemble import
GradientBoostingClassifier
   [breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-
fram[commandchars=
{},codes*=] def model(X_train, X_test, y_train, y_test): if __name__ ==
'__main__':
   param_grid = {'learning_rate': [0.075, 0.7], 'max_depth': [13, 14],
 ↪ 'min_samples_leaf': [15, 16], 'max_features': [1.0], 'n_estimators': [100,
200]}
   estimator = GridSearchCV(estimator=GradientBoostingClassifier(),
param_grid=param_grid, n_jobs=-1)
   estimator.fit(X_train, y_train)
   best_params = estimator.best_params_
   print (best_params)
   y_pred = estimator.predict(X_test) metrics = calculate_metrics(y_test, y_pred)
conf_mat(metrics) return metrics
   [breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-
fram[commandchars=
{},codes*=] grad=model(X_train, X_test, y_train, y_test)
   [commandchars=
{},codes*=] {'learning_rate': 0.075, 'max_depth': 14, 'max_features': 1.0, 'min_samples_leaf': 15,
'n_estimators': 100}
```

max size=0.90.9output$_1$44$_1$.*png*

```
   [breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-
fram[commandchars=
{},codes*=] print("acurracy for gradient boosting classifier",grad)
   [commandchars=
{},codes*=] acurracy for gradient boosting classifier 0.7901683501683502
   [breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-
fram[commandchars=
{},codes*=] def conf_mat(metrics): '''plot and display the confusion matrix'''
array = metrics['confusion_matrix'] classes=['functional','functional need
repair','non functional'] df_cm = pd.DataFrame(array, index = [i for i in
classes], columns = [i for i in classes]) plt.figure(figsize = (10,7)) sn.
 ↪heatmap(df_cm, annot=True,cmap="Blues") plt.show()
   [breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-
fram[commandchars=
{},codes*=] metrics = calculate_metrics(grad) conf_mat(metrics)
   [breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, col-
fram[commandchars=
{},codes*=] import imblearn from imblearn.over_sampling import SMOTE ###
class Balance Oversampling from imblearn.over_sampling import SMOTE # Apply
oversampling with SMOTE X_train, y_train = SMOTE().fit_sample(X_train, y_train)
print("X shape after oversampling: ", X_train.shape)
```

[commandchars=
{},codes*=] X shape after oversampling: (72582, 99)

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=] sns.countplot(x=y_train,palette="pastel", edgecolor=".6")
```

[commandchars=
{},codes*=] <matplotlib.axes._subplots.AxesSubplot at 0x1a2c71d050>

max size=0.90.9output$_1$49$_1$.*png*

[breakable, size=fbox, boxrule=1pt, pad at break*=1mm,colback=cellbackground, colframe=cellborder]

```
{},codes*=]
```