UWS
2020_2021

Advanced Data Science

PROJECT REPORT

GEORGES TAING B00398521

PROFESSOR |Zeeshan Pervez

Contents

Part A:

Chosen Dataset:
https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/page/25/

Link to the Colab Notebook:
https://colab.research.google.com/drive/1goQ2utIQLS9QKkIQSWPf7snGPgUXsQmE?usp=sharing

The dataset is hosted by the website DrivenData which provides challenges in Data Science for students or professional. The competition is entitled “Pump it up: Data Mining the Water Table”: our goal is to predict the operating condition of a waterpoint for each record in the Tanzania dataset.

This is a classification problem where the target is the attribute “status_group” which values are:

- Functional (the waterpoint is operational, no repairs needed)
- Functional needs repair (the waterpoint is operational but repairs are needed)
- Non functional (the waterpoint is not operational)

I chose this dataset because I found it relevant to the UWS course Advanced Data Science and it will also allow me to submit this project to see my performance in the competition. I believe that this project would be an asset in my portfolio as a junior engineer.

Data Exploration

Statistical information

To obtain a quick statistical description of the numerical features of the data, we can use the method describe() with DataFrame objects.

It gives us information on each numerical feature such as the mean, the standard-deviation, the min-max values and the quartiles…

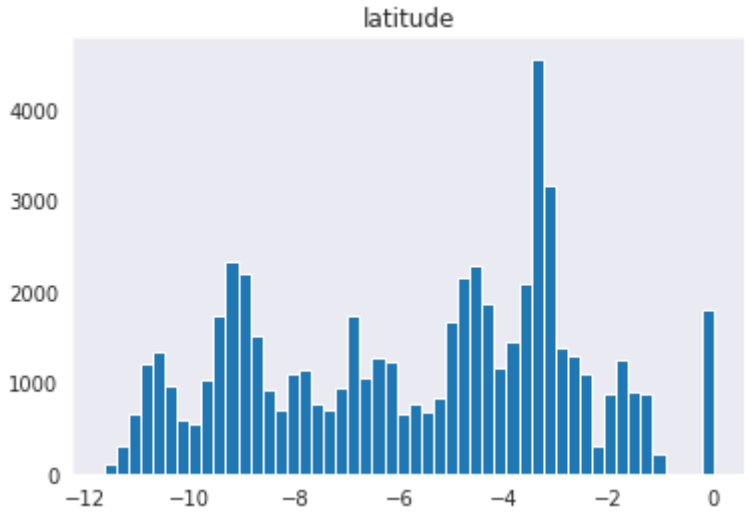
Most frequent values

To know the most frequent values, we just needed to use the methods pd.value_counts() and pd.head()

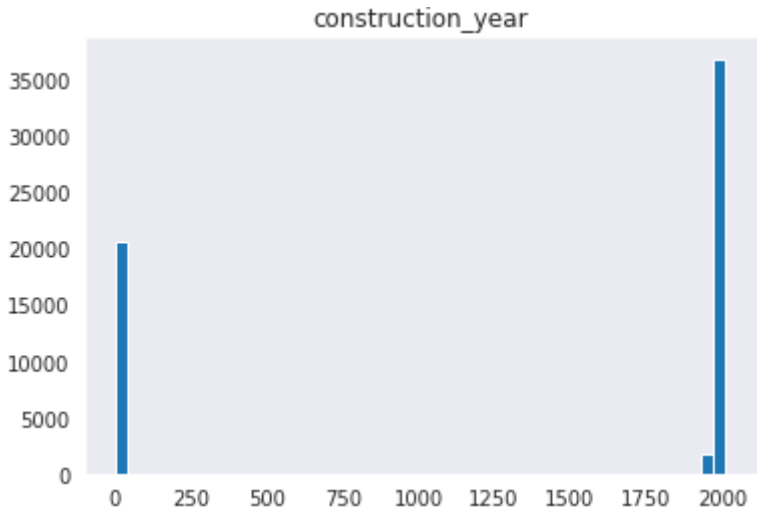
Indeed, value_counts() returns the different values of a feature and the number of repetitions for each value.

Plotting numerical features

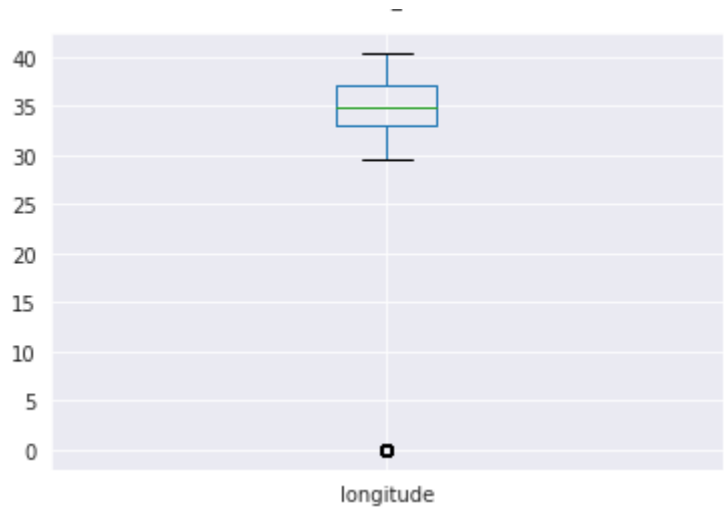
In order to observe the distribution of our numerical values, we used the method hist() to plot histograms of the features.



Observing outliers (numerical features)

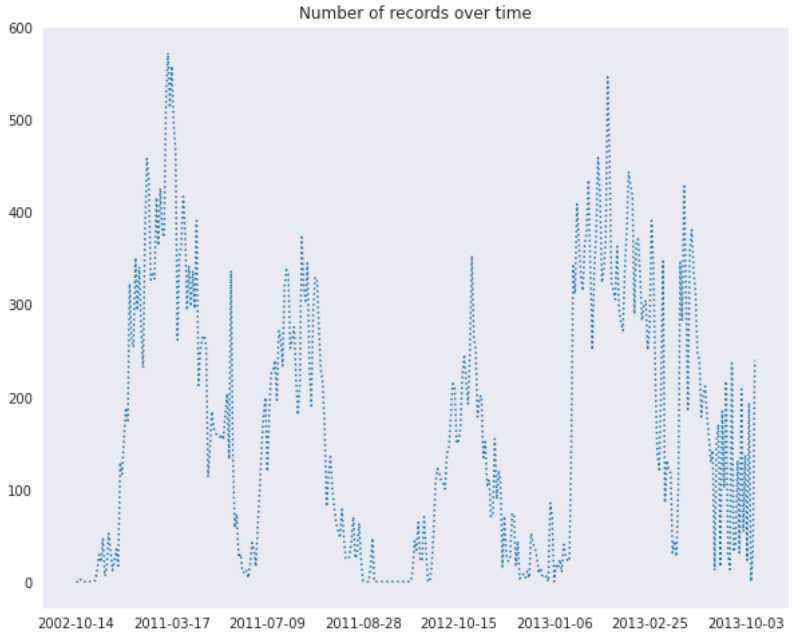


We can highlight outliers by printing boxplots of the features, we called the boxplot() method on each column to obtain them.

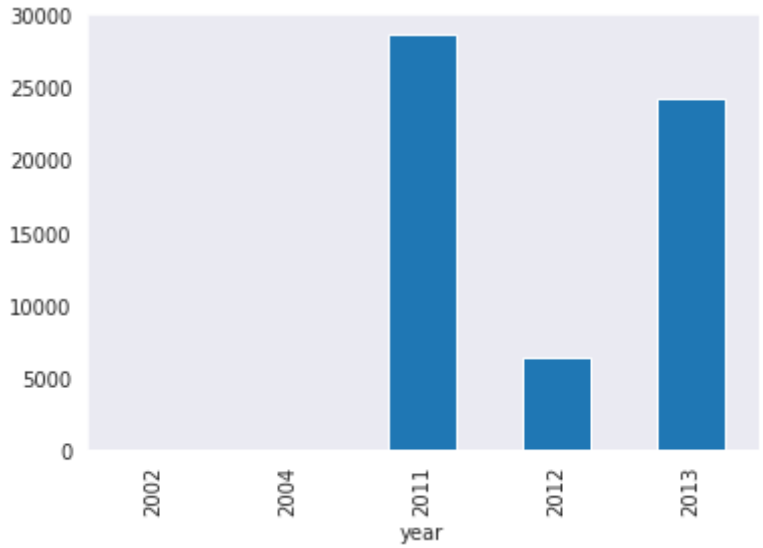


Date/time data

To handle data/time data, we used pd.to_datetime() and the various attributes dt.month, dt.year to have our values in the appropriate format.

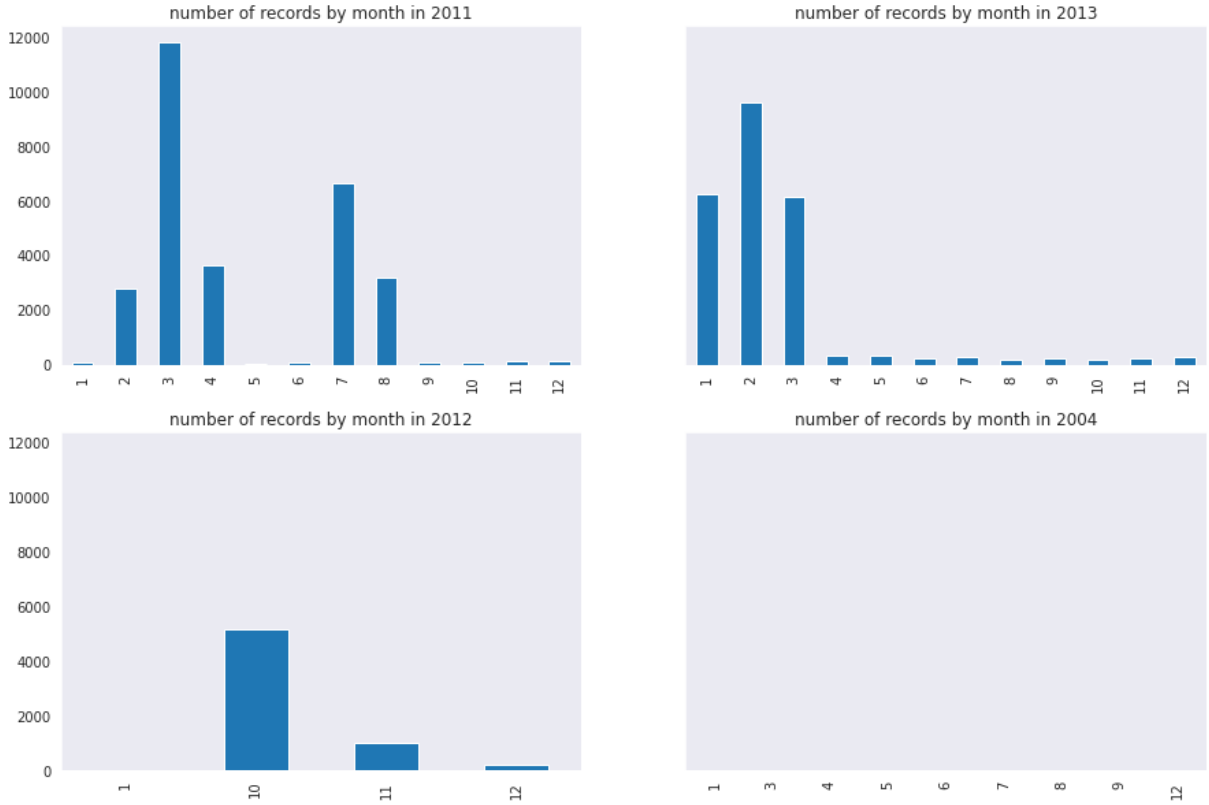


We could also print the bar chart of the number of records by year (or month) by using the the groupby() method for the DataFrame class. (plot(kind='bar'…))



Plotting many plots in one output

We can call the subplots() method to create a figure with different plots in it.

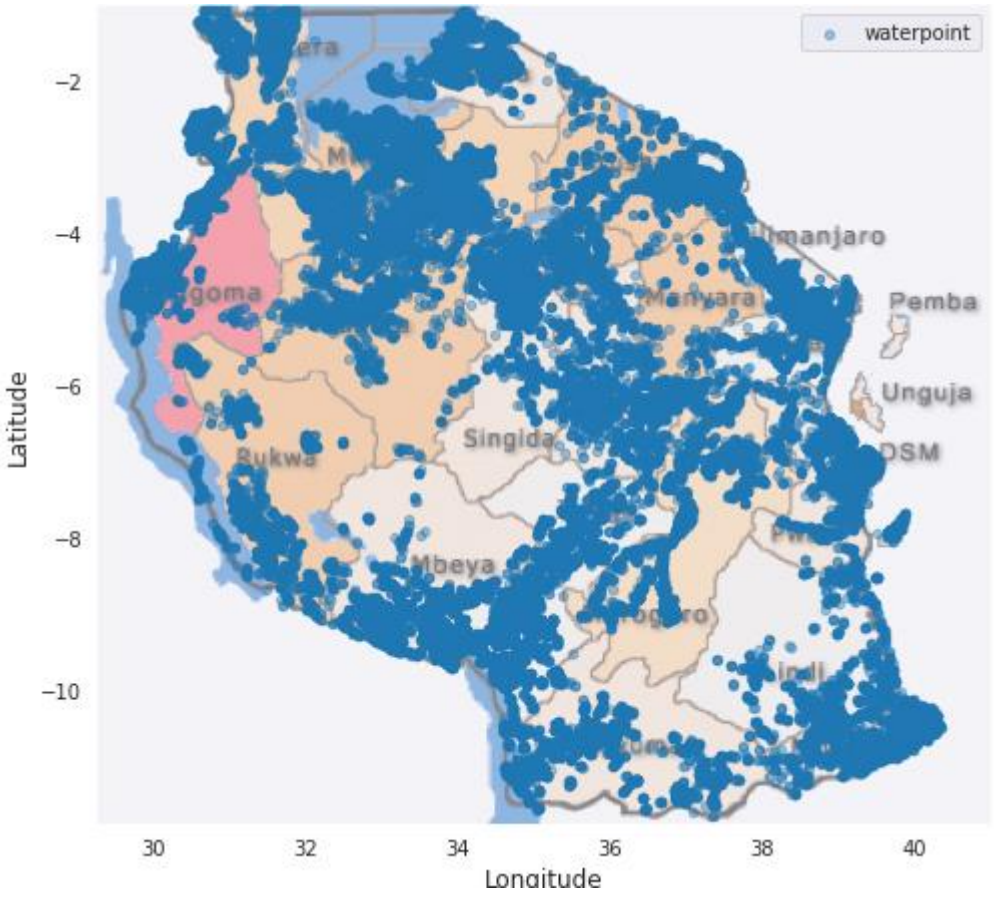


To filter some rows in a dataset and counting them:

With the Numpy library, we use np.where() to filter rows depending on a specific criteria. The sum() method helped us in counting the number of rows that verified the criteria.

Geographical data

We mainly used scatterplots and the imshow() to print the waterpoints on a map of Tanzania.



Data pre-processing

Creating test-train sets

To split our dataset into train and test sets, Sickit-Learn provides a function train_test_split() to handle that. It can also split the data while considering stratification: it means that the train and test sets will have the same repartition of the target variable for example.

Data cleaning

To have an overall view of our dataset, we can call pd.info() to have the different columns, their types and the number of non-null values. It is an easy way to see missing values in the dataset.

```
train_set.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 47520 entries, 3607 to 56422
Data columns (total 19 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   amount_tsh             47520 non-null  float64
 1   gps_height             47520 non-null  int64
 2   longitude              47520 non-null  float64
 3   latitude               47520 non-null  float64
 4   population             47520 non-null  int64
 5   construction_year      47520 non-null  int64
 6   basin                  47520 non-null  object
 7   permit                 45081 non-null  object
 8   public_meeting         44831 non-null  object
 9   scheme_management      44418 non-null  object
 10  extraction_type_class  47520 non-null  object
 11  management_group       47520 non-null  object
 12  payment                47520 non-null  object
 13  quality_group          47520 non-null  object
 14  quantity_group         47520 non-null  object
 15  source_class           47520 non-null  object
 16  source_type            47520 non-null  object
 17  waterpoint_type_group  47520 non-null  object
 18  status_group           47520 non-null  object
dtypes: float64(3), int64(3), object(13)
memory usage: 7.3+ MB
```

Our pre-processing stage was divided into two parts to handle first the numerical features and then the categorical features.

Numerical features

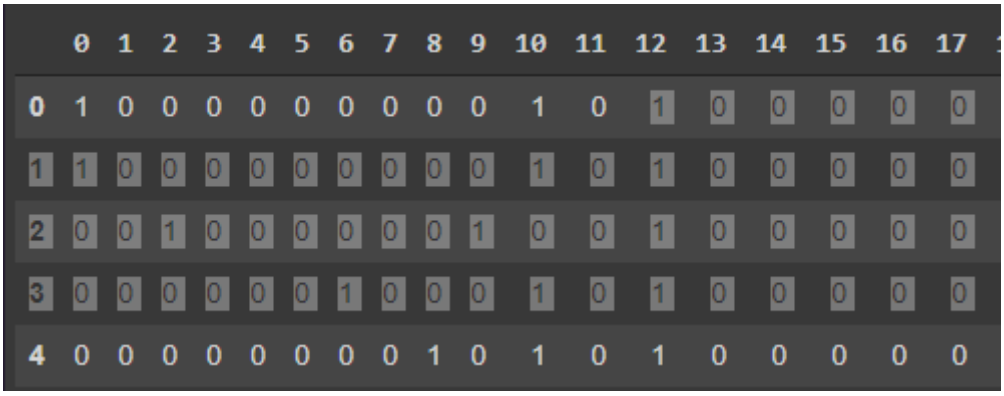
To drop the null values for the ‘longitude’ for example. We could have used drop() but here we could also use filtering: Df[Df[‘longitude’] != 0]

An important part in preparing numerical features is scaling. Indeed, many algorithms work better (and need it sometimes) with scaled data. To do that, we use a RobustScaler() from Sickit-Learn because the data were quite spread and a SimpleScaler is sensitive to outliers. In our case, even if we did take care of null values, there are some features that have a wide range of values which are not necessarily outliers.

Categorical features

Usually, it is preferable to convert categorical features into numerical features because most algorithms need numerical inputs. This is when encoding takes place, it consists to transform categorical data into numerical data.

For the input features (X_train), we used OneHotEncoding technique through Sickit-Learn OneHotEncoder().



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0

For the target variable ('status_group'), we used LabelEncoding to convert the values ['functional', 'functional needs repair', 'non functional'] to [0, 1, 2].

For the missing values, we filled them with the mode (the most frequent value for a feature) by using a SimpleImputer().

Finally, we created a pipeline to process data with all the various pre-processing techniques that we mentioned previously. Pipelines are crucial in Data Science in order to help automate machine learning workflows. The data are transformed through a sequence of operations (scaling, imputing, filling missing values…).

Data Modelling

Before passing data to algorithms, we needed to binarize the target. To do so, we used the method label_binarize().

To observe and evaluate the performances of our models, we mainly used cross-validation with cross_val_score() and cross_val_predict(). Cross-validation is convenient because it uses our data more efficiently by creating subsets for testing-training models.

Because this is a classification task, the metrics that we considered were the accuracy, the confusion matrix and the ROC Curve.
Sickit-Learn provides a scoring input for cross_val_score() and various functions to compute the ROC Curve and the confusion matrix (roc_curve() and confusion_matrix()).

Classifiers that we considered for this problem

SVC (Support Vector Classifier)

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points. there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes

RandomForest

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction

SGD

Stochastic Gradient Descent (SGD) classifier basically implements a plain SGD learning routine supporting various loss functions and penalties for classification. Stochastic Gradient Descent (SGD) is a simple, yet efficient optimization algorithm used to find the values of parameters/coefficients of functions that minimize a cost function. In other words, it is used for discriminative learning of linear classifiers under convex loss functions such as SVM and Logistic regression

Logistic Regression

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression)

Part B

Setup:

- You need to change the links to the files (csv and map of Tanzania)
- Install data science libraries if needed (sickit-learn, pandas, numpy, etc)

Results/findings:

Please, open the Colab Notebook with the following to see a detailed analysis of our results: https://colab.research.google.com/drive/1goQ2utIQLS9QKkIQSWPf7snGPgUXsQmE?usp=sharing

Comparison of the models

Model	SVC	RandomForest	SGD	Logistic Regression
Cross_val_score (accuracy)	0.63 (63 %)	0.75 (75%)	0.52 (52%)	0.54 (54%)

The best model that we have for the moment is the Random Forest Classifier.

ROC Curve of the RandomForestClassifier