Australian Weather Prediction

Marta Matute - 1496672
David Pacheco - 1565824

Abstract

In this report we will be analyzing data on different meteorological phenomena in different locations in Australia, to be able to determine if it will rain or not. To do so, we will first perform an exploratory data analysis and some feature engineering, including outliers and null values removal, among other techniques. Next we will train our data using several different models, from logistic regression to a random forest classifier, and choose the best one basing ourselves on numerous criteria.

Contents

1 Introduction

In this project we will be working with a dataset[1] assembled by Jeffrey Jyhuang. The former contains the information about different weather phenomena (such as pressure, rainfall, and wind among many others) in different cities of Australia, and through the span of several years (from 1944 to 2016). This data will hopefully help us create a model to predict the possibility of rain on the day after the parameters are taken.

This paper will explain in detail all the steps and all the techniques we've employed in order to analyse and predict as correctly as possible, our target attribute (existence of rain or not). We'll be starting by an extensice Exploratory Data Analysis (EDA) and preprocessing, that will mainly focus on transforming the data so it gains usability. Next, we'll try different models to predict our target, and later asses the results using cross validation. Finally we will explain the hyperparameters used.

2 Exploratory Data Analysis

2.1 Attributes' classification

In this section we will thoroughly look through all of the attributes in our dataset and we will determine the importance of them in relation to the target attribute.

Firstly, let's paint a broad picture of our dataset. We count with a total of 145 460 rows, and 23 columns, i.e. 145 460 instances and 23 variables. These variables are

`Date`: Indicates the date in which the date from the instance was collected. It's formatted as YYYY-MM-DD.

`Location`: Shows the name of the Australian city from which we have the data.

`MinTemp`: As the name implies, indicates the minimum recorded temperature of the day in case.

`MaxTemp` : Similarly, indicates the maximum recorded temperature of the day in case.

`Rainfall` : It numerically indicates the cm^3 of rain mesured using a rain gauge.

`Evaporation` : Once again numerically, indicates the rate of evaporation assumably using a Class A evaporation pan.

`Sunshine` : Measures the amount of sunlight with the units Kilowatt hour per square meter (kWh/m^2).

`WindGustDir` : Records the maximum wind gust direction of the day.

`WindGustSpeed` : Similarly, records the maximum wind gust speed of the day.

`WindDir9am` / `WindDir3pm` : Shows the wind direction at 9am and 3pm respectively.

`Humidity9am` / `Humidity3pm` : Percentage for the relative humidity at 9am and 3pm respectively.

`Pressure9am` / `Pressure3pm` : Numerical measure for pressure at 9am and 3pm respectively.

`Cloud9am` / `Cloud3pm` : Cloud amount measured in *oktas* (a.k.a. *eighths*) at 9am and 3pm respectively.

`Temp9am` / `Temp3pm` : The temperature in celsius at 9am and 3pm respectively.

`RainToday` : Binary variable that indicates the presence or absence of rain on the day it was recorded.

`RainTomorrow` : Binary variable that indicates the presence or absence of rain on the day after the data of the same row was recorded.

It should come as no surprise that our target variable would be the prediction of `RainTomorrow`.

For the sake of classification, we can gather all these attributes into 2 cathegories: numerical and categorical. The division looks like this:

Numerical variables	Categorical Variables
`MinTemp`, `MaxTemp`, `Rainfall`, `Evaporation`, `Sunshine`, `WindGustSpeed`, `WindSpeed9am`, `WindSpeed3pm`, `Humidity9am`, `Humidity3pm`, `Pressure9am`, `Pressure3pm`, `Cloud9am`, `Cloud3pm`, `Temp9am`, `Temp3pm`, `Year`, `Month`, `Day`	`Date`, `Location`, `WindGustDir`, `WindDir9am`, `WindDir3pm`, `RainToday`, `RainTomorrow`

2.2 Target attribute

In the table above, we see that our target attribute has categorical values, these being `Yes` and `No`.

To have a better understanding of the this variable, let's see how many times does each of these possible results appear. We'll see this result both in a bar plot and a pie chart, to get a good idea of the proportions.

Bar plot 1

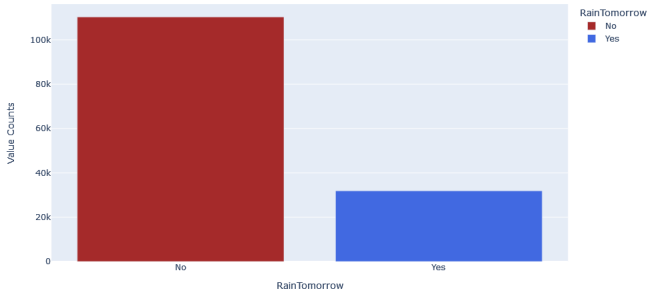


Figure 1: Bar plot for proportion of `Yes` and `No` in `RainTomorrow`

Pie chart 1

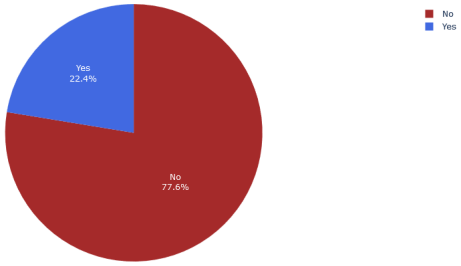


Figure 2: Pie chart for proportion of `Yes` and `No` in `RainTomorrow`

We conclude from Figures 1 and 2 that our target attributes is in fact *not* balanced. As we can see, the target `No` is more than 3 times more frequent than the target `Yes`. Since it is such a notable difference, when we perform our predictions, we will have to take this into account, oversampling or removing samples from our dataset.

2.3 Correlations

Next, we will check for any correlation between our variables, paying special attention to those who are strongly correlated in order to find relations that would help us understand our data and give some answers to help us predict the target variable. Hopefully, then we will be able to check for seasonality in some of our variables, which will help our model with the predictions.

We will be doing separately the correlation between numerical variables and the correlation between all of them (both numerical and categorical variables).

2.3.1 Correlation between numerical variables

Let's start by observing the correlation between numerical variables through a heat map.

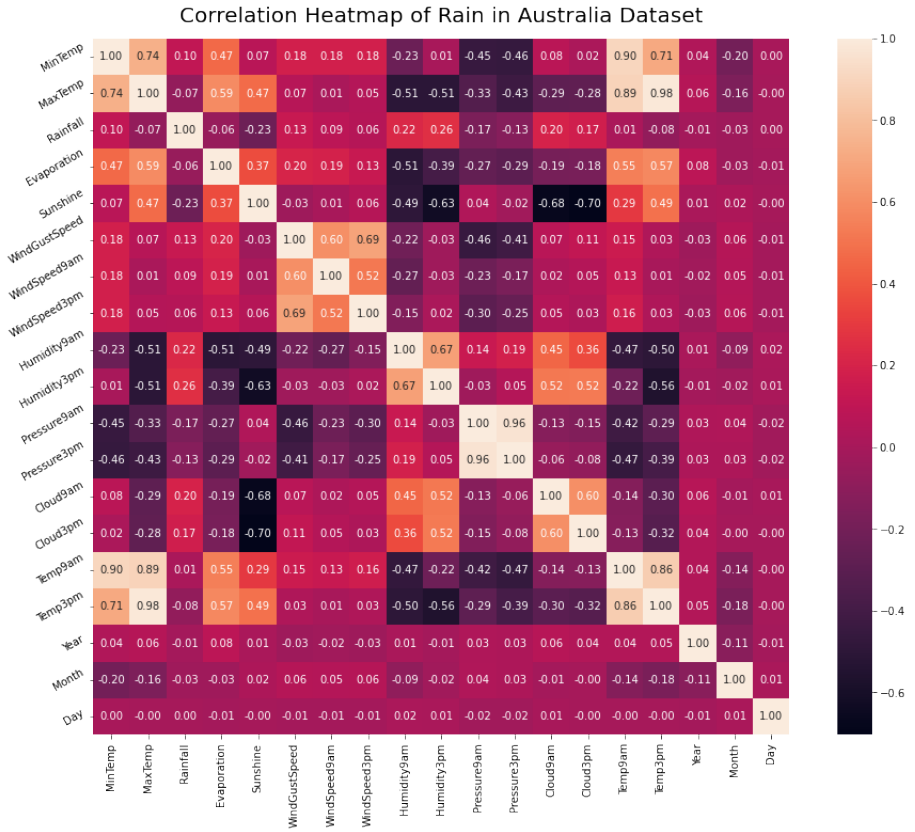


Figure 3: Heat map of correlations between numerical variables.

Despite the correlation plot show in figure 3 being very visual, we will also compute numerically the most correlated variables. For that, we will take the absolute value of the correlation between each pair, and then sort them in descending order. Additionally, we will remove correlations equal to 1 since, as we have seen in the plot, they only occur on the diagonal and that doesn't provide us with any information. In doing so, we obtain the following ranking:

Correlated pair		Correlation value
MaxTemp	Temp3pm	0.984562
Pressure9am	Pressure3pm	0.961348
MinTemp	Temp9am	0.901813
Temp9am	MaxTemp	0.887020
Temp3pm	Temp9am	0.860574
MinTemp	MaxTemp	0.736267
Temp3pm	MinTemp	0.708865
Sunshine	Cloud3pm	0.704202
WindSpeed3pm	WindGustSpeed	0.686419
Cloud9am	Sunshine	0.675610

From this information we conclude that the `MaxTemp` and `Temp3pm` have the highest correlation, getting to up to a 98.4%. Which doesn't come as a big surprise, since the time of highest temperatures is usually around mid-day, and 3pm is close to it. On the same line, we get that `MinTemp` and `Temp9am` are strongly correlated as well (90.2%).

`Pressure` doesn't seem to vary a lot within a day, since we get a correlation of 96.1% between the pressure at 9am and 3pm.

We also see that we get a pretty high correlation between `MaxTemp` and `MinTemp`, which pretty much might be explained by seasons.

Moreover, we get a pretty decent correlation of 70.4% between `Sunshine` and `Cloud3pm` (With `Cloud9am` is 67.5%). Probably due to the increased chance of cloudiness when there are less hours or sunlight.

Finally, we also see a correlation of 68.6% between `WindSpeed3pm` and `WindGustSpeed`.

2.3.2 Association between numerical and categorical variables

Despite not having yet `one-hot-encoded` categorical variables, with the library `dython` we can get some idea of the association we should expect between variables. When computing those we get the heat map in figure 4.

At first glance we can at least see there is some decent correlation between `Temperature` and `Location` (about 60%), same with `Evaporation` and `Location`. However, the most significant seems to be the association between `Pressure` and `Location`, which gets up to 93%.

Despite all these high correlations, if we look at our target variable `RainTomorrow`, the highest association is with `Humidity` (40%), which might not come as a big surprise as an association with `RainToday` of 32%. Note that there is a relatively high association also between `RainToday` and `Humidity` getting up to 34%.

2.3.3 Pair plots for highly related variables

After having seen the plots all through the year of several variables in every location, we will now show the pair-plot between the most correlated variables in figure 20 (found in appendix).

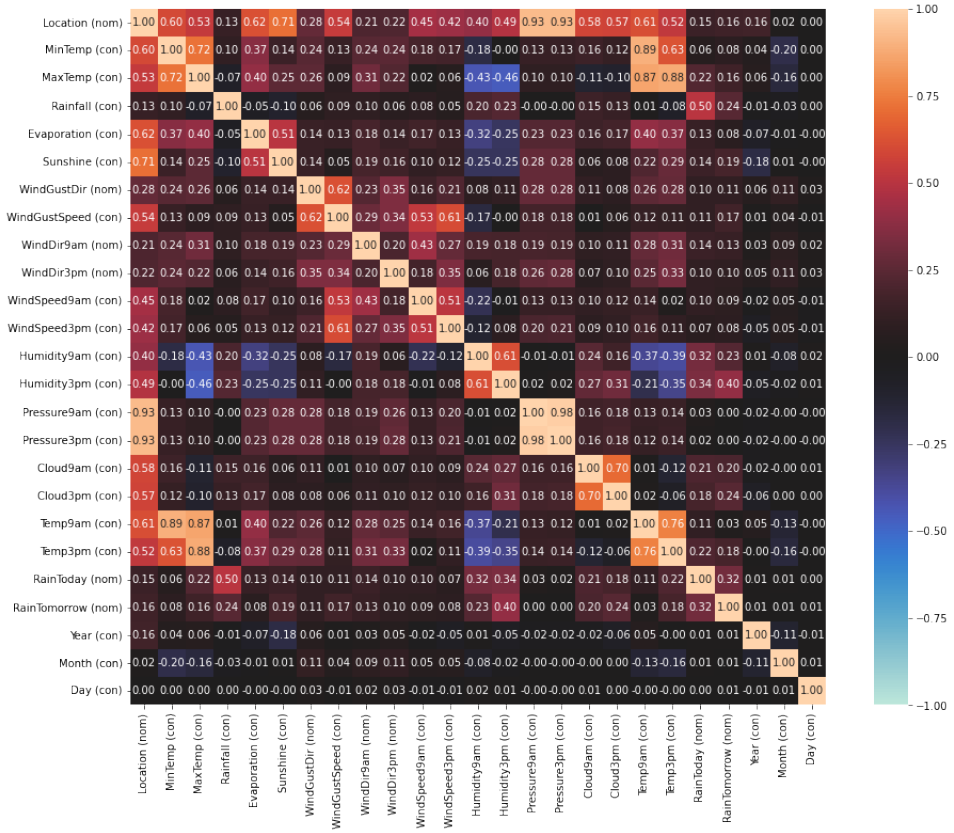


Figure 4: Heat map of correlations between numerical and categorical variables.

We see almost normal-like distributions and pretty much linear relationships between `Temperatures` (with each other) and `Pressures` (with each other).

2.4 Seasonality

Since some of the highly correlated variables seem to be season related, we will try to see if we can objectively find that relationship.

2.4.1 Temperature Distribution across the year on each location

For the maximum and minimum temperatures, we'll plot the yearly distribution for each location in the hopes of finding a pattern. We will also only display the minimum temperatures in this page, not to cram the text, leaving the second set of plots in the appendix. Let's then analyse what we see in figure 5.

Now, when we perform our correlation plot, we expect to have a high correlation between these two variables (`MinTemp` and `MaxTemp`). Moreover, we are also going to see a strong correlation

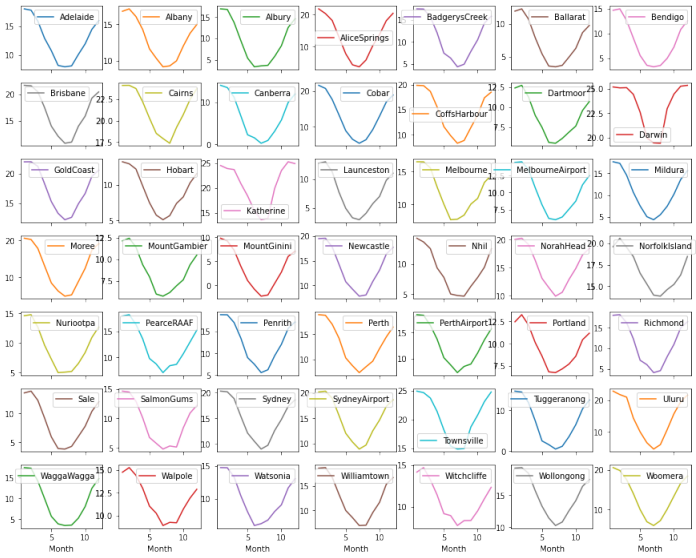


Figure 5: Minimum Temperatures throughout the year for each location of our dataset.

between `Location` and the temperature and between the month and temperature, just as we could expect.

Here, we can see that the correlation seen in the previous correlation plot, holds. The distribution of `MinTemp` and `MaxTemp` on each `Location` through the year, look pretty similar. We can say there is a high chance of seasonality.

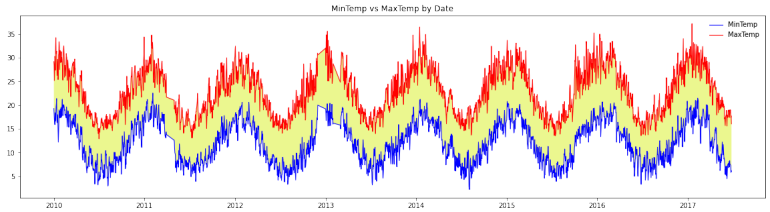
2.4.2 Clouds, sunshine and pressure Distribution across the year on each location

We can do the same with the clouds, sunshine and pressure to see whether we can discover some more seasonality. For the clouds we can conclude from the plots we see in figure 17 (again displayed in the appendix) that there isn't much seasonality related, and neither there is for the sunshine. It shouldn't come as a surprise that these last two show the same result, since there was a high correlation between clouds and sunshine (inverse, in this case). Therefore, in the plots for seasonality (or rather the lack of it), these two variables almost look inverse to each other.

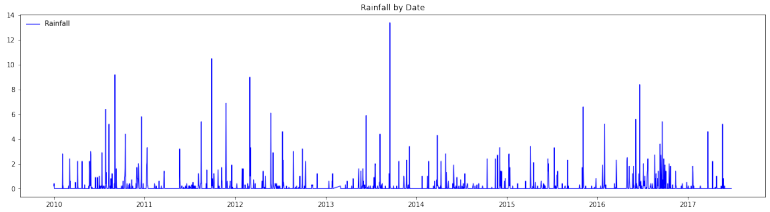
Finally, for the pressure distribution in figure 19 we see once more some seasonality, which is consistent on most locations. In fact, in Perth is pretty much smooth. If we look at our `Temperature` distributions, this looks almost like and upside-down `MinTemp` distribution.

2.4.3 Date plots

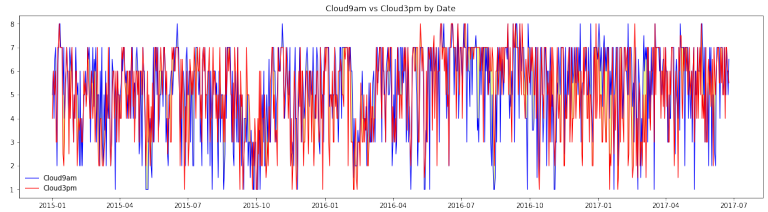
We have previously seen high chances of stationarity in some of our variables, let's plot how some of our variables behave through a given period. With this purpose we have defined a function named `multivariate_dateplot` that given from one to three attributes, yields the plot of them all through the dates in our dataset. The plots that we have obtained are those from figures 6a, 6b, 6c, 6d, 6e, 6f.



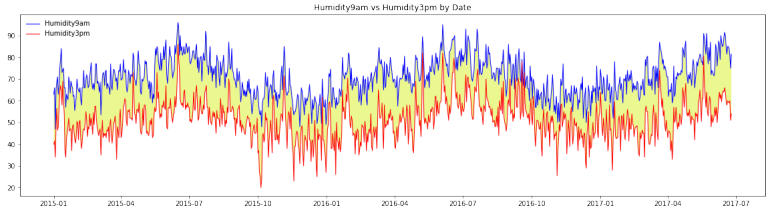
(a) `MinTemp` vs `MaxTemp` from 2010 to mid-2017.



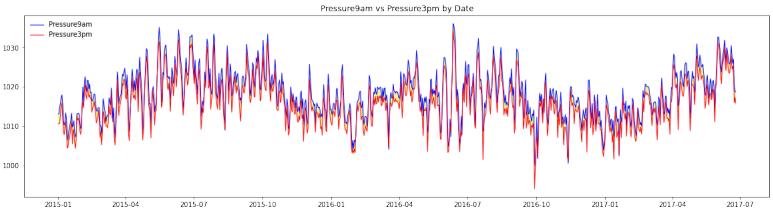
(b) `Rainfall` from 2010 to mid-2017.



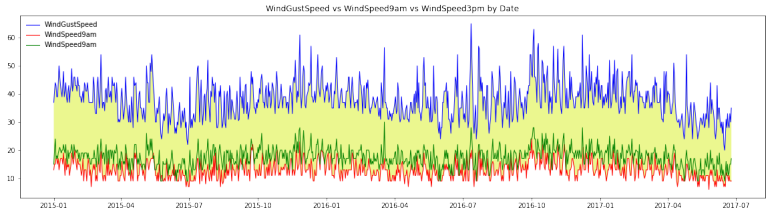
(c) `Clouds` from 2010 to mid-2017.



(d) `Humidity` from 2010 to mid-2017.



(e) `Pressure` from 2010 to mid-2017.



(f) `Winds` from 2010 to mid-2017.

Figure 6: Box plots for all the numerical variables except date.

We can clearly patterns in figures 6a and 6b, although a bit less noticeable in the latter. We conclude from plots in figure 6 that we can assume some seasonality for some of the variables such as temperatures, and the one we are most interested in, rainfall.

3 Preprocessing

In this next section we will be dealing and purifying our data. That means, detecting and removing null values and outliers.

Before continuing, and to make our analysis easier, we'll take the variable `Date` and transform it into three different numerical variables: `Year`, `Month` and `Day`.

3.1 Null values detection

Since our dataset has two main different types of variables: `float64` (numeric) and `object` (categorical), we will proceed to analyze them separately in order to perform a more in-depth study. Chances are our dataset contains lots of NaN's, before diving into a more in-detail analysis of NaN's, let's plot the distribution of NaN's across our variables:

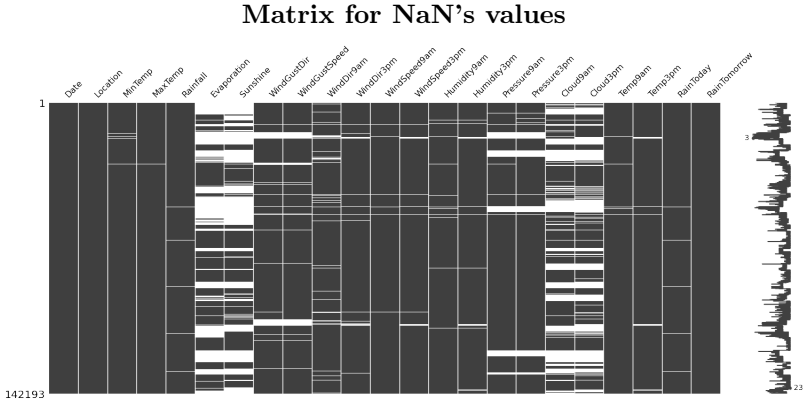


Figure 7: Matrixes indicating in white where a Nan value is found in the column.

From the graphic we can gather that the variables with the most amount of null values are `Evaporation` and `Sunshine`, followed by `Cloud9am` and `Cloud3pm`. More specifically, if we look at the exact number of null values that our categorical variables have, and gather the information in a table we get:

Variable name	Total number of null values
Date	0
Location	0
WindGustDir	9330
WindDir9am	10013
WindDir3pm	3778
RainToday	1406
RainTomorrow	0

Fortunately, 3 out of the 7 categorical variables don't have any null values, but we still have some that do. In particular, those that measure wind speed and direction seem to have lots of NaN values. We will take care of these null values a bit later, in section ().

Next we will check the null values for our numerical variables. If we do a table like the one we did above, we get:

Variable name	Total number of null values
MinTemp	637
MaxTemp	322
Rainfall	1406
Evaporation	60843
Sunshine	67816
WindGustSpeed	9270
WindSpeed9am	1348
WindSpeed3pm	2630
Humidity9am	1774
Humidity3pm	3610
Pressure9am	14014
Pressure3pm	13981
Cloud9am	53657
Cloud3pm	57094
Temp9am	904
Temp3pm	2726
Year	0
Month	0
Day	0

3.2 Outlier detection

To finish up, we will detect all the outliers and remove them, leaving a dataset that will be more reliable and, in fact, more representative of reality.

In order to visualize those attributes that contain the outliers, we have created a function that shows a box plot, making the existence or absence of outliers really easy to observe.

By looking at the six figures, we can quickly see that many of the variables contain outliers. Let's analyse a little more in detail each plot individually. For the box plot of the temperatures, figure 8a, we can see that they seem to be quite stable. However, there are some outliers on both sides (upper and lower). The most stable seems to be `MinTemp`. For figure 8b, all wind distributions seem to have several outliers, all on the upper part. Nonetheless, the one with the most extreme outliers is `WindGustSpeed`. Both `Pressure9am` and `Pressure3pm` boxplots seem to have quite a few values out of quantile range, but there doesn't seem to be extreme values to be considered as outliers. In figure 8d, despite `Humidity9am` having some outliers, they don't look like they are a majority. However, they will have to be dealt with. `Humidity3pm` doesn't seem to have outliers at all. Next, if we look at the box plot for clouds in figure 8e, there are no outliers in sight, so we should be good here. Finally, in the last figure, 8f, `Sunshine` doesn't show outliers either. However, as expected, `Rainfall` and `Evaporation` contain a large amount of them.

To wrap up the outlier detection through the plots, it seems like we'll have to possibly correct all the numerical variables, except from `Humidity3pm`, `Cloud9am` and `Cloud3pm`, and `Sunshine`. The ones we'll be correcting the most seem to be `Rainfall`, `Evaporation` and the different measurements of wind.

3.3 Distribution of certain variables

Since we got 5 variables with extreme values, we will see which distribution do they present. If we get Normal Distributions, we can check for outliers with the help of p-values. If it is skewed, we will use the Interquantile range.

When plotting the distribution for the aforementioned variables we obtain the following results:

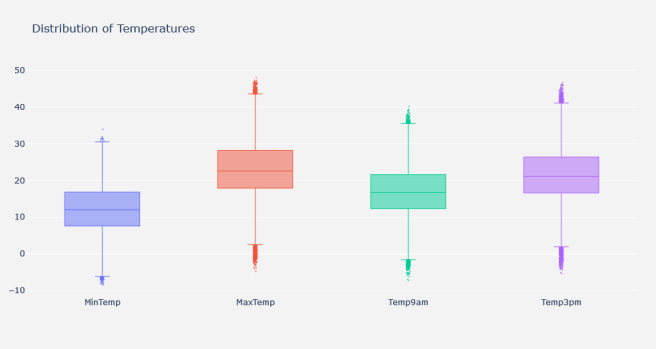
They all look pretty skewed, therefore we will have to calculate the Inter Quantile range to see which values we will consider to be outliers. For this purpose we have defined a function that yields the range of values between which we will consider the outliers. These will be:

Variable name	Inter Quantile Range
`Rainfall`	$[-2.40, 3.20]$
`Evaporation`	$[-11.8, 21.8]$
`WindGustSpeed`	$[-20.0, 99.0]$
`WindSpeed9am`	$[-29.0, 55.0]$
`WindSpeed3pm`	$[-20.0, 57.0]$

3.4 Null values and outliers removal

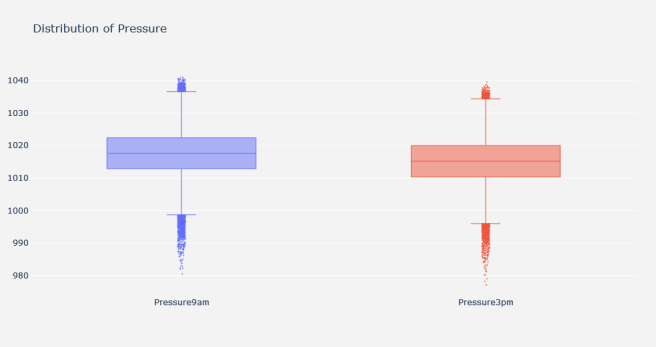
Now, finally using all the information we have gathered in the previous sections, we can remove the null values and the outliers. For the NaN's in the numerical values, we will assume that they are located in random positions of the dataset. As a result, we will fill numeric NaN's with the median, since it is robust to the plenty of outliers we have seen in our previously.

On the other hand, for Categorical Variables, we will use the mode, since now we have discrete values and most robust option will be to use the most frequent value to replace null values. However, first, we will have to Label Encode our strings.



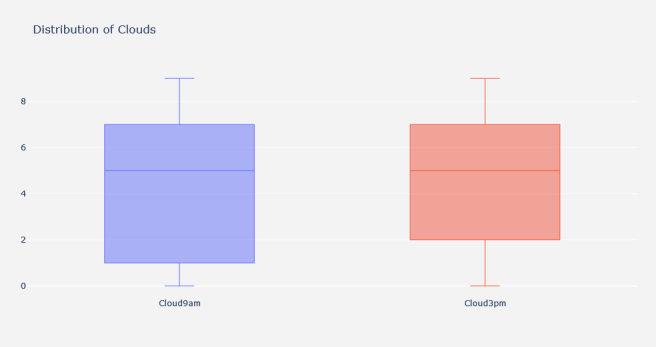
(a) `MinTemp`, `MaxTemp`, `Temp9am` and `Temp3pm` box plots.

(b) `WinGustSpeed`, `WindSpeed9am` and `WindSpeed3pm` box plots.

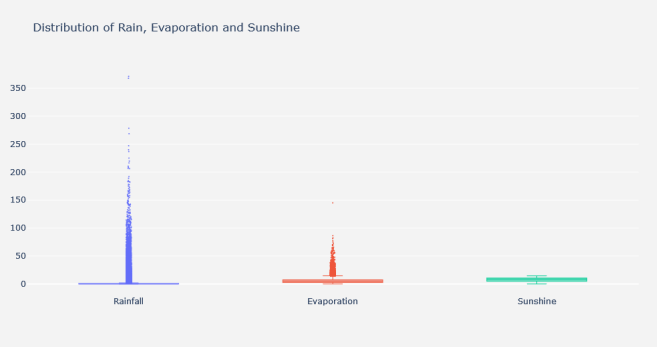


(c) `Pressure9am` and `Pressure3pm` box plots.

(d) `Humidity9am` and `Humidity3pm` box plots.



(e) `Clouds9am` and `Clouds3pm` box plots.



(f) `Rainfall`, `Evaporation` and `Sunshine` box plots.

Figure 8: Box plots for all the numerical variables except date.

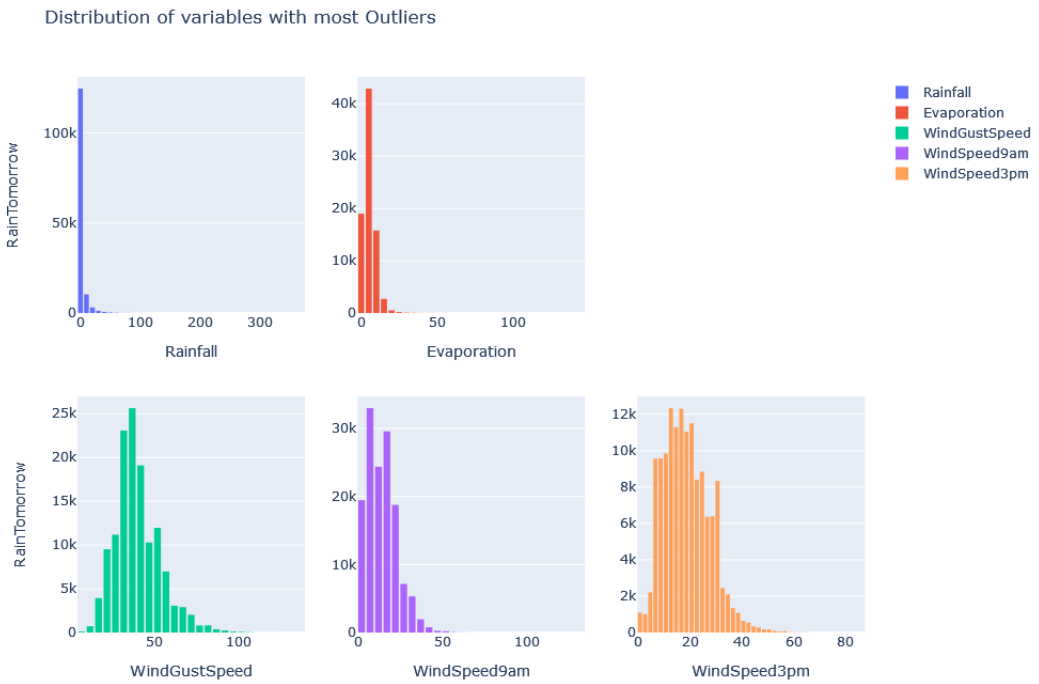


Figure 9: Distributions for the attributes `Rainfall`, `Evaporation`, `WindGustSpeed`, `WindSpeed9am` and `WindSpeed3pm`

Finally, despite having analysed and shown the Inter Quantile range only for 5 of the variables in this document, in the dataset we will actually compute the same range for all the variables and remove the outliers for all of them. However, it will have most effect on the 5 we just mentioned.

3.5 Data Normalization

Now that the null values and outliers have been successfully detected and removed, we will normalize our data. In order to do so, we'll firstly need to encode the categorical variables. Otherwise, there would be no way of using them.

3.5.1 Categorical variables encoding

In order to encode our categorical variables, we will make use of the `One-Hot-Encoding`. This means that, for every categorical variable, the different labels are considered and converted into columns of binary values. Those instances that had the label contain a number 1, and those that didn't the number 0.

3.5.2 Numerical normalization

For this section, we want to normalize all the values in our dataset, i.e. we want to make sure that they are all within the same magnitude scale. We will try two different normalization methods: `MinMaxScaler` and `StandardScaler`. The first one normalizes all the values in the dataset considering as the extremes of the range the minumum and maximum values of the dataset. Instead, the Standard Scaler normalizes every value between 0 and 1. After having tried all of the models with both of these normalizations, we've found the standard scaling to show better results. Hence, from now on, we will only consider the data that has been normalized between 0 and 1, and we will show the results for only that case.

With these we reach the end of our Exploratory Data Analysis and Feature engineering, having successfully preprocessed our data and creating a new dataset with a higher reliability, that will most certainly give better results when training. We will separate our data into train and test using a 80 to 20 ratio; 80% of the samples to train and 20% of them to test. We will also stratify the variable to predict, since it will make our random split have the same proportions of `No` and `Yes` we had in the original dataset. Hence, not modifying the probabilities of rain. And probably preserving more statistical properties.

4 Classification

Finally, we are ready to get our hands dirty with what's really important: making a prediction for our dataset. For this we will be trying a total of 12 different models, from which we'll select those that perform the best. Next, we will optimize them, to see if we can get an even better classification, and lastly, we will try to perform an ensemble of methods, where we will tune each model first, and then will try to put them all together to see if we can achieve greater performance.

4.1 List of used models

We can classify the models we have selected in two categories: basic models and ensemble models. They are:

Basic Models
Logistic Regression (`LogisticRegression`)
Decision Tree Classifier (`DecisionTreeClassifier`)
Linear Discriminant Analysis (`LinearDiscriminantAnalysis`)
K-Neighbors Classifier (`KNeighborsClassifier`)
Gaussian Naïve Bayes (`GaussianNB`)
Support Vector Classifier (`SVC`)
Stochastic Gradient Descent Classifier (`SGDClassifier`)

Ensemble Models
Optimised Gradient Boosting Classifier (`XGBClassifier`)
Light Gradient Boosting Machine Classifier (`LGBMClassifier`)
Random Forest Classifier (`RandomForestClassifier`)
Adaptative Boosting Classifier (`AdaBoostClassifier`)
Ensemble Decision Tree Classifier (`ExtraTreesClassifier`)
Voting Classifier `VotingClassifier`

It's important to remind ourselves that we are trying to predict when it will rain. Therefore, (and since our data has a bigger proportion of labels indicating that it will not rain), a classifier that has good accuracy thanks to mostly predicting that it will *not* rain, will be useless for our purpose. Similarly, a classifier with a big bias towards one particular label, where the error for one is minimal, and in the error for the other is greater, would also show a "good" result, but will in fact represent reality very poorly.

To try and fix this bias, we will train our dataset with and without *Synthetic Minority Oversampling TEchnique* (SMOTE). Basing ourselves on the results, we'll decide if the use of it will help us make a better prediction or not.

4.2 Model performance without SMOTE

For the same reasons we just listed to justify the use of SMOTE, we will also need to use a different metric from accuracy. The one we will be using instead will be the Area Under the curve (AUC), since it will quantify the ROC curve. This method will be 1 when we obtain a perfect classifier, and 0.5 if the classifier is random in its guesses.

We have opted for gathering all the results for each method in a table, where we will indicate the model name, the accuracy, the standard deviation and the time per iteration it took to train the model:

Model name	**AUC**	**Std. Deviation**	**Time/it**
`LogisticRegression`	0.871	0.00356	8.57s/it
`LinearDiscriminantAnalysis`	0.868	0.00359	9.92s/it
`KNeighborsClassifier`	0.760	0.00601	82.19s/it
`DecisionTreeClassifier`	0.699	0.00621	59.72s/it
`GaussianNB`	0.739	0.00626	38.81s/it
`SVM`	0.884	0.00338	3143.17s/it
`SGDClassifier`	0.855	0.00517	2121.01s/it
`XGBClassifier`	0.892	0.00266	1479.16s/it
`LGBMClassifier`	0.888	0.00282	1023.32s/it
`RandomForestClassifier`	0.885	0.00262	770.83s/it
`AdaBoostClassifier`	0.863	0.00416	560.45s/it
`ExtraTreesClassifier`	0.885	0.00357	836.14s/it

By looking at the results, we can see that the best performing model was `XGBoost` with an average area under the curve of 89.25%, closely followed by `LightGBM` with an average AUC of 88.82%.

However, these are ensemble models per se. Out of the basic models, the best performing one was the `SVM` model (88.4%), followed by the Logistic Regression (87.1%). `SVM` however, despite being the best performing basic model, was the slowest by far, taking about 3000s/it (it took for our computer around 2h30 minutes to compute).

4.3 Model performance with SMOTE

Let's follow the same steps we did in section 4.2, but applying the SMOTE technique this time. The table now looks like:

Model name	AUC	Std. Deviation	Time/it
`LogisticRegression`	0.871	0.00625	5.39s/it
`LinearDiscriminantAnalysis`	0.870	0.00615	6.04s/it
`KNeighborsClassifier`	0.786	0.00610	28.69s/it
`DecisionTreeClassifier`	0.719	0.00387	22.92s/it
`GaussianNB`	0.739	0.00714	15.01s/it
`SVM`	0.889	0.00155	275.34s/it
`SGDClassifier`	0.858	0.00524	189.58s/it
`XGBClassifier`	0.892	0.00604	149.67s/it
`LGBMClassifier`	0.889	0.00559	106.25s/it
`RandomForestClassifier`	0.887	0.00553	110.81s/it
`AdaBoostClassifier`	0.863	0.00562	91.16s/it
`ExtraTreesClassifier`	0.889	0.00444	104.13s/it

Even though we've shown here the results with a small precision, we have marked in green those values that have increased when using SMOTE. As we can see, SMOTE has obviously increased the accuracy of the prediction. Not only that, but it has also significantly reduced the time it takes to compute. This is due to the methods trying to balance the lack of samples with the label `Yes`, that SMOTE already covers. Therefore, we will be using it from here onward.

4.4 Optimization with SMOTE and best performing model

Let's try to optimize a little bit further the number of neighbors for the `BorderlineSMOTE` using our best performing algorithm, i.e. `XGBClassifier`. In doing so we obtain:

k neighbors	AUC	Time/it
K : 1	0.890194	242.37s/it
K : 2	0.890408	244.26s/it
K : 3	0.890126	244.54s/it
K : 4	0.890921	250.09s/it
K : 5	0.890089	248.14s/it
K : 6	0.890371	244.76s/it
K : 7	0.890105	245.90s/it
K : 8	0.890548	245.66s/it
K : 9	0.890583	242.63s/it
K : 10	0.890304	244.08s/it

After this small simulation, we get that, though there is not much difference, the best parameter for k is 4.

4.5 Feature Importance

If we look at our dataset right now, we see there are more than a hundred different columns, which might make our models struggle with predictions. Let's try to find the best number of features, so that it is both: as fast as possible and as accurate as possible. Let's get those features which have been the most important ones the `XGBoostClassifier` has used to perform the best classification among all. When we plot the `F score` against each feature we get the following graphic:

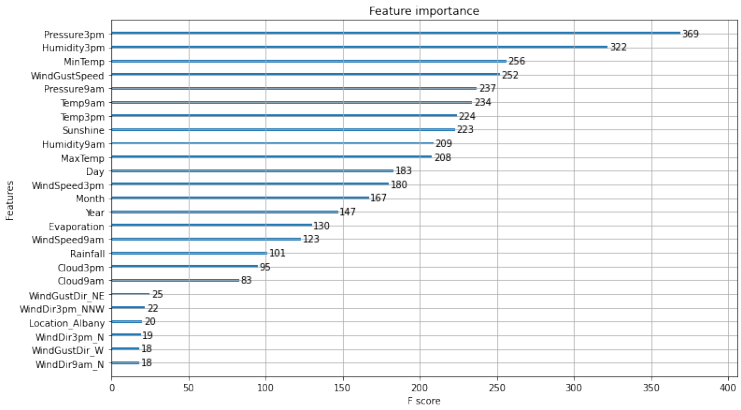


Figure 10: Feature importance using `XGBoostClassifier`.

We can see that the most important features in this case are `Pressure` and `Humidity`, by a significant amount. This plot represents the weights our model has put on each parameter.

Let's also plot the accuracy importance (gain) that each feature has to offer, also for the `XGBoostClassifier`. In this case we obtain:

In this case, `Location` seems to be significantly more relevant.

We now know there are some features that contribute more to the prediction than others. As a result, some of the smaller variables might be adding noise to the output (maybe even some of the ones with high weight). Also, reducing the number of features, will help our algorithm avoid overfitting.

Hence, we will now look for the best combination of k-features, so that we don't necessarily have to use all the features that were given. If that were the case, the model would potentially be faster and would have less tendency to overfit the data.

To know what's the best number of features we have performed different tests with different amounts of features each. To our surprise, the best result is obtained when we use all of the features in the dataset, meaning that all of our variables are to some extent relevant to predict the rain. Therefore, we will be using the entire dataset to make our prediction.

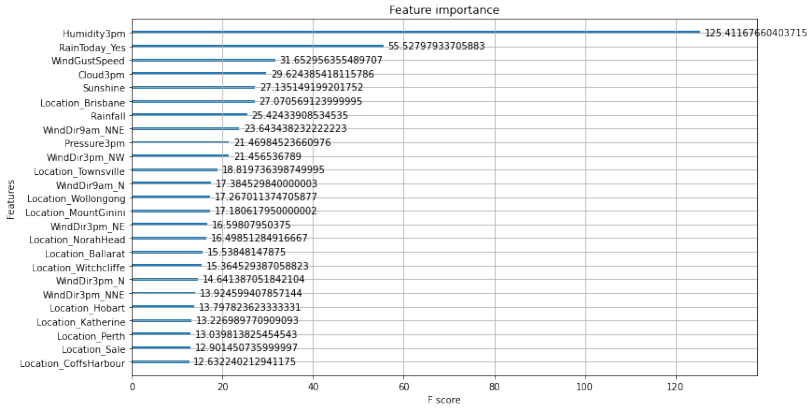


Figure 11: Accuracy gain using `XGBoostClassifier`.

4.6 Hyperparameter tuning using Bayesian Optimization

From the model training with SMOTE, we got that the top 2 models were: `XGBoostClassifier` and `LGBMClassifier`.

We will use Bayesian Optimization for both of these models, so that we get the highest AUC possible in the cross-validation. This type of optimization (unlike grid search and randomized search among others) uses informed search to optimize. This allows us to get closer to the real maximum values than we would using other methodologies.

After the optimization, we will build an ensemble with the top 4 best performing models (although only the first two will be optimized).

To tune the hyperparameters and to avoid as much as possible overfitting for the XGBoost Classifier with Bayes optimization, we will define a cross-validation function. Next, we intialize the Bayesian Optimization object, using the function we just created as one of their parameters.

We will also define a set of parameters that we want the optimizer to explore. This will make our model experiment with more values than just the ones it thinks are best. Since it is informed search, not only will it get better with iterations, but we will be adding an extra layer where it can learn parameters from outside the region where it thinks the optimal values are. In this case, we have defined 8 different initial values for each of the parameters of interest.

Once all this preamble is done, we can finally cross-validate the model. We will do so with 50 iterations and with 10 different initialization points.

From this computation, we get a maximum gini value of 0.801024. We save the parameters that have yielded the best result, and finally perform another training now with the parameters with just saved. This ends up giving us an AUC of 0.893 with a standard deviation of 0.0032, so far, the best accuracy we've obtained.

Without the need to get into as much detail as previously, and following the same procedure for the LightGBM algorithm (once more using Bayesian Optimization) we get a value for AUC of 0.8921, with a standard deviation of 0.00404.

This result is in fact quite better than without parameter optimization. It is now pretty close to the optimised XGBoost.

We can gather all the information we just explained in a table for better visualization:

Model	AUC before Bayesian Optimization	AUC after Bayesian Optimization
`XGBoost`	0.892	0.893
`LGBM`	0.889	0.8921

4.7 Ensemble model with tuned hyperparameters

We will now proceed to ensemble the four models that we just finished tuning. In essence, this will take all the results from the different models, and evaluate their answers. Like a voting system, it will choose as correct the label that gets the most "votes" from the different models. We do consider this technique to be a very useful one, since it certainly improves the accuracy in most cases. However, it does significantly increases the time, given the fact that it executes several models per iteration.

The definitive result we get from this final model, has an accuracy of 0.9044 with a standard deviation of 0.0053. This result more than satisfies our goals, so we will stop our model classification here and move onto testing with the data that we separated in the beginning for that same purpose.

Let's collect all the information and results we have obtained from the training. For the models that we have created so far we have the following results.

Summary table for train

Model	AUC	Std. Deviation
`XGBoost`	89.25%	0.00266
`XGBoost` + SMOTE	89.28%	0.00604
`XGBoost` + SMOTE + Bayesian Optimization	89.21%	0.00404
Ensemble model of `XGBoost`, `LGBM`, `SVM` and `RF`	90.44%	0.0053

Finally, we will test our model in the 20% of the dataset that we had set apart and hadn't used yet. We did it this way so that there wouldn't be any information leakage by using the test data when doing cross-validation. The final results then are:

4.8 Trivial classifiers

Since we have an imbalanced dataset, we will see what is the classification report we would get in case we defined a trivial classifier. We will look at the `f1-score` of label `Yes` with randomly choosing between the labels `Yes` and `No` (for *it will rain* and *it will not rain*), and when choosing only the label `No`. Let's gather the results in a table:

	50% `Yes` and 50% `No`	Always `No`
`f1-score:`	0.23	0.23

4.9 Test Results

After executing our test data with the different classifiers we have seen, we get the following AUC:

Summary table for test

Model	AUC
`XGBoost`	88.92%
Ensemble `XGBoost`	89.1%
Ensemble model of `XGBoost`, `LGBM`, `SVM` and `RF`	90.25%

5 Plots for ROC and Precision-Recall Curve

Before diving into ROC and PR curves, we will look at plots that will give us some insight about the ROC and PR curves.

Firstly, let's see trough a histogram, how spread are the scores according to their true label:

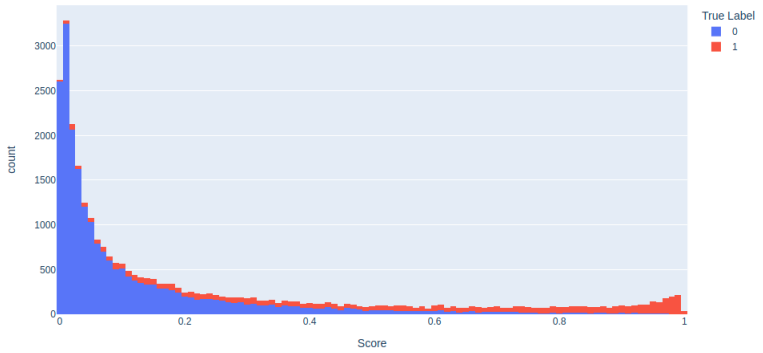


Figure 12: Scores Histogram.

There are certainly lots of value counts with label `No` (0). There are however some labels predicted as `Yes` (1).

Let's now see in figure 13 the proportion of true-positives and false negatives:

We see that the positive rates could be better, mainly the true positive rate.

Let's continue by showing the plot for the ROC Curve (Receiver Operating Characteristics).

The Area Under the Curve (AUC) is as high as 90.25%, a very good result.

Lastly, let's have a look at the Precision-Recall Curves (PR Curve)

The tradeoff between precison and recall is above the threshold, but could be better. When precision drops to as low as 0.2-ish, recall gets to 1, almost the same the other way around. High precision relates to a low false positive rate, and high recall relates to a low false negative rate.

6 Conclusions

By implementing an ensemble with the best XGBoost configurations found by Bayes Optimization, we have managed to get about three times better at predicting whether it will rain or not. Which is pretty good. Mixed ensembles with SVMs are incredibly slow, getting to almost 2h to train, while xgboost ensembles require about 5 minutes and get better performance.

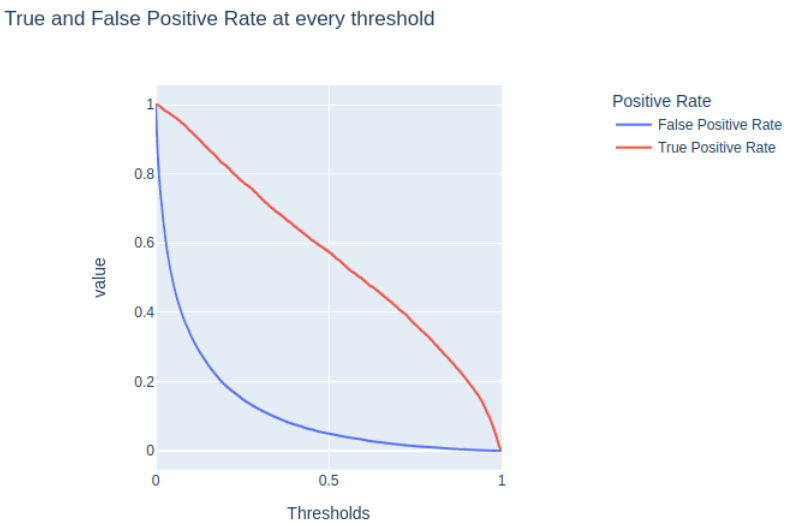


Figure 13: True-positive rate and False-positive rate.

Better feature selection could be done in order to get faster models, however, performance would decrease.

The AUC achived got to as much as 90.25 (XGB Ensemble) on the test set and to 90.44 (Mixed ensemble) with in the cross-validation.

Also, we've seen all throughout the project the importance of cross-validation. This technique is really helpful to determine which models to choose from. Given the fact that models usually compute with a certain degree of randomness, if we were to only execute it once, we would be risking the chance of the particular execution being far from the "real" accuracy of the method. By using cross-validation when choosing the models, we've made sure that we truly are choosing the model that gives the best results.

This has been a challenging yet fun project to carry out, although we did encountered some obstacles. For example, the execution time for some models significantly slowed down the work (we had to let it run overnight to be able to see results at all). Nonetheless, we got to very good results that paid off the hard work put into it.

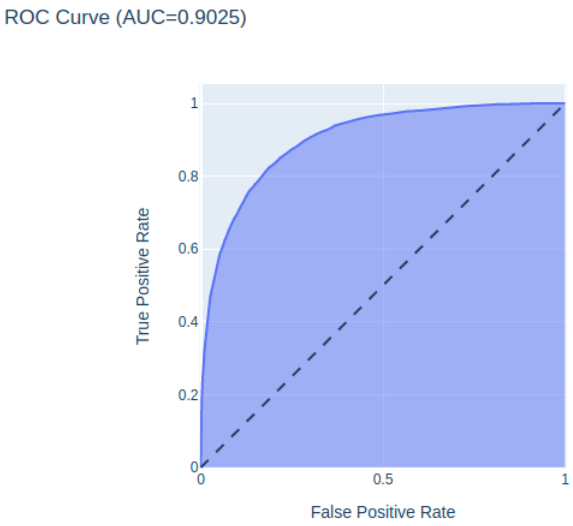


Figure 14: ROC curve.

References

[1] Jeffrey Jyhuang. (2021, October). Australian Weather: Logistic Regression Classifier. Retrieved from Kaggle Website.

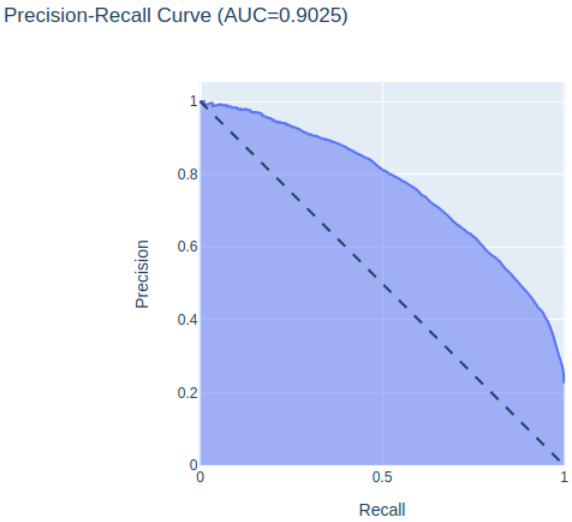


Figure 15: Precision-Recall Curve.

7 Appendix

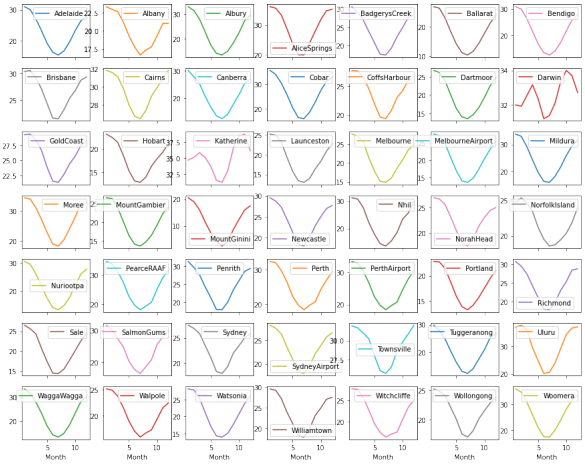


Figure 16: Maximum Temperatures throughout the year for each location of our dataset.

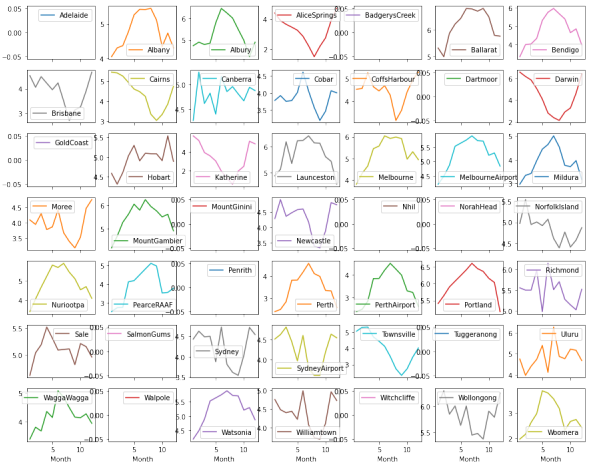


Figure 17: Clouds throughout the year for each location of our dataset.

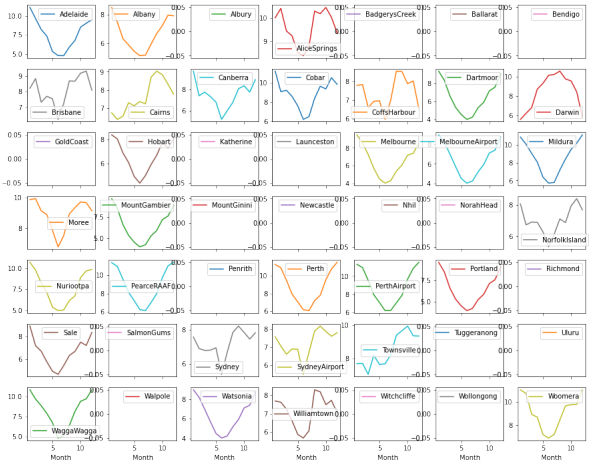


Figure 18: Sunshine throughout the year for each location of our dataset.

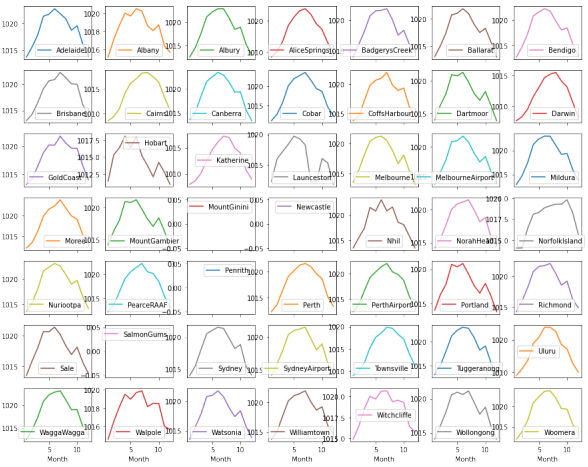


Figure 19: Pressure throughout the year for each location of our dataset.

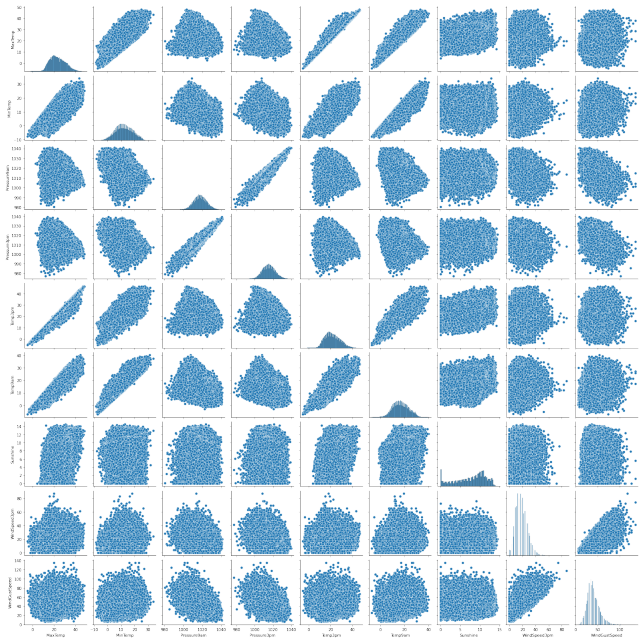


Figure 20: Plot of correlations between the highest correlated variables.