Weather prediction in Australia

Project report

Submitted to

DataScientest, France

to obtain the Certificate of Data Scientist

By

Abhishek Tiwari

Claudia Wiese

Shiksha Ajmera

DS-February 2024

26/04/2024

Contents - Report 1: exploration, data visualization and data pre-processing report

Rain prediction in Australia project

Report 1: exploration, data visualization and data pre-processing report

1. Introduction to the project

1.1 Overview

The following project aims at analysing the weather data in Australia from 2009 to 2017. We want to predict the incidence of rain, the temperature and wind speed. The dataset used contains 10 years of daily weather data from several locations across Australia.

1.2 Context

- Context of the project's integration into your business – The project could integrate the weather predictions into various business sectors, spanning from large-scale operations such as energy production from solar and wind sources to smaller-scale enterprises like food catering businesses and transportation services. Predictions derived from this project, including weather and traffic forecasts, hold significant economic implications. Rainfall and temperature is particularly interesting for the agricultural sector as well. Moreover, if wind prediction is included, it can impact flight services, contributing to both economic and safety considerations.

- From a technical point of view – Handling missing data (NaNs) and addressing dataset imbalance are crucial technical challenges in this project. The dataset contains numerous missing values and exhibits imbalanced distributions, posing significant obstacles to accurate analysis and prediction.

- From an economic point of view – As outlined in the context, the predictions generated by this project can influence the economic landscape by affecting both large and small businesses. Consequently, the accuracy and reliability of these predictions play a vital role in shaping economic decisions and strategies.

- From a scientific point of view – Weather prediction is a complex scientific endeavor influenced by various factors such as geography, terrain, temperature, humidity, and vegetation. Understanding and predicting such intricate phenomena is of scientific interest and significance. Especially rainfall and the temperature is interesting to predict because Australia is known for extensive droughts and extreme heat waves.

- General Info on the Climate in Australia – Due to its large size, Australia has a wide variety of climates. The largest part of the continent has a semi-arid

or desert-like climate. The southeast and southwest have more temperate climate and the northern part tropical climate. Most of our data comes from the temperate zones (See Annex, Figure 5, Map with locations).

1.3 Objectives

- What are the main objectives to be achieved? Describe in a few lines – The primary objective of the project is to utilize weather information to predict rain incidence, and secondarily the temperature and wind speed for the following day, and possibly the next week and month for the Australian locations in the dataset.

- For each member of the group, specify the level of expertise around the problem addressed? – Abhishek- Moderate experience, in dealing with weather data from his master's thesis with weather data as one of the focal points for a non-data science project. Claudia and Shiksha – no previous experience/expertise.

- Have you contacted business experts to refine the problem and the underlying models? If yes, detail the contribution of these interactions. – No.

- (Are you aware of a similar project within your company, or in your entourage? What is its progress? How has it helped you in the realization of your project? How does your project contribute to improving it?). – No.

2. Understanding and manipulation of data

2.1 Framework

- Which set(s) of data(s) did you use to achieve the objectives of your project?

  Rain in Australia: Weather in Australia, dataset for 10 years from 2007 to 2017 from numerous Australian weather stations.

- Are these data freely available? If not, who owns the data?

  The data is freely available at Kaggle under the following link: https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package

- Describe the volume of your dataset? – 145460 rows x 23 columns

- Description of columns in the dataset

Date	Date of observation
Location	The common name of the location of the weather station

MinTemp	The minimum temperature in degrees celsius
MaxTemp	The maximum temperature in degrees celsius
Rainfall	Amount of rainfall
Evaporation	The so-called Class A pan evaporation (mm) in the 24 hours to 9am
Sunshine	The number of hours of bright sunshine in the day.
WindGustDir	The direction of the strongest wind gust in the 24 hours to midnight
WindGustSpeed	The speed (km/h) of the strongest wind gust in the 24 hours to midnight
WindDir9am	Direction of the wind at 9am
WindSpeed3pm	Wind speed (km/hr) averaged over 10 minutes prior to 3pm
Humidity9am	Humidity (percent) at 9am
Humidity3pm	Humidity (percent) at 3pm
Pressure9am	Atmospheric pressure (hpa) reduced to mean sea level at 9am
Pressure3pm	Atmospheric pressure (hpa) reduced to mean sea level at 3pm
Cloud9am	Fraction of sky obscured by cloud at 9am. This is measured in "oktas", which are a unit of eigths. It records how many eigths of the sky are obscured by cloud. A 0 measure indicates completely clear sky whilst an 8 indicates that it is completely overcast.
Cloud3pm	Fraction of sky obscured by cloud at 3pm. (measured in the same way as Cloud9am)
Temp9am	Temperature (degrees C) at 9am
Temp3pm	Temperature (degrees C) at 3pm
RainToday	Boolean: 1 if precipitation (mm) in the 24 hours to 9am exceeds 1mm, otherwise 0
RainTomorrow	Boolean: 1 if yesterday's precipitation (mm) in the 24 hours to 9am exceeds 1mm, otherwise 0

2.2 Relevance

- Which variables seem most relevant to you with regard to your objectives?

Based on correlation data the following variables, we found that he following variables are relevant:

- RainToday,
- RainTomorrow,
- Rainfall
- WindGustSpeed,
- WindSpeed9am,
- WindSpeed3pm,
- MinTemp,
- Max Temp
- Temp9am
- Temp3pm
- Humidity9am,
- Humidity3pm,
- Cloud9am,
- Cloud3pm,
- Pressure9am
- Presure3pm
- Sunshine
- Evaporation

For more details and explanations see Annex, Figure 3, Heatmap.

We also think the Date and Location columns are very important for our analysis because with the Date we can observe trends and seasonality over time.

We also keep Wind direction data. By transforming it, as explained in the pre-processing section, we think it could potentially be interesting for predictions of our target variables.

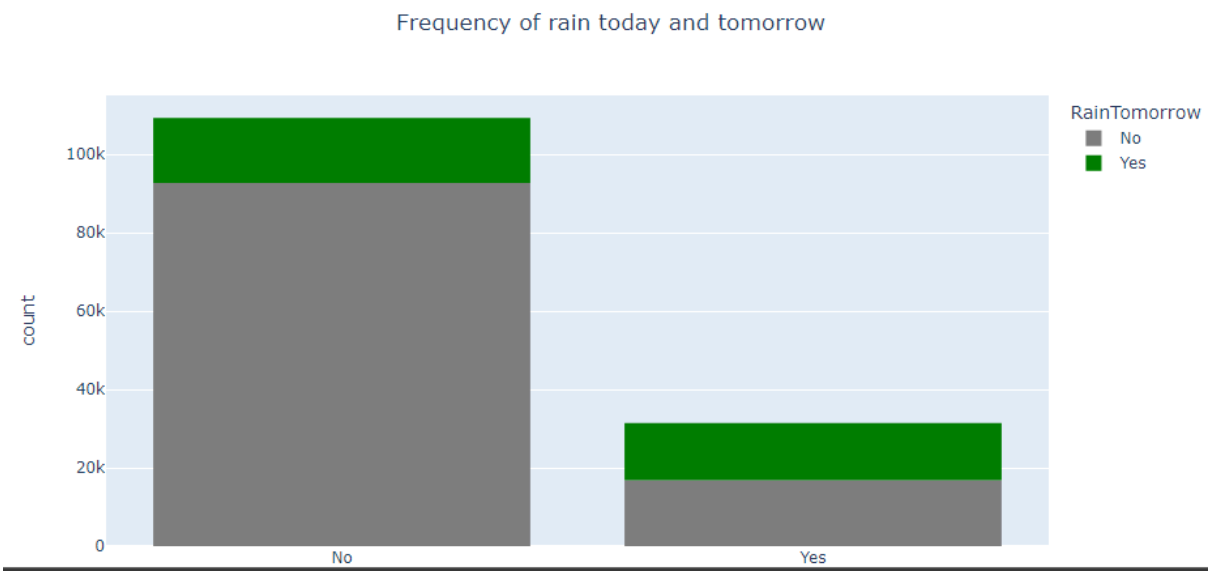
- What is the target variable?

Rain tomorrow is the primary target variable. It is a binary variable taking the values 0 if no rain and 1 if there is rain. If time permits we would also like to predict the Temperature (Mean temperature - created after preprocessing), and Wind speed (created after preprocessing) along with Rain for next week and month. Temperature and Wind Speed are both continuous variables.

- What features of your dataset can you highlight?

The data is special in several ways. First of all we have a time dimension, as we have daily weather data for 10 years and secondly we have data from 49 weather stations all across Australia (See Annex, Figure 5 Map with locations).

We have an imbalanced dataset when it comes to rainfall data (See Figure below)



Most of the distributions of our variables are not normal. Concerning our target variables, we can observe that the temperature variables (TempMax, TempMin, Temp9am, Temp3pm) seem to be distributed almost normally. Rainfall on the other hand is heavily skewed to the right and the Wind Speed variables (WindGustSpeed, WindSpeed9am, WindSpeed3am) are also slightly skewed to the right (See Annex, Figure 4 Distributions of variables).

We also have a lot of missing data for certain columns and for certain years as well as locations some data is non-existent, i.e. has not been collected (see limitations).

- Are you limited by some of your data?

  We have a lot of NaNs, i.e missing data. 4 columns really stick out: Sunshine, Evaporation, Cloud3am and Cloud9am (see Annex, Figure 1 Missing Values)

  The missing data is not missing at random, as it is missing for specific locations which limits us with respect to our strategies of filling the NaN values because most of the classic strategies such as deleting data listwise, using Mean/Mode/Median imputation assume that data is missing at random.

  Another problem is that for some years, data has simply not been collected for certain locations. So the data is not missing but none existing.

2.3 Pre-processing and feature engineering

- Did you have to clean and process the data? If yes, describe your treatment process

  Yes, we did have to clean the dataset with the following process:

Transforming Date Column, Adding Month and Year Columns

We had to transform the Date column into a DateTime column and then extract the Year as well as the Month from the column for further analysis. Extracting the month is useful to check for seasonality which is common when dealing with weather data.

Transforming RainToday and RainTomorrow Columns

We had to transform RainToday and RainTomorrow because initially these were object columns with the categories 'Yes' and 'No'. But to use them for our classification analysis the values have to be integers, 1 for 'Yes' and 0 for 'No'.

Transforming Wind Direction Columns

For better computation wind direction data at 9 am and 3 pm was removed, and the wind direction of the highest wind speed of the day was kept while changing its values from the "compass rose" to the "azimuthal system".

Adding Coordinates Columns

For our location data we added the coordinates, one column for the latitude and one column for the longitude. The coordinates can be useful for our KNN analysis.

Dropping data for 2007 and 2008

When doing our data audit we have seen that there is a lot of monthly data that is non existent, i.e. not been collected for 2007 and 2008. Additionally 2007 and 2008 do not have all locations and also fall into the Australia Millennium Drought Period in the 2000s, so we will drop these years from the dataset.

Dropping Locations

We drop all data of the location Uluru as it has only data for very few years and since it is in the middle of the desert while most of the other locations are close to the seaside we consider it as a location with extreme weather.

We also drop data for the locations Sydney, Melbourne and Perth because for these locations we have Sydney Airport, Melbourne Airport, Perth Airport data and in our data audit we checked that these locations which are basically the same locations have less missing values.

Since the missing data is very location dependent, i.e. not random at all we decided to drop the locations with the most missing data, i.e. if we dropped all NaN's these locations would no longer exist in the dataframe. This leaves us with data for 23 locations.

Dropping Outliers

We used the IQR method (interquartile method) to identify outliers. The interquartile (IQR) is the difference between Q1 and Q3 (the first quartile at 25% and the last quartile at 75% respectively). To detect outliers, all data points which fall below (Q1 – 1.5 * IQR) or above (Q3 + 1.5 *IQR) are considered as outliers. All the outliers seem to be realistic values, except some values for humidity.

We will drop all values where the humidity is zero because the theoretical humidity can not be zero. The other outliers are kept to ensure keeping the realistic and true data to avoid unnecessary bias.

Feature Reduction

We performed feature reduction by aggregating highly correlated columns, namely pressure, temperature, cloud and humidity, which were sampled at distinct times of the day (9am and 3pm). To address multicollinearity and streamline the dataset, we retained the mean values derived from both time points, ensuring a more parsimonious representation of the data while preserving the essential information encapsulated by these variables.

We computed the temperature fluctuation throughout the day by determining the difference between the minimum and maximum temperature values.
We also excluded the wind speed at distinct times of the day (9am and 3pm) to reduce the redundancy and to simplify the analysis.

Imputations

After getting rid of the locations with the most missing NaNs, we have less location dependent missing values and decided to deal with the remaining NaNs by using KNN imputation. KNN imputation or K-Nearest Neighbor Imputation works by finding the nearest neighbors to the missing observation and then imputing it with the K-nearest neighbor. It can be used when data is missing at random. The advantage of this imputation method is that it is easy to implement and doesn't make any assumptions about the data.

Over / Undersampling

To balance the data out we will certainly do some over or undersampling during the modelling process to make better predictions on rain data. We will compare different over and undersampling methods to see what works better in our case. We might start with SMOTE (Synthetic Minority Over-sampling Technique), which is an oversampling technique used to address class imbalance in datasets. SMOTE works by generating synthetic examples of the minority hence balancing the class distribution.

- Did you have to carry out normalization/standardization type transformations of your data? If yes, why?

Yes, because the normalization ensures that all features have the same scale, and we have many variables on different magnitudes than others, and it will prevent larger magnitude variables from dominating.

- Are you considering dimension reduction techniques in the modeling part? If yes, why?

  As mentioned previously in the feature reduction section, we are doing feature engineering. Briefly, this includes Feature Extraction, in which we are taking the mean of multiple columns and creating a new column; e.g., creating a new column of cloud from the mean of cloud9am and cloud3pm. For a complete list of features added and reduced please refer to the feature reduction section above.

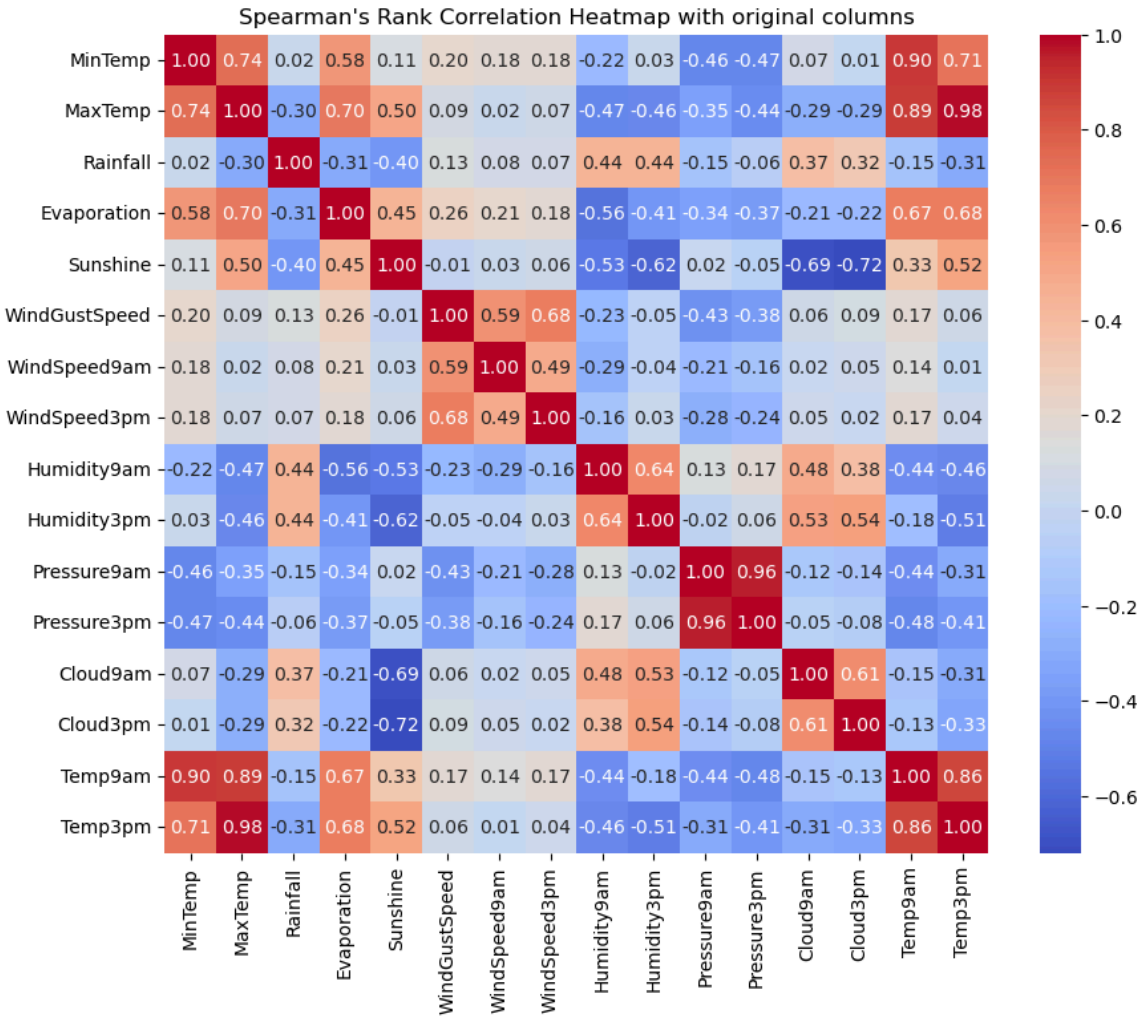
  After reduction, we get the following columns: Date, Year, Month, Location, Rainfall, Evaporation, Sunshine, WindGustSpeed, Latitude, Longitude, WindGustDir_angle, Cloud, Pressure, Humidity, Temp, temp_fluctuation, RainToday, and RainTomorrow.

2.4 Visualizations and Statistics

- Have you identified relationships between different variables? Between explanatory variables? and between your explanatory variables and the target(s)?

  Several variables including explanatory variables are correlated to each other. For exact correlation between all the target and feature variables we have attached a correlation heatmap with detailed explanations below (Figure 1, Heatmap).

Figure 1 - Heatmap showing the correlation between different features and target variables



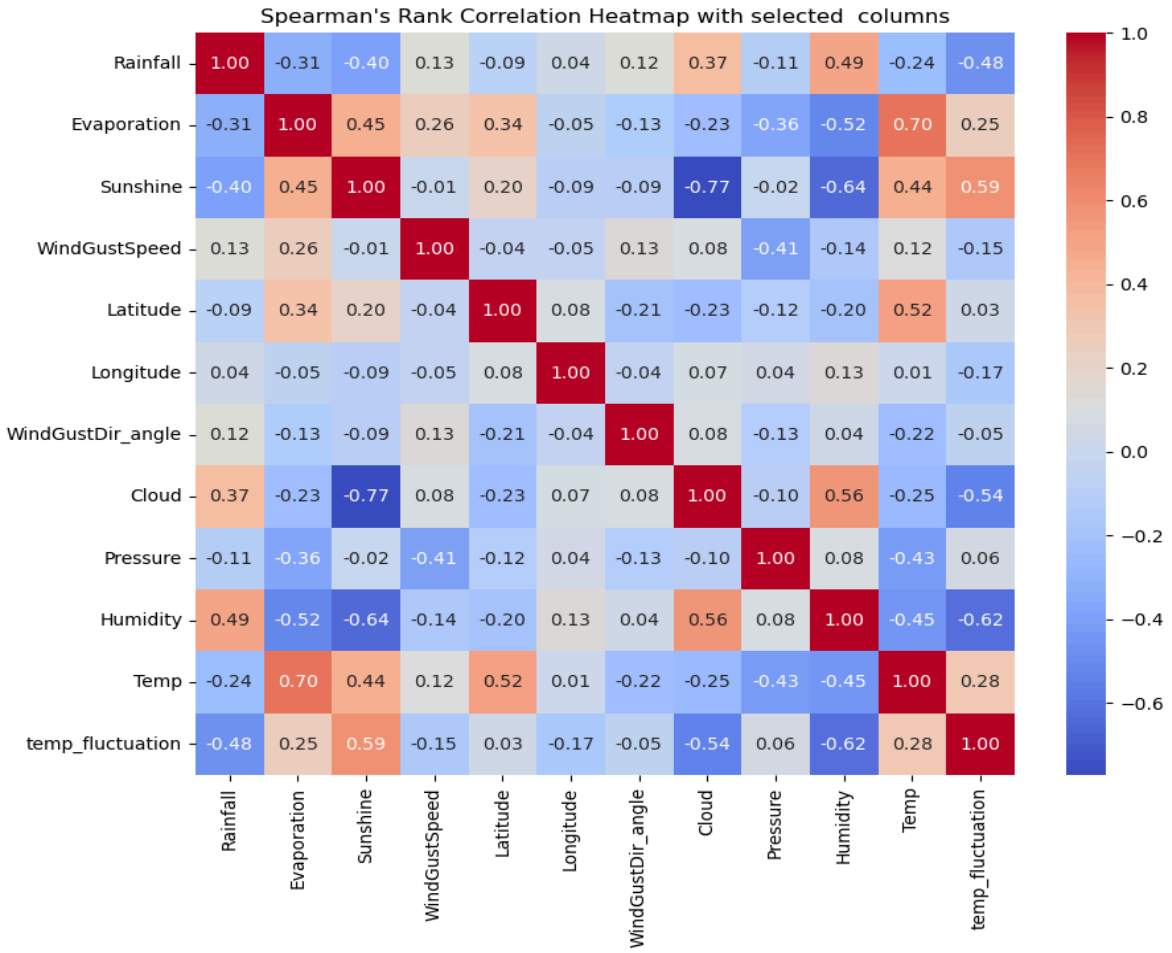
We tested the Spearman's rank correlation of different features with Rainfall on which the target RainTomorrow is based. Even without any feature engineering we can already see that humidity has a significant and the strongest positive correlation with the Rainfall. Cloud data seems to have a positive correlation with Rainfall as well and finally Sunshine also seems to have a quite strong negative correlation with Rainfall. These observed relationships are not very surprising as it seems quite logical that on rainy days there are more clouds and less sunshine.

When it comes to Temperature, it seems to be most strongly correlated with Evaporation, followed by Sunshine (both have positive correlations) and the humidity variables as well as pressure variables (all negative correlations)

Finally wind speed seems to be quite related to the pressure variables (negative correlation)

The heatmap also clearly shows the high correlation within the different temperature variables, the cloud variables, pressure variables and wind speed variables. That is why we fused these variables to avoid multicollinearity as explained in the data engineering section.

Here is the heatmap showing correlations after feature engineering:



As we can see the correlations we observed before are pretty much still the same.

Most interestingly, we can see a quite strong correlation between temperature and Latitude which probably can be explained by the different climate zones in Australia as explained in the context section above. These different climate zones go hand in hand with different temperatures.

- Describe the distribution of these data, distribution, outliers.. (pre/post processing if necessary)

We have identified 3 outliers in humidity (value 0) which is an implausible value, hence dropped. Other outliers are kept as they seem plausible data points with imbalance in the dataset as the identified reason for categorizing them as outliers, so they are kept and used in the modelling.

For more details on the distribution of the data see Annex, Figure 4, Distributions of variables.

- Present the statistical analysis used to confirm the information present on the graphs.

  The graphs for our analysis can all be seen in the annex. For the correlation of all the variables (See Annex, Figure 3 - Heatmap) we tested them statistically using Spearman correlation and Mann-whitneyU test. They were all significant with p-values very close to 0.

- Draw conclusions from the elements noted above allowing them to project themselves into the modeling part

  As explained before the biggest challenge for this data set was to handle the missing values. The Sunshine variable by far exceeds all other variables with 48% missing data, followed by Evaporation with 43% and the Cloud data with around 40%. We handled the missing values in the preprocessing step using the knn method. In this step we have also engineered some new features and dropped the non-correlated features to clean up our data.

  We are not able to make predictions of rain, temperature and wind speed data for all locations present in the original dataset because we had to drop a lot of locations since data was either missing or simply non existent for these locations.

  Another major challenge is the imbalance in our dataset. This bar plot shows the imbalance of the data set when it comes to rain data. There are a lot more dry days than rainy days – before modelling we would need to use oversampling or undersampling methods.

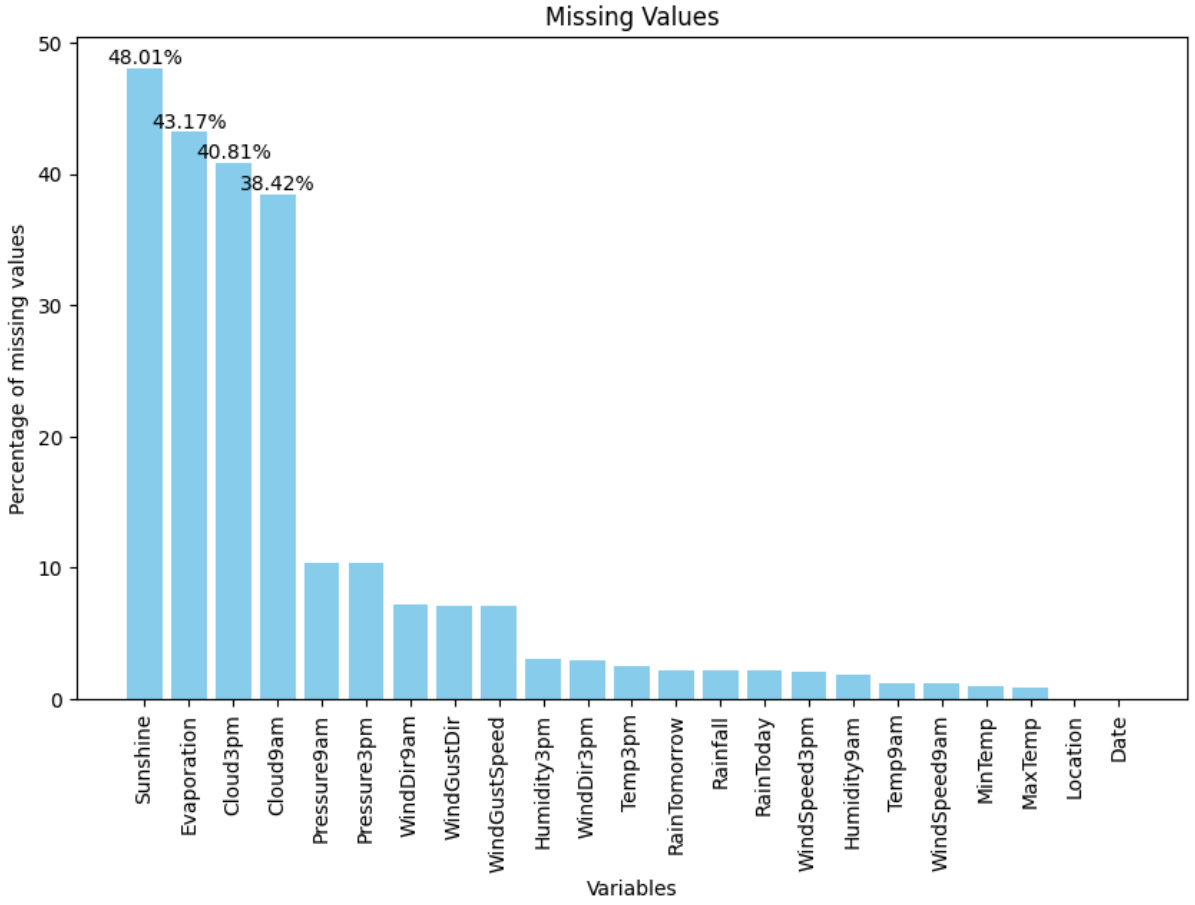
  For the modelling itself, we will predict our target RainTomorrow with a classification model such as XGBoost as it is a binary variable. Here the challenge will be to handle the imbalance of the dataset by using over/under sampling strategies.

  If we have time to work on temperature and wind predictions, we will likely use some advanced regression models.

  We also have to take into consideration the time dimension of the data and handle this by using models specific for time series.

2.5 Annex

Figure 1 - Missing Values



The Sunshine variable by far exceeds all other variables with 48% missing data, followed by Evaporation with 43% and the Cloud data with around 40%.

Figure 2 - Imbalanced Rain Data

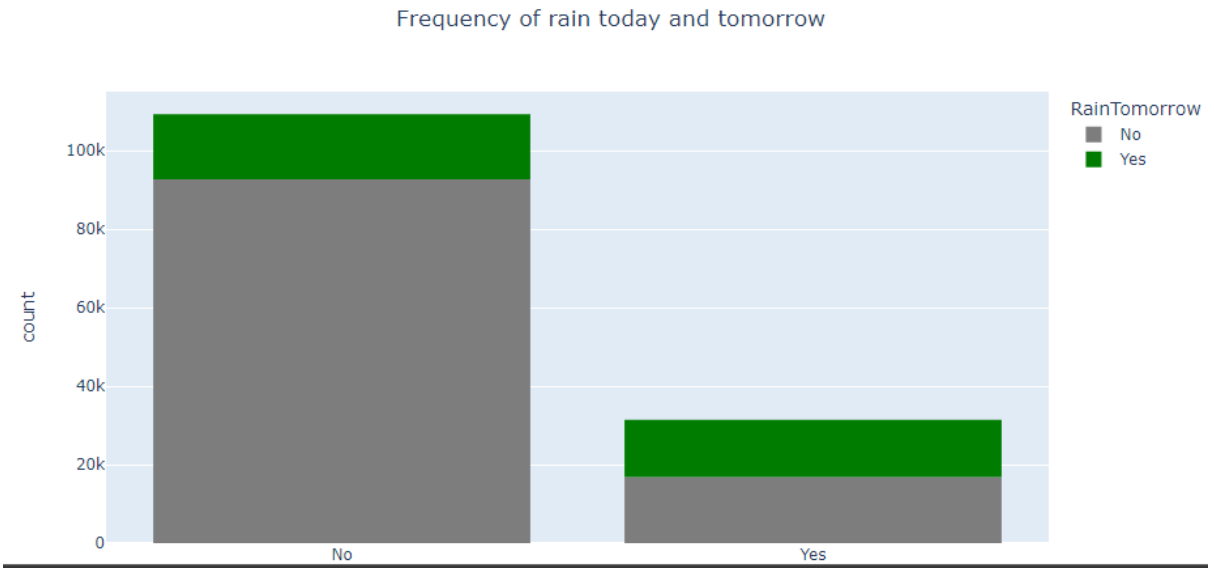
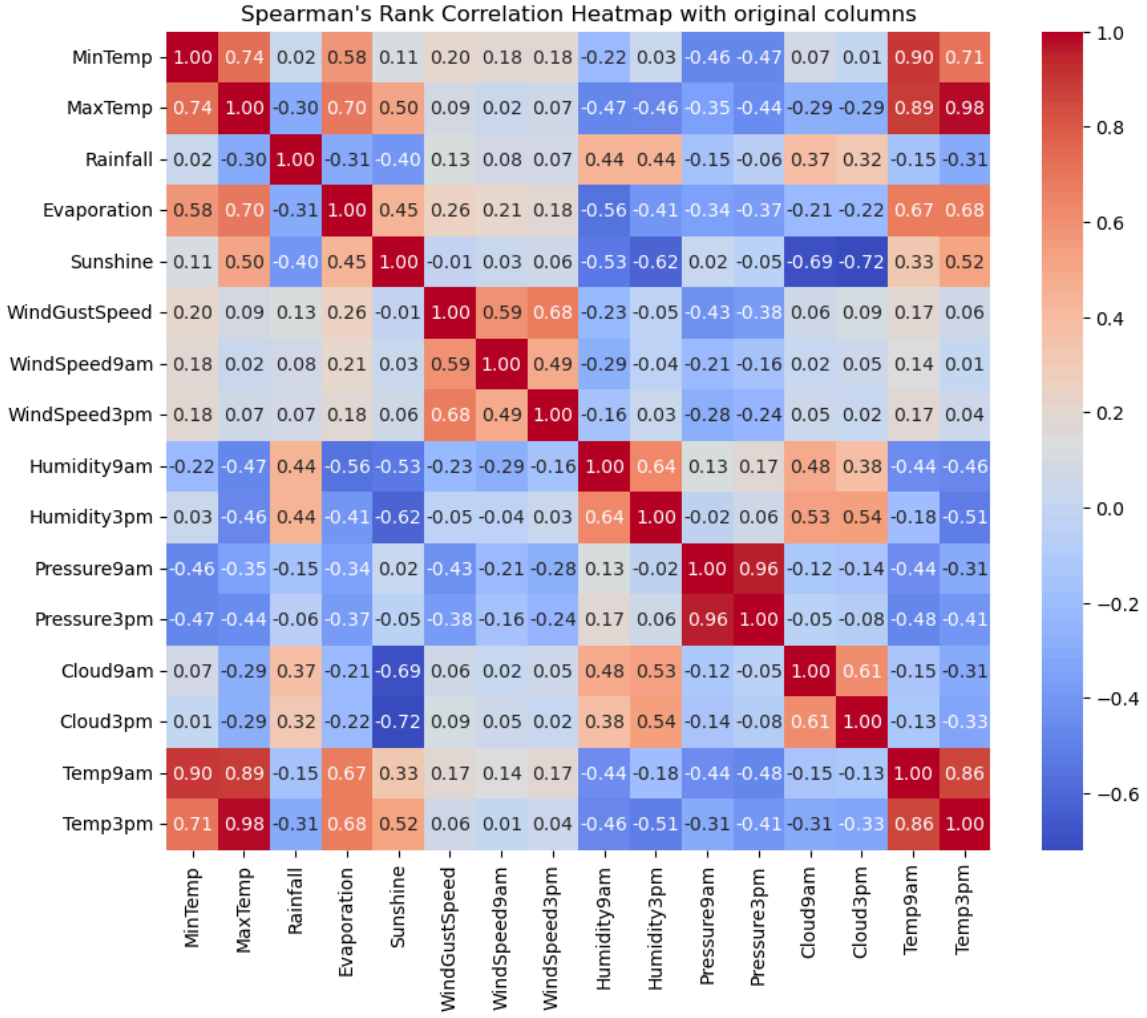


Figure 3 - Heat Map



Here is the heatmap showing correlations after feature engineering:

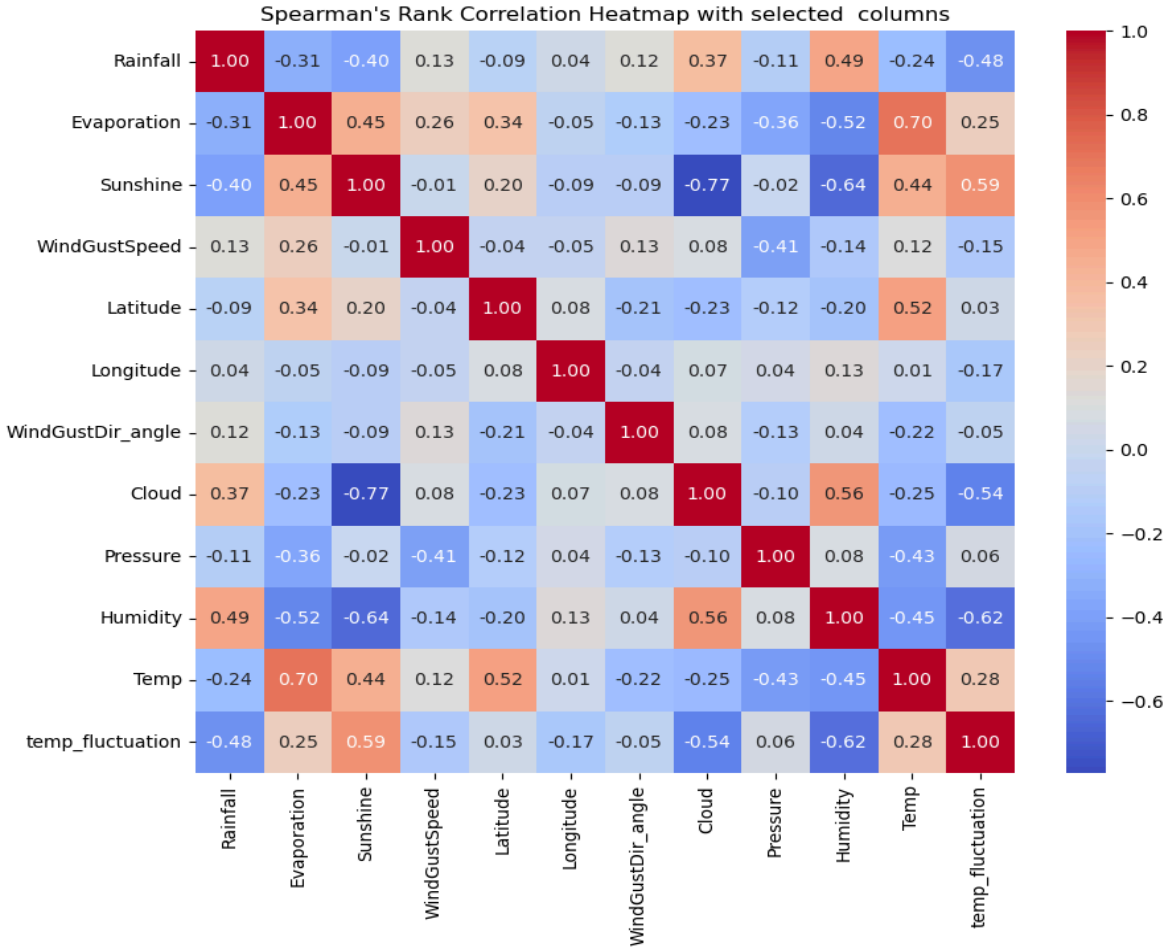
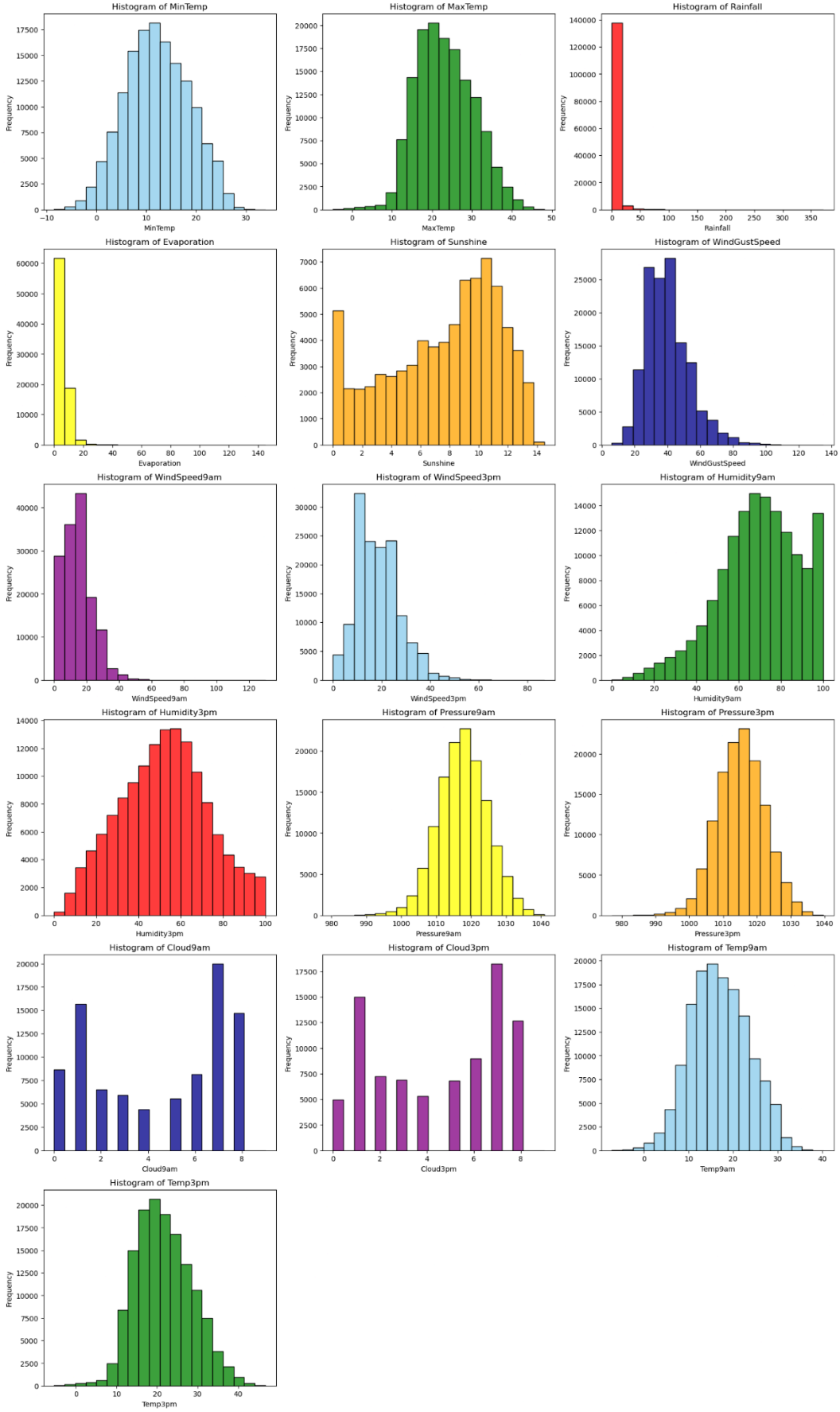


Figure 4 - Distributions of variables



From these graphs we can conclude that most of the data is not normally distributed and hence we will be using non-parametric statistical tests to check for correlations between the variables and any other statistical tests which would be needed later in the study.

Figure 5 - Map with locations

The geographical distribution of the data collection sites is shown above. We can see that most of the locations are in the Southeast. Since the locations are mostly clustered together and have neighbours, it is therefore possible to use the KNN method to handle the missing values.

Contents - Report 2: modeling report

Rain prediction in Australia project

Report 2: modeling report

Stages of the project

---

1. Classification of the problem

- **What kind of machine learning problem is your project like? (classification, regression, clustering, etc)**

  Our project is a classification project.

- **What task does your project relate to? (fraud detection, facial recognition, sentiment analysis, etc)?**

  We have to predict if it rains or not in Australia, using Australian weather data from 2007 to 2010.

- **What is the main performance metric used to compare your models? Why this one?**

  The main metrics were precision, recall/ F1-score i.e. we generated a classification report for each of our models.

  We chose these metrics because our dataset is imbalanced. In well balanced dataset model accuracy might be the best metric to use because it is very easy to understand for a non technical public. It represents the proportion of correctly classified instances out of the total number of instances in the dataset.

  But let's imagine we have an accuracy of 99% simply because 99% of our observations belong to class 0 and just 1% to class 1. Does that mean our model detects the 1% of observations belonging to class 1 well? Maybe or maybe not. To be sure we should look at the precision, recall and F1 scores of our predictions.

  Precision measures the accuracy of the positive predictions, i.e. it shows how often a model is correct when predicting the target class. Recall measures the ability of the model to capture all the positive instances (the ability to find all objects of the target class) and F1-score is the harmonic mean of precision and recall.

The problem is that increasing one of the scores (recall or precision) often leads to the decrease of the other score. There is a constant trade off between precision and recall. In our case we want to predict rainy days.

On the one hand increasing the precision score means that we reduce the number of false alarms (false positives) So if we predict rainy days, they are really rainy days. Reducing the number of false positives could be very cost efficient. For example if rainy days means to take specific precautions in the transport sector for example and these precautions are costly, then we want to avoid false alarms. But on the downside we might simply miss certain rainy days, we have to deal with false negatives.

On the other hand if we focus on increasing the recall score, we are likely to correctly identify all rainy days with certain non rainy days being classified as rainy days as well (false positives). This approach is interesting if the consequences/costs of not predicting rain on rainy days are significant for example for the agricultural sector or water management systems. As Australia is a particularly dry continent, it might be of great importance to correctly predict all rainy days even at the risk of including certain non rainy days.

We observed that our recall scores of class 1 (rainy days) are particularly low in the baseline machine learning models. So our primary goal was to first identify ways to increase the recall score. Unfortunately by increasing the recall scores our precision scores greatly dropped. So we tried other strategies to get a good balance between precision and recall.

Also, although our main focus was not the accuracy of the model, we always had an eye on it to make sure none of our modelling strategies significantly lowered the overall model accuracy.

- **Did you use other qualitative or quantitative performance metrics? If yes, detail it**.

  We graphed the confusion matrix which describes the performance of the model by class. More precisely, it allows us to get an idea of the true positive, true negative, false positive and false negative rates.

  True Positive (TP): The cases in which the model predicted positive and the actual label was positive.

  True Negative (TN): The cases in which the model predicted negative and the actual label was negative.

False Positive (FP): Also known as Type I error, these are the cases in which the model predicted positive but the actual label was negative
.
False Negative (FN): Also known as Type II error, these are the cases in which the model predicted negative but the actual label was positive.

We also graphed the ROC (receiver operating characteristic curve). This curve is a graphical representation of the performance of our successive classification models. It plots the true positive rate against the false positive rate at various thresholds.

True Positive Rate (Sensitivity): This is the proportion of positive instances that are correctly classified by the model.

False Positive Rate (1 - Specificity): This is the proportion of negative instances that are incorrectly classified as positive by the model.

A good model typically has an ROC curve that hugs the top-left corner of the plot, indicating a high true positive rate and a low false positive rate across various threshold settings.

To generate the ROC curve, you would typically need the predicted probabilities of the positive class (class 1) from your model on the test set. With these probabilities, you can vary the classification threshold and calculate TPR and FPR at each threshold. Then, plot these values to visualize the ROC curve.

2. Model choice and optimization

2.1. Machine Learning Models

- **What algorithms have you tried?**

  We have tried Logistic Regression Classification, KNN Classification, Decision Tree Classification, Random Forest Classification, XGBoost. We have tested all these models with random oversampling, SMOTE and random undersampling because our dataset is very unbalanced.

  We did not consider SVM Classification because it took too much time to compute. Similarly the Cluster Centroids undersampling method took too much time to compute. A computationally long time also means that it is costly and some problems require a fast or more economical solution. So, we decided to skip these two methods to improve efficiency.

- **Describe which one(s) you selected and why?**

2.1.1 Logistic Regression

Logistic regression is a statistical technique for predicting a binary result, such as yes or no, present or not present based on an independent data set of previous observations.

A logistic regression model predicts dependent data variables by evaluating the relationship between one or more independent variables. For example, logistic regression can be used to explain whether an image will have flowers or not, it can predict how many flowers it will have. These binary results allow an easy decision to be made between two alternatives. Logistic regression models can take into account multiple input criteria. Based on historical data and past results related to the same inclusion criteria, it evaluates new cases based on their likelihood of falling into one of two outcome categories.

First we ran Logistic Regression without any hyperparameter tuning or over/undersampling methods, .i.e with the default parameters: random_state:42. We got rather disappointing results. While class 0 (days without rain) was well predicted, class 1 (days with rain) had very low precision, recall and f1 scores.

Baseline Classification Report for Logistic Regression(with default params)

	precision	recall	f1-score	support
0.0	0.88	0.94	0.91	10663
1.0	0.70	0.51	0.59	2882
accuracy			0.85	13545
macro avg	0.79	0.72	0.75	13545
weighted avg	0.84	0.85	0.84	13545

As we can see, for class 0 (days without rain) we have a rather high precision of 0.88, meaning that out of all instances predicted as class 0, 88% were class 0. The recall score is also quite high with 0.94 suggesting out of all instances that were class 0, 94% were correctly predicted as class 0. Consequently, the F1-score, i.e. the harmonic mean of the precision and recall score is also high at 0.91. But for class 1 (days with rain), the precision score is 0.7, and the recall score is even only at 0.51, i.e. only 51% of the instances belonging to class 1 have been correctly identified which is hardly better than a random guessing. These results are the same as for the other baseline models without hyperparameter tuning, under, and oversampling so we follow the same steps and try to improve it.

We did hyperparameter tuning by keeping, C: 10000, fit_intercept: True, 'max_iter: 50, random_state: 42, solver: newton-cg.

The results with hyper-tuning were only slightly better, so random over-sampling was the next selected.

Classification Report for Logistic Regression with Random Oversampling

Random Over Sampler was selected for oversampling, the recall score for class 1 was 0.79 compared to the based model which was 0.51 with the best parameter model. However, the precision score of class 1 went very low with 0.51. Also, the recall and f1 scores for class 0 are lower than before. which was 0.91 but now it is 0.86.

	precision	recall	f1-score	support
0.0	0.93	0.80	0.86	10663
1.0	0.51	0.79	0.62	2882
accuracy			0.80	13545
macro avg	0.72	0.79	0.74	13545
weighted avg	0.84	0.80	0.81	13545

Accuracy Train Set: 0.797 Accuracy Test Set: 0.795

Random under sampling also showed the same results as random oversampling.

Classification Report for Logistic Regression with Random Oversampling

Oversampling was done with SMOTE, and the recall score for class 1 decreased by 0.01 to 0.78 compared to the model using Random Oversampling. But it is still better than the baseline model which was 0.51 with the best parameter model. But, the precision score of class 1 is still low with 0.51. Also, the recall and f1 scores for class 0 are lower than before. which was 0.91 but now it is 0.86.

	precision	recall	f1-score	support
0.0	0.93	0.80	0.86	10663
1.0	0.51	0.78	0.62	2882
accuracy			0.80	13545
macro avg	0.72	0.79	0.74	13545
weighted avg	0.84	0.80	0.81	13545

Accuracy Train Set: 0.798 Accuracy Test Set: 0.796

SMOTE with hyperparameter tuning did not lead to any better results, so boosting with SMOTE was selected which did not lead to better results..

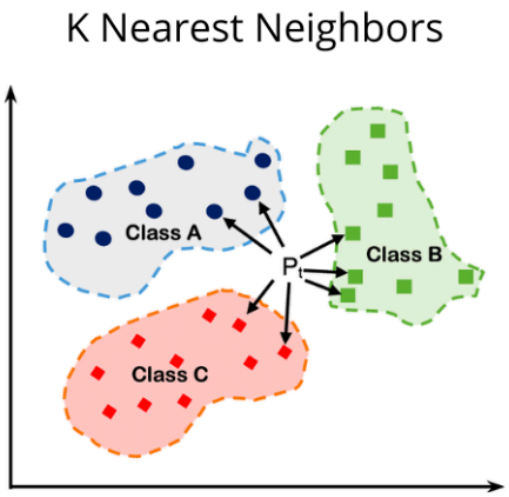
Overall the Logistic Regression is not suitable for this classification project.

When we tried Bagging, it did not give us better results, but it took too much time to compute as well.

2.1.2 KNN Classification

A) Modelling Steps

KNN or K nearest neighbor classification is a supervised learning model that allows us to classify instances based on their similarity with other instances in the dataset. During the training phase, the algorithm stores all data and their corresponding labels, then while predicting a new instance it calculates the distance between the new instance and other distances. This step the K nearest neighbor is selected based on the distances calculated. The final class label of the new instance is then attributed based on the majority class among the K nearest neighbors. The most important parameters to choose are: the number of neighbors being considered and the distance metric. It is a very popular method for classification tasks because it is easy to understand.



First we ran KNN without any hyperparameter tuning or over/undersampling methods, .i.e with the default parameters: n_neighbors: 5, metric: minkowski'. We got rather disappointing results.

While class 0 (days without rain) was well predicted, class 1 (days with rain) had very low precision, recall and f1 scores.

Baseline Classification Report for KNN (with default params)

	precision	recall	f1-score	support
0.0	0.88	0.93	0.90	10663
1.0	0.67	0.51	0.58	2882
accuracy			0.84	13545
macro avg	0.77	0.72	0.74	13545
weighted avg	0.83	0.84	0.83	13545

As we can see, for class 0 (days without rain) we have a rather high precision of 0.88, meaning that out of all instances predicted as class 0, 88% were actually class 0. The recall score is also quite high with 0.93 suggesting out of all instances that were actually class 0, 93% were correctly predicted as class 0. Consequently the F1-score, i.e. the harmonic mean of the precision and recall score is also high with 0.90. But for class 1 (days with rain), the scores are all below 0.7, the recall score is even only at 0.51, i.e. only 51% of the instances belonging to class 1 have been correctly identified which is hardly better than a random guessing.

We got very similar results when we did hyperparameter tuning, i.e. with the following parameters

- Metrics included: 'manhattan', 'minkowski', 'chebyshev'
- Number of KNN neighbors: KNN neighbors from 1 to 41.

We thought that the poor scores for class 1 can be attributed to the unbalanced nature of the dataset. That is why we tried Random Oversampling, SMOTE and Random Undersampling to balance the data out. Our main goal was to get a better than random recall score.

Indeed, we managed to greatly increase the respective recall scores with all three strategies (between 0.8 to 0.82). But on the downside we got very low scores for the respective precision scores (around 0.50). The scores of all three strategies were not very different but in terms of recall score and model accuracy we got the best results when applying Random Undersampling (0.80).

Classification Report for KNN with Random Undersampling

	precision	recall	f1-score	support
0.0	0.94	0.80	0.86	10663
1.0	0.52	0.82	0.64	2882
accuracy			0.80	13545
macro avg	0.73	0.81	0.75	13545
weighted avg	0.85	0.80	0.82	13545

Since the precision score is so low, we decided to do some hyperparameter tuning on top of the undersampling with Random Undersampling. We used the following parameters:

- Metrics included: 'manhattan'
- Number of KNN neighbors: KNN neighbors from 1 to 30.

When we recomputed the KNN model with Random Oversampling and the best parameters (n_neighbors: 1, metric: 'manhattan'), we got the following classification result:

Classification Report for KNN with Random Undersampling and best parameters

	precision	recall	f1-score	support
0.0	0.88	0.87	0.88	10663
1.0	0.54	0.57	0.56	2882
accuracy			0.81	13545
macro avg	0.71	0.72	0.72	13545
weighted avg	0.81	0.81	0.81	13545

Hyperparameter tuning did not lead to better scores nor to better accuracy. Maybe our tuning was too limited. But we cannot include a lot of parameters due to our limited computing power. Therefore we decided to just stick to the baseline model with Random Undersampling and check if the results are persistent when cross validated with 10 test folds.

Fold	Test Precision	Test Recall	Test F1-Score
1	0.853	0.861	0.848
2	0.843	0.852	0.836
3	0.838	0.848	0.831
4	0.838	0.849	0.835
5	0.850	0.857	0.843
6	0.836	0.847	0.834
7	0.842	0.852	0.838
8	0.841	0.850	0.836
9	0.840	0.850	0.837
10	0.845	0.854	0.840

As we can see our scores are persistent over the 10 testfolds, i.e. between 0.836 and 0.853 for precision, between 0.847 and 0.861 for recall and between 0.831 to 0.848 for the F1 score.

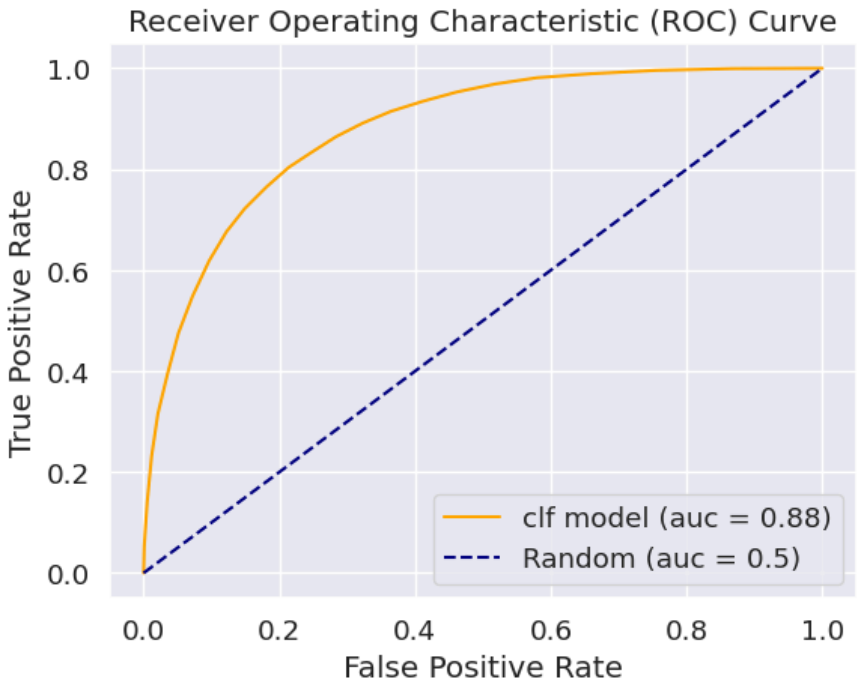
B) Interpretation of Results

AUC-ROC Curve

We looked at the AUC-ROC curves for all our best ML models. The AUC-ROC curve, or Area Under Receiving Operating Characteristic curve, represents the performance of our models across different threshold values for class assignment. It plots the true positive rate (recall) against the false positive rate. ROC is a probability curve and AUC determines the degree of separation between two or more classes.

AUC values range from 0 to 1. where 0.5 is random guessing. Values close to 1 indicate higher ability of the model to distinguish between class 0 and class 1.

ROC Curve KNN Classifier with undersampling and best parameters



The AUC-ROC curve has a very good AUC score score with 0.88, meaning the model has a 88% chance of distinguishing between the two classes that we want to predict.

Model Performance across Location

To analyze our results in more detail, we wanted to look at the model accuracy, precision, recall and F1 scores across locations in detail for the best model.

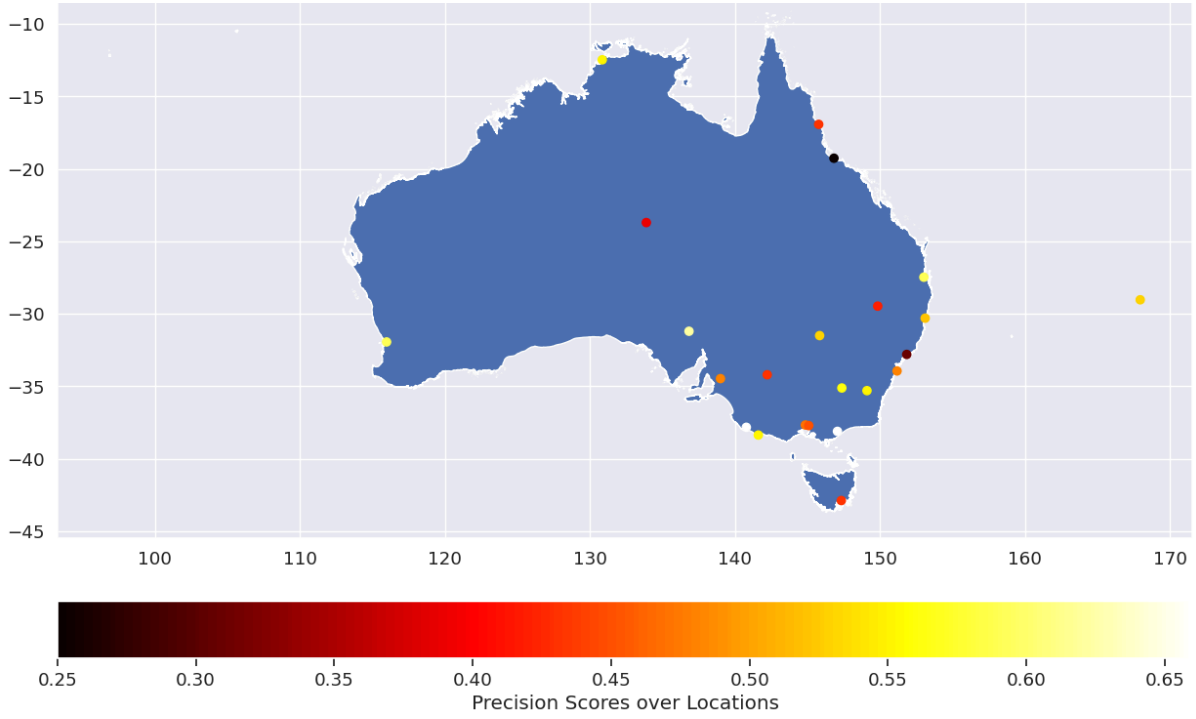
Results for KNN Classifier with undersampling

	Location	Accuracy	Precision	Recall	F1
0	Cobar	0.81	0.38	0.72	0.50
1	CoffsHarbour	0.78	0.60	0.78	0.68
2	Moree	0.85	0.43	0.85	0.57
3	NorfolkIsland	0.73	0.55	0.67	0.61
4	SydneyAirport	0.77	0.53	0.79	0.63
5	WaggaWagga	0.82	0.51	0.80	0.62
6	Williamtown	0.79	0.53	0.78	0.63
7	Canberra	0.80	0.46	0.69	0.56
8	Sale	0.75	0.45	0.76	0.56
9	MelbourneAirport	0.77	0.45	0.79	0.57
10	Mildura	0.84	0.41	0.85	0.55
11	Portland	0.78	0.67	0.78	0.72
12	Watsonia	0.78	0.53	0.79	0.63
13	Brisbane	0.78	0.47	0.75	0.58
14	Cairns	0.78	0.59	0.74	0.65
15	Townsville	0.83	0.56	0.79	0.65
16	MountGambier	0.82	0.67	0.80	0.73
17	Nuriootpa	0.82	0.50	0.78	0.61
18	Woomera	0.87	0.28	0.64	0.39
19	PerthAirport	0.83	0.54	0.84	0.66
20	Hobart	0.75	0.46	0.68	0.55
21	AliceSprings	0.85	0.32	0.82	0.46
22	Darwin	0.82	0.61	0.83	0.71

The first thing we can notice about the scores is that the precision scores for all regions are generally lower than their respective recall scores. Most of them are barely better than what we get with a random classification. We can also note that the range of different precision scores is quite large from 0.28 up to 0.67. The range of the recall scores is not as large from 0.63 to 0.85. The accuracy range is even lower from 0.73 to 0.85. In terms of accuracy we can almost say that the scores are persistent over locations but looking at the details, we see that they are actually quite different. Therefore we conclude it could be interesting to group locations based on similar precision or recall scores and do the KNN modelling again.

Our further analysis focuses on the precision scores. As noted above there is very high variation for precision, so we were wondering if we can detect

any pattern when looking at the distribution of precision scores across locations on a map of Australia. For example maybe inland locations have higher scores than seaside locations.



Unfortunately we cannot see any interesting clusters such as inland vs seaside, north versus south or east vs west, on the map. High as well as low precision scores are spread all over the continent. We can see that most of the locations are in the Southeast but they do not share similar values. On the contrary, sometimes very remote island locations share similar scores with certain seaside locations in the Southeast.

Hence we tried a different approach and looked at the locations with the best precision scores, i.e. over 0.60. We selected data only from these locations (1795 observations in total) and using these observations, we recomputed the KNN model, i.e. KNN with random undersampling. Then we computed the classification report again.

	precision	recall	f1-score	support
0.0	0.90	0.78	0.84	1237
1.0	0.63	0.81	0.70	558
accuracy			0.79	1795
macro avg	0.76	0.79	0.77	1795
weighted avg	0.81	0.79	0.80	1795

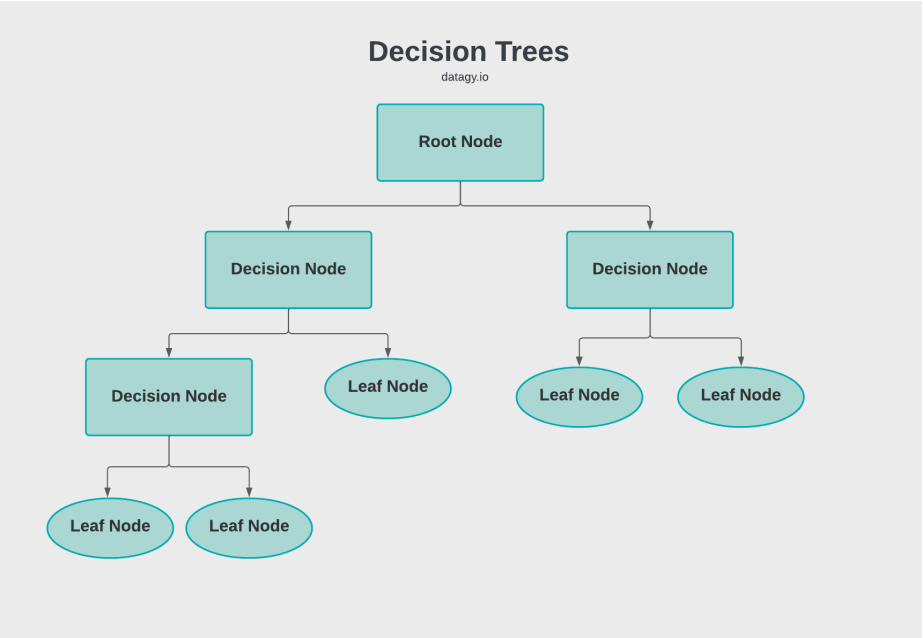
Indeed, we see a slight improvement in the predictions. Before we had a precision score close to 0.5 for class 1 and now it is at 0.63. The precision, recall and f1 score did not significantly change for class 0 and the recall as well as f1 scores for class 1 also did not change. The model accuracy also didn't change.

It would be interesting to take the analysis further and see if we get better results for the locations with particular low recall score, when using a different KNN approach with random oversampling for example or other parameters. However if we pursue this modelling strategy, we might not get a pertinent model for all locations, we would get several different models for very specific locations.

2.1.3 Decision Tree Classification

A) Modelling Steps

Decision Tree Classification is a supervised learning algorithm that classifies instances into one of several classes. The algorithm starts to split the entire dataset at the root node and then selects a feature as well as a split point which allows further division of the dataset into two or more subsets (nodes). The criteria we considered for splitting the dataset are gini impurity and entropy. This process is repeated recursively for each child node until the stopping criteria such as maximum depth or no further improvement in impurity measures is met. The nodes become Leaf Nodes once the dataset cannot be further split. To classify a new instance, the model runs down the whole decision tree to a Leaf Node. The class label of the Leaf Node is then assigned to the new instance. Decision Tree classification is widely used because it can be easily interpreted and can handle non linear relationships.



As before with the KNN model we first computed a baseline model with the default parameters. Second we tried hyperparameter tuning to get better results. Third we applied oversampling and undersampling methods. Finally, we did some hyperparameter tuning with the best oversampling method and on top we also applied boosting.

For the baseline classification we ran the model with the default parameters, i.e gini as split criterion and no maximum depth.

Baseline Classification Report for Decision Tree Model

	precision	recall	f1-score	support
0.0	0.88	0.86	0.87	10663
1.0	0.51	0.55	0.53	2882
accuracy			0.79	13545
macro avg	0.70	0.70	0.70	13545
weighted avg	0.80	0.79	0.80	13545

While the performance of the model is quite good for class 0, the results are not satisfactory at all for class 1. They are barely better than what random classification would give us.

Therefore we tuned the hyperparameters with GridSearch with a very small range of parameters because of our limited computing power:

- Split criterion: gini, entropy
- Max Depth: range from 2 to 12

The best parameters retained were:

- Split criterion: entropy
- Max Depth: 8

When running the model again with the best parameters of our grid search results we got better scores for the precision score for class 1 with 0.68. But the recall score did not improve. The scores of class 0 continued to be high.

Again, we implemented over- and undersampling strategies to see if we can increase the recall scores of class 1. All the strategies managed to greatly increase the recall score of class 1 (between 0.72 and 0.83) but at the same time greatly decreased the precision score of class 1 (around 0.5). When applying SMOTE the model featured the greatest accuracy with 0.791 which is less than the hyperparameter tune baseline model (0.843).

Classification Report for Decision Tree Model with SMOTE

	precision	recall	f1-score	support
0.0	0.91	0.81	0.86	10663
1.0	0.51	0.72	0.59	2882
accuracy			0.79	13545
macro avg	0.71	0.76	0.73	13545
weighted avg	0.83	0.79	0.80	13545

We wanted to see if we can increase our precision score from the model with SMOTE by hypertuning the parameters. Unfortunately, the scores did not change. So, we tried adaptive boosting. It combines the predictions of multiple weak learners in a sequential manner, focusing more on misclassified instances to iteratively improve the overall classification performance of the model.
With SMOTE oversampling, hyperparameter tuning and boosting, we got the following performance scores for Decision Tree Classification.

Classification Report for Decision Tree Model with SMOTE, hyperparameter tuning and boosting

	precision	recall	f1-score	support
0.0	0.90	0.92	0.91	10663
1.0	0.69	0.64	0.66	2882
accuracy			0.86	13545
macro avg	0.80	0.78	0.79	13545
weighted avg	0.86	0.86	0.86	13545

As we can see, adaptive boosting with SMOTE leads to better model performance. The scores for class 0 are all above or equal to 0.90. For class 1 we get scores above 0.6 which is better than what we would get with random classification. While both precision and recall could be better, adaptive boosting gave us the best balance, i.e we don't have one very high score and one very low score. In terms of model accuracy (0.86) , the model is also better than the hyperparameter tuned baseline model (0.84).

We checked the persistence of our results with cross validation and got persistent scores over 5 test folds around 0.85 for precision, 0.86 for recall and 0.85 for the F1 score.

Fold	Test Precision	Test Recall	Test F1-Score
1	0.854	0.862	0.853
2	0.850	0.859	0.848
3	0.854	0.862	0.853
4	0.851	0.860	0.851
5	0.856	0.864	0.855

Note, we used only 5 test folds because of our limited computing power.

B) Interpretation of Results

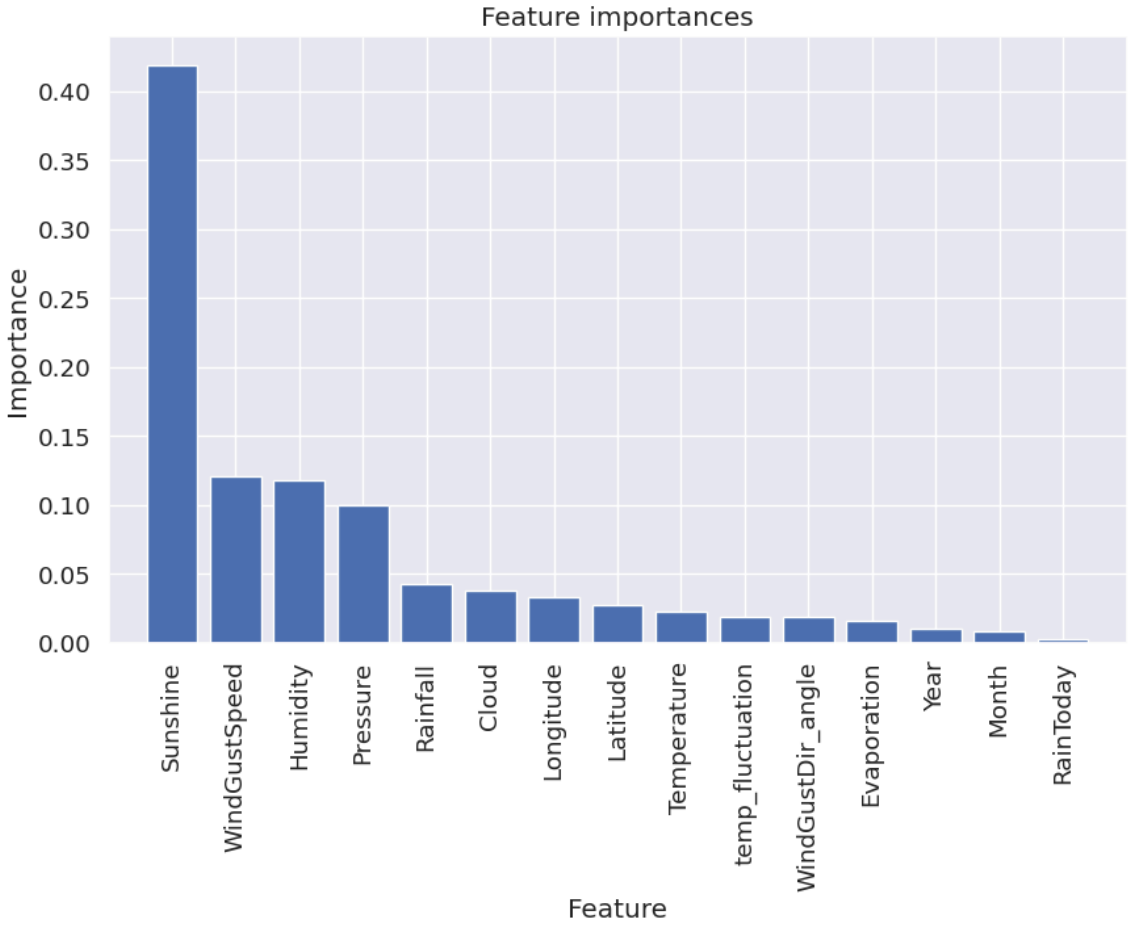
Feature Importance

First we looked at the feature importance. We wanted to use SHAP, but it took too long to be computed. So we just used the built- in feature_importance_ for the Decision Tree Classifier.

Decision Tree Classifier with SMOTE and best parameters

Note for some reason, we are not able to compute the feature importance for the classifier with adaptive boosting. We computed the feature importance for the next best model.

```
Feature ranking:
1. feature 5 (0.419012) - Sunshine
2. feature 6 (0.121101) - WindGustSpeed
3. feature 13 (0.118319) - Humidity
4. feature 11 (0.100018) - Pressure
5. feature 3 (0.043140) - Rainfall
6. feature 10 (0.038121) - Cloud
7. feature 9 (0.033634) - Longitude
8. feature 8 (0.027890) - Latitude
9. feature 12 (0.022708) - Temperature
10. feature 14 (0.018782) - temp_fluctuation
11. feature 7 (0.018678) - WindGustDir_angle
12. feature 4 (0.016562) - Evaporation
13. feature 0 (0.010422) - Year
14. feature 1 (0.008479) - Month
15. feature 2 (0.003131) - RainToday
```



According to the feature importance method of our decision tree model, the hours of Sunshine play the most important role when it comes to the decision to classify certain instances. With a value of 0.42 it is way more important than the other values which are between 0.003 and 0.12. The

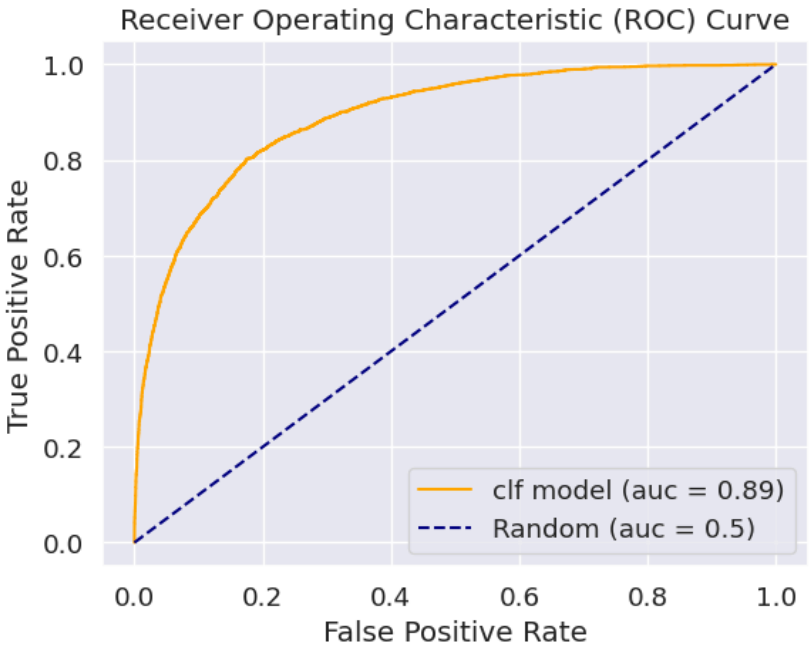
second, third and fourth most important features (WindGustSpeed, Humidity, Pressure) are all over 0.10.

While it is not surprising that the model relies heavily on hours of sunshine to predict rainy days, it is interesting to note that WindGustSpeed is the second most important feature. We thought that humidity, Pressure and also RainToday would play a bigger role in the classification of rainy days.

AUC - ROC Curve

The AUC-ROC curve looks not much different from the one we get with the KNN model. It has only a slightly better AUC score for the Decision Tree Classifier of 0.89, i.e. there is an 89% chance that the model will be able to distinguish between rainy days and non rainy days.

ROC Curve Decision Tree Classifier with SMOTE, best parameters and adaptive boosting



Model Performance across Locations

As before, we looked at the model accuracy, precision, recall and F1 scores across locations in detail for the best model.

Results for Decision Tree Classifier with SMOTE

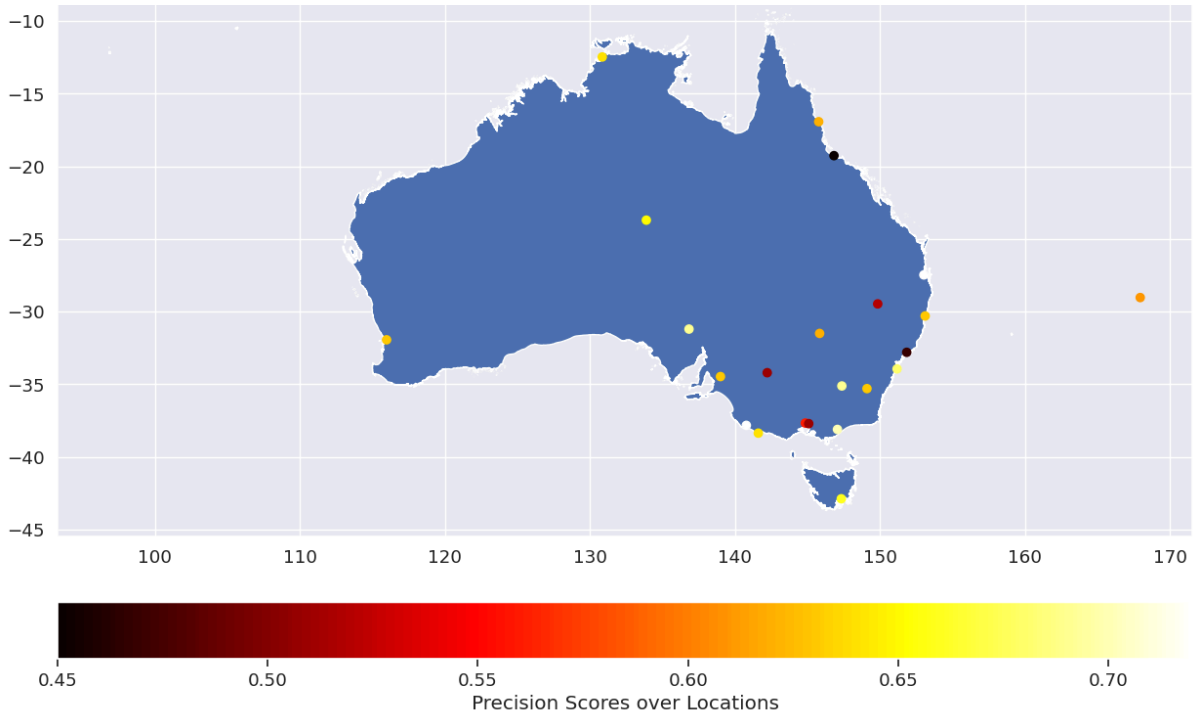
	Location	Accuracy	Precision	Recall	F1
0	Cobar	0.91	0.64	0.67	0.65
1	CoffsHarbour	0.83	0.73	0.66	0.69
2	Moree	0.91	0.58	0.69	0.63
3	NorfolkIsland	0.77	0.63	0.60	0.62
4	SydneyAirport	0.81	0.62	0.64	0.63
5	WaggaWagga	0.87	0.66	0.63	0.65
6	Williamtown	0.83	0.64	0.60	0.62
7	Canberra	0.87	0.63	0.63	0.63
8	Sale	0.81	0.55	0.56	0.56
9	MelbourneAirport	0.80	0.48	0.54	0.51
10	Mildura	0.89	0.54	0.62	0.58
11	Portland	0.79	0.72	0.72	0.72
12	Watsonia	0.80	0.57	0.66	0.61
13	Brisbane	0.85	0.60	0.63	0.62
14	Cairns	0.80	0.63	0.65	0.64
15	Townsville	0.85	0.62	0.59	0.60
16	MountGambier	0.83	0.70	0.76	0.73
17	Nuriootpa	0.89	0.66	0.75	0.70
18	Woomera	0.93	0.49	0.46	0.47
19	PerthAirport	0.89	0.70	0.80	0.75
20	Hobart	0.79	0.54	0.54	0.54
21	AliceSprings	0.92	0.48	0.56	0.52
22	Darwin	0.85	0.70	0.76	0.73

The first thing that jumps into the eyes are the precision scores that seem to be higher across locations than for the KNN model. Also there is less variation across regions with a range of 0.48 to 0.73.

We can also note that precision and recall scores are quite close to each other. In the KNN model, recall scores were definitely higher than the precision score. Here we get a more balanced result. That's in line with what we get when computing the overall classification report.

Model accuracy across regions is also higher than in the KNN model. We barely get any region with accuracy scores below 0.8

Since the variation of precision scores is not as big as for the KNN model, and precision scores are generally close to recall scores, we decided to focus on recall scores this time and wanted to check if we can identify clusters of similar recall scores on a map of Australia.



Just like for the KNN model, we cannot see a real tendency for the recall scores. The high scores seem to be spread all over the continent. We can observe that the lower scores seem to cluster in the Southeast but we also can find lots of high scores in the Southeast.

As the spread of recall scores remains rather inconclusive we select the observations with the highest scores and create a new dataset (with 3550 observations) which we use to recompute the decision tree model with SMOTE, best parameters and adaptive boosting. Based on the results we get the following classification report:

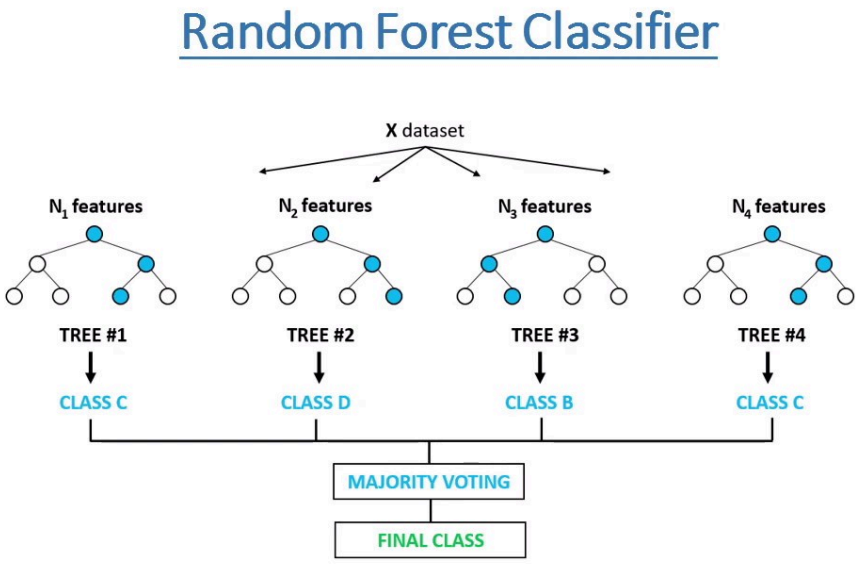
	precision	recall	f1-score	support
0.0	0.91	0.91	0.91	2699
1.0	0.72	0.72	0.72	851
accuracy			0.87	3550
macro avg	0.82	0.81	0.82	3550
weighted avg	0.87	0.87	0.87	3550

We get almost the same precision, recall or f1 scores for the class 0. Not surprisingly we can observe the greatest improvement for the recall score of class 1 from 0.64 to 0.72. Although we do get better scores for class 1, we do not get significantly higher scores that would justify pursuing this strategy. Maybe we could try some hyperparameter tuning for these observations to get better results, or include more/less observations.

2.1.4 Random Forest Classification

A) Modelling Steps

Random Forest Classification is an ensemble learning method which builds a multitude of decision trees during training and outputs the class that is the mode of the classes of the individual trees. More precisely, it works with bootstrap sampling by creating random samples with replacement from the original dataset. From each sample a decision tree is trained. The features of the decision trees, i.e. at each split, are randomly selected to reduce correlation between individual trees. Finally once the trees have been constructed, the voting phase starts and a classification prediction is given to each tree in the forest. The one with the most votes is selected for final prediction. Random Classification is very accurate, robust and allows to easily handle high dimensional data sets. It is also less prone to overfitting due to the averaging effect to the multiple trees.



As before, let's first have a look at the baseline model using the default parameters,

Baseline Classification Report of Random Forest

	precision	recall	f1-score	support
0.0	0.88	0.95	0.91	10663
1.0	0.74	0.52	0.61	2882
accuracy			0.86	13545
macro avg	0.81	0.73	0.76	13545
weighted avg	0.85	0.86	0.85	13545

Overall, the model performed quite well in terms of precision, recall, and F1-score for class 0, but its performance is notably lower for class 1, especially the recall and F1 scores are low . This indicates that the model struggles more with correctly identifying instances of class 1, i.e. the days with rains. The result is quite consistent with what we find in our previously baseline models: Good scores for class 0, low scores for class 1.

To get betters results we did some gridsearch on the following parameters:

- Split criterion: gini, entropy
- Number of trees: 50, 150, 200
- Max Depth of tree : 10, 20, 30
- Minimum number of samples required to split an internal node : 3, 5
- Minimum number of samples required to be at a leaf node : 2, 4

The parameters retained are:

- Split criterion: entropy
- Number of trees: 150
- Max Depth of tree : 20
- Minimum number of samples required to split an internal node : 5
- Minimum number of samples required to be at a leaf node : 2

The GridSearch did not lead to any improvements and as before the main problem seems to be that the dataset is unbalanced. That's why we tried to rerun the model with random oversampling, SMOTE and random undersampling.

As before the scores are quite similar for the different methods and we always decrease precision scores and increase recall scores for class 1. However only with the Random Undersampling strategy we get a very low precision score of 0.52 and a very high recall score with 0.8. That is quite close to what we get with the KNN models. For the Random Oversampling and SMOTE we get more balanced results, with both precision and recall scores ranging between 0.6

and 0.7. To continue our analysis we picked the model with random oversampling because it had the best model accuracy.

Classification Report of Random Forest with Random Oversampling

	precision	recall	f1-score	support
0.0	0.90	0.93	0.91	10663
1.0	0.69	0.60	0.64	2882
accuracy			0.86	13545
macro avg	0.79	0.76	0.78	13545
weighted avg	0.85	0.86	0.85	13545

For further improvement of the scores we did hyperparameter tuning and adaptive boosting with random oversampling.. But the results did not improve in any way.

To check the robustness of the results from Random Oversampling, we also used cross validation with 10 test folds:

Fold	Test Precision	Test Recall	Test F1-Score
1	0.858	0.865	0.856
2	0.855	0.863	0.852
3	0.850	0.858	0.846
4	0.848	0.857	0.846
5	0.855	0.863	0.852
6	0.848	0.857	0.847
7	0.852	0.860	0.851
8	0.847	0.856	0.847
9	0.857	0.864	0.855
10	0.856	0.864	0.854

As we can see, the performance of the scores remains stable over all 10 test folds.

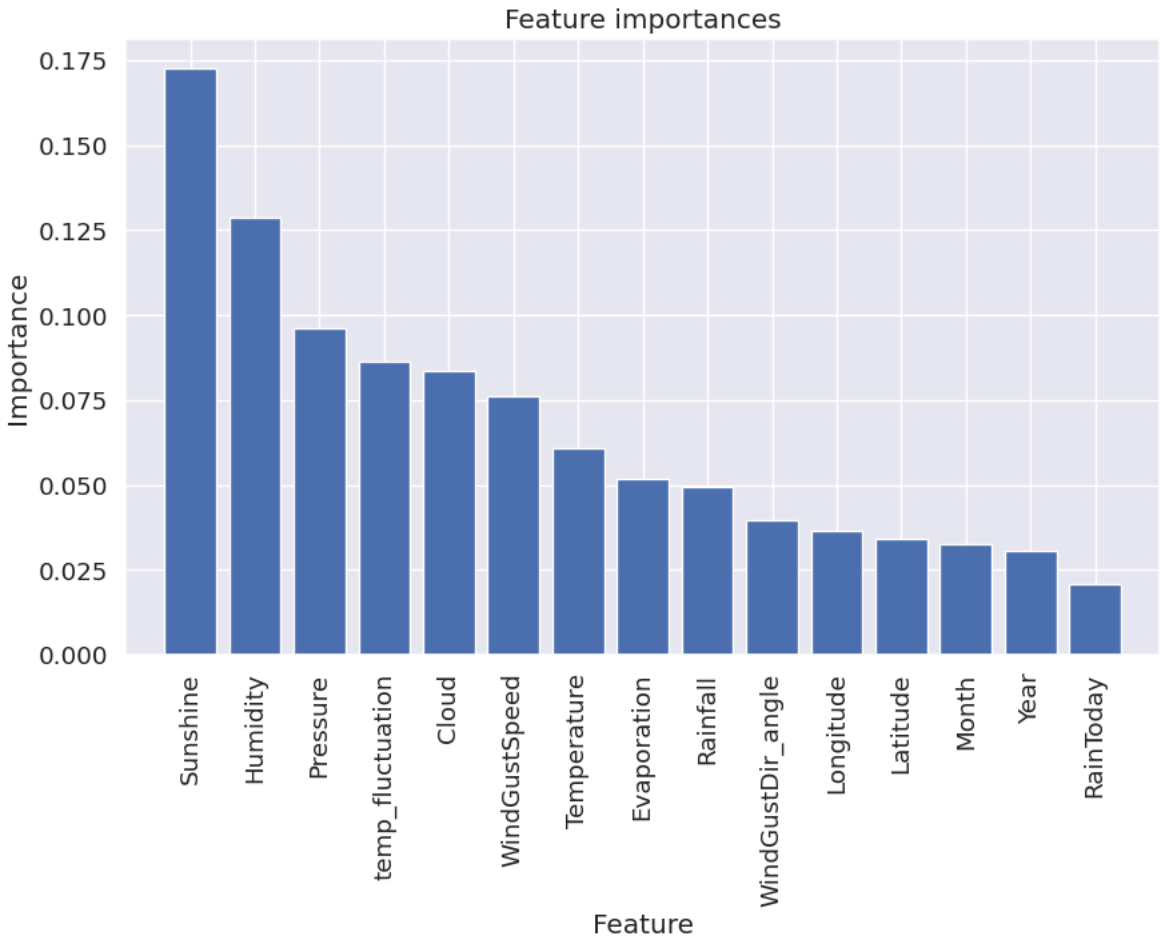
B) Interpretation of Results

Feature Importance

Just as before we looked at the feature importance. Again, we wanted to use SHAP, but it took too long to be computed. So we just used the built- in feature_importance_ for the Random Forest Classifier.

Random Forest Classifier with Random Oversampling

```
Feature ranking:
1. feature 5 (0.172664) - Sunshine
2. feature 13 (0.128890) - Humidity
3. feature 11 (0.095977) - Pressure
4. feature 14 (0.086468) - temp_fluctuation
5. feature 10 (0.083754) - Cloud
6. feature 6 (0.076133) - WindGustSpeed
7. feature 12 (0.060925) - Temperature
8. feature 4 (0.051692) - Evaporation
9. feature 3 (0.049258) - Rainfall
10. feature 7 (0.039446) - WindGustDir_angle
11. feature 9 (0.036431) - Longitude
12. feature 8 (0.034334) - Latitude
13. feature 1 (0.032713) - Month
14. feature 0 (0.030491) - Year
15. feature 2 (0.020822) - RainToday
```



Just like before in the Decision Tree Model, the hours of Sunshine seems to be the most important feature when it comes to predicting rainy days. Yet, it is less dominant than before with a value of only 0.17. It is much closer to the second, third and fourth most important features (Humidity, Pressure, temp_fluctuation) which are all close to 0.10. This time there is no surprise variable in the sense that it is very logical that sunshine hours, humidity,

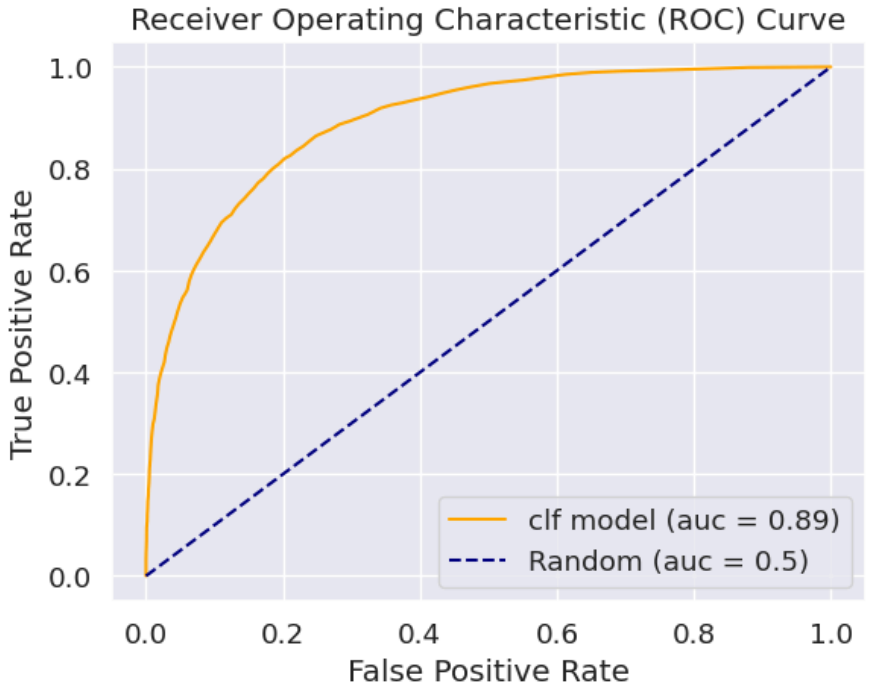
pressure and temp_flucation play the most important role when it comes to classifying rainy days.

However just as before it is quite surprising that RainToday almost plays no role for the classification.

AUC-ROC Curve

The AUC-ROC curve has exactly the same AUC score as the Decision Tree model and we cannot see any difference in the shape.

ROC Curve Random Forest with Random Oversampling



Model Performance across Locations

Again we looked at the model accuracy, precision, recall and F1 scores across locations in detail for the best model (see next page).

Results for Random Forest Classifier with Random Oversampling

	Location	Accuracy	Precision	Recall	F1
0	Cobar	0.82	0.39	0.72	0.51
1	CoffsHarbour	0.78	0.60	0.75	0.66
2	Moree	0.85	0.43	0.85	0.57
3	NorfolkIsland	0.73	0.55	0.64	0.59
4	SydneyAirport	0.77	0.52	0.76	0.62
5	WaggaWagga	0.82	0.51	0.79	0.62
6	Williamtown	0.79	0.54	0.80	0.65
7	Canberra	0.77	0.42	0.70	0.52
8	Sale	0.75	0.44	0.70	0.54
9	MelbourneAirport	0.76	0.44	0.81	0.57
10	Mildura	0.83	0.39	0.86	0.54
11	Portland	0.78	0.67	0.77	0.72
12	Watsonia	0.77	0.50	0.79	0.61
13	Brisbane	0.79	0.48	0.77	0.59
14	Cairns	0.77	0.57	0.72	0.64
15	Townsville	0.83	0.55	0.77	0.65
16	MountGambier	0.80	0.64	0.80	0.71
17	Nuriootpa	0.83	0.51	0.79	0.62
18	Woomera	0.84	0.23	0.62	0.34
19	PerthAirport	0.83	0.55	0.84	0.67
20	Hobart	0.74	0.46	0.68	0.55
21	AliceSprings	0.85	0.31	0.78	0.44
22	Darwin	0.82	0.62	0.84	0.71

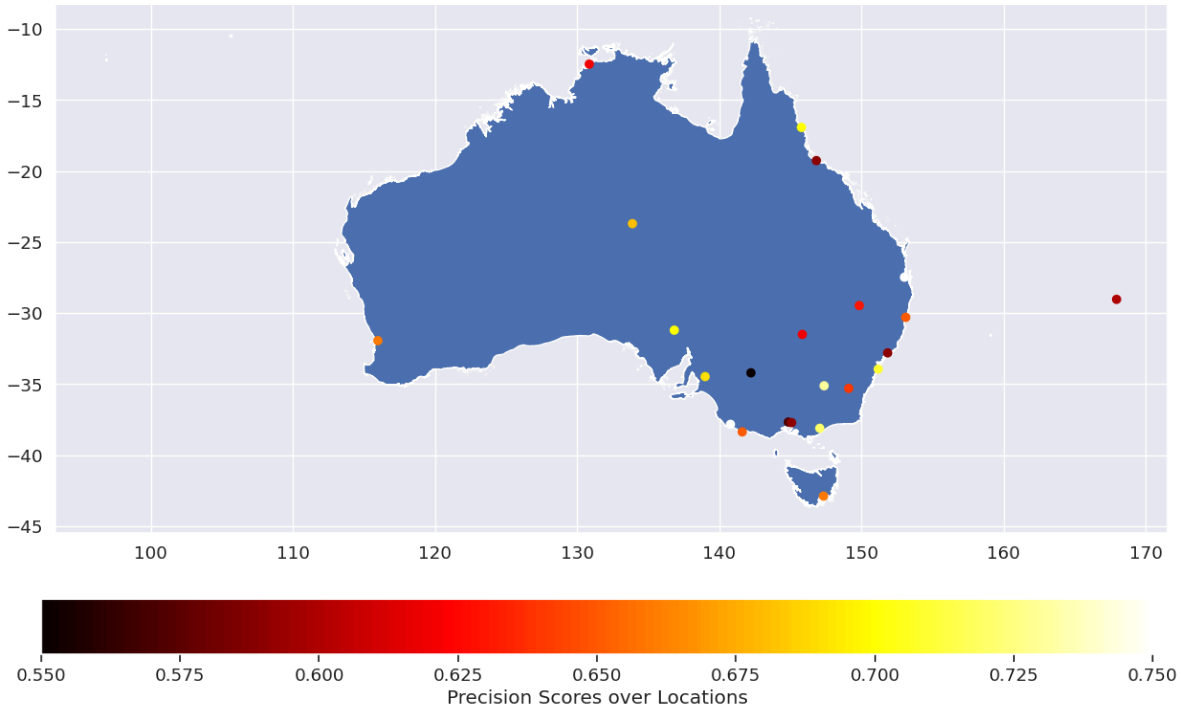
The Random Forest Classifier is more similar to the KNN model than the Decision Tree Model.

On the one hand precision scores across all locations are very low, close to what we get with random classification (0.5) and even lower. Again the range is quite large from 0.23 ro 0.67. Thus the precision scores are not persistent across locations.

On the other hand recall scores are significantly higher and range from 0.62 to 0.86. So they are more persistent across locations.

Finally, the model accuracy is rather high and ranges from 0.73 to 0.85. However it does not compare to the mode accuracy we get with the Decision Tree classifier.

Again, we wanted to see if the high variation of precision scores is linked to certain locational patterns (inland vs seaside locations for example). To do so, we computed a map showing the precision scores across locations.



However as before we cannot see any particular cluster of locations with the similar precision scores. High and low scores can be found all over the continent. Hence we tried again to select data from the locations with the best precision scores, i.e. over 0.70. This left us with 1780 observations. Using these observations, we computed again the classification report for our random forest classifier with random oversampling.

	precision	recall	f1-score	support
0.0	0.89	0.92	0.90	2601
1.0	0.77	0.69	0.73	978
accuracy			0.86	3579
macro avg	0.83	0.81	0.82	3579
weighted avg	0.86	0.86	0.86	3579

This approach led to better scores for class 1 but not for class 0. As expected we get a better precision score for class 1, increasing from 0.69 to 0.77. The recall score also increases 0.6 to 0.69. However the changes are not so massive that it justifies changing our whole strategy and focusing only on locations with similar precision scores. Again it could be interesting to use a different random forest model with different hyperparameters for the locations with very low scores.

2.1.5 XGBoosting

A) Modelling Steps

XGBoost, short for Extreme Gradient Boosting, is an optimized and scalable machine learning library that implements gradient boosting algorithms. It sequentially combines weak learners (typically decision trees) to create a strong predictive model. It incorporates several key features such as regularization, parallelization, tree pruning, and customization options.

The baseline model with the default parameters is as follows:

```
-------------------------
              precision    recall  f1-score   support

         0.0       0.89      0.94      0.92     10663
         1.0       0.73      0.56      0.63      2882

    accuracy                           0.86     13545
   macro avg       0.81      0.75      0.77     13545
weighted avg       0.85      0.86      0.85     13545

-------------------------
Accuracy Train Set: 0.906 Accuracy Test Set: 0.862
```

Although the model performed quite well in terms of precision, recall, and F1-score for class 0, its performance is notably lower for class 1. which is yet again consistent with our other baseline models.

To get betters results we did some gridsearch on the following parameters:

- Number of trees: 100, 200, 300
- Max Depth of tree : 3, 5, 7
- Learning_rate: 0.01, 0.1, 0.3

The parameters retained are:

- Number of trees: 300
- Max Depth of tree : 7
- Learning_rate: 0.1

```
-------------------------
              precision    recall  f1-score   support

         0.0       0.89      0.95      0.92     10663
         1.0       0.74      0.57      0.64      2882

    accuracy                           0.87     13545
   macro avg       0.81      0.76      0.78     13545
weighted avg       0.86      0.87      0.86     13545

-------------------------
Accuracy Train Set: 0.928 Accuracy Test Set: 0.865
```

The GridSearch did not lead to any significant changes to the base model. So, we focussed on the problem of the dataset being unbalanced. That's why we tried to rerun the model with random oversampling, SMOTE and random undersampling.

As before the scores are quite similar for the different methods and we always decrease precision scores and increase recall scores for class 1. With SMOTE the recall increased but the precision decreased.

```
-------------------------
              precision    recall  f1-score   support

         0.0       0.90      0.93      0.91     10663
         1.0       0.71      0.60      0.65      2882

    accuracy                           0.86     13545
   macro avg       0.80      0.77      0.78     13545
weighted avg       0.86      0.86      0.86     13545

-------------------------
Accuracy Train Set: 0.917 Accuracy Test Set: 0.862
```

Contrarily, only with the Random Undersampling strategy we observe the reverse.

```
------------------------
              precision    recall  f1-score   support

         0.0       0.94      0.80      0.87     10663
         1.0       0.53      0.82      0.64      2882

    accuracy                           0.80     13545
   macro avg       0.73      0.81      0.75     13545
weighted avg       0.85      0.80      0.82     13545

------------------------
Accuracy Train Set: 0.863 Accuracy Test Set: 0.804
```

As random undersampling had the highest recall for class 1 we performed optimization for hyperparameters for this model, but the result did not improve.
We checked the robustness of the results from Random undersampling, by cross validating with 10 test folds. The performance of the model remained stable as can be seen below:

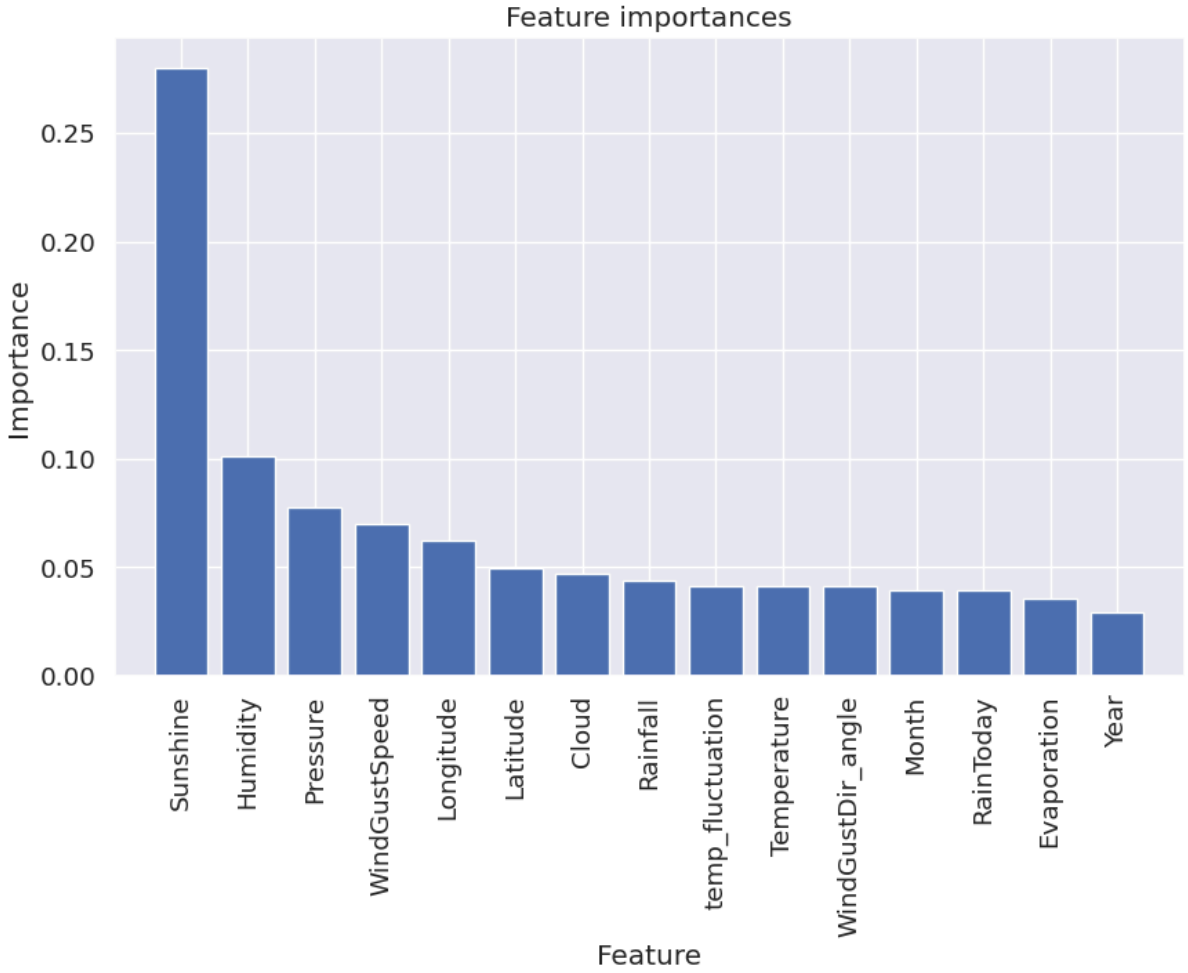
Fold	Test Precision	Test Recall	Test F1-Score
1	0.863	0.870	0.862
2	0.857	0.865	0.857
3	0.856	0.863	0.856
4	0.854	0.861	0.855
5	0.864	0.871	0.864
6	0.851	0.858	0.852
7	0.860	0.867	0.861
8	0.861	0.868	0.861
9	0.857	0.865	0.858
10	0.856	0.864	0.857

B) Interpretation of Results

Feature Importance

XGBoost with Random Undersampling

```
Feature ranking:
1. feature 5 (0.279986) - Sunshine
2. feature 13 (0.100930) - Humidity
3. feature 11 (0.077702) - Pressure
4. feature 6 (0.069781) - WindGustSpeed
5. feature 9 (0.062560) - Longitude
6. feature 8 (0.049839) - Latitude
7. feature 10 (0.047220) - Cloud
8. feature 3 (0.043739) - Rainfall
9. feature 14 (0.041438) - temp_fluctuation
10. feature 12 (0.041434) - Temperature
11. feature 7 (0.041409) - WindGustDir_angle
12. feature 1 (0.039672) - Month
13. feature 2 (0.039154) - RainToday
14. feature 4 (0.035695) - Evaporation
15. feature 0 (0.029442) - Year
```

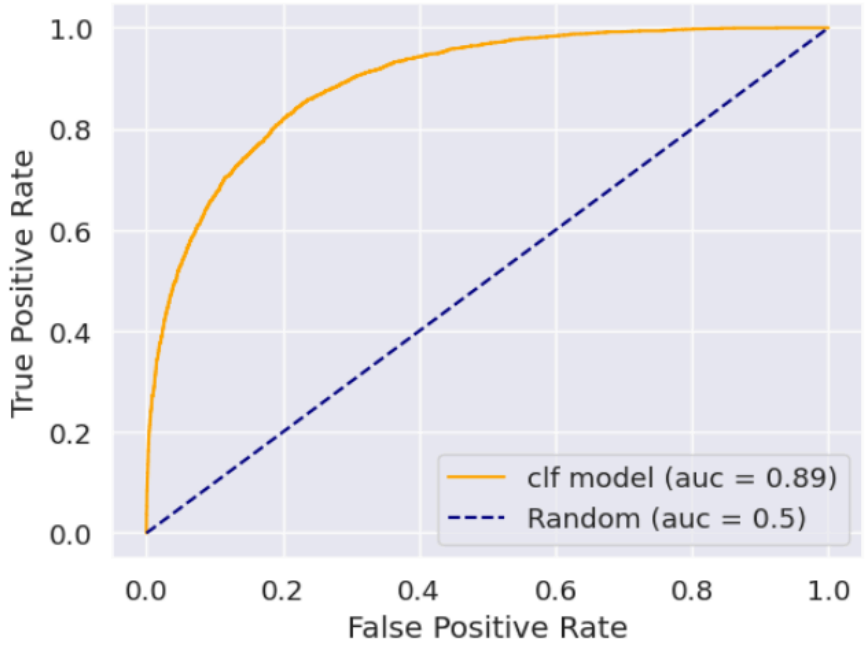


Sunshine remains to be the most important feature for predicting rainy days. Yet, it is less dominant than the decision tree classifier. It is followed with some distance by humidity and pressure and wind speed, with these last two being very close to each other in importance. Interestingly for the first time in our modelling process Longitude has a score over 0.05 for the first time.

AUC-ROC Curve

Again we get an AUC-ROC curve with an AUC score of 0.89 for the XGboosting classifier. There is not much difference in shape either.

ROC Curve XGBoost with Random Undersampling

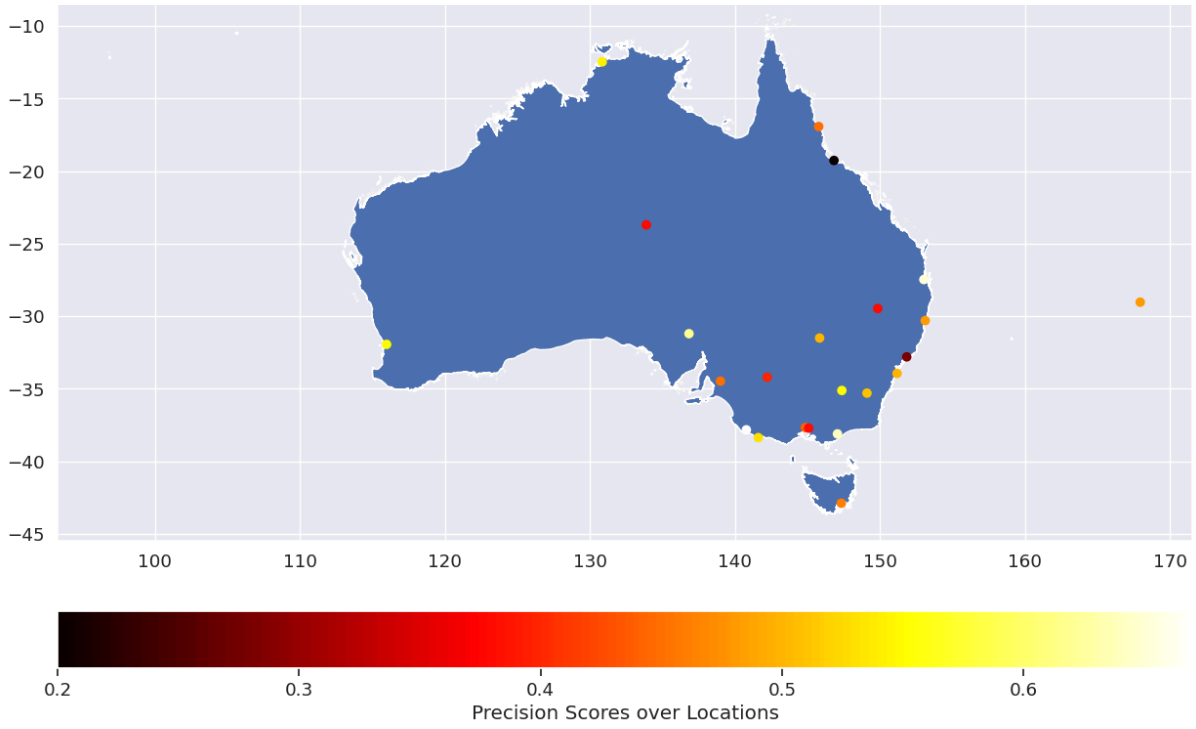


Model Performance across Locations

index	Location	Accuracy	Precision	Recall	F1
0	Cobar	0.86	0.49	0.81	0.61
1	CoffsHarbour	0.79	0.62	0.72	0.66
2	Moree	0.85	0.44	0.89	0.59
3	NorfolkIsland	0.71	0.53	0.69	0.6
4	SydneyAirport	0.75	0.5	0.79	0.61
5	WaggaWagga	0.83	0.53	0.85	0.65
6	Williamtown	0.78	0.52	0.84	0.64
7	Canberra	0.77	0.43	0.81	0.56
8	Sale	0.75	0.45	0.81	0.58
9	MelbourneAirport	0.73	0.41	0.86	0.56

10	Mildura	0.81	0.37	0.89	0.52
11	Portland	0.77	0.66	0.78	0.71
12	Watsonia	0.77	0.51	0.82	0.63
13	Brisbane	0.76	0.44	0.73	0.55
14	Cairns	0.76	0.55	0.75	0.63
15	Townsville	0.82	0.53	0.79	0.64
16	MountGambier	0.81	0.65	0.85	0.74
17	Nuriootpa	0.83	0.51	0.91	0.65
18	Woomera	0.8	0.22	0.82	0.35
19	PerthAirport	0.86	0.6	0.88	0.71
20	Hobart	0.67	0.37	0.68	0.48
21	AliceSprings	0.83	0.3	0.89	0.44
22	Darwin	0.83	0.64	0.81	0.72

We can see that there is very high variation for precision (0.3-0.6), yet we have a more homogenous recall and F1-score, across different locations. Interestingly, we see that the recall here is very high, while the precision rate is lower than the other models we saw previously. This was true for all models that when we try to enhance one parameter the others decrease.
In fact the recall score here is even higher than the previous models (0.69 - 0.89).



However like in other models we cannot see any particular cluster of locations with the similar precision scores. High and low scores can be

found all over the continent, and no clear pattern can be seen. Hence we tried again to select data from the locations with the best precision scores, i.e. over 0.70. Using these observations, we computed again the classification report for our XGBoost with random undersampling.

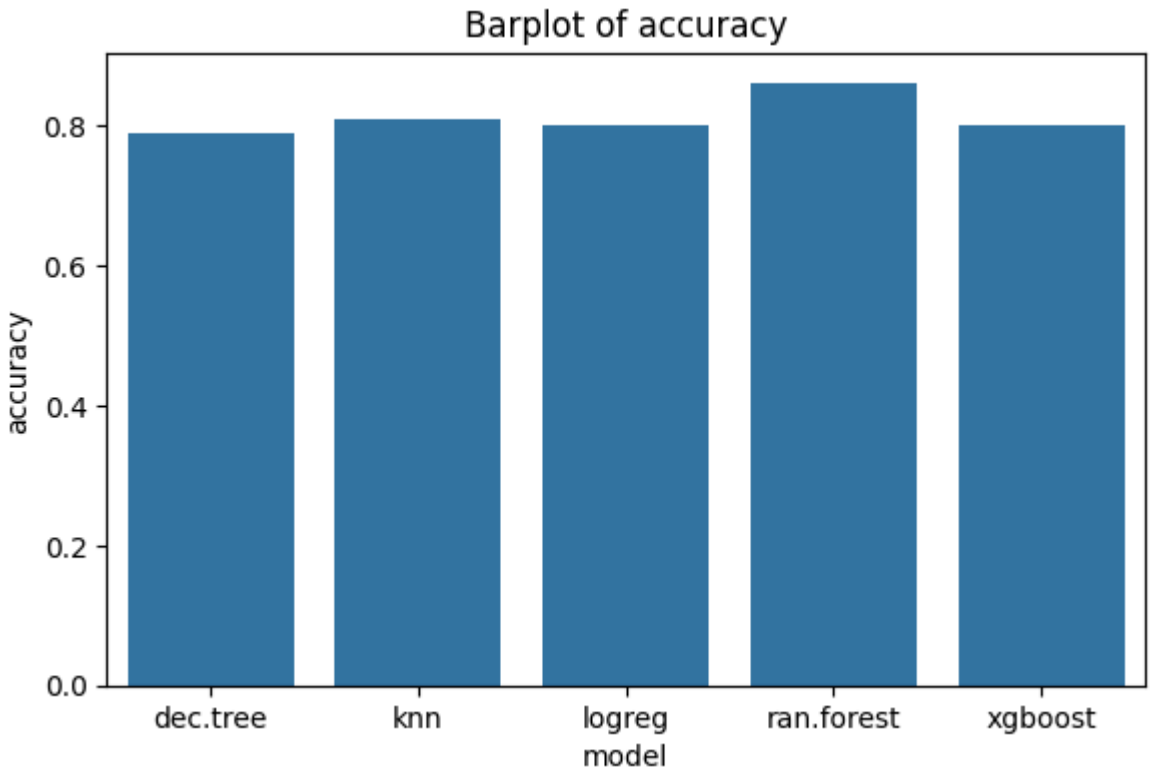
	precision	recall	f1-score	support
0.0	0.91	0.90	0.90	2601
1.0	0.73	0.75	0.74	978
accuracy			0.86	3579
macro avg	0.82	0.83	0.82	3579
weighted avg	0.86	0.86	0.86	3579

This approach led to better scores for class 1 but not for class 0. As expected we get a better precision score for class 1, increasing from 0.53 to 0.73. The recall score decreased from 0.82 to 0.75 but the F1-score increases from 0.64 to 0.74. However the changes are not so massive that it justifies changing our whole strategy and focusing only on locations with similar precision scores. Since two scores increased significantly, it might be interesting to use an independent XGBoost model with different hyperparameters for the locations with low scores.

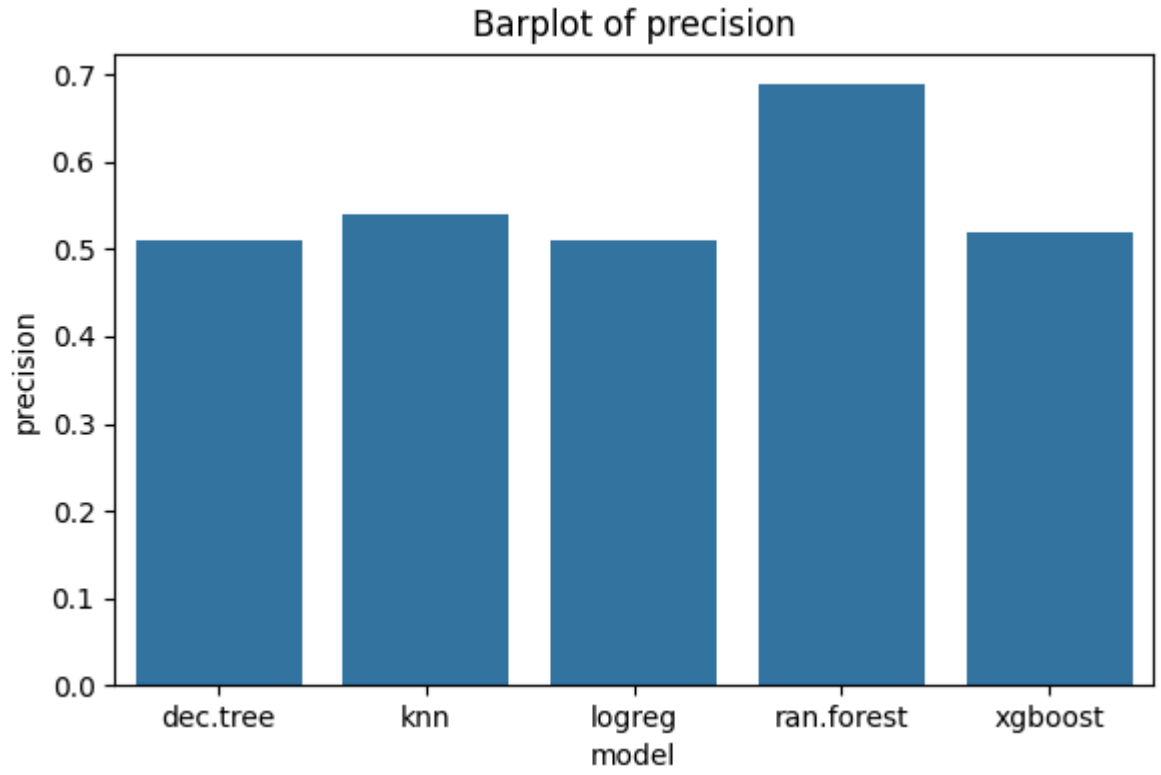
2.1.5 General Conclusion for Machine Learning Models

For the machine learning models, over- and undersampling methods generally increased the recall score of class 1, the days with rain. In the case of the Decision Tree and Random Forest models, adaptive boosting was also helpful to get a balance between precision and recall scores because in the baseline model. GirdSearch was generally not very effective, but due to limited computing power, we couldn't perform very large grid searches.

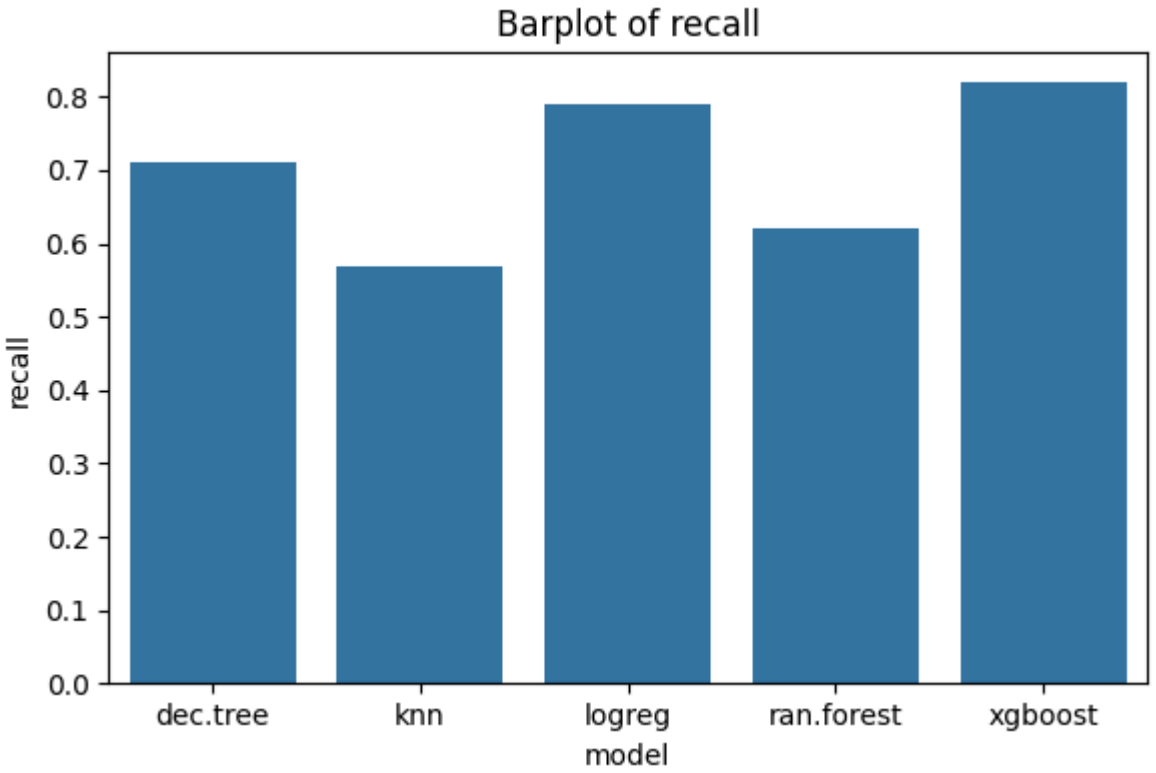
For the best model with the under/oversampling of all machine learning models we compare the different scores. We look at the accuracy, precision, recall and F1 scores for decision tree (dec.tree), knn, logistic regression(logreg), random forest (ran.forest), XGBoost.



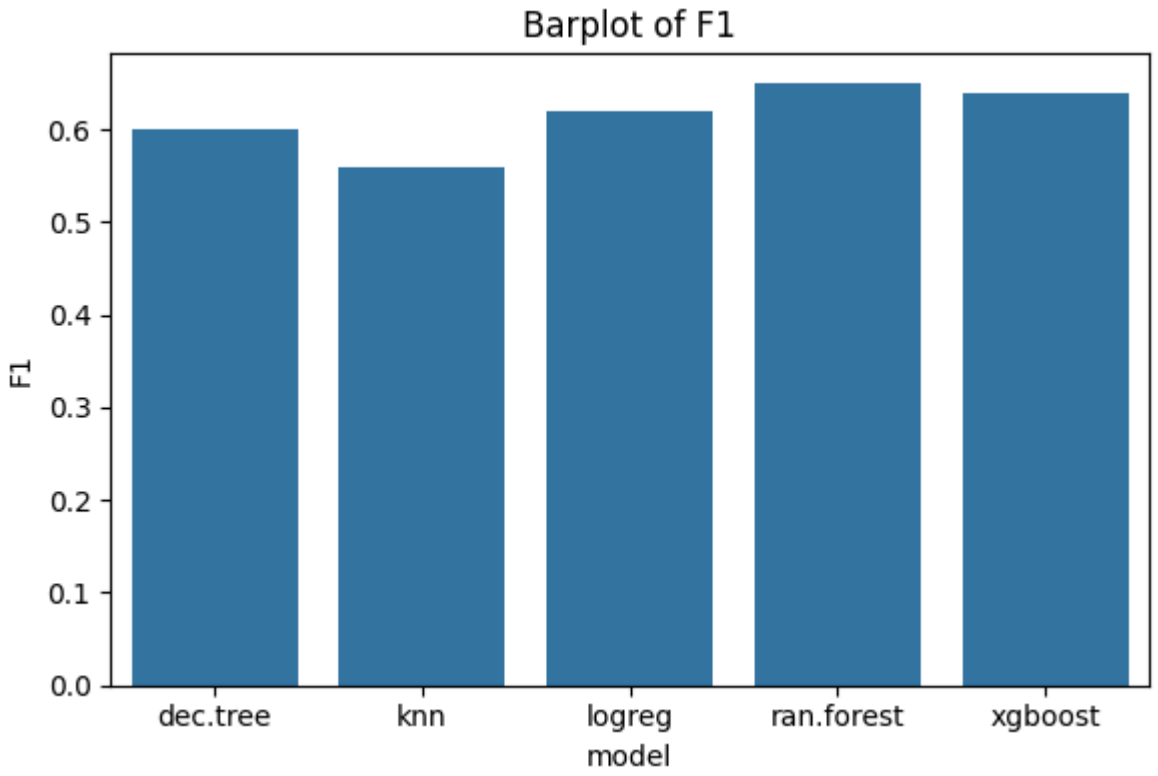
Random forest had the highest accuracy followed by knn and XGBoost. Let's see the pattern for other scores.



Again random forest model had the highest precision alongside accuracy followed by knn and XGBoost.



XGBoost had the highest recall followed by logistic regression and decision tree. Both knn and random forest, which had high scores previously, don't do so well in recall. Let's check the pattern for the last - F1 score.



Once again, random forest had the highest F1-score, followed by XGBoost and logistic regression.

We can see from these graphs that random forest seems to be the best model for three out of four scores, followed by XGBoost. So, we can generally say that different models can be ranked differently based on the scoring method. For us, random forest performs the best amongst the tested machine learning models. Next we will dive deeper into each of these models' performance.

For each model, we previously saw that in general the model performance varies with the location. We looked at the spread of precision and recall over locations on a map for the KNN, Decision Tree and Random Forest model and XGBoost. The idea was to identify clusters of locations with similar scores but unfortunately there were not apparent clusters on the map.

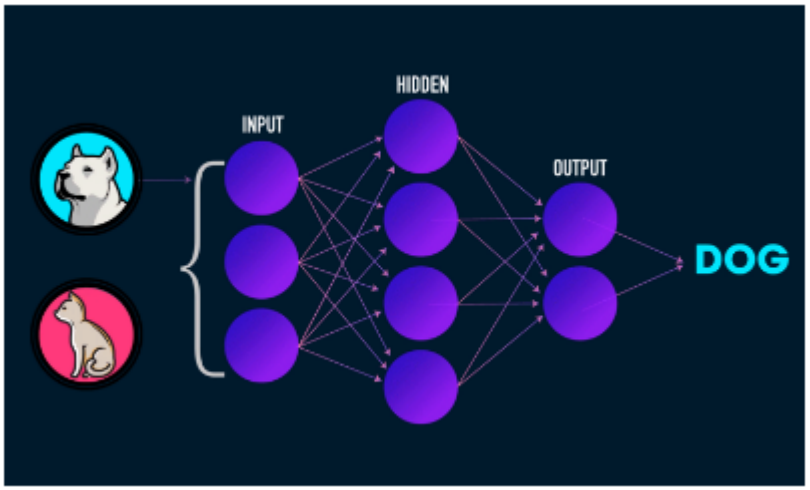
That is why we chose to focus on locations with particular high precision scores (for KNN and Random Forest) or high recall scores (for Decision Tree). We created new dataframes based on these locations and recomputed the respective models with their classification results. Although we generally did get better precision, recall and f1 scores for class 1, the improvement was not highly significant, i.e. does not justify pursuing this strategy. However we could try to optimize certain parameters for the locations chosen and we could also look at the locations with particular low scores.

We also saw from the feature importance that different models weigh the features in different ways, which might stem from variations across the locations. As it would also make more sense that perhaps a change in humidity for example, might be more reliable and stronger in inland regions, than the fluctuation in humidity for coastal regions where humidity is mostly constantly high.

2.2 Deep Learning Models

2.2.1 Dense Neural Network Model (DNN Model), First Approach

We tried to model our binary classification problem with a dense neural network. The idea behind this approach is to mimic the functioning of the human brain, in the sense that you have neurons that process information and are connected between each other. In such a network you have an input and output layer which are connected by several intermediate layers, called hidden layers. These layers have several neurons. The input vector goes through all the layers, each transforming the data before being interpreted and finally returning an output value. Let's look a graphical representation:



This model classifies images of cats and dogs. It has an input layer with 3 neurons, one hidden layer with 4 neurons and an output layer of 2 neurons. Similarly we want to classify rainy and non rainy days.

A) Baseline Model

To do so we first tried a model with 1 layer to establish a baseline model. The model has 1 neuron and a sigmoid activation function to take into account the binary nature of the problem. An activation function, or transfer function is a function applied to the output of each neuron in a neural network layer. This function introduces non-linearity into the network, thus allowing it to learn complex patterns and relationships in the data.

Then we compiled the model with 20 epochs and a batch_size of 100. One epoch is one pass through the entire training dataset. During each epoch, the model goes through each training example and updates its parameters according to the optimization algorithm. The batch size refers to the number of training examples in one iteration.

We used binary_crossentropy for the loss function, i.e. the function that quantifies the difference between the predicted output and the target output. The binary_crossentropy function is typically used for binary classification problems

Classification Report Baseline DNN Model

	precision	recall	f1-score	support
0.0	0.85	0.97	0.91	10652
1.0	0.79	0.39	0.52	2893
accuracy			0.85	13545
macro avg	0.82	0.68	0.72	13545
weighted avg	0.84	0.85	0.83	13545

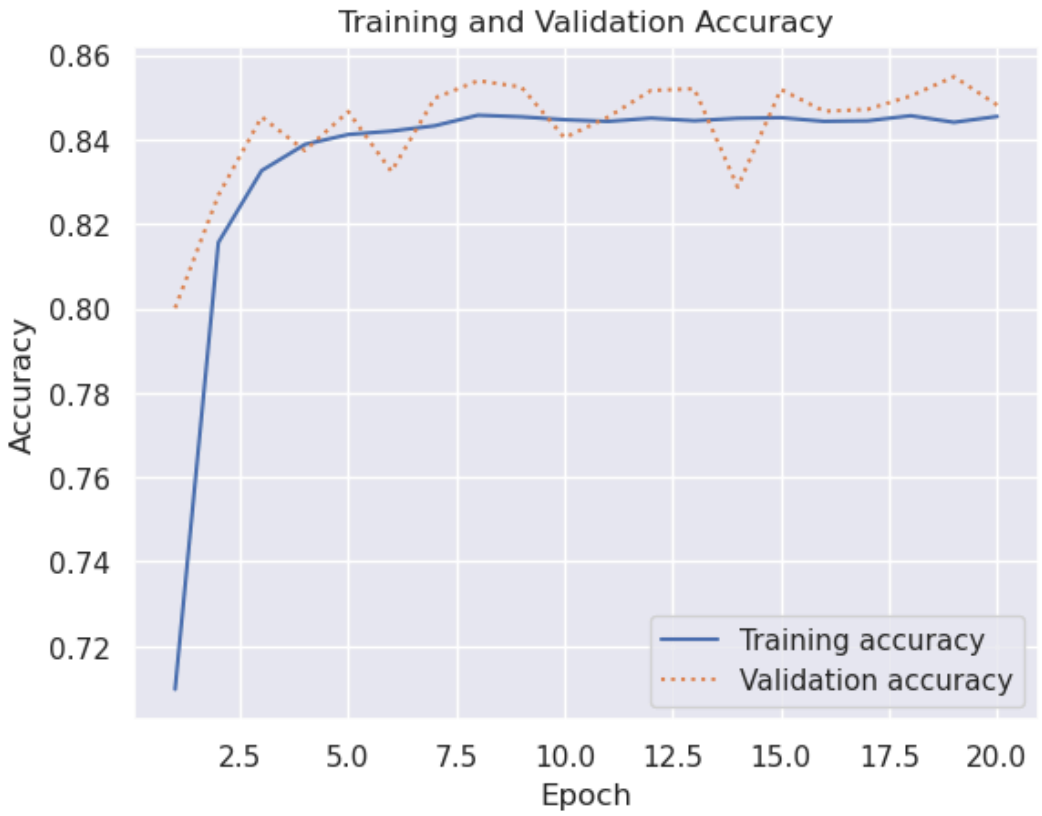
The baseline model seems to perform quite well for class 0 but not for class 1, we see the same problem as before with the Machine Learning models the recall score is extremely low. Another problem is that the model is unstable, i.e. each run leads to a different classification report.

When training the model, we observed that the Gradient Descent converges to different local minima each time we run the model, meaning that the function we are trying to minimize is not convex and the algorithm becomes unpredictable.

Here below you can see an example table summarizing the training history of our baseline model. As you can see the loss function stops decreasing at epoch 9 and then slightly increases and slightly decreases again till epoch 20.

	loss	accuracy	val_loss	val_accuracy
0	5.384057	0.709801	0.451486	0.800074
1	0.413284	0.815578	0.380482	0.826800
2	0.373925	0.832614	0.353367	0.845330
3	0.360681	0.838815	0.367887	0.837357
4	0.358157	0.841159	0.350304	0.846585
5	0.356007	0.841971	0.387911	0.832410
6	0.353748	0.843226	0.347717	0.849760
7	0.351463	0.845736	0.336535	0.853894
8	0.351374	0.845349	0.341294	0.852418
9	0.351210	0.844666	0.366960	0.840384
10	0.353966	0.844260	0.355140	0.845257
11	0.351723	0.845035	0.342186	0.851606
12	0.352170	0.844426	0.339194	0.851975
13	0.352604	0.844998	0.385414	0.828719
14	0.354927	0.845146	0.342282	0.851827
15	0.352332	0.844278	0.351603	0.846659
16	0.354179	0.844408	0.350819	0.847102
17	0.351703	0.845607	0.344412	0.850277
18	0.351326	0.844057	0.336098	0.854854
19	0.352919	0.845441	0.350032	0.848136

Additionally we computed a graph showing the training accuracy vs the validation accuracy over the number of epochs. Note that due to the instability of the model the graph changed each time when rerunning the model (see here below)



We can see that already at epoch 5 the training curve stops increasing in the terms of model accuracy. The validation accuracy curve doesn't follow at all the same pattern as the validation accuracy curve, indicating some overfitting problem.

To get a more stable model with better results, we firstly increased the number of hidden layers to 2.

B) Adding more layers

We add the following 4 layers to the model:

Input Layer: 64 neurons, activation = 'relu'

Dense hidden Layer 1: 32 neurons activation='relu'

Dense hiddenLayer 2: 16 neurons , activation='relu'

Output Layer 3: 1 neuron, activation='sigmoid'

Activation refers to the activation function. Relu stands for Rectified Linear Unit. It is one of the most commonly used activation functions in Deep learning. It will output the input if positive and otherwise will output it to

zero. Again the output layer has a sigmoid activation function due to the binary nature of our classification problem.

Unfortunately adding more layers did not stabilize the model and again we run into the problem of different results each time the model is run. Here below, we have three different classification reports from exactly the same model:

First run:

	precision	recall	f1-score	support
0.0	0.86	0.97	0.91	10652
1.0	0.78	0.42	0.54	2893
accuracy			0.85	13545
macro avg	0.82	0.69	0.73	13545
weighted avg	0.84	0.85	0.83	13545

Second run:

	precision	recall	f1-score	support
0.0	0.92	0.84	0.88	10652
1.0	0.56	0.74	0.64	2893
accuracy			0.82	13545
macro avg	0.74	0.79	0.76	13545
weighted avg	0.85	0.82	0.83	13545

Third run:

	precision	recall	f1-score	support
0.0	0.87	0.96	0.91	10652
1.0	0.75	0.45	0.56	2893
accuracy			0.85	13545
macro avg	0.81	0.71	0.74	13545
weighted avg	0.84	0.85	0.84	13545

As you can see the results are quite different for each run. The first and the second classification report are similar but the second run leads to a completely different classification report. In the first run and third run the recall score of class 1 is very low with 0.42, in the second run it is rather high with 0.74.

We tried to rerun the exact same model with epochs=10 instead of 20 to control for overfitting, batch_size=32 instead of 100 to decrease memory usage and check if this optimization in efficiency affects the results.

Unfortunately the problem of different results each run persists. Below, we can see two very different classification reports from exactly the same model:

First Run:

	precision	recall	f1-score	support
0.0	0.83	0.99	0.90	10663
1.0	0.86	0.24	0.37	2882
accuracy			0.83	13545
macro avg	0.84	0.61	0.64	13545
weighted avg	0.83	0.83	0.79	13545

Second Run:

	precision	recall	f1-score	support
0.0	0.90	0.90	0.90	10663
1.0	0.62	0.61	0.62	2882
accuracy			0.84	13545
macro avg	0.76	0.76	0.76	13545
weighted avg	0.84	0.84	0.84	13545

Because of this model instability, it is hard to conclude whether adding layers to the baseline model improves the precision, recall and f1 scores.

That is why in the following we try to focus on strategies that improve model stability.

C) Gridsearch on Learning Rate, Increasing n° of epochs

First we did a gridsearch on the learning rate and we also increased the number of epochs to 300 when compiling the model to get more stability. ,

We did a gridsearch with the following learning rates: 0.3, 0.2, 0.1, 0.01. It turned out that the best learning rate is 0.03. So we re-ran our model with the latter learning rate and 300 epochs, keeping all the other parameters equal. But again the model is very unstable and we cannot conclude anything from the classification reports.

D) Dropouts and Early Stopping Rate

We added one more layer, dropouts and an early stopping rate to the model. Dropouts and early stopping rate are a good way of preventing overfitting.

We set the dropout rate to 0.25, i.e. 25% of neurons are randomly being dropped out in each layer during training to prevent over dependency on a specific neuron.

The early stopping rate allows monitoring the value loss during training and stopping it when it starts to increase. We set its patience parameter to 5, meaning that after 5 epochs without any improvement in the value loss, we stop the training process. This way we also reduce computing time of the model.

We hoped that with this new strategy, we could stabilize the model. This gives us the following model structure:

Input Layer: 128 neurons, activation='relu'

Dropout: 0.25

Hidden Layer 1: 64 neurons, activation='relu'

Dropout: 0.25

Hidden Layer 2: 32 neurons, activation='relu'

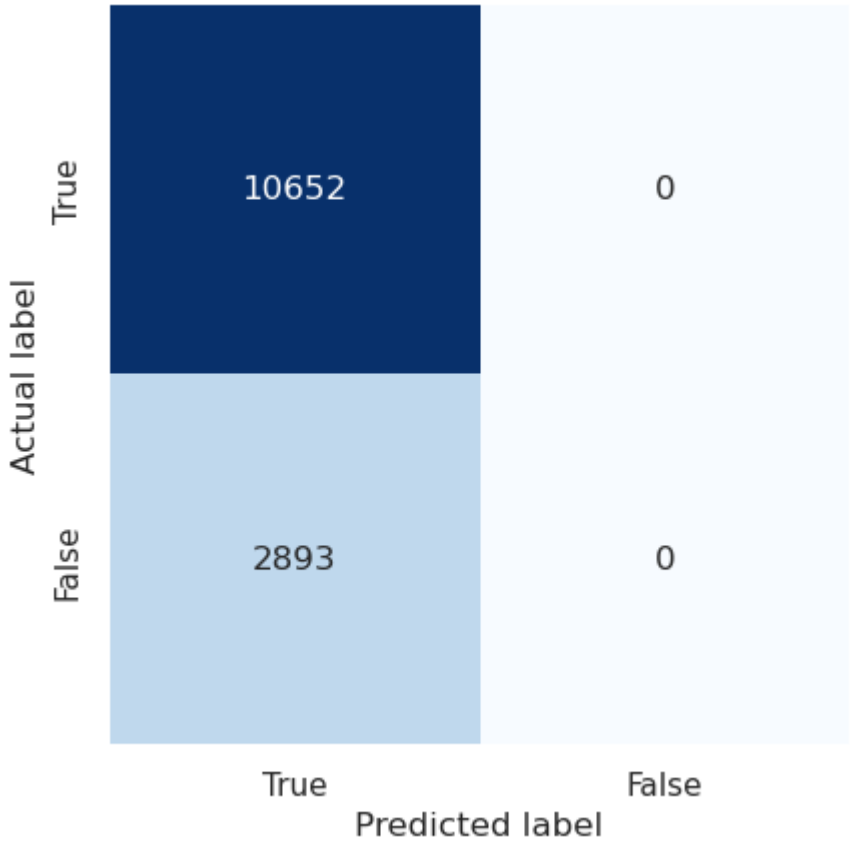
Dropout: 0.25

Hidden Layer 3: 16 neurons, activation='relu'

Dropout: 0.25

Output Layer: 1 neuron, activation='sigmoid'

Just as before the model is quite unstable, but it consistently predicts the true negatives and false negatives to be 0 (see the confusion matrix below)



That also means the precision, recall and f1 scores of class 1 are consistently equal to 0. Adding the dropout seems to be the problem. So we leave it out for the following models

E) Over and undersampling

We tried to compute the same model as before without dropouts and with random oversampling, SMOTE and random undersampling.

For each strategy we computed the classification report several times to check model stability. Unfortunately the models are quite unstable. Sometimes the precision score is very high for class 1 but then when we run the model again it becomes very low. To illustrate the problem we chose to show the two different classification reports of the SMOTE strategy (see below)

SMOTE

First Run

	precision	recall	f1-score	support
0.0	0.89	0.92	0.91	10652
1.0	0.68	0.59	0.63	2893
accuracy			0.85	13545
macro avg	0.78	0.75	0.77	13545
weighted avg	0.85	0.85	0.85	13545

Second Run

	precision	recall	f1-score	support
0.0	0.93	0.83	0.88	10652
1.0	0.55	0.77	0.64	2893
accuracy			0.82	13545
macro avg	0.74	0.80	0.76	13545
weighted avg	0.85	0.82	0.83	13545

While the scores for class 0 are quite stable, the scores for class 1 vary each run. Here we can see that at first the recall score for class 1 was rather low for class 1 with 0.59 then after a second run the recall score for class 1 was significantly higher with 0.77.

Again we cannot draw any conclusions from these classification reports.

F) Other strategies

As a last attempt to stabilize the model we tried batch normalization and learning rate scheduling.

- Batch normalization: is a technique to improve stability and speed of training by normalizing the activations of each layer
- Learning Rate Scheduling: is a technique to adjust the learning rate during training to stabilize the training stability and convergence speed.

We also reduced the number of layers again, going back to the model in section B, the only thing that changes is that first we add batch normalization and in a second time we use learning rate scheduling.

For the learning rate scheduling, we start with the learning rate 0.03, then after 10 epochs we use 0.04 and after 20 epochs 0.05.

Both models remain unstable. So again we cannot conclude anything. See here below two examples of the classification report for Learning Rate Scheduling.

Learning Rate Scheduling
First Run

0.0	0.87	0.95	0.91	10652
1.0	0.72	0.50	0.59	2893
accuracy			0.85	13545
macro avg	0.80	0.72	0.75	13545
weighted avg	0.84	0.85	0.84	13545

Second Run

0.0	0.86	0.97	0.91	10652
1.0	0.77	0.41	0.54	2893
accuracy			0.85	13545
macro avg	0.82	0.69	0.72	13545
weighted avg	0.84	0.85	0.83	13545

2.1.5 General Conclusion for Deep Learning Models

The deep learning models we tried thus far were unstable, the tested strategies like early stopping did not help in improving the prediction scores. We could also try another deep learning algorithm but unfortunately, the gradient descent algorithm is one of the only algorithms that can be used in our context because it is the only efficient optimization algorithm given our computational capabilities.

2.1.5 General Conclusion

In this project we aimed to predict the event of rain the next day using weather data from several locations across Australia collected over approximately a decade from 2009 to 2017. For this we tried several machine learning and deep learning models with the goal of finding an efficient and reliable model.

We quickly realized during data exploration that our dataset had several missing values and it was an imbalance data set. Thus we first tackled the missing values followed by the decision to focus less on accuracy and more on precision, recall and F1 scores due to the imbalance. We also decreased

the bias by using strategies like over/undersampling. Keeping these in mind, we proceeded to the modeling part, where we explored both machine learning and deep learning models.

The deep learning models we tried thus far were unstable, the tested strategies like early stopping did not help in improving the prediction scores, and overall machine learning models were better suited to this dataset, given the things we could explore in this limited time.

Maybe in the end, we come back to the beginning and suggest that our data structure might be a problem where alternative strategies can be tested for improvement in predictability. One approach might include trying other ways to transform the data, or do some feature reduction. Or we could simply try to predict another target like temperature.

Nonetheless, in our modeling so far we think that some machine learning models like random forest and XGBoost are already performing well and we were also able to identify models with improved precision, recall and F1-score as we had set out to do, thus fulfilling the original aim and we have also highlighted areas for further improvement throughout the report, which we hope can be valuable for further work with this dataset and perhaps other similar goals.