Rainfall Prediction and Influence Analysis: A Comparative Study of Machine Learning Techniques on Imbalanced Datasets

Contents

Section-1: Introduction (Research Question and Data Understanding)

1.1 Problem Statement and Business Use Case:

Knowing if it rains the next day is a critical need for all of us in our daily lives as accurate weather predictions help individuals plan their activities for the next day better.
Not only is this useful for normal consumer usage but also for business industries such as the transportation and energy industry which can use the information rarely to prepare and use alternate methods and take precautions. The large-scale success of this project could be used in the agriculture industry where the majority of decisions are made based on the weather conditions.

1.2 Research Question:

This project deals with rain prediction and identifying the key factors behind the influence of rain based on the condition of the previous day. We use the dataset on Kaggle which gives us 10 years of weather observations in Australia across various locations on a daily basis.

Objective here is to run various classification models on this dataset in order to decide which model works best with an imbalanced dataset and to decode which factors affect rain the most.

Source:https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package/data

1.3 Data Dictionary:

Date: The 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: The amount of rainfall recorded for the day in mm

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

WindDir3pm: Direction of the wind at 3pm

WindSpeed9am: Wind speed (km/hr) averaged over 10 minutes prior to 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 eights. It records how many eights 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 (in "oktas": eighths) at 3pm. 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: The amount of next day rain in mm. Used to create response variable RainTomorrow. This is the target/dependent variable.

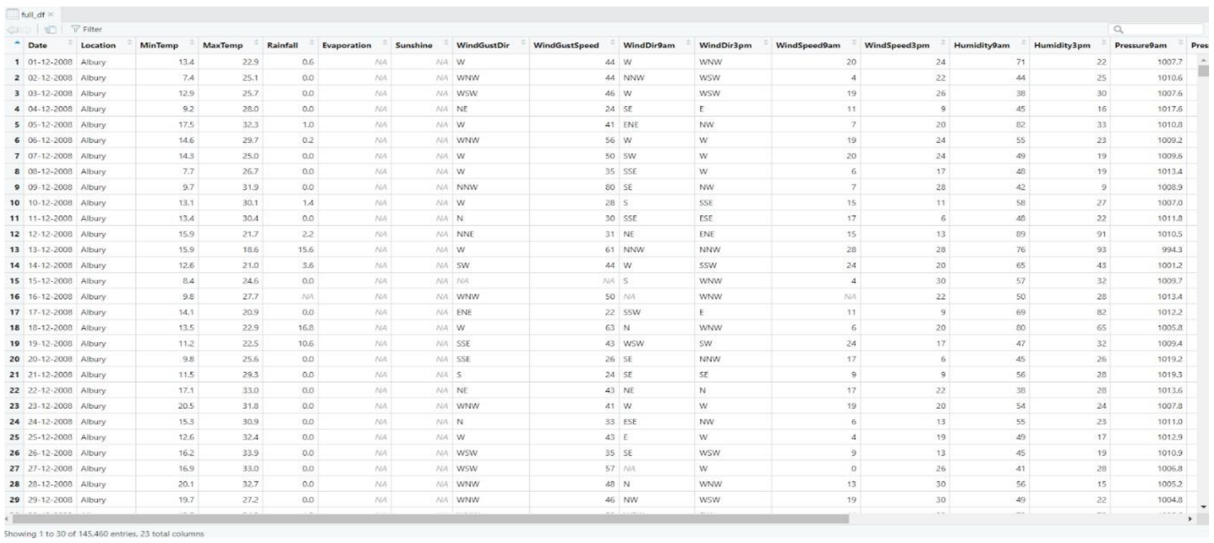
We have 23 columns in which we are using all the columns other than the target variable “Rain Tomorrow” to predict the target column.

Since we have 10 years of data, we have 145,460 observations or rows. It has a mix of outliers and null values along with a class imbalance which we will be looking to overcome.

CRISP-DM Approach – Initially we did the modelling but when we came to the conclusion that the results could be not what the users would want due to the imbalance in prediction, we had to re-iterate the whole process to see any potential improvements and changes to better improve the business requirements.

1.4 Data Loading and Sampling

We have loaded up the dataset in R of which we have sampled 10% i.e.: 14546 rows due to memory constraints, ease of computation and comparison.

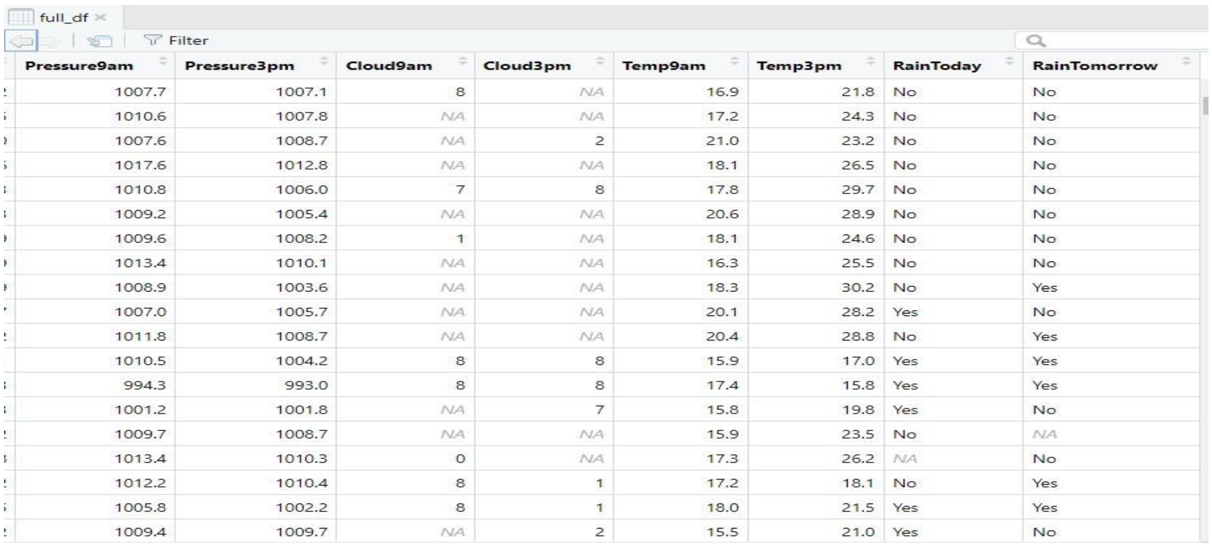


full_df

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am
1	01-12-2008	Albury	13.4	22.9	0.6	NA	NA	W	44	W	WNW	20	24	71	22	1007.7
2	02-12-2008	Albury	7.4	25.1	0.0	NA	NA	WNW	44	NNW	WSW	4	22	44	25	1010.6
3	03-12-2008	Albury	12.9	25.7	0.0	NA	NA	WSW	46	W	WSW	19	26	38	30	1007.6
4	04-12-2008	Albury	9.2	28.0	0.0	NA	NA	NE	24	SE	E	11	9	45	16	1017.6
5	05-12-2008	Albury	17.5	32.3	1.0	NA	NA	W	41	ENE	NW	7	20	82	33	1010.8
6	06-12-2008	Albury	14.6	29.7	0.2	NA	NA	WNW	56	W	W	19	24	55	23	1009.2
7	07-12-2008	Albury	14.3	25.0	0.0	NA	NA	W	50	SW	W	20	24	49	19	1009.6
8	08-12-2008	Albury	7.7	26.7	0.0	NA	NA	W	35	SSE	W	6	17	48	19	1013.4
9	09-12-2008	Albury	9.7	31.9	0.0	NA	NA	NNW	80	SE	NW	7	28	42	9	1008.9
10	10-12-2008	Albury	13.1	30.1	1.4	NA	NA	W	28	S	SSE	15	11	58	27	1007.0
11	11-12-2008	Albury	13.4	30.4	0.0	NA	NA	N	30	SSE	ESE	17	6	48	22	1011.8
12	12-12-2008	Albury	15.9	21.7	2.2	NA	NA	NNE	31	NE	ENE	15	13	89	91	1010.5
13	13-12-2008	Albury	15.9	18.6	15.6	NA	NA	W	61	NNW	NNW	28	28	76	93	994.3
14	14-12-2008	Albury	12.6	21.0	3.6	NA	NA	SW	44	W	SSW	24	20	65	43	1001.2
15	15-12-2008	Albury	8.4	24.6	0.0	NA	NA	NA	NA	S	WNW	4	30	57	32	1009.7
16	16-12-2008	Albury	9.8	27.7	NA	NA	NA	WNW	50	NA	WNW	NA	22	50	28	1013.4
17	17-12-2008	Albury	14.1	20.9	0.0	NA	NA	ENE	22	SSW	E	11	9	69	82	1012.2
18	18-12-2008	Albury	13.5	22.9	16.8	NA	NA	W	63	N	WNW	6	20	80	65	1005.8
19	19-12-2008	Albury	11.2	22.5	10.6	NA	NA	SSE	43	WSW	SW	24	17	47	32	1009.4
20	20-12-2008	Albury	9.8	25.6	0.0	NA	NA	SSE	26	SE	NNW	17	6	45	26	1019.2
21	21-12-2008	Albury	11.5	29.3	0.0	NA	NA	S	24	SE	SE	9	9	56	28	1019.3
22	22-12-2008	Albury	17.1	33.0	0.0	NA	NA	NE	43	NE	N	17	22	38	28	1013.6
23	23-12-2008	Albury	20.5	31.8	0.0	NA	NA	WNW	41	W	W	19	20	54	24	1007.8
24	24-12-2008	Albury	15.3	30.9	0.0	NA	NA	N	33	ESE	NW	6	13	55	23	1011.0
25	25-12-2008	Albury	12.6	32.4	0.0	NA	NA	W	43	E	W	4	19	49	17	1012.9
26	26-12-2008	Albury	16.2	33.9	0.0	NA	NA	WSW	35	SE	WSW	9	13	45	19	1010.9
27	27-12-2008	Albury	16.9	33.0	0.0	NA	NA	WSW	57	NA	W	0	26	41	28	1006.8
28	28-12-2008	Albury	20.1	32.7	0.0	NA	NA	WNW	48	N	WNW	13	30	56	15	1005.2
29	29-12-2008	Albury	19.7	27.2	0.0	NA	NA	WNW	46	NW	WSW	19	30	49	22	1004.8

Showing 1 to 30 of 145,460 entries, 23 total columns

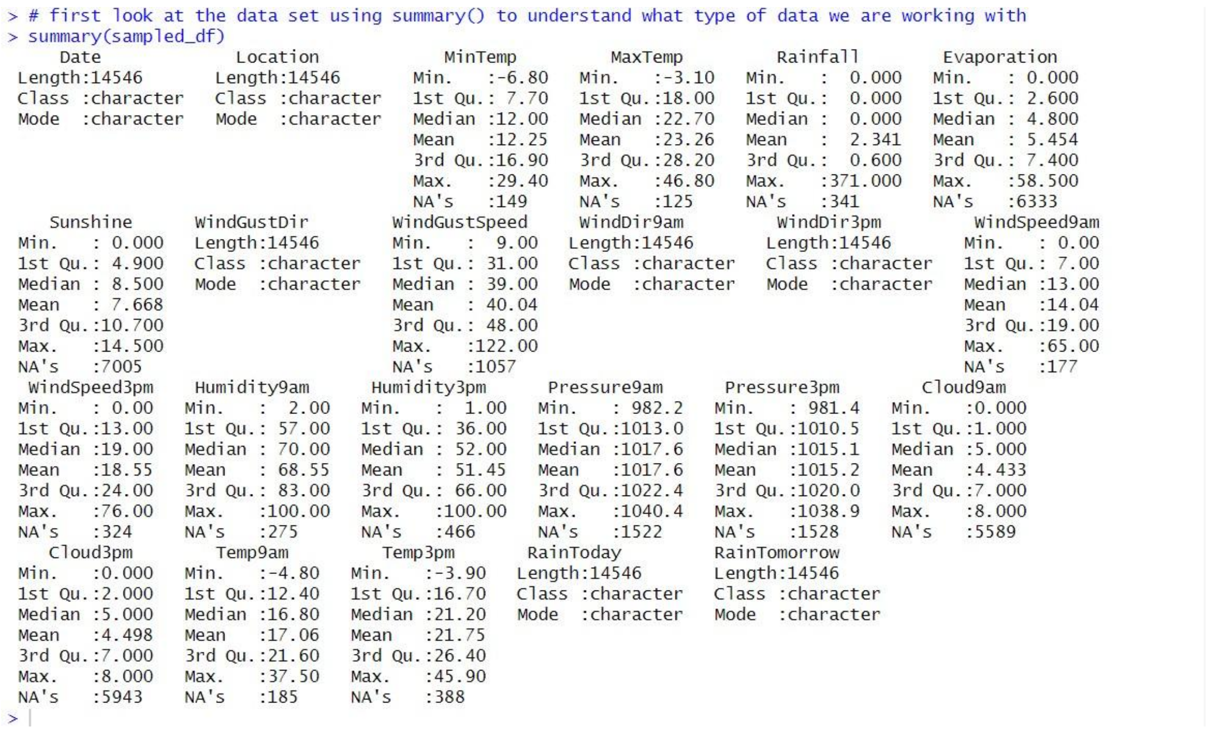
Fig 1.4.1 Initial Dataframe Imported


full_df
Filter

Pressure9am	Pressure3pm	Cloud9am	Cloud3pm	Temp9am	Temp3pm	RainToday	RainTomorrow
1007.7	1007.1	8	NA	16.9	21.8	No	No
1010.6	1007.8	NA	NA	17.2	24.3	No	No
1007.6	1008.7	NA	2	21.0	23.2	No	No
1017.6	1012.8	NA	NA	18.1	26.5	No	No
1010.8	1006.0	7	8	17.8	29.7	No	No
1009.2	1005.4	NA	NA	20.6	28.9	No	No
1009.6	1008.2	1	NA	18.1	24.6	No	No
1013.4	1010.1	NA	NA	16.3	25.5	No	No
1008.9	1003.6	NA	NA	18.3	30.2	No	Yes
1007.0	1005.7	NA	NA	20.1	28.2	Yes	No
1011.8	1008.7	NA	NA	20.4	28.8	No	Yes
1010.5	1004.2	8	8	15.9	17.0	Yes	Yes
994.3	993.0	8	8	17.4	15.8	Yes	Yes
1001.2	1001.8	NA	7	15.8	19.8	Yes	No
1009.7	1008.7	NA	NA	15.9	23.5	No	NA
1013.4	1010.3	0	NA	17.3	26.2	NA	No
1012.2	1010.4	8	1	17.2	18.1	No	Yes
1005.8	1002.2	8	1	18.0	21.5	Yes	Yes
1009.4	1009.7	NA	2	15.5	21.0	Yes	No

Fig 1.4.2 Initial Dataframe Imported Pt2

As we can see it has a lot of null values and potential outliers which will need processing later on, so we peek into the numerical analysis of the data frame we are working with.

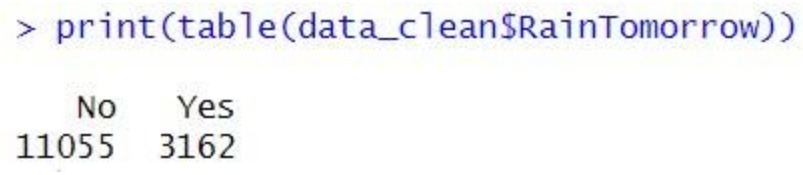


```
> # first look at the data set using summary() to understand what type of data we are working with
> summary(sampled_df)
     Date             Location            MinTemp         MaxTemp         Rainfall         Evaporation    
 Length:14546       Length:14546       Min.   :-6.80   Min.   :-3.10   Min.   :  0.000   Min.   : 0.000  
 Class :character   Class :character   1st Qu.: 7.70   1st Qu.:18.00   1st Qu.:  0.000   1st Qu.: 2.600  
 Mode  :character   Mode  :character   Median :12.00   Median :22.70   Median :  0.000   Median : 4.800  
                                       Mean   :12.25   Mean   :23.26   Mean   :  2.341   Mean   : 5.454  
                                       3rd Qu.:16.90   3rd Qu.:28.20   3rd Qu.:  0.600   3rd Qu.: 7.400  
                                       Max.   :29.40   Max.   :46.80   Max.   :371.000   Max.   :58.500  
                                       NA's   :149     NA's   :125     NA's   :341       NA's   :6333    
    Sunshine       WindGustDir        WindGustSpeed     WindDir9am         WindDir3pm         WindSpeed9am  
 Min.   : 0.000   Length:14546       Min.   :  9.00   Length:14546       Length:14546       Min.   : 0.00  
 1st Qu.: 4.900   Class :character   1st Qu.: 31.00   Class :character   Class :character   1st Qu.: 7.00  
 Median : 8.500   Mode  :character   Median : 39.00   Mode  :character   Mode  :character   Median :13.00  
 Mean   : 7.668                      Mean   : 40.04                                         Mean   :14.04  
 3rd Qu.:10.700                      3rd Qu.: 48.00                                         3rd Qu.:19.00  
 Max.   :14.500                      Max.   :122.00                                         Max.   :65.00  
 NA's   :7005                        NA's   :1057                                           NA's   :177    
  WindSpeed3pm    Humidity9am      Humidity3pm      Pressure9am      Pressure3pm        Cloud9am    
 Min.   : 0.00   Min.   :  2.00   Min.   :  1.00   Min.   : 982.2   Min.   : 981.4   Min.   :0.000  
 1st Qu.:13.00   1st Qu.: 57.00   1st Qu.: 36.00   1st Qu.:1013.0   1st Qu.:1010.5   1st Qu.:1.000  
 Median :19.00   Median : 70.00   Median : 52.00   Median :1017.6   Median :1015.1   Median :5.000  
 Mean   :18.55   Mean   : 68.55   Mean   : 51.45   Mean   :1017.6   Mean   :1015.2   Mean   :4.433  
 3rd Qu.:24.00   3rd Qu.: 83.00   3rd Qu.: 66.00   3rd Qu.:1022.4   3rd Qu.:1020.0   3rd Qu.:7.000  
 Max.   :76.00   Max.   :100.00   Max.   :100.00   Max.   :1040.4   Max.   :1038.9   Max.   :8.000  
 NA's   :324     NA's   :275      NA's   :466      NA's   :1522     NA's   :1528     NA's   :5589   
    Cloud3pm        Temp9am          Temp3pm         RainToday          RainTomorrow      
 Min.   :0.000   Min.   :-4.80   Min.   :-3.90   Length:14546       Length:14546      
 1st Qu.:2.000   1st Qu.:12.40   1st Qu.:16.70   Class :character   Class :character  
 Median :5.000   Median :16.80   Median :21.20   Mode  :character   Mode  :character  
 Mean   :4.498   Mean   :17.06   Mean   :21.75                                        
 3rd Qu.:7.000   3rd Qu.:21.60   3rd Qu.:26.40                                        
 Max.   :8.000   Max.   :37.50   Max.   :45.90                                        
 NA's   :5943    NA's   :185     NA's   :388                                          
> |
```

From here we can infer that we have a mix of categorical and numerical variables. It is also noted that there are several null values in some columns and different scales of data we are dealing with.

1.5 Exploratory Data Analysis

1.5.1 Examining Target Variable

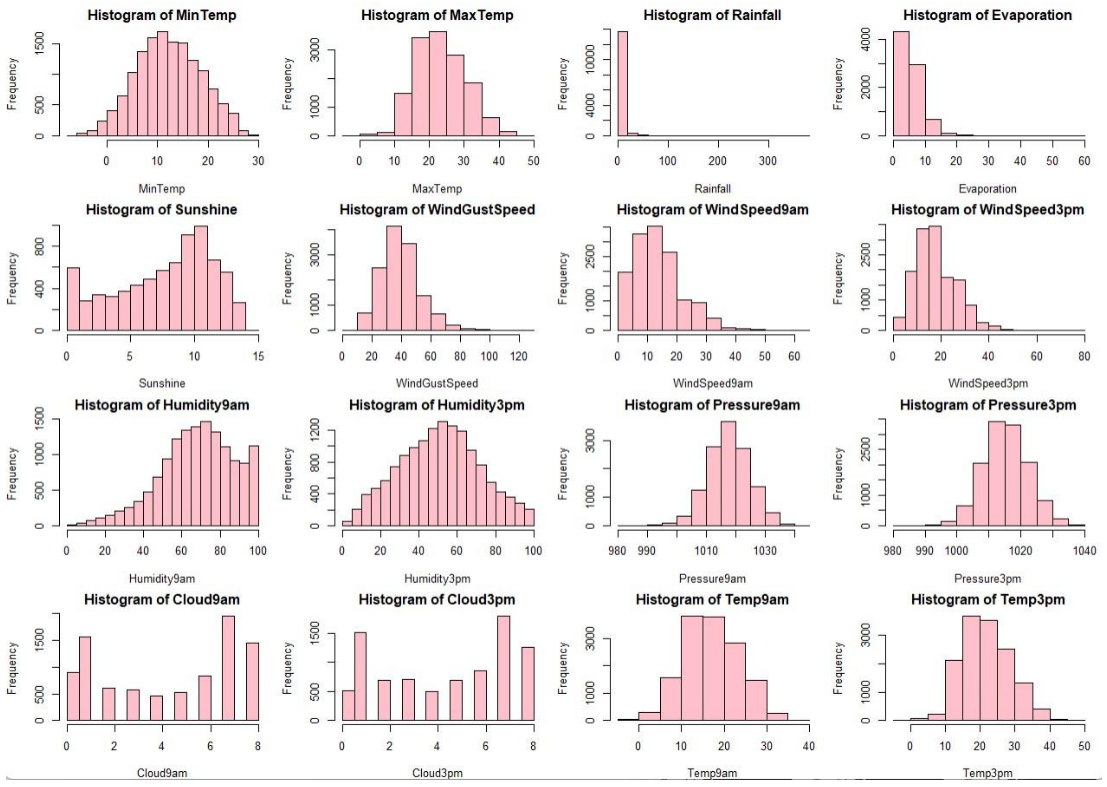


```
> print(table(data_clean$RainTomorrow))

   No   Yes 
11055  3162 
```

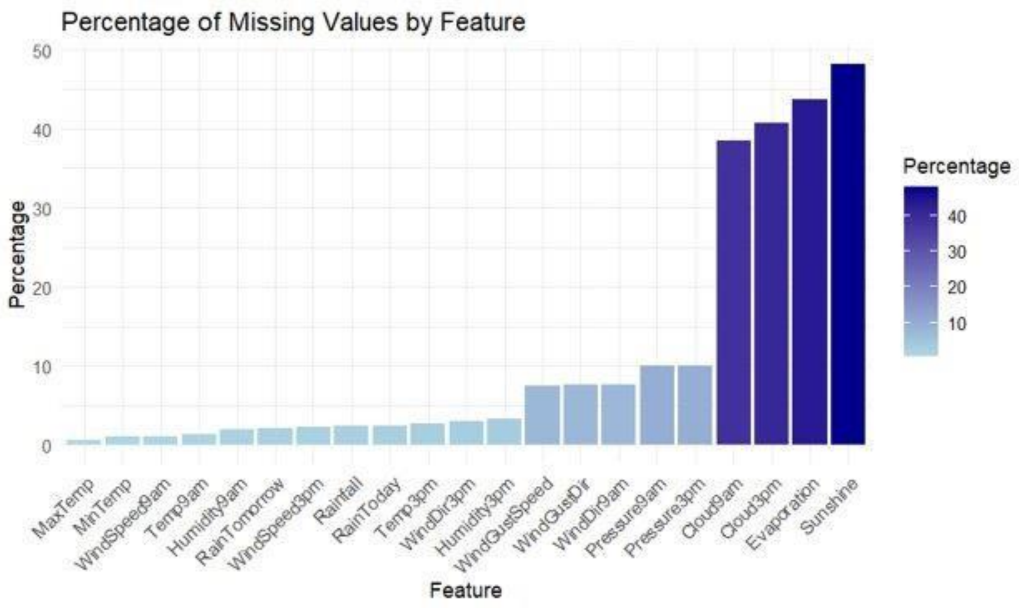
This is the class imbalance we are going to be wary of during our modelling with more instances of no rain compared to rain so our model might be biased towards predicting no rain better.

1.5.2 Examining the Distribution of Data



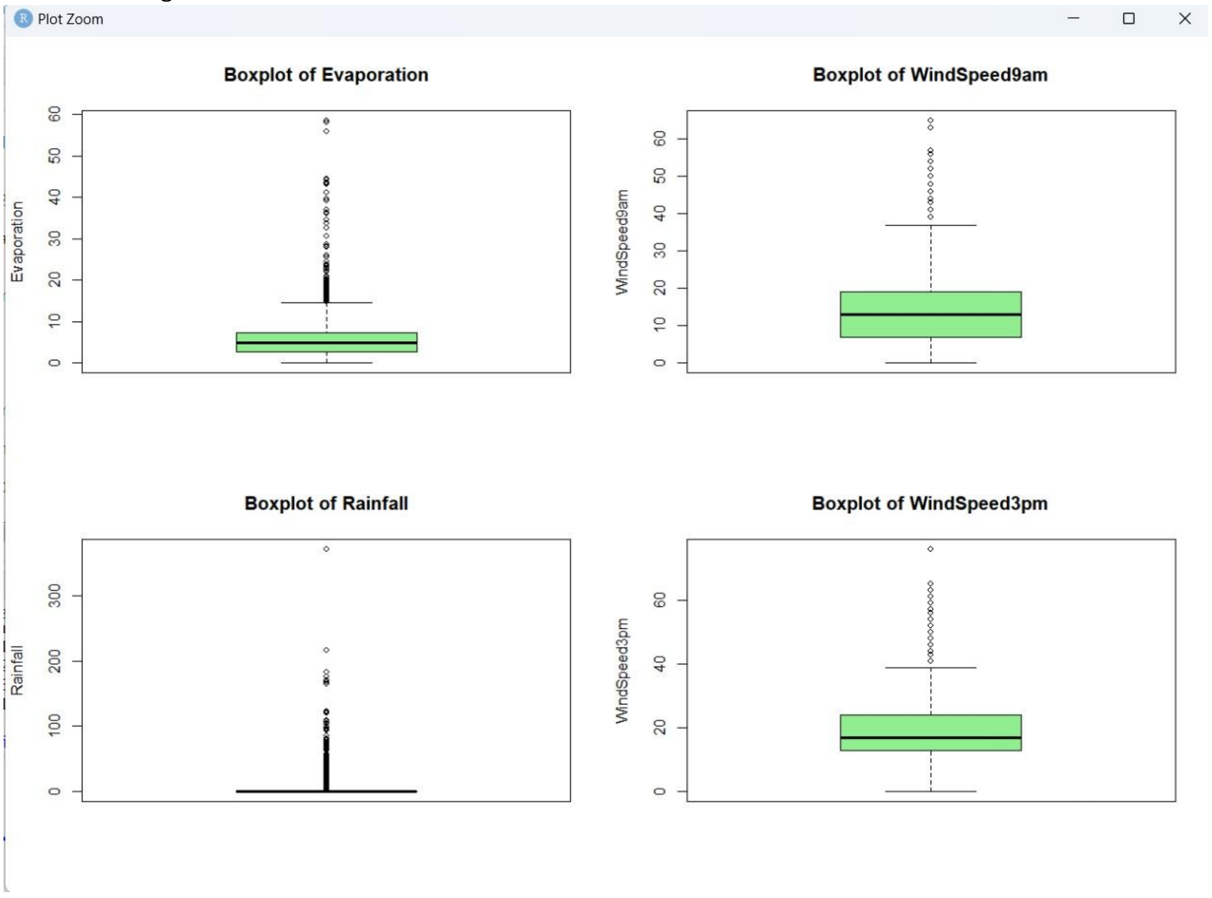
The above distributions indicate that the temperatures are the normal distribution of temperatures suggesting no need for transformation, but outliers will be checked.
As for Rainfall, given the huge right skew, we will examine if this column provides any use when modelling and will be dropped.
Evaporation and Sunshine were found to have a lot of null values which explains the skewed values (which will be explored later).
The distributions for Wind Speed are fairly regular but zero wind speed values might be placeholders for missing data, but we assumed in our analysis that it was not a windy day and we will be checking them for outliers as well.

1.5.3 Plotting the Null Values



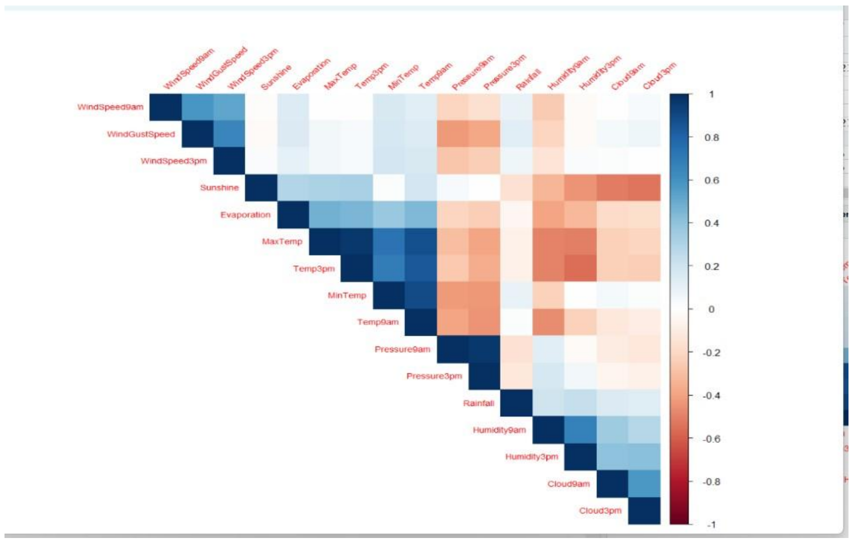
We have plotted the percentage of null values and have observed that Sunshine, Evaporation and Cloud 9am and 3pm have high per cent of null values so we have decided to make separate models by keeping the columns and dropping the columns to see the effect on accuracy on the model.

1.5.4 Detecting Outliers



Baked on the distribution and the rate of outliers, the columns Evaporation, Windspeed9am, Rainfall and WindSpeed3pm show plenty of values lying outside the outer fence so we can classify them as outliers and we will deal with them during our feature engineering phase.

1.5.5 Correlation Matrix



After cleaning the data, we have plotted the correlation to see which numerical features are correlated namely (Rainfall& RainToday, MinMax Temp with Temperatures during different time periods) as a lot of models don't work well with multi-collinearity, we will select the features based on this when we are doing the modelling phase.

Section 2: Data preparation and cleaning

2.1 Data Cleaning

2.1.1 Imputing null values

Since we had null values in the target variable (RainTomorrow), dropped those rows as we can't train the model with those. For the numerical columns we have replaced them with the median of the column since it is more robust to outliers in the data and for the categorical data, the mode imputation method was chosen.

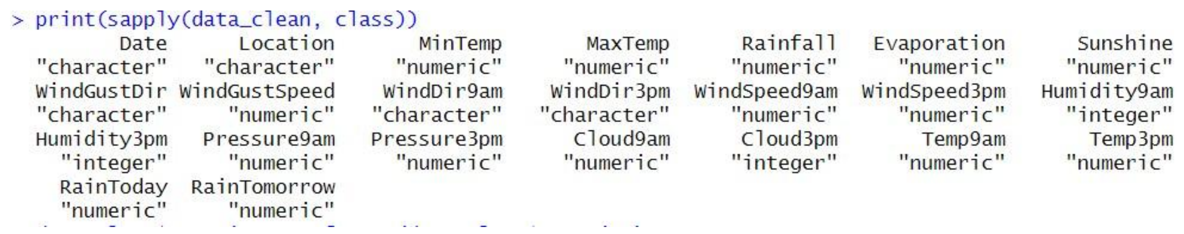
2.1.2 Outlier Removal

For removing the outliers, we identified them using the box plot, if they were below or above 3 times the Inter Quartile Range, they were labelled as outliers and were replaced with the maximum value of the column as removing them would lead to loss of data.

2.1.3 Model selection

For our common and initial model, we have chosen to go with logistic regression as it is well-suited for binary classification. Additionally, it fits our aim of giving us the feature importance as it has good interpretability and will us the reasons and the factors behind the decision it has made. Since our dataset is large, it will be easier for us to compute and compare multiple models at low cost. The assumption of linearity holds for most of our variables so we will be doing the feature engineering based on the model.

2.2 Feature Engineering

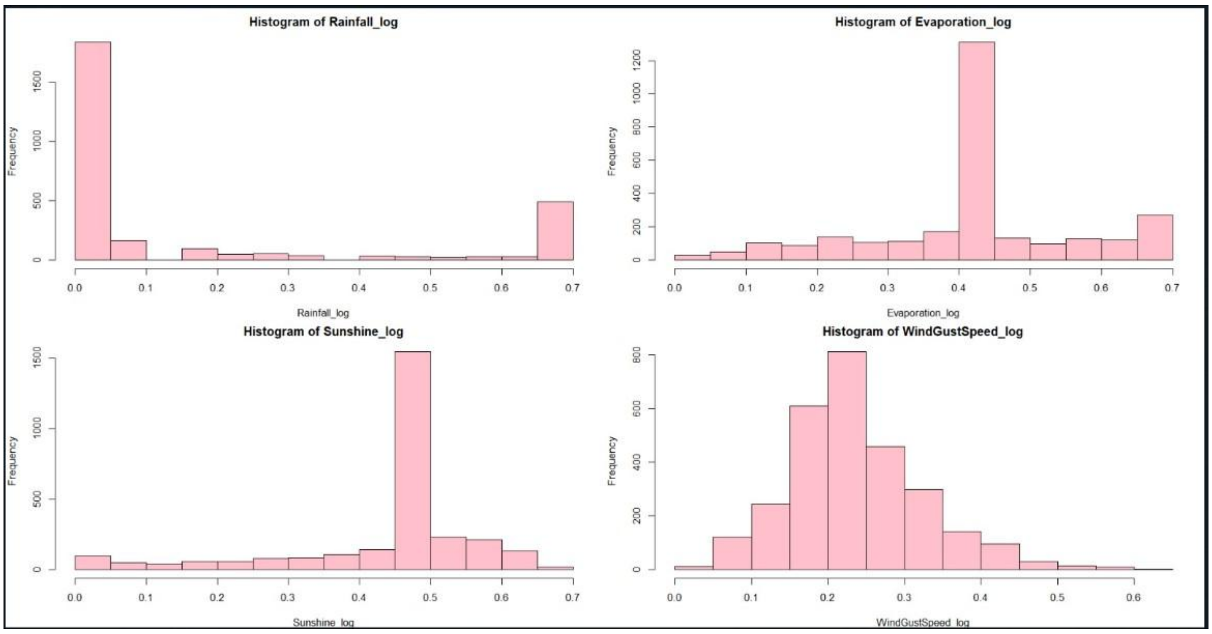


```
> print(sapply(data_clean, class))
        Date      Location       MinTemp       MaxTemp      Rainfall   Evaporation      Sunshine
 "character"   "character"     "numeric"     "numeric"     "numeric"     "numeric"     "numeric"
 WindGustDir WindGustSpeed    WindDir9am    WindDir3pm  WindSpeed9am  WindSpeed3pm   Humidity9am
 "character"     "numeric"   "character"   "character"     "numeric"     "numeric"     "integer"
 Humidity3pm   Pressure9am   Pressure3pm      Cloud9am      Cloud3pm       Temp9am       Temp3pm
   "integer"     "numeric"     "numeric"     "numeric"     "integer"     "numeric"     "numeric"
   RainToday  RainTomorrow
   "numeric"     "numeric"
```

Converted “Date” column from character to datetime format and extracted day, month, and year features. Optimized the dataset by dropping the original column. Following that, the categorical features (Yes/No) were then encoded to 0s and 1s. Min-max scaling was used to scale all the values of the data frame between 0 and 1 in order to put all the numerical values on the same scale.

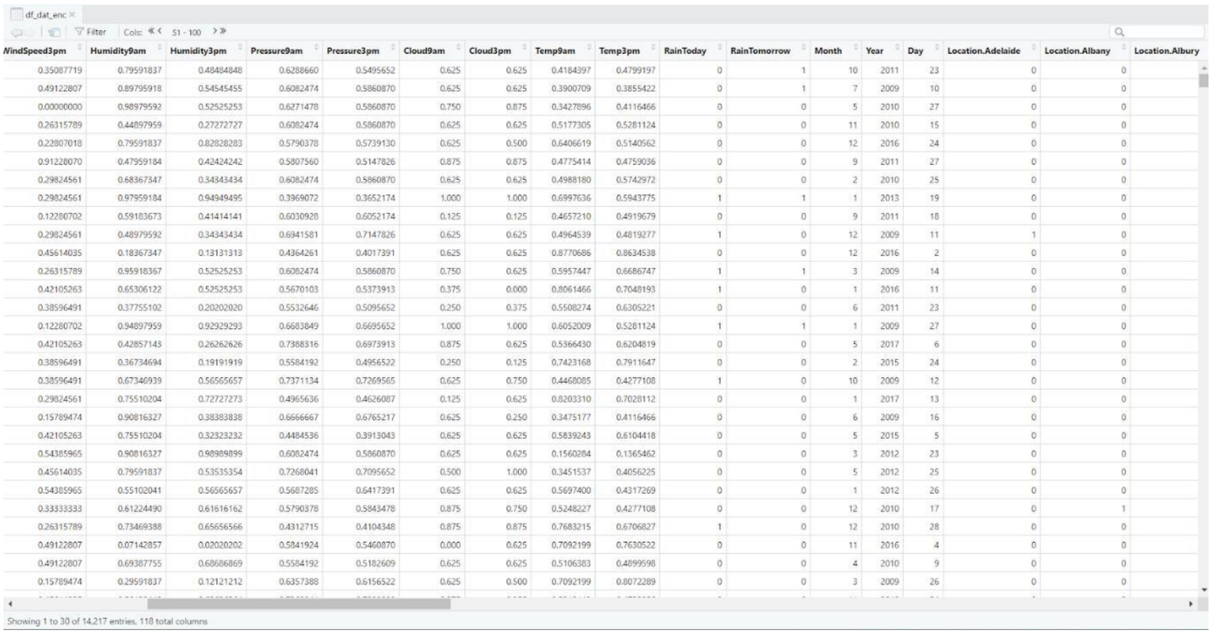
The directions and locations were encoded by using one-hot encoding to accurately capture potential time specific patterns in wind direction and the locations. Again, this was done for the main version of the dataset and we have created many versions of our main dataset which allowed us to encode the features differently and try out different models in different versions of our dataset.

Log transformation was also done on skewed columns to make it a normal distribution for statistical analysis.



For instance, date is encoded as a sine and cosine function to capture the phase and amplitude in one of the datasets, whereas location is alternatively label encoded to reduce the dimensionality and increase computation speed.

In order to fix the class imbalance, we have used the SMOTE method from the package ROSE which produces synthetic samples only for the train data in order to, prevent data leakage and provide a more accurate evaluation of the model's performance. All the 3 methods, (class balancing, undersampling and oversampling) will be tested on the models. Since over-sampling can add noise/overfitting, and undersampling can cause important data to be lost or underfitted, we will test out the model on all the datasets.


df_dat_enc

Filter Cols: « ‹ 51 - 100 › »

WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	Cloud9am	Cloud3pm	Temp9am	Temp3pm	RainToday	RainTomorrow	Month	Year	Day	Location.Adelaide	Location.Albany	Location.Albury
0.35087719	0.79591837	0.48484848	0.6288660	0.5495652	0.625	0.625	0.4184397	0.4799197	0	1	10	2011	23	0	0	
0.49122807	0.89795918	0.54545455	0.6082474	0.5860870	0.625	0.625	0.3900709	0.3855422	0	1	7	2009	10	0	0	
0.00000000	0.98979592	0.52525253	0.6271478	0.5860870	0.750	0.875	0.3427896	0.4116466	0	0	5	2010	27	0	0	
0.26315789	0.44897959	0.27272727	0.6082474	0.5860870	0.625	0.625	0.5177305	0.5281124	0	0	11	2010	15	0	0	
0.22807018	0.79591837	0.82828283	0.5790378	0.5739130	0.625	0.500	0.6406619	0.5140562	0	0	12	2016	24	0	0	
0.91228070	0.47959184	0.42424242	0.5807560	0.5147826	0.875	0.875	0.4775414	0.4759036	0	0	9	2011	27	0	0	
0.29824561	0.68367347	0.34343434	0.6082474	0.5860870	0.625	0.625	0.4988180	0.5742972	0	0	2	2010	25	0	0	
0.29824561	0.97959184	0.94949495	0.3969072	0.3652174	1.000	1.000	0.6997636	0.5943775	1	1	1	2013	19	0	0	
0.12280702	0.59183673	0.41414141	0.6030928	0.6052174	0.125	0.125	0.4657210	0.4919679	0	0	9	2011	18	0	0	
0.29824561	0.48979592	0.34343434	0.6941581	0.7147826	0.625	0.625	0.4964539	0.4819277	1	0	12	2009	11	1	0	
0.45614035	0.18367347	0.13131313	0.4364261	0.4017391	0.625	0.625	0.8770686	0.8634538	0	0	12	2016	2	0	0	
0.26315789	0.95918367	0.52525253	0.6082474	0.5860870	0.750	0.625	0.5957447	0.6686747	1	1	3	2009	14	0	0	
0.42105263	0.65306122	0.52525253	0.5670103	0.5373913	0.375	0.000	0.8061466	0.7048193	1	0	1	2016	11	0	0	
0.38596491	0.37755102	0.20202020	0.5532646	0.5095652	0.250	0.375	0.5508274	0.6305221	0	0	6	2011	23	0	0	
0.12280702	0.94897959	0.92929293	0.6683849	0.6695652	1.000	1.000	0.6052009	0.5281124	1	1	1	2009	27	0	0	
0.42105263	0.42857143	0.26262626	0.7388316	0.6973913	0.875	0.625	0.5366430	0.6204819	0	0	5	2017	6	0	0	
0.38596491	0.36734694	0.19191919	0.5584192	0.4956522	0.250	0.125	0.7423168	0.7911647	0	0	2	2015	24	0	0	
0.38596491	0.67346939	0.56565657	0.7371134	0.7269565	0.625	0.750	0.4468085	0.4277108	1	0	10	2009	12	0	0	
0.29824561	0.75510204	0.72727273	0.4965636	0.4626087	0.125	0.625	0.8203310	0.7028112	0	0	1	2017	13	0	0	
0.15789474	0.90816327	0.38383838	0.6666667	0.6765217	0.625	0.250	0.3475177	0.4116466	0	0	6	2009	16	0	0	
0.42105263	0.75510204	0.32323232	0.4484536	0.3913043	0.625	0.625	0.5839243	0.6104418	0	0	5	2015	5	0	0	
0.54385965	0.90816327	0.98989899	0.6082474	0.5860870	0.625	0.625	0.1560284	0.1365462	0	0	3	2012	23	0	0	
0.45614035	0.79591837	0.53535354	0.7268041	0.7095652	0.500	1.000	0.3451537	0.4056225	0	0	5	2012	25	0	0	
0.54385965	0.55102041	0.56565657	0.5687285	0.6417391	0.625	0.625	0.5697400	0.4317269	0	0	1	2012	26	0	0	
0.33333333	0.61224490	0.61616162	0.5790378	0.5843478	0.875	0.750	0.5248227	0.4277108	0	0	12	2010	17	0	1	
0.26315789	0.73469388	0.65656566	0.4312715	0.4104348	0.875	0.875	0.7683215	0.6706827	1	0	12	2010	28	0	0	
0.49122807	0.07142857	0.02020202	0.5841924	0.5460870	0.000	0.625	0.7092199	0.7630522	0	0	11	2016	4	0	0	
0.49122807	0.69387755	0.68686869	0.5584192	0.5182609	0.625	0.625	0.5106383	0.4899598	0	0	4	2010	9	0	0	
0.15789474	0.29591837	0.12121212	0.6357388	0.6156522	0.625	0.500	0.7092199	0.8072289	0	0	3	2009	26	0	0	

Showing 1 to 30 of 14,217 entries, 118 total columns

Feature Selection:

The fields that will be included in the data for the project have been selected based on their relevance to RainTomorrow and their potential impact on the predictive model. The included fields that are directly related to the target variable, such as rainfall, temperature, humidity, and wind speed, will be included.

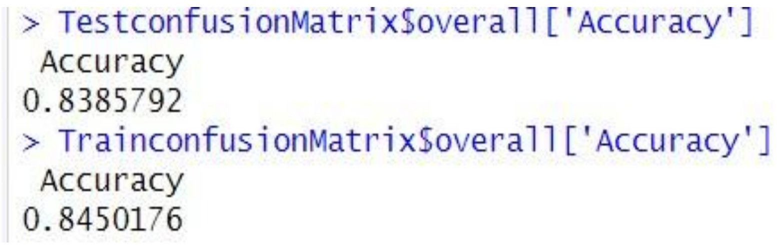
The fields that will be excluded from the project will be determined based on their lack of relevance, multicollinearity, or presence of nulls. Some of the datasets prepared to be modelled do not include date, location, rainfall, or evaporation.

Section 3: Modelling - Machine Learning Algorithms Implementation

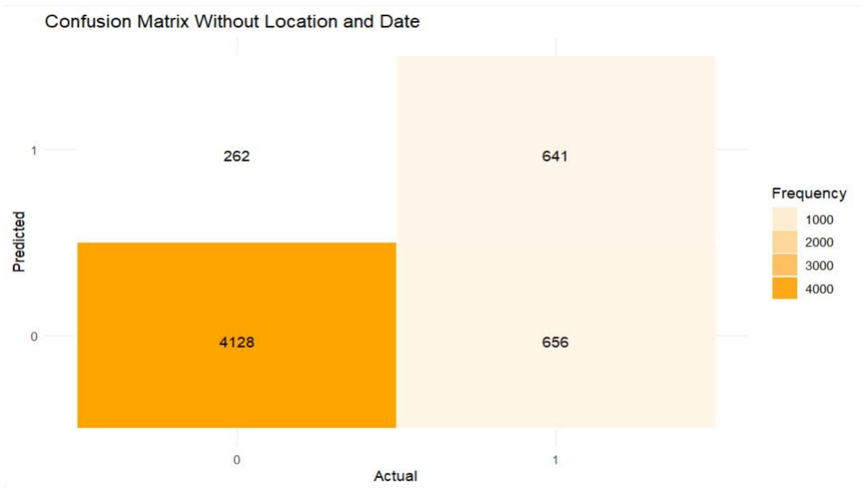
1. Logistic Regression Modelling

To reiterate from the report, the logistic regression model was chosen because it is well-suited for binary classification. Additionally, it fits our aim of giving us the feature importance as it has good interpretability and will us the coefficients so that we can decode the factors behind the rain prediction.

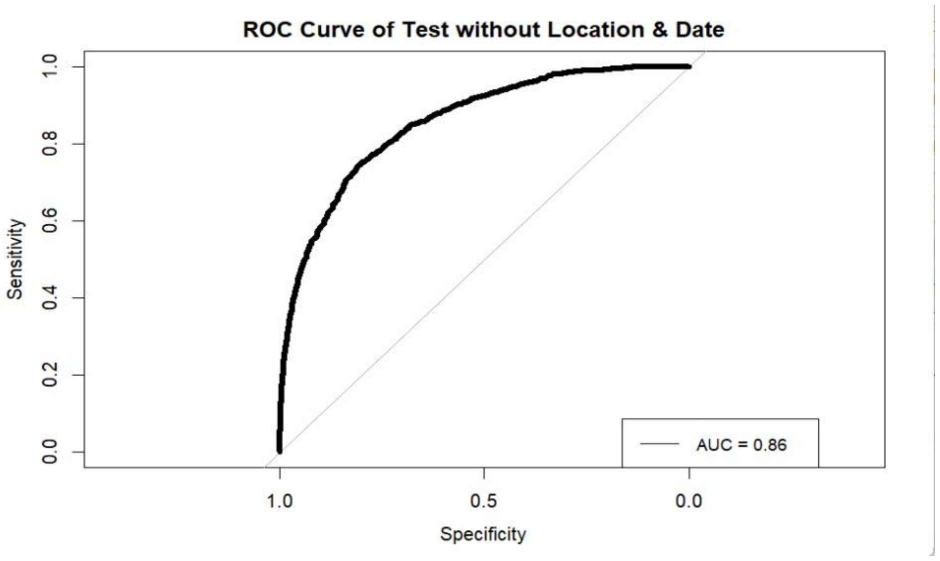
I first used the dataset without dates and location and have further added features and compared the performance. I have also created a data frame to record the performance metrics of the models and different as we go on with the modelling.
For all of the variations of the dataset, I have used a 60,40 split to incorporate maximum instances of the '0' variable so that the test data is not biased.
After the training using the default parameters on the training data and noting down the performance metrics by plotting the confusion matrix on the test set, I ran the evaluations on the training set in order to confirm the absence of overfitting on the training. Since the value was pretty close to the test accuracy, I concluded the model was able to generalize well for new data points.



```
> TestconfusionMatrix$overall['Accuracy']
 Accuracy
0.8385792
> TrainconfusionMatrix$overall['Accuracy']
 Accuracy
0.8450176
```

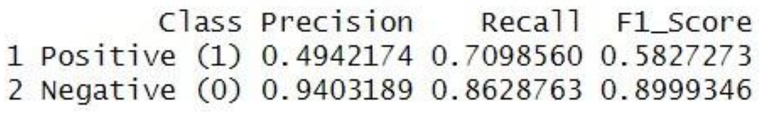


Following this, the ROC curve was plotted in order to find how the classes would be distinguished from each other.



This AUC of 0.85 indicates that this model can classify between the positive and negative classes pretty well.

The overall performance of the model is pretty good at first glance with 83% accuracy and 94% recall, but since we used an imbalanced dataset, we have to examine the results closely to see the full picture. So I have decided to plot the performance of the models by class in order to see the weakness of the model.

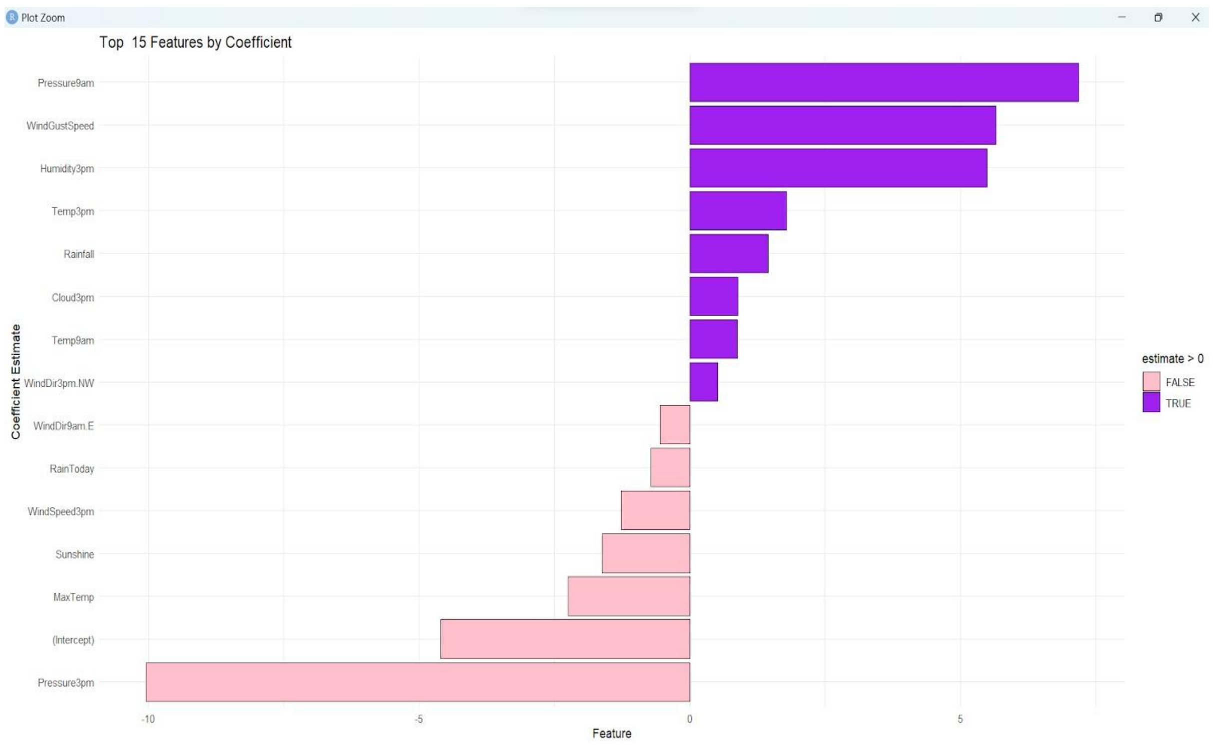


```
           Class Precision    Recall  F1_Score
1 Positive (1) 0.4942174 0.7098560 0.5827273
2 Negative (0) 0.9403189 0.8628763 0.8999346
```

As we can see here, the model does really well for predicting when there is no rain but the real use-case of the project would be in the successful prediction of the rain it is also clear better recall and it would be the ideal model as we want to maximize True Positives as being more false alerts would be fine rather than being totally unprepared for the rain. So will be doing similar metrics comparison for the following datasets with varying features by storing all the values in a dataframe to be compared in the end.

Following this dataset, I have added 1 hot encoded location and the days, months, and years extracted from the date to the data frame further increasing the dimensionality to see if it helps the model generalize better. However, the metrics showed me that these features only help increase the values of the negative which is not our goal with this model.

So, with these 2 basic models done, we can finally see which are the factors that contribute most to our decision-making, so I have plotted the estimate of the coefficients of the linear model below.



From this, we can infer that the most contributing features for the prediction are Pressure, Windspeed, Humidity and Temperature in that order. These are readings of the day before that provide the most valuable insight.

However, in our initial data preparation, we had seen that the value sunshine had a lot of null values, but when I dropped the column the accuracy dropped, so there is no bias with the imputation.

Following this, the hyperparameter tuning was done with, mixture and penalty, which were opted for finetuning. A grid was set to find the best combination for fine tuning, and this was set up for 10-fold cross validation. The model was trained with these parameters and found to have better performance on the negative class again which in turn increased the overall performance but not the bias for the predictions.

The 10-fold cross-validation only increases the performance marginally so we can say the training data is well-fitted and good at generalizing.

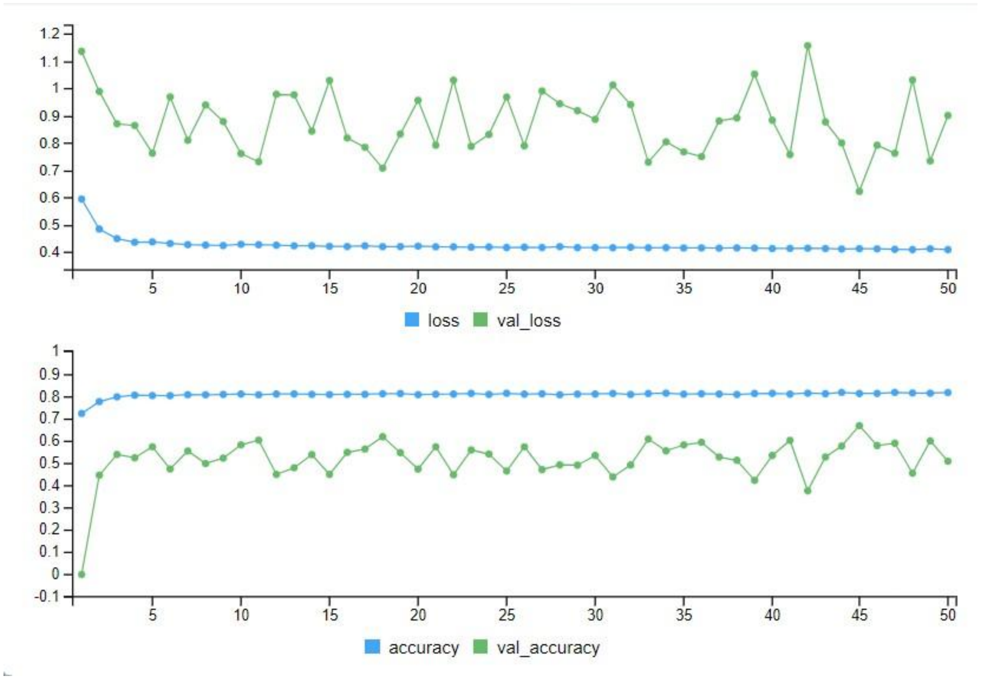
After the SMOTE sampling was done for the dataset, I ran the logistic model again to see how the class balancing would help the model, I did end up having the most precision out of all the models and better-balanced precision between the positive and negative class, but the recall was still behind.

2. Artificial Neural Network Modelling

For modelling with the ANN, I could not use the same dataset with the large number of dimensions as it took an unrealistic amount of computation time, so I had to do label encoding to the location and the wind direction columns. The months and days should be a cyclic continuous feature so that the model can understand the jump from 12 to 1 month and 30 to 1 for days. So, I transformed them using sin and cosine to get the amplitude and phase of the dates.

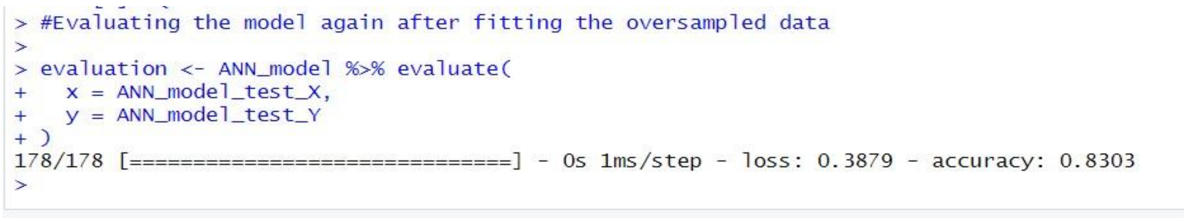
The Resampling technique was applied now using SMOTE to fix the class imbalance and overall practical use of the dataset. I have used SMOTE on a SMOTE balanced dataset for both oversampling of the minority class and undersampling of the majority class to create a balanced dataset. An oversampled dataset in which we have oversampled the minority dataset by adding 2000 generated values for the minority class. On the under sample, we have undersampled the majority dataset by removing values for the majority class. Because over-sampling can add noise/overfitting, and undersampling can cause important data to be lost or underfitting, I will test out the model on all the datasets.

For building the ANN, I chose the sequential model as there is a clear flow of data throughout the neurons. Layers of 5 dense layers were set with the neurons decreasing from 32 neurons with ReLU as an activation function to the sigmoid function in the output layer to squish the values to binary. A fairly low learning was given to the Adam optimiser to help the weights converge and prevent missing the minima.



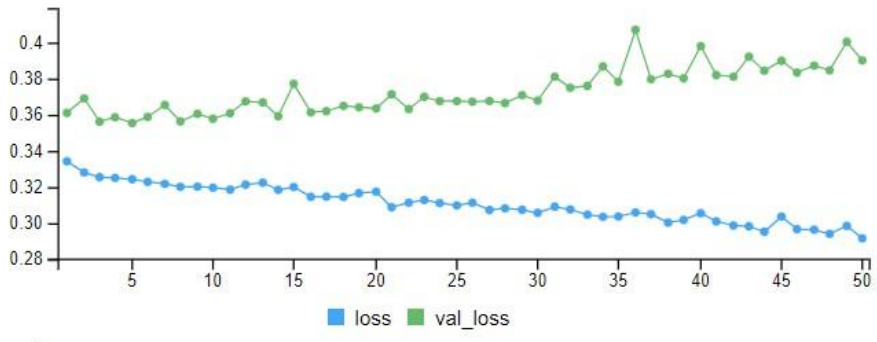
After fitting the balanced SMOTE data on the ANN, the history reveals a consistent improvement in the neural network's performance over 50 epochs. Starting with an accuracy of 72.1%, the model demonstrated a gradual increase, reaching a peak validation accuracy of 66.67%. Despite some fluctuations, both training and validation losses decreased, indicating effective learning from the data. The model's ability to generalize to unseen data is evident as it can balance under/overfitting, although there might be room for further improvement. The training time per epoch was reasonable at 4 seconds, shows good efficiency proving our encoding techniques earlier.

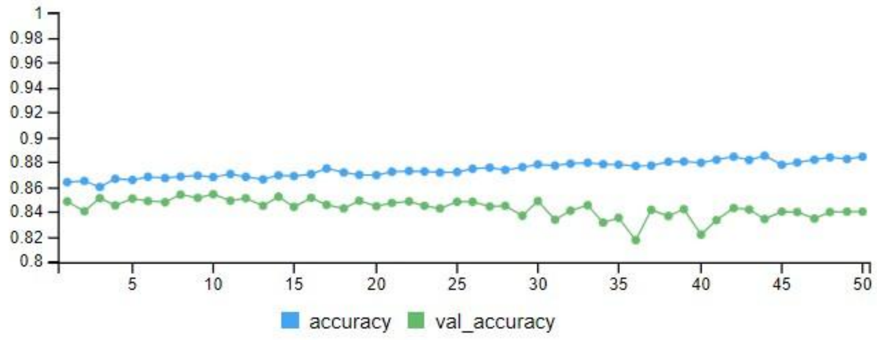
After evaluating the test set, observed that it does give 83.2% accuracy. After this, I did the evaluation on the over sampled data which gave me a loss in accuracy which indicated that it was adding noise to the data. When fitted on the test data, the training accuracy indicated improvements, but the test accuracy remains the same so, undersampling wasn't helping the model to distinguish the classes.



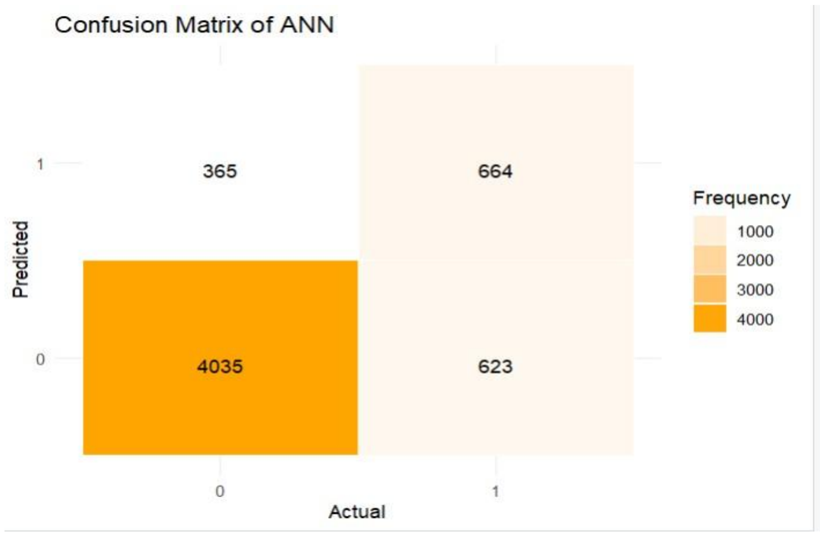
```
> #Evaluating the model again after fitting the oversampled data
>
> evaluation <- ANN_model %>% evaluate(
+   x = ANN_model_test_X,
+   y = ANN_model_test_Y
+ )
178/178 [==============================] - 0s 1ms/step - loss: 0.3879 - accuracy: 0.8303
>
```

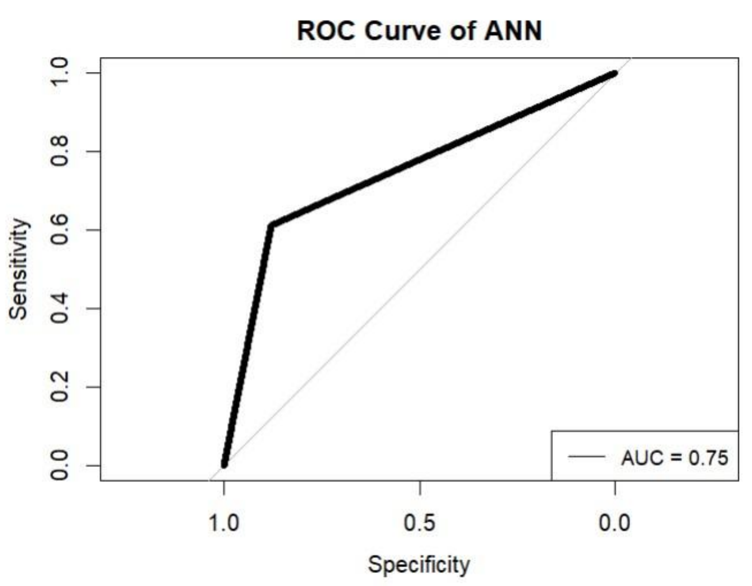
So, after this, it was time to consider the ANN without the sampling techniques and to find out the performance.



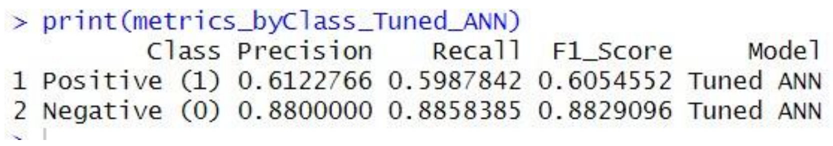


However as observed, while the validation accuracy was much higher than other models, the loss was starting to diverge which meant that it was starting to overfit the data, and the test accuracy was similar to the oversampling techniques. The confusion matrix was plotted, and all the metrics were added to the result dataframe for comparison. Although performance metrics are plotted in the R script, *I have not included all of the plots for all the models here, only the key findings.*





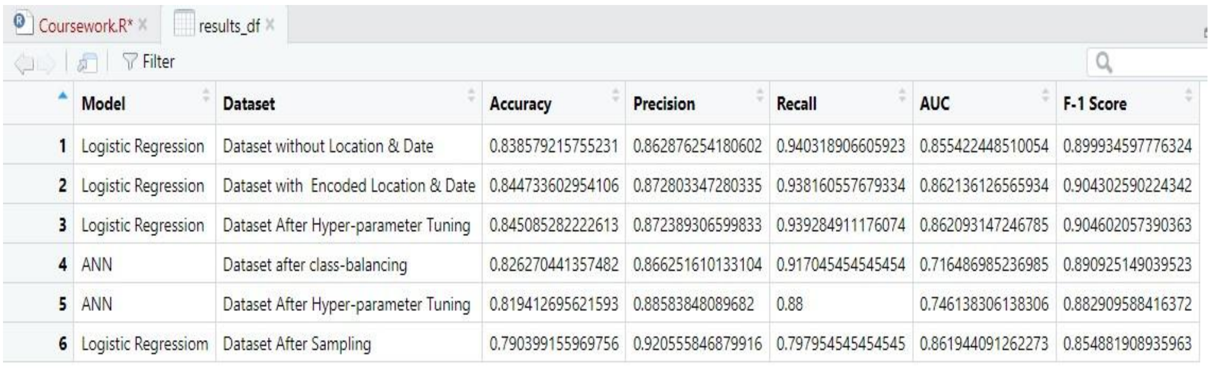
So based on all the above evidence, it was chosen that the best model for ANN would be the dataset balanced with sampling techniques. Hyperparameter tuning was then done on this model fitting this dataset.



```
> print(metrics_byClass_Tuned_ANN)
         Class Precision    Recall  F1_Score     Model
1 Positive (1) 0.6122766 0.5987842 0.6054552 Tuned ANN
2 Negative (0) 0.8800000 0.8858385 0.8829096 Tuned ANN
```

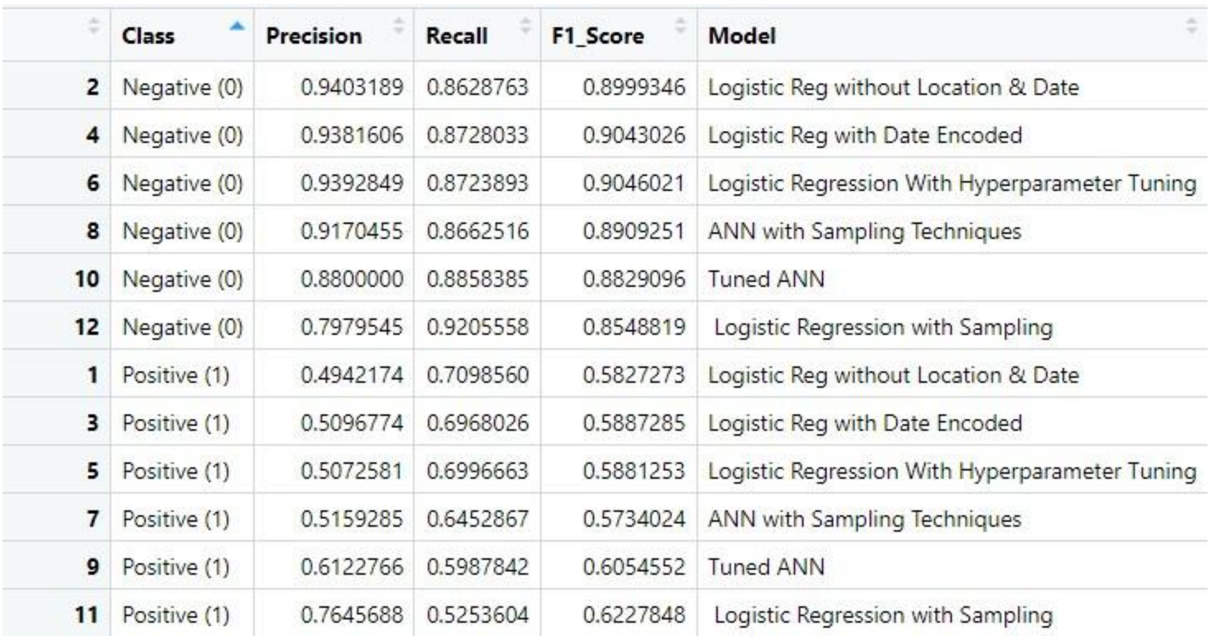
After training for more epochs, it was clear that this model performs much better for the positive class, this might be due to the resampling techniques we have done. The AUC value was obtained to be 0.75 which is decent.

Comparing both the models:



	Model	Dataset	Accuracy	Precision	Recall	AUC	F-1 Score
1	Logistic Regression	Dataset without Location & Date	0.838579215755231	0.862876254180602	0.940318906605923	0.855422448510054	0.899934597776324
2	Logistic Regression	Dataset with Encoded Location & Date	0.844733602954106	0.872803347280335	0.938160557679334	0.862136126565934	0.904302590224342
3	Logistic Regression	Dataset After Hyper-parameter Tuning	0.845085282222613	0.872389306599833	0.939284911176074	0.862093147246785	0.904602057390363
4	ANN	Dataset after class-balancing	0.826270441357482	0.866251610133104	0.917045454545454	0.716486985236985	0.890925149039523
5	ANN	Dataset After Hyper-parameter Tuning	0.819412695621593	0.88583848089682	0.88	0.746138306138306	0.882909588416372
6	Logistic Regressiom	Dataset After Sampling	0.790399155969756	0.920555846879916	0.797954545454545	0.861944091262273	0.854881908935963

Comparing the data frames with all the metrics, it is evident that the *Logistic regression model with the Hyperparameter tuning is the model that gives the best overall result* but to examine the class imbalance, we will dive into the metrics by class.



	Class	Precision	Recall	F1_Score	Model
2	Negative (0)	0.9403189	0.8628763	0.8999346	Logistic Reg without Location & Date
4	Negative (0)	0.9381606	0.8728033	0.9043026	Logistic Reg with Date Encoded
6	Negative (0)	0.9392849	0.8723893	0.9046021	Logistic Regression With Hyperparameter Tuning
8	Negative (0)	0.9170455	0.8662516	0.8909251	ANN with Sampling Techniques
10	Negative (0)	0.8800000	0.8858385	0.8829096	Tuned ANN
12	Negative (0)	0.7979545	0.9205558	0.8548819	Logistic Regression with Sampling
1	Positive (1)	0.4942174	0.7098560	0.5827273	Logistic Reg without Location & Date
3	Positive (1)	0.5096774	0.6968026	0.5887285	Logistic Reg with Date Encoded
5	Positive (1)	0.5072581	0.6996663	0.5881253	Logistic Regression With Hyperparameter Tuning
7	Positive (1)	0.5159285	0.6452867	0.5734024	ANN with Sampling Techniques
9	Positive (1)	0.6122766	0.5987842	0.6054552	Tuned ANN
11	Positive (1)	0.7645688	0.5253604	0.6227848	Logistic Regression with Sampling

From the data frame, we can see that the overall performance from the logistic regression model comes from the negative class but since we want to predict the rain, the positive class will be the priority for this case. The Hyperparameter tuning of ANN gives the best F1 score and will be the best overall, but since Recall tumps over the Precision in the data, **The logistic regression model without location and date** would be the best model for us to use out of these models.

3. Gradient Boosting Machine:

Gradient Boosting Machine, a member of the ensemble learning team, enhances accuracy and resilience by teaming up the forecasts of many models. It builds trees one after another, with each correcting the mistakes of the previous group.

Gradient Boosting demonstrates its effectiveness in binary classification through the following approach.:

Initialization:

The initial step involves making a prediction, which can be as simple as calculating the overall mean of the target variable in the training data.

Sequential Tree Building:

At each iteration, a new decision tree is optimized to learn from the current ensemble's prediction errors. This tree, weighted by a learning rate, joins the ensemble to improve overall accuracy while mitigating overfitting.

Combining Predictions:

The final prediction is generated by aggregating the individual predictions of all trees, with each contribution adjusted by the learning rate for optimal blend.

Training Process:

The training loop concludes upon hitting a pre-specified iteration count or when a stopping criterion, such as attaining a desired level of performance on the validation set, is satisfied.

Fitting GBM to the 'WeatherAus.csv' to predict the outcome RainTomorrow:

· Gradient Boosting Machine model was built using the XGBoost library. This model harnessed the "binary: logistic" objective function over 100 boosting rounds, effectively learning from the provided training data X_train (Independent variables in the dataset) and corresponding labels y_train (Outcome variable).

· Then, we make predictions on the test data with the predict function based on the extracted patterns from the trained model.

· Finally, we will obtain a confusion matrix, which provides a summary of the predicted and actual class labels.

· Based on the Confusion Matrix we calculate Accuracy, Precision and Recall. And we plotted the ROC (Receiver Operating Characteristics) curve to assess the performance of a binary classification model at various classification thresholds.

Here are the performance metrics for the Gradient Boosting Machine model:

Accuracy: 0.6815

Precision: 0.8664

Recall: 0.3963

Gradient Boosting Machine with Hyper parameter Tuning:

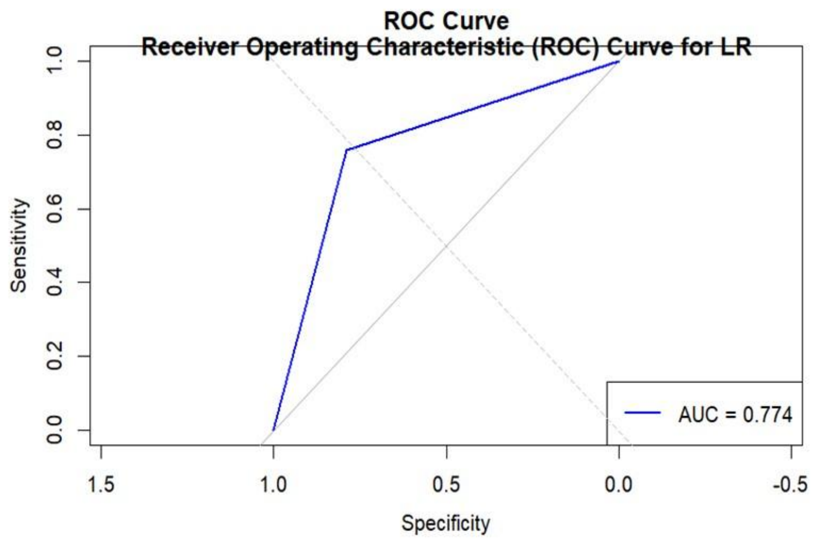
· Incorporated 5-fold cross-validation for comprehensive performance assessment across multiple data segments. Real-time iteration progress updates, enabled by 'verboseIter = TRUE, facilitates model behaviour observation and potential intervention during training.
· The hyper_grid_GBM sets up a search space for tuning the XGBoost GBM model. It fixes crucial settings like learning rate and tree depth, providing various combinations to test and find the best fit for your data.
· The train function is employed to train a GBM model optimized for accuracy, leveraging XGBoost's xgbTree method and a custom hyperparameter grid (hyper_grid_GBM). The resulting model, ready for evaluation and application, is saved as GBM_Model_Tuned.
· Similarly, predictions were made on the test data based on the tuned GBM model and evaluation metrics were calculated.
Performance metrics for Gradient Boosting Machine with Hyper Parameter Tuning:
Accuracy (Tuned): 0.7203 Precision
(Tuned): 0.7761
Recall (Tuned): 0.4245

To understand how data changes affect model performance, we trained the model on several modified datasets. Below you will find the performance metrics for each variation.

Dataset	Accuracy	Precision	Recall	AUC
Removing location field and with 10% over sampling and without hyper parameter tuning.	0.65	0.79	0.37	0.81
Removing location field and with 10% over sampling and with hyper parameter tuning.	0.71	0.74	0.4	0.72
With Location and no over sampling	0.85	0.53	0.73	0.86

Summary of Model:

Gradient Boosting Machine shows promise in handling the dataset, with the potential for improved performance after hyperparameter tuning. Optimizing hyperparameters like learning rate and tree depth through careful selection and fine-tuning is crucial for maximizing model performance. This process enables tailoring the model to the unique characteristics of the dataset, unlocking its full potential for accurate and reliable predictions. Below is the ROC curve with Hyper Parameter Tuning.

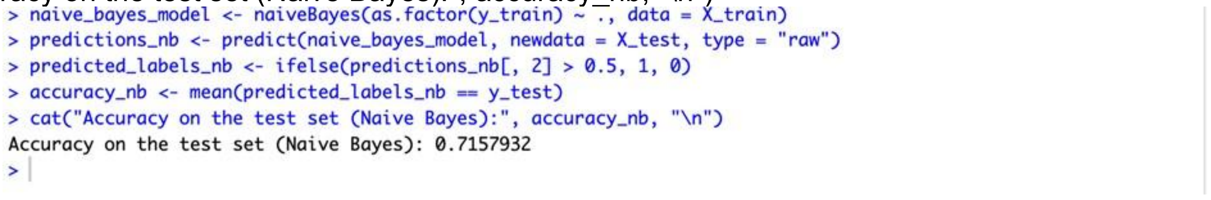


Analysing the Receiver Operating Characteristic (ROC) curve showed that the model was moderately good at distinguishing between positive and negative cases. It got an AUC (Area Under the Curve) score of 0.74, where higher values mean better performance. In this case, an AUC of 0.74 indicates a decent ability to balance the number of correctly identified positives (true positive rate) with the number of incorrectly identified positives (false positive rate).

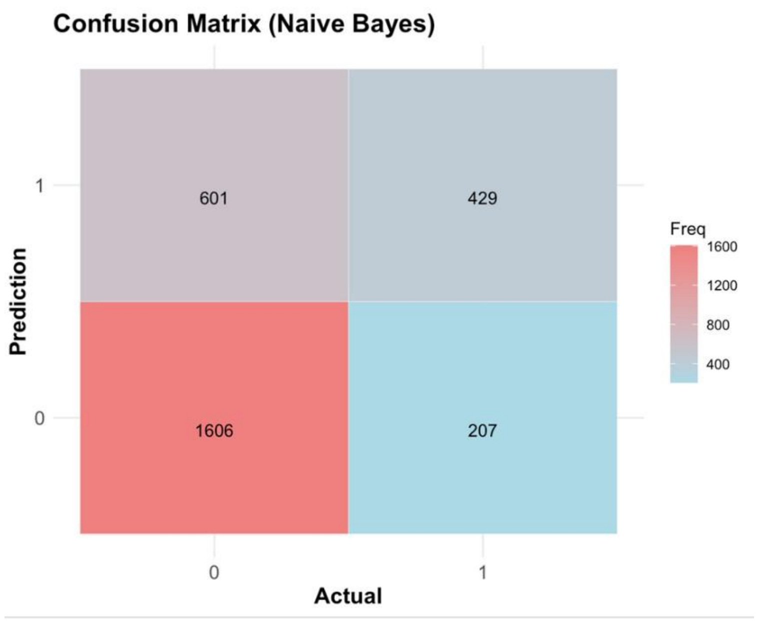
4. Naïve Bayes:

Model is trained using the naive Bayes function using library (1071) with the target variable y_train and features X_train.

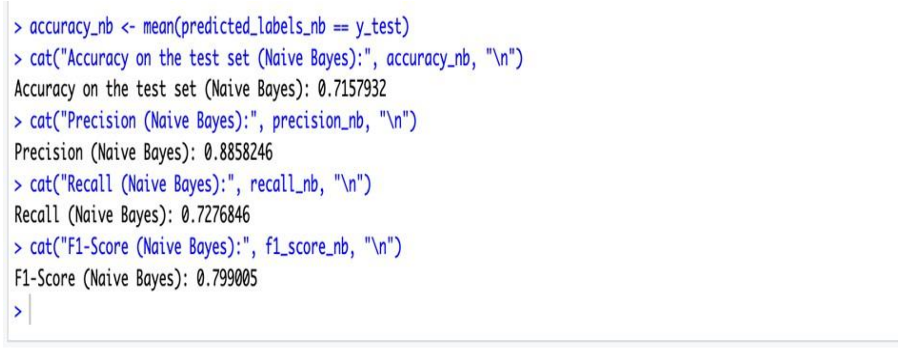
naive_bayes_model <- naiveBayes(as.factor(y_train) ~ ., data = X_train)
Predictions on the Test Set: The trained Naive Bayes model is used to make predictions on the test set (X_test).
Model Evaluation - Finding Accuracy:
Accuracy is calculated by comparing the predicted labels to the actual labels in the test set.
predictions_nb <- predict (naive_bayes_model, newdata = X_test, type = "raw")
predicted_labels_nb <- ifelse (predictions_nb[, 2] > 0.5, 1, 0)
accuracy_nb <- mean(predicted_labels_nb == y_test) cat
("Accuracy on the test set (Naive Bayes):", accuracy_nb, "\n")


```
> naive_bayes_model <- naiveBayes(as.factor(y_train) ~ ., data = X_train)
> predictions_nb <- predict(naive_bayes_model, newdata = X_test, type = "raw")
> predicted_labels_nb <- ifelse(predictions_nb[, 2] > 0.5, 1, 0)
> accuracy_nb <- mean(predicted_labels_nb == y_test)
> cat("Accuracy on the test set (Naive Bayes):", accuracy_nb, "\n")
Accuracy on the test set (Naive Bayes): 0.7157932
> 
```

Confusion Matrix: A confusion matrix is generated to evaluate the performance of the model on the test set. It gives the number of true positive, true negative, false positive, and false negative predictions made by the model. A heatmap is created to visually represent the confusion matrix.

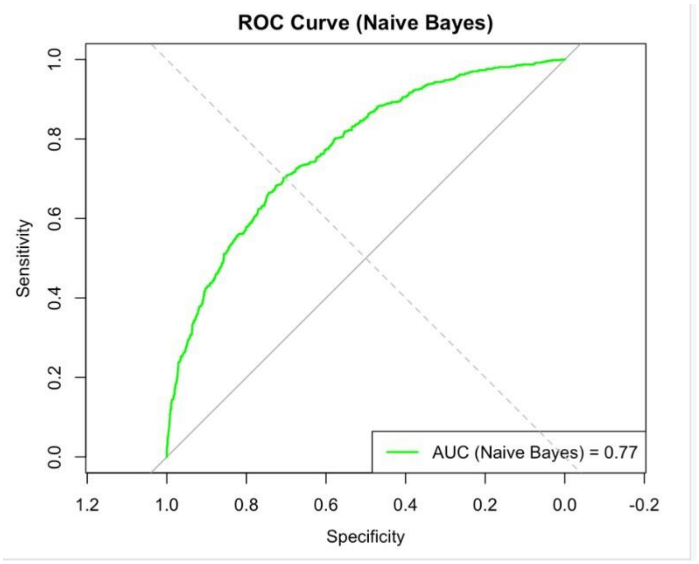


Accuracy, Precision, Recall, and F1-Score Calculation**:** The following Performance Metrices were calculated.



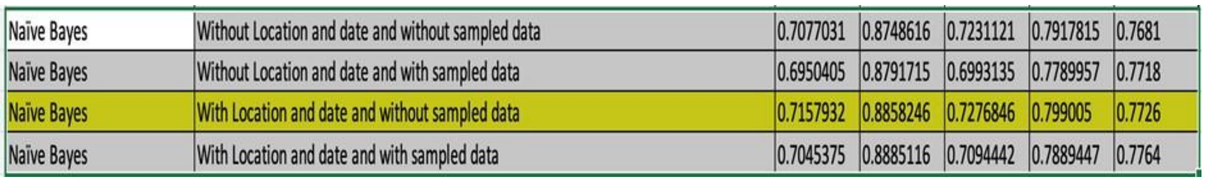
```
> accuracy_nb <- mean(predicted_labels_nb == y_test)
> cat("Accuracy on the test set (Naive Bayes):", accuracy_nb, "\n")
Accuracy on the test set (Naive Bayes): 0.7157932
> cat("Precision (Naive Bayes):", precision_nb, "\n")
Precision (Naive Bayes): 0.8858246
> cat("Recall (Naive Bayes):", recall_nb, "\n")
Recall (Naive Bayes): 0.7276846
> cat("F1-Score (Naive Bayes):", f1_score_nb, "\n")
F1-Score (Naive Bayes): 0.799005
>
```

ROC curve and AUC score:



ROC curve is plotted to compare the true positive rate to false positive rate.
AUC score is used to calculate the area that comes under the ROC curve

This model had the best Accuracy, Recall, and F-1 score among all the Naïve bayes models I created. I created models on two datasets with Date and location and without date and location, with doing oversampling and without oversampling. This model I created was on dataset which included date and time and without oversampling. Dataset with Date and location and which was oversampled had the highest Precision and AUC score.



Naïve Bayes	Without Location and date and without sampled data	0.7077031	0.8748616	0.7231121	0.7917815	0.7681
Naïve Bayes	Without Location and date and with sampled data	0.6950405	0.8791715	0.6993135	0.7789957	0.7718
Naïve Bayes	With Location and date and without sampled data	0.7157932	0.8858246	0.7276846	0.799005	0.7726
Naïve Bayes	With Location and date and with sampled data	0.7045375	0.8885116	0.7094442	0.7889447	0.7764

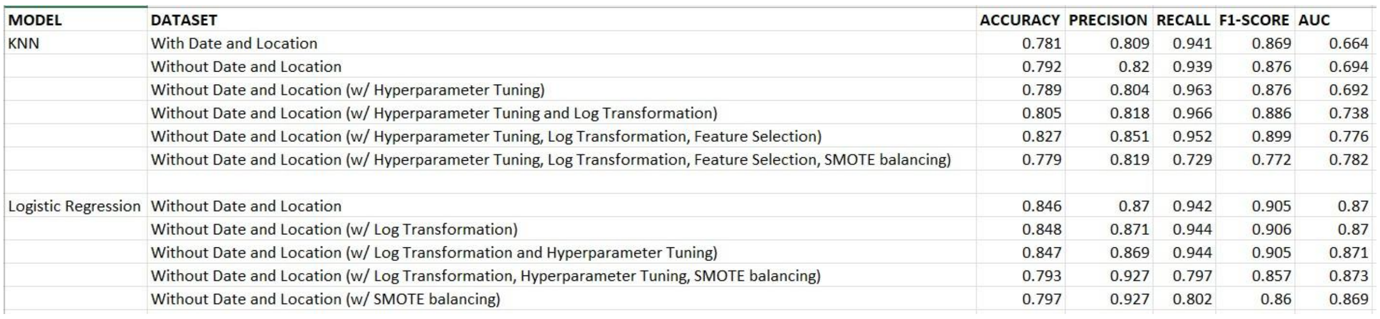
5. K-Nearest Neighbours (KNN):

1. With Date & Location
    - It demonstrates a reasonably good accuracy (78.1%) however accuracy alone may not provide a complete picture.
    - Precision (80.9%) and recall (94.1%) show good identification of positive cases and a good f1-score means a good balance between the two.
    - With a 66.4% AUC, the model shows moderate skill in distinguishing between positive and negative instances. It's not good but its also not the worse.
2. Without Date & Location
    - The metrics display a consistent pattern. It shows a slight improvement overall when compared to the model incorporating date and location.
3. Without Date & Location (w/ Hyperparameter Tuning)
    - There is an improvement in recall (96.3%). Hyperparameter tuning enhances the model's ability to catch positive instances.
    - But there's a decrease in precision (80.4%). Although there was an improvement in recall, it comes at the cost of slight decrease in precision.
4. Without Date & Location (w/ Hyperparameter Tuning & Log Transformation)
    - Log transformation shows a slight improvement overall with notable increase in metrics, indicating a better overall model performance.
5. Without Date & Location (w/ Hyperparameter Tuning, Log Transformation, Feature Selection) • Selecting features contributes to significant improvements in accuracy, precision, recall, f1score, and AUC.
6. Without Date & Location (w/ Hyperparameter Tuning, Log Transformation, Feature Selection, SMOTE Balancing)
    - Theres a drop in accuracy from 82.7% to 77.9%.
    - Theres a significant drop in both recall & precision.
    - Therefore, the model became less effective after applying SMOTE balancing.

Logistic Regression:

1. Without Date & Location
    - Started with high accuracy of 84.6%.
    - String precision (87.9%) and recall (94.2%) and a good f1-score (90.5%) with AUC at 87.0% showing an excellent distinguishing capability.
2. Without Date & Location (w/ Log Transformation)
    - Noticed a consistent improvement in metrics after log transformation.
3. Without Date & Location (w/ Hyperparameter Tuning & Log Transformation)
    - Hyperparameter tuning introduced very minor improvements.
4. Without Date & Location (w/ Hyperparameter Tuning, Log Transformation, SMOTE Balancing)
    - Like KNN, there's a drop in accuracy possibly due to overfitting of the model.
    - Achieved a good precision but the recall dropped affecting f1-score.
5. Without Date & Location (w/ SMOTE Balancing)

- Noticed a further drop in accuracy but maintained a good precision and recall.



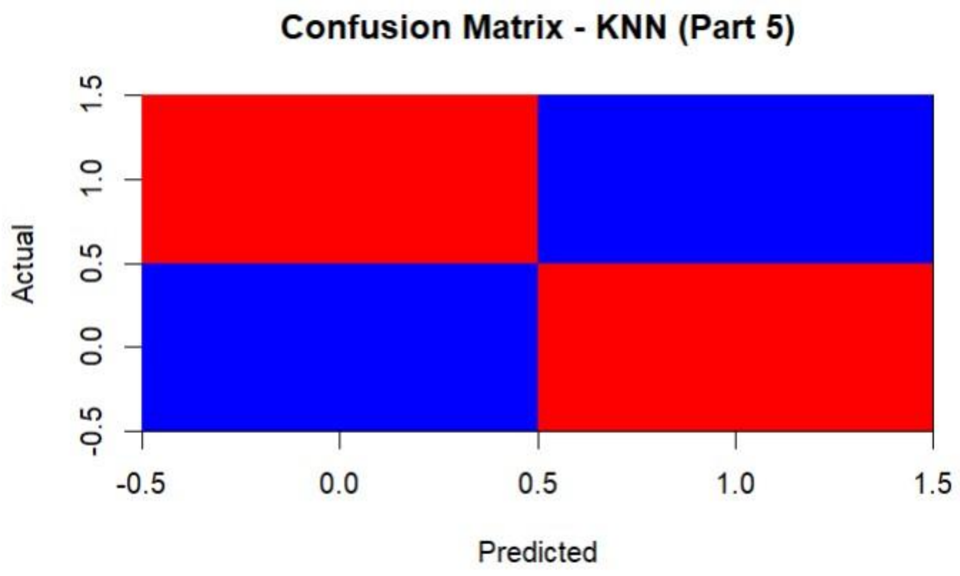
MODEL	DATASET	ACCURACY	PRECISION	RECALL	F1-SCORE	AUC
KNN	With Date and Location	0.781	0.809	0.941	0.869	0.664
	Without Date and Location	0.792	0.82	0.939	0.876	0.694
	Without Date and Location (w/ Hyperparameter Tuning)	0.789	0.804	0.963	0.876	0.692
	Without Date and Location (w/ Hyperparameter Tuning and Log Transformation)	0.805	0.818	0.966	0.886	0.738
	Without Date and Location (w/ Hyperparameter Tuning, Log Transformation, Feature Selection)	0.827	0.851	0.952	0.899	0.776
	Without Date and Location (w/ Hyperparameter Tuning, Log Transformation, Feature Selection, SMOTE balancing)	0.779	0.819	0.729	0.772	0.782
Logistic Regression	Without Date and Location	0.846	0.87	0.942	0.905	0.87
	Without Date and Location (w/ Log Transformation)	0.848	0.871	0.944	0.906	0.87
	Without Date and Location (w/ Log Transformation and Hyperparameter Tuning)	0.847	0.869	0.944	0.905	0.871
	Without Date and Location (w/ Log Transformation, Hyperparameter Tuning, SMOTE balancing)	0.793	0.927	0.797	0.857	0.873
	Without Date and Location (w/ SMOTE balancing)	0.797	0.927	0.802	0.86	0.869

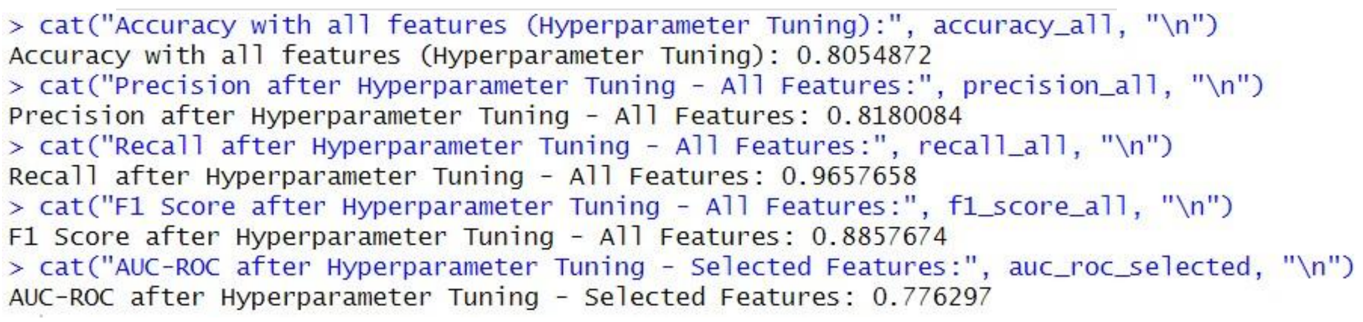
For K-Nearest Neighbour (KNN):

The K-Nearest Neighbour (KNN) model achieved an accuracy of 82.7%, with a particularly high recall of 95.2%, indicating its effectiveness in identifying positive cases. However, the precision, F1-Score, and AUC suggest a balanced performance across all metrics.

Configurations: Without Date & Location (w/ Hyperparameter Tuning, Log Transformation, Feature Selection) **Metrics:**

- Accuracy: 82.7%
- Precision: 85.1%
- Recall: 95.2%
- F1-Score: 89.9%
- AUC: 77.6%
- True Positives: 252
- True Negatives: 2113
- False Positives: 371
- False Negatives: 107





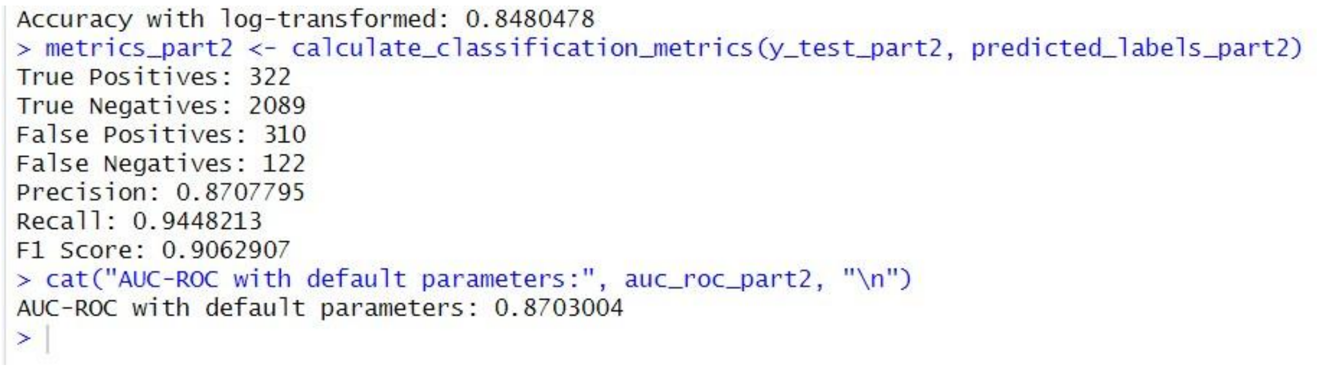
```
> cat("Accuracy with all features (Hyperparameter Tuning):", accuracy_all, "\n")
Accuracy with all features (Hyperparameter Tuning): 0.8054872
> cat("Precision after Hyperparameter Tuning - All Features:", precision_all, "\n")
Precision after Hyperparameter Tuning - All Features: 0.8180084
> cat("Recall after Hyperparameter Tuning - All Features:", recall_all, "\n")
Recall after Hyperparameter Tuning - All Features: 0.9657658
> cat("F1 Score after Hyperparameter Tuning - All Features:", f1_score_all, "\n")
F1 Score after Hyperparameter Tuning - All Features: 0.8857674
> cat("AUC-ROC after Hyperparameter Tuning - Selected Features:", auc_roc_selected, "\n")
AUC-ROC after Hyperparameter Tuning - Selected Features: 0.776297
```

For Logistic Regression:

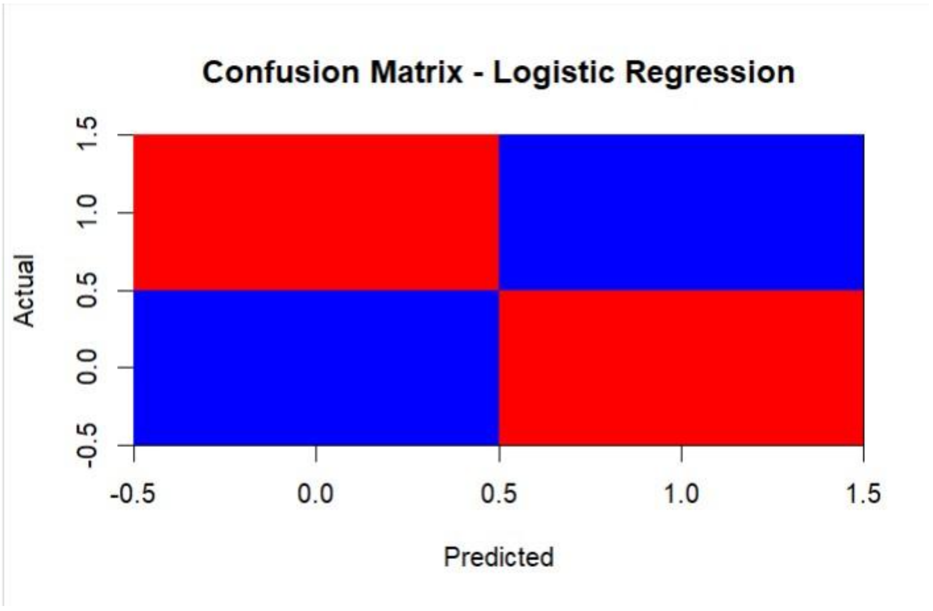
Logistic Regression model, despite a slightly higher accuracy of 84.8%, showed comparable precision and recall values. The AUC of 87.0% indicates good discriminative power.

Configurations: Without Date & Location (w/Log Transformation) **Metrics:**

- Accuracy: 84.8%
- Precision: 87.1% • Recall: 94.4%
- F1-Score: 90.6%
- AUC: 87%
- True Positives: 322
- True Negatives: 2089
- False Positives: 310
- False Negatives: 122



```
Accuracy with log-transformed: 0.8480478
> metrics_part2 <- calculate_classification_metrics(y_test_part2, predicted_labels_part2)
True Positives: 322
True Negatives: 2089
False Positives: 310
False Negatives: 122
Precision: 0.8707795
Recall: 0.9448213
F1 Score: 0.9062907
> cat("AUC-ROC with default parameters:", auc_roc_part2, "\n")
AUC-ROC with default parameters: 0.8703004
>
```



.

Comparing the models:

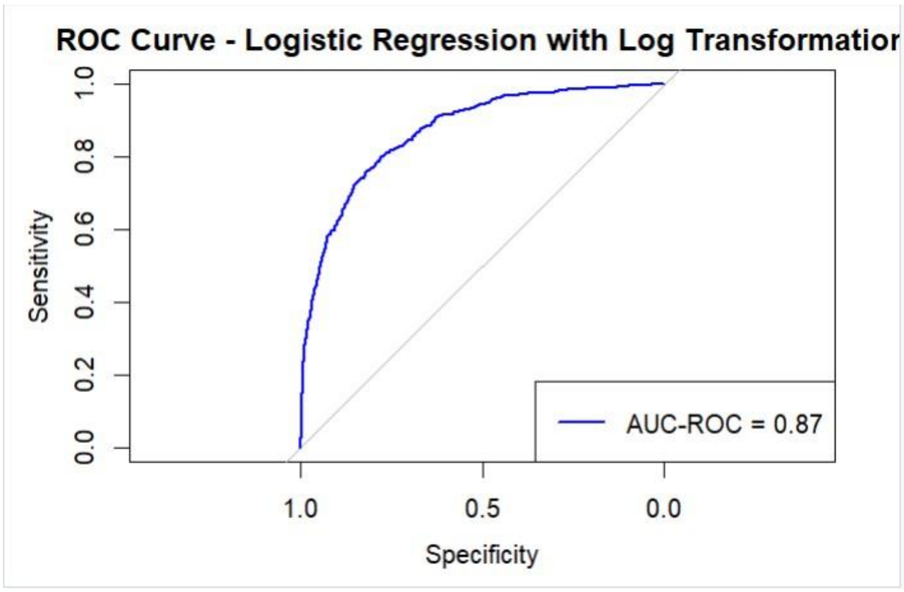
Both KNN and Logistic Regression performed well but Logistic Regression consistently achieved higher accuracy.

1. Accuracy: Logistic Regression shows a higher accuracy and indicates a better overall performance.
2. Precision: Logistic Regression Exhibits a superior precision which means it more accurately predicts positive outcomes.
3. Recall: KNN has a slightly better recall as compared to Logistic Regression which means better ability to capture positives instances.

4. F1-Score: Both models show similar scores and a good balance between recall and precision.
5. AUC: Logistic Regression has a better AUC score showing an enhanced ability to distinguish between positive and negative cases.

Overall Best Model: Logistic Regression Without Date & Location (w/Log Transformation)

Logistic Regression Without Date & Location (w/Log Transformation) with accuracy of 84.8%, recall of 94.4%, precision of 87.1%, f1-score of 90.6%, and AUC of 87%. This configuration achieves a best performance across all evaluation metrics.



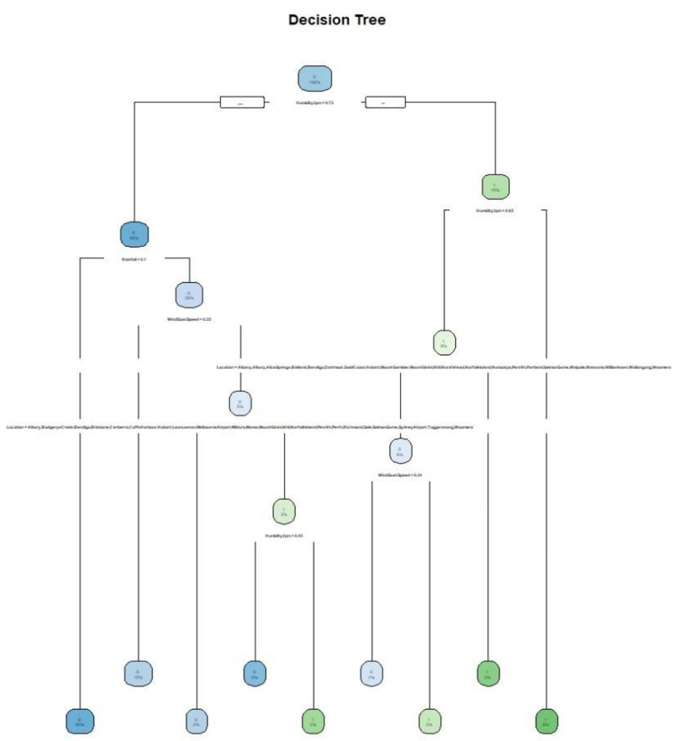
6. DECISION TREE ALGORITHM

The decision tree algorithm is versatile and simple as it can be used of both classification and regression tasks. The decision tree has the structure of flowcharts. The leaf nodes represent the class label and the internal nodes of the flowchart represents a test done on an attribute.

Initially the dataset is split into training and testing sets. In Decision Trees the target variable to be implemented, it must be in the form of a factor. So, the target variable (**RainTomorrow)** column, is then converted to a factor.

The Decision Tree model is built using the rpart function. The **RainTomorrow** target variable is predicted based on all other features in the **trained** dataset using the formula **RainTomorrow ~. The method = "class"** argument specifies that it's a classification tree the column RainTomorrow is categorical.

We then visualise the tree using the rpart.plot function and the output obtained is as follows:-



Using the **predict** function the predictions are made on the test dataset (**testData)**. The **type = "class"** argument returns the class prediction for each instance in the test set.

A confusion matrix is created using **confusionMatrix** from the **caret** package. This matrix is essential to evaluate the performance of the classification model.

Metrics such as Accuracy, Precision, Recall, and F1 score are calculated from the confusion matrix. These metrics provide insight into how well the model is performing.

The Receiver Operating Characteristic (ROC) curve and the Area Under the Curve (AUC) are implemented. The ROC and the AUC are important for measuring the performance of the model especially in datasets where classes are imbalanced.

The **pROC** package is used to implement the ROC object, and **auc** is used to estimate the AUC score.

The ROC curve is a graph which represents the true positive rate and the false positive rate at various threshold intervals. The AUC represents the degree or measure of separability achieved by the model.

The **plot** function is used to plot the ROC curve which helps to represent the performance of the model visually.

The estimated values of the performance metrics are as follows:-

Accuracy	**Precision**	**Recall**	**AUC**	**f-1 Score**
0.8304409	**0.8514503**	**0.9472256**	**0.8658108**	**0.896788**

Section 4: Discussion of the findings, Conclusion and Recommendations

1. Artificial Neural Network
2. K-nearest neighbours
3.Gradient Boosting
4. Decision Tree
5. Naive Bayes
6. Decision Tree

Model Performance Comparison:

Logistic Regression:

Best Configuration: Logistic regression with log transformation consistently achieved the highest accuracy (84%) across group members.
Feature Importance: Identified key features impacting rain prediction: Pressure, Temperature, Windspeed, and Humidity.
Impact of Date and Location: Minimal impact observed, suggesting that these variables may not contribute significantly.

Artificial Neural Network (ANN):

Best Precision: ANN demonstrated the best precision per class and a favorable positive-to-negative prediction ratio.
Considerations: Though precise, the time complexity of ANN was noted, impacting its efficiency compared to other models.

K-Nearest Neighbours (KNN):

Best Recall: KNN exhibited the highest recall at 96%, making it proficient in capturing positive instances. Considerations: Model performance suggested suitability for scenarios where correctly identifying rainy days is crucial.

Gradient Boosting:

Imbalance Handling: Despite an 85% accuracy, gradient boosting struggled with class imbalance.
Recommendation: Further hyperparameter tuning and addressing imbalance could enhance its performance.

Decision Tree:

Overall Best Model: Decision tree was considered the best overall model but faced challenges in generalizing the positive class.
Feature Importance: Like logistic regression, identified Pressure, Temperature, Windspeed, and Humidity as influential.

Naive Bayes:
Naive Bayes is computationally less expensive, making it suitable for large datasets or real-time applications. If the dataset predominantly consists of categorical features, Naive Bayes may perform well.

We analysed the performance of all the models by comparing their accuracy, precision, recall, f1 and AUC. It is to be noted that since sampling was done on the data, we can use accuracy to compare the metrics between them.

FINAL MODEL AFTER CONCLUSION: LOGISTIC REGRESSION

For the logistic regression model, comparing what we found out, the best accuracy (84%) for the logistic regression model with log transformation of the features. This suggests that the feature transformation is useful and gives us the best recall in most cases.

The Artificial Neural network is found to have the best precision per class and the ratio of positive to negative predictions. The KNN has the best recall out of all the models with 96 per cent. Gradient boosting did not perform much for this dataset even with 85% accuracy, it did not handle the class imbalance well. The decision tree model was seen to be the best overall model, but the positive class was not generalised properly in the model.

Hence from the modelling, the most important features based on the feature selection and the interpretation of the coefficient that increased the performance were Pressure, Temperature, Windspeed and Humidity.

The dates and the location were not found to have as much impact as we thought they did.

Our future recommendation that would increase the further performance of the model would be collecting more samples of data where it was raining which decreases the imbalance and would further improve the performance of all the models.

For business needs, the time taken is also an important factor while modelling so, we have compiled the time taken versus the accuracy which gave us:

Gradient Boosting >Logistic Regression>Decision Tree>KNN>ANN

Business Implications on the best model:

Stakeholders can use logistic regression for accurate and interpretable rain predictions with a low accurate computation time and cost.
It noted considering trade-offs between precision and recall based on specific business needs.
It is also easier to emphasize the ease of monitoring and updating models to adapt to changing data patterns.