Title: Statistics Project Technical Report

Authors: MOHAMMAD EHSANI (D03000022)

University: University of Naples Federico II

Course: STATISTICAL LEARNING AND DATA ANALYSIS

Instructor: ROBERTA SICILIANO

Date: 28 February 2024

Table of contents:

Introduction

In this project, my aim is to determine if there will be rainfall in Australia tomorrow. I utilize Python and Scikit-Learn to apply Logistic Regression.

My approach involves constructing a classifier to anticipate rainfall occurrences in Australia for the following day. I employ Logistic Regression to train a binary classification model, utilizing the Rain in Australia dataset for this analysis.

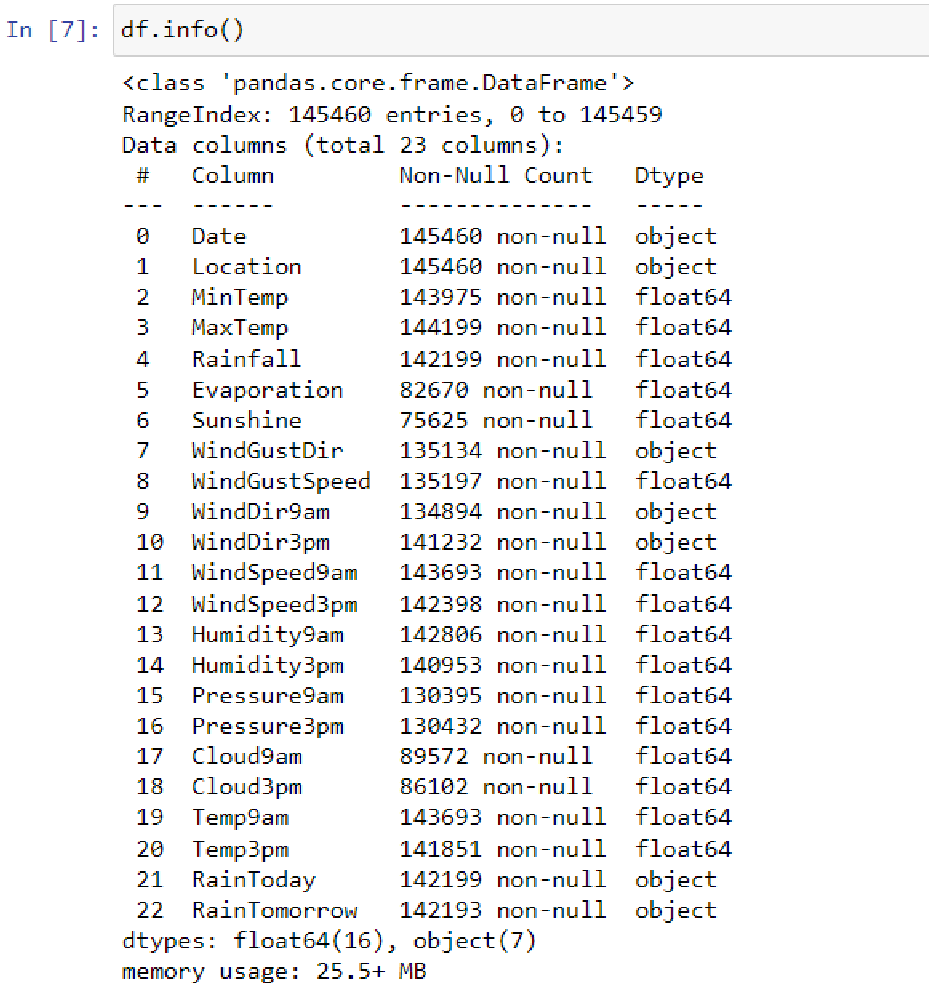
DataSet Understanding

We have 23 columns and 145,460 rows comprising both categorical and numerical variables. The dataset contains some missing values, which I will discuss further in subsequent sections. This dataset encompasses approximately 10 years of daily weather observations from numerous locations across Australia.

Table below summarize each variable's name, type, and unit:

Column Name	**Description**	**Data Type**
Date	Date of weather observation	String
Location	Location of weather observation	String
MinTemp	Minimum temperature	Celsius
MaxTemp	Maximum temperature	Celsius
Rainfall	Amount of rainfall	Millimeters (mm)
Evaporation	Amount of water evaporation	Millimeters (mm)
Sunshine	Number of hours of sunshine	Hours
WindGustDir	Direction of strongest wind gust	Categorical
WindGustSpeed	Speed of strongest wind gust	Kilometers per hour (km/h)
WindDir9am	Wind direction at 9 am	Categorical
WindDir3pm	Wind direction at 3 pm	Categorical
WindSpeed9am	Wind speed at 9 am	Kilometers per hour (km/h)
WindSpeed3pm	Wind speed at 3 pm	Kilometers per hour (km/h)

Humidity9am	Humidity level at 9 am	Percent (%)
Humidity3pm	Humidity level at 3 pm	Percent (%)
Pressure9am	Atmospheric pressure at 9 am	Hectopascals (hPa)
Pressure3pm	Atmospheric pressure at 3 pm	Hectopascals (hPa)
Cloud9am	Fraction of sky covered by clouds at 9 am	Eights (0-8)
Cloud3pm	Fraction of sky covered by clouds at 3 pm	Eights (0-8)
Temp9am	Temperature at 9 am	Celsius
Temp3pm	Temperature at 3 pm	Celsius
RainToday	Did it rain today? (Yes/No)	Binary
RainTomorrow	Will it rain tomorrow? (Yes/No)	Binary (Target Variable)



```
In [7]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 23 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   Date           145460 non-null  object 
 1   Location       145460 non-null  object 
 2   MinTemp        143975 non-null  float64
 3   MaxTemp        144199 non-null  float64
 4   Rainfall       142199 non-null  float64
 5   Evaporation    82670 non-null   float64
 6   Sunshine       75625 non-null   float64
 7   WindGustDir    135134 non-null  object 
 8   WindGustSpeed  135197 non-null  float64
 9   WindDir9am     134894 non-null  object 
 10  WindDir3pm     141232 non-null  object 
 11  WindSpeed9am   143693 non-null  float64
 12  WindSpeed3pm   142398 non-null  float64
 13  Humidity9am    142806 non-null  float64
 14  Humidity3pm    140953 non-null  float64
 15  Pressure9am    130395 non-null  float64
 16  Pressure3pm    130432 non-null  float64
 17  Cloud9am       89572 non-null   float64
 18  Cloud3pm       86102 non-null   float64
 19  Temp9am        143693 non-null  float64
 20  Temp3pm        141851 non-null  float64
 21  RainToday      142199 non-null  object 
 22  RainTomorrow   142193 non-null  object 
dtypes: float64(16), object(7)
memory usage: 25.5+ MB
```

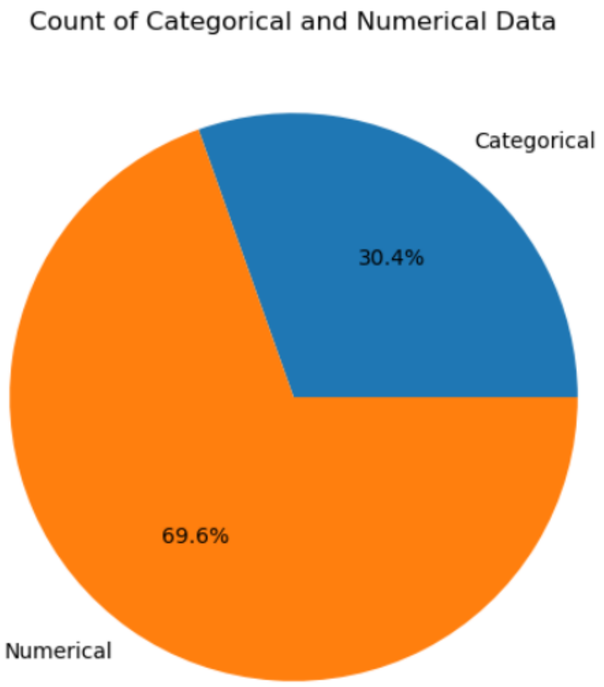
DataSet Structure

There are 16 numerical variables, which is :

MinTemp, MaxTemp, Rainfall, Evaporation, Sunshine, WindGustSpeed, WindSpeed9am, WindSpeed3pm, Humidity9am, Humidity3pm, Pressure9am, Pressure3pm, Cloud9am, Cloud3pm, Temp9am, Temp3pm

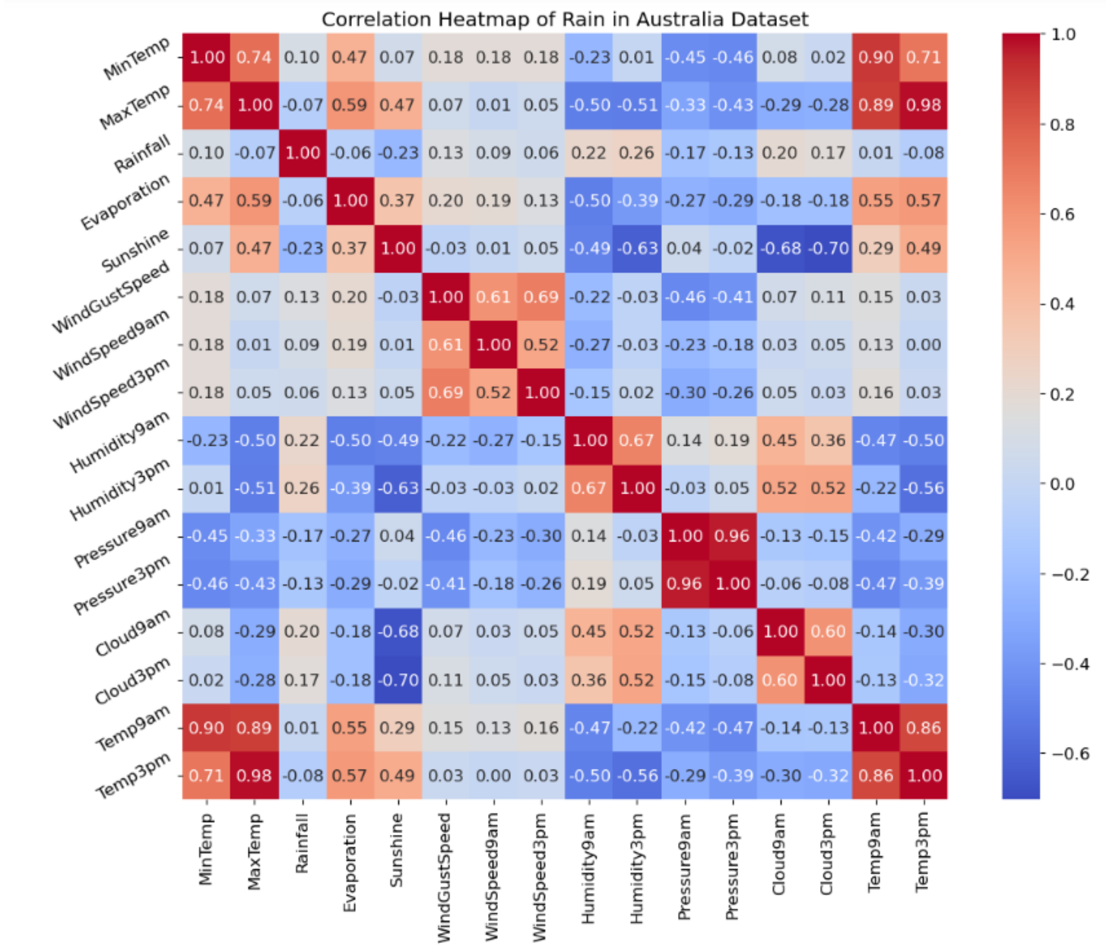
And, The categorical variables are:

Date, Location, WindGustDir, WindDir9am, WindDir3pm, RainToday, RainTomorrow



- The pie chart shows the distribution of categorical and numerical data in a dataset.
- Categorical data, represented by the orange color, makes up 30.4% of the data.
- Numerical data, represented by the blue color, makes up 69.6% of the data.

. Correlation for Bivariate Analysis



Correlation Heatmap of Rain in Australia Dataset

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	Cloud9am	Cloud3pm	Temp9am	Temp3pm
MinTemp	1.00	0.74	0.10	0.47	0.07	0.18	0.18	0.18	-0.23	0.01	-0.45	-0.46	0.08	0.02	0.90	0.71
MaxTemp	0.74	1.00	-0.07	0.59	0.47	0.07	0.01	0.05	-0.50	-0.51	-0.33	-0.43	-0.29	-0.28	0.89	0.98
Rainfall	0.10	-0.07	1.00	-0.06	-0.23	0.13	0.09	0.06	0.22	0.26	-0.17	-0.13	0.20	0.17	0.01	-0.08
Evaporation	0.47	0.59	-0.06	1.00	0.37	0.20	0.19	0.13	-0.50	-0.39	-0.27	-0.29	-0.18	-0.18	0.55	0.57
Sunshine	0.07	0.47	-0.23	0.37	1.00	-0.03	0.01	0.05	-0.49	-0.63	0.04	-0.02	-0.68	-0.70	0.29	0.49
WindGustSpeed	0.18	0.07	0.13	0.20	-0.03	1.00	0.61	0.69	-0.22	-0.03	-0.46	-0.41	0.07	0.11	0.15	0.03
WindSpeed9am	0.18	0.01	0.09	0.19	0.01	0.61	1.00	0.52	-0.27	-0.03	-0.23	-0.18	0.03	0.05	0.13	0.00
WindSpeed3pm	0.18	0.05	0.06	0.13	0.05	0.69	0.52	1.00	-0.15	0.02	-0.30	-0.26	0.05	0.03	0.16	0.03
Humidity9am	-0.23	-0.50	0.22	-0.50	-0.49	-0.22	-0.27	-0.15	1.00	0.67	0.14	0.19	0.45	0.36	-0.47	-0.50
Humidity3pm	0.01	-0.51	0.26	-0.39	-0.63	-0.03	-0.03	0.02	0.67	1.00	-0.03	0.05	0.52	0.52	-0.22	-0.56
Pressure9am	-0.45	-0.33	-0.17	-0.27	0.04	-0.46	-0.23	-0.30	0.14	-0.03	1.00	0.96	-0.13	-0.15	-0.42	-0.29
Pressure3pm	-0.46	-0.43	-0.13	-0.29	-0.02	-0.41	-0.18	-0.26	0.19	0.05	0.96	1.00	-0.06	-0.08	-0.47	-0.39
Cloud9am	0.08	-0.29	0.20	-0.18	-0.68	0.07	0.03	0.05	0.45	0.52	-0.13	-0.06	1.00	0.60	-0.14	-0.30
Cloud3pm	0.02	-0.28	0.17	-0.18	-0.70	0.11	0.05	0.03	0.36	0.52	-0.15	-0.08	0.60	1.00	-0.13	-0.32
Temp9am	0.90	0.89	0.01	0.55	0.29	0.15	0.13	0.16	-0.47	-0.22	-0.42	-0.47	-0.14	-0.13	1.00	0.86
Temp3pm	0.71	0.98	-0.08	0.57	0.49	0.03	0.00	0.03	-0.50	-0.56	-0.29	-0.39	-0.30	-0.32	0.86	1.00

Here, we observe in the Correlation heat map that the MinTemp variable demonstrates a high positive correlation with MaxTemp (correlation coefficient = 0.74),

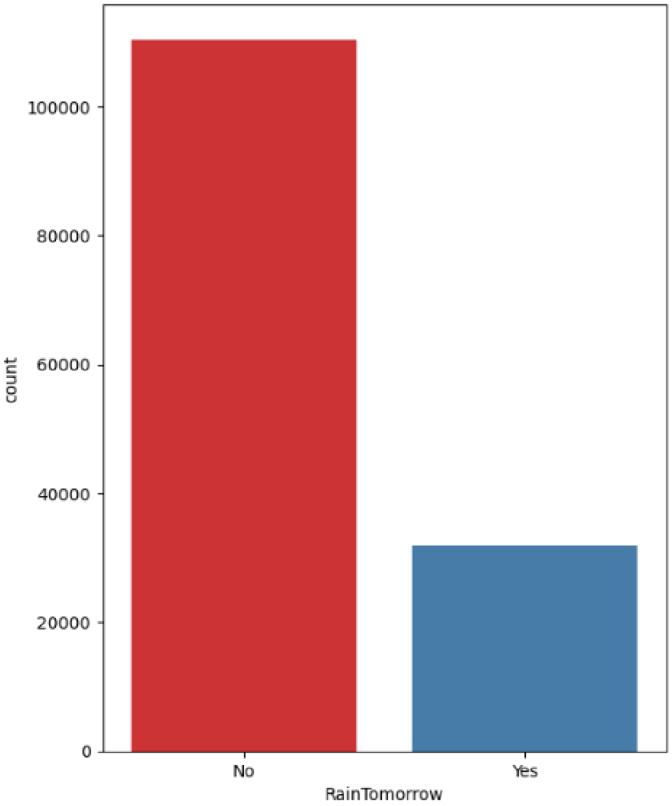
Temp3pm (correlation coefficient = 0.71), and Temp9am (correlation coefficient = 0.90).

Furthermore, MaxTemp also exhibits strong positive correlations with Temp9am (correlation coefficient = 0.89) and Temp3pm (correlation coefficient = 0.98).

Similarly, WindGustSpeed is highly positively correlated with WindSpeed3pm (correlation coefficient = 0.69), while Pressure9am correlates strongly with Pressure3pm (correlation coefficient = 0.96).

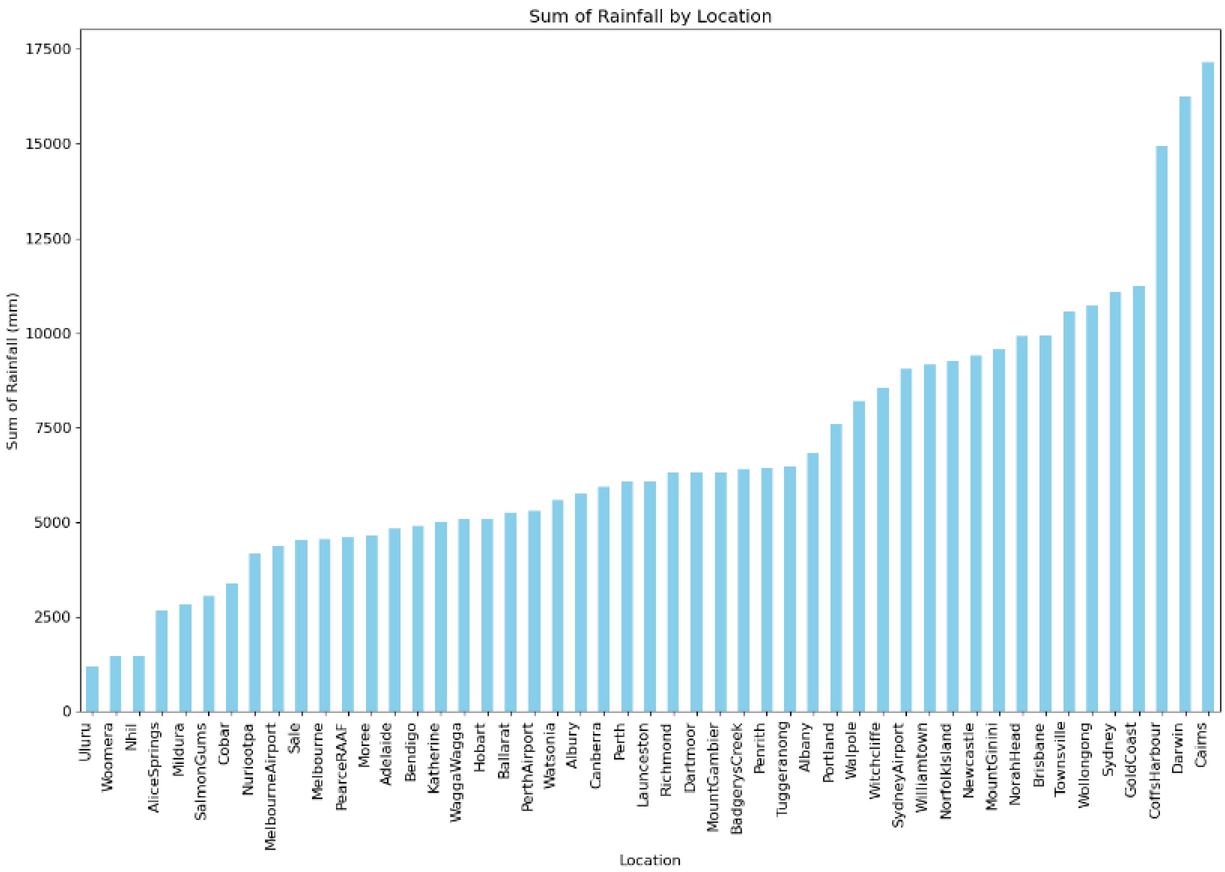
Moreover, Temp9am and Temp3pm display a strong positive correlation (correlation coefficient = 0.86).

. Data Visualization



. This graph shows the distribution of rain events.

It appears that it rains more often than it does not rain, based on the height of the 'yes' bar compared to the 'no' bar.

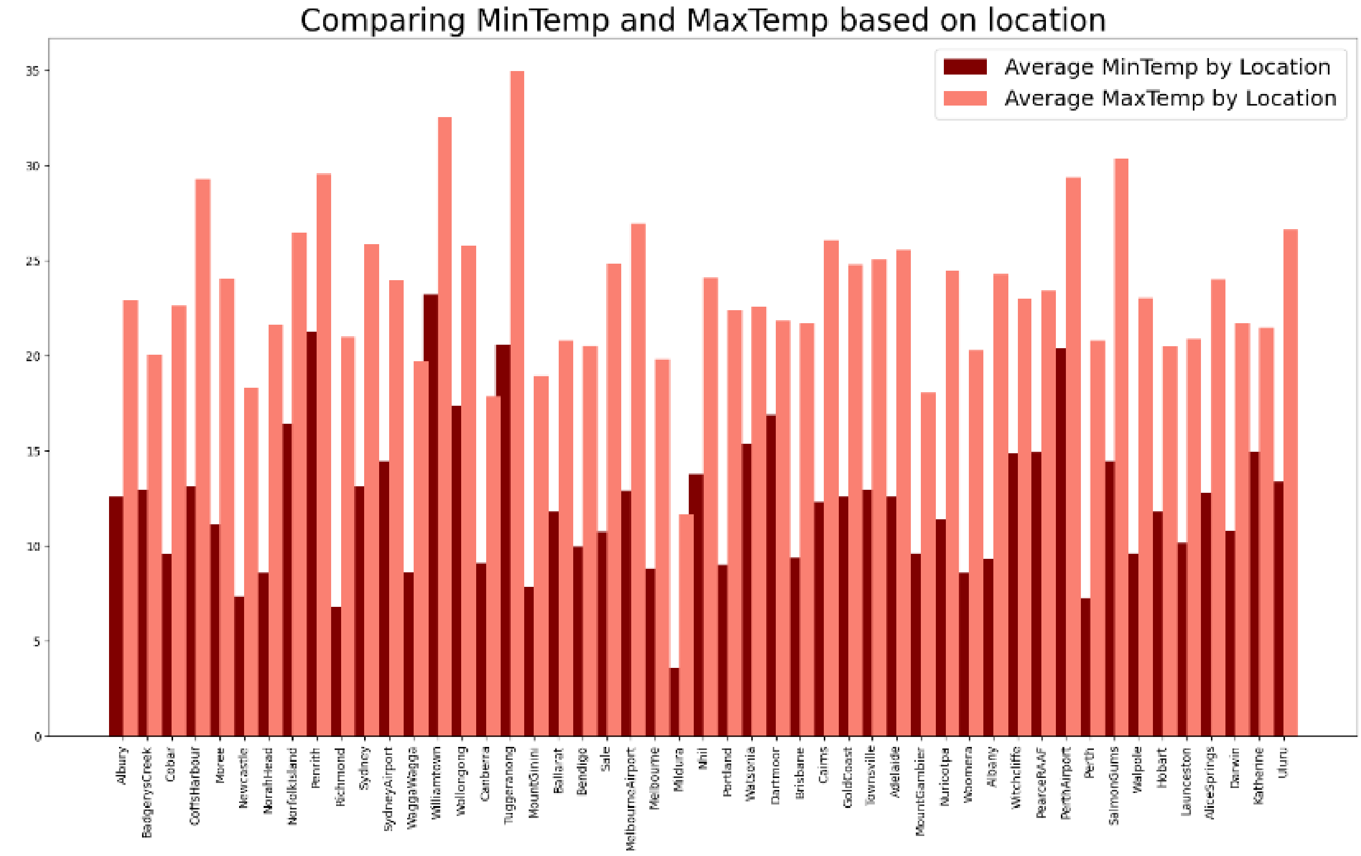


The graph shows the total amount of rainfall recorded at various locations in Australia. The locations with the lowest rainfalls are:

Uluru (17,500 mm)
Woomera (15,000 mm)
Nhil (12,500 mm)

The locations with the highest rainfalls are:

Darwin (2,500 mm)
Cairns (2,500 mm)
Brisbane (2,500 mm)

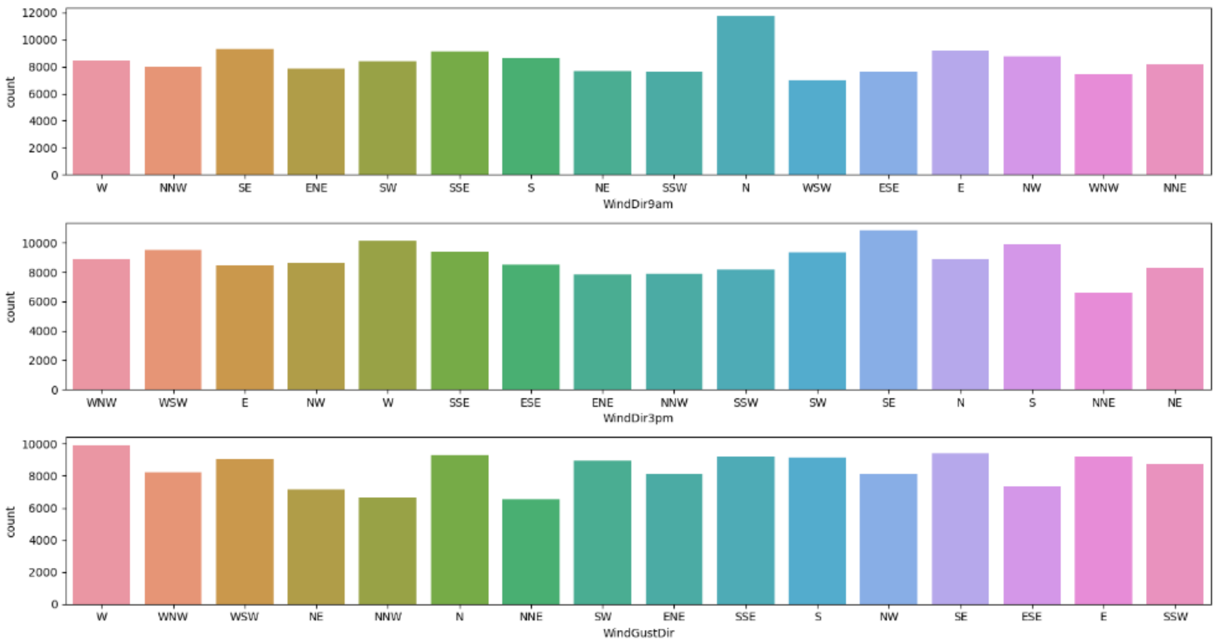


The bar chart presented herein depicts a comparative analysis between the variables 'MinTemp' and 'MaxTemp' across multiple cities.

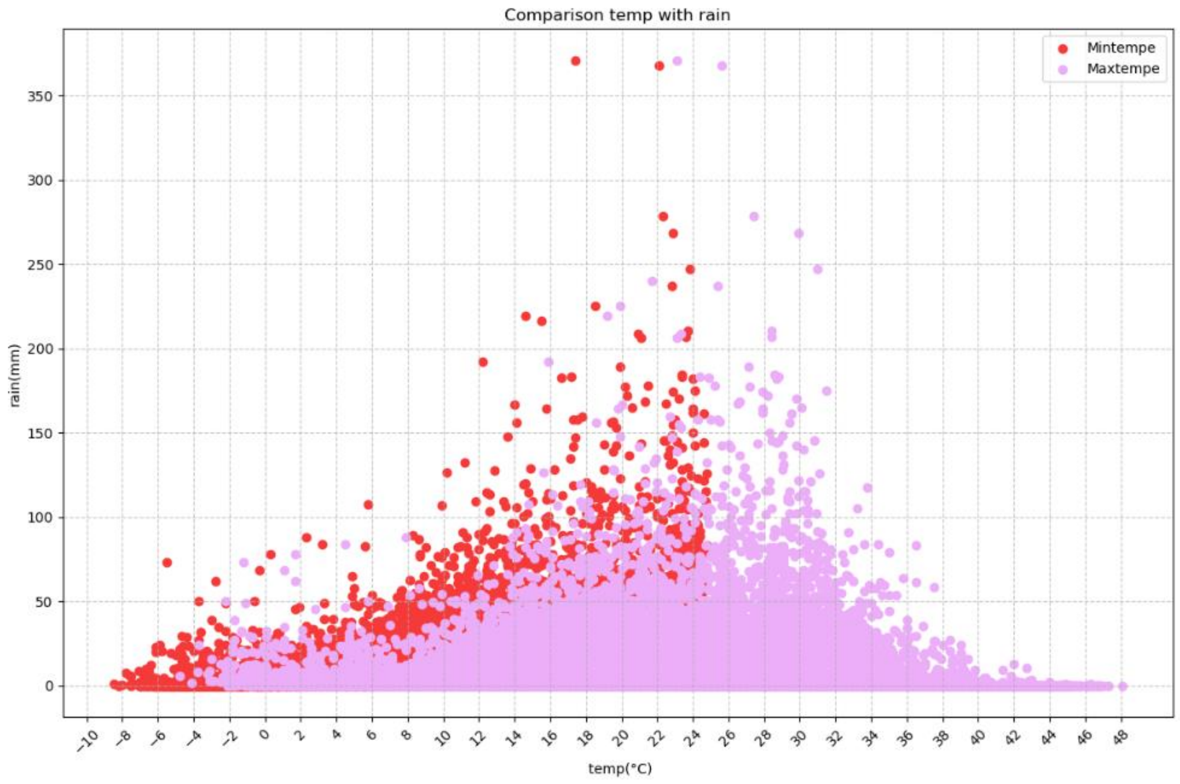
This visual representation facilitates a detailed examination of the temperature disparities and similarities among these locations.

By scrutinizing the data portrayed in the chart, one can discern notable variations in temperature patterns across diverse geographical regions.

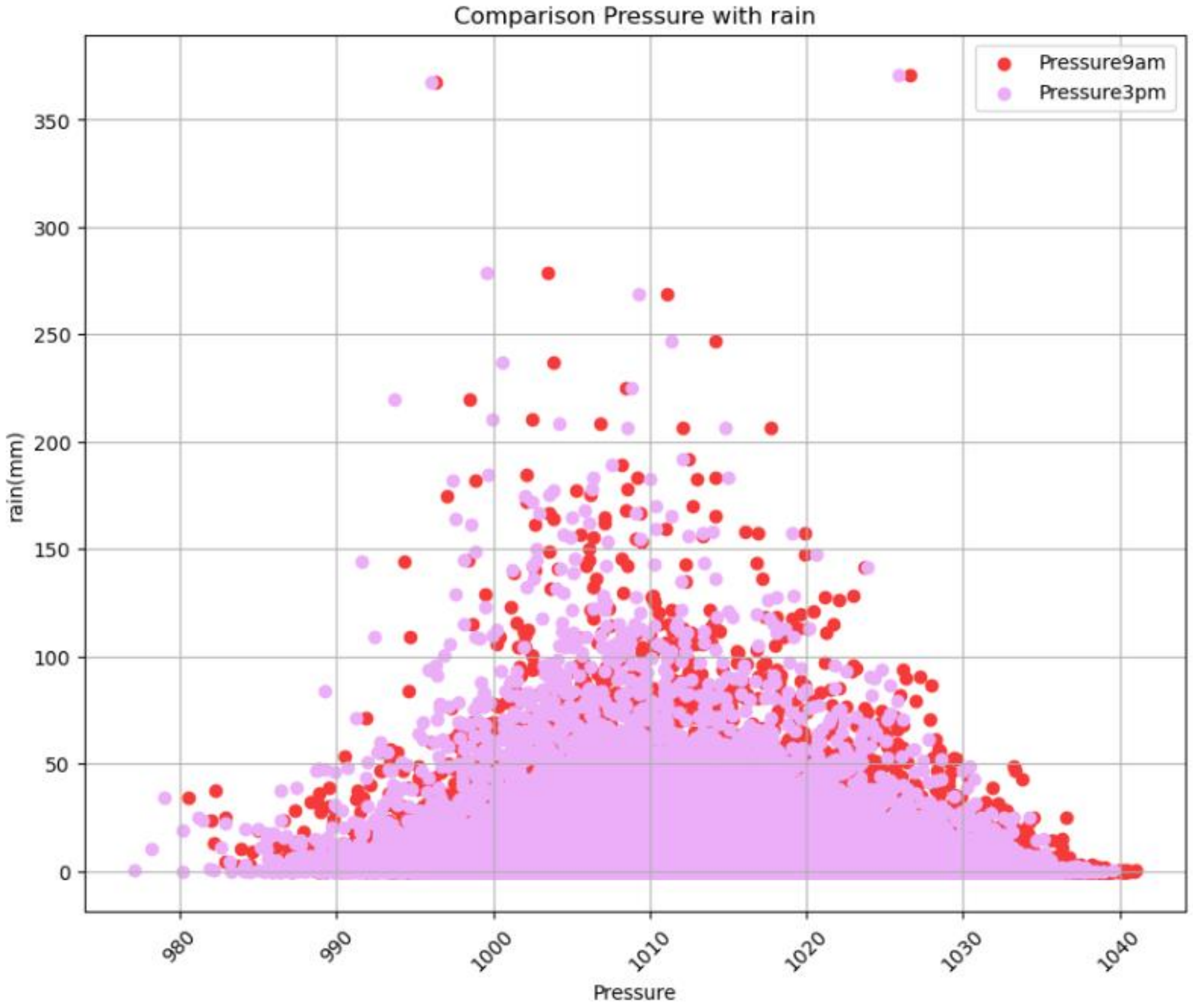
Such a comparison enables a comprehensive understanding of the relationship between 'MinTemp' and 'MaxTemp' within various city climates.



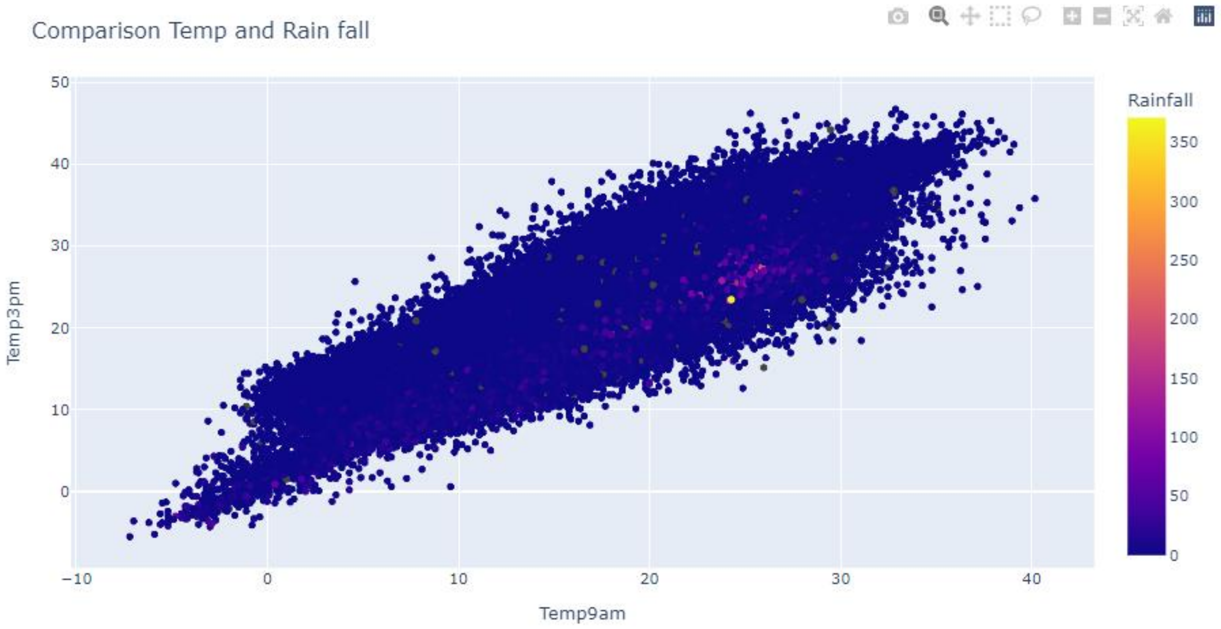
The provided count plot illustrates the distribution of wind direction at both 3 pm and 9 am, as well as highlights the prevailing wind direction. It is evident from the analysis that at 9 am, the majority of the wind is directed towards the north. Conversely, at 3 pm, predominant wind directions include south-east and west. Furthermore, upon closer examination, it is observed that the strongest wind gusts originate from the west, north, and south-east directions, respectively. This observation emphasizes the variability in wind patterns throughout the observed periods, with notable differences in direction and intensity.



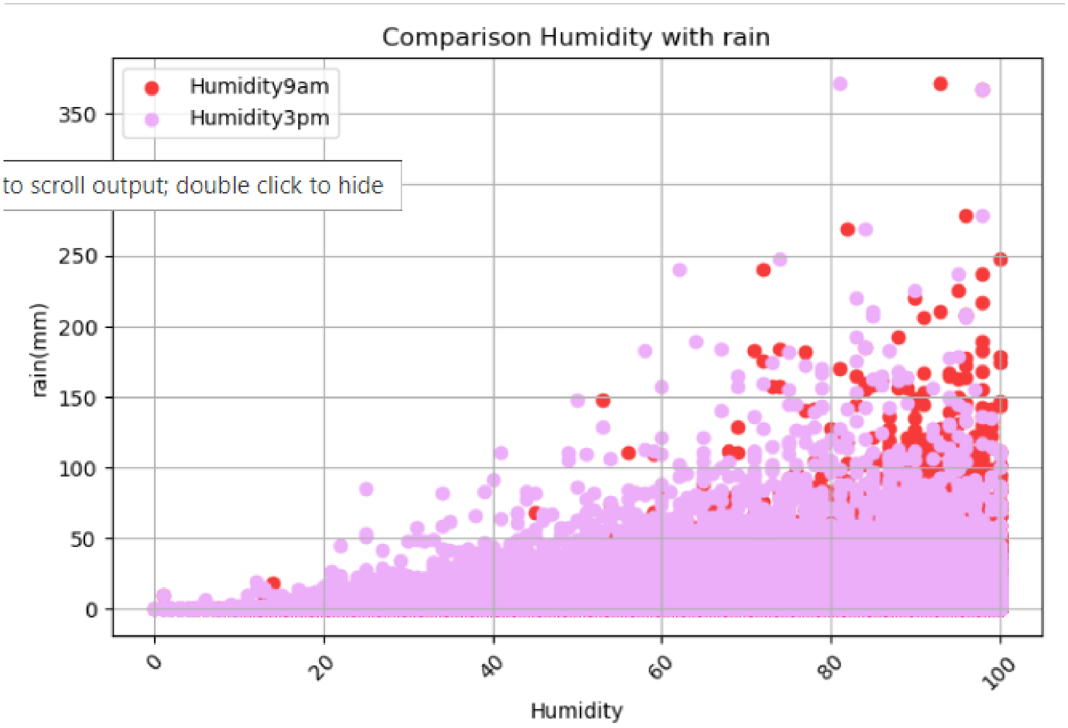
In this scatter plot, the distribution of rainfall is depicted, revealing that the majority of rainfall occurrences fall within the range of 10 to 32 degrees Celsius. Furthermore, it is notable that temperatures exceeding 38 degrees Celsius or dropping below 9 degrees Celsius correspond to the least amount of rainfall. This observation underscores the relationship between temperature fluctuations and rainfall patterns, suggesting that moderate temperatures are conducive to higher levels of precipitation while extreme temperatures are associated with decreased rainfall. Such insights into the interplay between temperature and rainfall can provide valuable information for understanding local climate dynamics and inform decision-making processes in various sectors, including agriculture, water resource management, and urban planning.



The scatter plot presented offers insights into the relationship between pressure readings at 9 am and 3 pm, alongside rainfall distribution. Notably, a predominant concentration of rainfall occurrences falls within the pressure range of 995 to 1030 hectopascals. Conversely, minimal rainfall is observed when pressure readings surpass 1030 hectopascals or dip below 990 hectopascals. This observation underscores the correlation between atmospheric pressure and rainfall patterns. Regions experiencing moderate pressure levels tend to exhibit higher levels of precipitation, while extremes in pressure correspond to reduced rainfall. Understanding these dynamics can contribute significantly to meteorological forecasting and climate modeling efforts, facilitating better-informed decisions across various sectors reliant on accurate weather predictions.

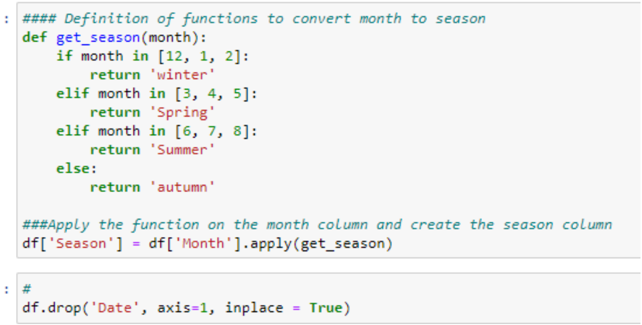


The scatter plot depicts the relationship between temperature and rainfall, revealing a weak positive correlation. This indicates that as temperatures increase, rainfall tends to rise as well. The plot resembles a heatmap, highlighting areas of elevated rainfall concentration. These insights aid in understanding regional climate dynamics and informing decision-making in sectors like agriculture and disaster preparedness.



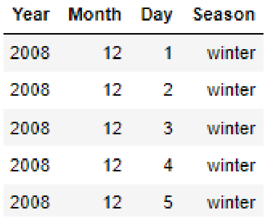
The scatter plot illustrates a clear correlation between humidity levels and rainfall. Rainfall is notably higher when humidity exceeds 70%, while it decreases significantly below 45% humidity. Understanding this relationship is crucial for various sectors, such as agriculture and urban planning, enabling better decision-making to manage risks associated with rainfall patterns and enhance resilience to changing climatic conditions.

. Data Pre-Processing and Feature Engineering



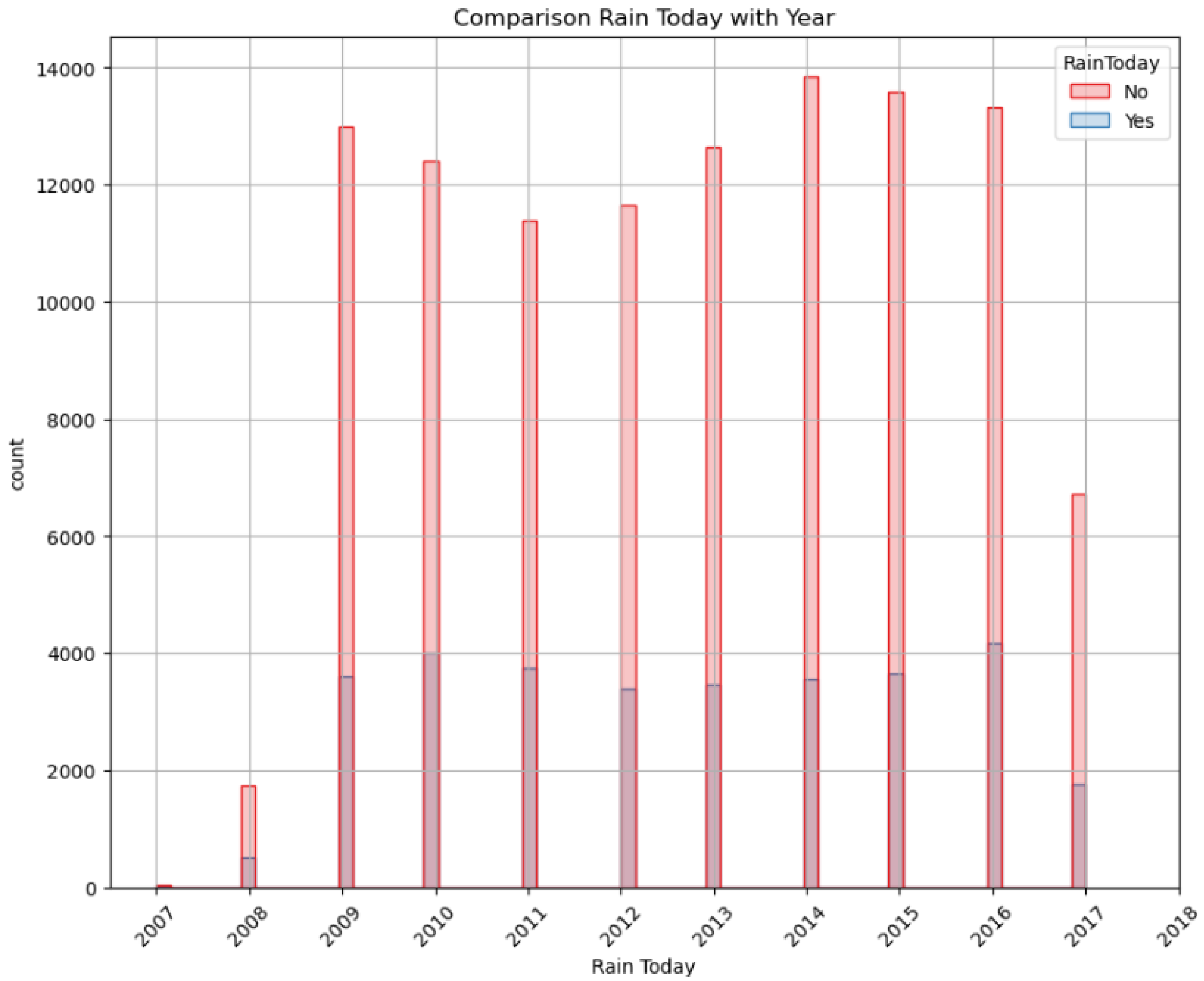
```
#### Definition of functions to convert month to season
def get_season(month):
    if month in [12, 1, 2]:
        return 'winter'
    elif month in [3, 4, 5]:
        return 'Spring'
    elif month in [6, 7, 8]:
        return 'Summer'
    else:
        return 'autumn'

###Apply the function on the month column and create the season column
df['Season'] = df['Month'].apply(get_season)
```

```
#
df.drop('Date', axis=1, inplace = True)
```

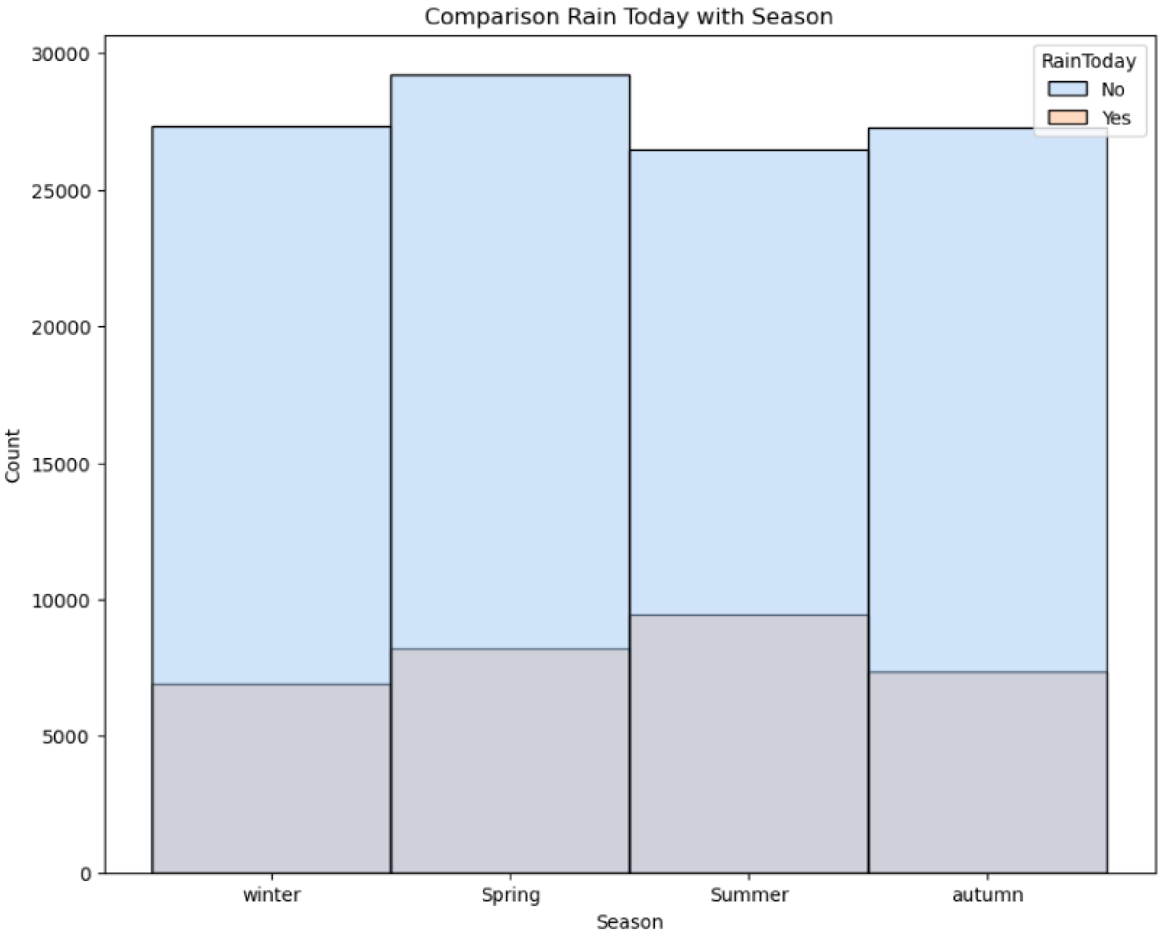


Year	Month	Day	Season
2008	12	1	winter
2008	12	2	winter
2008	12	3	winter
2008	12	4	winter
2008	12	5	winter

To improve visualization and gain deeper insights, I divided the date column into separate components: Year, Month, Day, and Season. This segmentation allows for a more detailed analysis of temporal patterns, particularly across different seasons. Subsequently, I removed the original Date column to streamline further analysis. This approach enhances the clarity of visualizations and facilitates a more focused exploration of seasonal trends, thereby bolstering the robustness of subsequent analyses and insights derived from the dataset.

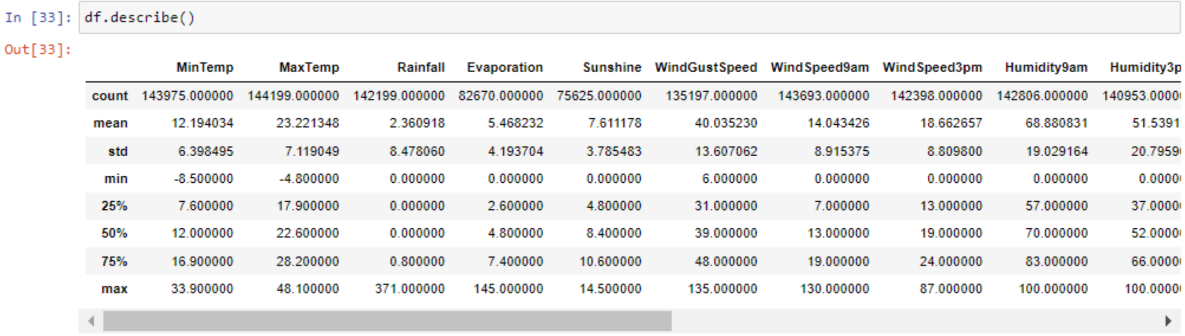


Most of the time, rainfall is absent, as depicted by the bar chart illustrating the distribution of rainfall between the years 2007 and 2018. Each bar in the chart is color-coded to indicate the count of rainy and non-rainy days within that year. This visualization offers valuable insights into the frequency and distribution of rainfall over the specified period, facilitating a clearer understanding of precipitation patterns throughout the years analyzed.



The histogram plot reveals a surprising trend where the majority of rainy days occur during summer and spring, while winter records the least rainfall. This observation could be attributed to specific weather conditions prevalent in Australia, influenced by factors such as humidity and temperature, as demonstrated in the preceding chart. Given these findings, such results were expected, indicating a correlation between seasonal variations and rainfall patterns, further underscoring the significance of local climate dynamics in shaping precipitation trends.

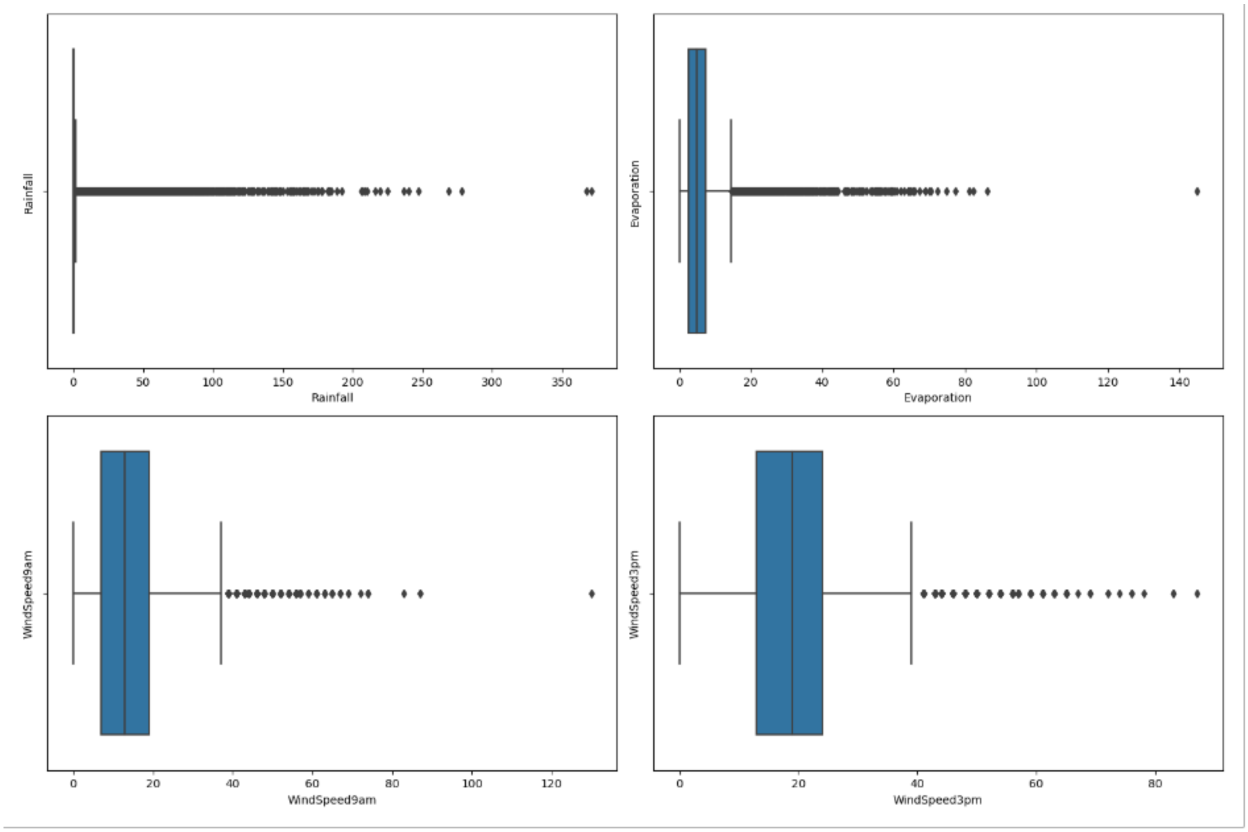
. Engineering outliers



```
In [33]: df.describe()
Out[33]:
```

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3p
count	143975.000000	144199.000000	142199.000000	82670.000000	75625.000000	135197.000000	143693.000000	142398.000000	142806.000000	140953.0000
mean	12.194034	23.221348	2.360918	5.468232	7.611178	40.035230	14.043426	18.662657	68.880831	51.5391
std	6.398495	7.119049	8.478060	4.193704	3.785483	13.607062	8.915375	8.809800	19.029164	20.7959
min	-8.500000	-4.800000	0.000000	0.000000	0.000000	6.000000	0.000000	0.000000	0.000000	0.0000
25%	7.600000	17.900000	0.000000	2.600000	4.800000	31.000000	7.000000	13.000000	57.000000	37.0000
50%	12.000000	22.600000	0.000000	4.800000	8.400000	39.000000	13.000000	19.000000	70.000000	52.0000
75%	16.900000	28.200000	0.800000	7.400000	10.600000	48.000000	19.000000	24.000000	83.000000	66.0000
max	33.900000	48.100000	371.000000	145.000000	14.500000	135.000000	130.000000	87.000000	100.000000	100.0000

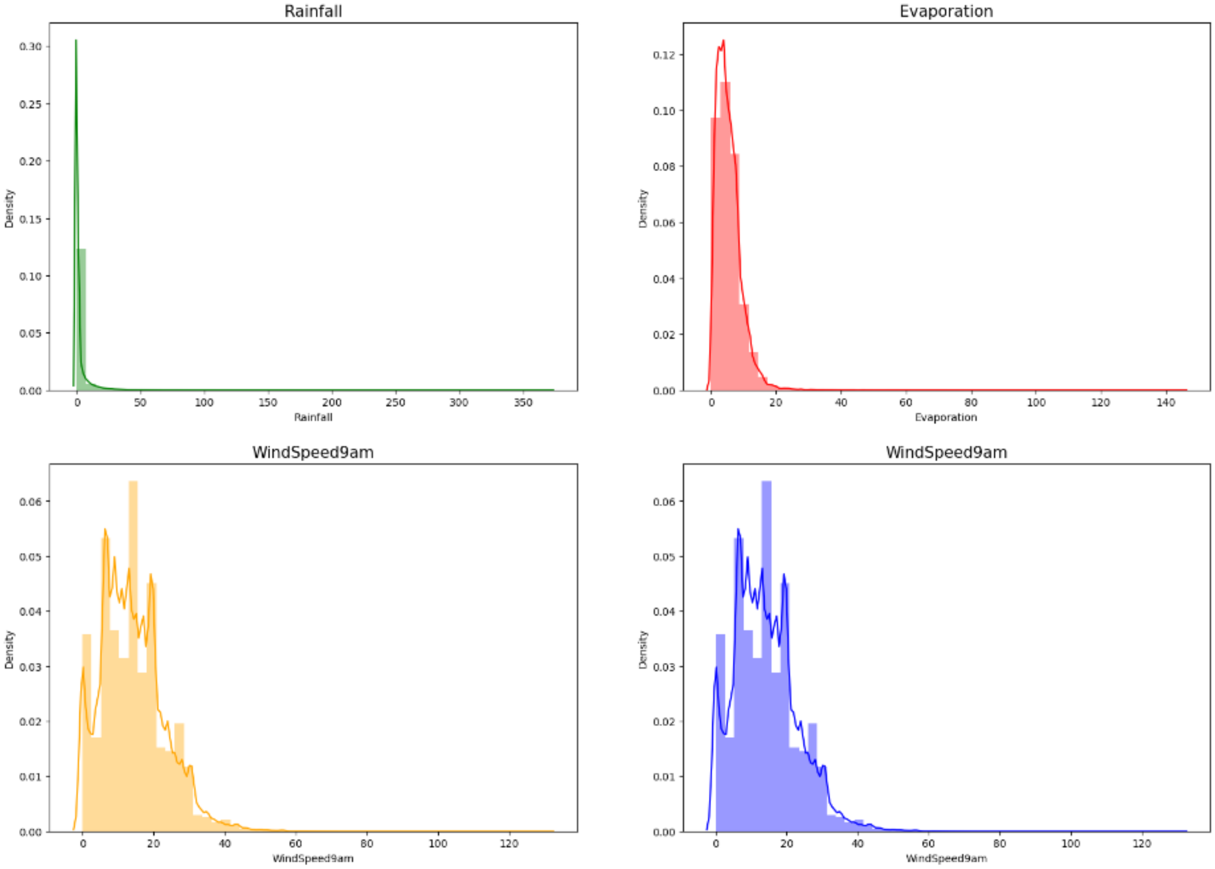
Upon closer examination, it becomes apparent that the Rainfall, Evaporation, WindSpeed9am, and WindSpeed3pm columns may contain outliers. This suspicion arises from the significant disparity between their 75th percentile values and their respective maximum values. To visualize and identify these outliers effectively, I will draw boxplots for the aforementioned variables.



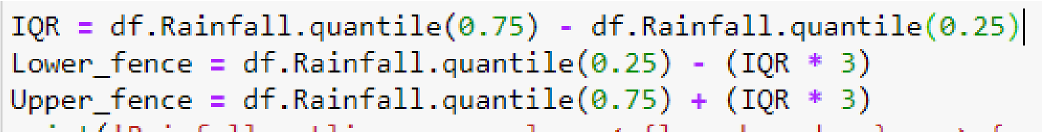
The boxplots above confirm the presence of numerous outliers in these variables.

Examining Variable Distributions

Next, I will plot histograms to assess the distributions and determine whether they are normal or skewed. If a variable follows a normal distribution, I will conduct Extreme Value Analysis. However, if the distributions are skewed, I will calculate the Interquartile Range (IQR).



It is evident that all four variables exhibit skewness. Therefore, I will utilize the Interquartile Range (IQR) method to identify outliers.



```
IQR = df.Rainfall.quantile(0.75) - df.Rainfall.quantile(0.25)
Lower_fence = df.Rainfall.quantile(0.25) - (IQR * 3)
Upper_fence = df.Rainfall.quantile(0.75) + (IQR * 3)
```

(IQR) method is:

Lower Fence = Q1 – (IQR * K)

Upper Fence = Q3 – (IQR * k)

Where:

Q1 is the first quartile (25th percentile)

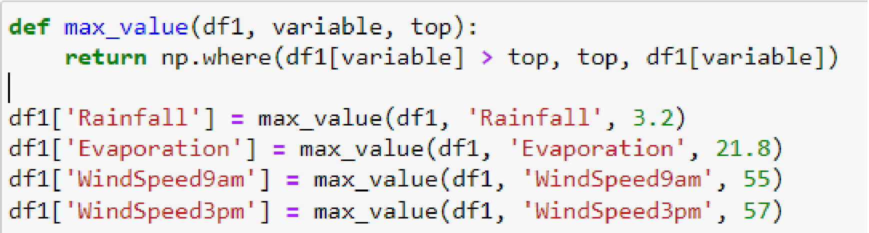
Q3 is the third quartile (75th percentile)

IQR is the Interquartile Range, calculated as Q3 – Q1

K is a constant multiplier that determines how far the fences are from the quartiles, which I determine here as 3

And we consider the amount bigger than Upper Fence and lower than Lower Fence as outliers.

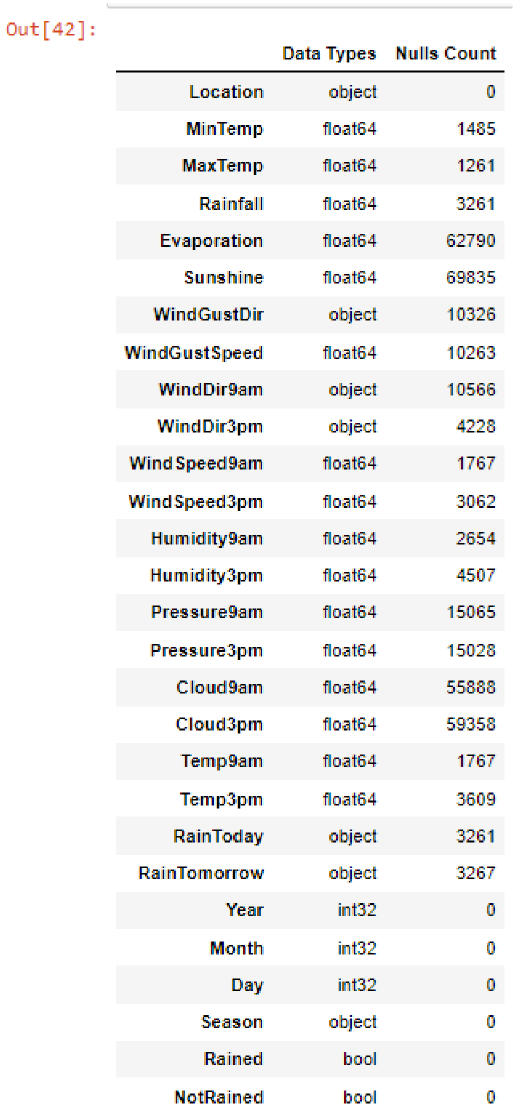
We have observed outliers in the Rainfall, Evaporation, WindSpeed9am, and WindSpeed3pm columns. To address this issue, I will employ a top-coding approach to cap the maximum values and subsequently remove outliers from these variables.



```
def max_value(df1, variable, top):
    return np.where(df1[variable] > top, top, df1[variable])

df1['Rainfall'] = max_value(df1, 'Rainfall', 3.2)
df1['Evaporation'] = max_value(df1, 'Evaporation', 21.8)
df1['WindSpeed9am'] = max_value(df1, 'WindSpeed9am', 55)
df1['WindSpeed3pm'] = max_value(df1, 'WindSpeed3pm', 57)
```

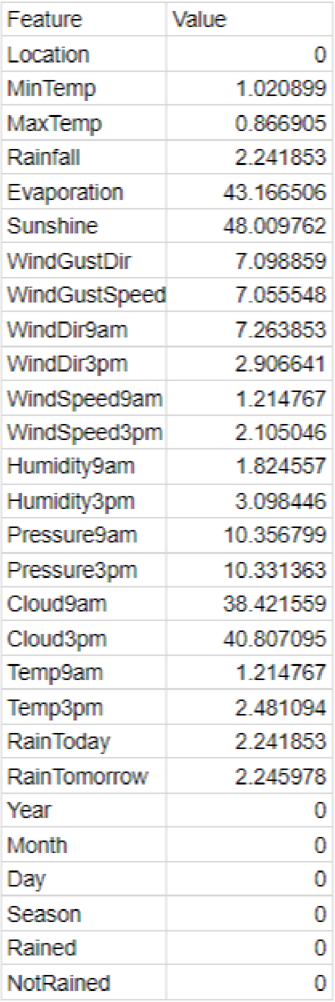
I will employ a top-coding method to cap the values at a maximum amount in specific columns and then delete outliers using this approach.

. Handling Missing values



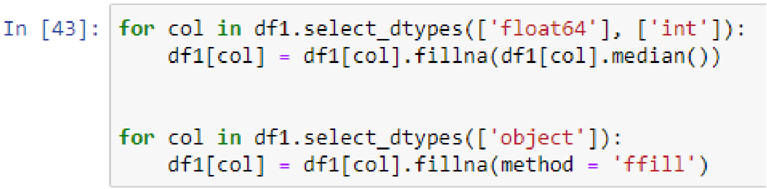
Out[42]:

	Data Types	Nulls Count
Location	object	0
MinTemp	float64	1485
MaxTemp	float64	1261
Rainfall	float64	3261
Evaporation	float64	62790
Sunshine	float64	69835
WindGustDir	object	10326
WindGustSpeed	float64	10263
WindDir9am	object	10566
WindDir3pm	object	4228
WindSpeed9am	float64	1767
WindSpeed3pm	float64	3062
Humidity9am	float64	2654
Humidity3pm	float64	4507
Pressure9am	float64	15065
Pressure3pm	float64	15028
Cloud9am	float64	55888
Cloud3pm	float64	59358
Temp9am	float64	1767
Temp3pm	float64	3609
RainToday	object	3261
RainTomorrow	object	3267
Year	int32	0
Month	int32	0
Day	int32	0
Season	object	0
Rained	bool	0
NotRained	bool	0



Feature	Value
Location	0
MinTemp	1.020899
MaxTemp	0.866905
Rainfall	2.241853
Evaporation	43.166506
Sunshine	48.009762
WindGustDir	7.098859
WindGustSpeed	7.055548
WindDir9am	7.263853
WindDir3pm	2.906641
WindSpeed9am	1.214767
WindSpeed3pm	2.105046
Humidity9am	1.824557
Humidity3pm	3.098446
Pressure9am	10.356799
Pressure3pm	10.331363
Cloud9am	38.421559
Cloud3pm	40.807095
Temp9am	1.214767
Temp3pm	2.481094
RainToday	2.241853
RainTomorrow	2.245978
Year	0
Month	0
Day	0
Season	0
Rained	0
NotRained	0

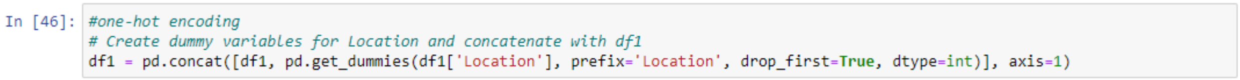
I created a simple data frame to assess the extent of missing values. Subsequently, I developed code to determine the percentage of missing values in each variable. This analysis revealed the presence of missing values in both categorical and numerical variables.



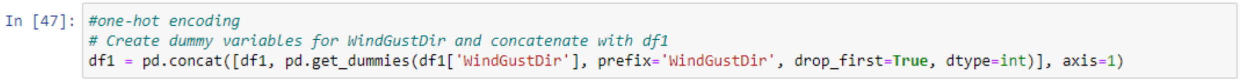
```
In [43]: for col in df1.select_dtypes(['float64'], ['int']):
             df1[col] = df1[col].fillna(df1[col].median())


         for col in df1.select_dtypes(['object']):
             df1[col] = df1[col].fillna(method = 'ffill')
```

The optimal approach for handling missing values in this scenario involves filling categorical variables with their mode. The mode, representing the value that occurs most frequently in the dataset, is a suitable choice for categorical data imputation. For numerical variables, filling them with their median is preferred. The median, which denotes the middle value when the data is sorted, is commonly used for filling missing values. Unlike the mean, the median is less influenced by outliers, rendering it a more robust measure of central tendency, particularly in instances of skewed data or the presence of outliers.

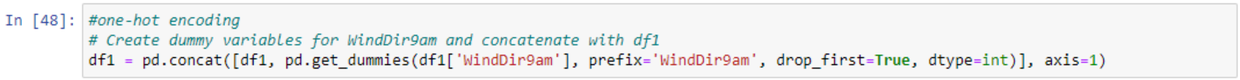
. Encoding Numerical and Categorical variables



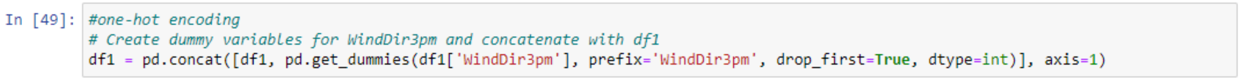
```
In [46]: #one-hot encoding
         # Create dummy variables for Location and concatenate with df1
         df1 = pd.concat([df1, pd.get_dummies(df1['Location'], prefix='Location', drop_first=True, dtype=int)], axis=1)
```



```
In [47]: #one-hot encoding
         # Create dummy variables for WindGustDir and concatenate with df1
         df1 = pd.concat([df1, pd.get_dummies(df1['WindGustDir'], prefix='WindGustDir', drop_first=True, dtype=int)], axis=1)
```



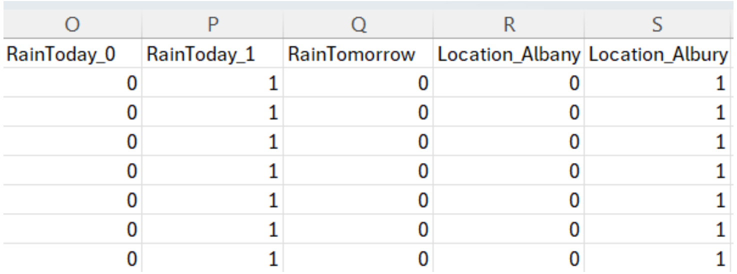
```
In [48]: #one-hot encoding
         # Create dummy variables for WindDir9am and concatenate with df1
         df1 = pd.concat([df1, pd.get_dummies(df1['WindDir9am'], prefix='WindDir9am', drop_first=True, dtype=int)], axis=1)
```



```
In [49]: #one-hot encoding
         # Create dummy variables for WindDir3pm and concatenate with df1
         df1 = pd.concat([df1, pd.get_dummies(df1['WindDir3pm'], prefix='WindDir3pm', drop_first=True, dtype=int)], axis=1)
```

I utilized one-hot encoding to convert categorical variables into numerical ones. One-hot encoding is a technique commonly employed in machine learning and data processing to transform categorical data into a binary format. This method is particularly advantageous when handling categorical variables in machine learning models, as numerous algorithms necessitate numerical input for effective processing and analysis.

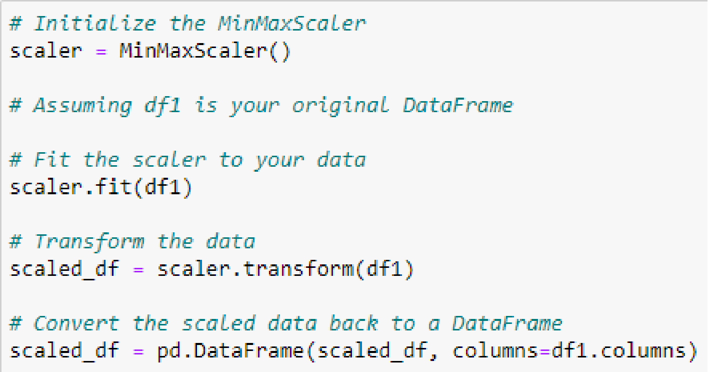
It changes our dataset like this:



O	P	Q	R	S
RainToday_0	RainToday_1	RainTomorrow	Location_Albany	Location_Albury
0	1	0	0	1
0	1	0	0	1
0	1	0	0	1
0	1	0	0	1
0	1	0	0	1
0	1	0	0	1
0	1	0	0	1

As an example of one-hot encoding, consider the categorical variable "Location" with 50 different locations. This process creates 50 distinct columns, where a value of 1 is assigned to rows corresponding to each specific location, while rows not matching the specific location are assigned a value of 0.

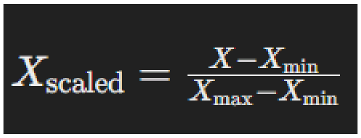
. Scaling Data



```
# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Assuming df1 is your original DataFrame

# Fit the scaler to your data
scaler.fit(df1)

# Transform the data
scaled_df = scaler.transform(df1)

# Convert the scaled data back to a DataFrame
scaled_df = pd.DataFrame(scaled_df, columns=df1.columns)
```

Another crucial step in modeling our dataset involves scaling the numerical variables. This step is essential because our dataset comprises various numerical variables, and machine learning algorithms may struggle to discern the differences between them. Thus, it's imperative to provide the algorithm with scaled data for optimal performance. To accomplish this, I utilized the MinMaxScaler library to scale our dataset.

The MinMaxScaler library:



$$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

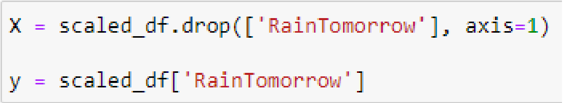
Variable is the original feature value.

VariableMin is the minimum value of the feature in the dataset.

VariableMax is the maximum value of the feature in the dataset.

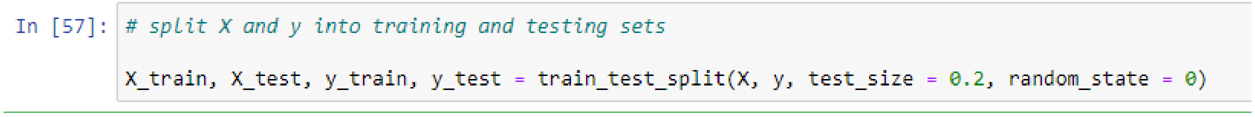
VariableScaled is the scaled value of the feature within the specified range.

. Model Training



```
X = scaled_df.drop(['RainTomorrow'], axis=1)

y = scaled_df['RainTomorrow']
```

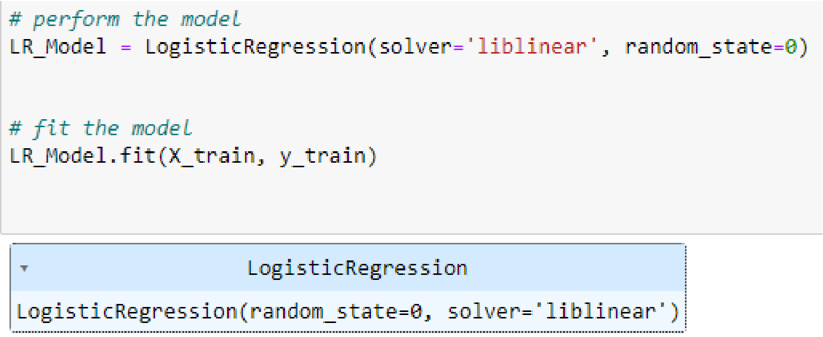
We possess two key variables: X, encompassing all features (independent variables) except 'RainTomorrow', and y, comprising the target variable 'RainTomorrow'. Such a setup is standard in supervised machine learning tasks, where a dataset includes features alongside a target variable, and the objective is to train a model capable of predicting the target variable based on the features provided.



```
In [57]: # split X and y into training and testing sets

         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

Here, I partitioned the dataset into training and testing sets, encompassing both features and the target variable. This division enables the evaluation of the machine learning model's performance on unseen data.

I implemented a test size of 20% of the entire dataset. This ensures that the model is trained on 80% of the data, leaving 20% as unseen data for testing the model's performance.

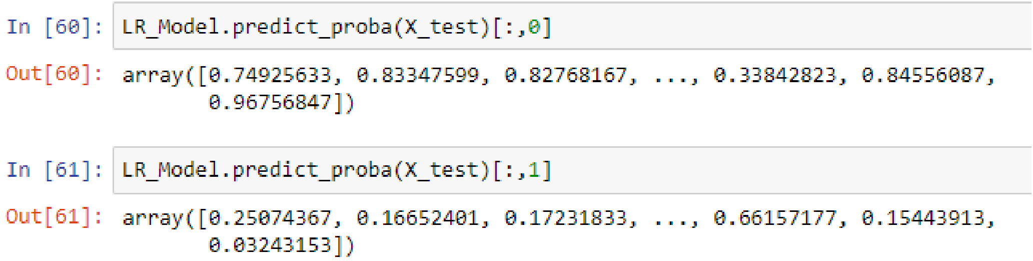


```
# perform the model
LR_Model = LogisticRegression(solver='liblinear', random_state=0)


# fit the model
LR_Model.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(random_state=0, solver='liblinear')
```

Here, I conducted the model fitting process on my training set. I opted for Logistic Regression, a regression analysis method utilized for predicting the probability of a binary outcome. I specifically selected the 'liblinear' solver from scikit-learn due to its suitability for small datasets and efficiency in binary classification tasks. Additionally, I set the random seed to 0 to ensure reproducibility of results. This parameter guarantees consistency in outcomes across multiple runs of the code, facilitating debugging and result sharing.

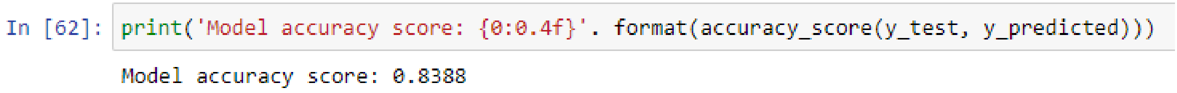
. Predict results



```
In [60]: LR_Model.predict_proba(X_test)[:,0]

Out[60]: array([0.74925633, 0.83347599, 0.82768167, ..., 0.33842823, 0.84556087,
                0.96756847])

In [61]: LR_Model.predict_proba(X_test)[:,1]

Out[61]: array([0.25074367, 0.16652401, 0.17231833, ..., 0.66157177, 0.15443913,
                0.03243153])
```

Our model demonstrates excellent performance. The outcome is presented in an array format, where 0 represents the probability of no rain and 1 represents the probability of rain.

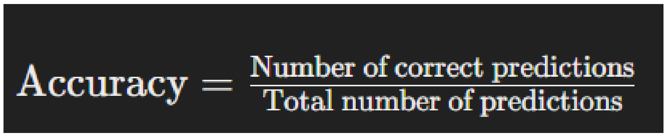
. Accuracy



```
In [62]: print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_predicted)))

         Model accuracy score: 0.8388
```

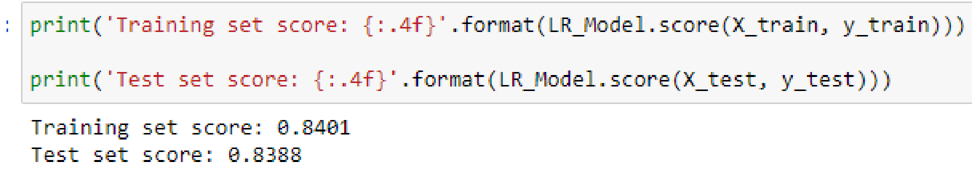
We observe an accuracy of approximately 83%, which signifies a highly favorable result. This indicates that our model performs exceptionally well, correctly predicting outcomes for 83% of our unseen data.

In our evaluation, y_test denotes the true labels (our unseen data), while y_predicted represents the predicted labels. The function compares these two sets of labels and computes the accuracy score using the formula outlined below.



$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

. Overfitting and Underfitting



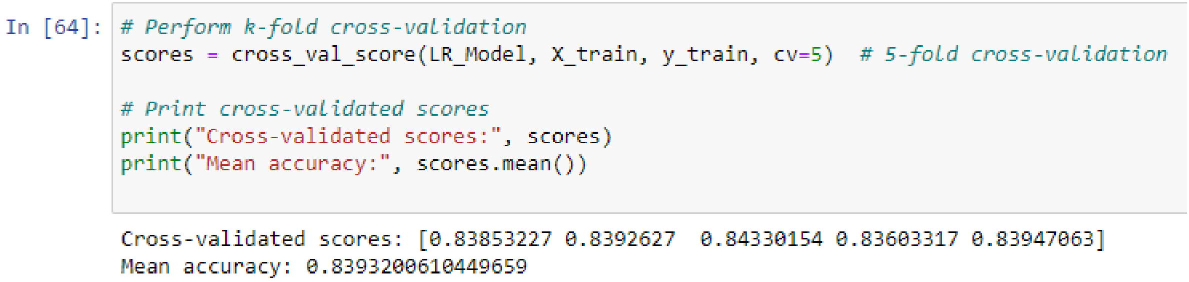
```
:  print('Training set score: {:.4f}'.format(LR_Model.score(X_train, y_train)))

   print('Test set score: {:.4f}'.format(LR_Model.score(X_test, y_test)))

   Training set score: 0.8401
   Test set score: 0.8388
```

The training-set accuracy score is 0.8401, while the test-set accuracy is 0.8388. These two values are highly comparable, indicating no signs of overfitting.

For instance, if the training set accuracy were considerably higher than the test-set accuracy, it would suggest overfitting. Overfitting occurs when the model learns too much from the training data, including noise, and fails to generalize well to unseen data.

Conversely, underfitting occurs when the model is too simplistic or inadequately trained, leading to its failure to capture patterns in the data and resulting in poor overall performance.

. K-Fold Cross validation



```
In [64]: # Perform k-fold cross-validation
         scores = cross_val_score(LR_Model, X_train, y_train, cv=5)  # 5-fold cross-validation

         # Print cross-validated scores
         print("Cross-validated scores:", scores)
         print("Mean accuracy:", scores.mean())

         Cross-validated scores: [0.83853227 0.8392627  0.84330154 0.83603317 0.83947063]
         Mean accuracy: 0.8393200610449659
```

Here, we assign cv to 5, which divides our dataset into 5 parts (each comprising 20% of the data). In each iteration, one of these parts serves as the test set while the remaining parts form the training set. Calculating accuracy scores for these 5 partitions provides us with a comprehensive understanding and ultimately yields our final accuracy.

Cross-validation serves as a robust method for evaluating, tuning, and selecting models in machine learning. It enhances generalization and ensures more reliable predictions on unseen data, contributing to the overall robustness of our model.

. Confusion Matrix

A confusion matrix serves as a valuable tool for assessing the performance of a classification algorithm. It provides a concise overview of the model's performance and the types of errors it generates, presented in a tabular format.

There are four possible outcomes when evaluating a classification model's performance, each described below:

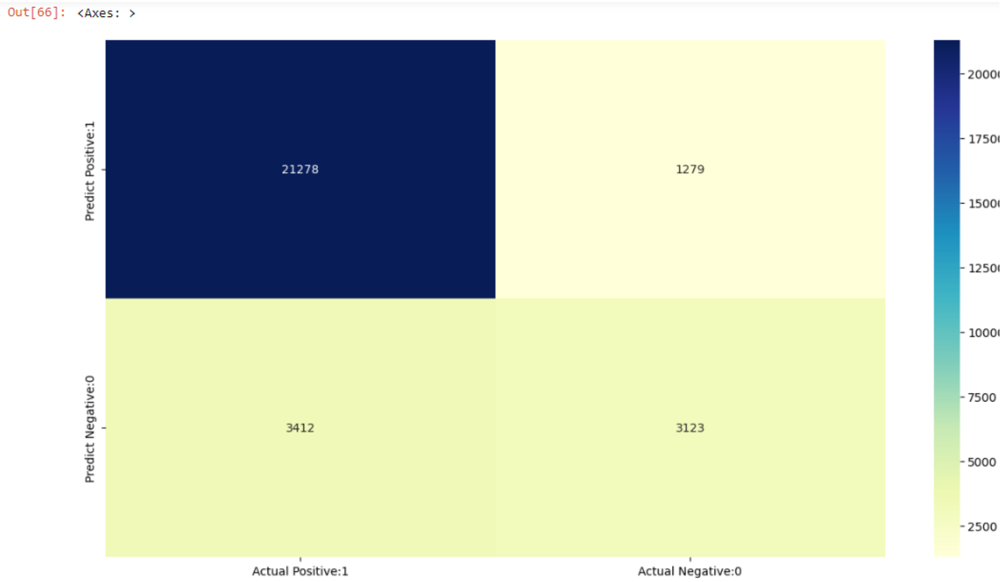
1. True Positives (TP): True Positives occur when the model correctly predicted positive instances.

2. True Negatives (TN): True Negatives occur when the model correctly predicted negative instances.

3. False Positives (FP): False Positives occur when the model predicted positive instances (false alarm). This type of error is known as Type I error.

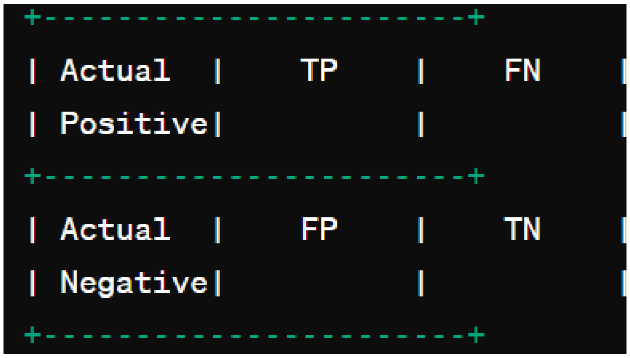
4. False Negatives (FN): False Negatives occur when the model incorrectly predicted negative instances (miss). This is a critical error known as Type II error.

These four outcomes are synthesized in the confusion matrix depicted below:



```
Out[66]: <Axes: >
```

	Actual Positive:1	Actual Negative:0
Predict Positive:1	21278	1279
Predict Negative:0	3412	3123



```
+-----------------------+
| Actual   |    TP    |    FN    |
| Positive|          |          |
+-----------------------+
| Actual   |    FP    |    TN    |
| Negative|          |          |
+-----------------------+
```

Another method for checking accuracy is:
(TP + TN) / (TP + TN + FP + FN)

Sum of TP and TN is all of our corrected predicted and sum of TP, TN, FP and FN is our total probability.

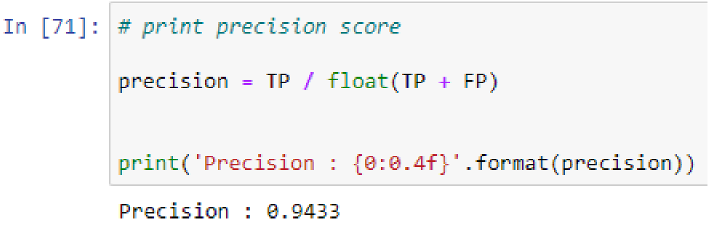
And If we want to check classification error, we can simply use this method:

(FP + FN) / (TP + TN + FP + FN)

Which sum of FP and FN is our total wrong outcome

. Classification Report

Precision:



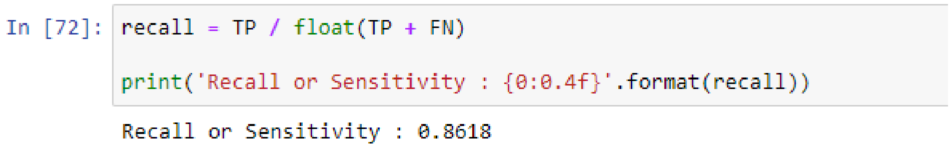
```
In [71]: # print precision score

         precision = TP / float(TP + FP)


         print('Precision : {0:0.4f}'.format(precision))

         Precision : 0.9433
```

Precision is defined as the percentage of correctly predicted positive outcomes out of all predicted positive outcomes. It can be calculated as the ratio of true positives (TP) to the sum of true positives and false positives (TP + FP).

In essence, precision highlights the proportion of correctly predicted positive outcomes and places more emphasis on the positive class than the negative class.

Mathematically, precision is expressed as the ratio of TP to (TP + FP).

Recall or Sensitivity:



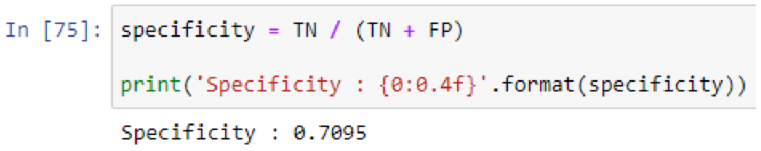
```
In [72]: recall = TP / float(TP + FN)

         print('Recall or Sensitivity : {0:0.4f}'.format(recall))

         Recall or Sensitivity : 0.8618
```

Recall, also known as Sensitivity, is defined as the percentage of correctly predicted positive outcomes out of all actual positive outcomes. It is calculated as the ratio of true positives (TP) to the sum of true positives and false negatives (TP + FN).

In essence, recall identifies the proportion of correctly predicted actual positives, thus providing insight into the model's ability to capture all positive instances.

Mathematically, recall is expressed as the ratio of TP to (TP + FN).

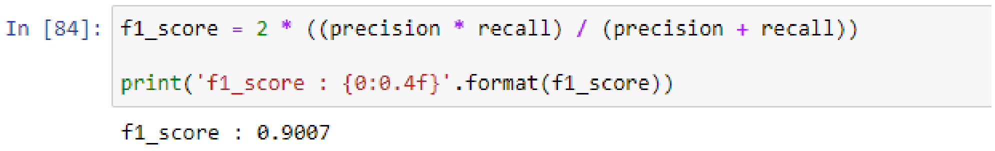
Specificity:



```
In [75]: specificity = TN / (TN + FP)

         print('Specificity : {0:0.4f}'.format(specificity))

         Specificity : 0.7095
```

Specificity, or the True Negative Rate, measures how well a binary classification model identifies negative outcomes correctly. It's calculated as the ratio of true negatives to the sum of true negatives and false positives. High specificity means the model distinguishes negative instances well, complementing sensitivity, which focuses on positive instances.

F1-Score:



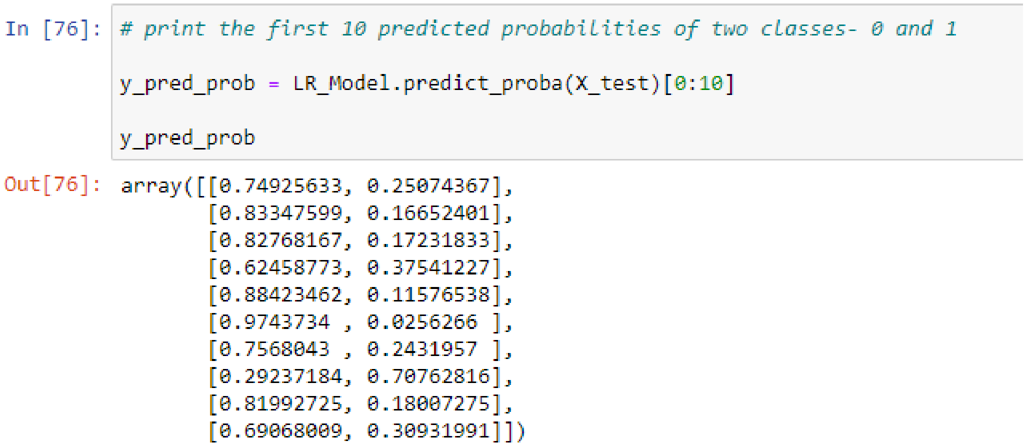
```
In [84]: f1_score = 2 * ((precision * recall) / (precision + recall))

         print('f1_score : {0:0.4f}'.format(f1_score))

         f1_score : 0.9007
```

The F1-score represents the weighted harmonic mean of precision and recall. A perfect F1-score is 1.0, while the worst possible score is 0.0. Since it accounts for both precision and recall, the F1-score typically falls below accuracy measures. It is essential to use the weighted average of F1-scores to compare classifier models rather than relying solely on global accuracy.

Support:

The "support" represents the number of occurrences for each class in the dataset. It is displayed in classification reports or confusion matrices, providing insights into class distribution and dataset imbalances. This value helps evaluate the reliability of performance metrics such as precision, recall, and F1-score, particularly in imbalanced datasets where some classes have fewer samples.

. Probabilities



```
In [76]: # print the first 10 predicted probabilities of two classes- 0 and 1

         y_pred_prob = LR_Model.predict_proba(X_test)[0:10]

         y_pred_prob

Out[76]: array([[0.74925633, 0.25074367],
                [0.83347599, 0.16652401],
                [0.82768167, 0.17231833],
                [0.62458773, 0.37541227],
                [0.88423462, 0.11576538],
                [0.9743734 , 0.0256266 ],
                [0.7568043 , 0.2431957 ],
                [0.29237184, 0.70762816],
                [0.81992725, 0.18007275],
                [0.69068009, 0.30931991]])
```

Observations:

Each row's numbers sum to 1.

There are 2 columns corresponding to 2 classes: 0 and 1.

Class 0 - predicted probability of no rain tomorrow.

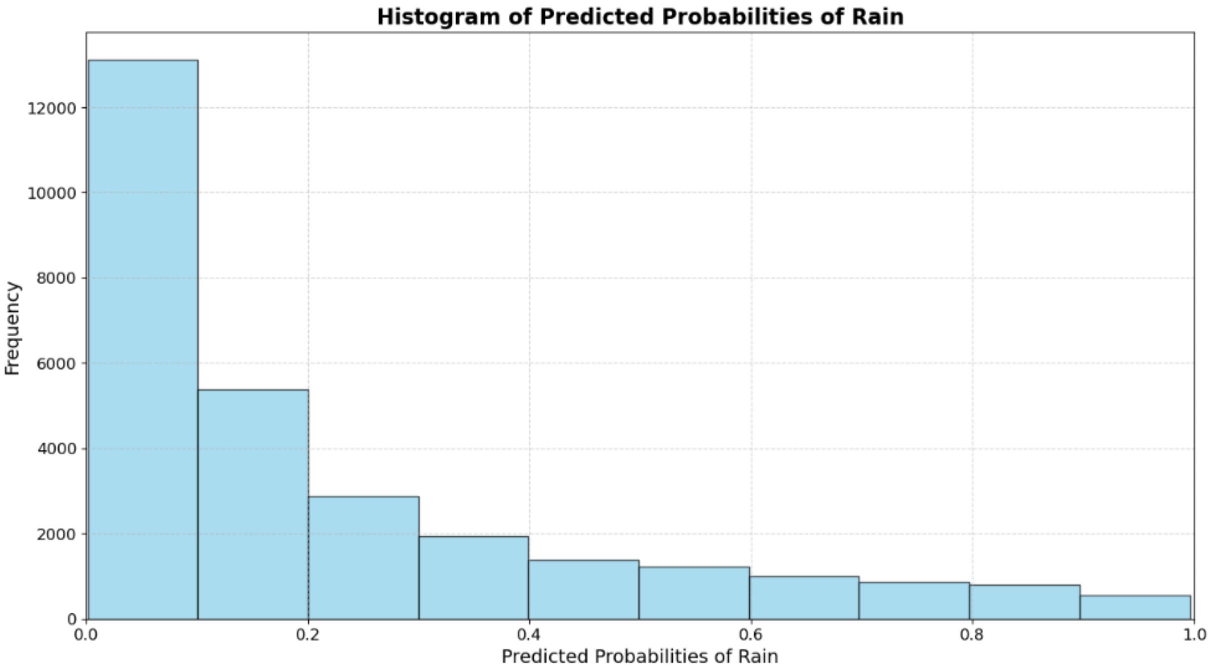
Class 1 - predicted probability of rain tomorrow.

Classification Threshold Level:

A classification threshold level of 0.5 is set.

- Class 1 - rain is predicted if the probability > 0.5.

- Class 0 - no rain is predicted if the probability < 0.5.



Observations:

The histogram above indicates a highly positively skewed distribution.

The first column shows approximately 15,000 observations with probabilities between 0.0 and 0.1.

There are only a few observations with probabilities greater than 0.5, suggesting rain tomorrow.

The majority of observations predict no rain tomorrow.

. ROC – AUC

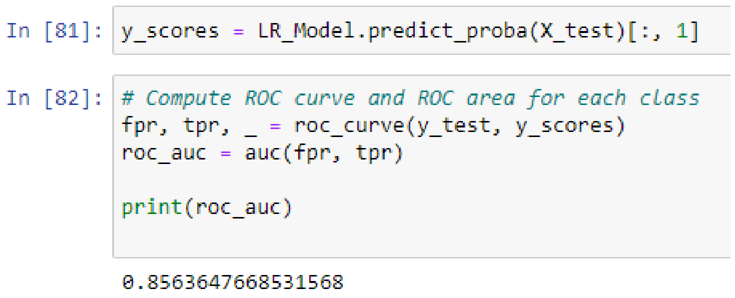
Another valuable tool for visually evaluating classification model performance is the ROC Curve, short for Receiver Operating Characteristic Curve. It plots the performance of a classification model across various classification threshold levels.

The ROC Curve illustrates the True Positive Rate (TPR) against the False Positive Rate (FPR) at different threshold levels.

The True Positive Rate (TPR), also known as Recall, is defined as the ratio of TP to (TP + FN).

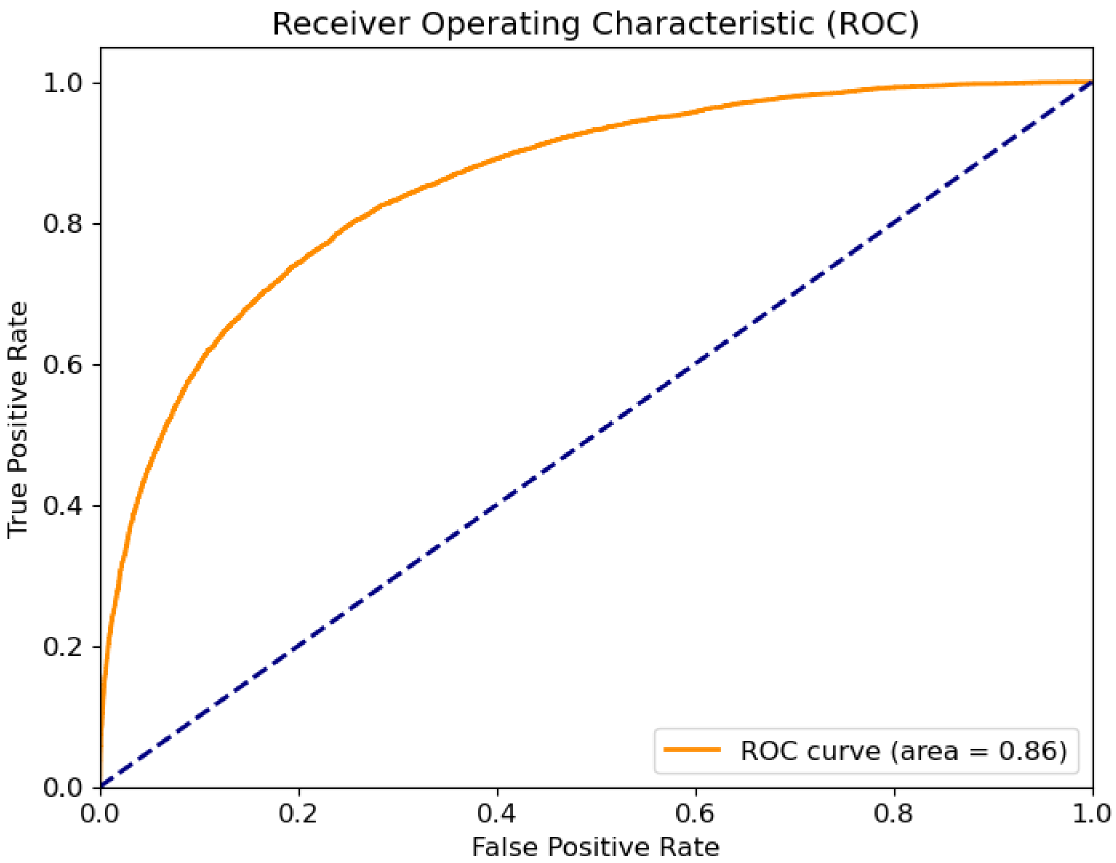
The False Positive Rate (FPR) is defined as the ratio of FP to (FP + TN).

When examining the ROC Curve, we focus on the TPR (True Positive Rate) and FPR (False Positive Rate) of a single point, offering insight into the overall performance across various threshold levels. Lowering the threshold levels may result in more items being classified as positive, leading to increased True Positives (TP) and False Positives (FP).



```
In [81]: y_scores = LR_Model.predict_proba(X_test)[:, 1]

In [82]: # Compute ROC curve and ROC area for each class
         fpr, tpr, _ = roc_curve(y_test, y_scores)
         roc_auc = auc(fpr, tpr)

         print(roc_auc)

         0.8563647668531568
```

Our AUC score is 0.85 (the area on the curve in the next photo).



The ROC AUC is a single-number summary of classifier performance. A higher value indicates better performance by the classifier.

Our model's ROC AUC approaches 1, suggesting that our classifier effectively predicts whether it will rain tomorrow.

The AUC score calculated from the ROC curve is printed here. It is represented as a floating-point number indicating the area under the ROC curve, which measures the model's ability to distinguish between classes. A higher AUC signifies better performance by the model.

. conclusion

1. With an accuracy score of 0.85, the logistic regression model demonstrates commendable performance in forecasting tomorrow's rainfall in Australia.

2. A limited subset of observations indicates the likelihood of rain tomorrow, while the prevailing majority suggests otherwise.

3. There are no discernible indications of overfitting within the model.

4. Elevating the threshold level yields a corresponding enhancement in accuracy.

5. The ROC AUC of our model converges towards 1, affirming the effectiveness of our classifier in forecasting tomorrow's rainfall occurrence.

. Resources & Libraries

Observations were drawn from numerous weather stations. The daily observations are available from http://www.bom.gov.au/climate/data.

DataSet from https://www.kaggle.com/datasets/sphyg/weather-dataset-rattle-package/data

An example of latest weather observations in Canberra:
http://www.bom.gov.au/climate/dwo/IDCJDW2801.latest.shtml

Definitions adapted from http://www.bom.gov.au/climate/dwo/IDCJDW0000.shtml

Data source: http://www.bom.gov.au/climate/dwo/ and
http://www.bom.gov.u/climate/data.

Libraries:

1. Pandas: `import pandas as pd` - Used for data manipulation and analysis.

2. NumPy: `import numpy as np` - Used for numerical computing.

3. Matplotlib: `import matplotlib.pyplot as plt` - Used for data visualization.

4. Seaborn: `import seaborn as sns` - Used for statistical data visualization.

5. Plotly Express: `import plotly.express as px` - Used for interactive data visualization.

6. Category Encoders: `import category_encoders as ce` - Used for encoding categorical variables.

7. Scikit-learn:

   - MinMaxScaler: `from sklearn.preprocessing import MinMaxScaler` - Used for feature scaling.

   - Train Test Split: `from sklearn.model_selection import train_test_split` - Used for splitting data into training and testing sets.

   - Logistic Regression: `from sklearn.linear_model import LogisticRegression` - Used for logistic regression modeling.

   - Cross-Validation: `from sklearn.model_selection import cross_val_score` - Used for cross-validation of models.

   - Accuracy Score, Confusion Matrix, Classification Report: `from sklearn.metrics import accuracy_score, confusion_matrix, classification_report` - Used for model evaluation.

   - ROC Curve, AUC Score: `from sklearn.metrics import roc_curve, auc` - Used for evaluating the performance of classification models.

8. Warnings: `import warnings` - Used to handle or ignore warning messages during execution.