An end-to-end Machine Learning model built for Rainfall Prediction-Weather Forecasting for the individuals new to this field of Data Science

"Kindly spare me if the article is lengthy, I have tried covering all the requisite and relevant information that an individual new to this field will require"

This article is a synopsis of and end-to-end Rain Prediction-Weather Forecasting ML project and will be a guiding light for individuals new to the field of data science. Personally, the initial phase of my journey into the field of data science was somewhat disoriented. There is a enormous amount of information present on the world wide web. This information being available in odds and ends would leave one disoriented. All of us need that one bright and shiny guiding light that will pilot us to that polestar, to our destination. There are countless projects already available on the web, the only problem being, they are pretty advanced and will terrify an individual. This in turn would make that individual loose interest in this flourishing field of data science. I would like to contribute my bit of work with the hope that it would help an individual to face Data Science and ML head on rather than running away from it, petrified.

This article contains a detailed synopsis of the Rain Prediction-Weather Forecasting model. I will be further covering all the requisites through:

1. Problem Definition
2. Data Analysis
3. EDA
4. Pre-processing Pipeline
5. Building Machine Learning Models
6. Concluding Remarks

1. Problem Statement

The foremost thing before building a Model is to understand the problem statement. One cannot afford to misinterpret a problem statement as the outcome would be a completely futile model.

I have used the rain prediction-weather forecast model which is formed using the rain in Australia Dataset. This dataset contains daily weather observation from various weather stations in Australia. The dataset for rain in Australia can be found here . The rain in Australia data set contains the daily weather observations namely:

- 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.
- WindGustDi r- 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.
- Cloud3pm -Fraction of sky obscured by cloud
- Temp9am-Temperature (degrees C) at 9am
- Temp3pm -Temperature (degrees C) at 3pm

- Rain Today -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. A kind of measure of the "risk"

Weather forecasting or rainfall forecasting, specifically, has been around for donkey years. Specifically, rainfall forecasting can not only be beneficial in anticipating floods but also in the draught condition as well. Forecasting how much it will rain tomorrow or how much Rainfall is there can extensively help in the agricultural field as well. I have applied different machine learning algorithms to predict whether or not it will rain tomorrow.

In this project we will be forecasting, whether or not it will rain tomorrow on the basis of the features provided. We can catch sight of the fact that the data for anticipating whether or not it will rain tomorrow will hold binary natured data (Yes or No), we will apply different classification algorithms to tackle this.

First and foremost, lets us import all the basic libraries required for building our model. I will be importing libraries as and when required.

```
import warnings
warnings.simplefilter("ignore")
warnings.filterwarnings("ignore")
import joblib

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Loading our raw data directly from GitHub

```
df=pd.read_csv('https://raw.githubusercontent.com/dsrscientist/dataset3/main/weatherAUS.csv')
df
```

2.Data Analysis

We can glance at our data frame and try to make some sense of the columns. Acknowledge the type of data present in different columns. Identify the features and target, the type of data they contain.

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity9am	Humidity3pm	Press
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	...	71.0	22.0	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	44.0	25.0	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	...	38.0	30.0	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	...	45.0	16.0	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	...	82.0	33.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
8420	2017-06-21	Uluru	2.8	23.4	0.0	NaN	NaN	E	31.0	SE	...	51.0	24.0	
8421	2017-06-22	Uluru	3.6	25.3	0.0	NaN	NaN	NNW	22.0	SE	...	56.0	21.0	
8422	2017-06-23	Uluru	5.4	26.9	0.0	NaN	NaN	N	37.0	SE	...	53.0	24.0	
8423	2017-06-24	Uluru	7.8	27.0	0.0	NaN	NaN	SE	28.0	SSE	...	51.0	24.0	
8424	2017-06-25	Uluru	14.9	NaN	0.0	NaN	NaN	NaN	NaN	ESE	...	62.0	36.0	

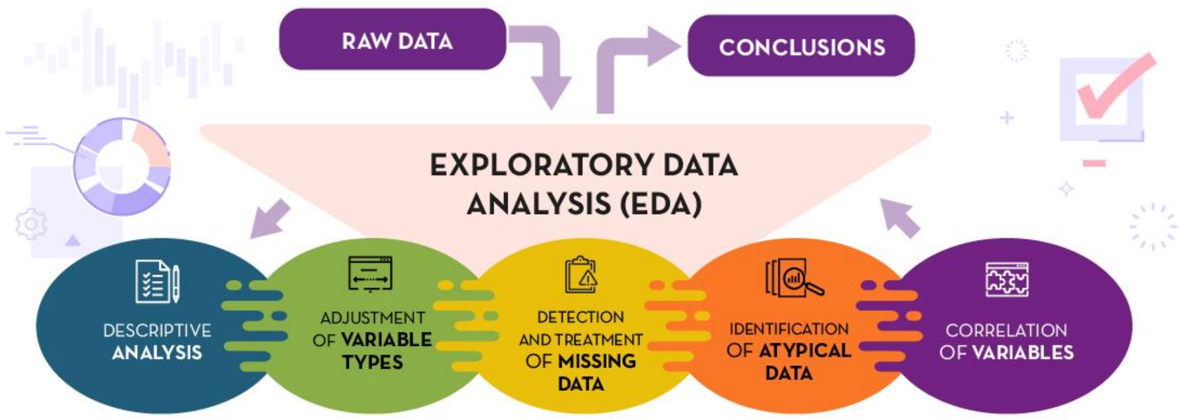
8425 rows × 23 columns

We have 8425 rows and 23 columns in our data frame. We can clearly see that some columns contain NaN values, which we will look into in detail while doing the EDA. We have a large data set considering 8425 rows times 23 columns. Since the dataset is large the visualization gets trimmed.

Since we have 23 columns, we can use the display.max function to display all the columns as well as the rows

```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

3.Exploratory Data Analysis (EDA)



Exploratory data analysis, commonly called as EDA is the most crucial step in building an accurate model. It is similar to an investigation, whether it be an investigation of a crime or a scrutiny leading to any particular unravelling of a mystery. Just as we scrutinize and use findings in order to get close to an unravelling, similar is the case of EDA. We scrutinize our data and manoeuvre these findings to build an accurate model.

After going through ample articles by expert Data Scientists I can clearly sum it up by saying EDA is an approach and not a technique, it is an approach on how data analysis should be carried out. The various steps like descriptive analysis, adjustment of variable types, detecting and treating missing data, finding anomalies and correlation of variables are all approaches that aid in building an accurate model.

Another thing to note is that doing an EDA will get you familiarized with the data, help you understand the data and provide you with insights that will help you to attain that accuracy in your model.

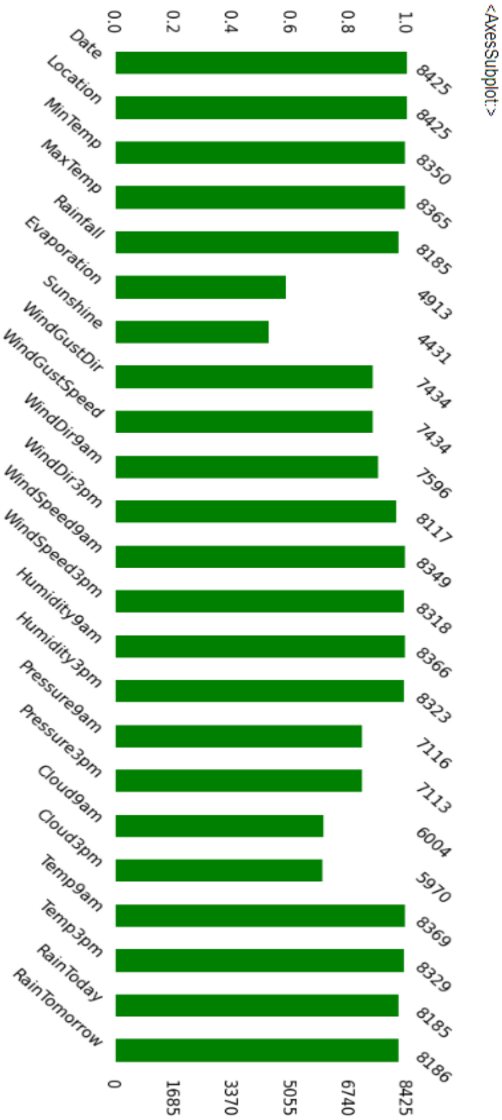
I can go on and on about EDA but wouldn't want you to get bored down with just EDA. Let's move on to the code

The first thing I want to do is to see if there are any null values present in our dataset using the df.isnull().sum() and visually using the missingno.bar

```
df.isnull().sum()
```

```
import missingno
missingno.bar(df, figsize = (20,5), color="green")
```

```
Date              0
Location          0
MinTemp          75
MaxTemp          60
Rainfall        240
Evaporation    3512
Sunshine       3994
WindGustDir     991
WindGustSpeed   991
WindDir9am      829
WindDir3pm      308
WindSpeed9am     76
WindSpeed3pm    107
Humidity9am      59
Humidity3pm     102
Pressure9am    1309
Pressure3pm    1312
Cloud9am       2421
Cloud3pm       2455
Temp9am          56
Temp3pm          96
RainToday       240
RainTomorrow    239
dtype: int64
```



We can see there are a lot of null values present in some columns, which will have to be taken care. The columns evaporation and sunshine had almost half of the rows filled with null values, it would not make any sense to fill half of them, therefore I have dropped them.

```
df.drop(['Evaporation','Sunshine'],axis=1,inplace=True)
```

Now, let us describe our data set using the describe function to check out the count value, mean data, standard deviation info and the 25%,50%,75% quartile details and minimum and maximum values. The describe function relates only to the numerical data, the object data type is ignored. Let us have a look at the code to understand better.

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
MinTemp	8350.0	13.193305	5.403596	-2.0	9.2	13.3	17.4	28.5
MaxTemp	8365.0	23.859976	6.136408	8.2	19.3	23.3	28.0	45.5
Rainfall	8185.0	2.805913	10.459379	0.0	0.0	0.0	1.0	371.0
WindGustSpeed	7434.0	40.174469	14.665721	7.0	30.0	39.0	50.0	107.0
WindSpeed9am	8349.0	13.847646	10.174579	0.0	6.0	13.0	20.0	63.0
WindSpeed3pm	8318.0	18.533662	9.766986	0.0	11.0	19.0	24.0	83.0
Humidity9am	8366.0	67.822496	16.833283	10.0	56.0	68.0	80.0	100.0
Humidity3pm	8323.0	51.249790	18.423774	6.0	39.0	51.0	63.0	99.0
Pressure9am	7116.0	1017.640233	6.828699	989.8	1013.0	1017.7	1022.3	1039.0
Pressure3pm	7113.0	1015.236075	6.766681	982.9	1010.4	1015.3	1019.8	1036.0
Cloud9am	6004.0	4.566622	2.877658	0.0	1.0	5.0	7.0	8.0
Cloud3pm	5970.0	4.503183	2.731659	0.0	2.0	5.0	7.0	8.0
Temp9am	8369.0	17.762015	5.627035	1.9	13.8	17.8	21.9	39.4
Temp3pm	8329.0	22.442934	5.980020	7.3	18.0	21.9	26.4	44.1

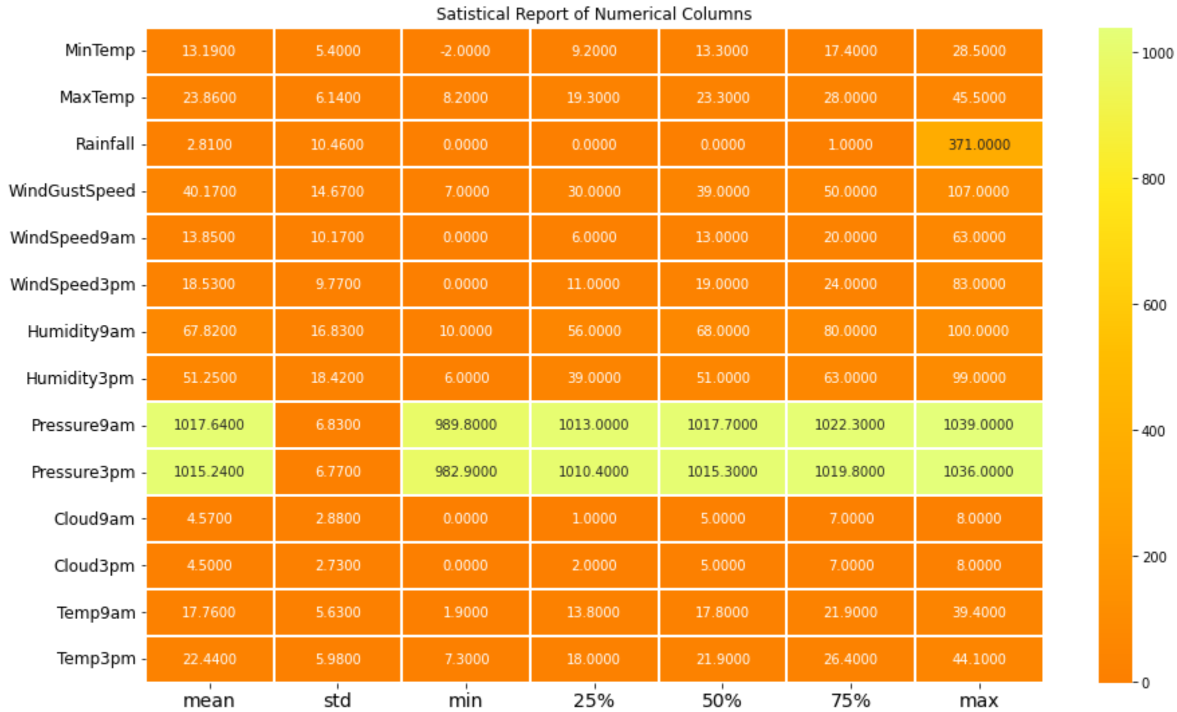
We observe that the mean value of the columns is almost equal to or slightly less than the median, 50% quartile

We can confirm again with the count values that null values are present

We see that some of the columns have the min value as 0 present in them

Let us describe the data frame visually also. The human brain is programmed in such a way that we learn and understand things better when we can visually see them

```
plt.figure(figsize = (15,9))
sns.heatmap(round(df.describe()[1:].transpose(),2), linewidth = 2, annot= True, fmt = ".4f", cmap="Wistia_r")
plt.title("Satistical Report of Numerical Columns")
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 12)
plt.show()
```



The pressure9am and pressure3pm columns stand out because of their high values

After taking insights from the description of the data frame let us move onto checking the data types. Whether the columns are of object datatype or float or int.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6762 entries, 0 to 8424
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Date           6762 non-null   object
 1   Location       6762 non-null   object
 2   MinTemp        6692 non-null   float64
 3   MaxTemp        6705 non-null   float64
 4   Rainfall       6624 non-null   float64
 5   WindGustDir    5820 non-null   object
 6   WindGustSpeed  5820 non-null   float64
 7   WindDir9am     5968 non-null   object
 8   WindDir3pm     6468 non-null   object
 9   WindSpeed9am   6699 non-null   float64
 10  WindSpeed3pm   6662 non-null   float64
 11  Humidity9am    6708 non-null   float64
 12  Humidity3pm    6666 non-null   float64
 13  Pressure9am    5454 non-null   float64
 14  Pressure3pm    5451 non-null   float64
 15  Cloud9am       4896 non-null   float64
 16  Cloud3pm       4860 non-null   float64
 17  Temp9am        6711 non-null   float64
 18  Temp3pm        6670 non-null   float64
 19  RainToday      6624 non-null   object
 20  RainTomorrow   6624 non-null   object
dtypes: float64(14), object(7)
memory usage: 1.1+ MB
```

Some of the columns are of object datatype which will have to be converted to numerical datatype in order for the machine to understand it. We can confirm again that null values are present in some of the columns

Another technique that can help us in getting insights, about are independent variables or features, is checking for the unique values which can be done with the help of. nunique function

```
df.nunique().to_frame('Unique Values')
```

	Unique Values
Date	3004
Location	12
MinTemp	285
MaxTemp	331
Rainfall	250
WindGustDir	16
WindGustSpeed	52
WindDir9am	16
WindDir3pm	16
WindSpeed9am	34
WindSpeed3pm	35
Humidity9am	90
Humidity3pm	94
Pressure9am	384
Pressure3pm	374
Cloud9am	9
Cloud3pm	9
Temp9am	304
Temp3pm	328
RainToday	2
RainTomorrow	2

Coming to our target variable Rain Tomorrow we see that it has two unique values, yes or no and hence contains binary data and will be built using classification algorithms

Before we get to the visualization of our data let us fill the missing data or else our model will be biased towards certain columns.

We can fill the missing values either with the mean value or with the mode value depending on the type of data present. We fill the missing values with mean if the data is continuous in nature and with the mode if it is categorical. The fill.na function is used to fill the missing values

```
df['MinTemp']=df['MinTemp'].fillna(np.mean(df['MinTemp']))
df['MaxTemp']=df['MaxTemp'].fillna(np.mean(df['MaxTemp']))
df['Rainfall']=df['Rainfall'].fillna(np.mean(df['Rainfall']))
df['WindGustDir']=df['WindGustDir'].fillna(df['WindGustDir'].mode()[0])
df['WindGustSpeed']=df['WindGustSpeed'].fillna(np.mean(df['WindGustSpeed']))
df['WindDir9am']=df['WindDir9am'].fillna(df['WindDir9am'].mode()[0])
df['WindDir3pm']=df['WindDir3pm'].fillna(df['WindDir3pm'].mode()[0])
df['WindSpeed9am']=df['WindSpeed9am'].fillna(np.mean(df['WindSpeed9am']))
df['WindSpeed3pm']=df['WindSpeed3pm'].fillna(np.mean(df['WindSpeed3pm']))
df['Humidity9am']=df['Humidity9am'].fillna(np.mean(df['Humidity9am']))
df['Humidity3pm']=df['Humidity3pm'].fillna(np.mean(df['Humidity3pm']))
df['Pressure9am']=df['Pressure9am'].fillna(np.mean(df['Pressure9am']))
df['Pressure3pm']=df['Pressure3pm'].fillna(np.mean(df['Pressure3pm']))
df['Cloud9am']=df['Cloud9am'].fillna(np.mean(df['Cloud9am']))
df['Cloud3pm']=df['Cloud3pm'].fillna(np.mean(df['Cloud3pm']))
df['Temp9am']=df['Temp9am'].fillna(np.mean(df['Temp9am']))
df['Temp3pm']=df['Temp3pm'].fillna(np.mean(df['Temp3pm']))
df['RainToday']=df['RainToday'].fillna(df['RainToday'].mode()[0])
df['RainTomorrow']=df['RainTomorrow'].fillna(df['RainTomorrow'].mode()[0])
```

Let us cross check using the df.isnull().sum() function to check if the values have been filled or not

```
df.isnull().sum()
```

```
Date             0
Location         0
MinTemp          0
MaxTemp          0
Rainfall         0
WindGustDir      0
WindGustSpeed    0
WindDir9am       0
WindDir3pm       0
WindSpeed9am     0
WindSpeed3pm     0
Humidity9am      0
Humidity3pm      0
Pressure9am      0
Pressure3pm      0
Cloud9am         0
Cloud3pm         0
Temp9am          0
Temp3pm          0
RainToday        0
RainTomorrow     0
dtype: int64
```

Moving on, I will be separating the data column into separate day, month and year columns by applying lambda and then dropping the Date column.

```
df['Date']=pd.to_datetime(df['Date'])
df['Day']=df['Date'].apply(lambda x:x.day)
df['Month']=df['Date'].apply(lambda x:x.month)
df['Year']=df['Date'].apply(lambda x:x.year)
df.drop('Date', axis=1, inplace=True)
df.head()
```

Let us have a look at our data frame

dSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	Cloud9am	Cloud3pm	Temp9am	Temp3pm	RainToday	RainTomorrow	Day	Month	Year
24.0	71.0	22.0	1007.7	1007.1	8.000000	4.320988	16.9	21.8	No	No	1	12	2008
22.0	44.0	25.0	1010.6	1007.8	4.336806	4.320988	17.2	24.3	No	No	2	12	2008
26.0	38.0	30.0	1007.6	1008.7	4.336806	2.000000	21.0	23.2	No	No	3	12	2008
9.0	45.0	16.0	1017.6	1012.8	4.336806	4.320988	18.1	26.5	No	No	4	12	2008
20.0	82.0	33.0	1010.8	1006.0	7.000000	8.000000	17.8	29.7	No	No	5	12	2008

The number of columns also increase but that shouldn’t be of any trouble.

Moving on to a very cardinal step in EDA, visualization. The way in which one is able to visualize the data and gather insights tells a lot about how good one is at analysis. The analytical skills vary from person to person and there is no substandard analysis. In the end all that matters is what your insights are and how well you are able to put it across to the target audience.

I have made a list for all the columns

```
col= ['Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'WindGustDir',
     'WindGustSpeed', 'WindDir9am', 'WindDir3pm', 'WindSpeed9am',
     'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am',
     'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm',
     'RainToday', 'RainTomorrow', 'Day', 'Month', 'Year']
print(col)
```

```
['Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm', 'RainToday', 'RainTomorrow', 'Day', 'Month', 'Year']
```

I will be visualizing the data and will be using different plotting techniques for different data

Let us begin with most basic plot, count plot.

Count plot for categorical or object data type columns.

I have written a function named count plot for the style and specifications of the count plot. I have then run the object data type list through a for loop.

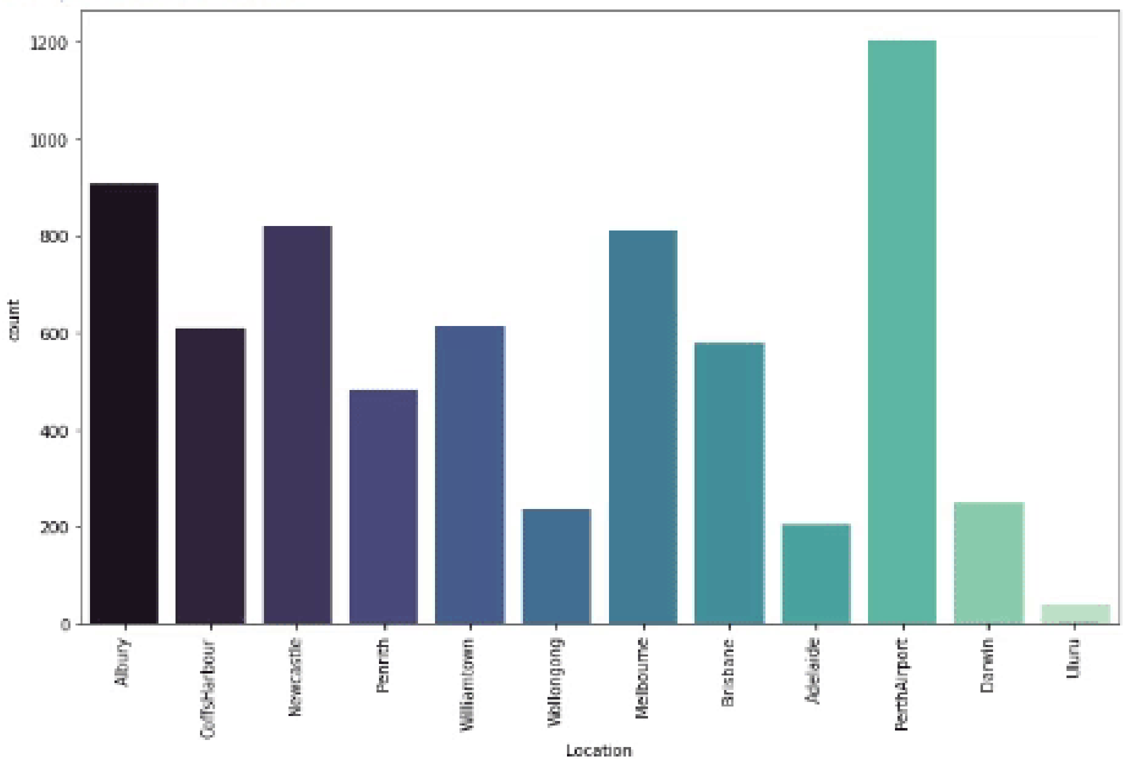
```
def count_plot(x):
    plt.figure(figsize=(10,7))
    sns.countplot(x,palette='mako')
    plt.xticks(rotation=90)
    plt.tight_layout()
    return plt.show()

obj_col = ['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday', 'RainTomorrow']

for i in df[obj_col]:
    print("Countplot for {} column:->".format(i))
    count_plot(df[i])
```

Let us check the output

Countplot for Location column:->



Scatter Plot for continuous columns

Here I have made a list, using the col list that I had made earlier, and stored it with columns that are not in the obj_col

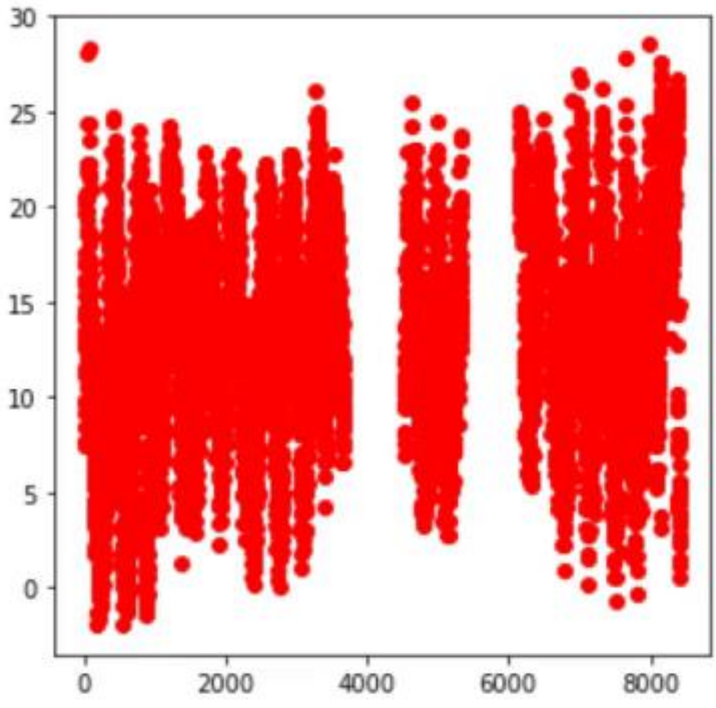
```
num_col = [x for x in col if x not in obj_col]
print(num_col)
```

```
['MinTemp', 'MaxTemp', 'Rainfall', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm', 'Day', 'Month', 'Year']
```

```
for j in df[num_col]:
    plt.figure(figsize=(5,5))
    print(f"Scatter plot for {j} column with respect to the rows covered ->")
    plt.scatter(df.index, df[j],color='red')
    plt.show()
```

Let us look at the outputs now

Scatter plot for MinTemp column with respect to the rows covered ->



Line plot for all the features with the target variable (RainTomorrow)

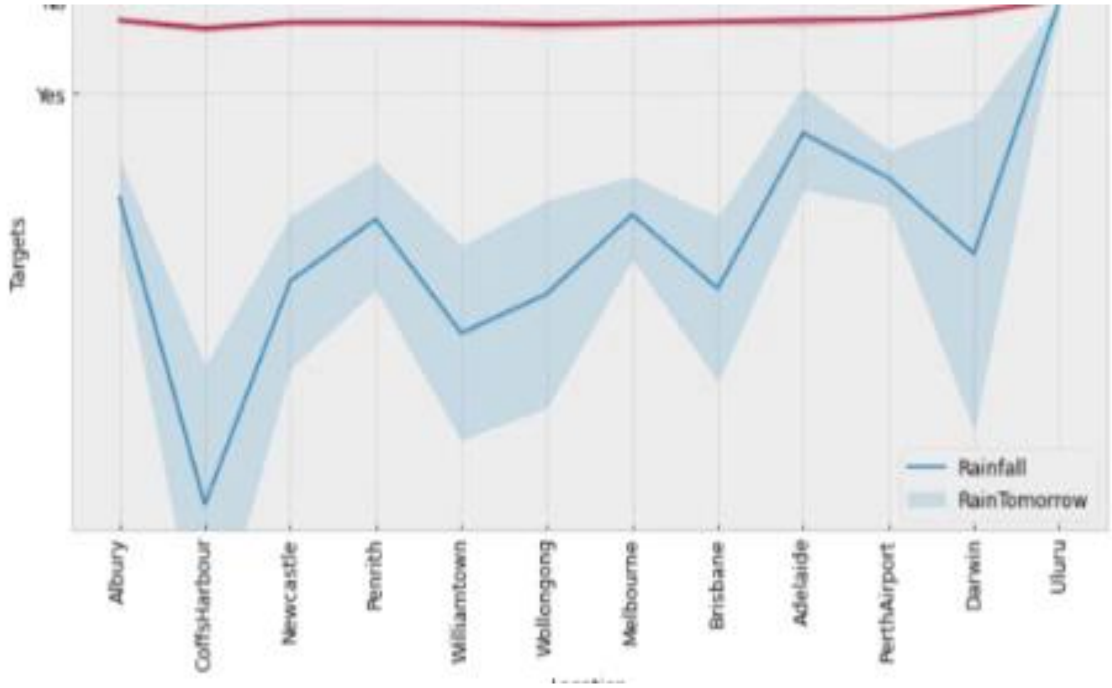
```
feature_columns = ['Location', 'MinTemp', 'MaxTemp', 'WindGustDir',
       'WindGustSpeed', 'WindDir9am', 'WindDir3pm', 'WindSpeed9am',
       'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am',
       'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm',
       'RainToday', 'Day', 'Month', 'Year']

target_columns = ['Rainfall', 'RainTomorrow']

plt.style.use('bmh')

for z in df[feature_columns]:
   plt.figure(figsize=(12,6))
   sns.lineplot(x=df[z], y=target_columns[0], data=df)
   sns.lineplot(x=df[z], y=target_columns[1], data=df)
   plt.ylabel("Targets")
   plt.xticks(rotation=90)
   plt.xticks(fontsize=12)
   plt.yticks(fontsize=12)
   plt.legend(['Rainfall', 'RainTomorrow'], fontsize=12)
   plt.show()
```

Let us have a look at our output and try to gather as much insights as possible



We have visualized our data and tried to gather as many insights as possible. We have had a look at the description of our data frame, identified and filled null values, visualized the data. One important thing that we gathered from visualization is that our classification target variable 'RainTomorrow' data is imbalanced, which will be taken care of later in pre-processing using SMOTE. These are some of the challenges that one faces and needs to take care of before even thinking about building a ML model. There is this very common saying, practice makes one perfect and the more one practices, the more hands-on experience you have, the easier it will be to overcome these challenges.

I have tried my best to cover everything with the visualization and any suggestions on anything I have missed out on or could have used a different approach, are most welcome.

4.__Pre-processing Data__

In the pre-processing step I am going to tackle all the miss fits and fix them one by one starting with encoding the object datatype values so that our Machine Learning models can understand the converted numeric values. I am making use of the encoding methods to convert all the object datatype values. For our label I am using Label Encoder while for other categorical feature columns I am using the Ordinal Encoder. Instead of the Ordinal Encoder I could have used the One Hot Encoder method but it would increase the number of columns, whereas applying Ordinal Encoder on data values would offer an order that deems to be a better option to me. I have seen quite a lot of people that use Label Encoding on the feature columns as well. It does not make any sense to me since the name itself suggests Label Encoder, and that it is use for encoding the label(s) columns. Hope my readers would keep this in mind and take it positively.

Code:

Label Encoding

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df["RainTomorrow"] = le.fit_transform(df["RainTomorrow"])
df.head()
```

Ordinal Encoding

```
from sklearn.preprocessing import OrdinalEncoder
oe=OrdinalEncoder()
for i in df.columns:
    if df[i].dtypes=='object':
        df[i]=oe.fit_transform(df[i].values.reshape(-1,1))
df
```

Output

	Location	MinTemp	MaxTemp	Rainfall	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidit
0	1.0	13.4	22.900000	0.6	13.0	44.000000	13.0	14.0	20.0	24.0	71.0	
1	1.0	7.4	25.100000	0.0	14.0	44.000000	6.0	15.0	4.0	22.0	44.0	
2	1.0	12.9	25.700000	0.0	15.0	46.000000	13.0	15.0	19.0	26.0	38.0	
3	1.0	9.2	28.000000	0.0	4.0	24.000000	9.0	0.0	11.0	9.0	45.0	
4	1.0	17.5	32.300000	1.0	13.0	41.000000	1.0	7.0	7.0	20.0	82.0	
...	...	...	...	...	...	...	...	...	...	...	...	
8420	9.0	2.8	23.400000	0.0	0.0	31.000000	9.0	1.0	13.0	11.0	51.0	
8421	9.0	3.6	25.300000	0.0	6.0	22.000000	9.0	3.0	13.0	9.0	56.0	
8422	9.0	5.4	26.900000	0.0	3.0	37.000000	9.0	14.0	9.0	9.0	53.0	
8423	9.0	7.8	27.000000	0.0	9.0	28.000000	10.0	3.0	13.0	7.0	51.0	
8424	9.0	14.9	23.859976	0.0	3.0	40.174469	2.0	2.0	17.0	17.0	62.0	

8425 rows × 23 columns

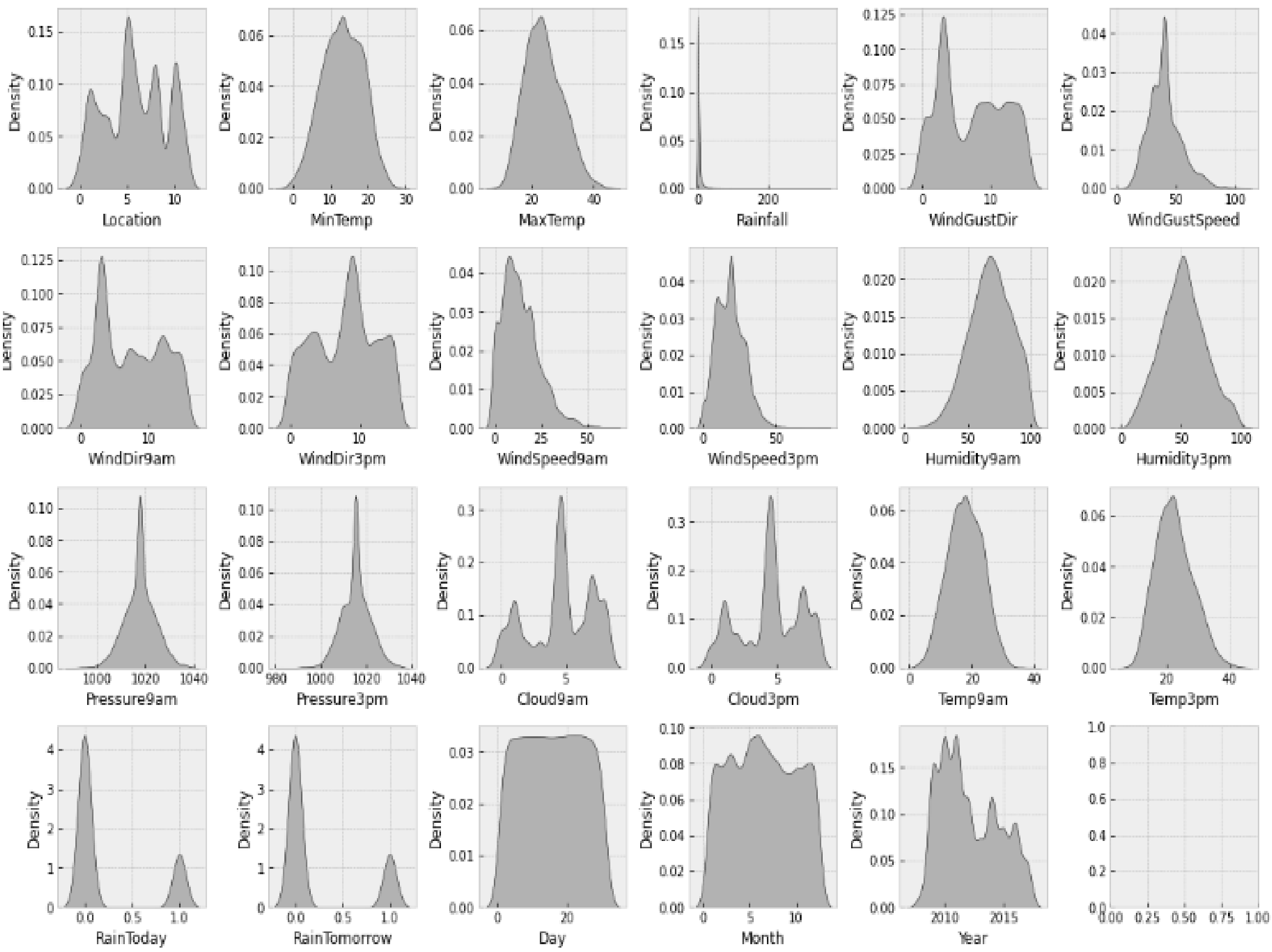
After encoding all our data, I will now plot the Normal Distribution curve: to check the skewness and Boxplot: to check for any outliers.

```
df.dtypes
```

```
Location         float64
MinTemp          float64
MaxTemp          float64
Rainfall         float64
WindGustDir      float64
WindGustSpeed    float64
WindDir9am       float64
WindDir3pm       float64
WindSpeed9am     float64
WindSpeed3pm     float64
Humidity9am      float64
Humidity3pm      float64
Pressure9am      float64
Pressure3pm      float64
Cloud9am         float64
Cloud3pm         float64
Temp9am          float64
Temp3pm          float64
RainToday        float64
RainTomorrow       int32
Day                int64
Month              int64
Year               int64
dtype: object
```

Normal Distribution Curve

With the shape of the normal distribution curve, we can make out whether or not positive or negative skewness is present in our data. A perfect bell curve indicates that no skewness is present



We can see that skewness is present in some of the columns. Let us use the df,skew function as well to check the values of skewness.

Code:

```
df.skew()
```

Output:

```
df.skew()
```

```
Location         -0.050456
MinTemp          -0.089989
MaxTemp           0.380654
Rainfall         13.218403
WindGustDir       0.119640
WindGustSpeed     0.757000
WindDir9am        0.172792
WindDir3pm       -0.119847
WindSpeed9am      0.960591
WindSpeed3pm      0.494217
Humidity9am      -0.256743
Humidity3pm       0.118281
Pressure9am      -0.024082
Pressure3pm      -0.010214
Cloud9am         -0.366503
Cloud3pm         -0.276294
Temp9am          -0.014748
Temp3pm           0.397331
RainToday         1.242362
RainTomorrow      1.241588
Day               0.004260
Month             0.039388
Year              0.418663
dtype: float64
```

I will be using Log transform to fix the skewness, as the square root method is used to fix moderate skewness.

Now let us plot a box plot to check any outliers

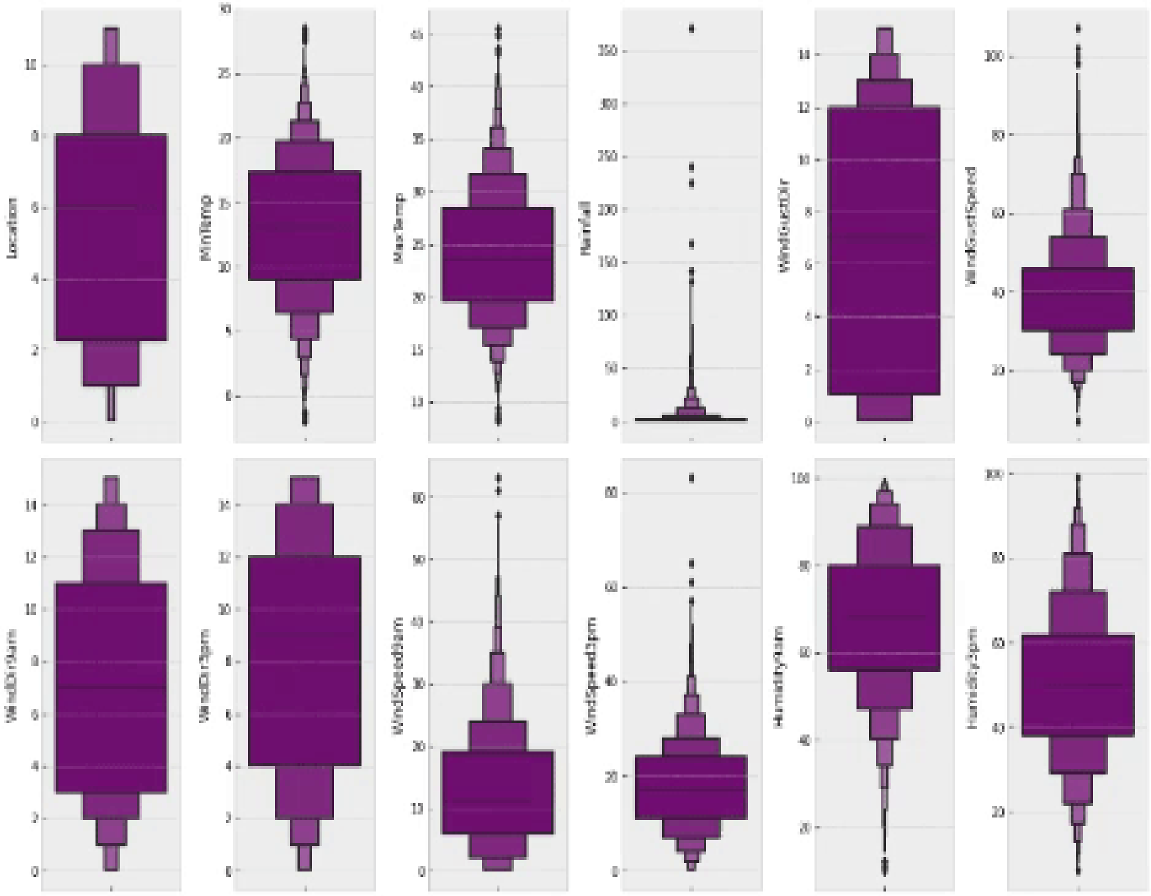
Outliers are the data points that lie far away from the outer quartiles. These outliers need to be removed or else they will bring down the accuracy of our model.

Code:

```
plt.style.use('fast')

fig, ax = plt.subplots(ncols=6, nrows=4, figsize=(15,20))
index = 0
ax = ax.flatten()
for col, value in df.items():
    sns.boxenplot(y=col, data=df, ax=ax[index], color="purple")
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.4, h_pad=1.0)
plt.show()
```

Output:



I will be using the z-score method to remove outliers. I can also use the IQR method but the data loss will be far greater and we want to retain as much data as possible

Let us remove the outliers now using z-score

```
df.shape
```

```
(8425, 23)
```

Code:

```
from scipy.stats import zscore
z=np.abs(zscore(df))
threshold=2.5
np.where(z>2.5)
```

Setting the value of threshold as 2.5

```
df_new_z=df[(z<2.5).all(axis=1)]
df_new_z
```

Output:

	Location	MinTemp	MaxTemp	Rainfall	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidit
0	1.0	13.4	22.900000	0.6	13.0	44.000000	13.0	14.0	20.0	24.0	71.0	
1	1.0	7.4	25.100000	0.0	14.0	44.000000	6.0	15.0	4.0	22.0	44.0	
2	1.0	12.9	25.700000	0.0	15.0	46.000000	13.0	15.0	19.0	26.0	38.0	
3	1.0	9.2	28.000000	0.0	4.0	24.000000	9.0	0.0	11.0	9.0	45.0	
4	1.0	17.5	32.300000	1.0	13.0	41.000000	1.0	7.0	7.0	20.0	82.0	
...	...	...	...	...	...	...	...	...	...	...	...	
8420	9.0	2.8	23.400000	0.0	0.0	31.000000	9.0	1.0	13.0	11.0	51.0	
8421	9.0	3.6	25.300000	0.0	6.0	22.000000	9.0	3.0	13.0	9.0	56.0	
8422	9.0	5.4	26.900000	0.0	3.0	37.000000	9.0	14.0	9.0	9.0	53.0	
8423	9.0	7.8	27.000000	0.0	9.0	28.000000	10.0	3.0	13.0	7.0	51.0	
8424	9.0	14.9	23.859976	0.0	3.0	40.174469	2.0	2.0	17.0	17.0	62.0	

7549 rows × 23 columns

Percentage Data Loss

```
Data_loss=((8425-7549)/8425)*100
print(Data_loss,'%')
```

10.397626112759644 %

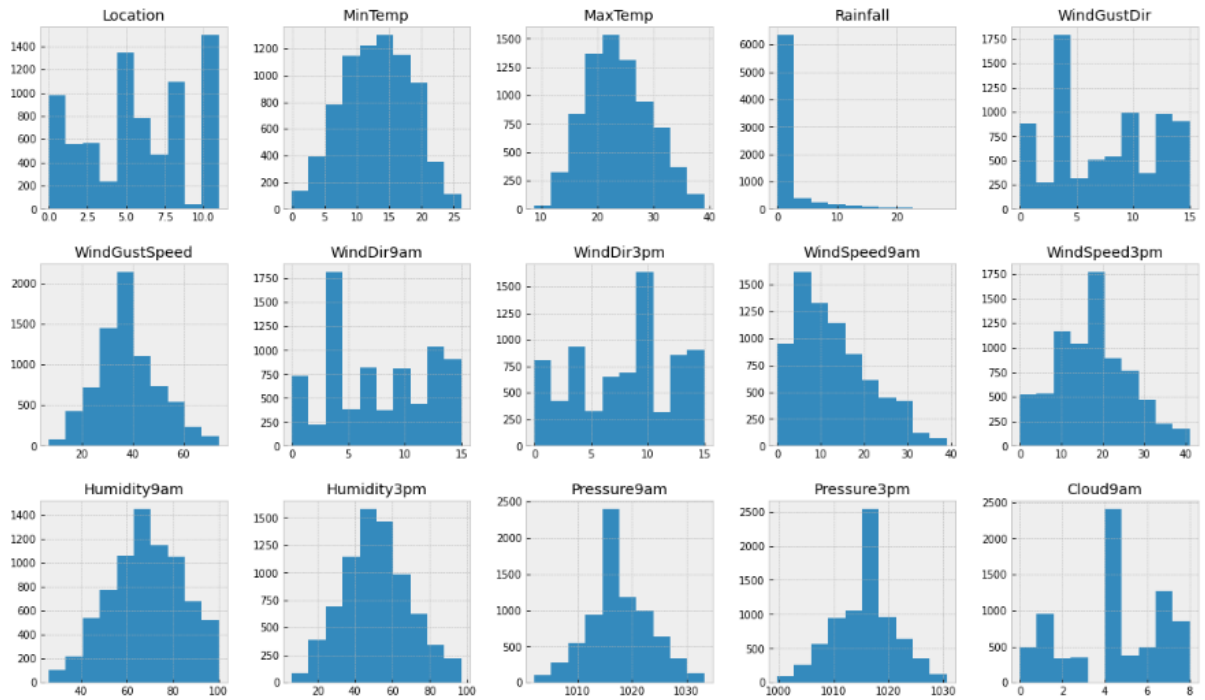
After applying Z score, I lost only about 10% of data but when I used the IQR method it took away almost 25% of the data and as a Data Scientist retaining valuable data and fixing it always takes priority rather than simply deleting it. Deleting it, is the last resort.

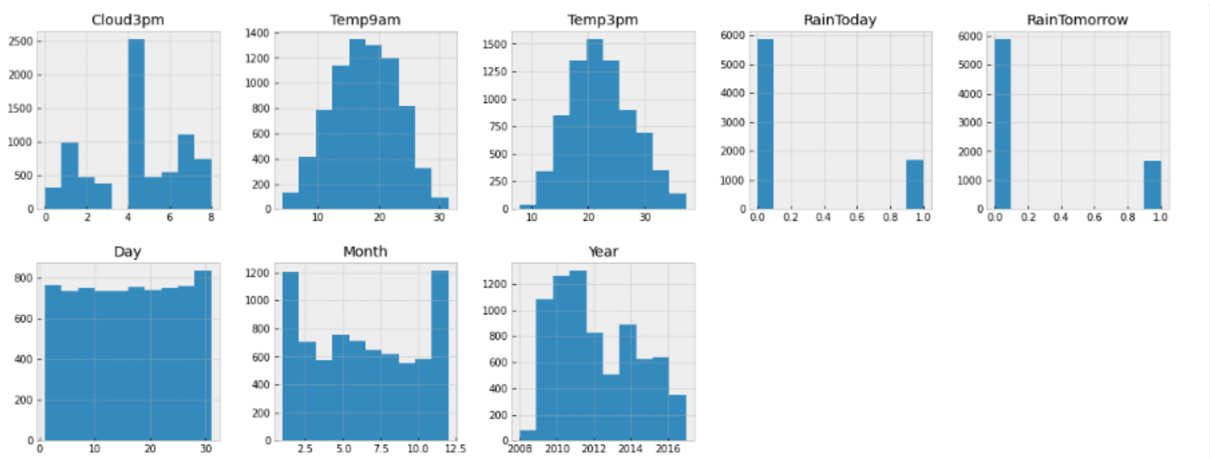
After this I am applying the Log transformation to deal with the skewness since the acceptable range lies between +/-0.5 value for each column.

Code:

```
for col in df_new_z:
    if df.skew().loc[col]>0.55:
        df[col]=np.log1p(df[col])
df_new_z.skew()
```

After removing the outliers and skewness I will now plot a histogram just to visually observe the changes in distribution.





Now let us move onto checking the correlation between our target variables and feature variables

Code:

correlation with target column RainTomorrow

```
df.corr()['RainTomorrow'].sort_values()
```

The sort_values function will sort the values in ascending order

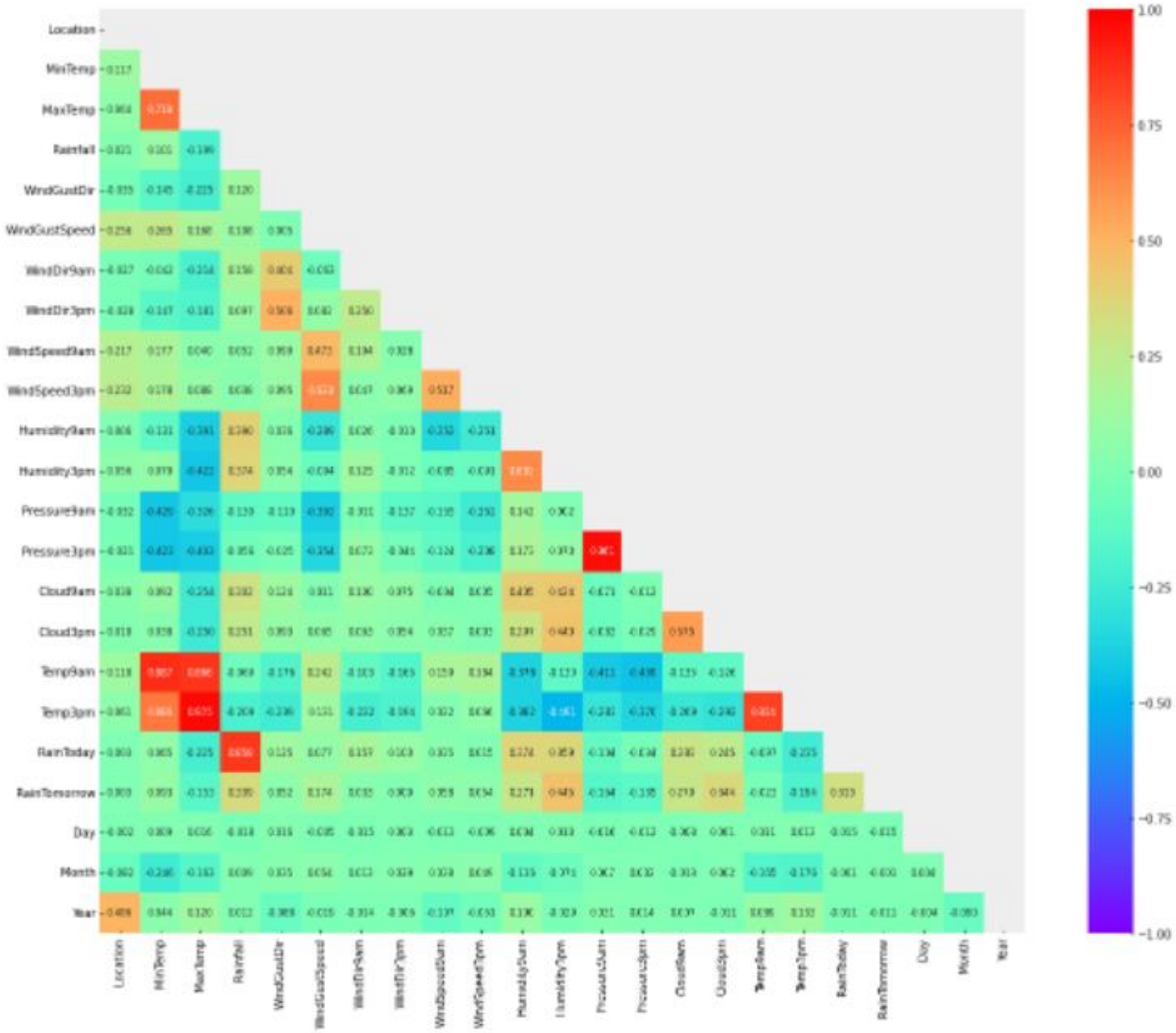
Output:

```
df.corr()['Rainfall'].sort_values()
```

```
Temp3pm         -0.208655
MaxTemp         -0.199356
Pressure9am     -0.129640
Temp9am         -0.068863
Pressure3pm     -0.055686
Day             -0.017810
Month            0.009203
Year             0.011853
Location         0.021456
WindSpeed3pm     0.037920
WindSpeed9am     0.051522
WindDir3pm       0.097174
MinTemp          0.100825
WindGustSpeed    0.107754
WindGustDir      0.119641
WindDir9am       0.157672
Cloud3pm         0.251084
Cloud9am         0.301596
RainTomorrow     0.339369
Humidity3pm      0.373927
Humidity9am      0.390166
RainToday        0.857659
Rainfall         1.000000
Name: Rainfall, dtype: float64
```

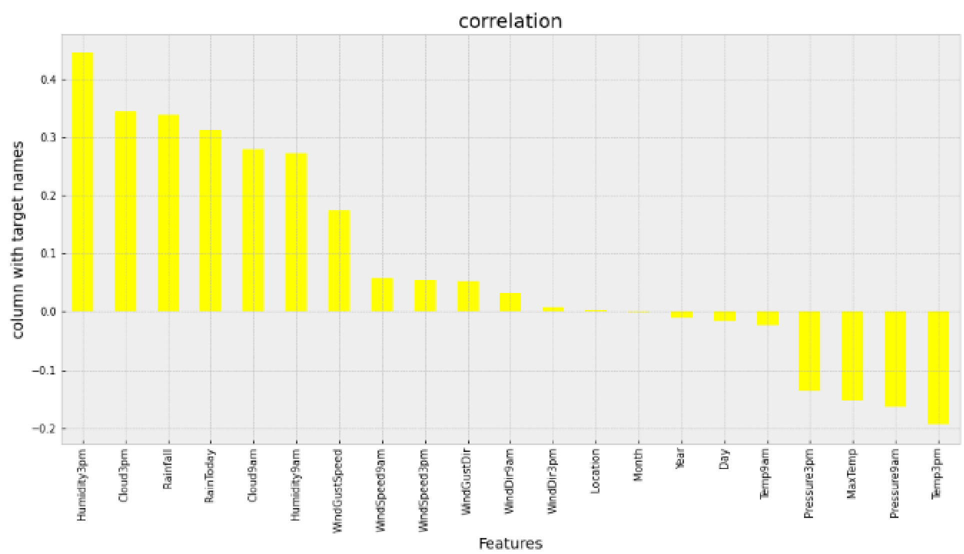
we see that there is only one high correlation value with RainToday, but I will retain it as it is an important feature in forecasting the RainTomorrow

Let us understand the correlation with the help of a heatmap



The heat map is not clear but trust me it is not clear in my Jupiter notebook either. The advantage is that we can identify high correlation values based on their colour as they stand out easily.

Let us also check out the bar plot for our target variable with all the features which will help us to clearly differentiate between the positively and negatively correlated values.



After taking care of the outliers, skewness and checking the correlation I will now split my data into features and target. For our target variable RainTomorrow (Classification Problem) I will split my data into features x1 and target y1

```
x1=df_new_z.drop('RainTomorrow',axis=1)
y1=df_new_z['RainTomorrow']
print(x.shape)
print(y.shape)
```

```
(7549, 22)
(7549,)
```

Applying Smote

We had observed during visualization that there was an imbalance between the label classes. There was a huge difference between the “Yes” and “No” data. Therefore, I will have to resolve it as the imbalance can make our machine learning model biased towards the “No” value.

```
y1.value_counts()
```

```
0    5873
1    1676
Name: RainTomorrow, dtype: int64
```

```
from imblearn.over_sampling import SMOTE
oversample=SMOTE()
x1,y1=oversample.fit_resample(x1,y1)
```

```
y1.value_counts()
```

```
0    5873
1    5873
```

PCA

```
from sklearn.decomposition import PCA
pca=PCA()
x1=pca.fit_transform(x1)
x1
```

I have performed PCA to simplify the complexity in high dimensional data and to denoise it as well

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x1=sc.fit_transform(x1)
x1
```

Then I have scaled the feature columns that are stored in the x1 variable to avoid any kind of biasness over column values. Some integers cover thousands place and some cover hundreds or tens place, therefore it can make the machine learning model assume that the column with thousands place has a higher importance when actually that won't be the case.

Power Transform

```
from sklearn.preprocessing import power_transform
x1=power_transform(x1,method='yeo-johnson')
x1
```

I have used power transform to increase the symmetry of the distribution of the features.

We have completed all the major steps involved in EDA and pre-processing pipeline that will assist us in building an accurate Machine Learning model.

Now let us move onto building our Machine Learning Model

5.__Building Machine Learning Models__

Let us import all the necessary libraries first

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier


from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

Before applying different machine learning algorithms let us find the best random state. I have divided my training and testing data, 80% for training and 20% for testing. There is no hard and fast rule to split the data, you have to use hit and trial to get the best accuracy at a random state.

```
maxAccu=0
maxRS=0

for i in range(1, 1000):
    x_train, x_test, y_train, y_test = train_test_split(x1, y1, test_size=0.2, random_state=i)
    lr=LogisticRegression()
    lr.fit(x_train, y_train)
    pred = lr.predict(x_test)
    acc_score = (accuracy_score(y_test, pred))*100

    if acc_score>maxAccu:
        maxAccu=acc_score
        maxRS=i

print("Best accuracy is", maxAccu,"at Random State", maxRS)
```

```
Best accuracy is 80.7570977917981 at Random State 589
```

Now I will be choosing my random state value ‘i’ as 589

It is always best to build at least 5 machine learning models so that one can choose from the best performing model and then apply hyper parameter tuning to make it perform even better.

Logistic Regression

```
x_train,x_test,y_train,y_test=train_test_split(x1,y1,test_size=.2,random_state=589)

lr=LogisticRegression()
#training the model
lr.fit(x_train,y_train)

#Predicting y_test
pred=lr.predict(x_test)

#accuracy score
acc_score=(accuracy_score(y_test,pred))*100
print('Accuracy Score: ',acc_score)

#classification report
class_report=classification_report(y_test,pred)
print('\nClassification Report \n',class_report)

#cross Validation score
cv_score=(cross_val_score(lr,x1,y1,cv=5).mean())*100
print('Cross Validation Score:',cv_score)

# Result of accuracy minus cv scores
result = acc_score - cv_score
print("\nAccuracy Score - Cross Validation Score is", result)
```

```
Accuracy Score:  76.85106382978724

Classification Report
               precision    recall  f1-score   support

           0       0.76      0.79      0.78      1194
           1       0.78      0.75      0.76      1156

    accuracy                           0.77      2350
   macro avg       0.77      0.77      0.77      2350
weighted avg       0.77      0.77      0.77      2350

Cross Validation Score: 69.80299448384555

Accuracy Score - Cross Validation Score is 7.048069345941684
```

Support vector classifier ¶

```
x_train,x_test,y_train,y_test=train_test_split(x1,y1,test_size=.2,random_state=589)

svc=SVC()
#training the model
svc.fit(x_train,y_train)

#Predicting y_test
pred=svc.predict(x_test)

#accuracy score
acc_score=(accuracy_score(y_test,pred))*100
print('Accuracy Score: ',acc_score)

#classification report
class_report=classification_report(y_test,pred)
print('\nClassification Report \n',class_report)

#cross Validation score
cv_score=(cross_val_score(svc,x1,y1,cv=5).mean())*100
print('Cross Validation Score:',cv_score)

# Result of accuracy minus cv scores
result = acc_score - cv_score
print("\nAccuracy Score - Cross Validation Score is", result)
```

```
Accuracy Score:  88.55319148936171

Classification Report
               precision    recall  f1-score   support

           0       0.89      0.88      0.89      1194
           1       0.88      0.89      0.88      1156

    accuracy                           0.89      2350
   macro avg       0.89      0.89      0.89      2350
weighted avg       0.89      0.89      0.89      2350

Cross Validation Score: 74.93648179850186

Accuracy Score - Cross Validation Score is 13.61670969085985
```

Decision Tree Classifier

```
x_train,x_test,y_train,y_test=train_test_split(x1,y1,test_size=.2,random_state=589)

dt=DecisionTreeClassifier()
#training the model
dt.fit(x_train,y_train)

#Predicting y_test
pred=dt.predict(x_test)

#accuracy score
acc_score=(accuracy_score(y_test,pred))*100
print('Accuracy Score: ',acc_score)

#classification report
class_report=classification_report(y_test,pred)
print('\nClassification Report \n',class_report)

#cross Validation score
cv_score=(cross_val_score(dt,x1,y1,cv=5).mean())*100
print('Cross Validation Score:',cv_score)

# Result of accuracy minus cv scores
result = acc_score - cv_score
print("\nAccuracy Score - Cross Validation Score is", result)
```

```
Accuracy Score:  87.06382978723404

Classification Report 
              precision    recall  f1-score   support

           0       0.89      0.86      0.87      1194
           1       0.86      0.89      0.87      1156

    accuracy                           0.87      2350
   macro avg       0.87      0.87      0.87      2350
weighted avg       0.87      0.87      0.87      2350

Cross Validation Score: 78.9465576116591

Accuracy Score - Cross Validation Score is 8.11727217557494
```

KNeighborsClassifier

```
x_train,x_test,y_train,y_test=train_test_split(x1,y1,test_size=.2,random_state=589)

kn=KNeighborsClassifier()
#training the model
kn.fit(x_train,y_train)

#Predicting y_test
pred=kn.predict(x_test)

#accuracy score
acc_score=(accuracy_score(y_test,pred))*100
print('Accuracy Score: ',acc_score)

#classification report
class_report=classification_report(y_test,pred)
print('\nClassification Report \n',class_report)

#cross Validation score
cv_score=(cross_val_score(kn,x1,y1,cv=5).mean())*100
print('Cross Validation Score:',cv_score)

# Result of accuracy minus cv scores
result = acc_score - cv_score
print("\nAccuracy Score - Cross Validation Score is", result)
```

```
Accuracy Score:  87.02127659574468

Classification Report 
               precision    recall  f1-score   support

           0       0.96      0.78      0.86      1194
           1       0.81      0.96      0.88      1156

    accuracy                           0.87      2350
   macro avg       0.88      0.87      0.87      2350
weighted avg       0.88      0.87      0.87      2350

Cross Validation Score: 71.9394835285273

Accuracy Score - Cross Validation Score is 15.081793067217376
```

Random Forest Classifier

```
x_train,x_test,y_train,y_test=train_test_split(x1,y1,test_size=.2,random_state=589)

rf=RandomForestClassifier()
#training the model
rf.fit(x_train,y_train)

#Predicting y_test
pred=rf.predict(x_test)

#accuracy score
acc_score=(accuracy_score(y_test,pred))*100
print('Accuracy Score: ',acc_score)

#classification report
class_report=classification_report(y_test,pred)
print('\nClassification Report \n',class_report)

#cross Validation score
cv_score=(cross_val_score(rf,x1,y1,cv=5).mean())*100
print('Cross Validation Score:',cv_score)

# Result of accuracy minus cv scores
result = acc_score - cv_score
print("\nAccuracy Score - Cross Validation Score is", result)
```

```
Accuracy Score:  94.46808510638299

Classification Report
               precision    recall  f1-score   support

           0       0.94      0.95      0.95      1194
           1       0.95      0.94      0.94      1156

    accuracy                           0.94      2350
   macro avg       0.94      0.94      0.94      2350
weighted avg       0.94      0.94      0.94      2350

Cross Validation Score: 85.60388395242884

Accuracy Score - Cross Validation Score is 8.864201153954141
```

The best model is the one which gives us a good accuracy and at the same time has the least difference between the Accuracy Score-Cross Validation Score and that is why I have chosen Random Forest Classifier for hyperparameter tuning.

Hyperparameter tuning on Random Forest Classifier

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]
}

rf = RandomForestClassifier()

grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                          cv = 3, n_jobs = -1, verbose = 2)
```

We can refer the sklearn webpage to get the parameters of any classifier

```
grid_search.fit(x_train, y_train)
```

```
grid_search.best_params_
```

```
grid_search.best_score_
```

```
0.8847381864623244
```

Now we just have to choose our best parameters to get the best accuracy.

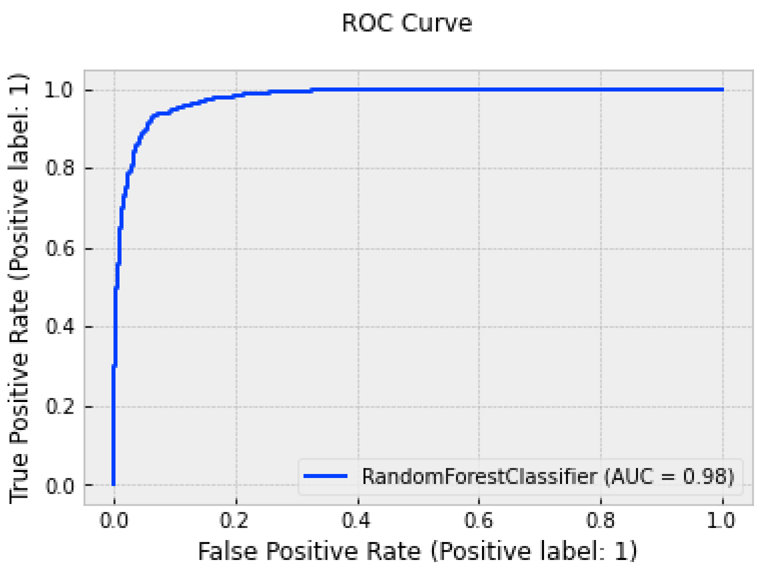
```
Final_Model = RandomForestClassifier(bootstrap=True,max_features=3, max_depth=80, min_samples_leaf=3,
                                     min_samples_split=8,n_estimators=1000)
Classifier = Final_Model.fit(x_train, y_train)
fmod_pred = Final_Model.predict(x_test)
fmod_acc = (accuracy_score(y_test, fmod_pred))*100
print("Accuracy score for the Best Model is:", fmod_acc)
```

```
Accuracy score for the Best Model is: 92.8936170212766
```

I have also plotted the AUC ROC curve and the confusion matrix as well

AOC ROC Curve

```
from sklearn import metrics
disp = metrics.plot_roc_curve(Final_Model, x_test, y_test)
disp.figure_.suptitle("ROC Curve")
plt.show()
```

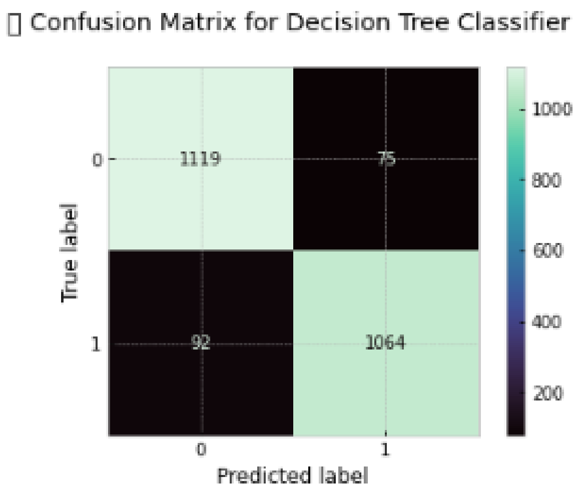


The AUC ROC score for my model is 98%

The more area the curve covers the better the AUC score

Confusion Matrix

```
class_names = df.columns
metrics.plot_confusion_matrix(Classifier, x_test, y_test, cmap='mako')
plt.title('\t Confusion Matrix for Decision Tree Classifier \n')
plt.show()
```



From the confusion matrix we can see that our TP=1119, FP=75, FN=92, TN=1064

that means that are 1119 positive values were correctly classified

75 negative values were correctly classified

92 positive values were incorrectly classified

1064 negative values were incorrectly classified

Once you have gone through all the previous steps and are satisfied with the outcome you can then save the final model using either joblib or pickle. I have used the joblib method to save and then load my model from the same, saved as filename.

MODEL SAVING

```
import pickle
filename = "FinalModel_Blog.pkl"
joblib.dump(Final_Model, filename)
```

```
['FinalModel_Blog.pkl']
```

Loading The model

```
load_model = joblib.load(filename)
result = load_model.score(x_test,y_test)*100
print(result)
```

```
92.8936170212766
```

6.***Concluding Remarks***

Let us have a quick rundown of all the steps that we have accomplished from the onset, from understanding the Problem Definition to performing the Data Analysis and EDA. We executed the necessary Pre-processing Data steps before building our final Machine Learning Model.

In the initial phase of your journey into building ML models I would recommend to use a reference model for building your model. Having a look at multiple models and try to gain insights at what techniques and steps should be performed for what datasets. It will only come after practice and soon it'll come to you naturally. The more you write your code yourself and try to gather maximum insights from every line of code, the better you will get at it. This will be beneficial for you in the long run. Build up your own unique model and then compare it with a good model on the internet, try to understand what could have made your model better, what steps are you lacking in or what steps have you missed. What plots should be used for what type of data, which techniques should be used in the pre-processing pipeline. Put a lot of effort into EDA, as it is the basis for building an accurate ML model, practice it again and again using different approaches.

I would like to conclude by saying that nothing in life comes easy, you have to put in efforts from your end. Remember the time when you were struggling to walk and now it comes to you naturally. We can apply the same thinking to data science as well, you will have to struggle in the begin to enjoy the fruits of your effort later on.