A Blog On

Rainfall Prediction with Machine Learning

Anshul Dubey

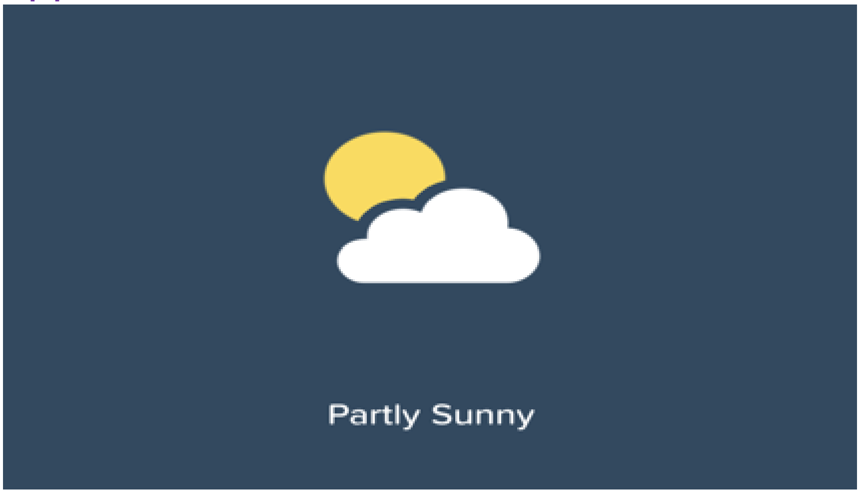
(Data Science Student of

“Data Trained”)

Date:-25/07/22

Introduction

Rainfall Prediction is one of the difficult and uncertain tasks that have a significant impact on human society. it’s applications in the various sectors such



as in agriculture, utility company and in day to day life. Timely and accurate forecasting can proactively help reduce human and financial loss. This study presents a set of experiments that involve the use of common machine learning techniques to create models that can predict whether it will rain tomorrow or not based on the weather data for that day in major cities in Australia.

TABLE OF CONTENTS

Problem Statement:-

Climate is a important aspect of human life. So, the Prediction should accurate as much as possible. In this paper we try to deal with the prediction of the rainfall which is also a major aspect of human life and which provide the major resource of human life which is Fresh Water. Fresh water is always a crucial resource of human survival – not only for the drinking purposes but also for farming, washing and many other purposes.

Making a good prediction of climate is always a major task now a day because of the climate change.

Now climate change is the biggest issue all over the world. Peoples are working on to detect the patterns in climate change as it affects the economy in production to infrastructure. So as in rainfall also making prediction of rainfall is a challenging task with a good accuracy rate. Making prediction on rainfall cannot be done by the traditional way, so scientist is using machine learning and deep learning to find out the pattern for rainfall prediction.

A bad rainfall prediction can affect the agriculture mostly framers as their whole crop is depend on the rainfall and agriculture is always an important part of every economy. So, making an accurate prediction of the rainfall somewhat good. There are number of techniques are used of machine learning but accuracy is always a matter of concern in prediction made in rainfall. There are number of causes made by rainfall affecting the world ex. Drought, Flood and intense summer heat etc. And it will also affect water resources around the world.

Most of the world says that the main cause of this current climate change or global warming is human expansion of the greenhouse gases.

This climate change is impacting the mankind and increasingly influencing their life. This also effecting all the area on which human are depending upon, 3 major area are Water, food and air these are the most important things required by the human to survive. But all these 3 areas are affected due to global warming.

Due to climate change it become difficult for farmer to grow crops, raise animals and catch fish in the same ways and same places as they have done in the past. This also effects the agricultural production. Farmer as the main source of the food for the human but due to this its also get affected. Where as air is also become poisonous there are several no of harmful gases are mixed up in the air which effecting the humans. More people are becoming ill due to this air pollution as for humans it is necessary to breath.

And water which is also an important resource of survival of humans. But due to this climate change availability of the fresh water is decreasing rapidly. No of countries ae facing the shortage of fresh water to drink. This climate changes are not just changing the temperature. The whole water cycle is also get affected. Warmer the world becomes means the atmosphere ha the capacity to hold grater moisture. So, there are changes in the volume of water vapour, rainfall and the flow of water in the atmosphere.

Fresh water is always a crucial resource of human survival – not only for the drinking purposes but also for farming, washing and many other purposes. It is expected to become increasingly scarce in future, and this partly due to climate change.

The scope of this research is wide. Currently peoples are facing major problem due to this climate change. If we able to make a good prediction of weather this will very helpful for the whole mankind human kind.

Import The Dataset:-

Let’s start this task of rainfall prediction by importing the data

```
import numpy as np
df=pd.read_csv("weatherAUS.csv")
```

```
In [355]: df
Out[355]:
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity9am	Humidity3pm	Pressu
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	

I have imported the dataset which was in the csv file using pandas. The dataset contains 8425 rows and 23 columns having both numerical and categorical data. Here I can make use of PCA to reduce the columns, but this will give huge data loss. To avoid this, I am keeping the dataset as it is.

In this dataset " **RainTomorrow** " is our target variable which has two classes. So this is a "Classification type" problem.

you can download the dataset from the below link.

Dataset link:
https://raw.githubusercontent.com/dsrscientist/dataset3/main/weatherAUS.csv

About The Dataset:-

Number of columns: 23

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

RainToday -Boolean: 1 if precipitation (mm) in the 24 hours to 9am exceeds 1mm, otherwise 0

RainTomorrow -The amount of next day rain in mm. Used to create response variable . A kind of measure of the "risk".

Tools Used:-

- Python 3.8
- NumPy
- Pandas
- Matplotlib
- Seaborn
- Data science
- Machine learning

Importing Necessary Libraries:-

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import PowerTransformer
from sklearn.preprocessing import StandardScaler
from scipy.stats import zscore
from matplotlib import rcParams
from sklearn.metrics import plot_confusion_matrix
from sklearn.naive_bayes import GaussianNB
from sklearn import linear_model
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import accuracy_score,
confusion_matrix, classification_report
from sklearn.metrics import precision_score ,
accuracy_score , recall_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve
import warnings
warnings.filterwarnings('ignore')
```

Data Analysis:-

In this part I will check

- Shape of the Dataset
- Null Values
- Information
- Description Of The Dataset(mean,std,min,max)

```
df.shape
```

```
(8425, 23)
```

Shape of the Dataset: our dataframe has 8425 rows and 23 columns.It means we have 8425 Day's 23 type of data

- *null Values*
- ```
  df.isnull().sum()
  ```

  ```
  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
  ```
- *:*

  ```
  #percentage of missing values
  (df.isnull().sum()/8425*100).sort_values(ascending=False)
  ```

  ```
  Sunshine         47.406528
  Evaporation      41.685460
  Cloud3pm         29.139466
  Cloud9am         28.735905
  Pressure3pm      15.572700
  Pressure9am      15.537092
  WindGustDir      11.762611
  WindGustSpeed    11.762611
  WindDir9am        9.839763
  WindDir3pm        3.655786
  RainToday         2.848665
  Rainfall          2.848665
  RainTomorrow      2.836795
  WindSpeed3pm      1.270030
  Humidity3pm       1.210682
  Temp3pm           1.139466
  WindSpeed9am      0.902077
  MinTemp           0.890208
  MaxTemp           0.712166
  Humidity9am       0.700297
  Temp9am           0.664688
  Location          0.000000
  Date              0.000000
  dtype: float64
  ```
-
  ```

We can clearly see from the dataset that apart from the date and location we have null values in almost all of the features and also the target values. We have to handle it

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8425 entries, 0 to 8424
Data columns (total 23 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Date           8425 non-null   object
 1   Location       8425 non-null   object
 2   MinTemp        8350 non-null   float64
 3   MaxTemp        8365 non-null   float64
 4   Rainfall       8185 non-null   float64
 5   Evaporation    4913 non-null   float64
 6   Sunshine       4431 non-null   float64
 7   WindGustDir    7434 non-null   object
 8   WindGustSpeed  7434 non-null   float64
 9   WindDir9am     7596 non-null   object
 10  WindDir3pm     8117 non-null   object
 11  WindSpeed9am   8349 non-null   float64
 12  WindSpeed3pm   8318 non-null   float64
 13  Humidity9am    8366 non-null   float64
 14  Humidity3pm    8323 non-null   float64
 15  Pressure9am    7116 non-null   float64
 16  Pressure3pm    7113 non-null   float64
 17  Cloud9am       6004 non-null   float64
 18  Cloud3pm       5970 non-null   float64
 19  Temp9am        8369 non-null   float64
 20  Temp3pm        8329 non-null   float64
 21  RainToday      8185 non-null   object
 22  RainTomorrow   8186 non-null   object
dtypes: float64(16), object(7)
memory usage: 1.5+ MB
```

- **Information:** Two type of value present in our Data set
  -object
  -float

- **Description Of The Dataset(mean,std,min,max)**

```
df.describe()
```

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressur
count	8350.000000	8365.000000	8185.000000	4913.000000	4431.000000	7434.000000	8349.000000	8318.000000	8366.000000	8323.000000	7116.0
mean	13.193305	23.859976	2.805913	5.389395	7.632205	40.174469	13.847646	18.533662	67.822496	51.249790	1017.6
std	5.403596	6.136408	10.459379	5.044484	3.896235	14.665721	10.174579	9.766986	16.833283	18.423774	6.8
min	-2.000000	8.200000	0.000000	0.000000	0.000000	7.000000	0.000000	0.000000	10.000000	6.000000	989.8
25%	9.200000	19.300000	0.000000	2.600000	4.750000	30.000000	6.000000	11.000000	56.000000	39.000000	1013.0
50%	13.300000	23.300000	0.000000	4.600000	8.700000	39.000000	13.000000	19.000000	68.000000	51.000000	1017.7
75%	17.400000	28.000000	1.000000	7.000000	10.700000	50.000000	20.000000	24.000000	80.000000	63.000000	1022.3
max	28.500000	45.500000	371.000000	145.000000	13.900000	107.000000	63.000000	83.000000	100.000000	99.000000	1039.0

This gives the statistical information of the dataset. There are no negative values and invalid values are present. This gives the summary of numerical data.

EDA:-

EDA or Exploratory Data Analysis is **an expression utilized for data science in which the focus is to understand insights of the data through Visualization or by Statistical Analysis**. The steps involved are - Firstly we focus on variable identification - we identify the data type and category of variables

• For categoriacal columns:

```
categorical_columns=[]
index_c=[]
m=0
for i in df.dtypes.index:
    if df.dtypes[i]=='object':
        categorical_columns.append(i)
        index_c.append(m)
    m=m+1
print(categorical_columns)
print(index_c)
```

```
['Date', 'Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday', 'RainTomorrow']
[0, 1, 7, 9, 10, 21, 22]
```

•For numerical columns:

```
numerical_columns=[]
index_n=[]
m=0
for i in df.dtypes.index:
    if df.dtypes[i]!='object':
        numerical_columns.append(i)
        index_n.append(m)
    m=m+1
print(numerical_columns)
print(index_n)
```

```
['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am',
'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm']
[2, 3, 4, 5, 6, 8, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

I have seprated our columns as per their type they are having. Means I have created two types of list namely categorical_column and numerical_column. I have kept all object type data into categorical_column list and all numerical type data into numerical_columns. We can see that length of categorical column is less than length of numerical column.

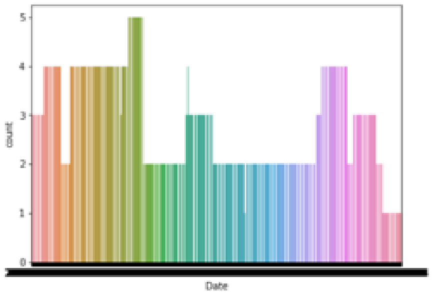
Univariate Analysis Of Categorical Columns

- **Date:**

```python
#Categorical column
i=0 #using the column index
print("column Name:",df.columns[i])
print("\n")
print(df[df.columns[i]].value_counts())
print("len=",len(df[df.columns[i]].value_counts()))
plt.figure(figsize=(7,5))
sns.countplot(df[df.columns[i]])
plt.show()
print('\n')
print("#"*100)
```

```
column Name: Date


2011-05-31    5
2011-05-07    5
2011-06-05    5
2011-03-22    5
2011-03-14    5
             ..
2013-03-12    1
2013-03-01    1
2013-05-04    1
2013-05-09    1
2013-05-22    1
Name: Date, Length: 3004, dtype: int64
len= 3004
```



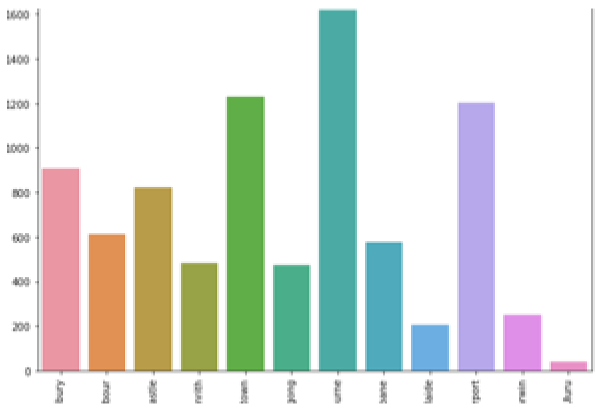
We can see that weather reports of many places have been stored in one day. We separate the column into three columns, day, month and year for better prediction by the code below and drop the Date column.

```python
df["Day"]=pd.to_datetime(df["Date"]).dt.day
```

```python
df["Month"]=pd.to_datetime(df["Date"]).dt.month
```

```python
df["Year"]=pd.to_datetime(df["Date"]).dt.year
```

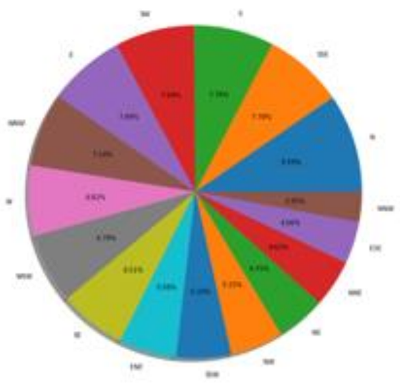
➢ Location:



Count Plot of Location Column

12 location's weather report present in our data. We have the highest rainfall data from Melbourne and least from Uluru

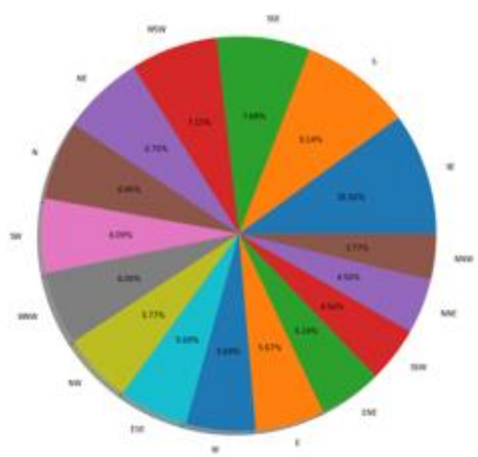
- **WindGustDir:**

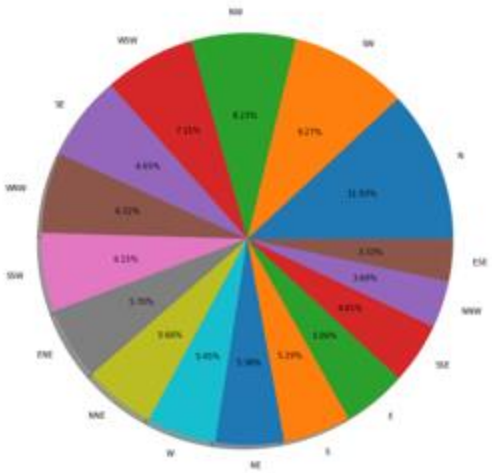


Pie Plot Of WindGusdir

We can clearly see that the wind gust was strongest towards the North, followed by the SW, SSE, S,WNW

- **Windir9am And Windir3pm:**

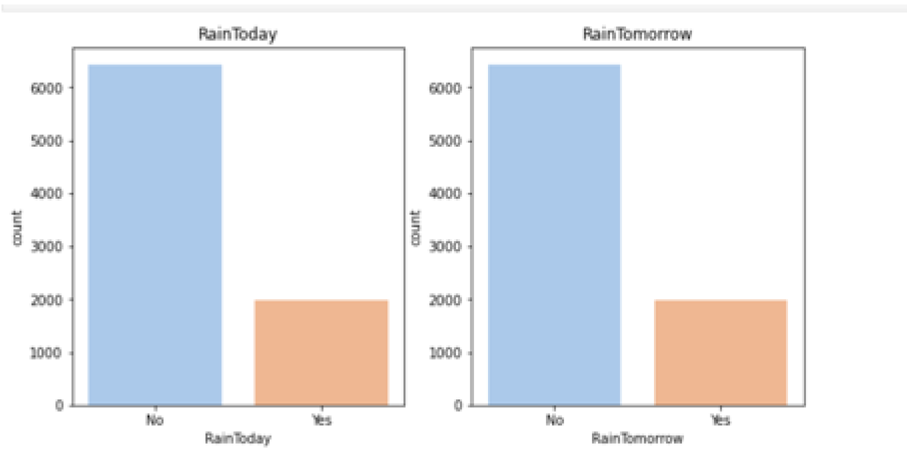




Pie plot of windir3pm

In the plots we can clearly see that the wing direction was towards the N at 9am and in the SE at 3pm

➢ Rain Today And Rain Tomorrow:



count plot of rain today and rain tomorrow

From the count plot, we cannot see any difference in the rainfall today and tomorrow. Here Rain Tomorrow column is our target column. Since in the target column number of yes less than number of no, the class of target column is imbalanced. We have to handle it.

Univariate Analysis of the Numerical columns

Dist Plot:-

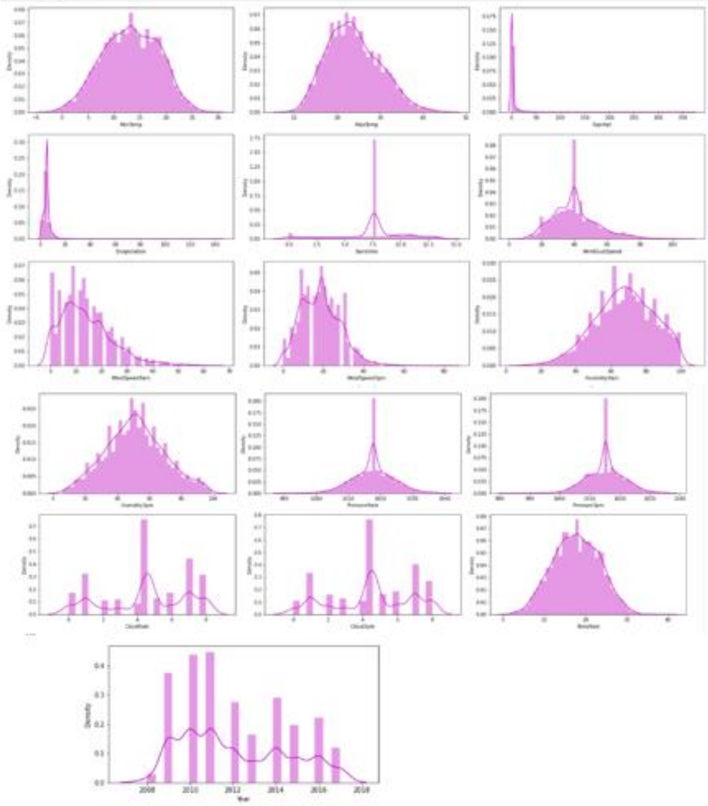
A distplot plots a univariate distribution of observations. The distplot() function combines the matplotlib hist function with the seaborn kdeplot() and rugplot() functions. Seaborn distplot lets you show a histogram with a line on it.

distplot() function is used to plot the distplot. The distplot represents the univariate distribution of data i.e. data distribution of a variable against the density distribution. The seaborn. distplot() function accepts the data variable as an argument and returns the plot with the density distribution.

To explore a more about distplot **Distplot**.

➢ Dist Plot of Every Numerical columns:

```
plt.figure(figsize=(20,25),facecolor='white')
plotnumber=1
for column in num_col:
    if plotnumber<=20:
        ax=plt.subplot(7,3,plotnumber)
        sns.distplot(df[column],color="m")
        plt.xlabel(column,fontsize=10)
        plt.yticks(rotation=0,fontsize=10)
    plotnumber+=1
plt.tight_layout()
```



- ✓ No skewness Columns(like as a normal distribution curve):
  -MinTemp
  -Humidity3pm
  -Pressure9am
  -Pressure3pm
  -Cloud9am
  -Temp9am
  -Temp3pm

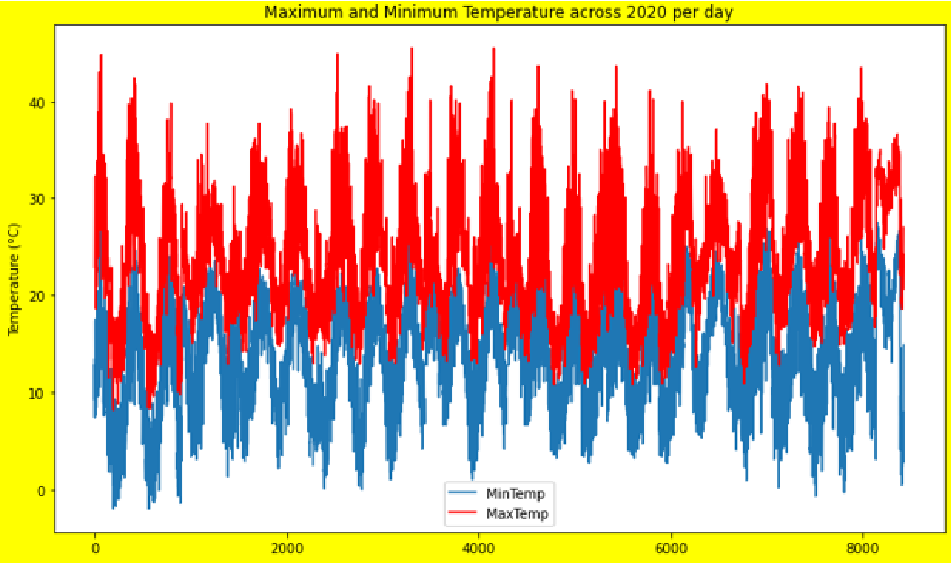
positive Skewness columns:
-MaxTemp
-Rainfall
-Evaporation
-WindSpeed9am
-WindSpeed3pm

- ✓ Negative Skewness Columns:
  -Sunshine
  -Humidity9am
  -Pressure3pm

Bivariate Analysis:

➢ MinTemp Vs MaxTemp

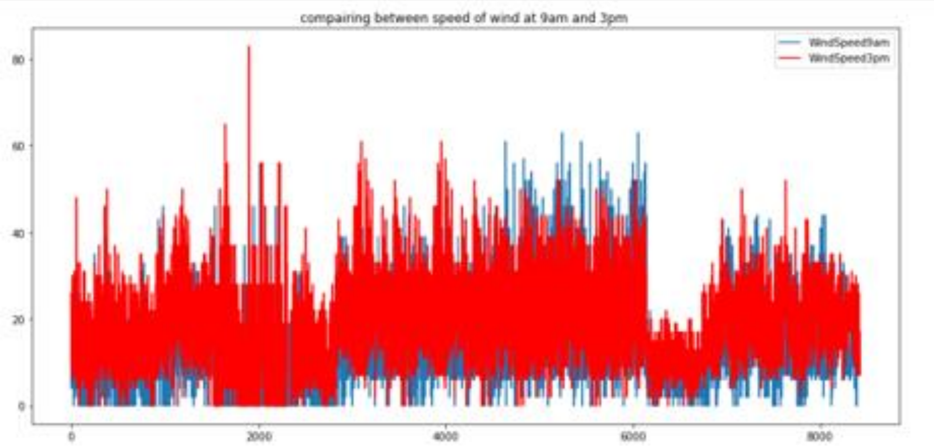
```
max_temp = df['MaxTemp']
min_temp = df['MinTemp']
min_temp.plot(figsize=(12,7), legend=True)
max_temp.plot(figsize=(12,7), color='r', legend=True)
plt.title('Maximum and Minimum Temperature across 2020 per day')
plt.ylabel('Temperature (°C)')
plt.show()
```



From the above plot we can see the difference between the daily minimum and maximum temperature

➢ WindSpeed9am Vs WindSpeed3pm

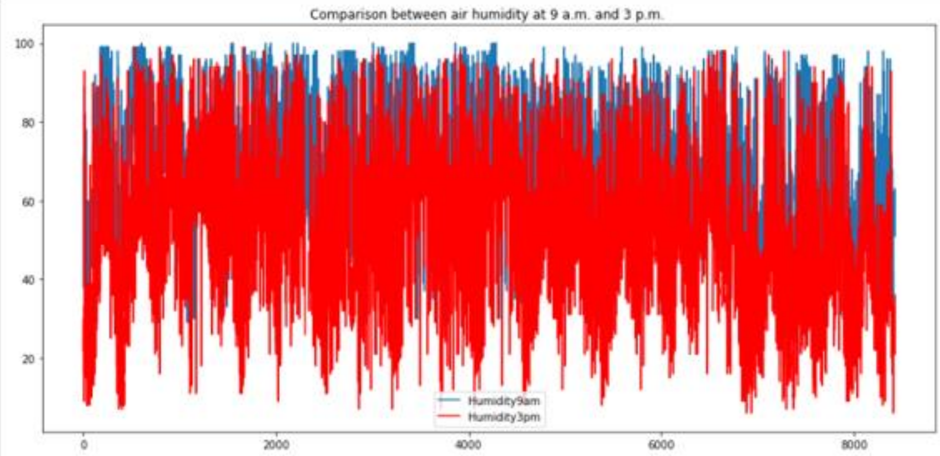
```
df['WindSpeed9am'].plot(figsize=(15,7), legend=True)
df['WindSpeed3pm'].plot(figsize=(15,7), color='r', legend=True)
plt.title('compairing between speed of wind at 9am and 3pm ')
plt.show()
```



We see the difference between wind speed at 9 in the morning and wind speed at 3 in the afternoon. The wind speed fluctuates at 3 in the afternoon rather than the wind speed at 9 in the morning

➢ Humidity9am Vs Humidity3pm

```
df['Humidity9am'].plot(figsize=(15,7), legend=True)
df['Humidity3pm'].plot(figsize=(15,7), color='r', legend=True)
plt.title('Comparison between air humidity at 9 a.m. and 3 p.m. ')
plt.show()
```

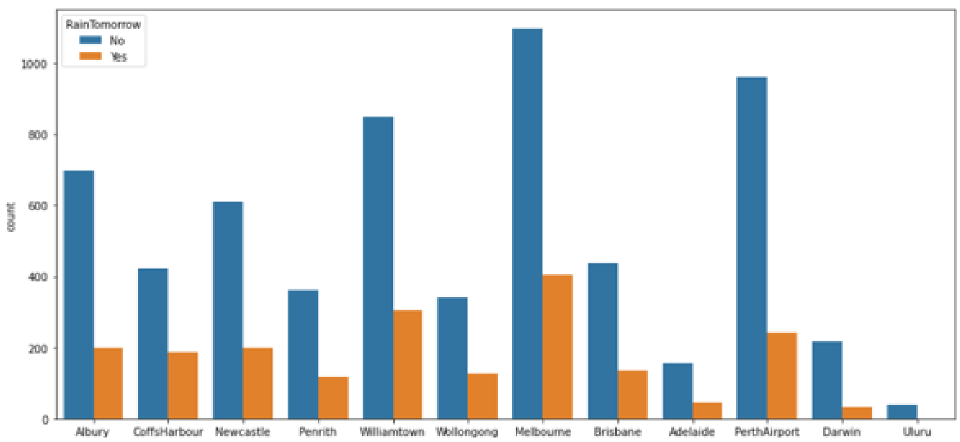


We see the difference between Humidity at 9 in the morning and Humidity at 3 in the afternoon. Morning humidity is always higher than afternoon humidity

➢ Location Vs Rain Tomorrow

```
plt.figure(figsize=(15,7))
sns.countplot(df[df.columns[1]],hue=df['RainTomorrow'])
```

```
<AxesSubplot:xlabel='Location', ylabel='count'>
```

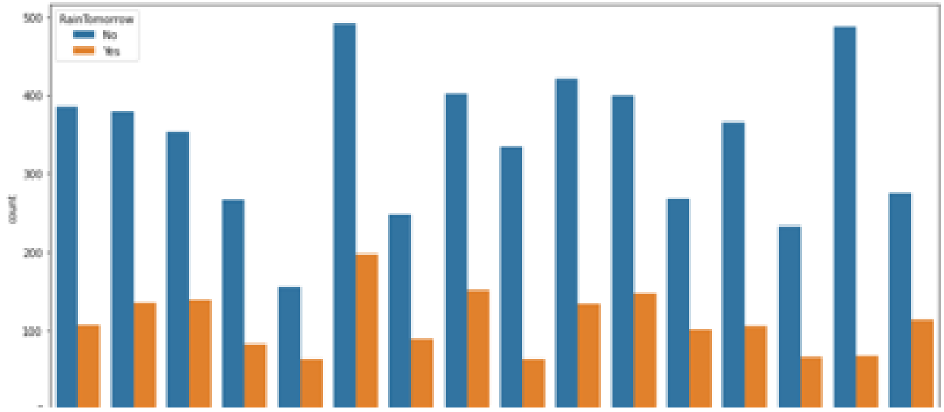


Number of yes less than number of No in rain tomorrow column for each location.we can not see number of yes for Uluru location.

➢ WindGustdir Vs Rain Tomorrow

```
plt.figure(figsize=(15,7))
sns.countplot(df[df.columns[7]],hue=df['RainTomorrow'])
```

```
<AxesSubplot:xlabel='WindGustDir', ylabel='count'>
```



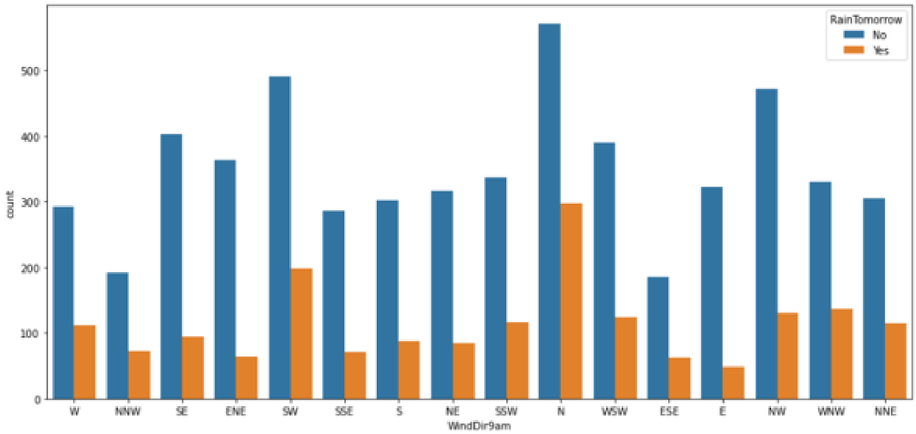
- ✓ If the wind blows east, the chances of rain tomorrow are less than the wind in the other direction.

- ✓ If the wind blows NNW, the chances of rain tomorrow are more than the wind in other direction.

➢ Windir9am Vs Rain Tomorrow

```
plt.figure(figsize=(15,7))
sns.countplot(df[df.columns[9]],hue=df['RainTomorrow'])
```

```
<AxesSubplot:xlabel='WindDir9am', ylabel='count'>
```

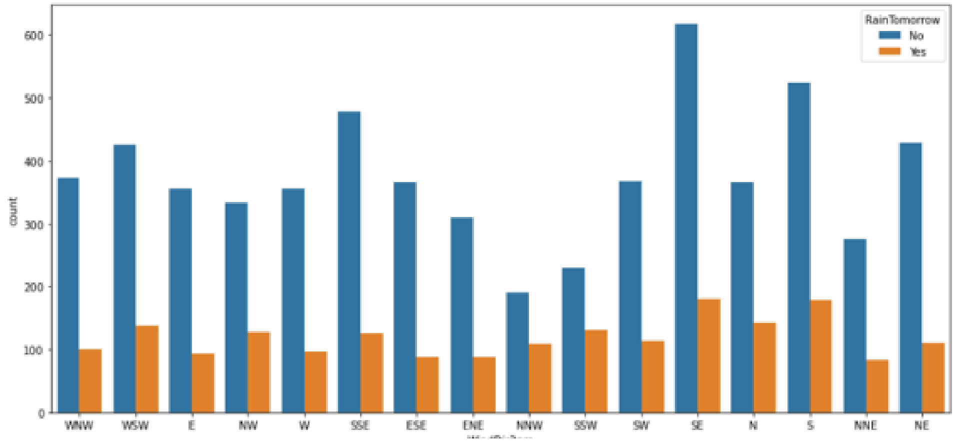


- ✓ If the wind blows ENE in the morning, the chances of rain tomorrow are less than the wind in the other direction.
- ✓ If the wind blows North in the morning, the chances of rain tomorrow are more than the wind in other direction.

➢ WEindir3pm Vs Rain Tomorrow

```
plt.figure(figsize=(15,7))
sns.countplot(df[df.columns[10]],hue=df['RainTomorrow'])
```

```
<AxesSubplot:xlabel='WindDir3pm', ylabel='count'>
```



- ✓ If the wind blows NE in the after noon, the chances of rain tomorrow are less than the wind in the other direction.

- ✓ If the wind blows SSW in the afternoon, the chances of rain tomorrow are more than the wind in other direction.

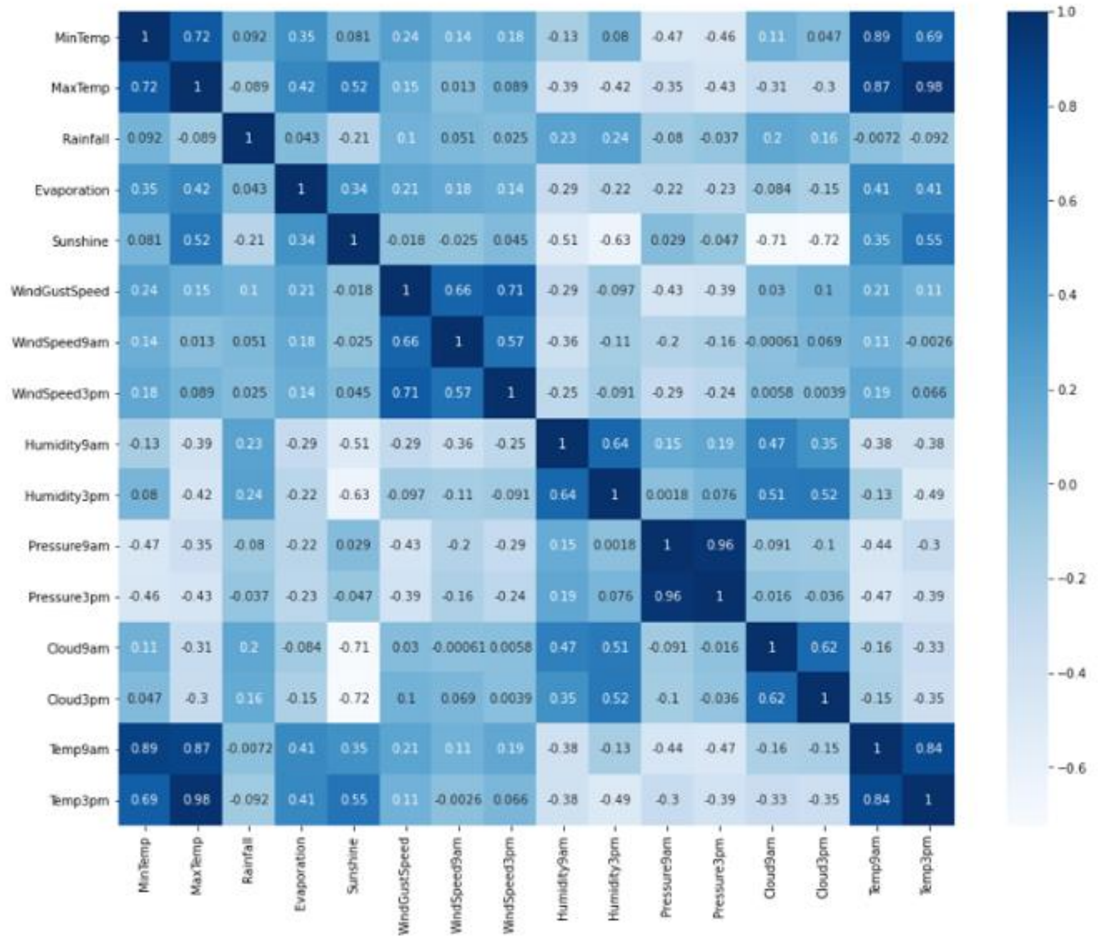
Correlation Using Heatmap:

Correlation is a statistical term describing the degree to which two variables move in coordination with one another. If the two variables move in the same direction, then those variables are said to have a positive correlation. If they move in opposite directions, then they have a negative correlation.

Let’s see the correlation of our dataset using heatmap

```
#checking correation using heatmap
plt.figure(figsize=(15,12))
sns.heatmap(df.corr(),annot=True,cmap='Blues')
```

```
<AxesSubplot:>
```



The following feature pairs have a strong correlation with each other:

- MaxTemp and MinTemp
- Pressure9h and pressure3h
- Temp9am and Temp3pm
- Evaporation and MaxTemp
- MaxTemp and Temp3pm

In no case is the correlation value equal to a perfect “1”. We are therefore not removing any functionality

Preprocessing:-

Data processing is a process of preparing the raw data and making it suitable for it a machine learning model. It is the first and crucial step while creating a machine learning model.

When creating machine learning project, it is note always a case that become across the clean and formatted data and while doing any operation with data it is a mandatory to clean it and put in formatted way. So, for this we use data pre-processing task

In this part I will do all the parts below

- Missing Value Handling
- Outliers Checking
- Outliers Removing
- Percentage Of Data Loss
- Label Encoding
- Dividing Data In Features And Vectors
- Multicolinearty Checking
- Oversampling
- Skewness Removing
- Data Standardizing
- ***Missing Value Handling:***
- **There are 2 primary ways of handling missing values:**
- Deleting the Missing values:-Generally, this approach is not recommended. It is one of the quick and dirty techniques one can use to deal with missing values.
-
- Imputing the Missing Values:- There are different ways of replacing the missing values
- Replacing With Mean
- Replacing With Mode
- Replacing With Median,etc.
- For my dataset,I will use mode for categorical features and median for numerical features by the code below.

```
yes_rain['MinTemp'].fillna(yes_rain['MinTemp'].mode()[0],inplace=True)
no_rain['MinTemp'].fillna(no_rain['MinTemp'].mode()[0],inplace=True)

yes_rain['MaxTemp'].fillna(yes_rain['MaxTemp'].mode()[0],inplace=True)
no_rain['MaxTemp'].fillna(no_rain['MaxTemp'].mode()[0],inplace=True)

yes_rain['Temp9am'].fillna(yes_rain['Temp9am'].mode()[0],inplace=True)
no_rain['Temp9am'].fillna(no_rain['Temp9am'].mode()[0],inplace=True)

yes_rain['Temp3pm'].fillna(yes_rain['Temp3pm'].mode()[0],inplace=True)
no_rain['Temp3pm'].fillna(no_rain['Temp3pm'].mode()[0],inplace=True)

yes_rain['Humidity3pm'].fillna(yes_rain['Humidity3pm'].mode()[0],inplace=True)
no_rain['Humidity3pm'].fillna(no_rain['Humidity3pm'].mode()[0],inplace=True)

yes_rain['Humidity9am'].fillna(yes_rain['Humidity9am'].mode()[0],inplace=True)
no_rain['Humidity9am'].fillna(no_rain['Humidity9am'].mode()[0],inplace=True)

yes_rain['Sunshine'].fillna(yes_rain['Sunshine'].median(),inplace=True)
no_rain['Sunshine'].fillna(no_rain['Sunshine'].median(),inplace=True)

yes_rain['Evaporation'].fillna(yes_rain['Evaporation'].median(),inplace=True)
no_rain['Evaporation'].fillna(no_rain['Evaporation'].median(),inplace=True)

yes_rain['Cloud3pm'].fillna(yes_rain['Cloud3pm'].median(),inplace=True)
no_rain['Cloud3pm'].fillna(no_rain['Cloud3pm'].median(),inplace=True)

yes_rain['Cloud9am'].fillna(yes_rain['Cloud9am'].median(),inplace=True)
no_rain['Cloud9am'].fillna(no_rain['Cloud9am'].median(),inplace=True)

yes_rain['Pressure3pm'].fillna(yes_rain['Pressure3pm'].median(),inplace=True)
no_rain['Pressure3pm'].fillna(no_rain['Pressure3pm'].median(),inplace=True)

yes_rain['Pressure9am'].fillna(yes_rain['Pressure9am'].median(),inplace=True)
no_rain['Pressure9am'].fillna(no_rain['Pressure9am'].median(),inplace=True)

yes_rain['WindGustDir'].fillna(yes_rain['WindGustDir'].mode()[0],inplace=True)
no_rain['WindGustDir'].fillna(no_rain['WindGustDir'].mode()[0],inplace=True)

yes_rain['WindGustSpeed'].fillna(yes_rain['WindGustSpeed'].median(),inplace=True)
no_rain['WindGustSpeed'].fillna(no_rain['WindGustSpeed'].median(),inplace=True)

yes_rain['WindDir9am'].fillna(yes_rain['WindDir9am'].mode()[0],inplace=True)
no_rain['WindDir9am'].fillna(no_rain['WindDir9am'].mode()[0],inplace=True)

yes_rain['WindDir3pm'].fillna(yes_rain['WindDir3pm'].mode()[0],inplace=True)
no_rain['WindDir3pm'].fillna(no_rain['WindDir3pm'].mode()[0],inplace=True)

yes_rain['WindSpeed3pm'].fillna(yes_rain['WindSpeed3pm'].median(),inplace=True)
no_rain['WindSpeed3pm'].fillna(no_rain['WindSpeed3pm'].median(),inplace=True)

yes_rain['WindSpeed9am'].fillna(yes_rain['WindSpeed9am'].median(),inplace=True)
no_rain['WindSpeed9am'].fillna(no_rain['WindSpeed9am'].median(),inplace=True)

yes_rain['Rainfall'].fillna(yes_rain['Rainfall'].median(),inplace=True)
no_rain['Rainfall'].fillna(no_rain['Rainfall'].median(),inplace=True)

yes_rain['RainToday'].fillna(yes_rain['RainToday'].mode(),inplace=True)
no_rain['RainToday'].fillna(no_rain['RainToday'].mode(),inplace=True)
```

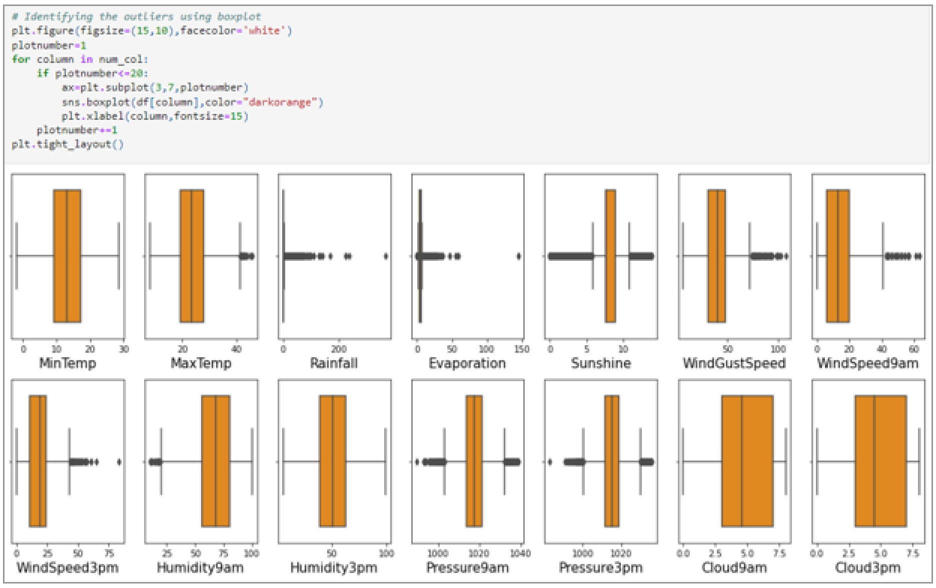
- After running this code ,we are free from missing values

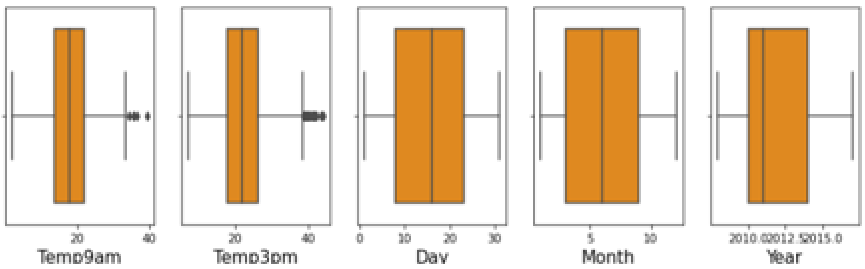
- ***<u>Outliers Checking:</u>*** **Identification of potential outliers is important for the following reasons.**

- An outlier may indicate bad data. For example, the data may have been coded incorrectly or an experiment may not have been run correctly. If it can be determined that an outlying point is in fact erroneous, then the outlying value should be deleted from the analysis (or corrected if possible).

- In some cases, it may not be possible to determine if an outlying point is bad data. Outliers may be due to random variation or may indicate something scientifically interesting. In any event, we typically do not want to simply delete the outlying observation. However, if the data contains significant outliers, we may need to consider the use of robust statistical techniques.
- Boxplots, histograms, and scatterplots can highlight outliers.
- Boxplots display asterisks or other symbols on the graph to indicate explicitly when datasets contain outliers. These graphs use the interquartile method with fences to

find outliers. The boxplot below displays our dataset. It’s clear that the outlier is quite different than the typical data value.





From the above boxplots, we can see that features having outliers are: MaxTemp,Rainfall,Evaporation,Sunshine,WindGustSpeed,WindSpeed9am,Windspeed3pm, Humidity9am, Pressure9am, Pressure3pm, Temp9am, Temp3pm.

➢ ***<u>Outliers Removing:</u>***

There are many ways to detect and remove Outliers.Here I have used Z-score function defined in scipy library to detect the outliers.

The formula for calculating a z-score is is $z = (x-\mu)/\sigma$, where x is the raw score, μ is the population mean, and σ is the population standard deviation.

We have created a new dataframe df_new where

```
#import zscore
from scipy.stats import zscore
```

```
outliers_columns=df[['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm'
```

```
z=np.abs(zscore(outliers_columns))
z
```

```
array([[0.63561368, 1.31942925, 0.26729818, ..., 0.76835198, 1.65173498,
        1.64908411],
       [0.04798793, 1.07392299, 0.26729818, ..., 0.54091631, 1.65173498,
        1.64908411],
       [0.50988091, 0.3580133 , 0.05746013, ..., 0.42719847, 1.65173498,
        1.64908411],
       ...,
       [1.76263255, 0.23920172, 0.26729818, ..., 0.70997992, 0.11086714,
        1.98444785],
       [1.4300696 , 0.50187506, 0.26729818, ..., 0.82369776, 0.11086714,
        1.98444785],
       [0.98665234, 0.51744214, 0.26729818, ..., 0.9374156 , 0.11086714,
        1.98444785]])
```

No outliers present.Let's see the code.

➢ ***Percentage Of Data Loss:***

```
df_new=df[(z<3).all(axis=1)]
df_new
```

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	...	Pressure3pm	Cloud9am	Cl
0	Albury	9.7	31.9	0.0	3.8	4.2	NNW	80.0	SE	NW	...	1003.6	7.0	
1	Albury	13.4	30.4	0.0	3.8	4.2	N	30.0	SSE	ESE	...	1008.7	7.0	
2	Albury	15.9	21.7	2.2	3.8	4.2	NNE	31.0	NE	ENE	...	1004.2	8.0	
4	Albury	14.1	20.9	0.0	3.8	4.2	ENE	22.0	SSW	E	...	1010.4	8.0	
5	Albury	13.5	22.9	16.8	3.8	4.2	W	63.0	N	WNW	...	1002.2	8.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
8181	Uluru	3.5	21.8	0.0	4.8	9.6	E	31.0	ESE	E	...	1021.2	4.0	
8182	Uluru	2.8	23.4	0.0	4.8	9.6	E	31.0	SE	ENE	...	1020.3	4.0	
8183	Uluru	3.6	25.3	0.0	4.8	9.6	NNW	22.0	SE	N	...	1019.1	4.0	
8184	Uluru	5.4	26.9	0.0	4.8	9.6	N	37.0	SE	WNW	...	1016.8	4.0	
8185	Uluru	7.8	27.0	0.0	4.8	9.6	SE	28.0	SSE	N	...	1016.5	3.0	

7663 rows × 25 columns

We lost 9 percent of our total data.

➢ ***Label Encoding:***

Label Encoding refers to converting the labels into a numeric form so as **to convert them into the machine-readable form**. Machine learning algorithms can then decide in a better way how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning.

I have applied label encoding method to our cleaned dataframe df_new and converted the categorical columns into numerical

```
# checking for categorical columns
categorical_columns=[]
for i in df_new.dtypes.index:
    if df_new.dtypes[i]=='object':
        categorical_columns.append(i)
print(categorical_columns)
```

```
['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday', 'RainTomorrow']
```

```
list_c=['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday', 'RainTomorrow']
```

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
for i in list_c:
    df_new[i]=le.fit_transform(df_new[i]).astype(float)
```

➢ ***Dividing Data In Features And Vectors:***

We have to divide our dataset columns into X and Y. X variables so that we could have all the attribute columns and our label / target variable with the Y variable

```
x=df_new.drop("RainTomorrow",axis=1) #Independent variable
y=df_new.iloc[:,-4] #Dependent variable
```

➢ ***Multicolinearity Checking:***

Multicollinearity occurs when two or more independent variables are highly correlated with one another in a regression model. This means that an independent variable can be predicted from another independent variable in a regression model.

The best way to identify the multicollinearity is to calculate the Variance Inflation Factor (VIF) corresponding to every independent Variable in the Dataset. VIF tells us about how well an independent variable is predictable using the other independent variables.

Let's see the code how to get VIF score:

```
# checking for numerical columns
numerical_columns=[]
for i in df.dtypes.index:
    if df.dtypes[i]!='object':
        numerical_columns.append(i)
print(numerical_columns)
```

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

```
len(numerical_columns)
```

```
19
```

```
p=x[['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am',
```

```python
from statsmodels.stats.outliers_influence import variance_inflation_factor
#define a function to calculate VIF score
def vif_clac():
    vif=pd.DataFrame()
    vif["VIF Factor"]=[variance_inflation_factor(p.values,i) for i in range(p.shape[1])]
    vif["features"]=p.columns
    print(vif)
```

```python
#checking VIF score
vif_clac()
```

```
       VIF Factor       features
0       58.226518        MinTemp
1      428.231075        MaxTemp
2        1.419604       Rainfall
3        8.476148    Evaporation
4       16.206798       Sunshine
5       22.841686  WindGustSpeed
6        5.875466   WindSpeed9am
7        9.231133   WindSpeed3pm
8       67.534248    Humidity9am
9       48.481854    Humidity3pm
10  518536.993289    Pressure9am
11  519092.043543    Pressure3pm
12       8.354604       Cloud9am
13       8.678990       Cloud3pm
14     188.983176        Temp9am
15     512.768419        Temp3pm
16       4.200768            Day
17       5.248096          Month
18   42847.736841           Year
```

We find the multicollinearity in columns below

- ✓ Pressure9am
- ✓ Temp9am
- ✓ Temp3pm
- ✓ Year
- ✓ Pressure3pm

We drop this column from feature of the dataset.

➢ ***Oversampling:***

When one class of data is the underrepresented minority class in the data sample, over sampling techniques maybe used to duplicate these results for a more balanced amount of positive results in training. Over sampling is used when the amount of data collected is insufficient.

SMOTE is an oversampling technique where the synthetic samples are generated for the minority class. This algorithm helps to overcome the overfitting problem posed by random oversampling.

I have separated feature and label into x and y. And there is a data imbalance issue in target variable. Let’s use SMOTE to overcome from this.

```
from imblearn.over_sampling import SMOTE
smt=SMOTE()
```

```
x,y=smt.fit_resample(x,y)
```

```
y.value_counts()
```

```
0.0    5903
1.0    5903
Name: RainTomorrow, dtype: int64
```

➢ kewness Removing:

If there are too much skewness in the data, then many statistical model don't work .In skewed data, the tail region may act as an outlier for the statistical model and we know that **outliers adversely affect the model's performance especially regression-based models**.

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

```
#print skewness after power transform
df_new1=pd.DataFrame(x)
df_new1.skew()
```

```
0    -0.171669
1    -0.094372
2    -0.011278
3     0.624919
4     0.039669
5    -0.194701
6    -0.185355
7    -0.002709
8    -0.148276
9    -0.228968
10   -0.116658
11   -0.088553
12   -0.102193
13   -0.068445
14   -0.402746
15   -0.347735
16    0.675677
17   -0.203307
18   -0.131737
dtype: float64
```

I used power transformation (method= yeo-johnson) method to remove skewness of the independent dataset.

From above we can observe that we have successfully removed skewness from our independent dataset.

➢ *Data Standardizing:*

Data standardization is a data processing workflow that converts the structure of different datasets into one common format of data.Data standardization helps improve the quality of your data by transforming and standardizing it. Think of it like a uniform for your databases. By taking this step, you are formatting your records in a way that creates consistency across your systems and makes it easy for businesses to use.

Now let's see how I did standardize the data:

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

```
array([[-1.62140226, -0.68753025,  1.41877699, ..., -0.73170524,
        -0.67307296,  1.65173851],
       [-1.62140226,  0.01716676,  1.20902054, ..., -0.73170524,
        -0.42490975,  1.65173851],
       [-1.62140226,  0.49188581, -0.16506044, ...,  1.4407457 ,
        -0.30545611,  1.65173851],
       ...,
       [-1.68655254, -1.54187414, -1.35954753, ...,  1.4407457 ,
        -0.54733047,  0.28936877],
       [ 0.02575626, -0.03079603, -0.57952934, ...,  1.4407457 ,
         0.467331  ,  0.28936877],
       [-1.78360518, -1.11988939, -1.70545696, ...,  1.4407457 ,
         0.2560823 , -0.01032262]])
```

Since I have done all the pre-processing now my data is ready to build the model. Let's get the predictions by creating some classification algorithms as it is a classification problem.

Building Machine Learning:-

In here we will use various classification algorithm to predict our target. Let's have an overview of the algorithms we will use for our predictions. To read more about these algorithms, just click on the algorithms name.

- **Logistic Regression :-** Logistic regression uses an equation as the representation, very much like linear regression. Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is a *binary value* (0 or 1) rather than a numeric value. Below is an example logistic regression equation:

$y = e^{(b0 + b1x)} / (1 + e^{(b0 + b1x)})$

Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x).

- **K-Nearest Neighbors :-** In KNN, K is the number of nearest neighbors. The number of neighbors is the core deciding factor. K is generally an odd number if the number of classes is 2. When K=1, then the algorithm is known as the nearest neighbor algorithm. This is the simplest case. Suppose P1 is the point, for which label needs to predict. First, knn find the one closest point to P1 and then the label of the nearest point assigned to P1. Suppose P1 is the point, for which label needs to predict. First, knn find the k closest point to P1 and then classify points by majority vote of its k neighbors. Each object votes for their class and the class with the most votes is taken as the prediction. For finding closest similar points, knkn find the distance between points using distance measures such as Euclidean distance, Hamming distance, Manhattan distance and Minkowski distance.

- **Support Vector Classifier (SVC) :-** In the SVM algorithm, we plot each data item as a point in n- dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well. Support Vectors are simply the co-ordinates of individual observation. The SVM classifier is a frontier which best segregates the two classes (hyper-plane/ line).
- **Random Forest Classifier :-** It technically is an ensemble method (based on the divide-and-conquer approach) of decision trees generated on a randomly split dataset. This collection of decision tree classifiers is also known as the forest. The individual decision trees are generated using an attribute selection indicator such as information gain, gain ratio, and Gini index for each attribute. Each tree depends on an independent random sample. In a classification problem, each tree votes and the most popular class is chosen as the final result.
- **AdaBoost Classifier :-** Ada-boost or Adaptive Boosting is one of ensemble boosting classifier proposed by Yoav Freund and *Robert Schapira* in 1996. It combines multiple classifiers to increase the accuracy of classifiers. AdaBoost is an iterative ensemble method. AdaBoost classifier builds a strong classifier by combining multiple poorly performing classifiers so that you will get high accuracy strong classifier. The basic concept behind Adaboost is to set the weights of classifiers and training the data sample in each iteration such that it ensures the accurate predictions of unusual observations.
- **Decision Tree Classifier :-** A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps you in decision making.
- **Gaussian :-** Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

❖ ***Finding Best Random State:-***

An algorithm might have multiple points that introduce randomness to the process and thus

```python
#finding best random state
maxAccu=0
maxRS=0
for i in range(0,2000):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=i)
    lr=LogisticRegression()
    lr.fit(x_train,y_train)
    predlr=lr.predict(x_test)
    acc=accuracy_score(y_test,predlr)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print("Best accuracy score is :",maxAccu,"on random state  ",maxRS)
```

```
Best accuracy score is : 0.8972332015810277 on random state   1146
```

introduce randomness to the result. One method to make sure our results are constant is to set every possible *random_state* available in the functions that we use. We can find the best random state by the code below

After we have found the value for best random state, we proceeded with the train test split function to create new training and testing datasets and fit them into the models to find our ideal models.

➢ ***Train_Test_Split:-***

The Train_Test_Split is a technique for evaluating the performance of a machine learning algorithm. It can be used for classification or regression problems and can be used for any supervised learning algorithm. The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

```python
#train test split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=maxRS)
```

```python
#chacking shape of all variables which are geting from train test split function
print("x_train shape =",x_train.shape)
print("x_test shape =",x_test.shape)
print("y_train shape =",y_train.shape)
print("y_test shape =",y_test.shape)
```

```
x_train shape = (8264, 19)
x_test shape = (3542, 19)
y_train shape = (8264,)
y_test shape = (3542,)
```

Now I will be using 8 differents classification algorithms to get the ideal one for prediction.

I have used evaluation metrics like classification report, confusion matrix, roc score and and cross validation score to get the difference from the model accuracy.

```
lg=LogisticRegression()
gnb=GaussianNB()
svc=SVC()
dtc=DecisionTreeClassifier()
knn=KNeighborsClassifier()
rfc=RandomForestClassifier()
grb=GradientBoostingClassifier()
adb=AdaBoostClassifier()

#creat a list of all above model
model=[lg,gnb,svc,dtc,knn,rfc,grb,adb]
```

```
for m in model:
    m.fit(x_train,y_train)
    predm=m.predict(x_test)
    print('Accuracy score of ',m,'is')
    print(accuracy_score(y_test,predm))
    print(confusion_matrix(y_test,predm))
    print(classification_report(y_test,predm))
    print('*'*125)
    print('_'*125)
    print("\n")
```

Accuracy score of Logistic Regression() is0.8972332015810277
[[1589 167]
[197 1589]]   precision   recall f1-score   support
0.0   0.89   0.90   0.90   1756

1.0   0.90   0.89   0.90   1786   accuracy
0.90   3542   macro avg   0.90
0.90   0.90   3542   weighted avg
0.90   0.90   0.90   3542
****

Accuracy score of GaussianNB() is   0.8430265386787126
[[1487 269]
[287 1499]]
precision   recall f1-score   support
0.0   0.84   0.85   0.84   1756
1.0   0.85   0.84   0.84   1786
accuracy   0.84   3542
macro avg   0.84   0.84   0.84   3542
weighted avg   0.84   0.84   0.84   3542
****

Accuracy score of SVC() is
0.932806324110672
[[1633 123]
[115 1671]]

	precision	recall	f1-score	support
0.0	0.93	0.93	0.93	1756
1.0	0.93	0.94	0.93	1786
accuracy			0.93	3542
macro avg	0.93	0.93	0.93	3542
weighted avg	0.93	0.93	0.93	3542

****

Accuracy score of DecisionTreeClassifier() is
0.9367588932806324
[[1639 117]
[107 1679]]

	precision	recall	f1-score	support
0.0	0.94	0.93	0.94	1756
1.0	0.93	0.94	0.94	1786
accuracy			0.94	3542
macro avg	0.94	0.94	0.94	3542
weighted avg	0.94	0.94	0.94	3542

****

Accuracy score of KNeighborsClassifier() is
0.9285714285714286
[[1553 203]
[50 1736]]

	precision	recall	f1-score	support
0.0	0.97	0.88	0.92	1756
1.0	0.90	0.97	0.93	1786
accuracy			0.93	3542
macro avg	0.93	0.93	0.93	3542
weighted avg	0.93	0.93	0.93	3542

****

Accuracy score of RandomForestClassifier() is
0.9686617730095991
[[1685 71]
[40 1746]]
precision recall f1-score support
0.0 0.98 0.96 0.97 1756
1.0 0.96 0.98 0.97 1786
accuracy 0.97 3542
macro avg 0.97 0.97 0.97 3542
weighted avg 0.97 0.97 0.97 3542
****
*

Accuracy score of GradientBoostingClassifier() is

0.9474872953133823
[[1664 92]
[94 1692]]
precision recall f1-score support
0.0 0.95 0.95 0.95 1756
1.0 0.95 0.95 0.95 1786 accuracy
0.95 3542 macro avg 0.95 0.95
0.95 3542 weighted avg 0.95 0.95
0.95 3542
****

Accuracy score of AdaBoostClassifier() is
0.9282891022021457
[[1627 129]
[125 1661]]

precision recall f1-score support
0.0 0.93 0.93 0.93 1756
1.0 0.93 0.93 0.93 1786
accuracy 0.93 3542
macro avg 0.93 0.93 0.93 3542
weighted avg 0.93 0.93 0.93 3542
****

Cross Validation Score:

The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting or selection bias and to give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem).
Now we check cross val score of the above models by the code below:

```
from sklearn.model_selection import cross_val_score
for i in range(2,10):
    print("For CV =",i)
    for m in model:
        scr=cross_val_score(m,x,y,cv=i)
        print("cross validation score of",m,"is =",scr.mean())
    print("\n")
    print('*'*125)
    print('*'*125)
    print("\n")
```

For CV = 2cross validation score of
LogisticRegression() is = 0.8449940708114518cross validation score of
GaussianNB() is = 0.8167880738607487cross validation score of
SVC() is = 0.8514314755209216cross validation score of
DecisionTreeClassifier() is = 0.7944265627646959cross validation score of
KNeighborsClassifier() is = 0.7996781297645266cross validation score of
RandomForestClassifier() is = 0.8320345586989666cross validation score of
GradientBoostingClassifier() is = 0.7519058106047772 cross validation score of
AdaBoostClassifier() is = 0.746569540911401
****
****
** For
CV = 3cross validation score of
LogisticRegression() is = 0.8543109489227039cross validation score of
GaussianNB() is = 0.8167029739706546cross validation score of
SVC() is = 0.8650652928860927cross validation score of
DecisionTreeClassifier() is = 0.7042087848610379cross validation score of
KNeighborsClassifier() is = 0.82085375753694 cross validation score of
RandomForestClassifier() is = 0.8820048777044315cross validation score of
GradientBoostingClassifier() is = 0.8383827388146816cross validation score of
AdaBoostClassifier() is = 0.7309728205287135
****
****
****
For CV = 4cross validation score of
LogisticRegression() is = 0.8589676722970212cross validation score of
GaussianNB() is = 0.820597795841564cross validation score of
SVC() is = 0.8779354513512942cross validation score of
DecisionTreeClassifier() is = 0.7890688207754664 cross validation score of
KNeighborsClassifier() is = 0.838635294498489cross validation score of

RandomForestClassifier() is = 0.8800487857682712cross validation score of
GradientBoostingClassifier() is = 0.8249905984742667cross validation score of
AdaBoostClassifier()   is = 0.809151438261248
****
****
****

For CV = 5cross validation score of
LogisticRegression() is = 0.8628653023428626cross validation score of
GaussianNB() is = 0.8208522558754471 cross validation score of
SVC() is = 0.8904786035854295cross validation score of
DecisionTreeClassifier() is = 0.8446503494371743cross validation score of
KNeighborsClassifier() is = 0.8590548645233851cross validation score of
RandomForestClassifier() is = 0.9202086832277686cross validation score of
GradientBoostingClassifier() is = 0.8632866281419668cross validation score of
AdaBoostClassifier() is = 0.8404157525926707
****
****
*** For
CV = 6 cross validation score of
LogisticRegression() is = 0.8629527274555918cross validation score of
GaussianNB() is = 0.818655167995503cross validation score of
SVC() is = 0.895390473986082cross validation score of
DecisionTreeClassifier() is = 0.8789478461002201cross validation score of
KNeighborsClassifier() is = 0.8432185773942131cross validation score of
RandomForestClassifier() is = 0.9231702753288337cross validation score of
GradientBoostingClassifier() is = 0.8643827420734805cross validation score of
AdaBoostClassifier() is = 0.8449022695615874
****
****
****

For CV = 7cross validation score of
LogisticRegression() is = 0.8687057552159535cross validation score of
GaussianNB() is = 0.8225469304982518cross validation score of
SVC() is = 0.9008042401260795cross validation score of
DecisionTreeClassifier() is = 0.9053758683964127cross validation score of
KNeighborsClassifier() is = 0.8675211730562783cross validation score of
RandomForestClassifier() is = 0.9399381435656321cross validation score of
GradientBoostingClassifier() is = 0.8962296987429517cross validation score of
AdaBoostClassifier() is = 0.879796929920652
****
****
****
****

For CV = 8cross validation score of
LogisticRegression() is = 0.8732028845712186cross validation score of
GaussianNB() is = 0.8226384869780902cross validation score of
SVC() is = 0.9031022920398696cross validation score of
DecisionTreeClassifier() is = 0.9191034518396031cross validation score of
KNeighborsClassifier() is = 0.8851439988976161cross validation score of
RandomForestClassifier() is = 0.9507003008589409cross validation score of
GradientBoostingClassifier() is = 0.9007280901198842cross validation score of
AdaBoostClassifier() is = 0.8853136052546966

****

For CV = 9cross validation score of
LogisticRegression() is = 0.8706622254056255cross validation score of
GaussianNB() is = 0.8228936696784969cross validation score of
SVC() is = 0.8983589969385581cross validation score of
DecisionTreeClassifier() is = 0.8834491322125273cross validation score of
KNeighborsClassifier() is = 0.880230910432674cross validation score of
RandomForestClassifier() is = 0.9345243447667122cross validation score of
GradientBoostingClassifier() is = 0.8687154269998079 cross validation score of
AdaBoostClassifier() is = 0.8679522629036359

Model Name	Accuracy Score	Cross Val Score (CV=8)
LogisticRegression	89.72%	87.32%
GaussianNB	84.30%	82.26%
SVC	93.28%	90.31%
DecisionTreeClassifier	93.67%	91.91%
KNeighborsClassifier	92.85%	88.51%
RandomForestClassifier	96.86%	95.07%
GradientBoostingClassifier	94.74%	90.07%
AdaBoostClassifier	92.82%	88.53%

Since *RandomForestClassifier* Classifier is giving best Accuracy and CV score, I choose *RandomForestClassifier* Classifier as best fitting model. Let's check whether we can increase the Accuracy score by using Hyper parameter tuning

Hyperparameter tuning of RandomForestClassifier:-

Parameters which define the model architecture are referred to as hyperparameters and thus this

```
#creating parameter list to pass in GreadSearchCV
parameters={'max_features':['auto','sqrt','log2'],'max_depth':[4,5,6,7,8],'criterion':['gini','entropy']}
gcv1=GridSearchCV(RandomForestClassifier(),parameters,cv=8,scoring='accuracy')
gcv1.fit(x_train,y_train)
gcv1.best_params_
```

```
{'criterion': 'gini', 'max_depth': 8, 'max_features': 'log2'}
```

```
gcv1.best_score_
```

```
0.9244917715392063
```

```
gcv1.best_estimator_
```

```
RandomForestClassifier(max_depth=8, max_features='log2')
```

process of searching for the ideal model architecture is referred to as hyperparameter tuning. There are two types of hyperparameter tuning - Grid Search CV and Randomised Search , here we'll use grid search cv for our further tuning. Read more about **Hyperparameter Tuning**. We did not get best accuracy score from

```
final_model.fit(x_train,y_train)
predm1=final_model.predict(x_test)
predm2=final_model.predict(x_train)
print('Test Accuracy score of final model =',accuracy_score(y_test,predm1))
print('Train Accuracy score of final model =',accuracy_score(y_train,predm2))
print(confusion_matrix(y_test,predm))
print(classification_report(y_test,predm))
```

```
Test Accuracy score of final model = 0.9435347261434218
Train Accuracy score of final model = 0.9532913843175218
[[1627  129]
 [ 125 1661]]
              precision    recall  f1-score   support

         0.0       0.93      0.93      0.93      1756
         1.0       0.93      0.93      0.93      1786

    accuracy                           0.93      3542
   macro avg       0.93      0.93      0.93      3542
weighted avg       0.93      0.93      0.93      3542
```

RandomForestClassifier after Parameter Tuning . But still we will create the final model with it to avoid underfit or overfit.

we can see that test accuracy score is very close to the test accracy score.so,our model is not underfit or overfit. so this our best model. We save this model for future prediction.

I have created an ROC curve plot and Confusion matrix for the final model.

➢ ROCAUC Curve:

AUC - ROC curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes.

The area under the ROC curve (AUC) results were considered excellent for AUC values between 0.9-1
There is a lot more to learn about Roc_Auc curve , visit the link attached below to explore about roc_auc curve . For now the ROC_AUC curve helps us visualize how well our machine learning classifier is performing. Go through this article to have in depth knowledge of **Roc_Auc curve**.

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
y_pred_prob=lg.predict_proba(x_test)[:,1]
y_pred_prob
```

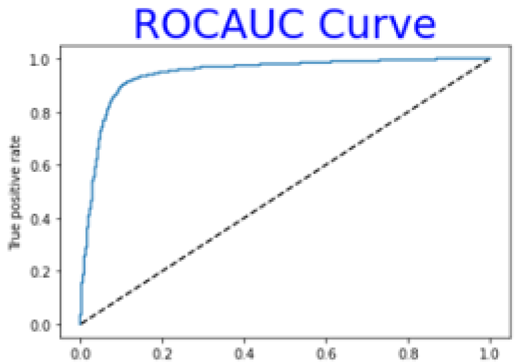
```
array([0.97459167, 0.41968848, 0.88112332, ..., 0.01515252, 0.7341639 ,
       0.62648329])
```

```
fpr,tpr,thresholds=roc_curve(y_test,y_pred_prob)
```

```
auc_score=roc_auc_score(y_test,final_model.predict(x_test))
print("roc_auc_score=",auc_score)
```

```
roc_auc_score= 0.9433680588326824
```

```
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr)
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROCAUC Curve',color='Blue',size=30)
plt.show()
```



This is the ROC curve for the final model RandomForest and AUC for RandomForest is 94% which is considered a very good score.

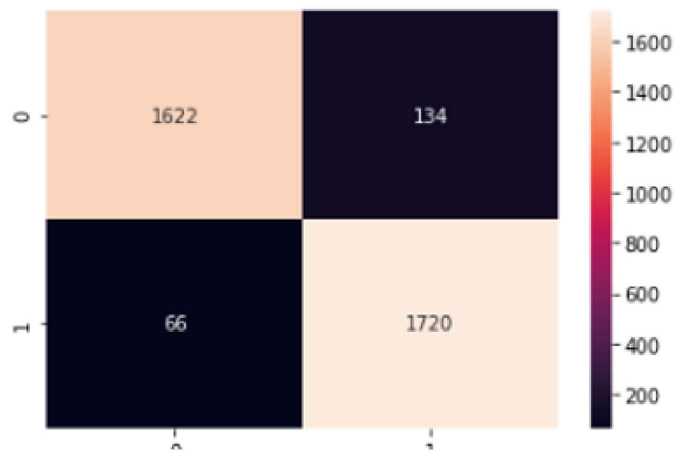
➢ ***Confusion Matrix:***

From our confusion matrix, we can calculate five different metrics measuring the validity of our model. **Accuracy (all correct / all) = TP + TN / TP + TN + FP + FN**. Misclassification (all incorrect / all) = FP + FN / TP + TN + FP + FN. Precision (true positives / predicted positives) = TP / TP + FP. A Confusion matrix is an N x N matrix used for **evaluating the performance of a classification model**, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model.

Read more about **confusion matrix**.

```
confusion_matrix(y_test,predm1)
sns.heatmap(confusion_matrix(y_test,predm1),annot=True, fmt='d')
```

```
<AxesSubplot:>
```



From the above confusion matrix we can see that our dataset is having 1622 as True positive and 1720 as True negative where as 134 as True negative and 66 as True negative value present in our dataset .

Final Model Applying On Test Data:

We can see from the above observation, Predicted and the Original values are matching .

```
MLdf=pd.DataFrame([final_model.predict(x_test)[:],y_test[:]],index=["Predicted","Original"]).T
MLdf
```

	Predicted	Original
0	1.0	1.0
1	0.0	0.0
2	1.0	1.0
3	1.0	1.0
4	1.0	1.0
...	...	...
3537	1.0	1.0
3538	0.0	0.0
3539	0.0	0.0
3540	1.0	1.0
3541	1.0	1.0

3542 rows × 2 columns

each other, which means model performance is good.

Saving Final Model:

There are various ways through which we can save a machine learning model , in here we are using joblib to save our best model .

Read more about saving a machine learning model **here**

```
#Save the final model
import joblib
joblib.dump(final_model,'Rainfall Prediction Model.pkl')
```

```
['Rainfall Prediction Model.pkl']
```

Conclussion:-

- This particular problem needs a good vision on data, and in this problem Feature Engineering is the most crucial thing.

You can see how we have handled numerical and categorical data and how we build different machine learning models on the same dataset.

It is always advised to all of us that at least we need to use 5 Algorithm in order to figure out which one is performing best among them and we choose that one and we send that for hyper parameter tuning to know that best parameter.

Using hyper parameter tunning we can improve our model accuracy, for instance in this model the accuracy remained same.

- Weather forecasts are increasingly accurate and useful, and their benefits extend widely across the economy. While much has been accomplished in improving weather forecasts, there remains much room for improvement.

- For future improvements, following step we thought to took-
- Replacing model with a latest/different model
- Using other robust datasets
- Predicting result on more attributes
- Training model on higher-end GPU

- Also, while performing weather forecasting, there was a lot of complexities involved. There are a lot of variables/attributes to consider for forecasting weather and if all or most of them are used, then we need a lot of computation power to get weather information. And, Real time weather forecasting is very difficult to forecast correctly.

❖ For any of machine learning project my suggestion is first you have to understand the problem on ground level. if you don’t allow yourself to work with diligence. if you don’ t work harder anything that you are doing or will do , not only in case of machine learning but also in life cycle would be futile. Maybe, my endeavor assists you whenever you will get stuck.