Rainfall Prediction - Weather Forecasting

Submitted By

Amruta Shah

Problem Statement:

Weather forecasting is the application of science and technology to predict the conditions of the atmosphere for a given location and time. Weather forecasts are made by collecting quantitative data about the current state of the atmosphere at a given place and using meteorology to project how the atmosphere will change.
Rain Dataset is to predict whether or not it will rain tomorrow. The Dataset contains about 10 years of daily weather observations of different locations in Australia. Here, predict two things:

Problem Statements:

a) Design a predictive model with the use of machine learning algorithms to forecast whether or not it will rain tomorrow.

b) Design a predictive model with the use of machine learning algorithms to predict how much rainfall could be there.

Dataset Description:

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".

Methodology: The steps followed in this work, right from the dataset preparation to obtaining results are represented in Fig.

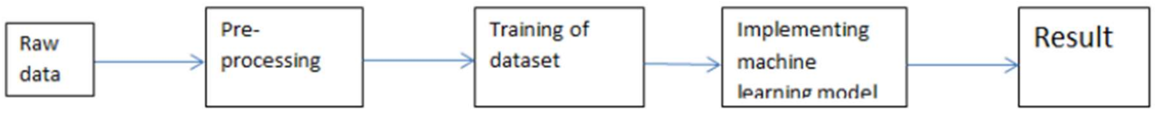


Fig1: Steps followed for obtaining results

Data Analysis:

In this project, we have a dataset which has the details of the weather forecasting to predict the conditions of the atmosphere for a given location and time.

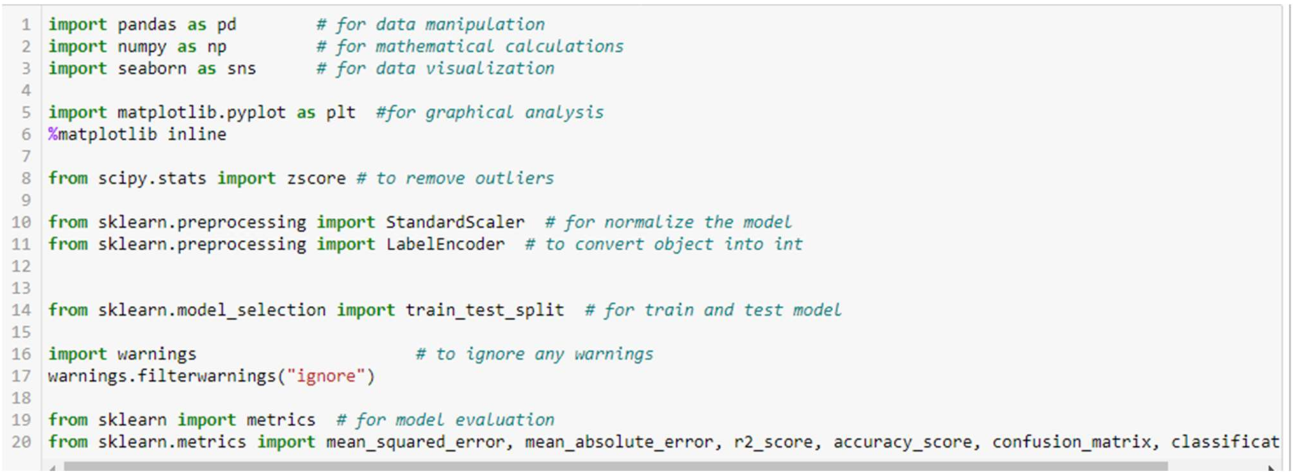
The given dataset contains 8425 rows × 23 columns. The column names like Date, Location, MinTemp, MaxTemp, Rainfall, Evaporation, Sunshine, WindGustDir, WindGustSpeed, WindDir9am, Humidity9am, Humidity3pm, Pressure9am, Pressure3pm, Cloud9am,Cloud3pm,Temp9am, Temp3pm,RainToday,RainTomorrow,etc.

EDA (Exploratory Data Analysis):

We should first perform an EDA as it will connect us with the dataset at an emotional level and yes, of course, will help in building good hypothesis function. EDA is a very crucial step. It gives us a glimpse of what our data set is all about, its uniqueness, its anomalies and finally it summarizes the main characteristics of the dataset for us. In order to perform EDA, we will require the following python packages.

Import libraries:

Let's use collected data set to solve the problem. For that we need to import some necessary python libraries.

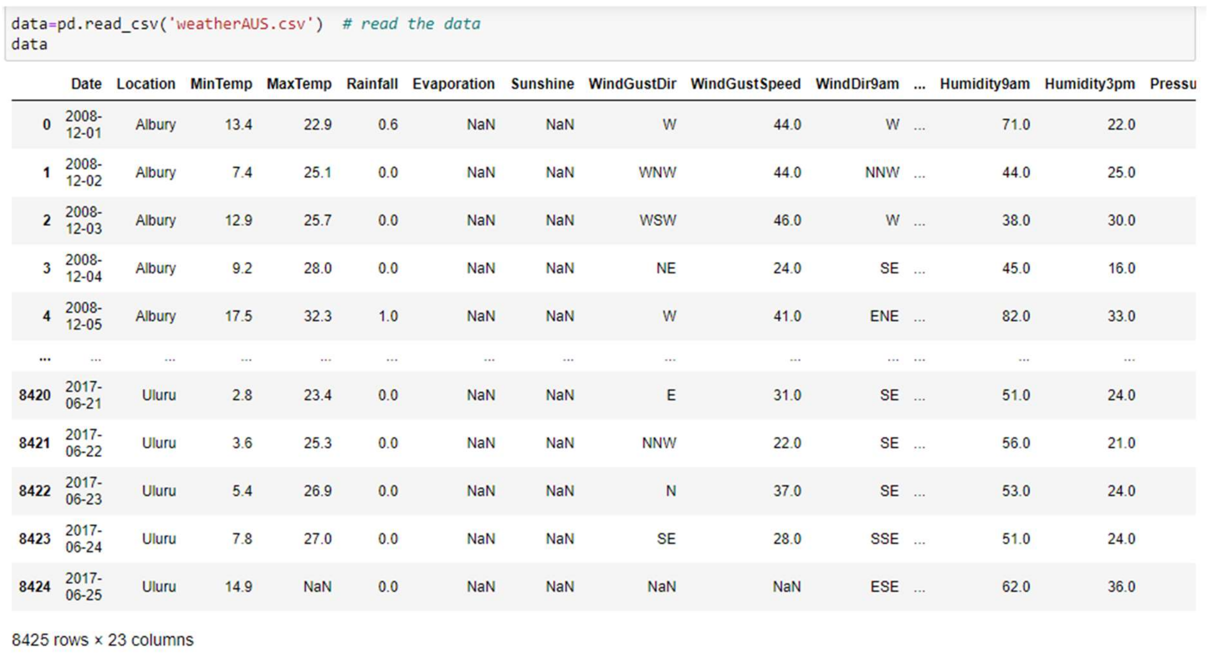


```
import pandas as pd        # for data manipulation
import numpy as np         # for mathematical calculations
import seaborn as sns      # for data visualization

import matplotlib.pyplot as plt  #for graphical analysis
%matplotlib inline

from scipy.stats import zscore # to remove outliers

from sklearn.preprocessing import StandardScaler  # for normalize the model
from sklearn.preprocessing import LabelEncoder  # to convert object into int


from sklearn.model_selection import train_test_split  # for train and test model

import warnings                      # to ignore any warnings
warnings.filterwarnings("ignore")

from sklearn import metrics  # for model evaluation
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, accuracy_score, confusion_matrix, classificat
```

Once we have imported the packages successfully, we will move on to importing our dataset.

Load Dataset:

This collected data is in the .csv form so we need to use Pandas. read method to read the data.



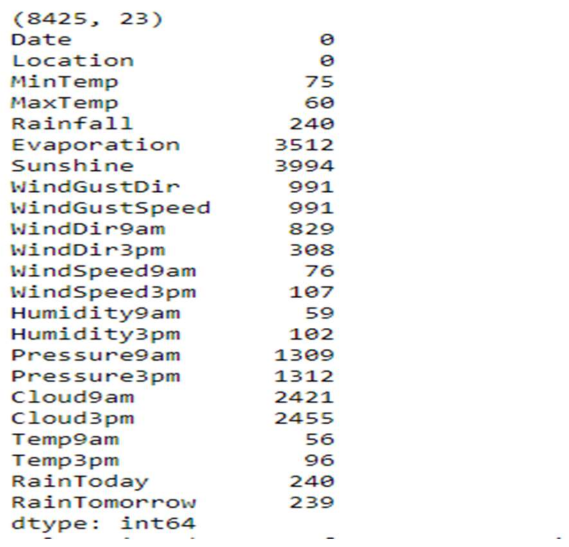
```
data=pd.read_csv('weatherAUS.csv')  # read the data
data
```

	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	

8425 rows × 23 columns

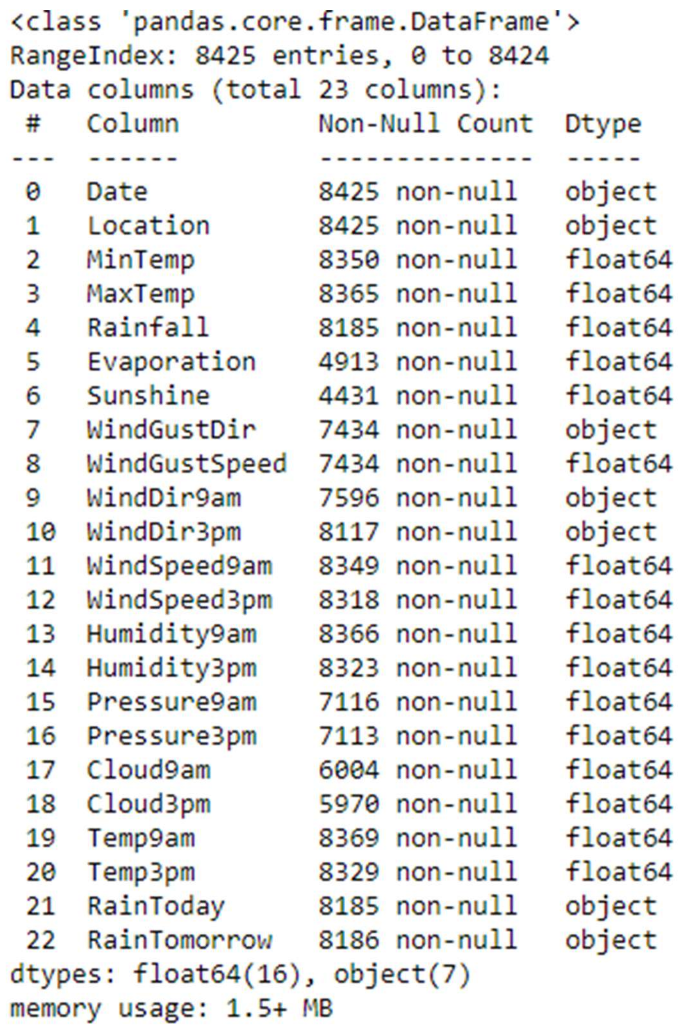
Dimensions of Dataset:

The dataset has been successfully imported. Let's have a look at the dataset. head () gives us a glimpse of the dataset. It can be considered similar to select * from database table limit 5 in SQL. Let's go ahead and explore a little bit more about the different fields in the dataset. info () gives uses all the relevant information on the dataset. If your dataset has more numerical variables, consider using describe () too to summarize data along mean, median, standard variance, variance, unique values, frequency etc. isna (). sum () gives us sum of null values are present in the dataset. See below screenshot.



```
(8425, 23)
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
```

The number of null values present is as above.



```
<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
```

Dataset has object as well as numeric types. Object type in pandas is similar to strings. Now let's try to classify these columns as Categorical, Ordinal or Numerical/Continuous.

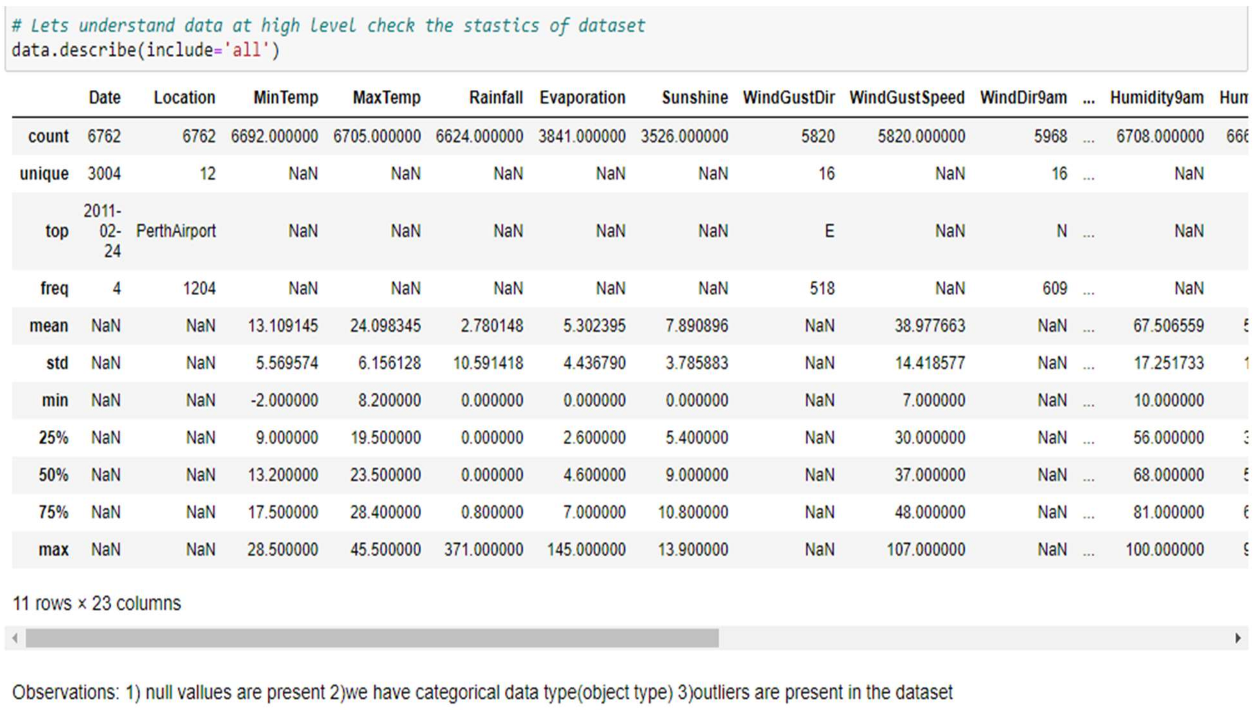
Categorical Variables:
Categorical variables are those data fields that can be divided into definite groups. In this case, RainToday & RainTomorrow (Yes OR No) is categorical variables.

Ordinal Variables:
Ordinal variables are the ones that can be divided into groups, but these groups have some kind of order. Like, high, medium, low. Dependents field can be considered ordinal since the data can be clearly divided into 4 categories: 0, 1, 2, 3 and there is a definite ordering also. In this case we have direction like S, N, S, SN, SW, W, etc.

Numerical or Continuous Variables:
Numerical variables are those that can take up any value within a given range. In this case Date, Location, MinTemp, MaxTemp, Rainfall, Evaporation, Sunshine, WindGustDir, WindGustSpeed, WindDir9am, Humidity9am, Humidity3pm, Pressure9am, Pressure3pm, Cloud9am,Cloud3pm,Temp9am, Temp3pm,& many.

Statistical Summary:

In the raw data, there can be various types of underlying patterns which also gives an in-depth knowledge about subject of interest and provides insights about the problem. But caution should be observed with respect to data as it may contain null values, or redundant values, or various types of ambiguity, which also demands for pre-processing of data. Dataset should therefore be explored as much as possible. Various factors important by statistical means like mean, standard deviation, median, count of values and maximum value etc. are shown in Fig below.



```
# Lets understand data at high level check the stastics of dataset
data.describe(include='all')
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity9am	Hum
count	6762	6762	6692.000000	6705.000000	6624.000000	3841.000000	3526.000000	5820	5820.000000	5968	...	6708.000000	666
unique	3004	12	NaN	NaN	NaN	NaN	NaN	16	NaN	16	...	NaN	
top	2011-02-24	PerthAirport	NaN	NaN	NaN	NaN	NaN	E	NaN	N	...	NaN	
freq	4	1204	NaN	NaN	NaN	NaN	NaN	518	NaN	609	...	NaN	
mean	NaN	NaN	13.109145	24.098345	2.780148	5.302395	7.890896	NaN	38.977663	NaN	...	67.506559	5
std	NaN	NaN	5.569574	6.156128	10.591418	4.436790	3.785883	NaN	14.418577	NaN	...	17.251733	1
min	NaN	NaN	-2.000000	8.200000	0.000000	0.000000	0.000000	NaN	7.000000	NaN	...	10.000000	
25%	NaN	NaN	9.000000	19.500000	0.000000	2.600000	5.400000	NaN	30.000000	NaN	...	56.000000	3
50%	NaN	NaN	13.200000	23.500000	0.000000	4.600000	9.000000	NaN	37.000000	NaN	...	68.000000	5
75%	NaN	NaN	17.500000	28.400000	0.800000	7.000000	10.800000	NaN	48.000000	NaN	...	81.000000	6
max	NaN	NaN	28.500000	45.500000	371.000000	145.000000	13.900000	NaN	107.000000	NaN	...	100.000000	9

11 rows × 23 columns

Observations: 1) null vallues are present 2)we have categorical data type(object type) 3)outliers are present in the dataset

Data Visualization:

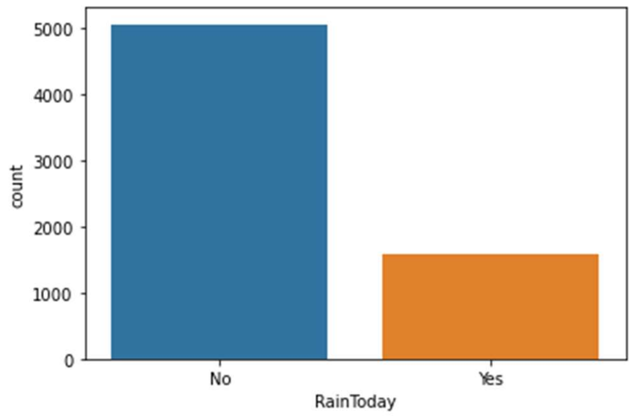
We now have a basic idea about the data. We need to extend that with some visualizations. We are going to look at three types of plots:

1.Univariate plots to better understand each variable.
2.Bivariate plots to find relationship between two variables,
3.Multivariate plots to better understand the relationships between variables.

Univariate Plots:

We start with some univariate plots, that is, plots of each individual variable. Given that the input variables are numeric, we can create box or count plots of each. Now we are all set to perform Univariate Analysis. Univariate analysis involves analysis of one variable at a time. Let's say "RainToday" then we will analyse only the "RainToday" field in the dataset. The analysis is usually summarized in the form of count. For visualization, we have many options such as frequency tables, bar graphs, pie charts, histograms etc. We will stick to count charts.

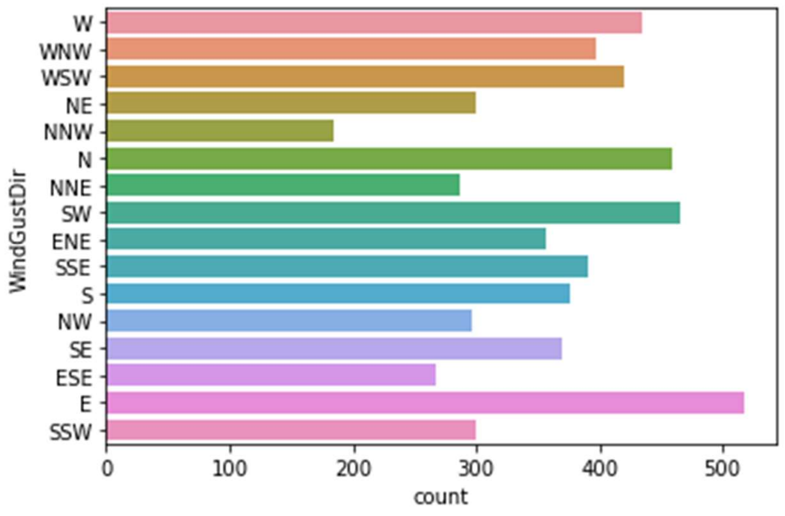
```
#plot each class frequency
sns.countplot(x='RainToday',data=data)
plt.show()
print(data['RainToday'].value_counts())
```



```
No      5052
Yes     1572
Name: RainToday, dtype: int64
```

The count of no means precipitation (mm) in the 24 hours to 9am not exceeds 1mm is more.

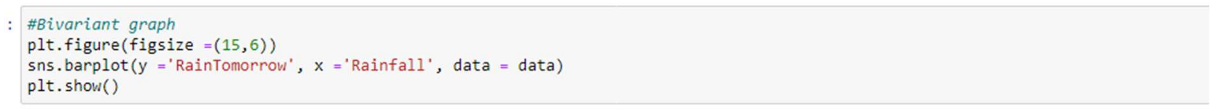
```
#plot each class frequency
sns.countplot(y='WindGustDir',data=data)
plt.show()
print(data['WindGustDir'].value_counts())
```



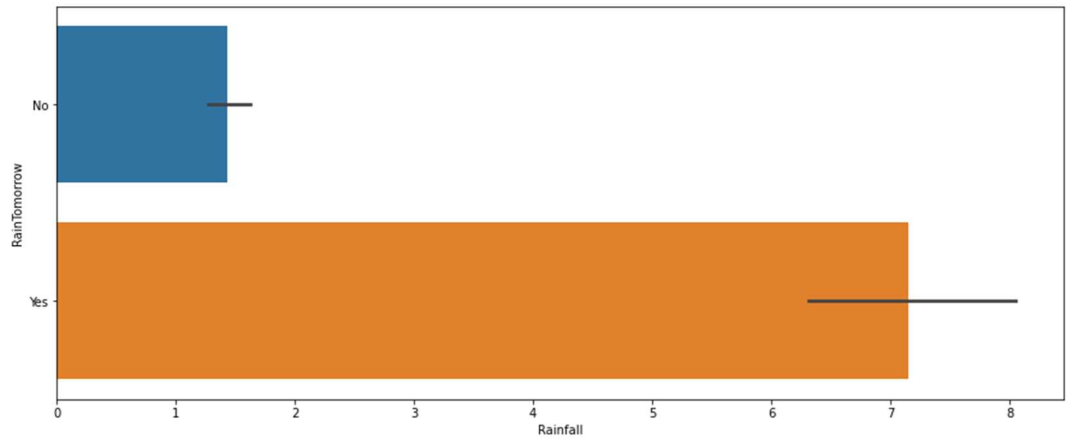
Similarly, I have plotted for the other independent variables and concluded.

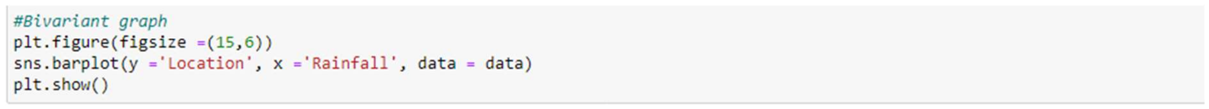
Bivariate Plot:

Now let's find some relationship between two variables, particularly between the target variable and a predictor variable from the dataset. Formally, this is known as bivariate analysis. Hear we have two target variable one is Rainfall & other is RainTomorrow, so let's check this relationship between two variables and other independent variables. For visualization, we will be using seaborn. countplot (). It can be considered similar to the histogram for categorical variables

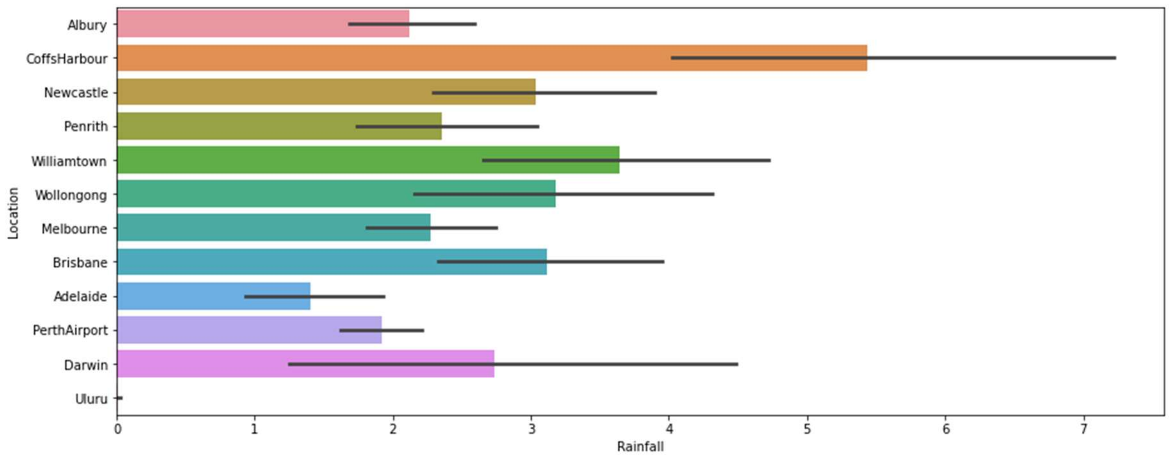


```
: #Bivariant graph
plt.figure(figsize =(15,6))
sns.barplot(y ='RainTomorrow', x ='Rainfall', data = data)
plt.show()
```





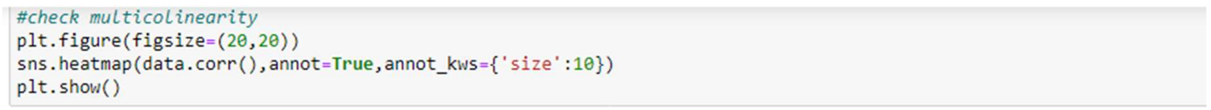
```
#Bivariant graph
plt.figure(figsize =(15,6))
sns.barplot(y ='Location', x ='Rainfall', data = data)
plt.show()
```



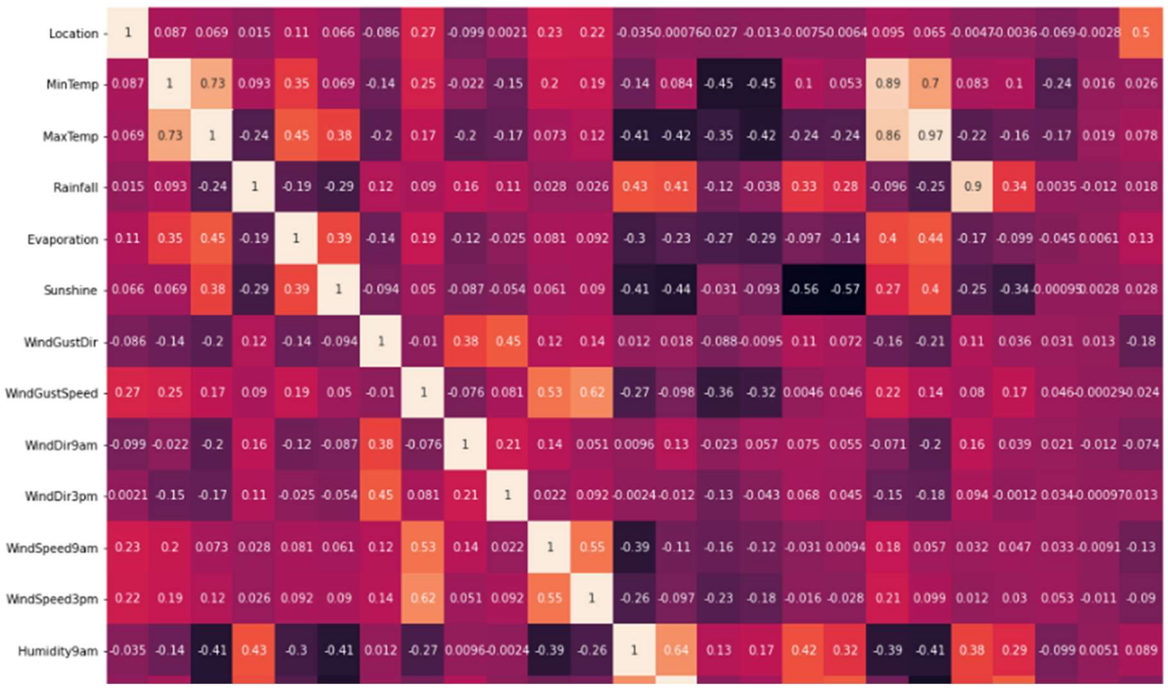
We can see the maximum rainfall is done in CoffsHarbour station.

Multivariate Plot:

Let's move on to analysing more than two variables now. We call it "Multivariate analysis". Now we can look at the interactions between the variables. First, let's look at scatterplots of all pairs of attributes. This can be helpful to spot structured relationships between input variables by using heatmap. Let's visualize the data in this correlation matrix using a heat map.



```
#check multicolinearity
plt.figure(figsize=(20,20))
sns.heatmap(data.corr(),annot=True,annot_kws={'size':10})
plt.show()
```



Data pre-processing:

Pre-processing of this dataset includes doing analysis on the independent variables like checking for null values in each column and then replacing or filling them with supported appropriate data types like mean, mode method or Imputer methods, so that analysis and model fitting is not hindered from its way to accuracy.

Shown above are some of the representations obtained by using Pandas tools which tells about variable count for numerical columns and model values for categorical columns. Maximum and minimum values in numerical columns, along with their percentile values for median, plays an important factor in deciding which value to be chosen at priority for further exploration tasks and analysis.

Data types of different columns are used further in label processing and Label encoding scheme during model building.

There are many stages involved in data pre-processing.

Data cleaning attempts to impute missing values, removing outliers from the dataset.

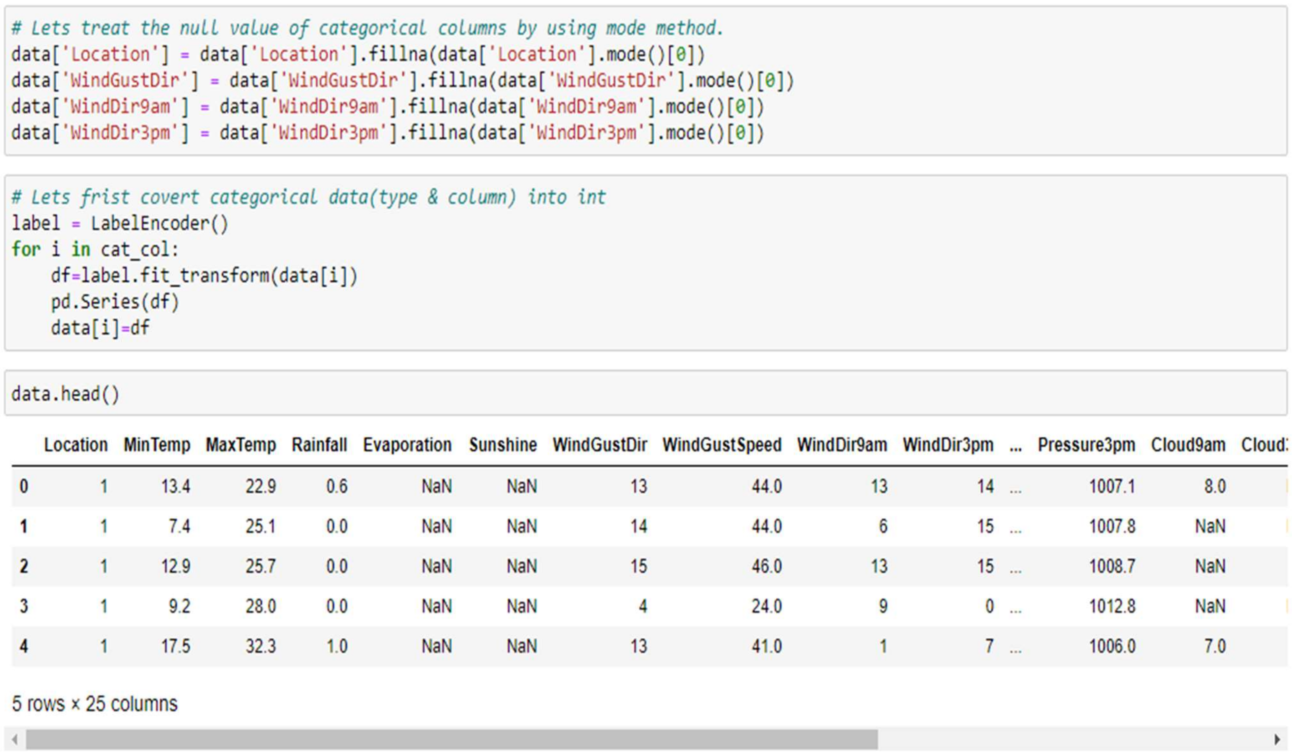
Data integration integrates data from a multitude of sources into a single data warehouse.

Data transformation such as normalization, may be applied. For example, normalization may improve the accuracy and efficiency of mining algorithms involving distance measurement.

Data reduction can reduce the data size by dropping out redundant features. Feature selection and feature extraction techniques can be used.

As in our dataset have null values & objective data type so let’s use Imputer techniques & Encoding techniques to convert data into numerical form as shown below.

Label Encoder refers to converting the labels into numeric form so as to convert it into the machine-readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning. Label encoding in python can be imported from the Sklearn library. Sklearn provides a very efficient tool for encoding. Label encoders encode labels with a value between 0 and n_classes-1.



```python
# Lets treat the null value of categorical columns by using mode method.
data['Location'] = data['Location'].fillna(data['Location'].mode()[0])
data['WindGustDir'] = data['WindGustDir'].fillna(data['WindGustDir'].mode()[0])
data['WindDir9am'] = data['WindDir9am'].fillna(data['WindDir9am'].mode()[0])
data['WindDir3pm'] = data['WindDir3pm'].fillna(data['WindDir3pm'].mode()[0])
```

```python
# Lets frist covert categorical data(type & column) into int
label = LabelEncoder()
for i in cat_col:
    df=label.fit_transform(data[i])
    pd.Series(df)
    data[i]=df
```

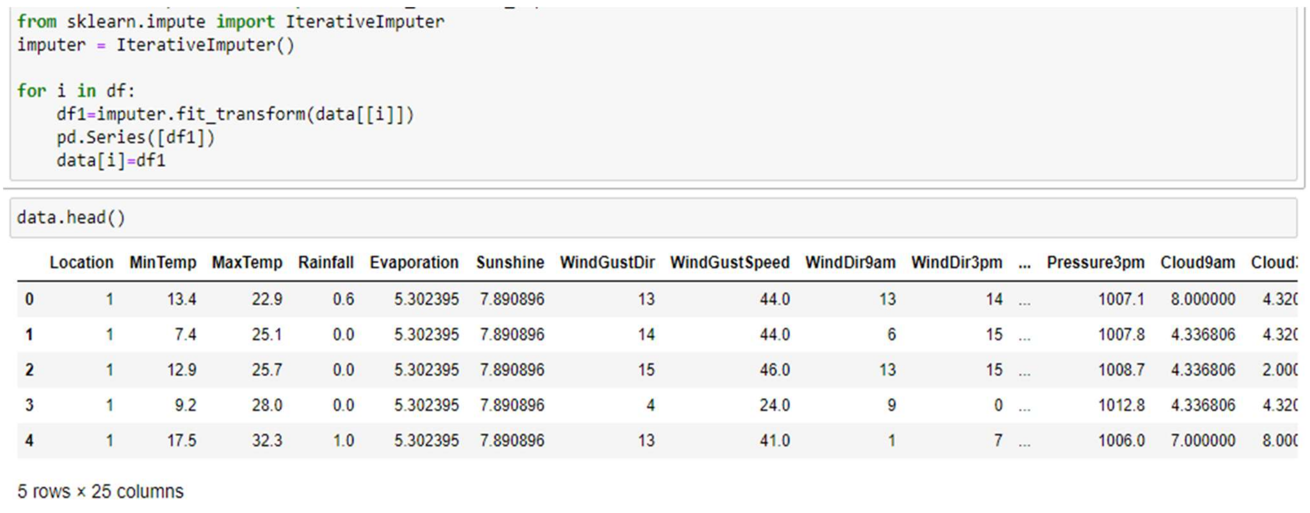
```python
data.head()
```

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	...	Pressure3pm	Cloud9am	Cloud:
0	1	13.4	22.9	0.6	NaN	NaN	13	44.0	13	14	...	1007.1	8.0	
1	1	7.4	25.1	0.0	NaN	NaN	14	44.0	6	15	...	1007.8	NaN	
2	1	12.9	25.7	0.0	NaN	NaN	15	46.0	13	15	...	1008.7	NaN	
3	1	9.2	28.0	0.0	NaN	NaN	4	24.0	9	0	...	1012.8	NaN	
4	1	17.5	32.3	1.0	NaN	NaN	13	41.0	1	7	...	1006.0	7.0	

5 rows × 25 columns

Iterative imputer is a hidden gem of the sklearn library in python.

The iterative imputer library provides us with tools to tackle the problem mentioned above. Instead of just replacing values with mean/median, we can have a regressor (Linear/Decision Tree/Random Forest/KNN) to impute missing values. By using the iterative imputer we can intelligently impute the missing values, avoid bias, maintain the relationship between variables, and can get better results.



```
from sklearn.impute import IterativeImputer
imputer = IterativeImputer()

for i in df:
    df1=imputer.fit_transform(data[[i]])
    pd.Series([df1])
    data[i]=df1
```

```
data.head()
```

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	...	Pressure3pm	Cloud9am	Cloud
0	1	13.4	22.9	0.6	5.302395	7.890896	13	44.0	13	14	...	1007.1	8.000000	4.32
1	1	7.4	25.1	0.0	5.302395	7.890896	14	44.0	6	15	...	1007.8	4.336806	4.32
2	1	12.9	25.7	0.0	5.302395	7.890896	15	46.0	13	15	...	1008.7	4.336806	2.00
3	1	9.2	28.0	0.0	5.302395	7.890896	4	24.0	9	0	...	1012.8	4.336806	4.32
4	1	17.5	32.3	1.0	5.302395	7.890896	13	41.0	1	7	...	1006.0	7.000000	8.00

5 rows × 25 columns

Also , as I say data pre-processing is a predominant step in machine learning to yield highly accurate and insightful results. Greater the quality of data, the greater is the reliability of the produced results.

Incomplete, noisy, and inconsistent data are the inherent nature of real-world datasets. Data pre-processing helps in increasing the quality of data by filling in missing incomplete data, smoothing noise, and resolving inconsistencies.

Incomplete data can occur due to many reasons. Appropriate data may not be persisted due to a misunderstanding, or because of instrument defects and malfunctions.

Noisy data can occur for a number of reasons (having incorrect feature values). The instruments used for the data collection might be faulty. Data entry may contain human or instrument errors. Data transmission errors might occur as well.

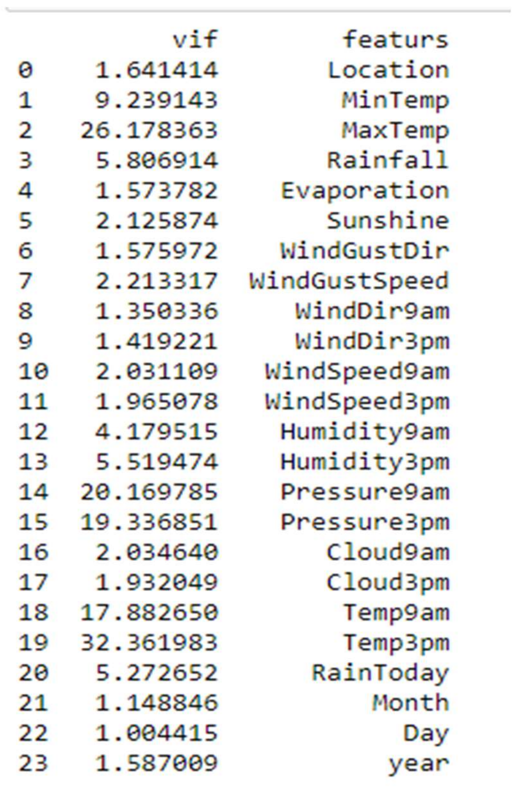
Outliers are data points that are distant from other similar points. They may be due to variability in the measurement or may indicate experimental errors. If possible, outliers should be excluded from the data set. However, detecting that anomalous instance might be very difficult, and is not always possible.

Methods we used to remove outliers from our dataset is:

Z-score — Call scipy.stats.zscore() with the given data-frame as its argument to get a numpy array containing the z-score of each value in a dataframe. Call numpy.abs() with the previous result to convert each element in the dataframe to its absolute value. Use the syntax (array < 3).all(axis=1) with array as the previous result to create a boolean array.

Correlations among variables:

Heatmap was plotted for variables with correlation coefficient. The Variance Inflation Factor (VIF) measures the severity of multicollinearity in regression analysis. It is a statistical concept that indicates the increase in the variance of a regression coefficient as a result of collinearity. Variance inflation factor (VIF) is used to detect the severity of multicollinearity in the ordinary least square (OLS) regression analysis. Multicollinearity inflates the variance and type II error. It makes the coefficient of a variable consistent but unreliable. VIF measures the number of inflated variances caused by multicollinearity. Checking correlation between dependent and independent variables.



```
          vif        featurs
0    1.641414       Location
1    9.239143        MinTemp
2   26.178363        MaxTemp
3    5.806914       Rainfall
4    1.573782    Evaporation
5    2.125874       Sunshine
6    1.575972    WindGustDir
7    2.213317  WindGustSpeed
8    1.350336     WindDir9am
9    1.419221     WindDir3pm
10   2.031109   WindSpeed9am
11   1.965078   WindSpeed3pm
12   4.179515    Humidity9am
13   5.519474    Humidity3pm
14  20.169785    Pressure9am
15  19.336851    Pressure3pm
16   2.034640       Cloud9am
17   1.932049       Cloud3pm
18  17.882650        Temp9am
19  32.361983        Temp3pm
20   5.272652      RainToday
21   1.148846          Month
22   1.004415            Day
23   1.587009           year
```

As we see the values of above predictors is more than 10, so we dropped MaxTemp','Pressure9am','Pressure3pm','Temp9am', 'Temp3pm' the same & reduce the collinearity.

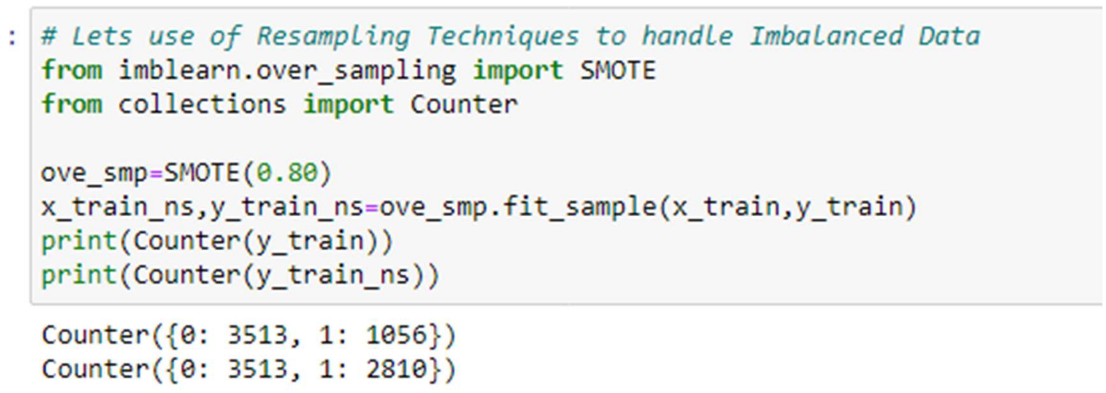
Balancing our imbalanced data:

There are different algorithms present to balance the target variable. We use the SMOTE() algorithm to make our data balance.

NOTE: SMOTE(Synthetic minority oversampling technique) works by randomly picking a point from the minority class and computing the k-nearest neighbors of this point. The synthetic points are added between the chosen point and its neighbours.

SMOTE algorithm works in 4 simple steps:

- Choose a minority class as input vector.
- Find its k-nearest neighbors.
- Choose one of these neighbors and place a synthetic point anywhere on the line joining the point under consideration and its chosen neighbors.
- Repeat the step until the data is balanced.



```
# Lets use of Resampling Techniques to handle Imbalanced Data
from imblearn.over_sampling import SMOTE
from collections import Counter

ove_smp=SMOTE(0.80)
x_train_ns,y_train_ns=ove_smp.fit_sample(x_train,y_train)
print(Counter(y_train))
print(Counter(y_train_ns))
```

```
Counter({0: 3513, 1: 1056})
Counter({0: 3513, 1: 2810})
```

The SMOTE algorithm balances our data with the highest number of values present in it.

Building machine learning models:

For building machine learning models there are several models present inside the Sklearn module.

Sklearn provides two types of models i.e., regression and classification. Our datasets contain both the target variable **first is to predict rain will came or not** and **second is how much rainfall**. So, for this kind of problem, we use classification as well as regression models. But before fitting our dataset to its model first we have to separate the predictor variable and the target variable, then scaled the independent variables or predictors by using StandardScaler method and then we pass this variable to the train_test_split method to create a random test and train subset.

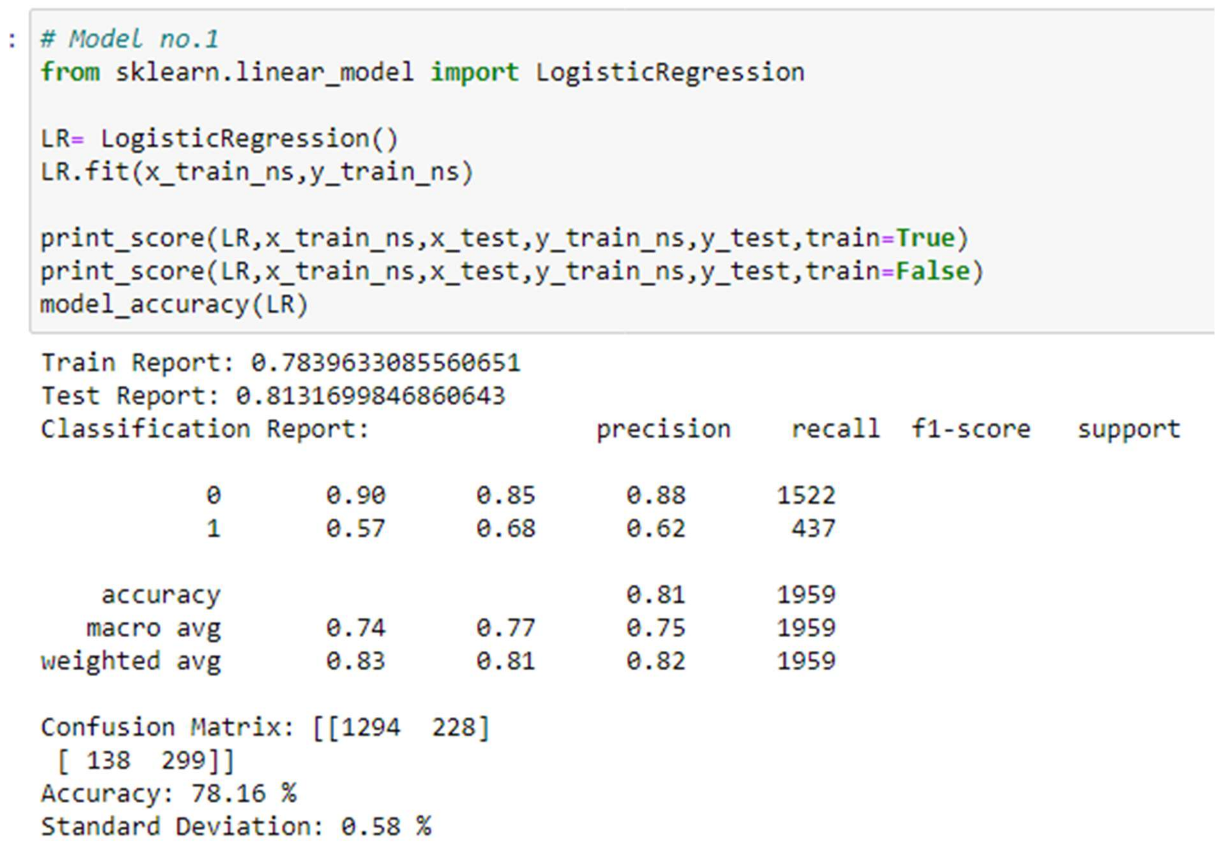
What is train_test_split, it is a function in sklearn model selection for splitting data arrays into two subsets for training data and testing data. With this function, you don't need to divide the dataset manually. By default, sklearn train_test_split will make random partitions for the two subsets.

However, you can also specify a random state for the operation. It gives us four outputs x_train, x_test, y_train and y_test. The x_train and x_test contains the training and testing predictor variables while y_train and y_test contains the training and testing target variable. After performing train_test_split we have to choose the models to pass the training variable.

We can build as many models as we want to compare the accuracy given by these models and to select the best model among them.

I have selected 5 models for the first problem statement that is prediction of RainTomarrow:

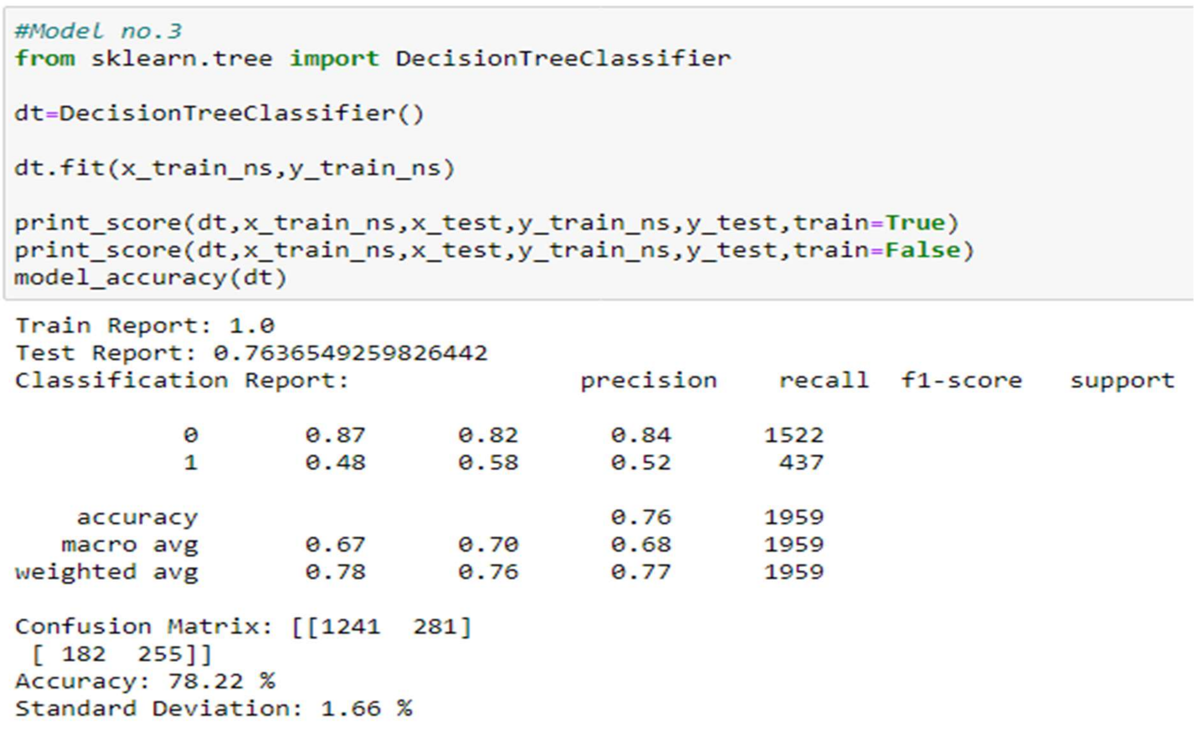
1) **Logistic Regression from sklearn.linear_model:** Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of target or dependent variable is binary, which means there would be only two possible classes 1 (stands for success/yes) or 0 (stands for failure/no). Mathematically, a logistic regression model predicts P(Y=1) as a function of X. It is one of the simplest ML algorithms that can be used for various classification problems such as spam detection, Diabetes prediction, cancer detection etc.



```
: # Model no.1
  from sklearn.linear_model import LogisticRegression

  LR= LogisticRegression()
  LR.fit(x_train_ns,y_train_ns)

  print_score(LR,x_train_ns,x_test,y_train_ns,y_test,train=True)
  print_score(LR,x_train_ns,x_test,y_train_ns,y_test,train=False)
  model_accuracy(LR)
```

```
Train Report: 0.7839633085560651
Test Report: 0.8131699846860643
Classification Report:               precision    recall  f1-score   support

           0       0.90      0.85      0.88      1522
           1       0.57      0.68      0.62       437

    accuracy                           0.81      1959
   macro avg       0.74      0.77      0.75      1959
weighted avg       0.83      0.81      0.82      1959

Confusion Matrix: [[1294  228]
 [ 138  299]]
Accuracy: 78.16 %
Standard Deviation: 0.58 %
```

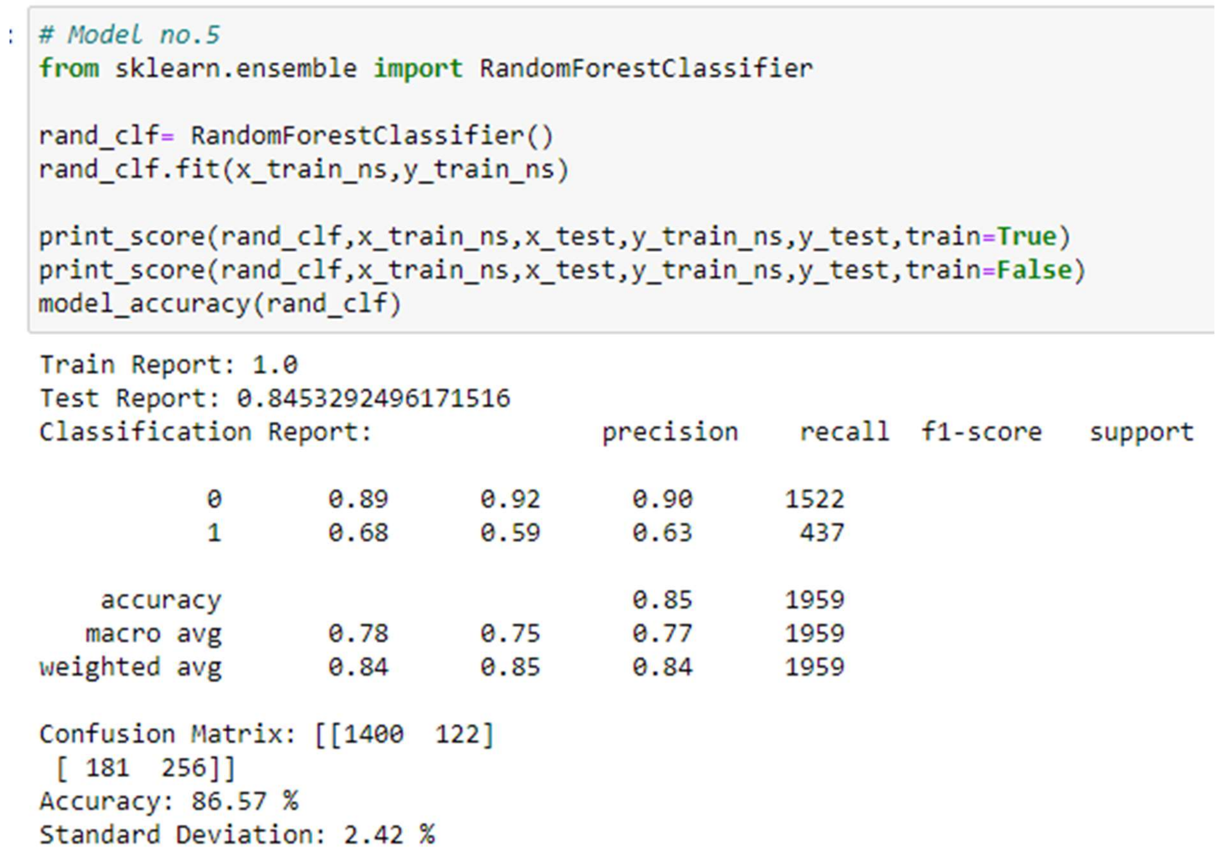
2) **DecisionTreeClassifier from sklearn.tree:** Decision trees can be constructed by an algorithmic approach that can split the dataset in different ways based on different conditions. The two main entities of a tree are decision nodes, where the data is split and leaves, where we get the outcome.



```
#Model no.3
from sklearn.tree import DecisionTreeClassifier

dt=DecisionTreeClassifier()

dt.fit(x_train_ns,y_train_ns)

print_score(dt,x_train_ns,x_test,y_train_ns,y_test,train=True)
print_score(dt,x_train_ns,x_test,y_train_ns,y_test,train=False)
model_accuracy(dt)
```

```
Train Report: 1.0
Test Report: 0.7636549259826442
Classification Report:               precision    recall  f1-score   support

           0       0.87      0.82      0.84      1522
           1       0.48      0.58      0.52       437

    accuracy                           0.76      1959
   macro avg       0.67      0.70      0.68      1959
weighted avg       0.78      0.76      0.77      1959

Confusion Matrix: [[1241  281]
 [ 182  255]]
Accuracy: 78.22 %
Standard Deviation: 1.66 %
```

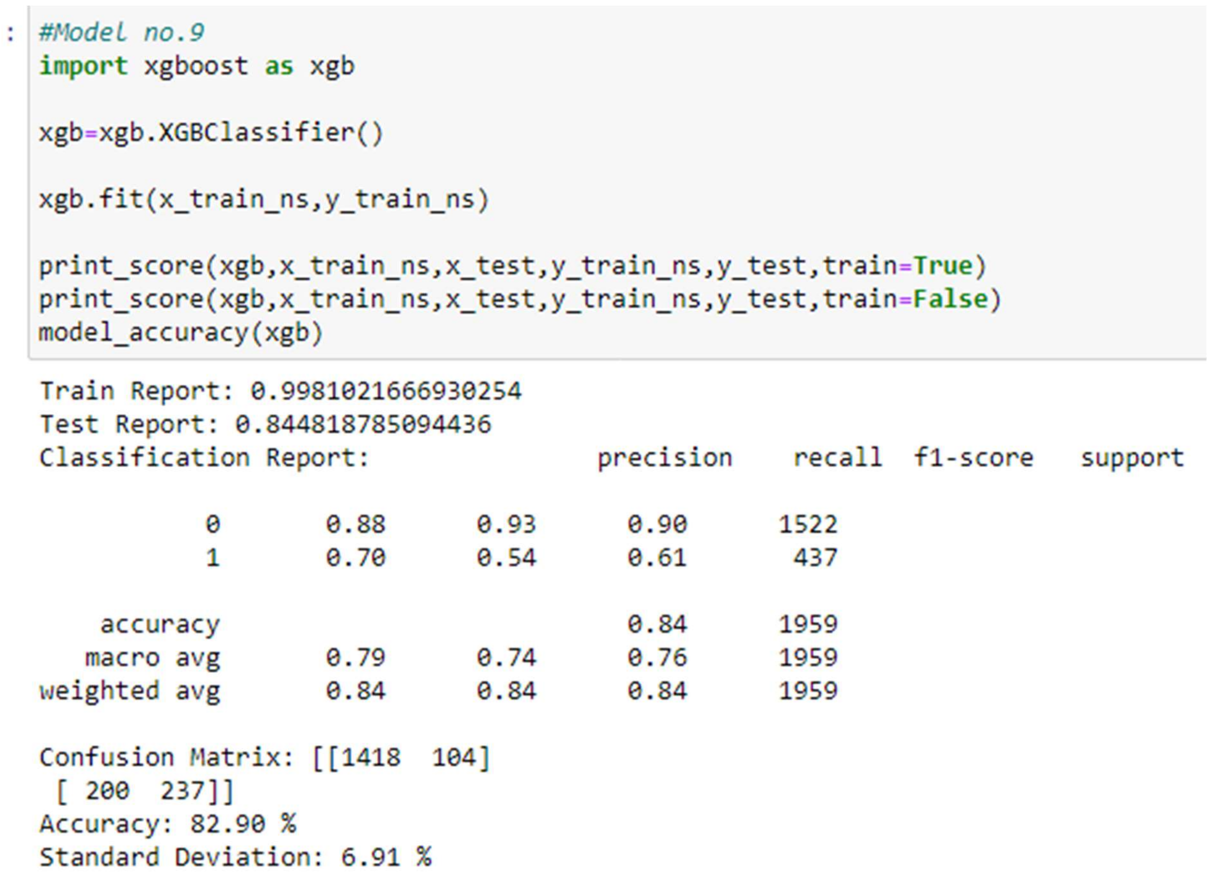
3) **RandomForestClassifier from sklearn.ensemble:** As we know that a forest is made up of trees and more trees means more robust forest. Similarly, a random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.



```
# Model no.5
from sklearn.ensemble import RandomForestClassifier

rand_clf= RandomForestClassifier()
rand_clf.fit(x_train_ns,y_train_ns)

print_score(rand_clf,x_train_ns,x_test,y_train_ns,y_test,train=True)
print_score(rand_clf,x_train_ns,x_test,y_train_ns,y_test,train=False)
model_accuracy(rand_clf)
```

```
Train Report: 1.0
Test Report: 0.8453292496171516
Classification Report:               precision    recall  f1-score   support

           0       0.89      0.92      0.90      1522
           1       0.68      0.59      0.63       437

    accuracy                           0.85      1959
   macro avg       0.78      0.75      0.77      1959
weighted avg       0.84      0.85      0.84      1959

Confusion Matrix: [[1400  122]
 [ 181  256]]
Accuracy: 86.57 %
Standard Deviation: 2.42 %
```

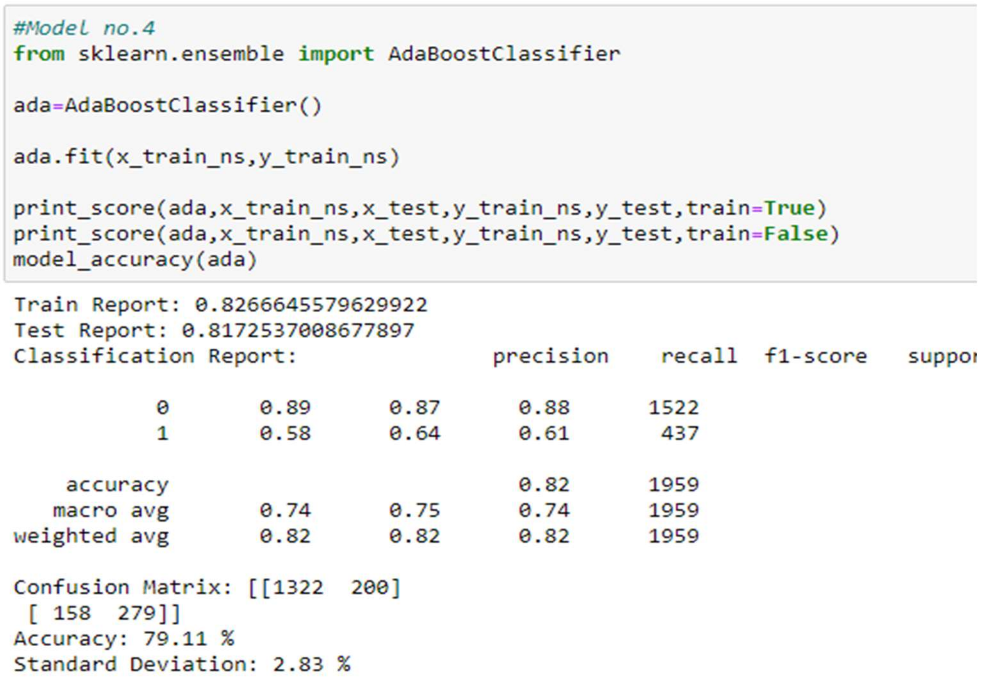
4) **XGBClassifier from XGBoost:** XGBoost is short for "eXtreme Gradient Boosting." The "eXtreme" refers to speed enhancements such as parallel computing and cache awareness that makes XGBoost approximately 10 times faster than traditional Gradient Boosting. In addition, XGBoost includes a unique split-finding algorithm to optimise trees, along with built-in regularisation that reduces over-fitting. Generally speaking, XGBoost is a faster, more accurate version of Gradient Boosting.



```
#Model no.9
import xgboost as xgb

xgb=xgb.XGBClassifier()

xgb.fit(x_train_ns,y_train_ns)

print_score(xgb,x_train_ns,x_test,y_train_ns,y_test,train=True)
print_score(xgb,x_train_ns,x_test,y_train_ns,y_test,train=False)
model_accuracy(xgb)
```

```
Train Report: 0.9981021666930254
Test Report: 0.844818785094436
Classification Report:              precision    recall  f1-score   support

           0       0.88      0.93      0.90      1522
           1       0.70      0.54      0.61       437

    accuracy                           0.84      1959
   macro avg       0.79      0.74      0.76      1959
weighted avg       0.84      0.84      0.84      1959

Confusion Matrix: [[1418  104]
 [ 200  237]]
Accuracy: 82.90 %
Standard Deviation: 6.91 %
```

5) **ADABoostClassifier:** An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.



```
#Model no.4
from sklearn.ensemble import AdaBoostClassifier

ada=AdaBoostClassifier()

ada.fit(x_train_ns,y_train_ns)

print_score(ada,x_train_ns,x_test,y_train_ns,y_test,train=True)
print_score(ada,x_train_ns,x_test,y_train_ns,y_test,train=False)
model_accuracy(ada)
```

```
Train Report: 0.8266645579629922
Test Report: 0.8172537008677897
Classification Report:              precision    recall  f1-score   suppor

           0       0.89      0.87      0.88      1522
           1       0.58      0.64      0.61       437

    accuracy                           0.82      1959
   macro avg       0.74      0.75      0.74      1959
weighted avg       0.82      0.82      0.82      1959

Confusion Matrix: [[1322  200]
 [ 158  279]]
Accuracy: 79.11 %
Standard Deviation: 2.83 %
```

To select the best model I have used Roc_Auc curve.

Similarly,I have selected 5 models for the second problem statement (for more details refer python code) that is prediction of RainFall and selected best model.
Conclusion from models

1] We got **our first** best model looking at accuracy, ROC AUC Curve & confusion matrix is **XGBClassifier** with Kfold cross validation method with the accuracy score of 82.90 % .Here our model predicts 1418 true positive for rain prediction out of 1522 positive rain prediction and 200 true negative rain prediction out of 437 cases. It predicts 104 false positive rain prediction out of 1522 positive rain prediction and 232 false negative rain prediction out of 437 rain prediction. It gives the f1 score of 90%.

Understand what does precision recall and f1 score and accuracy do

F1 score: this is the harmonic mean of precision and recall and gives a better measure of the incorrectly classified cases than the accuracy matrix.

$$\text{F1-score} = \left(\frac{\text{Recall}^{-1} + \text{Precision}^{-1}}{2}\right)^{-1} = 2 * \frac{(\text{Precision}*\text{Recall})}{(\text{Precision}+\text{Recall})}$$

Precision: It is implied as the measure of the correctly identified positive cases from all the predicted positive cases. Thus, it is useful when the costs of False Positives are high.

$$\text{Precision} = \frac{\text{True Positive}}{(\text{True Positive} + \text{False Positive})}:$$

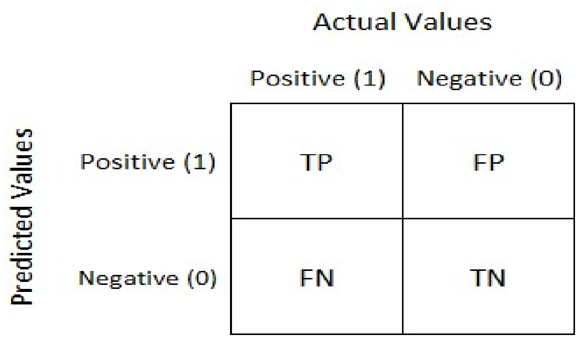
Recall: It is the measure of the correctly identified positive cases from all the actual positive cases. It is important when the cost of False Negatives is high.

$$\text{Recall} = \frac{\text{True Positive}}{(\text{True Positive} + \text{False Negative})}:$$

Accuracy: One of the more obvious metrics, it is the measure of all the correctly identified cases. It is most used when all the classes are equally important.

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{(\text{True Positive} + \text{False Positive} + \text{True Negative} + \text{False Negative})}$$

Confusion matrix
A table that is often used to describe the performance of a classification model (or 'classifier') on a set of test data for which the true values are known.



Predicted Values		Actual Values	
		Positive (1)	Negative (0)
	Positive (1)	TP	FP
	Negative (0)	FN	TN

NOTE:

TN/True Negative: the cases were negative and predicted negative.
TP/True Positive: the cases were positive and predicted positive.
FN/False Negative: the cases were positive but predicted negative.
TN/True Negative: the cases were negative but predicted positive.

Hyper parameter tuning:

Hyper parameter optimisation in machine learning intends to find the hyper parameters of a given machine learning algorithm that deliver the best performance as measured on a validation set. Hyper parameters, in contrast to model parameters, are set by the machine learning engineer before training.

The number of trees in a random forest is a hyper parameter while the weights in a neural network are model parameters learned during training. I like to think of hyper parameters as the model settings to be tuned so that the model can optimally solve the machine learning problem.

We will use RandomizedSearchCV for the hyper parameter tuning.

RandomizedSearchCV :

Randomized search on hyper parameters. RandomizedSearchCV implements a “fit” and a “score” method. It also implements “score_samples”, “predict”, “predict_proba”, “decision_function”, “transform” and “inverse_transform” if they are implemented in the estimator used.

```
]: # Hyper tuning by using RandomizedSearchCV With XGB
   from sklearn.model_selection import RandomizedSearchCV

   para={'n_estimators':[7,10,8],'gamma':[0.25,0.1,0.3,0.4],'max_depth':[2,4,6,7,8,9],'random_state':[20,10,30]}
   rand=RandomizedSearchCV(estimator=xgb, cv=5,param_distributions=para)
   rand.fit(x_train,y_train)

   rand.best_params_

]: {'random_state': 30, 'n_estimators': 10, 'max_depth': 6, 'gamma': 0.25}
```

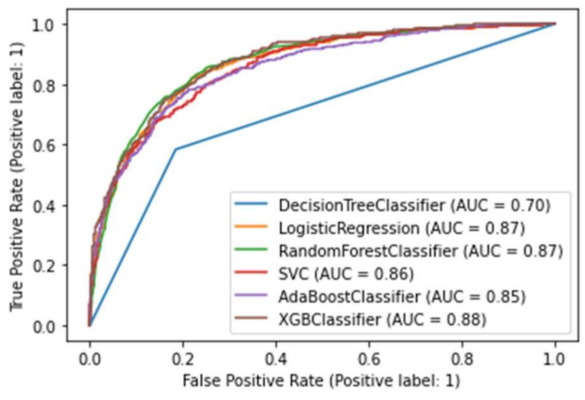
```
]: #Model no.6
   import xgboost as xgb

   xgb=xgb.XGBClassifier(random_state= 30, n_estimators= 10, max_depth= 6, gamma= 0.09)

   xgb.fit(x_train_ns,y_train_ns)

   print_score(xgb,x_train_ns,x_test,y_train_ns,y_test,train=True)
   print_score(xgb,x_train_ns,x_test,y_train_ns,y_test,train=False)
   model_accuracy(xgb)

   Train Report: 0.8956191681164004
   Test Report: 0.8453292496171516
   Classification Report:                precision    recall  f1-score   support

              0       0.89      0.91      0.90      1522
```

ROC curve:
It 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.

Higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s. By analogy, the Higher the AUC, the better the model is at distinguishing between rain fall tomorrow or not. The ROC curve is plotted with TPR against the FPR where TPR is on the y-axis and FPR is on the x-axis.

Lets check how our model works on test data

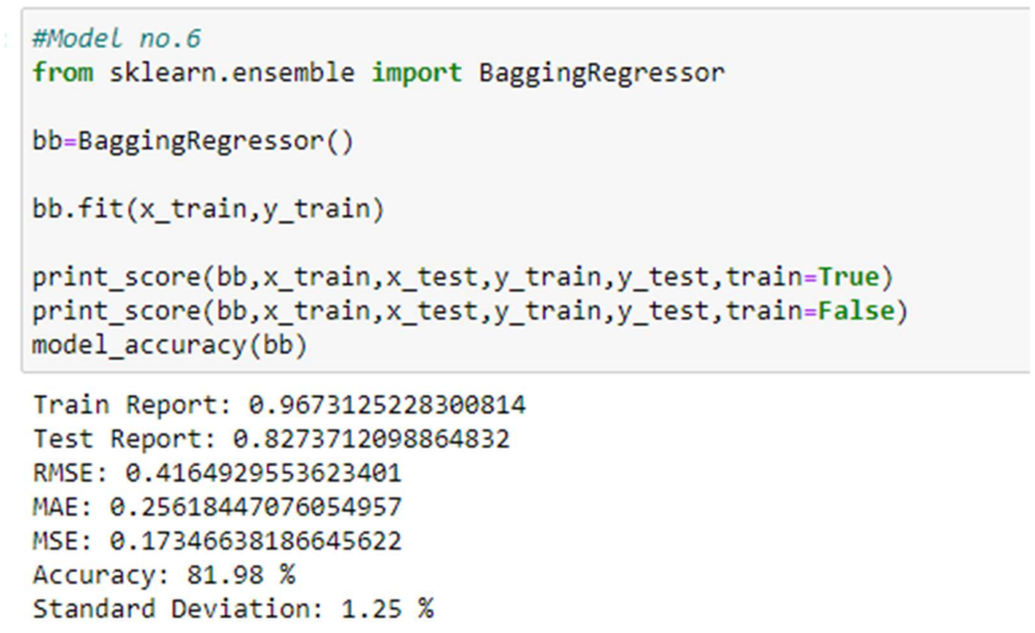
```
]: disp=plot_roc_curve(dt,x_test,y_test)
   plot_roc_curve(LR,x_test,y_test,ax=disp.ax_)
   plot_roc_curve(rand_clf,x_test,y_test,ax=disp.ax_)
   plot_roc_curve(svc,x_test,y_test,ax=disp.ax_)
   plot_roc_curve(ada,x_test,y_test,ax=disp.ax_)
   plot_roc_curve(xgb,x_test,y_test,ax=disp.ax_)

]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x24d113f1760>
```



Comparing ROC curves for all the models

2] Lets select the model for Rainfall prediction

We got our best model to predict **Rainfall** looking at train and test report & cross validation score i.e **BaggingRegressor** with accuracy score 81.98 %



```
#Model no.6
from sklearn.ensemble import BaggingRegressor

bb=BaggingRegressor()

bb.fit(x_train,y_train)

print_score(bb,x_train,x_test,y_train,y_test,train=True)
print_score(bb,x_train,x_test,y_train,y_test,train=False)
model_accuracy(bb)
```

```
Train Report: 0.9673125228300814
Test Report: 0.8273712098864832
RMSE: 0.4164929553623401
MAE: 0.25618447076054957
MSE: 0.17346638186645622
Accuracy: 81.98 %
Standard Deviation: 1.25 %
```

Remarks:

This project has built a both type of models that can detect RainTomarrow & RainFall. The challenge behind predicting both models is climate change is always a major issue for whole world and making any prediction on that is now days pretty difficult and unpredictable.

Climate change is due to the current global warming trend is human expansion. Due to this air and oceans are warming, sea level is rising and flooding and drought etc. One of the serious consequences due to this climate change is on Rainfall. Rainfall prediction now days is an arduous task which is taking into the consideration of most of the major world-wide authorities.

This Paper has presented a supervised rainfall learning model which used machine learning algorithms to classify rainfall data. We used different machine learning algorithm to check the accuracy of rainfall prediction.

References:

- XGBClassifier
- Hyper-parameter Tuning
- SMOTE
- Getting started with XGBoost
- scikit

In a nutshell….

Exploring and knowing your datasets is a very essential step. It not only helps in finding anomalies, uniqueness and pattern in the dataset but also helps us in building better hypothesis functions. If you wish to see the entire code, here Link is the to my Jupiter notebook