Weather Forecasting & Rainfall Prediction

# Introduction:

Weather Forecasting is an application of science and technology, which predicts the condition of the atmosphere for a given location at a given time.

Forecasting means to estimate or to predict and prediction of the weather conditions for a given location in simple words can be termed as Weather Forecasting.

Rainfall Prediction is a subset of weather forecasting which deals with a specific topic of predicting the amount of rainfall over an area or a region.

In the pointers below we deep dive in to the core question why and in what domains does the weather prediction play an important role.

* Agricultural Domain
  + Farmers depend on the weather conditions to plant the crops.
  + Forecasting weather conditions helps the farmers to know when they should fertilize the crop and in what quantity.
  + Weather can influence the kind of pests and the quantity of it, weather prediction can help the farmers a great deal to be prepared to control the pest and prevent the crop from being destroyed.
  + Soil moisture is the most important factor in agriculture and an accurate regional weather forecast can help the farmers in deciding their strategy to work in the field.
  + With accurate weather forecasts a farmer can plan the right irrigational strategies for the field and crop.
* Water resource management
  + It plays a very important role for water storage bodies like dams, where in if a heavy rainfall is predicted they could release water stored in the dam before it leading to over flooding of dams or water storage bodies causing property damages.
* Weather conditions also play an important role in analyzing if the driving conditions are safe on the road for the drivers.
  + According to https://nap.nationalacademies.org/ , The annual impacts of adverse weather on the national highway system and roads are staggering: 1.5 million weather-related crashes with 7,400 deaths, more than 700,000 injuries, and $42 billion in economic losses (BTS, 2007).
* Aviation
  + $4.2 billion is lost each year as a result of weather-related air traffic delays (NOAA, 2010).

The main goal of weather prediction is to provide appropriate information to people and organizations that can be used to reduce the weather-related losses and enhance societal benefits, including protection of life and property, public health and safety and support of economic prosperity and quality of life.

Australia is among the world's largest agricultural exporters (Gunasekera *et al* [2007](https://iopscience.iop.org/article/10.1088/1748-9326/ab9e98#erlab9e98bib18)). Australian wheat commodity contributes roughly 15% of global annual wheat trade ([www.aegic.org.au](https://iopscience.iop.org/article/10.1088/1748-9326/www.aegic.org.au)). Thus, Australian agricultural sector is very important to ensure global food supply and security (Qureshi *et al* [2013](https://iopscience.iop.org/article/10.1088/1748-9326/ab9e98#erlab9e98bib37)). However, highly variable inter-annual seasonal rainfall exerts serious adverse impacts on Australian agricultural productivity (Cobon and Toombs [2013](https://iopscience.iop.org/article/10.1088/1748-9326/ab9e98#erlab9e98bib11)). The drought in 2018 has resulted in yield loss of 53% in eastern Australia compared to the average of past two decades (<https://www.agriculture.gov.au/abares>).

Hence weather forecast plays a very vital role in any country's economy.

# Problem Definition:

In this project, we would:

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**.

At present there are 2 ways forecasting weather in Australia:

1. Dynamical Models
2. Statistical Models

As a Data Scientist we would use statistical methodology using machine learning to make predictions.

The problem in itself is divided into 2 parts where in Part A (forecast **whether or not it will rain tomorrow)** is a binary classification problem and Part B (**predict how much rainfall could be there)** is a regression problem. We would first work on classification problem and predict if it would rain tomorrow and then work on the regression part to determine the amount of rainfall.

# Pre-Processing and Data Analysis:

Let’s start by importing the necessary libraries to read and visualize the data.

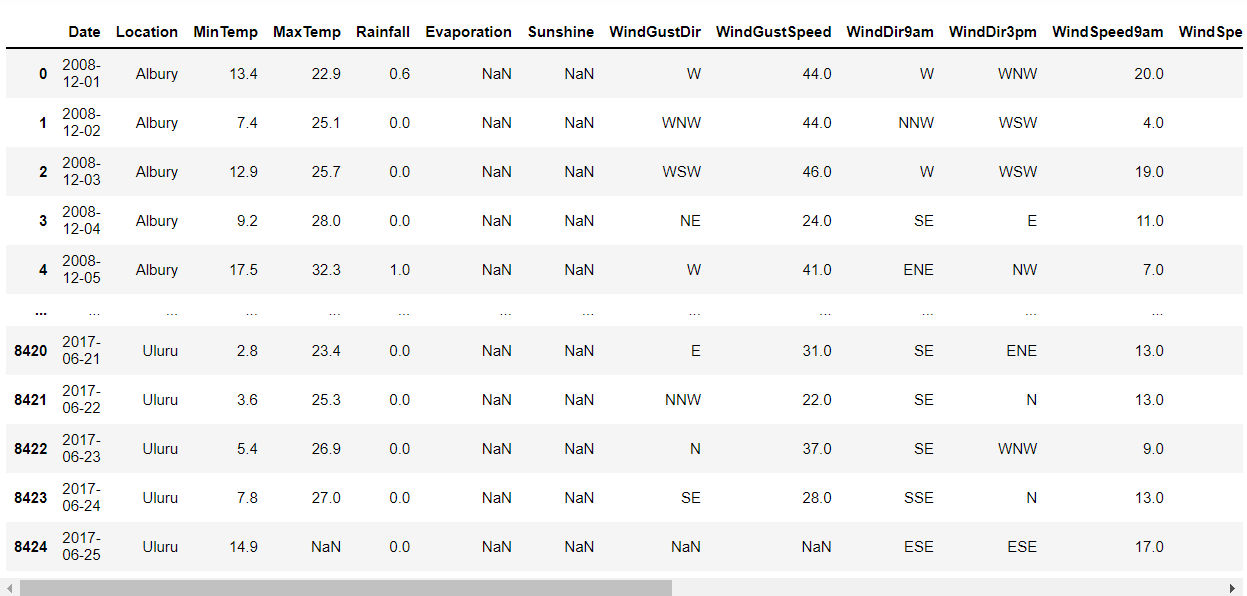
|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns  pd.options.display.max\_columns = None  import warnings  warnings.filterwarnings('ignore') |

In the code block above, we have 4 main libraries.

* NumPy - It’s a library that offers comprehensive mathematical functions.
* Pandas – It is an open-source library used for data analysis and manipulation.
* Matplotlib – Data visualization library.
* Seaborn – Another Hi-End visualization library based on Matplotlib.

|  |
| --- |
| df = pd.read\_csv('weatherAUS.csv')  df |

The dataset is in a csv format, we use read\_csv method from pandas to read the file and store it in a dataframe df, please refer below for the dataset representation.

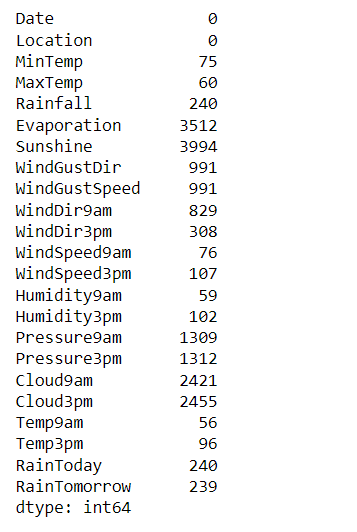


In the above, we see presence of NaN values which are empty values and would have to be delt with.

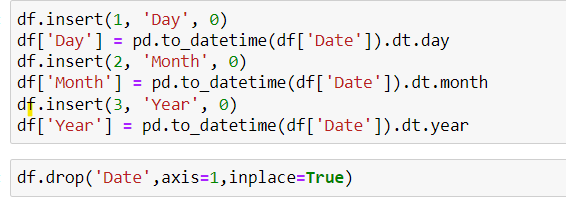
We use the command below to check all the Nan//Null values in the dataset.

|  |
| --- |
| df.isnull().sum() |

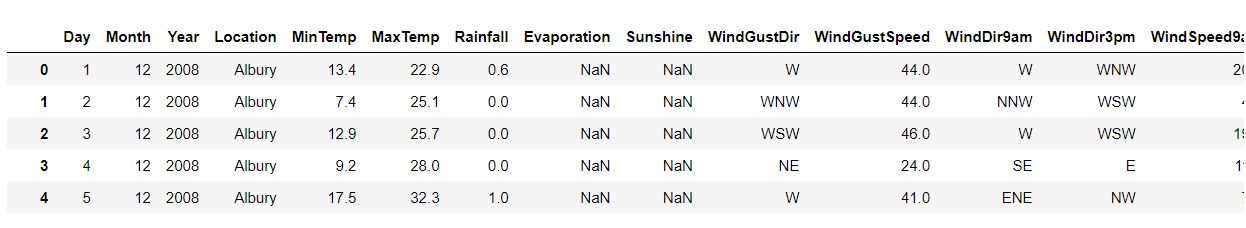
From the above code, we see we have Null//NaN values in all the columns other than Date and Location.



Let's work on Date column and convert it into a format that would be acceptable for our analysis.



Here we separate Day, Month and Year from the Date column into 3 separate columns as seen below and drop the Date column as we no longer need it.



Let's create a list of object type and non-object type features by using the command below.

|  |
| --- |
| cat\_columns = []  numeric\_columns = []  for i in df.dtypes.index:  if df[i].dtypes == 'object':  cat\_columns.append(i)  else:  numeric\_columns.append(i)  cat\_columns,numeric\_columns |
|  |

The output is as below:

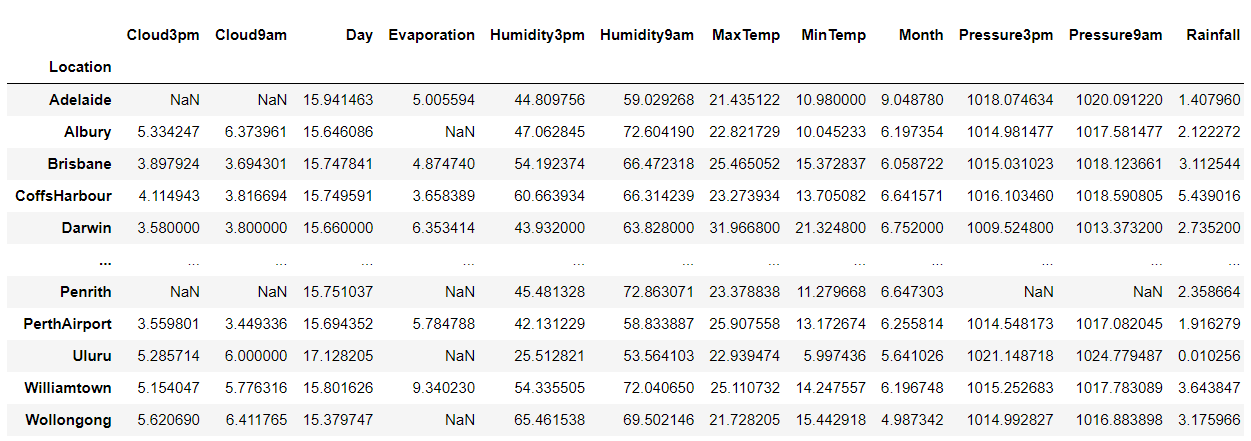


We have 2 lists above one with the object type variables and the other without.

Let's work on Null values for columns that are not object type, we could fill the null values with an overall Mean for that column but that would not be a right approach as the climatic conditions vary based on Location and Month. Filling null values with column’s Mean would not consider this hence we use pivot table to see the mean based on location, try and fill as many possible values based on location and the rest on the Month.

We plot a pivot table for the same using the code below

|  |
| --- |
| pivot\_table1 = df.pivot\_table(values=numeric\_columns,index=['Location'],aggfunc=(lambda x:x.mean()))  pivot\_table1 |



From the Pivot table above, we try to fill as many null values as possible using the code below:

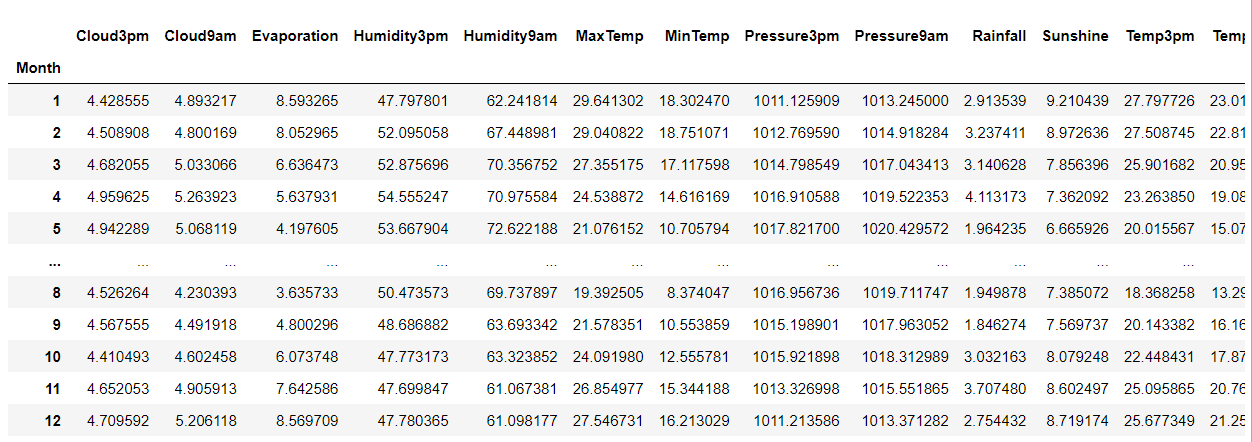
|  |
| --- |
| for l in numeric\_columns:  missing = df[l].isnull()  for i,item in enumerate(df['Location']):  if missing[i]:  df[l][i] = pivot\_table1.loc[item][l] |

We see the null values have reduced from

|  |  |
| --- | --- |
| **Initial Null Values** | **Null Count after filling Based on Location** |
|  |  |

Let’s try to replace the remaining Null values for the numerical columns based on Month of the year.

|  |
| --- |
| pivot\_table2 = df.pivot\_table(values=['MinTemp','MaxTemp','Rainfall','Evaporation','Sunshine','WindGustSpeed','WindSpeed9am','WindSpeed3pm','Humidity9am','Humidity3pm','Pressure9am','Pressure3pm','Cloud9am','Cloud3pm','Temp9am','Temp3pm'],index=['Month'],aggfunc=(lambda x:x.mean()))  pivot\_table2 |



Filling the Null values based on the values from the pivot\_table2.

|  |
| --- |
| for l in numeric\_columns:  missing = df[l].isnull()  for i,item in enumerate(df['Month']):  if missing[i]:  df[l][i] = pivot\_table2.loc[item][l] |

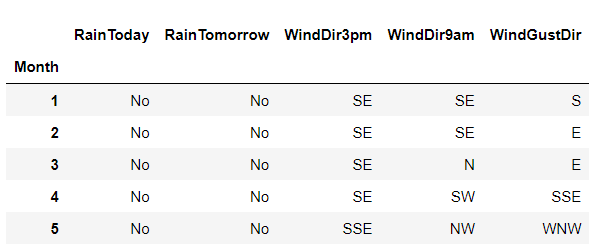
Checking Null Values:

|  |  |
| --- | --- |
| Previous Null Count | Null Values Now |
|  |  |

We see all the values in non-Object type columns are filled.

Let’s work on Categorical Columns and fill the missing values based on Month with Mode:

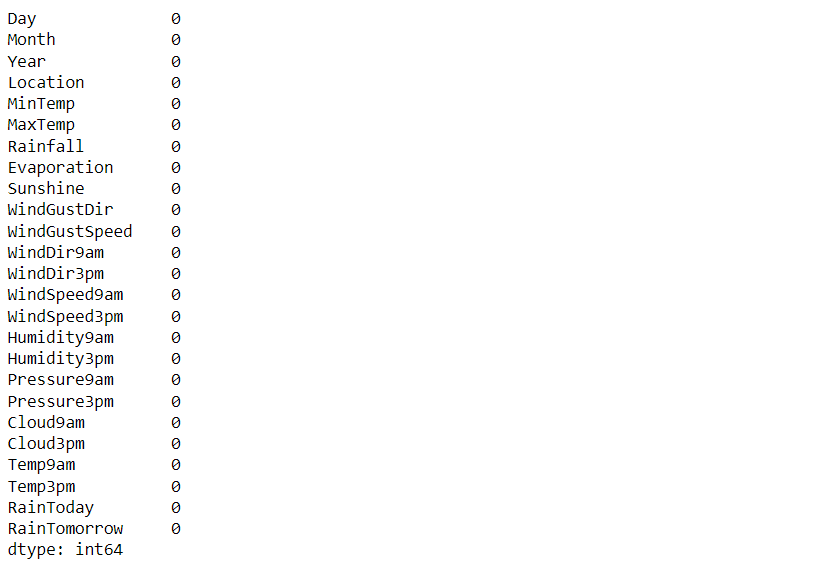
|  |
| --- |
| pivot\_table3 = df.pivot\_table(values=['WindGustDir','WindDir9am','WindDir3pm','RainToday','RainTomorrow'], index=['Month'],aggfunc=(lambda x:x.mode()[0]))  pivot\_table3 |



|  |
| --- |
| for l in cat\_columns:  missing = df[l].isnull()  for i,item in enumerate(df['Month']):  if missing[i]:  df[l][i] = pivot\_table3.loc[item][l] |

Checking Null values.

|  |
| --- |
| df.isnull().sum() |

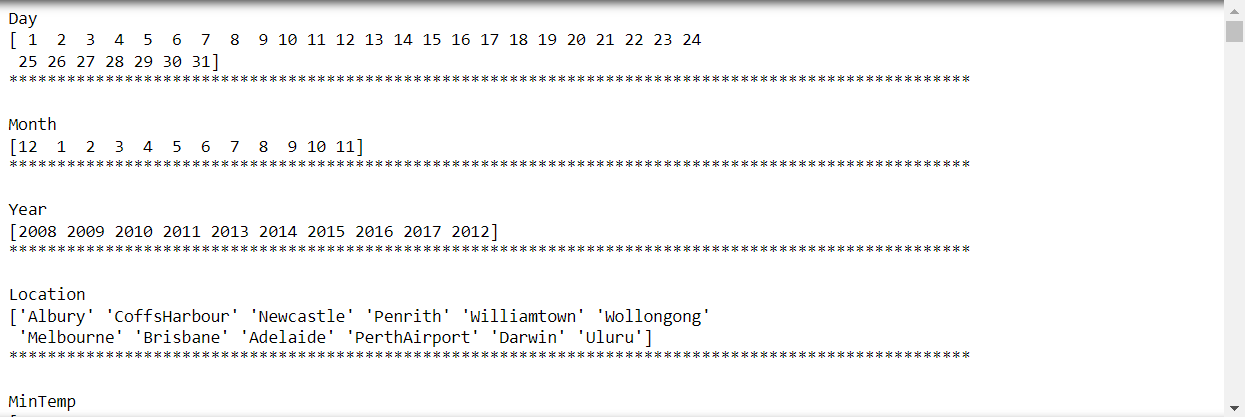


From above, we see that we do not have any null values in the dataset anymore.

Checking for any unwanted or abnormal data patterns in the dataset such as spaces, \*,?...etc. in the dataset.

|  |
| --- |
| for i in df.columns:  print(i)  print(df[i].unique())  print('\*'\*100)  print() |

Output of the code above will display all the unique values in a particular column, as shown below.



We see no unusual data patterns in the dataset and the data is good for processing.

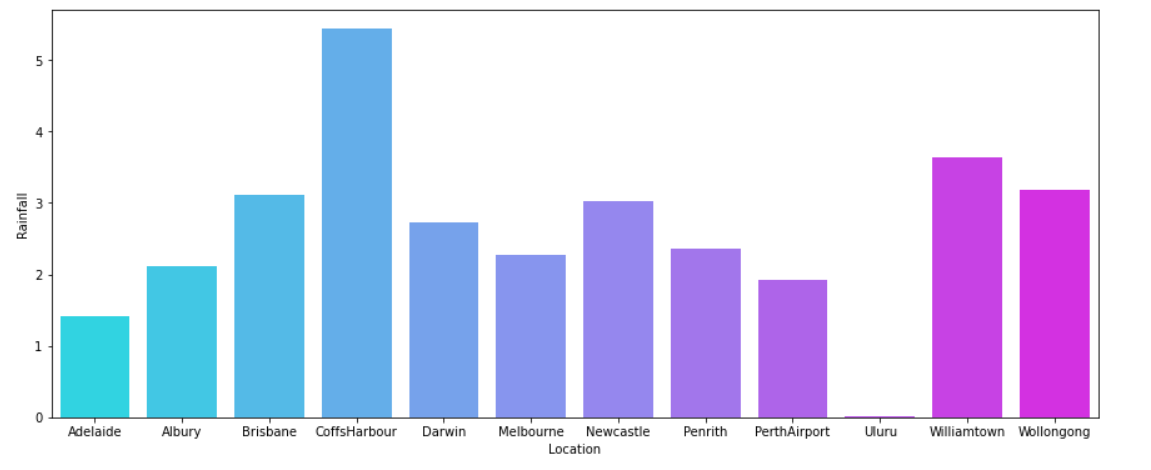
### Data Visualization:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Observations:

* June to August seems to be the coldest time in Australia, where the temperature is below 10°C.
* Jan and Feb seem to be the hottest months where the temperature reaches almost 30°C.
* March seems to be the month that receives the highest rainfall and July to Sept. the least.
* Evaporation seems to be the high from November to February, peaking out in Dec and Jan and lowest in July.
* We see the longest hrs. of sunshine in Jan and feb and lowest between May to July.
* We see Wind Gust Speed to be Highest in the Months of Nov to Feb and lowest in May, June.
* Wind Speeds are High between the Months Sept to January and lowest in June.
* Humidity seems to be highest in the month of June and lowest in Nov, Dec.
* Pressure looks to be the highest between May to July and lowest in Dec, Jan.
* April,May,June months are cloudy.

|  |
| --- |
| pt4 = df.pivot\_table(values = 'Rainfall',index = ['Location'])  plt.figure(figsize=(15,6))  fig = sns.barplot(data = pt4, x=pt4.index, y='Rainfall', palette='cool')  plt.show() |



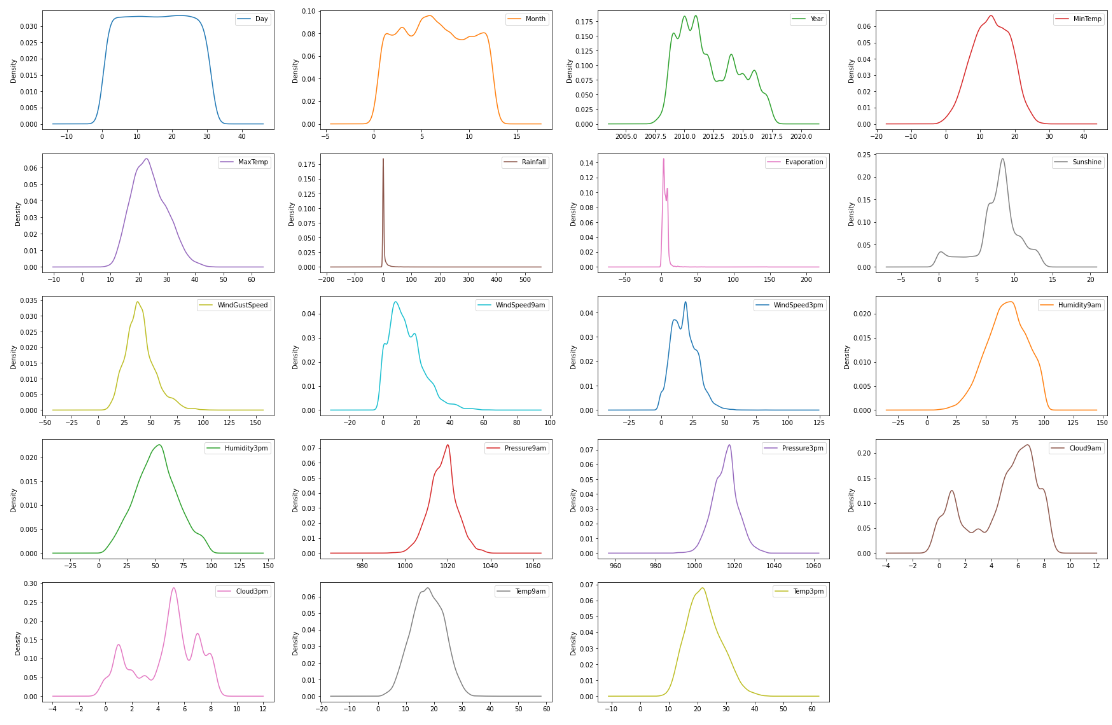
* From Above, we see that Coffs Harbour receives the highest rainfall and Uluru the lowest.

## Check the skewness of the Data:

**Skewness is** the **measure** of the asymmetry of an ideally symmetric probability distribution.

The code below helps us to visualize the skewness.

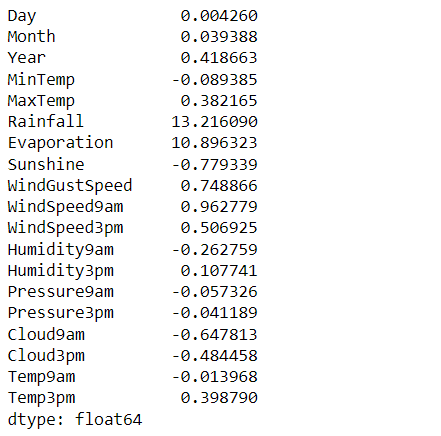
|  |
| --- |
| df.plot(kind = 'density',subplots=True,layout=(5,4),figsize=(30,20),sharex=False)  plt.show() |



Checking the skewness

|  |
| --- |
| df.skew() |

The output is displayed as below:



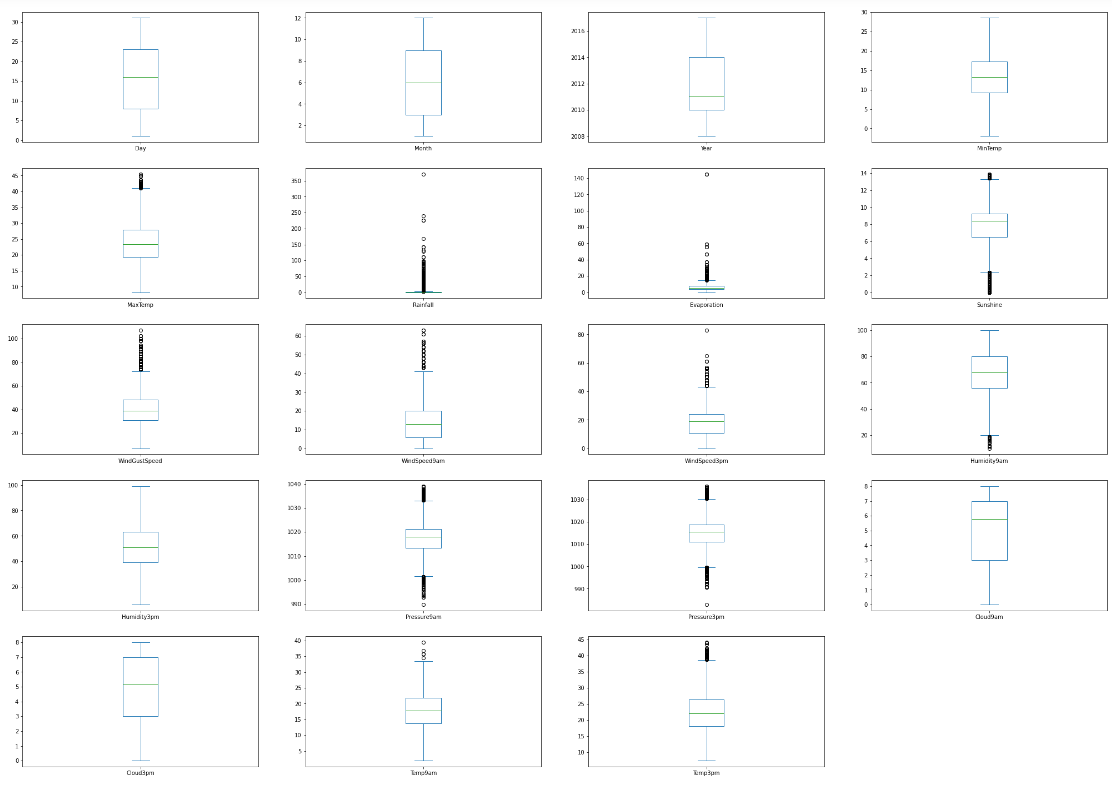
Setting Skewness Threshold as +/- 1, we see high skewness in Evaporation and Rainfall.

### Checking Outliers:

Outlier is **an observation that lies an abnormal distance from other values in a random sample from a population**.

|  |
| --- |
| df.plot(kind='box',subplots=True,layout=(5,4),figsize=(35,25),sharex=True)  plt.show() |

Output:

We see presence of outliers in Rainfall, Evaporation, Wind Gustspeed, wind speed at 9am, pressure at 9am, pressure at 3pm and Temp 9am

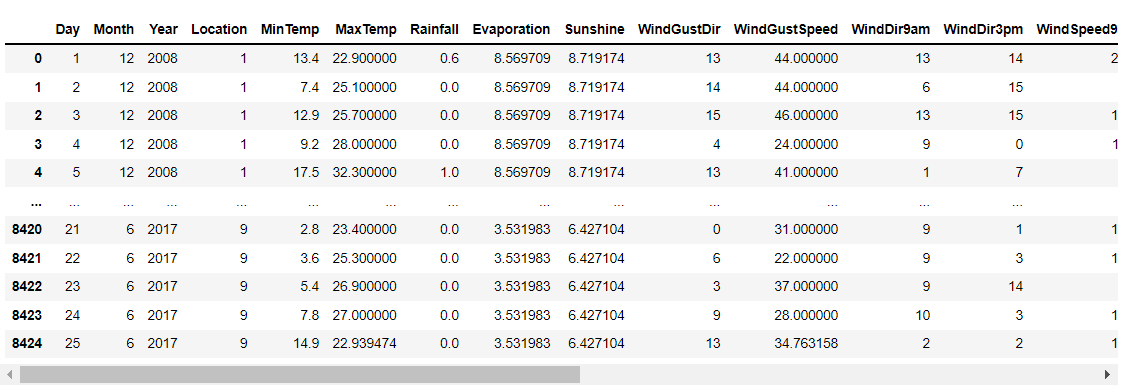
### Encoding:

It is a process of converting object type data which a machine cannot understand to a numeric format that can be understood by the machine.

|  |
| --- |
| from sklearn.preprocessing import LabelEncoder  le = LabelEncoder()  for i in cat\_columns:  df[i] = le.fit\_transform(df[i])  df |

Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

Lable encoder is a very effective tool provided by sklearn that converts categorical features in to numeric values.



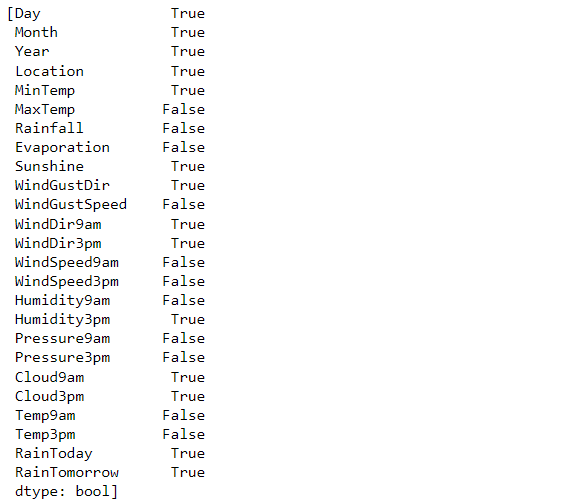
From above we could see the Categorical Columns such as Location, WindGustSpeed have been converted to Numerical values.

## Removing Outliers:

Most of the Machine Learning algorithms are highly sensitive to outliers hence it is important to remove them.

|  |
| --- |
| from scipy.stats import zscore  [(np.abs(zscore(df))<3).all()] |

We are using zscore method to remove the outliers, the above code shows the presence of outlier in the features as below.



According to zscore we see presence of outliers in 'MaxTemp','Rainfall','Evaporation','WindGustSpeed','WindSpeed9am','WindSpeed3pm','Humidity9am','Pressure9am','Pressure3pm','Temp9am','Temp3pm'.

|  |
| --- |
| z = np.abs(zscore(df))  z.shape |



|  |
| --- |
| df = df[(z<3).all(axis=1)]  df.shape |



We see that we have 8005 rows left after removing the outliers. Let’s now calculate the percentage data loss.

### Percentage Data Loss:

|  |
| --- |
| percentage\_loss = (8425-8005)/8425\*100  percentage\_loss |

4.985163204747774

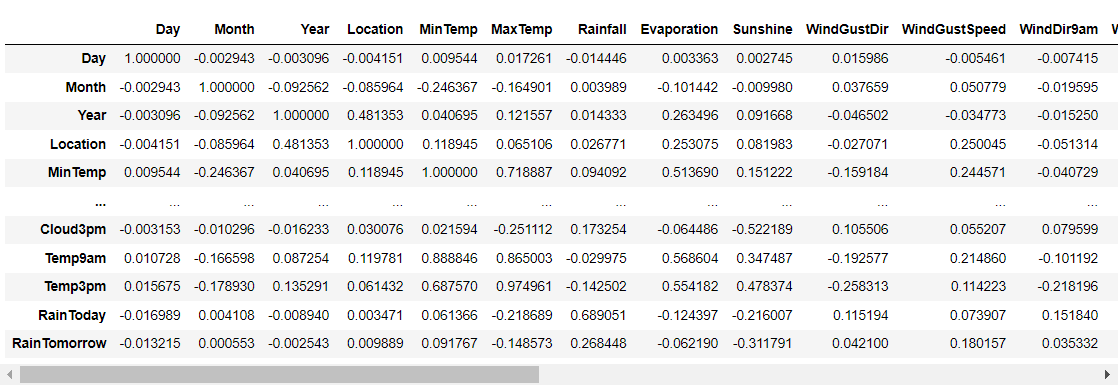
We see we have about 5% data loss which is ok.

## Checking Correlation:

We check correlations to determine the strength of relationship between two variables. We could have the correlation values between+1 to –1, wherein +ve values represent positive relationship e.g. If the height increases, so does the weight for a person it is termed as a positive correlation and if the weight of the person decreases with the increase in height it is called a negative correlation. The values closer to 0 represent a weak correlation and the values close to 1 a strong correlation.

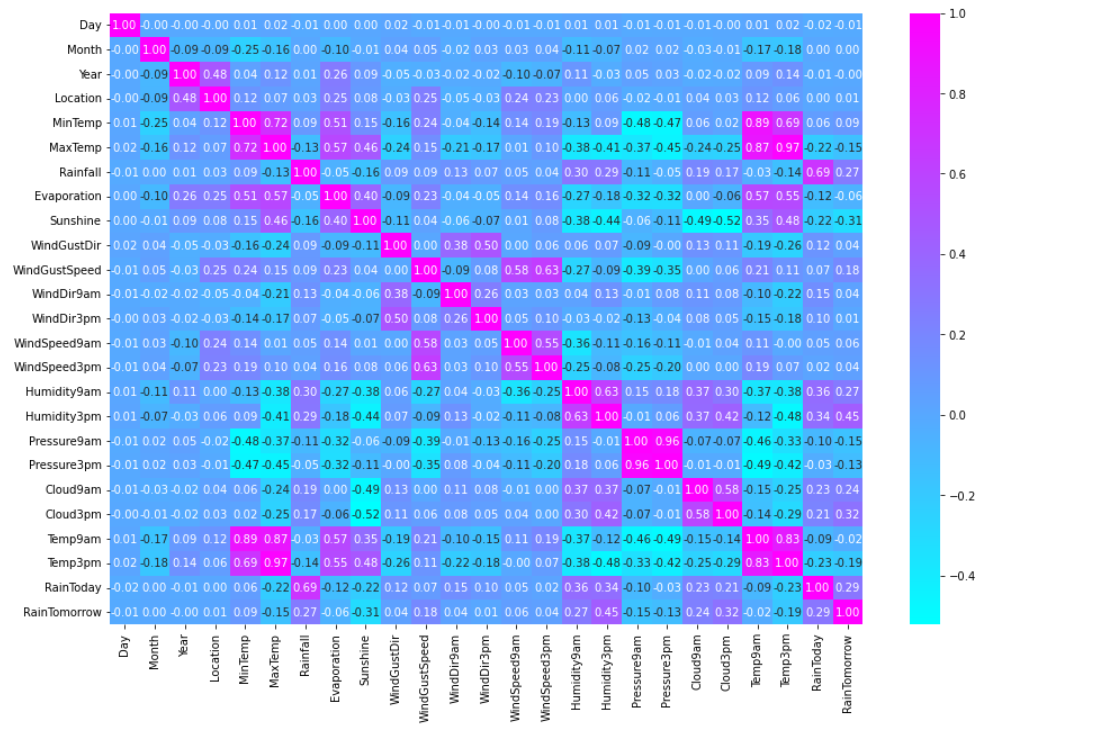
|  |
| --- |
| df.corr() |

Output:



|  |
| --- |
| plt.figure(figsize=(15,10))  sns.heatmap(df.corr(),annot=True,fmt='.2f',cmap='cool')  plt.show() |

With the above line of code, we can have a pictorial representation of the correlations.

We see that the features have both + and - correlations with the Target Variable(RainTomorrow).

Let’s split the dataframe into target and features for further analysis.

Features are represented by x.

|  |
| --- |
| x = df.drop('RainTomorrow',axis=1)  x.shape |

(8005,24)

Target is represented by y.

|  |
| --- |
| y = df['RainTomorrow']  y.shape |

(8005,)

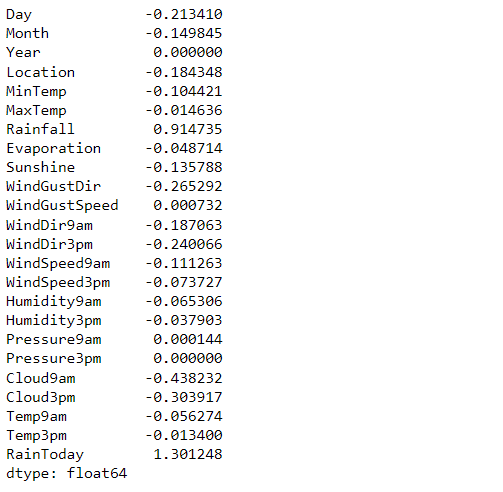
## Removing skewness using Power\_transform:

We use power\_transform which is present in sklearn to remove the skewness.

|  |
| --- |
| from sklearn.preprocessing import power\_transform  X = power\_transform(x)  X = pd.DataFrame(X,columns=x.columns)  X |

Let’s check if the skewness is within the threshold.

|  |
| --- |
| X.skew() |



We see that the skewness for all the features is within the threshold. The skewness for RainToday shows to be higher but it can be ignored as it is a categorical variable.

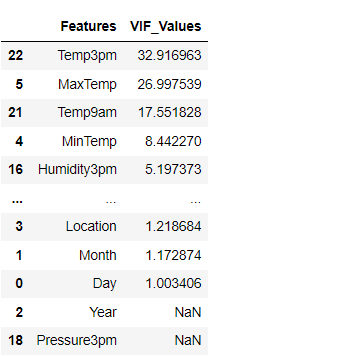
## Checking VIF:

## Variance inflation factor (VIF) is a measure of the amount of multicollinearity in a set of multiple regression variables.

The higher the correlation, higher would be the VIF value, generally the VIF value above 5 is considered to moderate to high and anything above 10 to be very high hence we set the VIF threshold to be 5.

|  |
| --- |
| from statsmodels.stats.outliers\_influence import variance\_inflation\_factor  def calculate\_vif(dataset):  vif = pd.DataFrame()  vif['Features'] = dataset.columns  vif['VIF\_Values'] = [variance\_inflation\_factor(dataset.values,i)for i in range(dataset.shape[1])]  return vif.sort\_values(by='VIF\_Values',ascending=False)  calculate\_vif(X) |

VIF module is imported from the statsmodels library as shown in the code above.

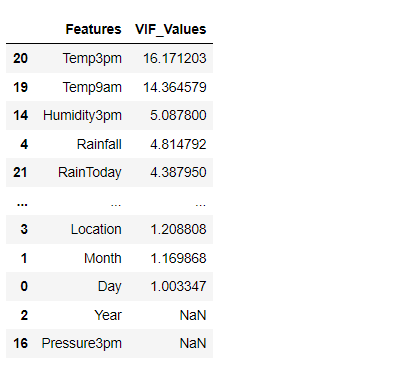


We see the VIF values for Temp3pm,MaxTemp, Temp9am are higher than the threshold, which means the information contained in these columns is already conveyed by other variables. Let's work on MaxTemp first.

Rather than dropping the columns, which would lead to information loss, let’s use a feature engineering technique wherein rather than having 2 columns MaxTemp and MinTemp, let's have a column which signifies the difference in the temperature and check the VIF.

|  |
| --- |
| X['Temp\_Difference'] = X['MaxTemp']-X['MinTemp']  X.drop(['MaxTemp','MinTemp'],axis=1,inplace=True)  calculate\_vif(X) |

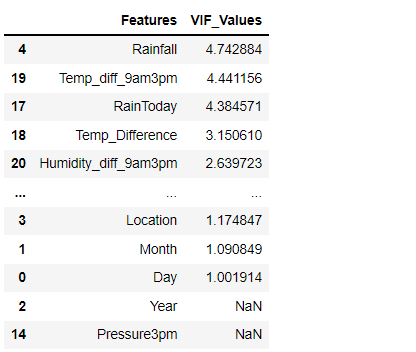
Output:



We see that the VIF has reduced, let us repeat the same for Temp3pm and Temp9am and Humidity3pm and Humidity9am.

Let’s find the difference and have a new column that signifies the difference and drop the original 2 columns and re check the VIF.

|  |
| --- |
| X['Temp\_diff\_9am3pm'] = X['Temp3pm']-X['Temp9am']  X['Humidity\_diff\_9am3pm'] = X['Humidity3pm']-X['Humidity9am']  X.drop(['Temp3pm','Temp9am','Humidity3pm','Humidity9am'],axis=1,inplace=True)  calculate\_vif(X) |



We see that all the values are within the set threshold but **we could further reduce the dimensionality by finding the difference for Pressure, Windspeed and Clouds as well**.

|  |
| --- |
| X['Pressure\_diff\_9am3pm'] = X['Pressure3pm']-X['Pressure9am']  X['WindSpeed\_diff\_9am3pm'] = X['WindSpeed3pm']-X['WindSpeed9am']  X['Clouds\_diff\_9am3pm'] = X['Cloud3pm']-X['Cloud9am']  X.drop(['Pressure3pm','Pressure9am','Cloud3pm','Cloud9am','WindSpeed3pm','WindSpeed9am'],axis=1,inplace=True)  X.shape |



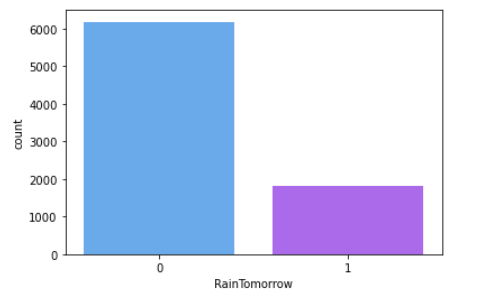
We see that using the above technique, we have reduced the number of features from 24 to 18, without having any information loss.

## Checking if the Dataset is Balanced:

It is very important for us to have a dataset that is balanced, else it would lead to a model that is biased.

|  |
| --- |
| sns.countplot(y,palette='cool')  plt.show() |

Output:



We see that the dataset is imbalanced and we’d have to balance it.

## Balancing Dataset:

For balancing the dataset we’d be using oversampling technique rather than undersampling, as undersampling would lead to data loss.

Importing the Library:

|  |
| --- |
| from imblearn.over\_sampling import SMOTE  sm = SMOTE() |

Fitting the data to balance:

|  |
| --- |
| X,y = sm.fit\_resample(X,y)  X.shape,y.shape |

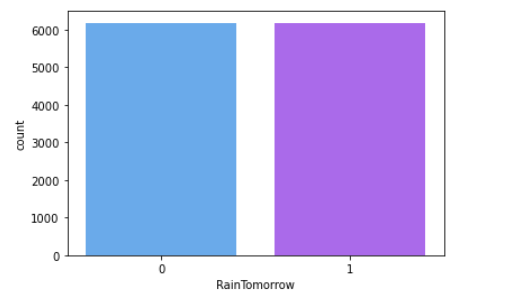
Output:



We see from 8005 rows we previously had; we now have 12362 rows.

Let's check if the dataset is now balanced.

|  |
| --- |
| sns.countplot(y,palette='cool')  plt.show() |



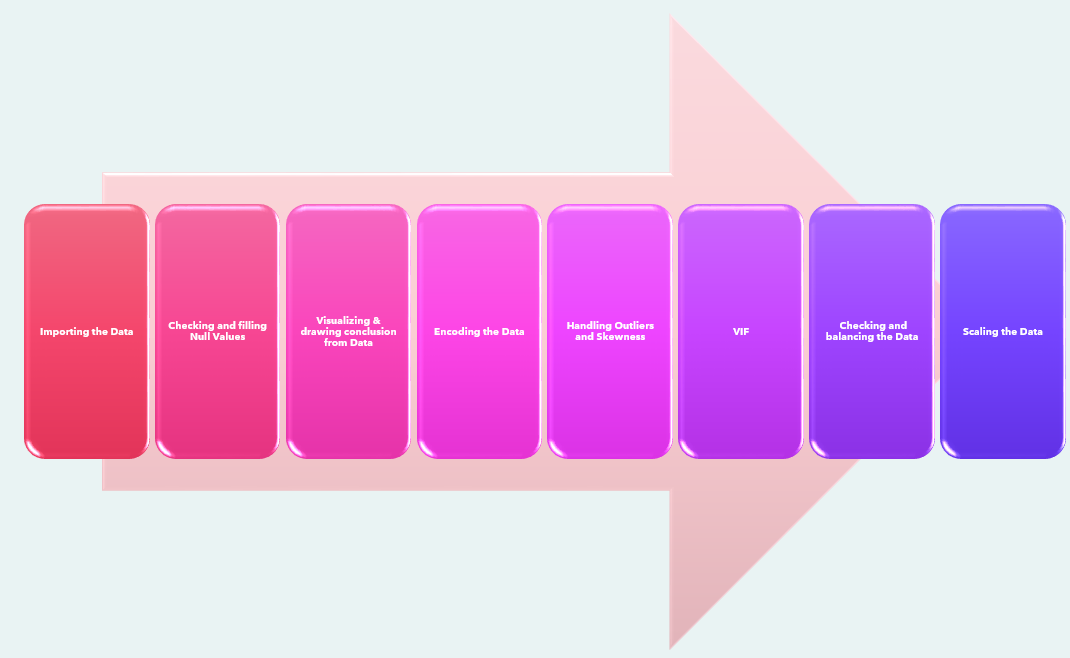
We see that the dataset is now balanced.

## Scaling the Data using StandardScaler:

We use a built-in function in sklearn to scale the data.

|  |
| --- |
| from sklearn.preprocessing import StandardScaler  sc = StandardScaler()  dfx = sc.fit\_transform(X)  X = pd.DataFrame(dfx,columns=X.columns) |

# Summary of Preprocessing:



## Model Building:

Importing the necessary libraries.

|  |
| --- |
| from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import accuracy\_score,confusion\_matrix,classification\_report  from sklearn.linear\_model import LogisticRegression  lr = LogisticRegression() |

In the first line of the code, we would import train\_test\_split which is used to split the data into 2 parts - Train and Test

In the second line of code, we import important metrices for evaluating the model.

In the 3rd line of the code, we import Logistic regression to set our base model.

|  |
| --- |
| max\_accuracy = 0  max\_random\_state = 0  for i in range(0,100):  x\_train,x\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.3,random\_state=i)  lr.fit(x\_train,y\_train)  pred = lr.predict(x\_test)  accuracy=accuracy\_score(y\_test,pred)  if accuracy > max\_accuracy:  max\_accuracy = accuracy  max\_random\_state = i  print('The best accuracy is ', max\_accuracy ,'for random\_state', max\_random\_state) |

In the above line of code, we try to find the best random\_state suited for our Base Model.

We initially split the data into training and testing then we train the model on training data and with the learnings from training data we try to predict the test data. We then try to find the best accuracy with the random state in range 0-100 and then print the best random\_state and accuracy for it.

Output:

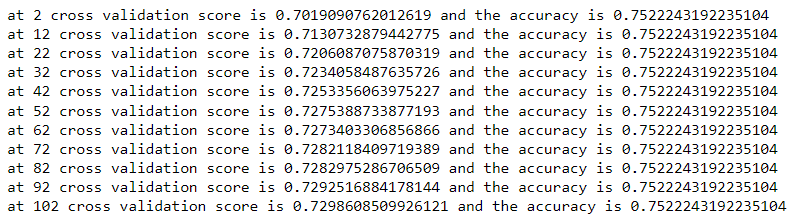


We see that the best random state is 8.

We then check the best cross validation score and at how many folds we achieve it.

|  |
| --- |
| from sklearn.model\_selection import cross\_val\_score  for j in range(2,103,10):  cv = cross\_val\_score(lr,X,y,cv=j)  cv\_score = cv.mean()  print('at',j,'cross validation score is',cv\_score,'and the accuracy is',accuracy) |

Output:



We have checked for the cv score for cross folds in range 2-103 with an interval of 10 and achieved the highest accuracy at 102 cross-folds.

Rather than repeating the code to check the Models performance we define a Model using the code below:

|  |
| --- |
| def Model(model):  model.fit(x\_test,y\_test)  pred = model.predict(x\_test)  accuracy = accuracy\_score(y\_test,pred)\*100  cv = cross\_val\_score(model,X,y,cv=102)  cv\_score = cv.mean()\*100  print('Report for model', model)  print('The Accuracy Score is', accuracy)  print('Confussion Matrix :','\n',confusion\_matrix(y\_test,pred))  print(classification\_report(y\_test,pred))  print('Cross Validation Score is ', cv\_score)  print()  print('Difference between accuracy score and cv is',accuracy-cv\_score) |

Now let's check the Model performances for various algorithms.

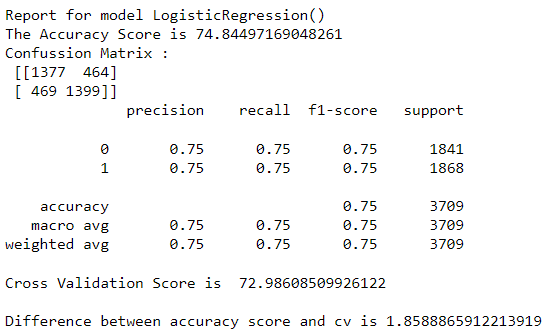
### Logistic Regression:

Logistic regression is a supervised learning algorithm used to predict a dependent categorical target variable.

Code:

|  |
| --- |
| Model(lr) |

Output:



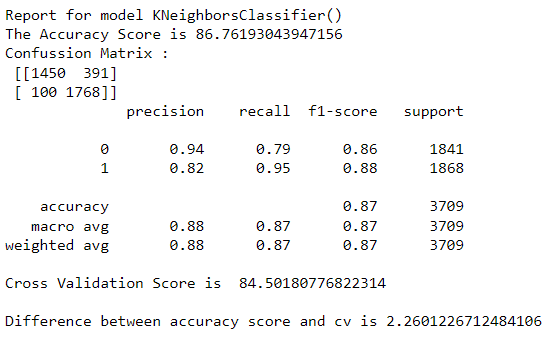
### KNeighborsClassifier:

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.

Code:

|  |
| --- |
| from sklearn.neighbors import KNeighborsClassifier  knn = KNeighborsClassifier()  Model(knn) |

Output:



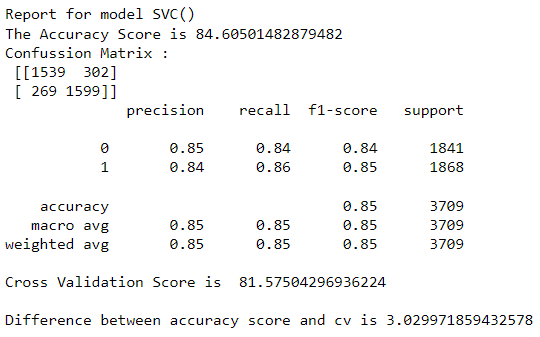
### Support Vector Classifier:

A support vector machine (SVM) is **a supervised machine learning model that uses classification algorithms for two-group classification problems**.

Code:

|  |
| --- |
| from sklearn.svm import SVC  svc = SVC()  Model(svc) |

Output:



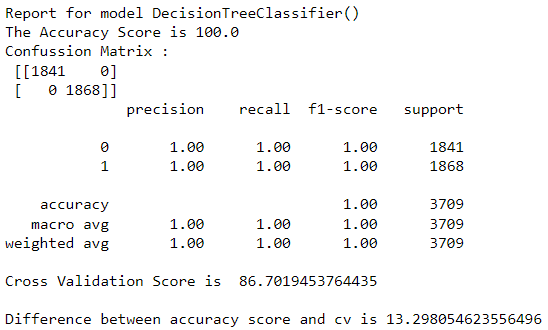
### DecisionTreeClassifier:

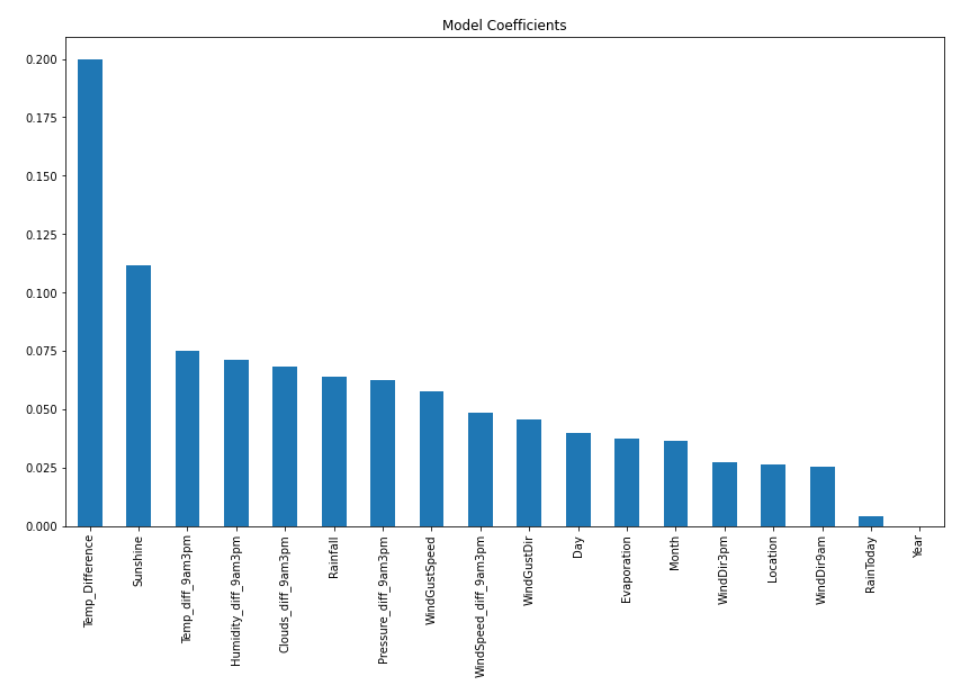
The decision tree classifier (Pang-Ning et al., 2006) creates the classification model by building a decision tree. Each node in the tree specifies a test on an attribute, each branch descending from that node corresponds to one of the possible values for that attribute.

Code:

|  |
| --- |
| from sklearn.tree import DecisionTreeClassifier  dtc = DecisionTreeClassifier()  Model(dtc)  coef = pd.Series(dtc.feature\_importances\_,X.columns).sort\_values(ascending=False)  coef.plot(kind= 'bar', title = 'Model Coefficients',figsize=(14,8))  plt.show() |

Output:



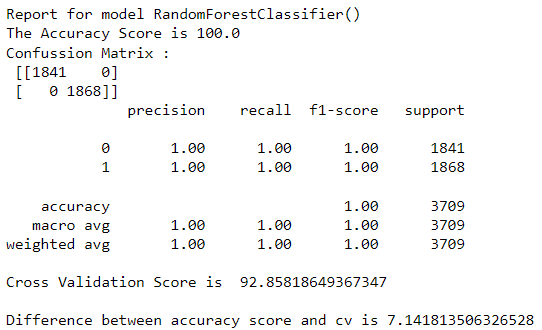


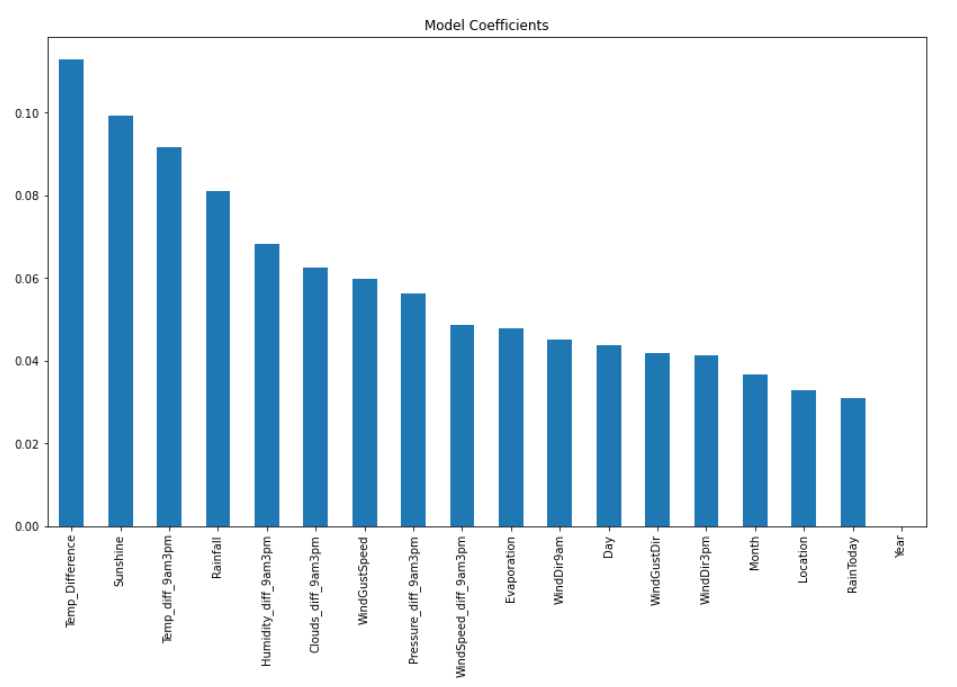
### RandomForestClassifier:

Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.

|  |
| --- |
| from sklearn.ensemble import RandomForestClassifier  rf = RandomForestClassifier()  Model(rf)  coef = pd.Series(rf.feature\_importances\_,X.columns).sort\_values(ascending=False)  coef.plot(kind= 'bar', title = 'Model Coefficients',figsize=(14,8))  plt.show() |

Output:





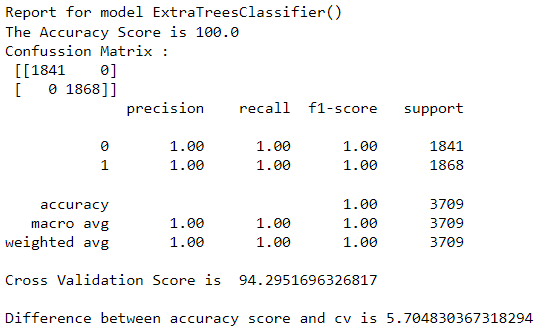
### ExtraTreesClassifier:

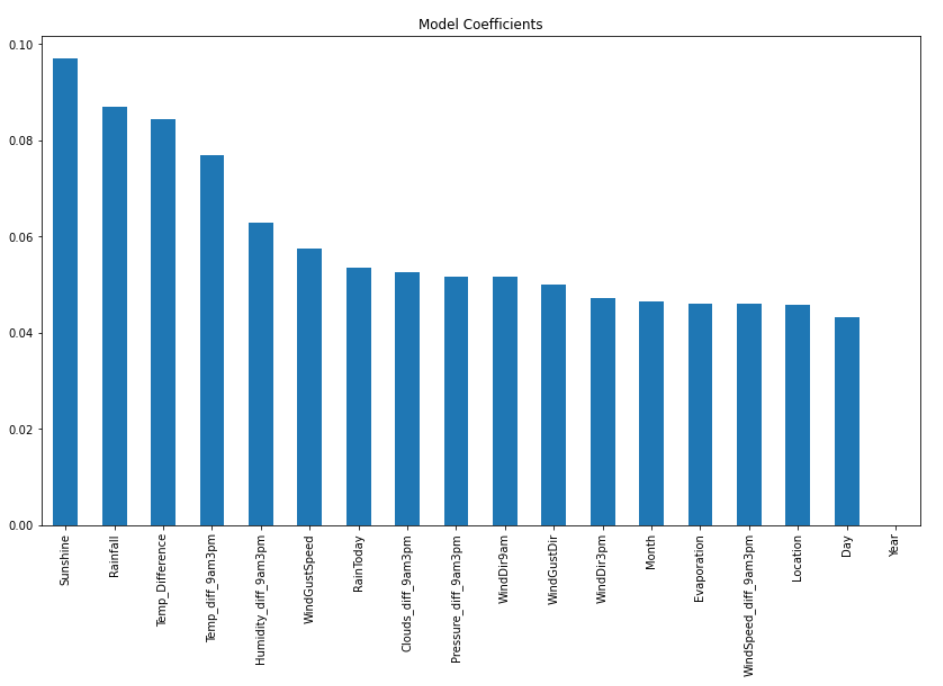
Extremely Randomized Trees Classifier (Extra Trees Classifier) is a type of ensemble learning technique which aggregates the results of multiple de-correlated decision trees collected in a “forest” to output its classification result.

Code:

|  |
| --- |
| from sklearn.ensemble import ExtraTreesClassifier  et = ExtraTreesClassifier()  Model(et)  coef = pd.Series(et.feature\_importances\_,X.columns).sort\_values(ascending=False)  coef.plot(kind= 'bar', title = 'Model Coefficients',figsize=(14,8))  plt.show() |

Output:





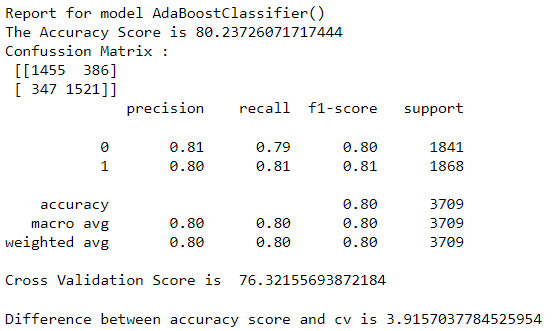
### AdaBoostClassifier:

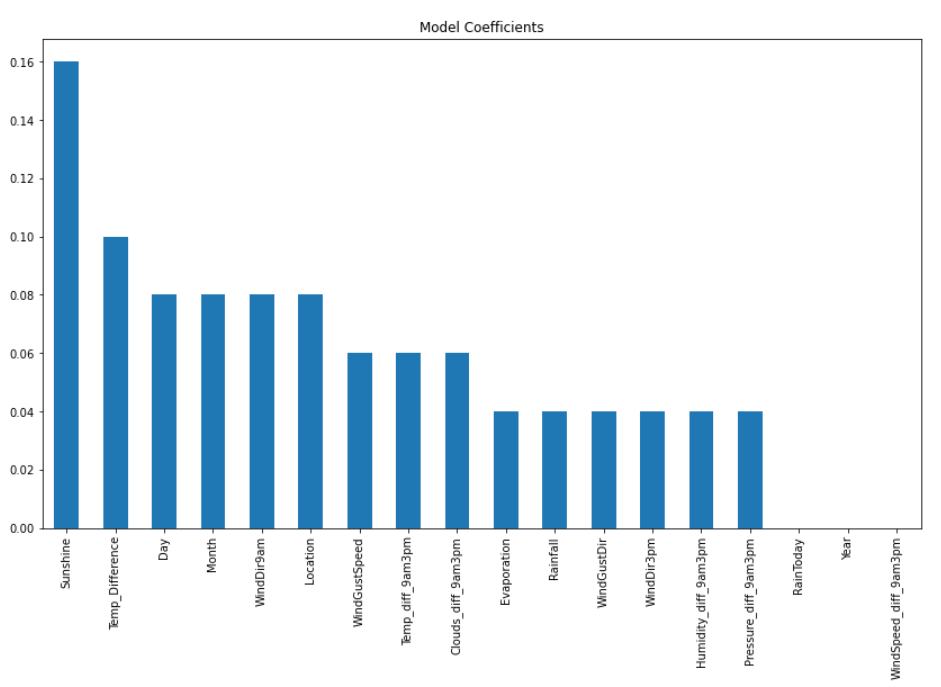
It is an ensemble technique which combines multiple classifiers to increase the accuracy of classifiers. AdaBoost is an iterative ensemble method.

Code:

|  |
| --- |
| from sklearn.ensemble import AdaBoostClassifier  ad = AdaBoostClassifier()  Model(ad)  coef = pd.Series(ad.feature\_importances\_,X.columns).sort\_values(ascending=False)  coef.plot(kind= 'bar', title = 'Model Coefficients',figsize=(14,8))  plt.show() |

Output:



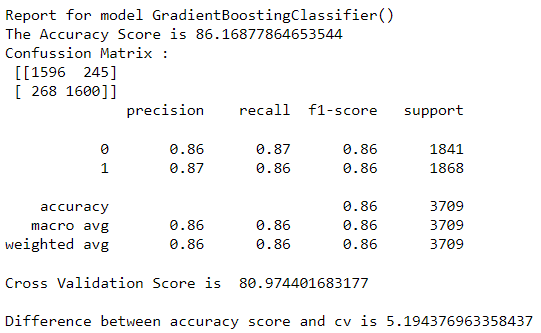
GradientBoostingClassifier:

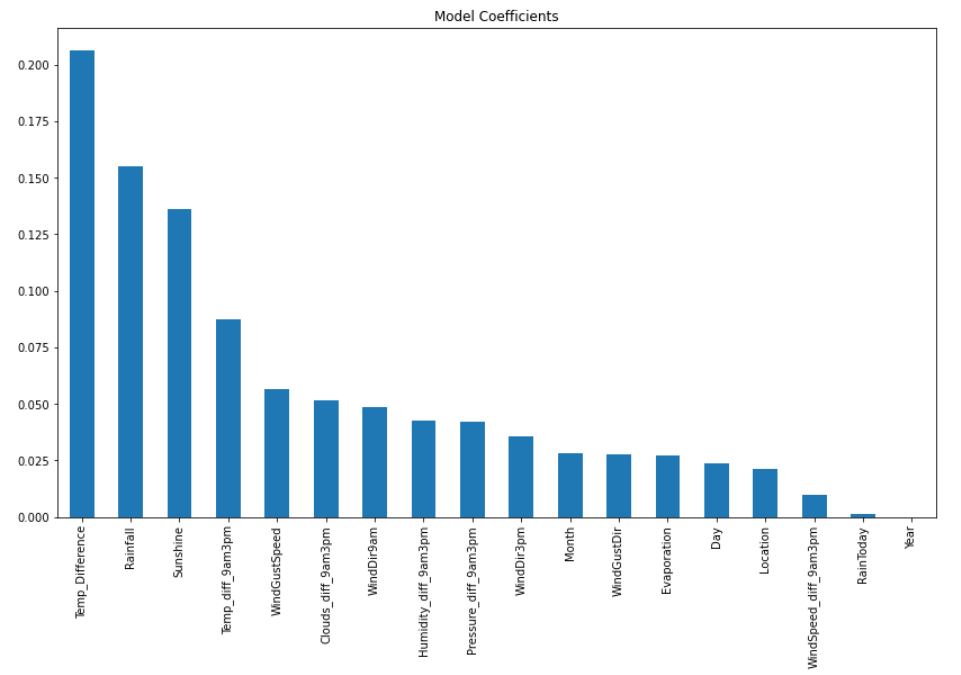
Gradient boosting classifiers are a group of machine learning algorithms that combine many weak learning models together to create a strong predictive model.

Code:

|  |
| --- |
| from sklearn.ensemble import GradientBoostingClassifier  gb = GradientBoostingClassifier()  Model(gb)  coef = pd.Series(gb.feature\_importances\_,X.columns).sort\_values(ascending=False)  coef.plot(kind= 'bar', title = 'Model Coefficients',figsize=(14,8))  plt.show() |

Output:



HyperParameter Tuning:

HyperParameter Tuning is a process where in we fine tune the parameters of an algorithm to try and achieve the best possible accuracy for a Model.

We would be using GridSearchCV to fine tune the parameters of an algorithm.

Importing the Library:

|  |
| --- |
| from sklearn.model\_selection import GridSearchCV |

### Tuning Logistic Regression:

Code:

|  |
| --- |
| lr = LogisticRegression()    parameters = {'penalty':['l1', 'l2', 'elasticnet', 'none'],'C':[0.001,0.0011,0.01,0.1,1,10],'solver':['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']}  gcv = GridSearchCV(lr,parameters,cv=102,n\_jobs=-1)  gcv.fit(x\_train,y\_train)  gcv.best\_params\_ |

Output:



Let’s check the best estimators.

|  |
| --- |
| gcv.best\_estimator\_ |

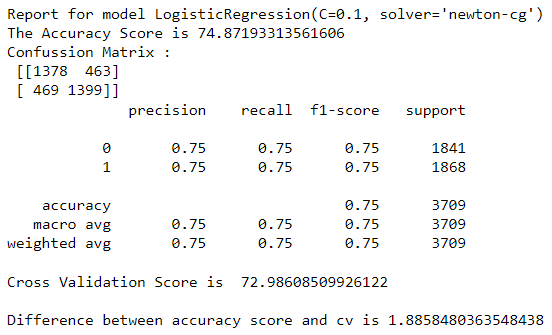
Output:



Checking the model performance with new hyper parameters

|  |
| --- |
| lr = LogisticRegression(C=0.1, solver='newton-cg')  Model(lr) |

Output:



### Tuning KNeighborsClassifier:

Code:

|  |
| --- |
| knn = KNeighborsClassifier()  parameters = {'n\_neighbors':list(range(5,15)),'algorithm':['auto', 'ball\_tree', 'kd\_tree', 'brute'],'leaf\_size':list(range(10,50,10))}  gcv = GridSearchCV(knn,parameters,cv=102,n\_jobs=-1)  gcv.fit(x\_train,y\_train)  gcv.best\_params\_ |

Output:



Let’s check the best estimators.

|  |
| --- |
| gcv.best\_estimator\_ |

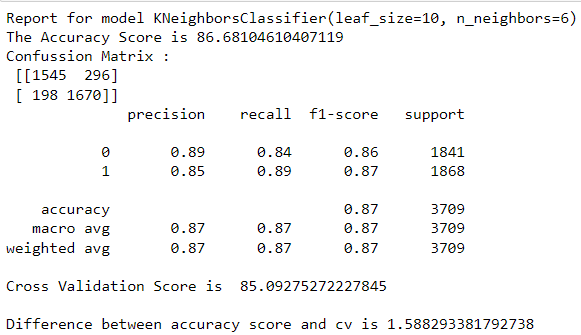
Output:



Checking the model performance with new hyper parameters

|  |
| --- |
| knn = KNeighborsClassifier(leaf\_size=10, n\_neighbors=6)  Model(knn) |

Output:

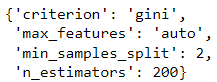


### Tuning ExtraTreesClassifier:

Code:

|  |
| --- |
| et = ExtraTreesClassifier()    parameters = {'n\_estimators':list(range(50,501,50)),  'criterion':["gini", "entropy"],  'max\_features':["auto", "sqrt", "log2"],  'min\_samples\_split':[2,3,4,5,6,7,8,9]  }  gcv = GridSearchCV(et,parameters,cv=102,n\_jobs=-1)  gcv.fit(x\_train,y\_train)  gcv.best\_params\_ |

Output:



Let’s check the best estimators.

|  |
| --- |
| gcv.best\_estimator\_ |

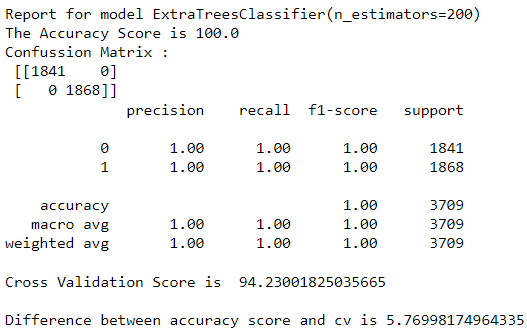
Output:



Checking the model performance with new hyper parameters

|  |
| --- |
| et = ExtraTreesClassifier(n\_estimators=200)  Model(et) |

Output:



### Tuning AdaBoostClassifier:

Code:

|  |
| --- |
| ad = AdaBoostClassifier()  parameters = {'n\_estimators':list(range(10,201,10)),'learning\_rate':[0.001,0.01,1,10],'algorithm':['SAMME', 'SAMME.R']}  gcv = GridSearchCV(ad,parameters,cv=102)  gcv.fit(x\_train,y\_train)  gcv.best\_params\_ |

Output:



Let’s check the best estimators.

Code:

|  |
| --- |
| gcv.best\_estimator\_ |

Output:

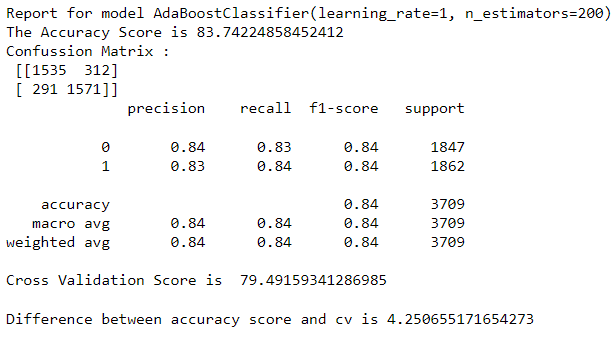


Checking the model performance with new hyper parameters

Code:

|  |
| --- |
| ad = AdaBoostClassifier(learning\_rate=1, n\_estimators=200)  Model(ad) |

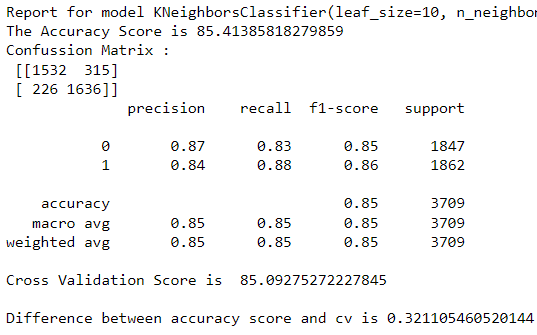
Output:



## Final Model: KNeighborsClassifier with HyperParameter Tuning¶

Code:

|  |
| --- |
| knn = KNeighborsClassifier(leaf\_size=10, n\_neighbors=6)  Model(knn) |



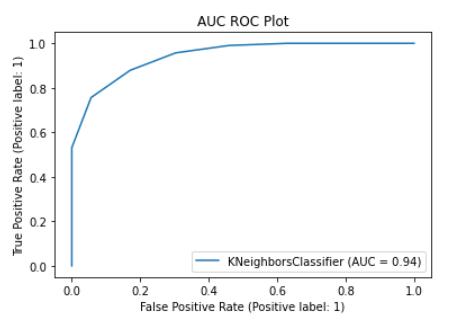
## AUC-ROC Curve:

The AUC-ROC metric clearly helps determine and tell us about the capability of a model in distinguishing the classes. The higher the score, the better is the Model performance.

Code:

|  |
| --- |
| from sklearn.metrics import plot\_roc\_curve  plot\_roc\_curve(knn,x\_test,y\_test)  plt.title('AUC ROC Plot')  plt.show() |

Output:



Saving the Model:

|  |
| --- |
| import pickle  file\_name = 'Weather\_Aus\_Rain\_Tomorrow.pkl'  pickle.dump(knn,open(file\_name,'wb')) |

# Part B of the Problem: Predict how much rainfall could be there.

The initial steps till splitting the dataset remain the same if we are to build a model just to predict the amount of rainfall. Here while splitting the Dataset, we need to have the target column as Rainfall. The rest of the steps are pretty much-repeated till model building.

Splitting the Dataset:

|  |
| --- |
| x = df.drop('Rainfall',axis=1)  x.shape |

(8005, 24)

|  |
| --- |
| y = df['Rainfall']  y.shape |

(8005,)

## Removing skewness using Power\_transform:

In the code below, we remove the skewness.

|  |
| --- |
| from sklearn.preprocessing import power\_transform  X = power\_transform(x)  X = pd.DataFrame(X,columns=x.columns) |

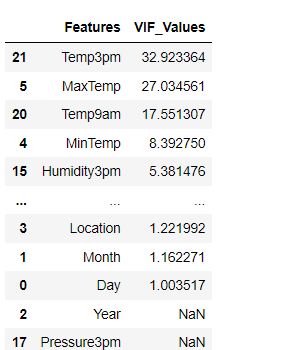
## VIF:

We set the vif threshold as 5 and in the code below we try and work on the columns with higher vif to bring it down below our threshold of 5.

|  |
| --- |
| from statsmodels.stats.outliers\_influence import variance\_inflation\_factor  def calculate\_vif(dataset):  vif = pd.DataFrame()  vif['Features'] = dataset.columns  vif['VIF\_Values'] = [variance\_inflation\_factor(dataset.values,i)for i in range(dataset.shape[1])]  return vif.sort\_values(by='VIF\_Values',ascending=False) |

|  |
| --- |
| calculate\_vif(X) |

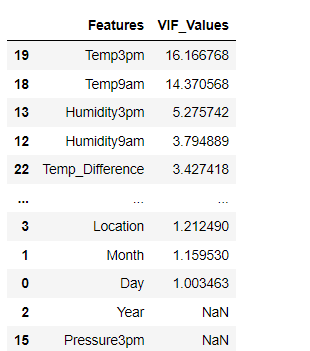
Output:



We use feature engineering as used earlier to reduce the VIF.

|  |
| --- |
| X['Temp\_Difference'] = X['MaxTemp']-X['MinTemp']  X.drop(['MaxTemp','MinTemp'],axis=1,inplace=True)  calculate\_vif(X) |

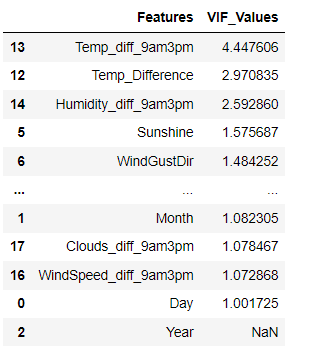
Calculating the VIF again.



Working on other columns with Higher VIF than threshold.

|  |
| --- |
| X['Pressure\_diff\_9am3pm'] = X['Pressure3pm']-X['Pressure9am']  X['WindSpeed\_diff\_9am3pm'] = X['WindSpeed3pm']-X['WindSpeed9am']  X['Clouds\_diff\_9am3pm'] = X['Cloud3pm']-X['Cloud9am']  X.drop(['Pressure3pm','Pressure9am','Cloud3pm','Cloud9am','WindSpeed3pm','WindSpeed9am'],axis=1,inplace=True)  calculate\_vif(X) |

Output:

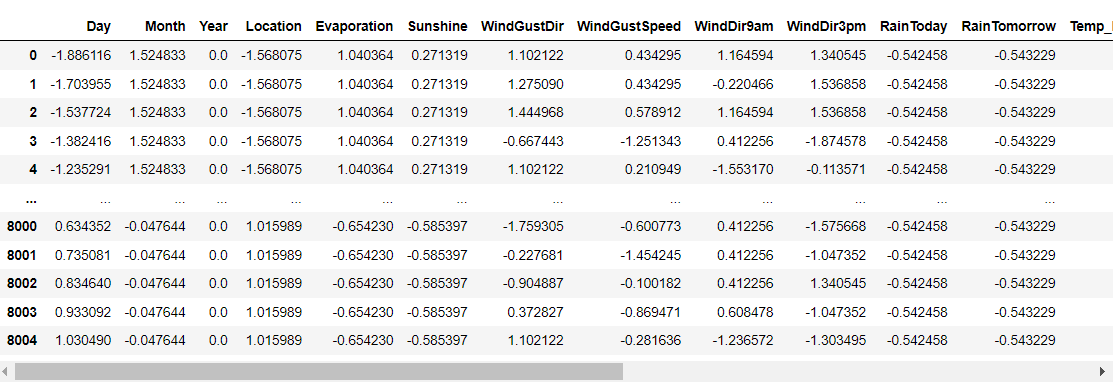


## Scaling the Data:

We use power\_transform method again to scale the data

|  |
| --- |
| from sklearn.preprocessing import StandardScaler  dfx = sc.fit\_transform(X)  X = pd.DataFrame(dfx,columns=X.columns)  X |

Output:



## Model Building:

This is pretty much the part from where thing change drastically in-terms of algorithms used and their functioning.

In the code below we import all the necessary libraries and the algorithms.

|  |
| --- |
| from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LinearRegression,Ridge,Lasso,ElasticNet  from sklearn.svm import SVR  from sklearn.neighbors import KNeighborsRegressor  from sklearn.ensemble import RandomForestRegressor  from sklearn.ensemble import ExtraTreesRegressor  from sklearn.ensemble import BaggingRegressor  from sklearn.ensemble import GradientBoostingRegressor  from sklearn.tree import DecisionTreeRegressor  from sklearn.model\_selection import cross\_val\_score  from sklearn.metrics import accuracy\_score,mean\_absolute\_error,r2\_score,mean\_squared\_error  lr = LinearRegression()  ridge = Ridge()  lasso = Lasso()  knn = KNeighborsRegressor()  rf = RandomForestRegressor()  et = ExtraTreesRegressor()  dt = DecisionTreeRegressor()  br = BaggingRegressor()  gbr = GradientBoostingRegressor()  en = ElasticNet()  models = [lr,ridge,lasso,knn,rf,et,dt,br,gbr,en] |

We choose our base Model to be ExtratreeRegressor and in the code below we try to find the best random\_state for our model.

|  |
| --- |
| max\_accuracy = 0  max\_random\_state = 0  for j in range(0,100):  x\_train,x\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.3,random\_state=j)  et.fit(x\_train,y\_train)  pred\_test = et.predict(x\_test)  accuracy = r2\_score(y\_test,pred\_test)  if accuracy > max\_accuracy:  max\_accuracy =accuracy  max\_random\_state = j  print('Model :', et)  print('Max Accuracy :',max\_accuracy)  print('Best Random State',max\_random\_state) |

Output:

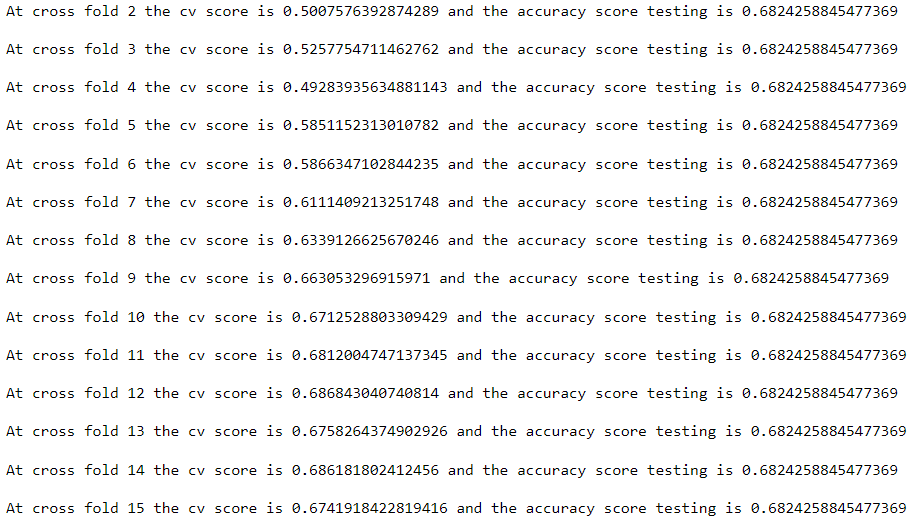


Finding the best fit Cross folds, that produce the highest cross-validation score

Code:

|  |
| --- |
| from sklearn.model\_selection import cross\_val\_score  for j in range(2,16):  cv\_score = cross\_val\_score(et,X,y,cv=j)  cv\_mean = cv\_score.mean()  print(f"At cross fold {j} the cv score is {cv\_mean} and the accuracy score testing is {max\_accuracy}")  print() |

Output:



We see that we achieve the highest cv score at 11 cross-folds.

In the code below we define the model, so we could use the defined Model rather than typing the same lines of code again and again.

|  |
| --- |
| def Model\_reg(Model):  Model.fit(x\_train,y\_train)  pred = Model.predict(x\_test)  cv\_score = cross\_val\_score(Model,X,y,cv = 11)  cv\_score = np.abs(np.mean(cv\_score))\*100  print('Model Report')  print('MSE',mean\_squared\_error(y\_test,pred))  print('Mean Absolute Error',mean\_absolute\_error(y\_test,pred\_test))  r2\_Score = r2\_score(y\_test,pred\_test)\*100  print('r2\_score',r2\_Score)  print('CV', cv\_score)  coef = pd.Series(Model.coef\_,X.columns).sort\_values(ascending=False)  coef.plot(kind= 'bar', title = 'Model Coefficients')  print('Difference between r2\_score and cv is ',r2\_Score-cv\_score) |

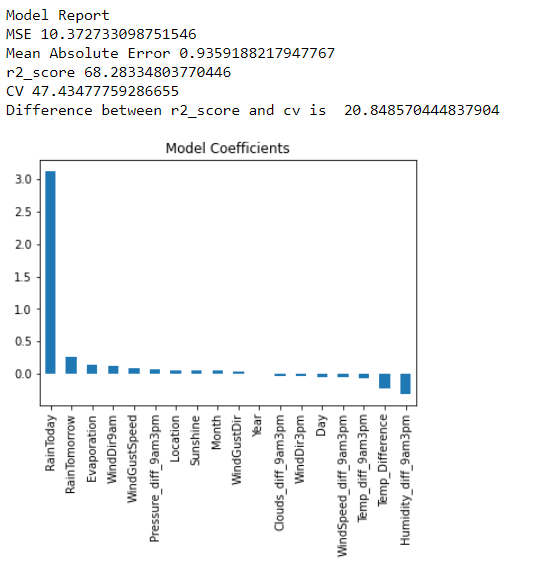
Let us now evaluate the performance using different Algorithms.

### Linear Regression:

Code:

|  |
| --- |
| Model\_reg(lr) |

Output:

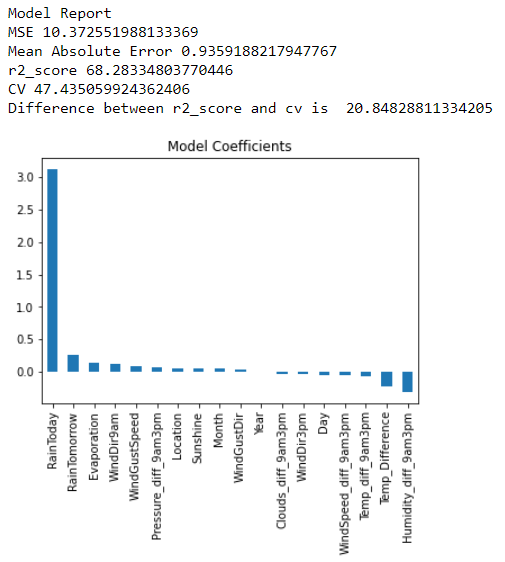


### Ridge:

Code:

|  |
| --- |
| Model\_reg(ridge) |

Output:



Let’s now try tree based and ensemble models.

Defining a Model for tree-based approach.

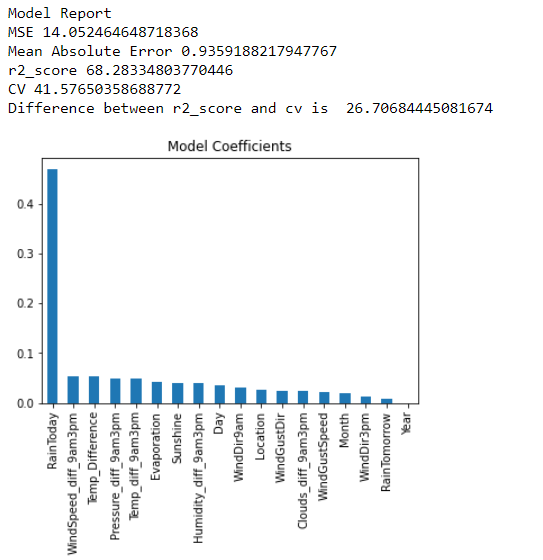
|  |
| --- |
| def Model\_Tree(Model):  Model.fit(x\_train,y\_train)  pred = Model.predict(x\_test)  cv\_score = cross\_val\_score(Model,X,y,cv = 11)  cv\_score = np.abs(np.mean(cv\_score))\*100  print('Model Report')  print('MSE',mean\_squared\_error(y\_test,pred))  print('Mean Absolute Error',mean\_absolute\_error(y\_test,pred\_test))  r2\_Score = r2\_score(y\_test,pred\_test)\*100  print('r2\_score',r2\_Score)  print('CV', cv\_score)  coef = pd.Series(Model.feature\_importances\_,X.columns).sort\_values(ascending=False)  coef.plot(kind= 'bar', title = 'Model Coefficients')  print('Difference between r2\_score and cv is ',r2\_Score-cv\_score) |

### Decision Tree:

Code:

|  |
| --- |
| Model\_Tree(dt) |

Output:

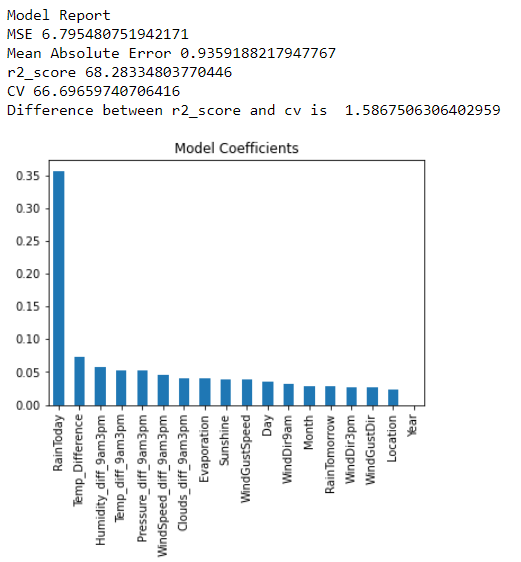


### RandomForest Regressor:

Code:

|  |
| --- |
| Model\_Tree(rf) |

Output:



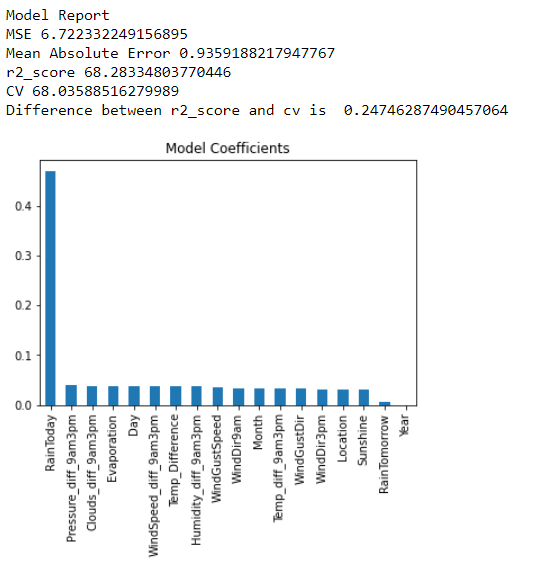
We see that up-until now RandomForest has given the best output where in the difference between the R2 Score and CV Score is the least.

### ExtraTree Regressor:

Code:

|  |
| --- |
| Model\_Tree(et) |

Output:



## HyperParameter Tuning:

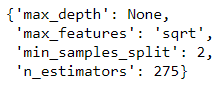
Importing GridSearchCV for HyperParameter Tuning.

|  |
| --- |
| from sklearn.model\_selection import GridSearchCV |

### Tuning The Parameters for RandomForest:

|  |
| --- |
| rf = RandomForestRegressor()  parameters = {'n\_estimators': list(range(25,501,25)),  'max\_features': ['auto', 'sqrt'],  'max\_depth': [2,3,5,None],  'min\_samples\_split': [1,2,3,4]}  gcv= GridSearchCV(rf,parameters,cv=3,n\_jobs=-1)  gcv.fit(x\_train,y\_train)  gcv.best\_params\_ |

Output:



Getting the best estimators:

|  |
| --- |
| gcv.best\_estimator\_ |

Output:

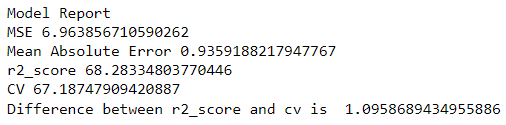


Checking the Models Performance after HyperParameter Tuning

Code:

|  |
| --- |
| rf = RandomForestRegressor(max\_features='sqrt', n\_estimators=275)  Model\_Tree(rf) |

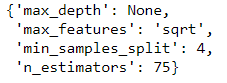
Output:



### Tuning ExtraTree Regressor:

|  |
| --- |
| et = ExtraTreesRegressor()  parameters = {'n\_estimators':[10,20,50,75],  'max\_depth':[None,5,10],  'max\_features':["auto", "sqrt", "log2"],  'min\_samples\_split':[1,4,5,6]}  gcv= GridSearchCV(et,parameters,cv=3,n\_jobs=-1)  gcv.fit(x\_train,y\_train)  gcv.best\_params\_ |

Output:



Checking Best estimators:

|  |
| --- |
| gcv.best\_estimator\_ |

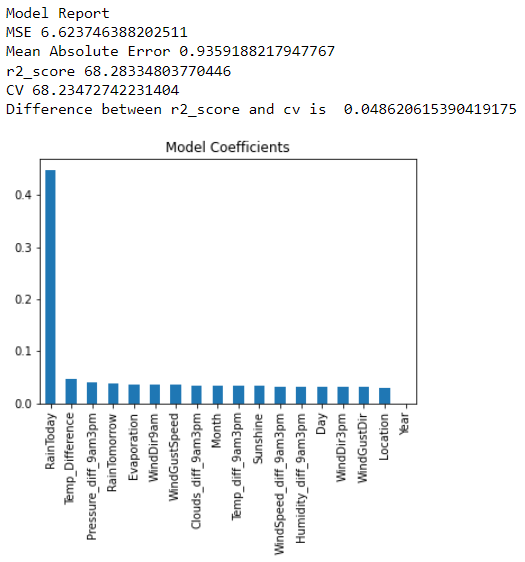
Output:



Checking Models Performance after HyperParameter Tuning

|  |
| --- |
| et = ExtraTreesRegressor(max\_features='sqrt', min\_samples\_split=4, n\_estimators=75)  Model\_Tree(et) |

Output:



We see that the best Model from all the Models is ExtraTreeRegressor with HyperParameter Tuning. Let's Save the Model.

|  |
| --- |
| import pickle  filename = 'Weather\_Australia.pkl'  pickle.dump(et,open(filename,'wb')) |

## Conclusion:

We see that this marks the end of the program. In this we have built a classification Model that predicts if it would Rain tomorrow or not in the Part A, where in we have finalized KNeighborsClassifier with HyperParameter Tuning as our Final Model as it was the model that had the best performance in terms of accuracy of around 85%and cross-validation. The AUC Scores were also good clocking in a score of around 94%.

In the Second Part where in we predict the amount of rainfall, the best performing models were Ensemble models as the base model was chosen to be an Ensemble model and the random\_state and cv were optimized for it. We did achieve a fairly good score of about 68% with a were low difference in Cross-Validation and a good score for RMSE, hence it was chosen as a Final Model.