# Rainfall Predictions – Weather Forecasting

## 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.

## Exploratory Data Analysis

First step in data analysis is to import the libraries

## Importing Required Libraries:

import numpy as np

import pandas as pd

import sklearn

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings('ignore')

## Loading Dataset:

Dataset can be loaded using a method **read\_csv**

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

df\_n

We have a lot of columns so to know the column names we are going to write a code which is

df\_n.columns

To know the shape of the dataset, then

df\_n.shape

Therefore we have a dataset of

1. Rows = 8425
2. Columns = 23

To know the First 5 columns then,

df\_n.head()

To know the last 5 rows of the dataset then,

df\_n.tail()

To know the random row of the dataset then

df\_n.sample()

To know the type of the dataset, and if any null values presented or not and to know how many columns are there, then

df\_n.info()

We can see the null values is presented we can treat them in further steps apart from this if we see our dataset we have many unnecessary columns therefore, we shall drop many columns with a function drop() and axis=1 refers to columns

df=df\_n.drop(['Date','Location','Rainfall','WindGustDir','WindGustSpeed','WindDir9am','WindDir3pm','WindSpeed9am','WindSpeed3pm','Humidity9am','Pressure9am','Pressure3pm','Cloud9am','Temp9am' ,'RainToday'],axis=1)

Now we can see the proper dataset with necessary columns for the model building and we have named the new dataset has df.

Now we are going to see the statistical summary of the new dataset by .describe() functions.

df.describe()

|  | **MinTemp** | **MaxTemp** | **Evaporation** | **Sunshine** | **Humidity3pm** | **Cloud3pm** | **Temp3pm** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 8350.000000 | 8365.000000 | 4913.000000 | 4431.000000 | 8323.000000 | 5970.000000 | 8329.000000 |
| **mean** | 13.193305 | 23.859976 | 5.389395 | 7.632205 | 51.249790 | 4.503183 | 22.442934 |
| **std** | 5.403596 | 6.136408 | 5.044484 | 3.896235 | 18.423774 | 2.731659 | 5.980020 |
| **min** | -2.000000 | 8.200000 | 0.000000 | 0.000000 | 6.000000 | 0.000000 | 7.300000 |
| **25%** | 9.200000 | 19.300000 | 2.600000 | 4.750000 | 39.000000 | 2.000000 | 18.000000 |
| **50%** | 13.300000 | 23.300000 | 4.600000 | 8.700000 | 51.000000 | 5.000000 | 21.900000 |
| **75%** | 17.400000 | 28.000000 | 7.000000 | 10.700000 | 63.000000 | 7.000000 | 26.400000 |
| **max** | 28.500000 | 45.500000 | 145.000000 | 13.900000 | 99.000000 | 8.000000 | 44.100000 |

We can clearly see that different counts for different columns. We can see the mean , standard deviation, minimum and maximum values of each columns in these statistical summary.

Now we are going to take a look at the categorical column to know how much values has the yes and no has

df['RainTomorrow'].value\_counts()

df['RainTomorrow'].hist(grid=False)

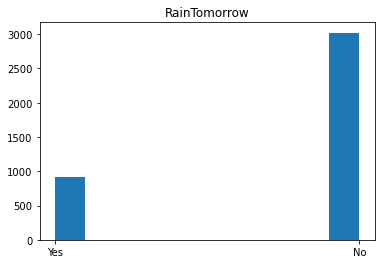
plt.title('RainTomorrow')

plt.show()

No 6195

Yes 1991

Name: RainTomorrow, dtype: int64



So we can see the output of ‘Raintomorrow that it has the counts of yes and no

Now we can check the datatype of the dataset to check how many object columns are there

df.dtypes

Its shows the output which contains

MinTemp float64

MaxTemp float64

Evaporation float64

Sunshine float64

Humidity3pm float64

Cloud3pm float64

Temp3pm float64

RainTomorrow object

dtype: object

Therefore we have only one column as object.

### Checking the null values:

sns.heatmap(df.isnull())

plt.title('Null Values')

plt.show()

It shows the diagram with null values. Therefore to know precisely how much null values is presented then

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

10533

So now we identified the null values. We shall remove the null values using the following function

df.dropna(inplace=True)

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

0

So now we are going to label the object column using lable encoder () function to change into integer column

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

df['RainTomorrow']=le.fit\_transform(df['RainTomorrow'])

So now we are going to check which number is assigned to yes and no by inverse\_transform()method

le.inverse\_transform([1,0])

array(['Yes', 'No'], dtype=object)

So number 1 is assigned to YES and number 0 is assigned to NO.

Let’s check the values that it has,

df['RainTomorrow'].value\_counts()

0 3021

1 921

Name: RainTomorrow, dtype: int64

Now the categorical column has changed to integer columns. So it’s totally fair to do operations,

df.dtypes

MinTemp float64

MaxTemp float64

Evaporation float64

Sunshine float64

Humidity3pm float64

Cloud3pm float64

Temp3pm float64

RainTomorrow int32

dtype: object

Now we have checked the every column dataytype and its clear that it doesn’t contain any object.

## Checking the correlation

To check which column in the dataset has strong correlation then we should use the .corr() function

df.corr()

|  | **MinTemp** | **MaxTemp** | **Evaporation** | **Sunshine** | **Humidity3pm** | **Cloud3pm** | **Temp3pm** | **RainTomorrow** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **MinTemp** | 1.000000 | 0.737507 | 0.497595 | 0.073721 | 0.053815 | 0.051234 | 0.709369 | 0.068591 |
| **MaxTemp** | 0.737507 | 1.000000 | 0.597566 | 0.517222 | -0.420550 | -0.304188 | 0.976599 | -0.189499 |
| **Evaporation** | 0.497595 | 0.597566 | 1.000000 | 0.349778 | -0.314454 | -0.188376 | 0.582955 | -0.106155 |
| **Sunshine** | 0.073721 | 0.517222 | 0.349778 | 1.000000 | -0.627181 | -0.723712 | 0.542085 | -0.480887 |
| **Humidity3pm** | 0.053815 | -0.420550 | -0.314454 | -0.627181 | 1.000000 | 0.506716 | -0.497081 | 0.489199 |
| **Cloud3pm** | 0.051234 | -0.304188 | -0.188376 | -0.723712 | 0.506716 | 1.000000 | -0.354987 | 0.427189 |
| **Temp3pm** | 0.709369 | 0.976599 | 0.582955 | 0.542085 | -0.497081 | -0.354987 | 1.000000 | -0.237371 |
| **RainTomorrow** | 0.068591 | -0.189499 | -0.106155 | -0.480887 | 0.489199 | 0.427189 | -0.237371 | 1.000000 |

Let’s print the correlation values of independent variables with target variable in target form.

corr\_matrix=df.corr()

corr\_matrix['RainTomorrow'].sort\_values(ascending=False)

RainTomorrow 1.000000

Humidity3pm 0.489199

Cloud3pm 0.427189

MinTemp 0.068591

Evaporation -0.106155

MaxTemp -0.189499

Temp3pm -0.237371

Sunshine -0.480887

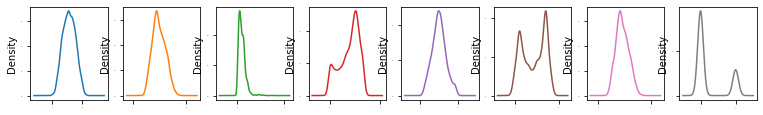
Name: RainTomorrow, dtype: float64

There is a strong positive correlation between Humidity3pm and RainTomorrow also there is weak positive correaltion between the Mintemp and RainTomorrow. There is a strong negative correlation between Evaporation and RainTomorrow also there is a weak negative correlation between Sunshine and RainTomorrow.

Let’s check the data distribution among the columns,

df.plot(kind='density',subplots=True,layout=(6,11),sharex=False,legend=False,fontsize=1,figsize=(18,12))

plt.show()



We can skewness in columns let’s handle the skewness in further steps.

Splitting the Independent and Target variables in x and y before removing the skewness

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

y=df['RainTomorrow']

x

| **MinTemp** | **MaxTemp** | **Evaporation** | **Sunshine** | **Humidity3pm** | **Cloud3pm** | **Temp3pm** |
| --- | --- | --- | --- | --- | --- | --- |
| **907** | 19.8 | 27.1 | 8.6 | 9.0 | 60.0 | 6.0 | 26.0 |
| **908** | 18.7 | 25.6 | 3.8 | 3.4 | 90.0 | 7.0 | 20.9 |
| **909** | 16.5 | 25.5 | 2.8 | 6.1 | 65.0 | 7.0 | 24.5 |
| **910** | 18.5 | 26.9 | 3.2 | 11.0 | 60.0 | 5.0 | 26.4 |
| **911** | 18.2 | 28.2 | 6.8 | 9.0 | 68.0 | 1.0 | 25.9 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **8381** | 25.2 | 34.5 | 4.0 | 9.2 | 46.0 | 7.0 | 33.4 |
| **8382** | 26.2 | 33.3 | 6.0 | 5.9 | 93.0 | 8.0 | 25.2 |
| **8383** | 24.9 | 32.8 | 6.2 | 8.7 | 55.0 | 6.0 | 32.0 |
| **8384** | 25.5 | 33.1 | 4.8 | 7.1 | 58.0 | 7.0 | 31.5 |
| **8385** | 24.9 | 34.3 | 5.6 | 8.6 | 46.0 | 3.0 | 33.2 |

3942 rows × 7 columns

y

907 1

908 1

909 0

910 0

911 1

..

8381 0

8382 1

8383 0

8384 0

8385 0

Name: RainTomorrow, Length: 3942, dtype: int32

Checking the skewness

x.skew().sort\_values(ascending=False)

Evaporation 2.106943

Temp3pm 0.252833

MaxTemp 0.251113

Humidity3pm 0.188305

MinTemp 0.030742

Cloud3pm -0.149415

Sunshine -0.587638

dtype: float64

We can skewness in most of the columns oof our dataset, we will remove the skewness using the

power\_transform function.

### Power Transform Technique:

from sklearn.preprocessing import power\_transform

x\_new=power\_transform(x)

# Checking skewness

pd.DataFrame(x\_new).skew().sort\_values(ascending=False)

2 0.000606

6 -0.020068

1 -0.020337

4 -0.025227

0 -0.070152

5 -0.257103

3 -0.401753

dtype: float64

type(x\_new)

Index(['MinTemp', 'MaxTemp', 'Evaporation', 'Sunshine', 'Humidity3pm',

'Cloud3pm', 'Temp3pm'],

dtype='object')

numpy.ndarray

x.columns

So now we are going to assign the x as the

x=pd.DataFrame(x\_new,columns=x.columns)

x

| **MinTemp** | **MaxTemp** | **Evaporation** | **Sunshine** | **Humidity3pm** | **Cloud3pm** | **Temp3pm** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 1.110607 | 0.454073 | 1.052568 | 0.264864 | 0.602861 | 0.694290 | 0.528344 |
| **1** | 0.907945 | 0.215410 | -0.228903 | -1.190045 | 2.144050 | 1.028530 | -0.327285 |
| **2** | 0.498292 | 0.199253 | -0.641813 | -0.525015 | 0.869056 | 1.028530 | 0.285454 |
| **3** | 0.870948 | 0.422641 | -0.466058 | 0.845611 | 0.602861 | 0.348617 | 0.592009 |
| **4** | 0.815364 | 0.624905 | 0.659641 | 0.264864 | 1.026756 | -1.221027 | 0.512357 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **3937** | 2.087563 | 1.544493 | -0.155788 | 0.321720 | -0.168453 | 1.028530 | 1.641296 |
| **3938** | 2.265581 | 1.376319 | 0.457968 | -0.576865 | 2.291774 | 1.353086 | 0.399636 |
| **3939** | 2.033994 | 1.305344 | 0.510266 | 0.180116 | 0.332097 | 0.694290 | 1.440332 |
| **3940** | 2.141055 | 1.347994 | 0.112484 | -0.260386 | 0.495129 | 1.028530 | 1.367566 |
| **3941** | 2.033994 | 1.516672 | 0.349192 | 0.152012 | -0.168453 | -0.387834 | 1.612832 |

3942 rows × 7 columns

x.skew().sort\_values(ascending=False)

Evaporation 0.000606

Temp3pm -0.020068

MaxTemp -0.020337

Humidity3pm -0.025227

MinTemp -0.070152

Cloud3pm -0.257103

Sunshine -0.401753

dtype: float64

sk=x.skew()

sk

MinTemp -0.070152

MaxTemp -0.020337

Evaporation 0.000606

Sunshine -0.401753

Humidity3pm -0.025227

Cloud3pm -0.257103

Temp3pm -0.020068

dtype: float64

sk[np.abs(sk)>0.5].all()

True

x.skew()[np.abs(x.skew())<0.25].all()

True

Skewness has been removed now we can proceed with further steps.

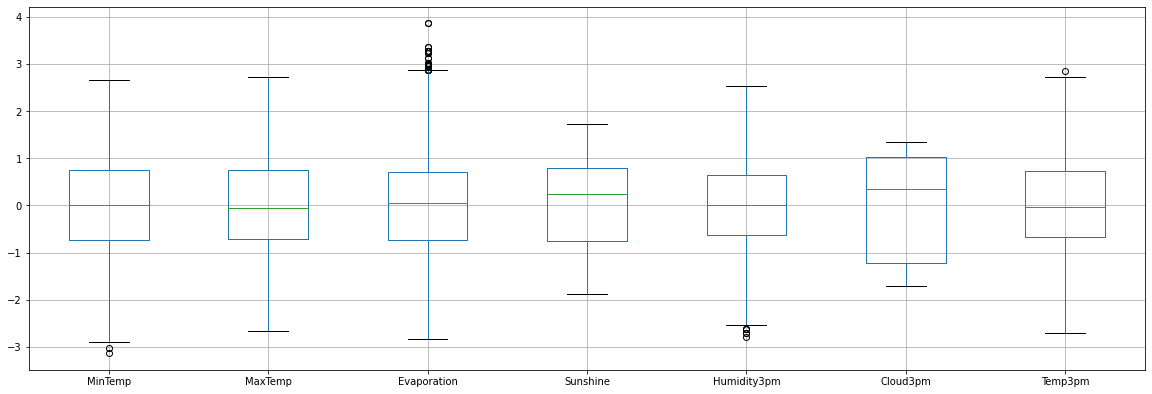
# Checking Outliers:

# Plotting Boxplot for the available columns

x.iloc[:,0:8].boxplot(figsize=[20,8])

plt.subplots\_adjust(bottom=0.25)

plt.show()



We can see outliers in few columns will remove outliers using the following technique,

from scipy.stats import zscore

(np.abs(zscore(x))<3).all()

MinTemp False

MaxTemp True

Evaporation False

Sunshine True

Humidity3pm True

Cloud3pm True

Temp3pm True

dtype: bool

Therefore’ Evaporation’ &’Min temparature’ are contains outliers.

## Remove Outliers:

features=['MinTemp', 'MaxTemp', 'Evaporation', 'Sunshine', 'Humidity3pm',

'Cloud3pm', 'Temp3pm']

# Define a functions called outliers which returns a list of index of outliers

# IQR = q3-q1

def outliers(df,ft):

q1=df[ft].quantile(0.25)

q3=df[ft].quantile(0.75)

iqr=q3-q1

lower\_limit=q1-1.5\*iqr

upper\_limit=q3+1.5\*iqr

ls=df.index[(df[ft]<lower\_limit) | (df[ft]>upper\_limit)]

return ls

In the above method first we save the columns into the features variable and then pass this into the function.

Quantile method is used for removing outliers.

# creating the empty list to store the output variable from multiple columns

index\_list=[]

for i in features:

index\_list.extend(outliers(df,i))

index\_list

Output shows that it stores the list of index values of the outliers present in the dataset.

Now we are going to define a function for removing the outliers from actual dataset and preserves the actual dataset.

# define a function called remove which returns a cleaned dataframe without outliers

def remove(df,ls):

ls=sorted(set(ls))

df=df.drop(ls)

return df

df\_cleaned=remove(df,index\_list)

Above command will store the outliers free dataset in a new dataset name called as df\_cleaned.

df\_cleaned.shape

(3837, 8)

Now the cleaned dataset has,

Rows = 3837

Columns = 8

Now we are going to check the outliers which is removed or not by comparing the older image with newer one ,before that we are going to split the datatset as we have a new cleaned dataset,so lets start

Splitting the independent and dependent variable ,

x=df\_cleaned.drop('RainTomorrow',axis=1)

y=df\_cleaned['RainTomorrow']

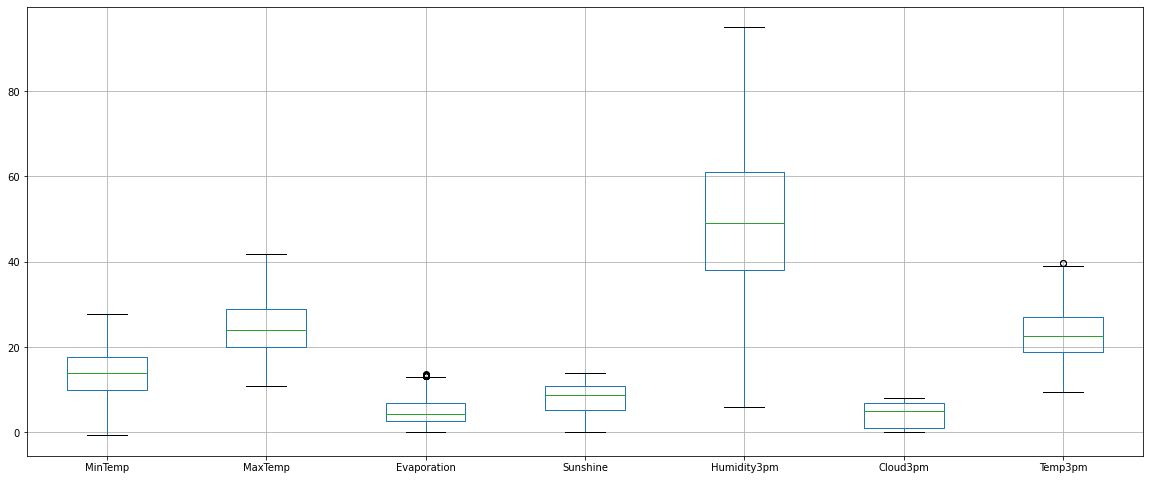
Now the dataset is splitted into x and y, now lets see the outliers,

# Plotting the boxplot for the columns

x.iloc[:,0:8].boxplot(figsize=[20,8])

plt.subplots\_adjust(bottom=0.1)

plt.show()



Above picture depicts the cleaned dataset but somehow it shows outliers because of noise only, lets see the older one with outliers

Outliers image:

### 

### Rainfall Outliers.png

We can compare the outliers with older one and newer one we can clearly see that most of the outliers is removed.

## EDA concluding remarks

Imported the libraries

Loaded the data

Categorical - Histogram ----------> frequency of the classes

Continuous – Density Plot ------> density of the data distribution

Balanced ------------> Frequency of the classes where not equal

Divided my data into -------> Independent variable(x) and dependent variable(y)

Y -------> categorical data ----------> Encoded this variable ==== LabelEncoding

X---------> continuous data ----------> Skewness of the data ===== skewness was found.

Removed the skewness using Transformation technique.

X ========> Power Transformion =======> Skewness was removed

X =========> Does’nt have any skewness (-1,1) or (-4,4) =====> Normal distribution (-0.5 to 0.5) and(-0.25 to 0.25)

Y ========> Encoded

Checked the data for outliers ====> Zscore ======> (-3,3) rule of thumb.

Zscore ===(-3,3) =====> this is the rule of thumb ====> This is the invalid operation

Mean =====> “Yes &No

The Data is Pre-processed and the data is ready for Training process.

## Training Process Begin

Importing required libraries for training process

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix,classification\_report

from sklearn.model\_selection import train\_test\_split

To find the best random state

Best Accuracy is 0.8645833333333334 random state 26

print('Best Accuracy is',maxaccu,'random state',maxrs)

maxaccu=0

maxrs=0

for i in range(1,200):

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.20,random\_state=i)

lr=LogisticRegression()

lr.fit(x\_train,y\_train)

y\_pred=lr.predict(x\_test)

acc=accuracy\_score(y\_test,y\_pred)

print('accuracy',acc,'random state',i)

if acc>maxaccu:

maxaccu=acc

maxrs=i

print('accuracy',maxaccu,'random state',i)

We have found the best random state as 26. We will create our train\_test\_split using this random\_state.

## Creating train\_test\_split:

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.20,random\_state=26)

From this code we can get the test and train dataset with random state 26.

We can also see the shape of test and train dataset.

x\_train.shape, y\_train.shape, x\_test.shape, y\_test.shape

(3069,7) , (768,7) , (3069,) , (768,)

## Model Evaluation:

### Logistic Regression Model :-

Logistic Regression model will be used when the dependent variable of the dataset contains categorical values ie: True or False , Yes or No , like this occurs then we should go for Logistic Regression.

Code to predict Logistic regression model is as follows:

from sklearn.linear\_model import LogisticRegression

lr=LogisticRegression()

lr.fit(x\_train,y\_train)

y\_pred=lr.predict(x\_test)

print('Accuracy',accuracy\_score(y\_test,y\_pred)\*100)

print(confusion\_matrix(y\_test,y\_pred))

print(classification\_report(y\_test,y\_pred))

Therefore we have created the code and should run by pressing Alt + Enter, we can get the accuracy score along with classification report is as follows :-

Accuracy 86.45833333333334

[[564 32]

[ 72 100]]

precision recall f1-score support

0 0.89 0.95 0.92 596

1 0.76 0.58 0.66 172

accuracy 0.86 768

macro avg 0.82 0.76 0.79 768

weighted avg 0.86 0.86 0.86 768

Now we have got an accuracy of 86.45 percentage, we shall move with second model before that lets compare the actual and predicted values through distribution plot.

### Distribution plot for Logistic Regression model : -

ax1=sns.distplot(df['RainTomorrow'],hist=False,color='r',label='Actual Values')

sns.distplot(y\_pred,hist=False,color='b',label='Fitted Values',ax=ax1)

### Rainfall Dist plot = Lr.png

Red distribution line shows the actual values and Blue distribution shows the predicted values,

We can clearly see that blue predicted values is quiet similar to actual values.

### Decision Tree Classifier Model :-

### 

from sklearn.tree import DecisionTreeClassifier

dt=DecisionTreeClassifier()

dt.fit(x\_train,y\_train)

y\_pred2=dt.predict(x\_test)

print('Accuracy',accuracy\_score(y\_test,y\_pred2)\*100)

print(confusion\_matrix(y\_test,y\_pred2))

print(classification\_report(y\_test,y\_pred))

### 

Accuracy 83.72395833333334

[[540 56]

[ 69 103]]

precision recall f1-score support

0 0.89 0.95 0.92 596

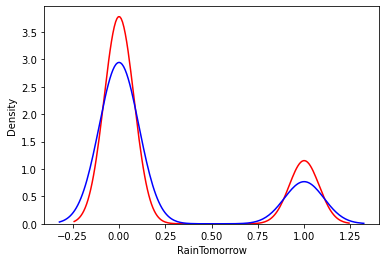
1 0.76 0.58 0.66 172

accuracy 0.86 768

macro avg 0.82 0.76 0.79 768

weighted avg 0.86 0.86 0.86 768

Accuracy of decision tree classifier model is 83.72 which is less than the accuracy of logistic regression model so lets see the distribution plot of decision tree



Distribution is not similar to the actual values. Therefore it will not be used for model predicting

### Random Forest Classifier Model :-

from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier()

rf.fit(x\_train,y\_train)

y\_pred3=rf.predict(x\_test)

print('Accuracy',accuracy\_score(y\_test,y\_pred3)\*100)

print(confusion\_matrix(y\_test,y\_pred3))

print(classification\_report(y\_test,y\_pred3))

Accuracy 88.671875

[[571 25]

[ 62 110]]

precision recall f1-score support

0 0.90 0.96 0.93 596

1 0.81 0.64 0.72 172

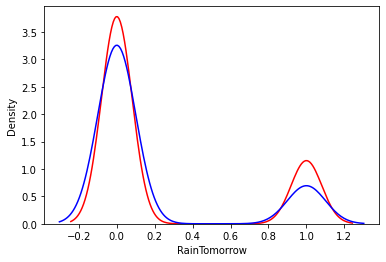
accuracy 0.89 768

macro avg 0.86 0.80 0.82 768

weighted avg 0.88 0.89 0.88 768

In these model we get an accuracy of 88.67 which is nearly 89 percent so this model predicts with an accuracy of 89 percent so let’s see the distribution plot.

### Distribution plot for Random Forest :-



This Distribution plot shows actual dataset is quite similar to the predicted dataset. So this dataset can be considered in model building.

### Support Vector Classifier Model :-

from sklearn.svm import SVC

svc=SVC()

svc.fit(x\_train,y\_train)

y\_pred4=svc.predict(x\_test)

print('Accuracy',accuracy\_score(y\_test,y\_pred4)\*100)

print(confusion\_matrix(y\_test,y\_pred4))

print(classification\_report(y\_test,y\_pred4))

Accuracy 86.71875

[[577 19]

[ 83 89]]

precision recall f1-score support

0 0.87 0.97 0.92 596

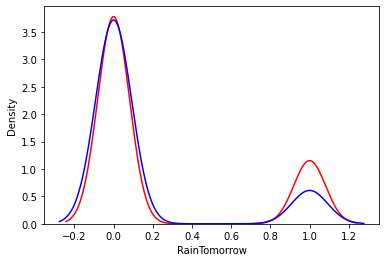
1 0.82 0.52 0.64 172

accuracy 0.87 768

macro avg 0.85 0.74 0.78 768

weighted avg 0.86 0.87 0.86 768

Support Vector model predicts with an accuracy of 86.71 nearly 87 percent which is greater than the Logistic regression model so let’s see the data distribution of SVC model.



Data distribution of Actual values and Predicted values of SVC is more accurate than Random forest model. So this model will be considered for building model.

### AdaBoost Classifier Model :-

from sklearn.ensemble import AdaBoostClassifier

ad=AdaBoostClassifier()

ad.fit(x\_train,y\_train)

y\_pred5=ad.predict(x\_test)

print('Accuracy',accuracy\_score(y\_test,y\_pred5)\*100)

print(confusion\_matrix(y\_test,y\_pred5))

print(classification\_report(y\_test,y\_pred5))

Accuracy 85.41666666666666

[[560 36]

[ 76 96]]

precision recall f1-score support

0 0.88 0.94 0.91 596

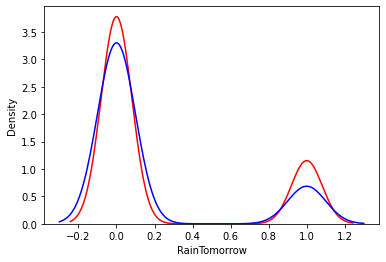
1 0.73 0.56 0.63 172

accuracy 0.85 768

macro avg 0.80 0.75 0.77 768

weighted avg 0.85 0.85 0.85 768

This AdaBoost model predicts with an accuracy of 85.416 ie 85 percent so this model is probably considered for model building. Let’s see the data distribution of adaboost model.



Data distribution of adaboost classifier is not better than SVC model but similar to Decision Tree model and so this model will not be considered for model building.

### Gradient Boosting Classifier Model :-

from sklearn.ensemble import GradientBoostingClassifier

gb=GradientBoostingClassifier()

gb.fit(x\_train,y\_train)

y\_pred6=gb.predict(x\_test)

print('Accuracy',accuracy\_score(y\_test,y\_pred6)\*100)

print(confusion\_matrix(y\_test,y\_pred6))

print(classification\_report(y\_test,y\_pred6))

Accuracy 86.328125

[[561 35]

[ 70 102]]

precision recall f1-score support

0 0.89 0.94 0.91 596

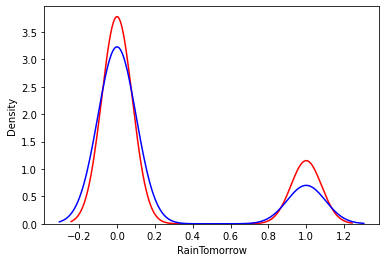
1 0.74 0.59 0.66 172

accuracy 0.86 768

macro avg 0.82 0.77 0.79 768

weighted avg 0.86 0.86 0.86 768

Accuracy of Gradient boosting model is greater than the decision tree model. Let’s see the data distribution of gradient boosting model.



Data distribution of gradient boosting is more similar to the Random Forest model. So this model could be considered for model building.

## Concluding Remarks :

Random Forest Classifier model perform well with an accuracy of 89% and after that Support vector is perfoming well with an accuracy of 87%, Logistic regression,Gradient Boosting performs well with an accuracy of 86.45%.