Rain Prediction using ML

Time Line of the Project :

- Data Analysis
- Handling Missing Values
- Handling Categorical Varibales
- Feature Engineering
- Model Building using ML
- Model Building using Auto ML i.e PyCaret

Importing Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report
from sklearn import metrics
import math
%matplotlib inline

from google.colab import drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).

df=  pd.read_csv("/content/weatherAUS.csv")
pd.set_option("display.max_columns", None)

df
```

{"type":"dataframe","variable_name":"df"}

```
df.nunique()

Date              3436
Location            49
MinTemp            389
MaxTemp            505
Rainfall           681
Evaporation        358
Sunshine           145
WindGustDir         16
WindGustSpeed       67
```

```
WindDir9am          16
WindDir3pm          16
WindSpeed9am        43
WindSpeed3pm        44
Humidity9am        101
Humidity3pm        101
Pressure9am        546
Pressure3pm        549
Cloud9am            10
Cloud3pm            10
Temp9am            441
Temp3pm            502
RainToday            2
RainTomorrow         2
dtype: int64
```

Discrete Variable are countable in finit amount of time while numerical variable are to much in number to count

```
# Numerical features
num_var = df.select_dtypes(include=['int64',
'float64']).columns.tolist()

# Discrete numerical features (few unique values)
discrete_var = [col for col in num_var if df[col].nunique() <= 25]

# Continuous numerical features
cont_var = [col for col in num_var if col not in discrete_var]

# Categorical features (object or category type)
categ_var = df.select_dtypes(include=['object']).columns.tolist()

df[categ_var]
```

{"type":"dataframe"}

Handling Missing Values

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

```
Date                0
Location            0
MinTemp          1485
MaxTemp          1261
Rainfall         3261
Evaporation     62790
Sunshine        69835
WindGustDir     10326
WindGustSpeed   10263
WindDir9am      10566
```

```
WindDir3pm        4228
WindSpeed9am      1767
WindSpeed3pm      3062
Humidity9am       2654
Humidity3pm       4507
Pressure9am      15065
Pressure3pm      15028
Cloud9am         55888
Cloud3pm         59358
Temp9am           1767
Temp3pm           3609
RainToday         3261
RainTomorrow      3267
dtype: int64
```

the percentage of missing values in each column.

```
df.isnull().sum()*100/len(df)
```

```
Date              0.000000
Location          0.000000
MinTemp           1.020899
MaxTemp           0.866905
Rainfall          2.241853
Evaporation      43.166506
Sunshine         48.009762
WindGustDir       7.098859
WindGustSpeed     7.055548
WindDir9am        7.263853
WindDir3pm        2.906641
WindSpeed9am      1.214767
WindSpeed3pm      2.105046
Humidity9am       1.824557
Humidity3pm       3.098446
Pressure9am      10.356799
Pressure3pm      10.331363
Cloud9am         38.421559
Cloud3pm         40.807095
Temp9am           1.214767
Temp3pm           2.481094
RainToday         2.241853
RainTomorrow      2.245978
dtype: float64
```

```
def find_var_type(var):


    if var in discrete_var:
        print("{} is a Numerical Variable, Discrete in 
nature".format(var))
```

```
    elif var in cont_var :
        print("{} is a Numerical Variable, Continuous in 
nature".format(var))
    else :
        print("{} is a Categorical Variable".format(var))

find_var_type('Cloud3pm')
```

```
Cloud3pm is a Numerical Variable, Discrete in nature
```

Ramdom Sample Imputation for the our variables which are having the most percentage of Nul Vlaues

```
def RandomSampleImputation(df, feature):
    # Randomly sample from non-null values
    random_sample = df[feature].dropna().sample(
        df[feature].isnull().sum(), random_state=0, replace=True
    )
    # Align the sampled index to the null index
    random_sample.index = df[df[feature].isnull()].index

    # Fill in the missing values with the sampled values
    df.loc[df[feature].isnull(), feature] = random_sample

RandomSampleImputation(df, "Cloud9am")
RandomSampleImputation(df, "Cloud3pm")
RandomSampleImputation(df, "Evaporation")
RandomSampleImputation(df, "Sunshine")

df.isnull().sum()*100/len(df)
```

```
Date              0.000000
Location          0.000000
MinTemp           1.020899
MaxTemp           0.866905
Rainfall          2.241853
Evaporation       0.000000
Sunshine          0.000000
WindGustDir       7.098859
WindGustSpeed     7.055548
WindDir9am        7.263853
WindDir3pm        2.906641
WindSpeed9am      1.214767
WindSpeed3pm      2.105046
Humidity9am       1.824557
Humidity3pm       3.098446
Pressure9am      10.356799
Pressure3pm      10.331363
Cloud9am          0.000000
Cloud3pm          0.000000
```

```
Temp9am          1.214767
Temp3pm          2.481094
RainToday        2.241853
RainTomorrow     2.245978
dtype: float64
```

```
find_var_type('RainToday')
```

```
RainToday is a Categorical Variable
```

replace the null values of all the continuous feature which are having less number of null values

```
def MeanImputation(df, feature):
    df[feature]= df[feature]
    mean= df[feature].mean()
    df[feature]= df[feature].fillna(mean)
```

```
MeanImputation(df,'Pressure3pm')
```

```
MeanImputation(df, 'Pressure9am')
MeanImputation(df, 'MinTemp')
MeanImputation(df, 'MaxTemp')
MeanImputation(df, 'Rainfall')
MeanImputation(df, 'WindGustSpeed')
MeanImputation(df, 'WindSpeed9am')
MeanImputation(df, 'WindSpeed3pm')
MeanImputation(df, 'Pressure9am')
MeanImputation(df, 'Humidity9am')
MeanImputation(df, 'Humidity3pm')
MeanImputation(df, 'Temp3pm')
MeanImputation(df, 'Temp9am')
```

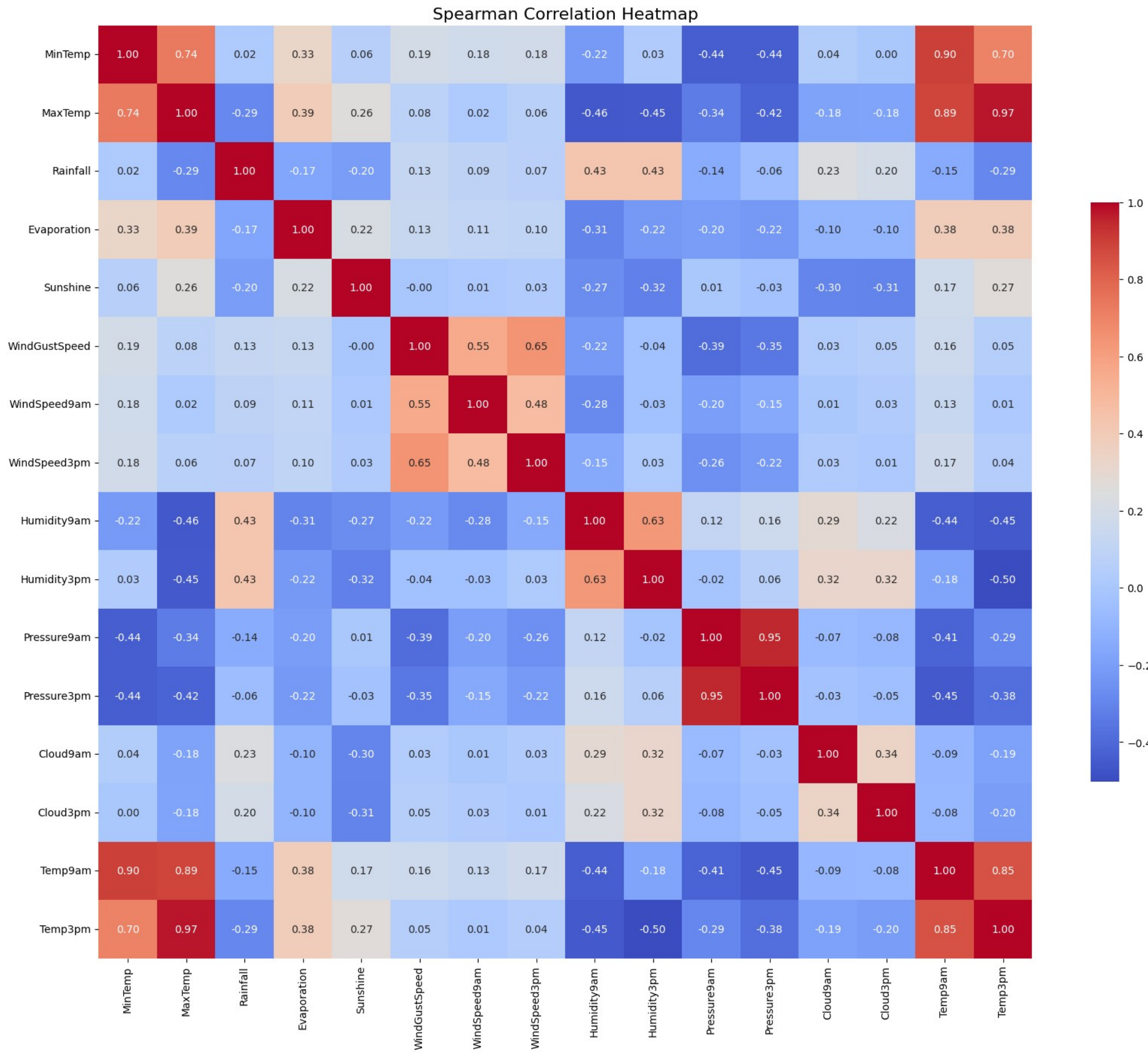
```
df.isnull().sum()*100/len(df)
```

```
Date             0.000000
Location         0.000000
MinTemp          0.000000
MaxTemp          0.000000
Rainfall         0.000000
Evaporation      0.000000
Sunshine         0.000000
WindGustDir      7.098859
WindGustSpeed    0.000000
WindDir9am       7.263853
WindDir3pm       2.906641
WindSpeed9am     0.000000
WindSpeed3pm     0.000000
Humidity9am      0.000000
Humidity3pm      0.000000
```

```
Pressure9am      0.000000
Pressure3pm      0.000000
Cloud9am         0.000000
Cloud3pm         0.000000
Temp9am          0.000000
Temp3pm          0.000000
RainToday        2.241853
RainTomorrow     2.245978
dtype: float64
```

Plotting a HeatMap for the numerical values

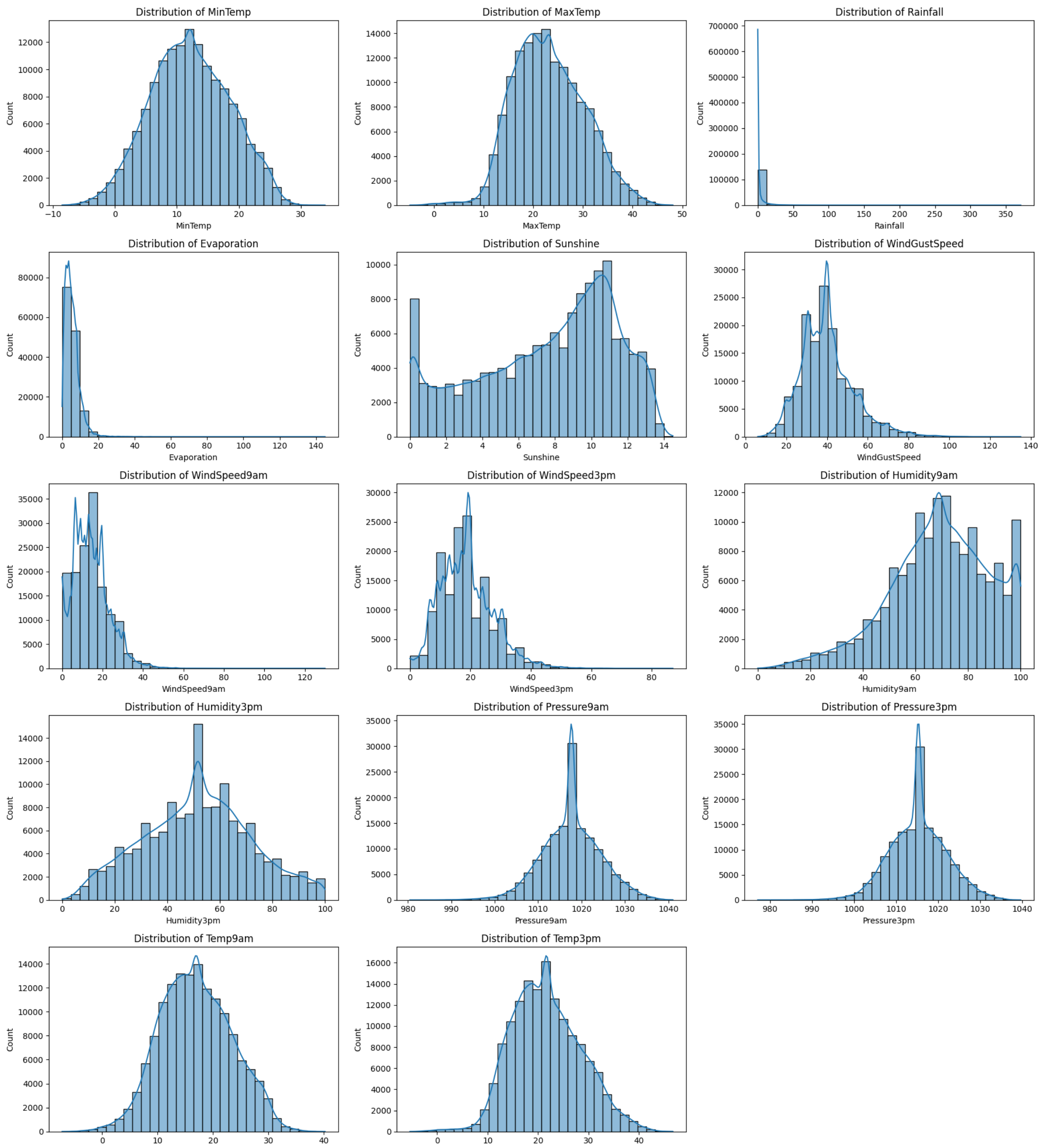
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Select only numeric columns for correlation
numeric_df = df.select_dtypes(include=['number'])

# Compute Spearman correlation matrix
corrmat = numeric_df.corr(method="spearman")

# Set plot size
plt.figure(figsize=(20, 20))

# Plot heatmap
sns.heatmap(corrmat, annot=True, fmt=".2f", cmap="coolwarm",
square=True, cbar_kws={"shrink": 0.5})
plt.title("Spearman Correlation Heatmap", fontsize=16)
plt.show()
```



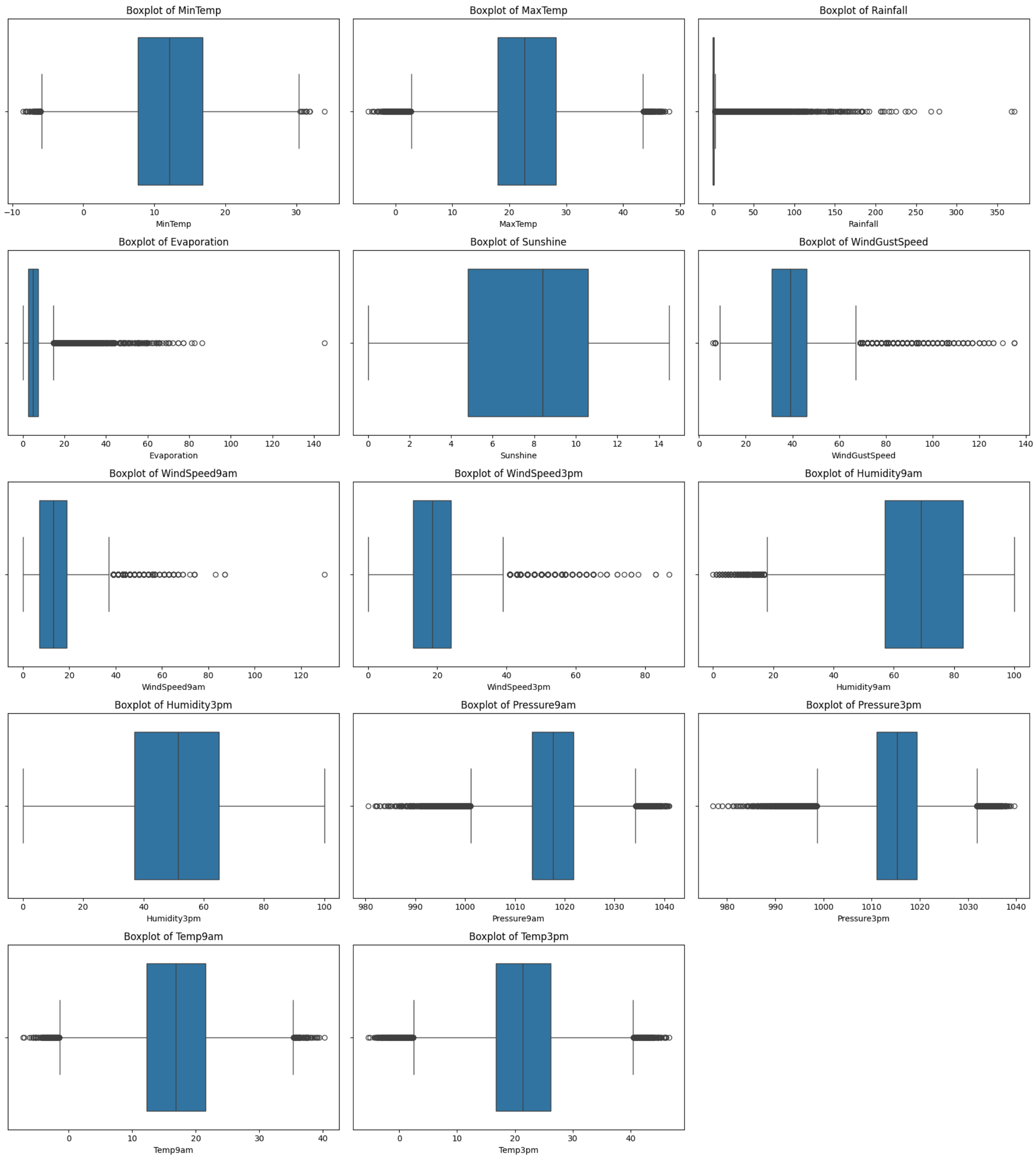
Analysis for Continuous variables

```python
# Number of plots
n = len(cont_var)
cols = 3  # Number of columns in the grid
rows = math.ceil(n / cols)

# Set figure size
plt.figure(figsize=(cols * 6, rows * 4))

for idx, feature in enumerate(cont_var):
    plt.subplot(rows, cols, idx + 1)
    sns.histplot(data=df, x=feature, kde=True, bins=30)
    plt.xlabel(feature)
```

```
    plt.ylabel("Count")
    plt.title(f"Distribution of {feature}")

plt.tight_layout()
plt.show()
```



```python
# Number of plots
n = len(cont_var)
cols = 3  # Number of columns in the grid
rows = math.ceil(n / cols)

# Set figure size
plt.figure(figsize=(cols * 6, rows * 4))

for idx, feature in enumerate(cont_var):
    plt.subplot(rows, cols, idx + 1)
    sns.boxplot(x=df[feature])
    plt.title(f"Boxplot of {feature}")
    plt.xlabel(feature)

plt.tight_layout()
plt.show()
```



One Hot Encoding

```
df["RainToday"] = pd.get_dummies(df["RainToday"], drop_first = True)
df["RainTomorrow"] = pd.get_dummies(df["RainTomorrow"], drop_first =
True)
df
```

{"type":"dataframe","variable_name":"df"}

Lable Encoding

```
for feature in categ_var:
    print(feature, (df.groupby([feature])
["RainTomorrow"].mean().sort_values(ascending = False)).index)
```

```
Date Index(['2007-12-15', '2007-11-01', '2007-11-02', '2007-12-27',
'2008-01-18',
       '2008-02-12', '2008-02-07', '2007-12-24', '2008-01-19', '2008-
01-12',
       ...
       '2009-10-21', '2009-10-22', '2007-12-14', '2007-12-17', '2007-
12-18',
       '2007-12-23', '2007-11-07', '2007-11-05', '2007-11-12', '2007-
11-06'],
      dtype='object', name='Date', length=3436)
Location Index(['Portland', 'Walpole', 'Cairns', 'Dartmoor',
'NorfolkIsland',
       'MountGambier', 'Albany', 'Witchcliffe', 'CoffsHarbour',
'MountGinini',
       'NorahHead', 'Darwin', 'Sydney', 'SydneyAirport', 'Ballarat',
       'GoldCoast', 'Watsonia', 'Newcastle', 'Hobart', 'Wollongong',
       'Williamtown', 'Launceston', 'Brisbane', 'MelbourneAirport',
'Adelaide',
       'Sale', 'Albury', 'Perth', 'Melbourne', 'Nuriootpa', 'Penrith',
       'BadgerysCreek', 'PerthAirport', 'Tuggeranong', 'Richmond',
'Bendigo',
       'Canberra', 'WaggaWagga', 'Townsville', 'Katherine',
'PearceRAAF',
       'SalmonGums', 'Nhil', 'Moree', 'Cobar', 'Mildura',
'AliceSprings',
       'Uluru', 'Woomera'],
      dtype='object', name='Location')
WindGustDir Index(['NNW', 'NW', 'WNW', 'N', 'W', 'WSW', 'NNE', 'S',
'SSW', 'SW', 'SSE',
       'NE', 'SE', 'ESE', 'ENE', 'E'],
      dtype='object', name='WindGustDir')
WindDir9am Index(['NNW', 'N', 'NW', 'NNE', 'WNW', 'W', 'WSW', 'SW',
'SSW', 'NE', 'S',
       'SSE', 'ENE', 'SE', 'ESE', 'E'],
      dtype='object', name='WindDir9am')
WindDir3pm Index(['NW', 'NNW', 'N', 'WNW', 'W', 'NNE', 'WSW', 'SSW',
'S', 'SW', 'SE',
       'NE', 'SSE', 'ENE', 'E', 'ESE'],
      dtype='object', name='WindDir3pm')
RainToday Index([True, False], dtype='bool', name='RainToday')
RainTomorrow Index([True, False], dtype='bool', name='RainTomorrow')
```

```
windgustdir = {'NNW':0, 'NW':1, 'WNW':2, 'N':3, 'W':4, 'WSW':5,
'NNE':6, 'S':7, 'SSW':8, 'SW':9, 'SSE':10,
```

```
       'NE':11, 'SE':12, 'ESE':13, 'ENE':14, 'E':15}
winddir9am = {'NNW':0, 'N':1, 'NW':2, 'NNE':3, 'WNW':4, 'W':5,
'WSW':6, 'SW':7, 'SSW':8, 'NE':9, 'S':10,
       'SSE':11, 'ENE':12, 'SE':13, 'ESE':14, 'E':15}
winddir3pm = {'NW':0, 'NNW':1, 'N':2, 'WNW':3, 'W':4, 'NNE':5,
'WSW':6, 'SSW':7, 'S':8, 'SW':9, 'SE':10,
       'NE':11, 'SSE':12, 'ENE':13, 'E':14, 'ESE':15}
df["WindGustDir"] = df["WindGustDir"].map(windgustdir)
df["WindDir9am"] = df["WindDir9am"].map(winddir9am)
df["WindDir3pm"] = df["WindDir3pm"].map(winddir3pm)

df["WindGustDir"] =
df["WindGustDir"].fillna(df["WindGustDir"].value_counts().index[0])
df["WindDir9am"] =
df["WindDir9am"].fillna(df["WindDir9am"].value_counts().index[0])
df["WindDir3pm"] =
df["WindDir3pm"].fillna(df["WindDir3pm"].value_counts().index[0])

df.isnull().sum()*100/len(df)
```

```
Date             0.0
Location         0.0
MinTemp          0.0
MaxTemp          0.0
Rainfall         0.0
Evaporation      0.0
Sunshine         0.0
WindGustDir      0.0
WindGustSpeed    0.0
WindDir9am       0.0
WindDir3pm       0.0
WindSpeed9am     0.0
WindSpeed3pm     0.0
Humidity9am      0.0
Humidity3pm      0.0
Pressure9am      0.0
Pressure3pm      0.0
Cloud9am         0.0
Cloud3pm         0.0
Temp9am          0.0
Temp3pm          0.0
RainToday        0.0
RainTomorrow     0.0
dtype: float64
```

```
df.head()
```

{"type":"dataframe","variable_name":"df"}

We have removed all the null values and handeled with categorical data

Now we will do the Label Encoding for our Location according to our Target variable

```
df_loc = df.groupby(["Location"])
["RainTomorrow"].value_counts().sort_values().unstack()

df_loc.head()
```


{"summary":"{\n \"name\": \"df_loc\",\n \"rows\": 49,\n \"fields\": [\n   {\n   \"column\": \"Location\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 49,\n   \"samples\": [\n   \"Darwin\",\n   \"Williamtown\",\n   \"Wollongong\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": false,\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 315,\n   \"min\": 1313,\n   \"max\": 2807,\n   \"num_unique_values\": 46,\n   \"samples\": [\n   1462,\n   2196,\n   2417\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": true,\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 219,\n   \"min\": 116,\n   \"max\": 1095,\n   \"num_unique_values\": 49,\n   \"samples\": [\n   852,\n   700,\n   713\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"df_loc"}


```
import pandas as pd

row = df.iloc[1]  # This is a Series
numeric_row = pd.to_numeric(row, errors='coerce')  # Convert values to numbers; non-numeric become NaN
numeric_row = numeric_row.dropna()  # Drop NaNs
numeric_row.sort_values(ascending=False)
```

Pressure9am	1010.6
Pressure3pm	1007.8
WindGustSpeed	44.0
Humidity9am	44.0
MaxTemp	25.1
Humidity3pm	25.0
Temp3pm	24.3
WindSpeed3pm	22.0
Temp9am	17.2
Sunshine	11.9
Cloud3pm	8.0
MinTemp	7.4
WindDir3pm	6.0

```
WindSpeed9am        4.0
Evaporation         2.6
Cloud9am            2.0
WindGustDir         2.0
Rainfall            0.0
WindDir9am          0.0
RainToday           0.0
RainTomorrow        0.0
Name: 1, dtype: float64
```

```python
# Assuming df_loc is grouped by "Location" and then counts of
"RainTomorrow" are calculated
df_loc = df.groupby(["Location"])
["RainTomorrow"].value_counts().sort_values().unstack()

# # Sorting the counts for the specific location (in this case, 1
refers to the second location, change if needed)
# sorted_values = df_loc.iloc[1].sort_values(ascending=False)

# # Display the sorted values
# print(sorted_values)

df_loc.head()
```

{"summary":"{\n \"name\": \"df_loc\",\n \"rows\": 49,\n \"fields\": [\n   {\n   \"column\": \"Location\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 49,\n   \"samples\": [\n   \"Darwin\",\n   \"Williamtown\",\n   \"Wollongong\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": false,\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 315,\n   \"min\": 1313,\n   \"max\": 2807,\n   \"num_unique_values\": 46,\n   \"samples\": [\n   1462,\n   2196,\n   2417\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": true,\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 219,\n   \"min\": 116,\n   \"max\": 1095,\n   \"num_unique_values\": 49,\n   \"samples\": [\n   852,\n   700,\n   713\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"df_loc"}

```python
df_loc[True].sort_values(ascending=False)
```

```
Location
Portland            1095
Cairns               950
Walpole              949
Dartmoor             922
MountGambier         920
NorfolkIsland        919
```

```
Albany              902
Witchcliffe         879
CoffsHarbour        869
Sydney              865
Darwin              852
MountGinini         819
NorahHead           808
Ballarat            781
GoldCoast           775
SydneyAirport       774
Hobart              761
Watsonia            738
Newcastle           731
Wollongong          713
Brisbane            709
Williamtown         700
Launceston          699
Adelaide            688
MelbourneAirport    653
Perth               645
Sale                643
Melbourne           636
Canberra            629
Albury              618
Penrith             595
Nuriootpa           592
BadgerysCreek       583
Tuggeranong         568
PerthAirport        567
Bendigo             562
Richmond            560
WaggaWagga          536
Townsville          519
PearceRAAF          505
SalmonGums          472
Moree               394
Cobar               386
Mildura             327
Katherine           265
AliceSprings        244
Nhil                242
Woomera             202
Uluru               116
Name: True, dtype: int64
```

```
df_loc[True].sort_values(ascending=False).index
```

```
Index(['Portland', 'Cairns', 'Walpole', 'Dartmoor', 'MountGambier',
       'NorfolkIsland', 'Albany', 'Witchcliffe', 'CoffsHarbour',
'Sydney',
```

```
       'Darwin', 'MountGinini', 'NorahHead', 'Ballarat', 'GoldCoast',
       'SydneyAirport', 'Hobart', 'Watsonia', 'Newcastle',
'Wollongong',
       'Brisbane', 'Williamtown', 'Launceston', 'Adelaide',
'MelbourneAirport',
       'Perth', 'Sale', 'Melbourne', 'Canberra', 'Albury', 'Penrith',
       'Nuriootpa', 'BadgerysCreek', 'Tuggeranong', 'PerthAirport',
'Bendigo',
       'Richmond', 'WaggaWagga', 'Townsville', 'PearceRAAF',
'SalmonGums',
       'Moree', 'Cobar', 'Mildura', 'Katherine', 'AliceSprings',
'Nhil',
       'Woomera', 'Uluru'],
      dtype='object', name='Location')
```

```
len(df_loc[True].sort_values(ascending=False))
```

```
49
```

```
mapped_location = {'Portland':1, 'Cairns':2, 'Walpole':3,
'Dartmoor':4, 'MountGambier':5,
       'NorfolkIsland':6, 'Albany':7, 'Witchcliffe':8,
'CoffsHarbour':9, 'Sydney':10,
       'Darwin':11, 'MountGinini':12, 'NorahHead':13, 'Ballarat':14,
'GoldCoast':15,
       'SydneyAirport':16, 'Hobart':17, 'Watsonia':18, 'Newcastle':19,
'Wollongong':20,
       'Brisbane':21, 'Williamtown':22, 'Launceston':23,
'Adelaide':24, 'MelbourneAirport':25,
       'Perth':26, 'Sale':27, 'Melbourne':28, 'Canberra':29,
'Albury':30, 'Penrith':31,
       'Nuriootpa':32, 'BadgerysCreek':33, 'Tuggeranong':34,
'PerthAirport':35, 'Bendigo':36,
       'Richmond':37, 'WaggaWagga':38, 'Townsville':39,
'PearceRAAF':40, 'SalmonGums':41,
       'Moree':42, 'Cobar':43, 'Mildura':44, 'Katherine':45,
'AliceSprings':46, 'Nhil':47,
       'Woomera':48, 'Uluru':49}
df["Location"] = df["Location"].map(mapped_location)
```

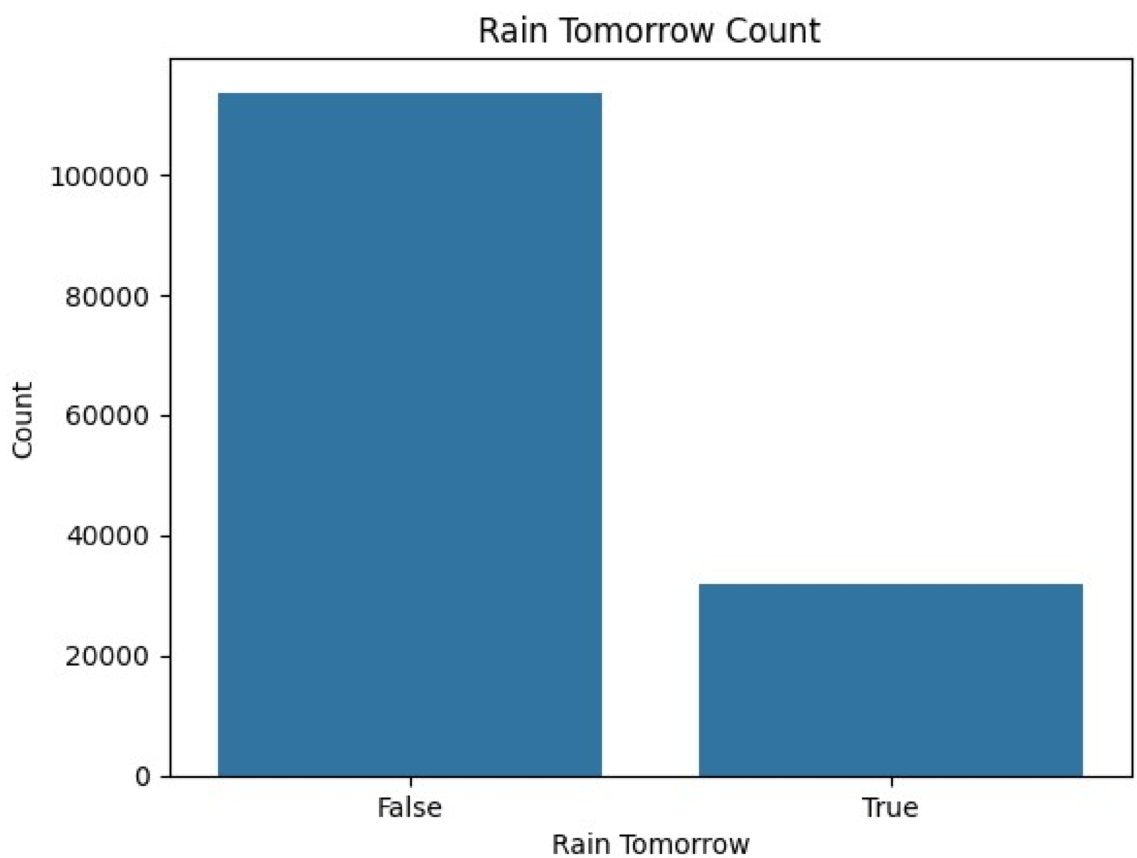
Mapping Data

```
# df["Date"] = pd.to_datetime(df["Date"], format = "%Y-%m-%dT", errors
= "coerce")
# df["Date_month"] = df["Date"].dt.month
# df["Date_day"] = df["Date"].dt.day

df["Date"] = pd.to_datetime(df["Date"], format="%Y-%m-%d",
errors="coerce")
```

```
df["Date_month"] = df["Date"].dt.month
df["Date_day"] = df["Date"].dt.day

df.head()
```

{"type":"dataframe","variable_name":"df"}

```
sns.countplot(x="RainTomorrow", data=df)
plt.title("Rain Tomorrow Count")
plt.xlabel("Rain Tomorrow")
plt.ylabel("Count")
plt.show()
```



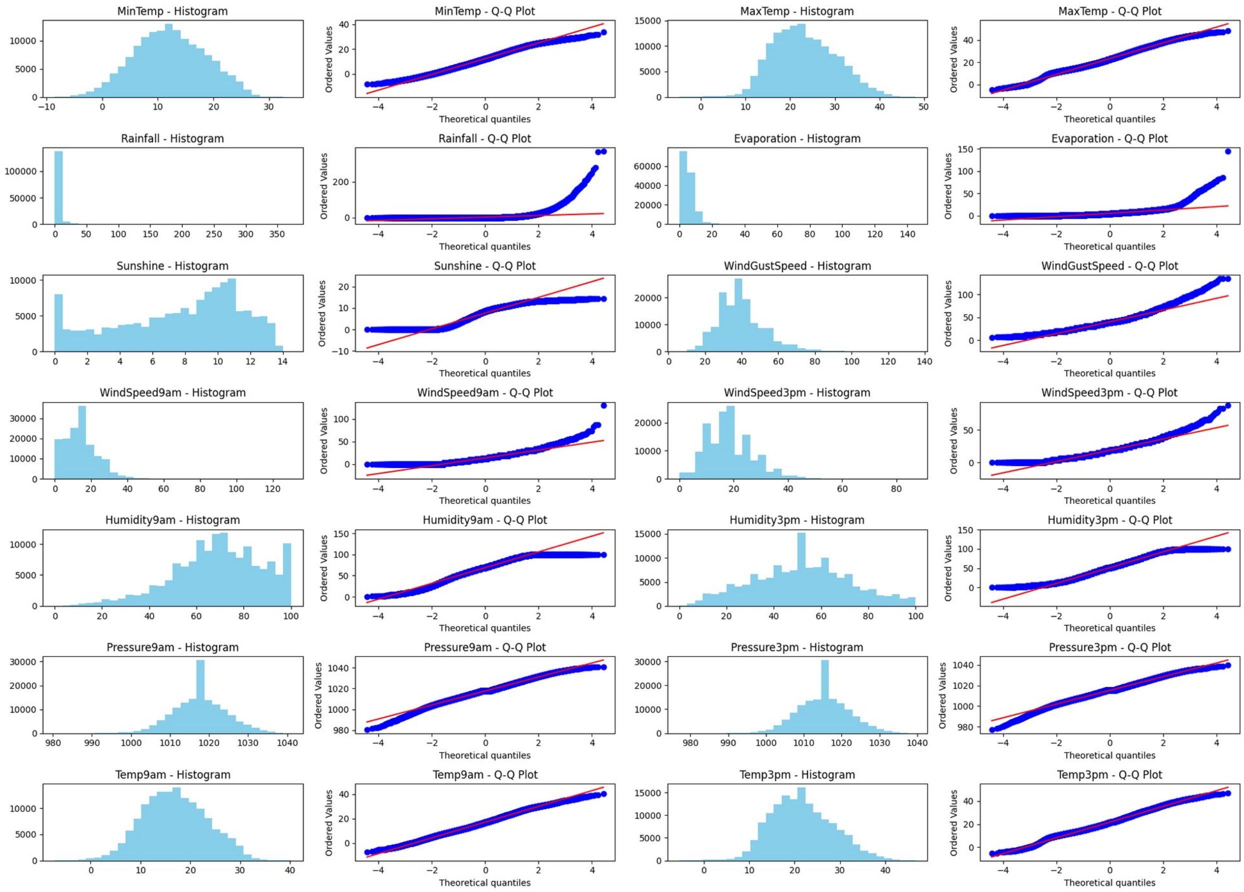

```
df= df.drop(['Date'],axis=1)

df.head()
```

{"type":"dataframe","variable_name":"df"}

Plotting Q-Q Plot

```
import scipy.stats as stats
import pylab
```

```
def plot_curve_grid(df, cont_var, rows=5, cols=4):
    fig, axes = plt.subplots(rows * 2, cols, figsize=(cols * 5, rows *
4))
    axes = axes.flatten()

    for i, feature in enumerate(cont_var):
        # Histogram
        axes[2 * i].hist(df[feature].dropna(), bins=30,
color='skyblue')
        axes[2 * i].set_title(f'{feature} - Histogram')

        # Q-Q plot
        stats.probplot(df[feature].dropna(), dist="norm", plot=axes[2
* i + 1])
        axes[2 * i + 1].set_title(f'{feature} - Q-Q Plot')

    # Hide any unused subplots
    for j in range(2 * len(cont_var), len(axes)):
        axes[j].axis('off')

    plt.tight_layout()
    plt.show()

# Call the function
plot_curve_grid(df, cont_var)
```



Splitting the data

```
x = df.drop(["RainTomorrow"], axis=1)
y = df["RainTomorrow"]

from sklearn.preprocessing import StandardScaler

scale=StandardScaler()

scale.fit(x)

StandardScaler()

X= scale.transform(x)
```

```
x.columns

Index(['Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation',
'Sunshine',
       'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm',
       'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
       'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm',
'Temp9am',
       'Temp3pm', 'RainToday', 'Date_month', 'Date_day'],
      dtype='object')

X=pd.DataFrame(X,columns=x.columns)

X.head()
```

{"type":"dataframe","variable_name":"X"}

```
y.head()

0    False
1    False
2    False
3    False
4    False
Name: RainTomorrow, dtype: bool

X_train, X_test, y_train, y_test = train_test_split(x,y, test_size
=0.2, random_state = 0)
```

Model Building using ML Models.

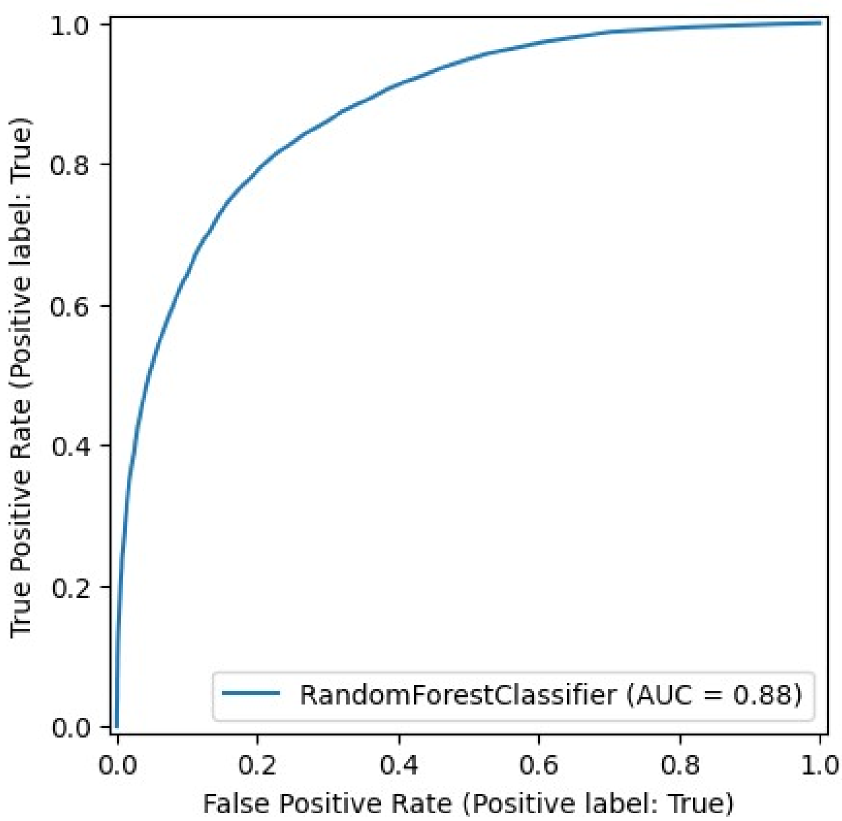
- RandomForestClassifier
- GaussianNB
- KNeighborsClassifier
- XGB Classifier

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

ranfor= RandomForestClassifier()

ranfor.fit(X_train,y_train)

RandomForestClassifier()

ypred= ranfor.predict(X_test)

print(confusion_matrix(y_test,ypred))
print(accuracy_score(y_test,ypred))
print(classification_report(y_test,ypred))
```

```
[[21786   940]
 [ 3300  3066]]
0.8542554654200467
              precision    recall  f1-score   support

       False       0.87      0.96      0.91     22726
        True       0.77      0.48      0.59      6366

    accuracy                           0.85     29092
   macro avg       0.82      0.72      0.75     29092
weighted avg       0.85      0.85      0.84     29092
```

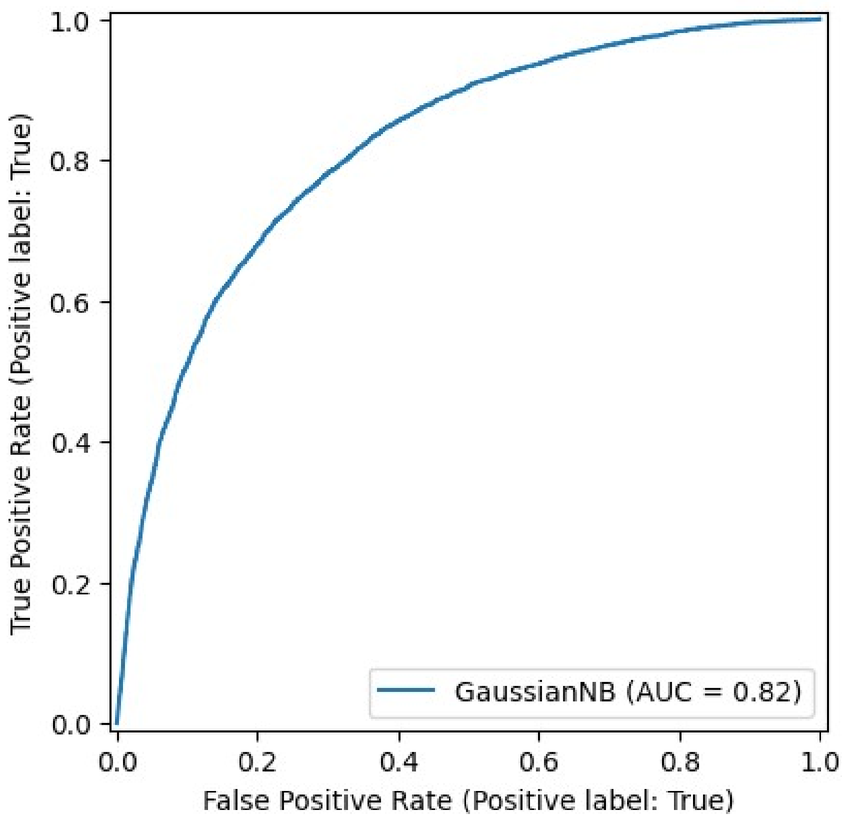
```python
from sklearn.metrics import RocCurveDisplay, roc_auc_score

# Plot ROC Curve
RocCurveDisplay.from_estimator(ranfor, X_test, y_test)

# Show the plot
plt.show()

# Compute AUC Score
print("AUC Score:", roc_auc_score(y_test, ypred))
```



```
AUC Score: 0.7201293979343144
```

Gaussian NB

```python
from sklearn.naive_bayes import GaussianNB

gnb= GaussianNB()

gnb.fit(X_train,y_train)

GaussianNB()

ypred2= gnb.predict(X_test)

print(confusion_matrix(y_test,ypred2))
print(accuracy_score(y_test,ypred2))
print(classification_report(y_test,ypred2))

[[19844  2882]
 [ 2708  3658]]
0.8078509555891654
              precision    recall  f1-score   support

       False       0.88      0.87      0.88     22726
        True       0.56      0.57      0.57      6366

    accuracy                           0.81     29092
   macro avg       0.72      0.72      0.72     29092
weighted avg       0.81      0.81      0.81     29092


from sklearn.metrics import RocCurveDisplay, roc_auc_score

# Plot ROC Curve
RocCurveDisplay.from_estimator(gnb, X_test, y_test)

# Show the plot
plt.show()

# Compute AUC Score
print("AUC Score:", roc_auc_score(y_test, ypred2))
```



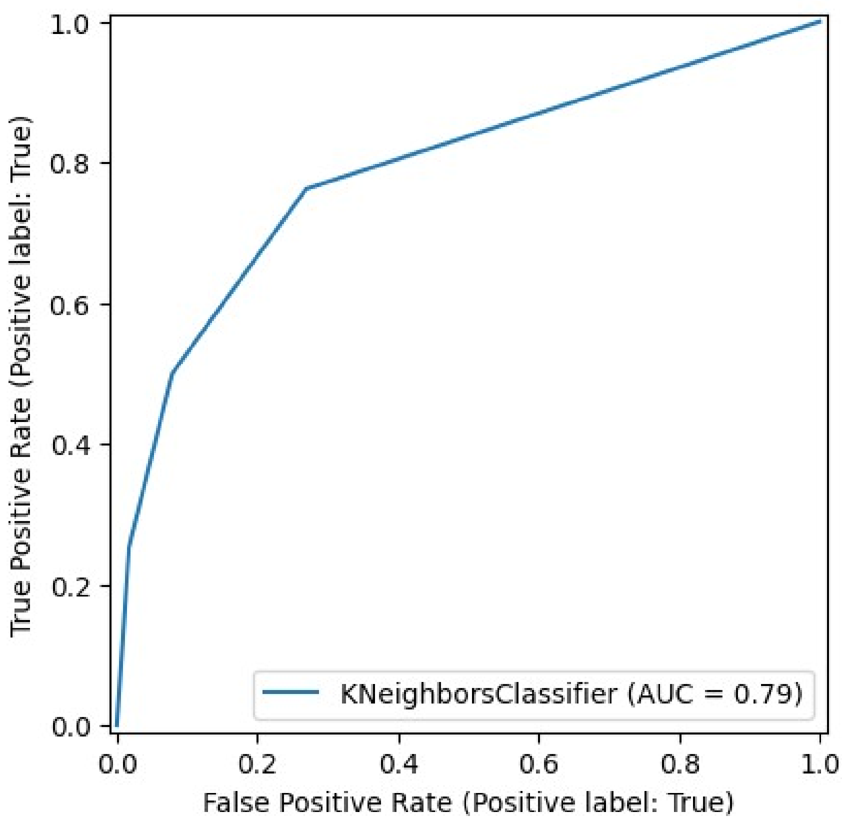

```
AUC Score: 0.7239000206506067
```

K Nearest Neighbors

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train,y_train)

KNeighborsClassifier(n_neighbors=3)

ypred3= knn.predict(X_test)

print(confusion_matrix(y_test,ypred3))
print(accuracy_score(y_test,ypred3))
print(classification_report(y_test,ypred3))

[[20939  1787]
 [ 3186  3180]]
0.8290595352674275
              precision    recall  f1-score   support

       False       0.87      0.92      0.89     22726
        True       0.64      0.50      0.56      6366
```

```
    accuracy                           0.83     29092
   macro avg       0.75      0.71      0.73     29092
weighted avg       0.82      0.83      0.82     29092
```

```python
from sklearn.metrics import RocCurveDisplay, roc_auc_score

# Plot ROC Curve
RocCurveDisplay.from_estimator(knn, X_test, y_test)

# Show the plot
plt.show()

# Compute AUC Score
print("AUC Score:", roc_auc_score(y_test, ypred3))
```




```
AUC Score: 0.7104481715255037
```

XGB Classifier

```python
from xgboost import XGBClassifier

xgb= XGBClassifier()
```

```
xgb.fit(X_train,y_train)
```

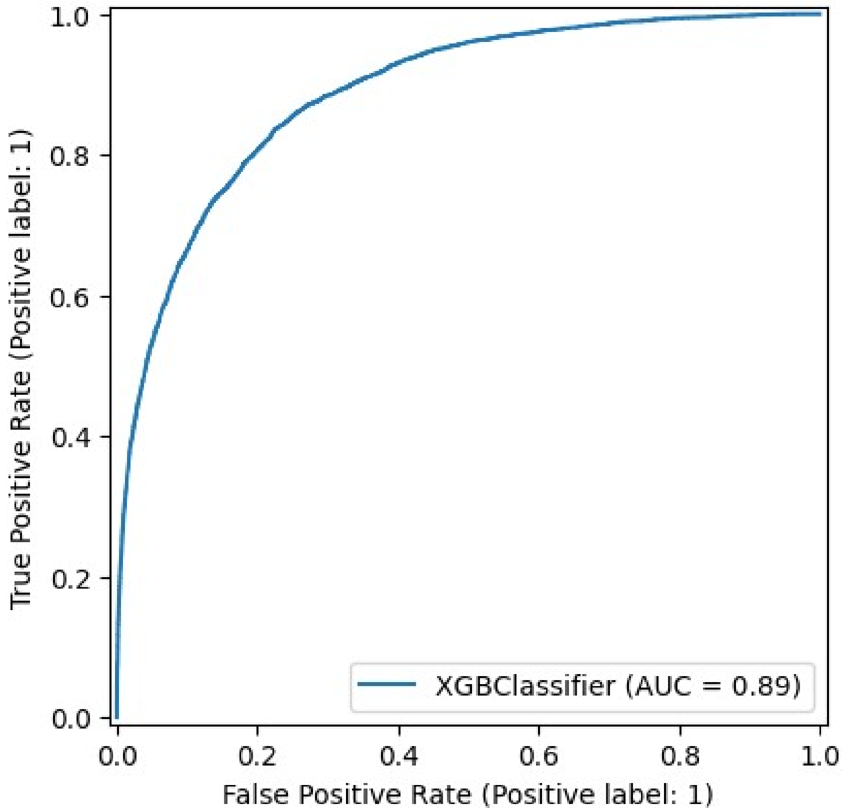
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None,
early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None,
feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None,
max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan,
monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

```
ypred4= xgb.predict(X_test)
```

```
print(confusion_matrix(y_test,ypred4))
print(accuracy_score(y_test,ypred4))
print(classification_report(y_test,ypred4))
```

```
[[21508  1218]
 [ 2896  3470]]
0.8585865530042623
              precision    recall  f1-score   support

       False       0.88      0.95      0.91     22726
        True       0.74      0.55      0.63      6366

    accuracy                           0.86     29092
   macro avg       0.81      0.75      0.77     29092
weighted avg       0.85      0.86      0.85     29092
```

```
from sklearn.metrics import RocCurveDisplay, roc_auc_score

# Plot ROC Curve
RocCurveDisplay.from_estimator(xgb, X_test, y_test)

# Show the plot
plt.show()

# Compute AUC Score
print("AUC Score:", roc_auc_score(y_test, ypred4))
```




```
AUC Score: 0.7457441267355018
```

save the best performing model i.e. XGB Classsifier model in our pickle file

```
import pickle

# Save the model
with open("rain_XGBnew_model.pkl", "wb") as file:
    pickle.dump(xgb, file)

# Load the model
with open("rain_XGBnew_model.pkl", "rb") as file:
    model = pickle.load(file)
```