Rainfall Prediction - 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.

1. Problem Statement:

- 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.

Imporing the librabries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Data Collection

In [2]:
```
df=pd.read_csv('Rainfall_Prediction_Weather.csv')
df.head() #show the first five rows
```

Out[2]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSp
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	4
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	4
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	4
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	2
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	4

5 rows × 23 columns

In [3]:
```
df.tail() #show last five rows from the dataset
```

Out[3]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGus
8420	2017-06-21	Uluru	2.8	23.4	0.0	NaN	NaN	E	
8421	2017-06-22	Uluru	3.6	25.3	0.0	NaN	NaN	NNW	
8422	2017-06-23	Uluru	5.4	26.9	0.0	NaN	NaN	N	
8423	2017-06-24	Uluru	7.8	27.0	0.0	NaN	NaN	SE	
8424	2017-06-25	Uluru	14.9	NaN	0.0	NaN	NaN	NaN	

5 rows × 23 columns

In [4]:
```
#check row and columns
df.shape
```

Out[4]:
```
(8425, 23)
```

In [5]:
```
#checking data types information
df.info()
```

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

In [6]:
```
#checking the missing values
df.isnull().sum()
```

Out[6]:
```
Date               0
Location           0
MinTemp           75
MaxTemp           60
Rainfall         240
Evaporation     3512
Sunshine        3994
WindGustDir      991
WindGustSpeed    991
WindDir9am       829
WindDir3pm       308
WindSpeed9am      76
WindSpeed3pm     107
Humidity9am       59
Humidity3pm      102
Pressure9am     1309
Pressure3pm     1312
Cloud9am        2421
Cloud3pm        2455
Temp9am           56
Temp3pm           96
RainToday        240
RainTomorrow     239
dtype: int64
```

In [8]:
```
#checking the missing values in percentage
```

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

Out[8]:

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

In [10]:

```
#remove the missing values
df = df.drop(["Evaporation","Sunshine","Cloud9am","Cloud3pm","Location", "Date"], axis
df.head()
```

Out[10]:

	MinTemp	MaxTemp	Rainfall	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	Winds
907	19.8	27.1	24.4	ESE	35.0	SW	ESE	
908	18.7	25.6	6.8	E	33.0	SW	S	
909	16.5	25.5	16.8	WSW	19.0	WSW	ESE	
910	18.5	26.9	0.0	ENE	30.0	SW	ENE	
911	18.2	28.2	0.0	NNE	50.0	NNW	NE	

In [11]:

```
df = df.dropna(axis = 0)
df.shape
```

Out[11]:

```
(3790, 17)
```

In [12]:

```
#after the removing missing values
df.isnull().sum()
```

Out[12]:
```
MinTemp          0
MaxTemp          0
Rainfall         0
WindGustDir      0
WindGustSpeed    0
WindDir9am       0
WindDir3pm       0
WindSpeed9am     0
WindSpeed3pm     0
Humidity9am      0
Humidity3pm      0
Pressure9am      0
Pressure3pm      0
Temp9am          0
Temp3pm          0
RainToday        0
RainTomorrow     0
dtype: int64
```

No missing values found

In [13]:
```
df.columns
```

Out[13]:
```
Index(['MinTemp', 'MaxTemp', 'Rainfall', 'WindGustDir', 'WindGustSpeed',
       'WindDir9am', 'WindDir3pm', 'WindSpeed9am', 'WindSpeed3pm',
       'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Temp9am',
       'Temp3pm', 'RainToday', 'RainTomorrow'],
      dtype='object')
```

In [14]:
```
df.describe()
```

Out[14]:

	MinTemp	MaxTemp	Rainfall	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Hu
count	3790.000000	3790.000000	3790.000000	3790.000000	3790.000000	3790.000000	3
mean	14.071029	24.733483	2.451768	41.203958	16.377836	20.099208	
std	5.161176	6.068364	7.772025	13.857829	9.516511	8.719867	
min	-0.700000	10.800000	0.000000	13.000000	2.000000	2.000000	
25%	10.200000	20.300000	0.000000	31.000000	9.000000	13.000000	
50%	14.000000	24.100000	0.000000	39.000000	15.000000	20.000000	
75%	17.900000	29.200000	0.800000	50.000000	22.000000	26.000000	
max	28.500000	43.600000	168.400000	102.000000	61.000000	52.000000	

Label Encoding

In [15]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['WindGustDir'] = le.fit_transform(df['WindGustDir'])
df['WindDir9am'] = le.fit_transform(df['WindDir9am'])
df['WindDir3pm'] = le.fit_transform(df['WindDir3pm'])
df['RainToday'] = le.fit_transform(df['RainToday'])
df['RainTomorrow'] = le.fit_transform(df['RainTomorrow'])
```

Splitting features and Target

In [16]:
```
x = df.drop(['RainTomorrow'], axis = 1)
y = df['RainTomorrow']
```

In [17]:
```
print(x.shape)
print(y.shape)
```

```
(3790, 16)
(3790,)
```

In [18]:
```
x.head()
```

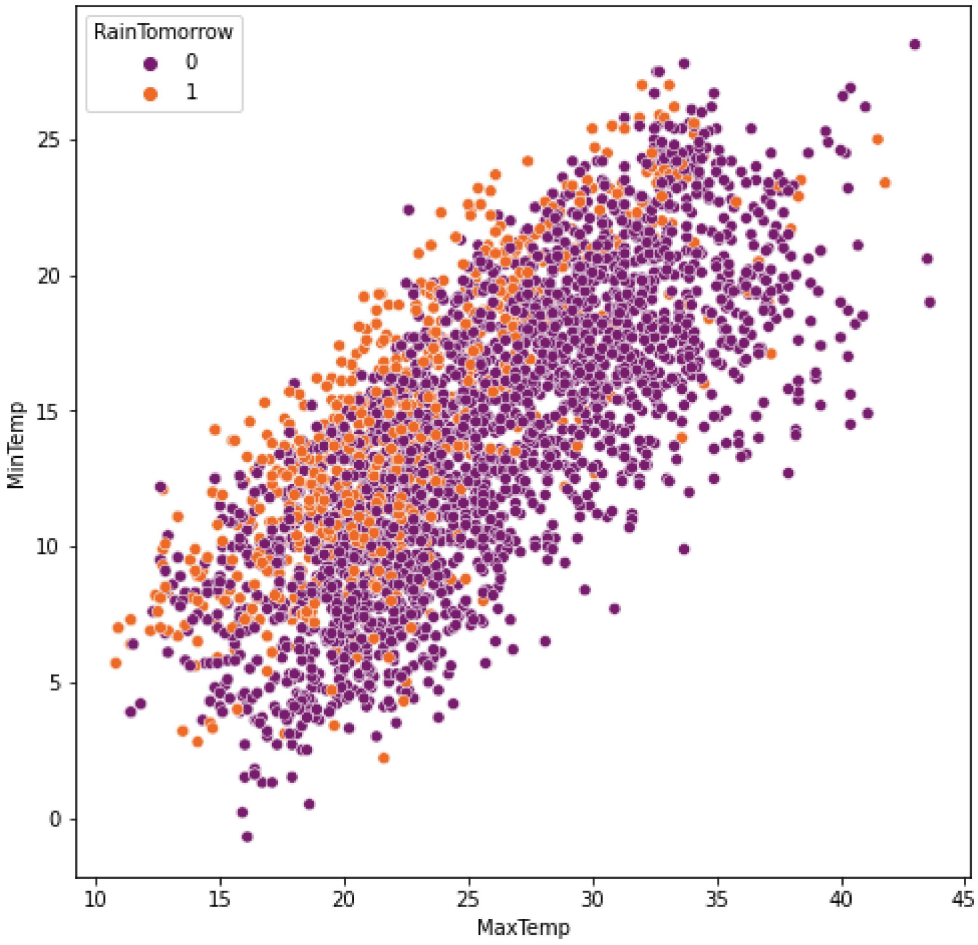
Out[18]:

	MinTemp	MaxTemp	Rainfall	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	Wind
907	19.8	27.1	24.4	2	35.0	12	2	
908	18.7	25.6	6.8	0	33.0	12	8	
909	16.5	25.5	16.8	15	19.0	15	2	
910	18.5	26.9	0.0	1	30.0	12	1	
911	18.2	28.2	0.0	5	50.0	6	4	

Data Visualization

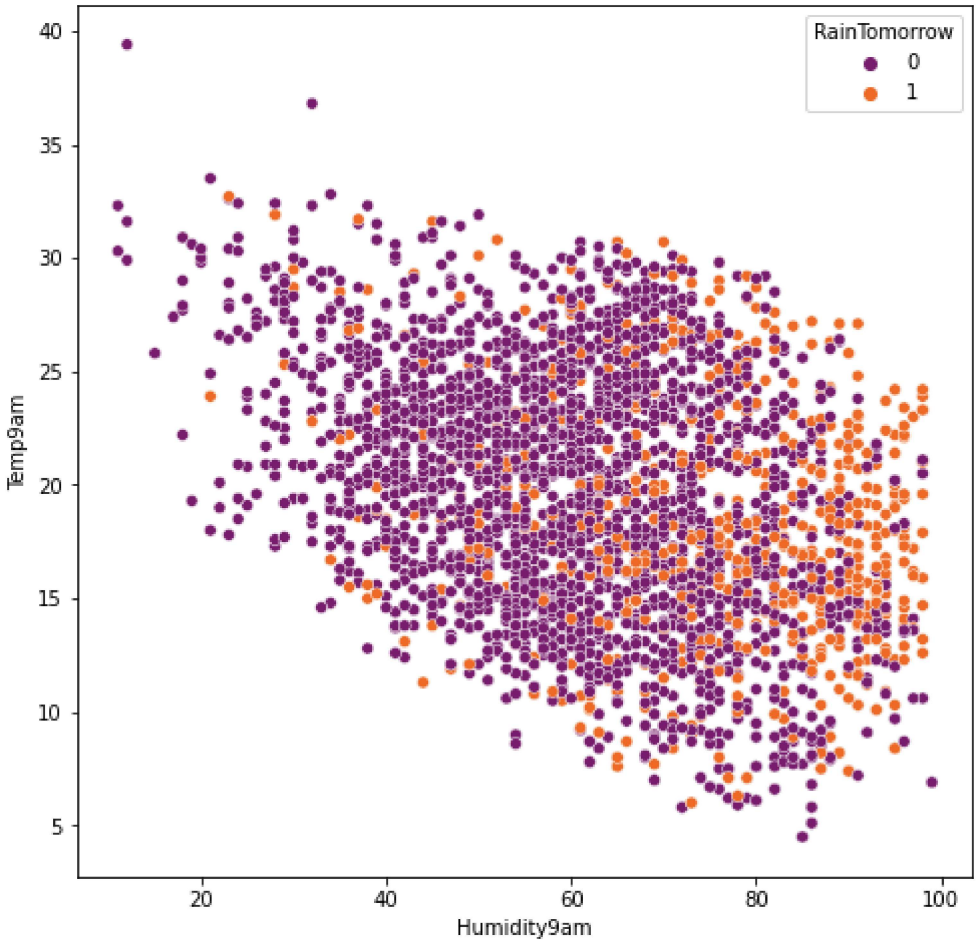
In [19]:
```
plt.figure(figsize = (8,8))
sns.scatterplot(x = 'MaxTemp', y = 'MinTemp', hue = 'RainTomorrow' , palette = 'inferr
```

Out[19]:
```
<AxesSubplot:xlabel='MaxTemp', ylabel='MinTemp'>
```



In [20]:
```
plt.figure(figsize = (8,8))
sns.scatterplot(x = 'Humidity9am', y = 'Temp9am', hue = 'RainTomorrow' , palette = 'ir
```

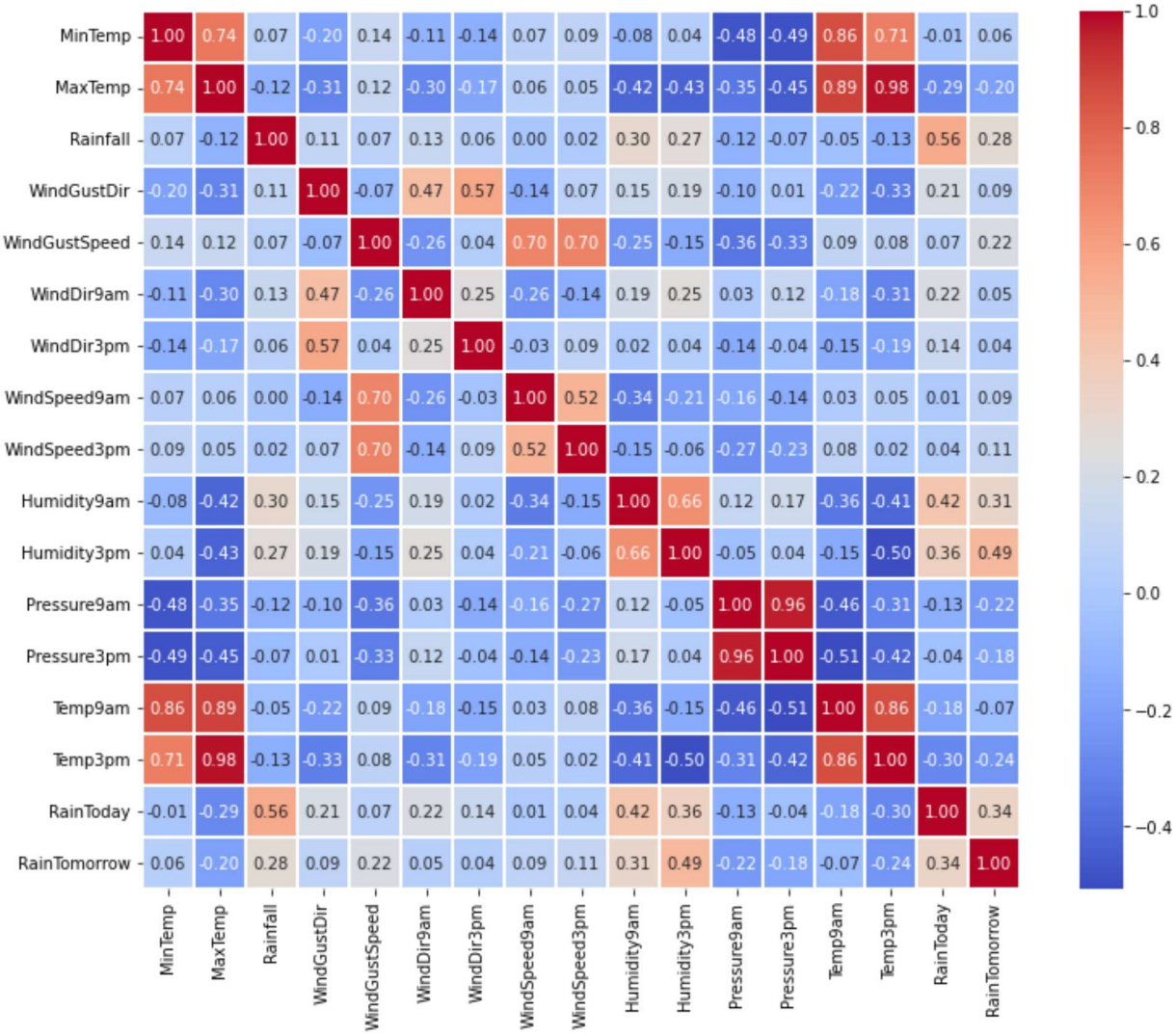
Out[20]:
```
<AxesSubplot:xlabel='Humidity9am', ylabel='Temp9am'>
```



"Correlation Matrix

In [21]:
```
plt.figure(figsize=(13,10))
sns.heatmap(df.corr(),cbar=True,square=True,fmt='.2f',linewidths=0.1,annot=True,annot_
```

Out[21]:
```
<AxesSubplot:>
```



Splitting the data into Training data & Testing Data

In [22]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2)
```

Model Training

In [23]:
```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

Logistic Regression

In [24]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(x_train,y_train)
predictions = lr.predict(x_test)
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
print(accuracy_score(y_test, predictions))
```

```
[[560  26]
 [ 86  86]]
              precision    recall  f1-score   support

           0       0.87      0.96      0.91       586
           1       0.77      0.50      0.61       172

    accuracy                           0.85       758
   macro avg       0.82      0.73      0.76       758
weighted avg       0.84      0.85      0.84       758

0.8522427440633246
```

Decision Tree

In [25]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(x_train,y_train)
predictions = dt.predict(x_test)
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
print(accuracy_score(y_test, predictions))
```

```
[[536  50]
 [ 49 123]]
              precision    recall  f1-score   support

           0       0.92      0.91      0.92       586
           1       0.71      0.72      0.71       172

    accuracy                           0.87       758
   macro avg       0.81      0.81      0.81       758
weighted avg       0.87      0.87      0.87       758

0.8693931398416886
```

Random Forest Classifier

In [26]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(x_train,y_train)
predictions = rf.predict(x_test)
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
print(accuracy_score(y_test, predictions))
```

```
[[572  14]
 [ 51 121]]
              precision    recall  f1-score   support

           0       0.92      0.98      0.95       586
           1       0.90      0.70      0.79       172

    accuracy                           0.91       758
   macro avg       0.91      0.84      0.87       758
weighted avg       0.91      0.91      0.91       758

0.9142480211081794
```

Support Vector Machine

In [27]:
```
from sklearn import svm
from sklearn.svm import SVC
svc=SVC(kernel='linear',random_state=0)
svc.fit(x_train,y_train)
predictions = svc.predict(x_test)
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
print(accuracy_score(y_test, predictions))
```

```
[[560  26]
 [ 86  86]]
              precision    recall  f1-score   support

           0       0.87      0.96      0.91       586
           1       0.77      0.50      0.61       172

    accuracy                           0.85       758
   macro avg       0.82      0.73      0.76       758
weighted avg       0.84      0.85      0.84       758

0.8522427440633246
```

Model Saving

In [28]:
```
import pickle
filename='Rainfall_Prediction_Weather.pkl'
pickle.dump(df, open(filename,'wb'))
```

In [29]:
```
a=np.array(y_test)
predicted=np.array(rf.predict(x_test))
df.com=pd.DataFrame({"original":a,"predicted":predicted},index=range(len(a)))
df.com
```

Out[29]:

	original	predicted
0	1	1
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
753	0	0
754	0	0
755	0	0
756	0	0
757	0	0

758 rows × 2 columns

Random Forest Classifier is best suitable model to give best accuracy

In [ ]:

In [ ]: