Project :Rainfall Prediction - Weather Forecasting

In [1]:
```
#imoprting pandas for loading the dataset and converting into dataframe by creating the va
#importing the numpy
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# loading the dataset to variable df
df=pd.read_csv('rainfalldata')
df
```

Out[2]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindD
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	
...	...	...	...	...	...	...	...	...	...	
8420	2017-06-21	Uluru	2.8	23.4	0.0	NaN	NaN	E	31.0	
8421	2017-06-22	Uluru	3.6	25.3	0.0	NaN	NaN	NNW	22.0	
8422	2017-06-23	Uluru	5.4	26.9	0.0	NaN	NaN	N	37.0	
8423	2017-06-24	Uluru	7.8	27.0	0.0	NaN	NaN	SE	28.0	
8424	2017-06-25	Uluru	14.9	NaN	0.0	NaN	NaN	NaN	NaN	

8425 rows × 23 columns

In [3]:
```
#checking for null values
df.isnull().sum()
```

Out[3]:
```
Date              0
Location          0
MinTemp          75
MaxTemp          60
Rainfall        240
Evaporation    3512
Sunshine       3994
WindGustDir     991
WindGustSpeed   991
WindDir9am      829
               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
```

By looking the above table, we can say that null values present in tha df dataset, we need to remove all the null values

In [4]:
```
# checking for dtypes in the df dataset
df.dtypes
```

Out[4]:
```
Date              object
Location          object
MinTemp          float64
MaxTemp          float64
Rainfall         float64
Evaporation      float64
Sunshine         float64
WindGustDir       object
WindGustSpeed    float64
WindDir9am        object
WindDir3pm        object
WindSpeed9am     float64
WindSpeed3pm     float64
Humidity9am      float64
Humidity3pm      float64
Pressure9am      float64
Pressure3pm      float64
Cloud9am         float64
Cloud3pm         float64
Temp9am          float64
Temp3pm          float64
RainToday         object
RainTomorrow      object
dtype: object
```

df dataset contains two dtypes, float64 and object

In [5]:
```
# checking for the columns name
df.columns
```

Out[5]:
```
Index(['Date', 'Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation',
       'Sunshine', 'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm',
       'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
       'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am',
       'Temp3pm', 'RainToday', 'RainTomorrow'],
      dtype='object')
```

In [6]:
```
# classifying the df dataset depanding on there dtype values into numerical_columns and ca
numerical_columns = [i for i in df.columns if df[i].dtypes != 'O']
discrete_columns=[i for i in numerical_columns if len(df[i].unique())<25]
continuous_columns = [i for i in numerical_columns if i not in discrete_columns]
categorical_columns = [i for i in df.columns if i not in numerical_columns]
print("cal columns Count {}".format(len(numerical_columns)))
```

```
print("Discrete columns Count {}".format(len(discrete_columns)))
print("Continuous columns Count {}".format(len(continuous_columns)))
print("Categorical columns Count {}".format(len(categorical_columns)))
```

```
Numerical columns Count 16
Discrete columns Count 2
Continuous columns Count 14
Categorical columns Count 7
```

In [7]:

```
#filling the null values
def randomsampleimputation(df, variable):
    df[variable]=df[variable]
    random_sample=df[variable].dropna().sample(df[variable].isnull().sum(),random_state=0)
    random_sample.index=df[df[variable].isnull()].index
    df.loc[df[variable].isnull(),variable]=random_sample
```

In [8]:

```
randomsampleimputation(df, "Cloud9am")
randomsampleimputation(df, "Cloud3pm")
randomsampleimputation(df, "Evaporation")
randomsampleimputation(df, "Sunshine")
```

In [9]:

```
# The columns like Cloud9am,Cloud3pm,Evaparation,Sunshine.Null values filled
df
```

Out[9]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindD
0	2008-12-01	Albury	13.4	22.9	0.6	3.0	13.8	W	44.0	
1	2008-12-02	Albury	7.4	25.1	0.0	2.2	7.6	WNW	44.0	
2	2008-12-03	Albury	12.9	25.7	0.0	4.6	7.7	WSW	46.0	
3	2008-12-04	Albury	9.2	28.0	0.0	1.8	8.0	NE	24.0	
4	2008-12-05	Albury	17.5	32.3	1.0	3.8	11.9	W	41.0	
...	...	...	...	...	...	...	...	...	...	...
8420	2017-06-21	Uluru	2.8	23.4	0.0	6.4	1.2	E	31.0	
8421	2017-06-22	Uluru	3.6	25.3	0.0	12.6	7.1	NNW	22.0	
8422	2017-06-23	Uluru	5.4	26.9	0.0	4.2	13.0	N	37.0	
8423	2017-06-24	Uluru	7.8	27.0	0.0	4.0	13.1	SE	28.0	
8424	2017-06-25	Uluru	14.9	NaN	0.0	8.4	4.4	NaN	NaN	

8425 rows × 23 columns

In [10]:

```
continuous_columns
```

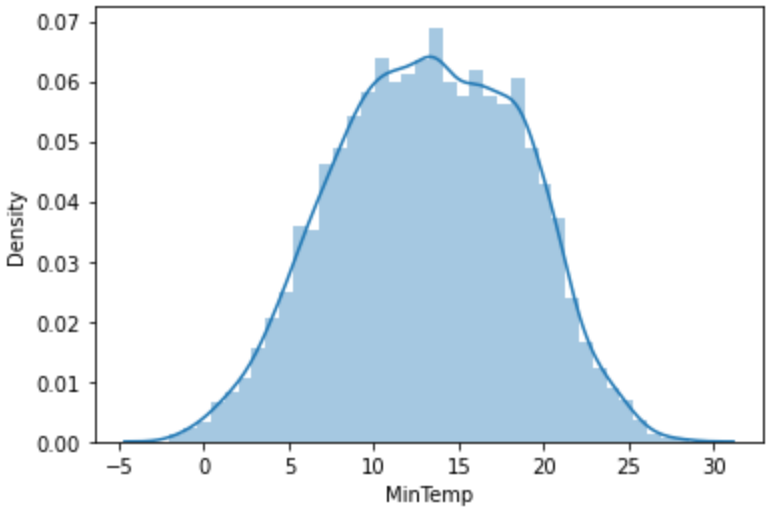
Out[10]:

```
['MinTemp',
 'MaxTemp',
```

```
'Rainfall',
'Evaporation',
'Sunshine',
'WindGustSpeed',
'WindSpeed9am',
'WindSpeed3pm',
'Humidity9am',
'Humidity3pm',
'Pressure9am',
'Pressure3pm',
'Temp9am',
'Temp3pm']
```

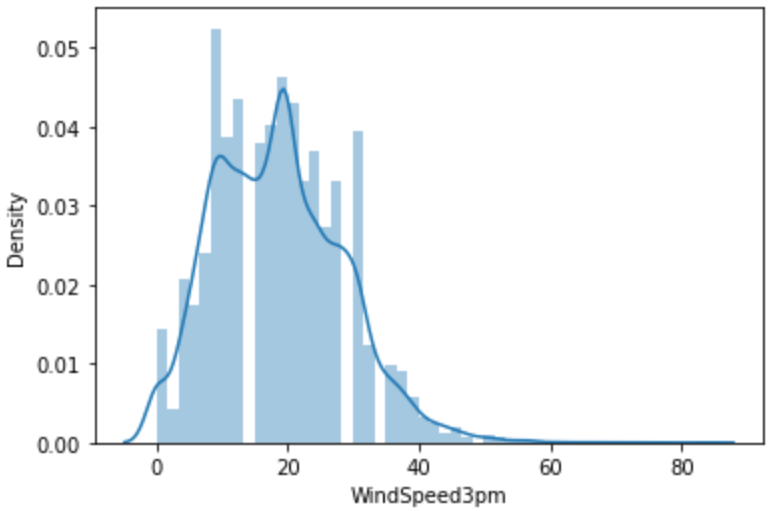
In [11]:
```
import seaborn as sns
sns.distplot(df['MinTemp'],kde=True)
```

Out[11]:
```
<AxesSubplot:xlabel='MinTemp', ylabel='Density'>
```



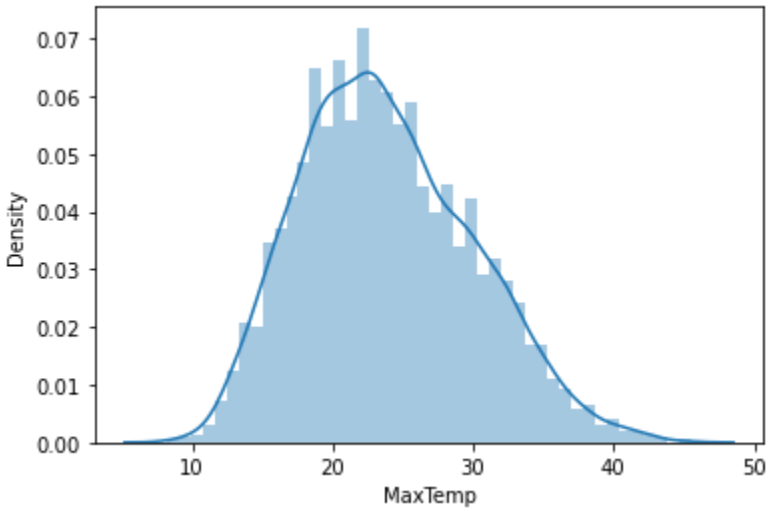
In [12]:
```
sns.distplot(df['WindSpeed3pm'],kde=True)
```

Out[12]:
```
<AxesSubplot:xlabel='WindSpeed3pm', ylabel='Density'>
```



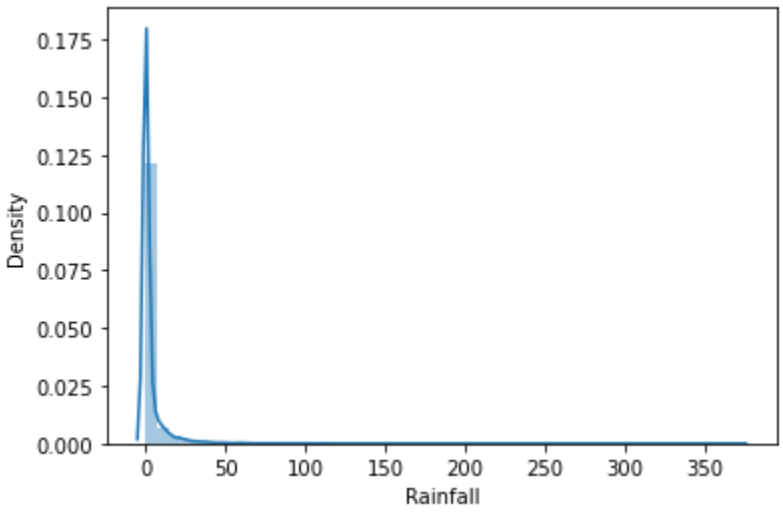
In [13]:
```
sns.distplot(df['MaxTemp'],kde=True)
```

Out[13]:
```
<AxesSubplot:xlabel='MaxTemp', ylabel='Density'>
```



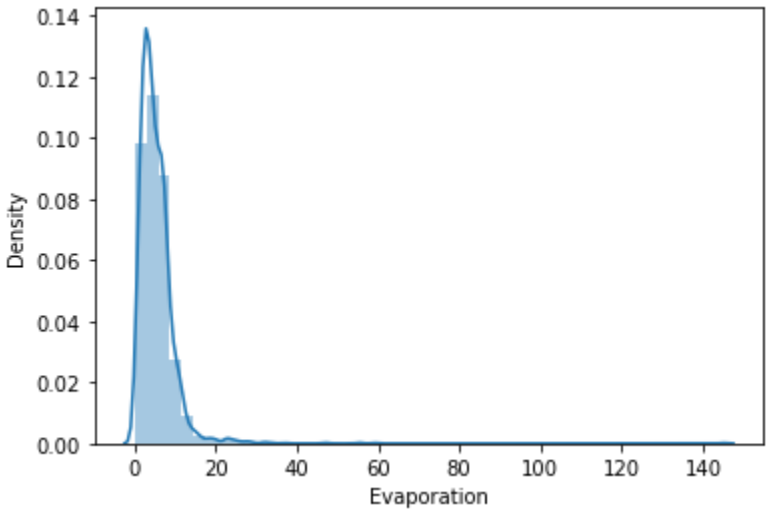
In [14]:
```
sns.distplot(df['Rainfall'],kde=True)
```

Out[14]:
```
<AxesSubplot:xlabel='Rainfall', ylabel='Density'>
```



In [15]:
```
sns.distplot(df['Evaporation'],kde=True)
```

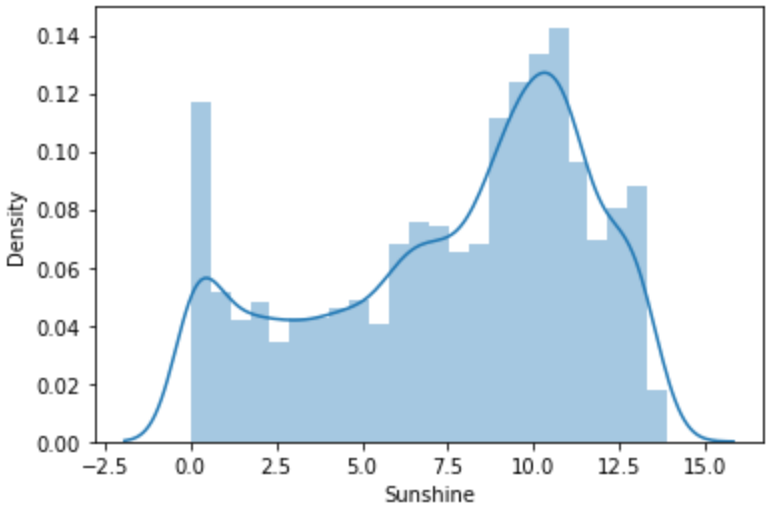
Out[15]:
```
<AxesSubplot:xlabel='Evaporation', ylabel='Density'>
```



In [16]:
```
sns.distplot(df['Sunshine'],kde=True)
```

```
label='Sunshine', ylabel='Density'>
```

Out[16]:

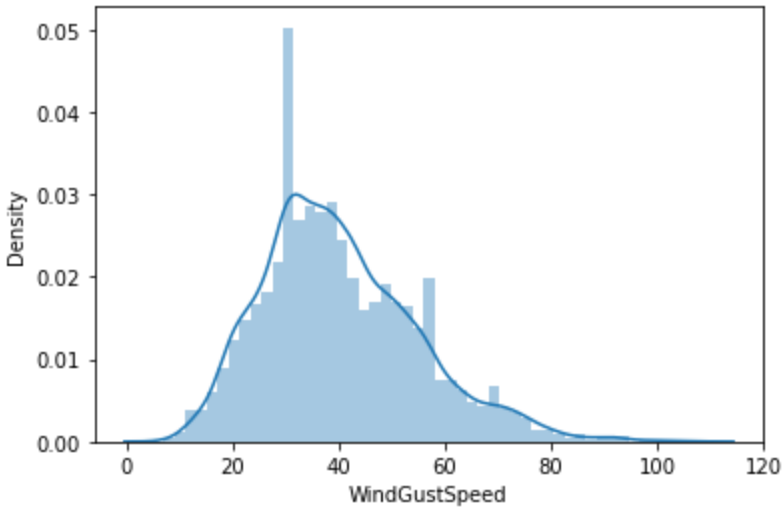


In [17]:

```
sns.distplot(df['WindGustSpeed'],kde=True)
```

Out[17]:

```
<AxesSubplot:xlabel='WindGustSpeed', ylabel='Density'>
```

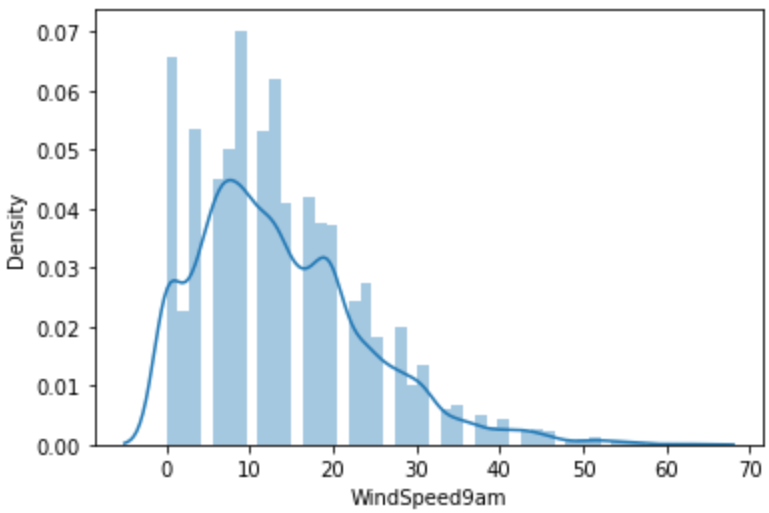


In [18]:

```
sns.distplot(df['WindSpeed9am'],kde=True)
```

Out[18]:

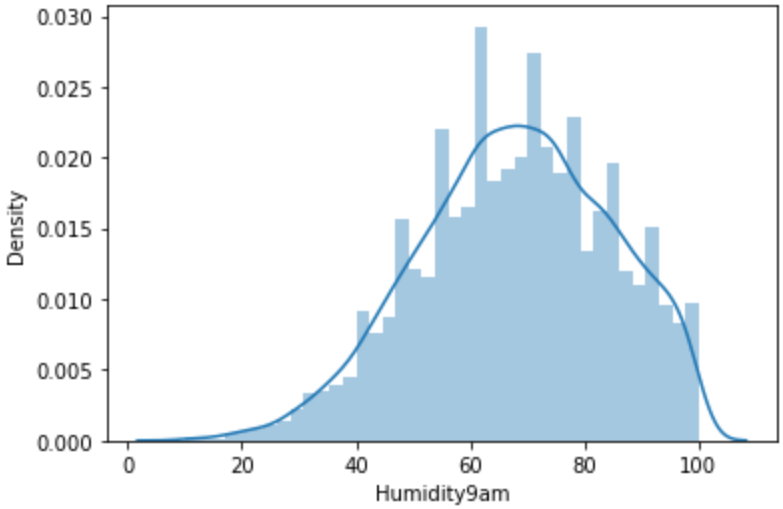
```
<AxesSubplot:xlabel='WindSpeed9am', ylabel='Density'>
```



In [19]:

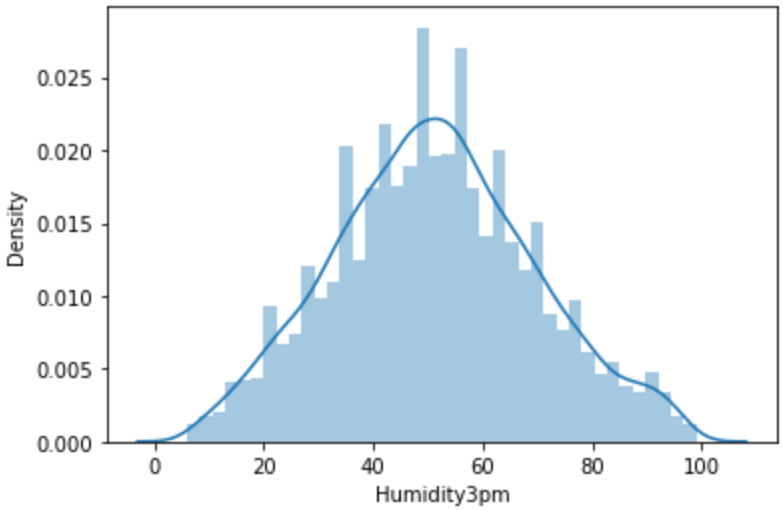
```
sns.distplot(df['Humidity9am'],kde=True)
```

Out[19]: <AxesSubplot:xlabel='Humidity9am', ylabel='Density'>



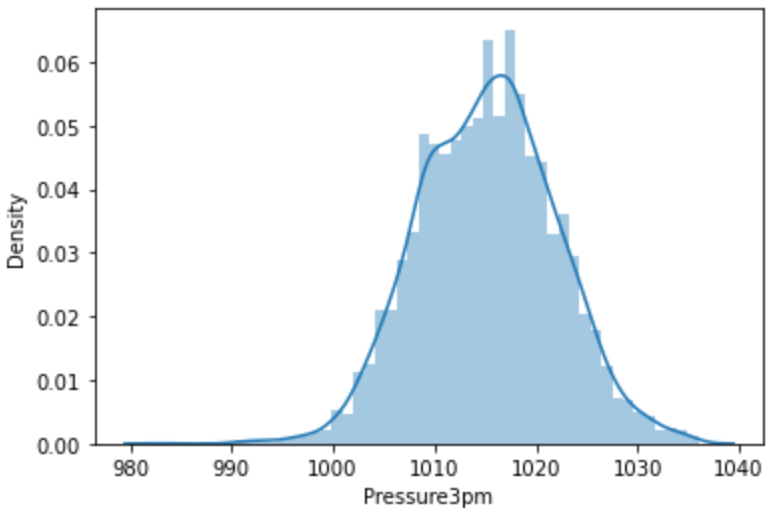
In [20]:
```
sns.distplot(df['Humidity3pm'],kde=True)
```

Out[20]: <AxesSubplot:xlabel='Humidity3pm', ylabel='Density'>



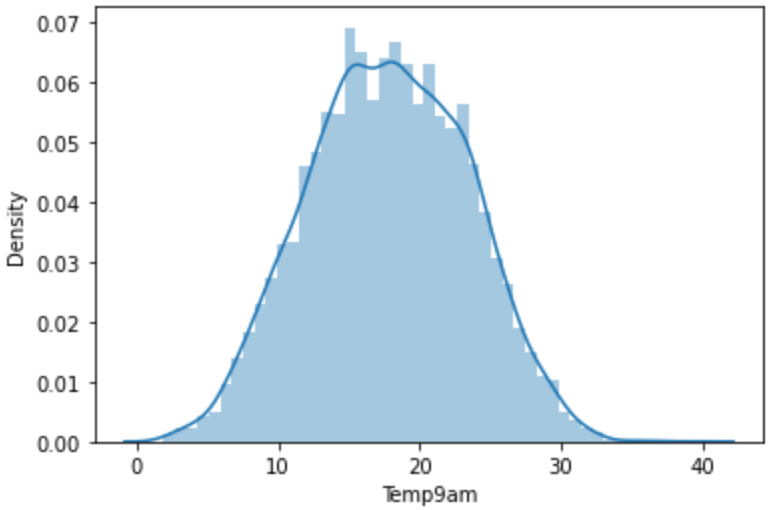
In [21]:
```
sns.distplot(df['Pressure3pm'],kde=True)
```

Out[21]: <AxesSubplot:xlabel='Pressure3pm', ylabel='Density'>



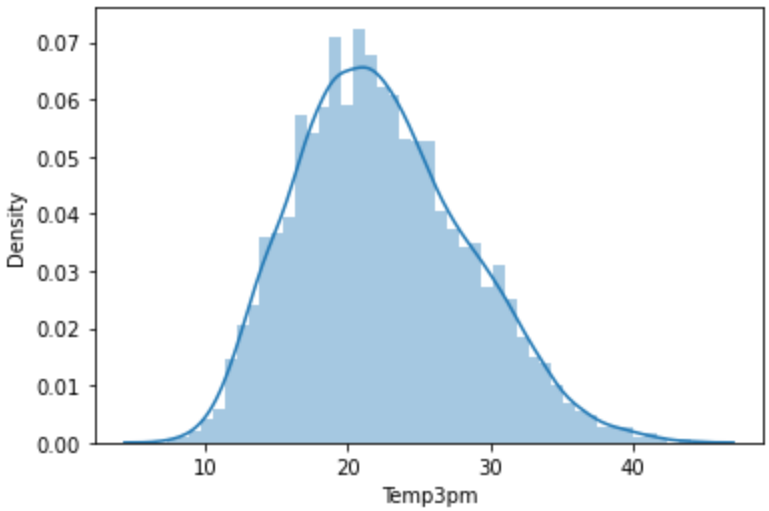
In [22]:
```
sns.distplot(df['Temp9am'],kde=True)
```

Out[22]:
```
<AxesSubplot:xlabel='Temp9am', ylabel='Density'>
```



In [23]:
```
sns.distplot(df['Temp3pm'],kde=True)
```

Out[23]:
```
<AxesSubplot:xlabel='Temp3pm', ylabel='Density'>
```



In [24]:
```
# null values are replaced by median values
for b in continuous_columns:
    if(df[b].isnull().sum()*100/len(df))>0:
        df[b] = df[b].fillna(df[b].median())
```

In [25]:
```
# again checking for null values
df.isnull().sum()
```

Out[25]:
```
Date              0
Location          0
MinTemp           0
MaxTemp           0
Rainfall          0
Evaporation       0
Sunshine          0
WindGustDir     991
WindGustSpeed     0
WindDir9am      829
WindDir3pm      308
WindSpeed9am      0
                  0
```

```
Humidity9am        0
Humidity3pm        0
Pressure9am        0
Pressure3pm        0
Cloud9am           0
Cloud3pm           0
Temp9am            0
Temp3pm            0
RainToday        240
RainTomorrow     239
dtype: int64
```

By seeing the above table, we can say that all the numerical_columns null values are fillied,but categorical_columns null values need to filled, so now we need to work on categorical columns to fill the null values

In [28]:
```
categorical_columns
```

Out[28]:
```
['Date',
 'Location',
 'WindGustDir',
 'WindDir9am',
 'WindDir3pm',
 'RainToday',
 'RainTomorrow']
```

In [29]:
```
# convering the categorical columns to numerical columns
windgustdir = {'NNW':0, 'NW':1, 'WNW':2, 'N':3, 'W':4, 'WSW':5, 'NNE':6, 'S':7, 'SSW':8, '
       '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, '
       '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, 'S
       '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)
```

In [30]:
```
# filling the null values
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])
```

In [31]:
```
# converting the categorical columns to numerical columns
df["RainToday"] = pd.get_dummies(df["RainToday"], drop_first = True)
df["RainTomorrow"] = pd.get_dummies(df["RainTomorrow"], drop_first = True)
df
```

Out[31]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindD
0	2008-12-01	Albury	13.4	22.9	0.6	3.0	13.8	4.0	44.0	
1	2008-12-02	Albury	7.4	25.1	0.0	2.2	7.6	2.0	44.0	
2	2008-12-03	Albury	12.9	25.7	0.0	4.6	7.7	5.0	46.0	
3	2008-12-04	Albury	9.2	28.0	0.0	1.8	8.0	11.0	24.0	

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindD
4	2008-12-05	Albury	17.5	32.3	1.0	3.8	11.9	4.0	41.0	
...	...	...	...	...	...	...	...	...	...	
8420	2017-06-21	Uluru	2.8	23.4	0.0	6.4	1.2	15.0	31.0	
8421	2017-06-22	Uluru	3.6	25.3	0.0	12.6	7.1	0.0	22.0	
8422	2017-06-23	Uluru	5.4	26.9	0.0	4.2	13.0	3.0	37.0	
8423	2017-06-24	Uluru	7.8	27.0	0.0	4.0	13.1	12.0	28.0	
8424	2017-06-25	Uluru	14.9	23.3	0.0	8.4	4.4	3.0	39.0	

8425 rows × 23 columns

In [32]:
```
df1 = df.groupby(["Location"])["RainTomorrow"].value_counts().sort_values().unstack()
df1
```

Out[32]:

RainTomorrow	0	1
Location		
Adelaide	160.0	45.0
Albury	708.0	199.0
Brisbane	444.0	135.0
CoffsHarbour	425.0	186.0
Darwin	218.0	32.0
Melbourne	1216.0	406.0
Newcastle	624.0	198.0
Penrith	366.0	116.0
PerthAirport	962.0	242.0
Uluru	39.0	NaN
Williamtown	924.0	306.0
Wollongong	348.0	126.0

In [33]:
```
df1[1].sort_values(ascending = False)
```

Out[33]:
```
Location
Melbourne       406.0
Williamtown     306.0
PerthAirport    242.0
Albury          199.0
Newcastle       198.0
CoffsHarbour    186.0
Brisbane        135.0
Wollongong      126.0
Penrith         116.0
Adelaide         45.0
                 32.0
```

```
Uluru              NaN
Name: 1, dtype: float64
```

In [34]:
```
print(df1[1].sort_values(ascending = False).index)
print(len(df1[1].sort_values(ascending = False).index))
```

```
Index(['Melbourne', 'Williamtown', 'PerthAirport', 'Albury', 'Newcastle',
       'CoffsHarbour', 'Brisbane', 'Wollongong', 'Penrith', 'Adelaide',
       'Darwin', 'Uluru'],
      dtype='object', name='Location')
12
```

In [35]:
```
#converting the categorical columns into numerical columns
location={'Melbourne':1, 'Williamtown':2, 'PerthAirport':3, 'Albury':4, 'Newcastle':5,
       'CoffsHarbour':6, 'Brisbane':7, 'Wollongong':8, 'Penrith':9, 'Adelaide':10,
       'Darwin':11, 'Uluru':12}
df['Location']=df['Location'].map(location)
```

In [36]:
```
# two new columns are added, Month and Day
df["Date"] = pd.to_datetime(df["Date"], format = "%Y-%m-%dT", errors = "coerce")
```

In [37]:
```
df["Month"] = df["Date"].dt.month
df["Day"] = df["Date"].dt.day
```

In [38]:
```
df
```

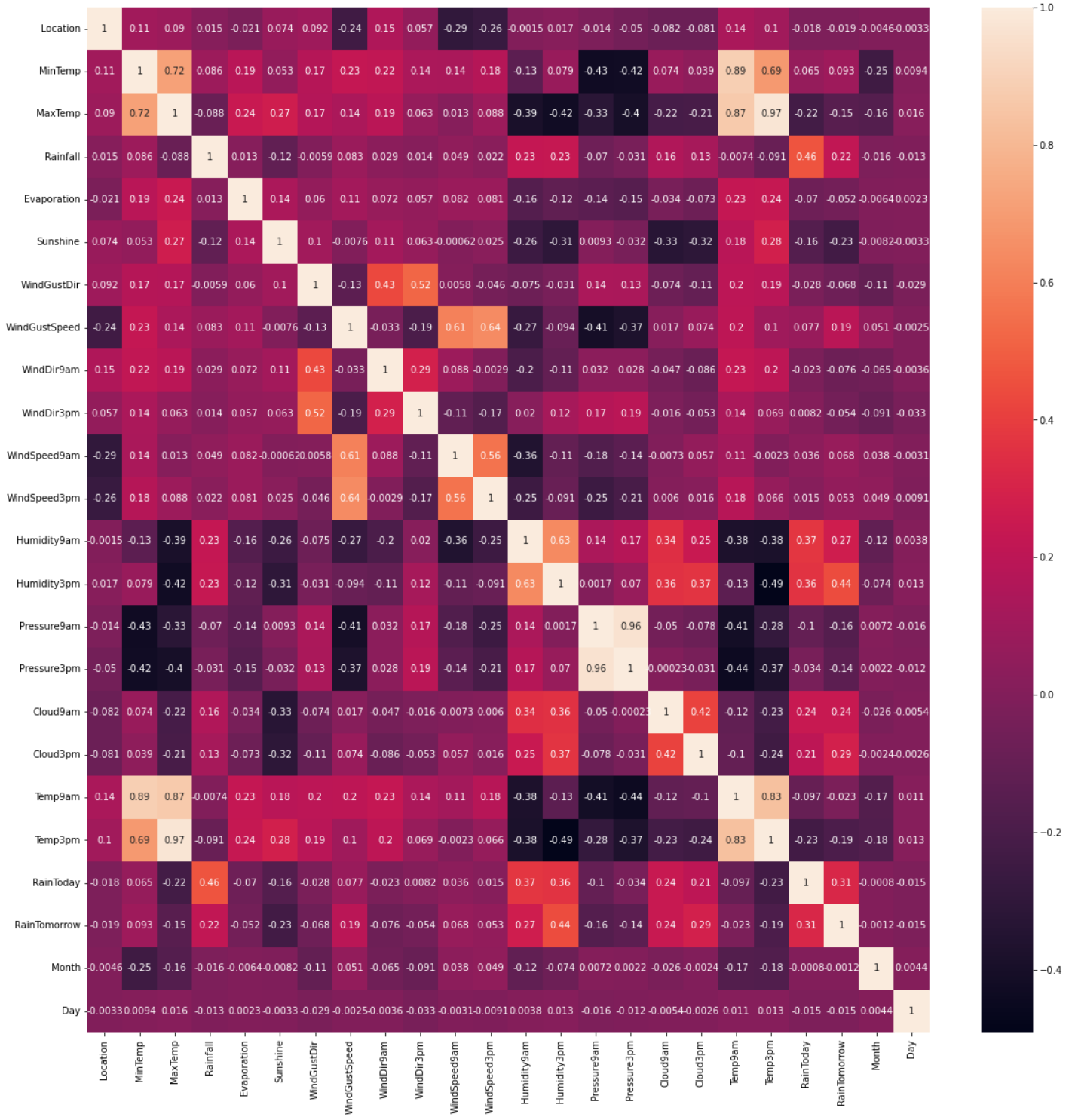
Out[38]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindD
0	2008-12-01	4	13.4	22.9	0.6	3.0	13.8	4.0	44.0	
1	2008-12-02	4	7.4	25.1	0.0	2.2	7.6	2.0	44.0	
2	2008-12-03	4	12.9	25.7	0.0	4.6	7.7	5.0	46.0	
3	2008-12-04	4	9.2	28.0	0.0	1.8	8.0	11.0	24.0	
4	2008-12-05	4	17.5	32.3	1.0	3.8	11.9	4.0	41.0	
...	...	...	...	...	...	...	...	...	...	...
8420	2017-06-21	12	2.8	23.4	0.0	6.4	1.2	15.0	31.0	
8421	2017-06-22	12	3.6	25.3	0.0	12.6	7.1	0.0	22.0	
8422	2017-06-23	12	5.4	26.9	0.0	4.2	13.0	3.0	37.0	
8423	2017-06-24	12	7.8	27.0	0.0	4.0	13.1	12.0	28.0	
8424	2017-06-25	12	14.9	23.3	0.0	8.4	4.4	3.0	39.0	

8425 rows × 25 columns

In [39]:
```
values are transverd to new columns Month and Day columns,so now we can drop
```

```
df=df.drop(columns='Date')
# checking the co-relation values with the help of heat map
import matplotlib.pyplot as plt
corrmat = df.corr()
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(corrmat,annot=True)
```



In [40]:

```
# checking for skewness of te df dataset
df.skew()
```

Out[40]:

```
Location        0.672991
MinTemp        -0.090519
MaxTemp         0.382572
Rainfall       13.200523
Evaporation    10.103278
                -0.516592
```

```
WindGustDir       0.328877
WindGustSpeed     0.786153
WindDir9am        0.338664
WindDir3pm       -0.192588
WindSpeed9am      0.962761
WindSpeed3pm      0.492365
Humidity9am      -0.256965
Humidity3pm       0.118776
Pressure9am      -0.028521
Pressure3pm      -0.015018
Cloud9am         -0.320472
Cloud3pm         -0.230308
Temp9am          -0.014883
Temp3pm           0.400388
RainToday         1.242362
RainTomorrow      1.241588
Month             0.039388
Day               0.004260
dtype: float64
```

In [41]:

```
# checking for null values in the df dataset
df.isnull().sum()
```

Out[41]:

```
Location         0
MinTemp          0
MaxTemp          0
Rainfall         0
Evaporation      0
Sunshine         0
WindGustDir      0
WindGustSpeed    0
WindDir9am       0
WindDir3pm       0
WindSpeed9am     0
WindSpeed3pm     0
Humidity9am      0
Humidity3pm      0
Pressure9am      0
Pressure3pm      0
Cloud9am         0
Cloud3pm         0
Temp9am          0
Temp3pm          0
RainToday        0
RainTomorrow     0
Month            0
Day              0
dtype: int64
```

By seeing the above table, we can say that there is no null values in the dataset df

In [42]:

```
# finding the IQR
IQR=df.MinTemp.quantile(0.75)-df.MinTemp.quantile(0.25)
lower_bridge=df.MinTemp.quantile(0.25)-(IQR*1.5)
upper_bridge=df.MinTemp.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
-2.6999999999999993 29.3
```

In [43]:

```
df.loc[df['MinTemp']>=29.3,'MinTemp']=29.3
df.loc[df['MinTemp']<=-2.69,'MinTemp']=-2.69
```

In [44]:
```
IQR=df.MaxTemp.quantile(0.75)-df.MaxTemp.quantile(0.25)
lower_bridge=df.MaxTemp.quantile(0.25)-(IQR*1.5)
upper_bridge=df.MaxTemp.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
6.250000000000002 41.05
```

In [45]:
```
df.loc[df['MaxTemp']>=41.05,'MaxTemp']=41.05
df.loc[df['MaxTemp']<=6.25,'MaxTemp']=6.25
```

In [46]:
```
IQR=df.Rainfall.quantile(0.75)-df.Rainfall.quantile(0.25)
lower_bridge=df.Rainfall.quantile(0.25)-(IQR*1.5)
upper_bridge=df.Rainfall.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
-1.2000000000000002 2.0
```

In [47]:
```
df.loc[df['Rainfall']>=2,'Rainfall']=2
df.loc[df['Rainfall']<=-1.2,'Rainfall']=-1.2
```

In [48]:
```
IQR=df.Evaporation.quantile(0.75)-df.Evaporation.quantile(0.25)
lower_bridge=df.Evaporation.quantile(0.25)-(IQR*1.5)
upper_bridge=df.Evaporation.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
-4.0 13.600000000000001
```

In [49]:
```
df.loc[df['Evaporation']>=13.6,'Evaporation']=13.6
df.loc[df['Evaporation']<=-4,'Evaporation']=-4
```

In [50]:
```
IQR=df.WindGustSpeed.quantile(0.75)-df.WindGustSpeed.quantile(0.25)
lower_bridge=df.WindGustSpeed.quantile(0.25)-(IQR*1.5)
upper_bridge=df.WindGustSpeed.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
5.5 73.5
```

In [51]:
```
df.loc[df['WindGustSpeed']>=63.5,'WindGustSpeed']=63.5
df.loc[df['WindGustSpeed']<=5.5,'WindGustSpeed']=5.5
```

In [52]:
```
IQR=df.WindSpeed9am.quantile(0.75)-df.WindSpeed9am.quantile(0.25)
lower_bridge=df.WindSpeed9am.quantile(0.25)-(IQR*1.5)
upper_bridge=df.WindSpeed9am.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
-15.0 41.0
```

In [53]:
```
df.loc[df['WindSpeed9am']>=41,'WindSpeed9am']=41
df.loc[df['WindSpeed9am']<=-15,'WindSpeed9am']=-15
```

In [54]:
```
IQR=df.WindSpeed3pm.quantile(0.75)-df.WindSpeed3pm.quantile(0.25)
lower_bridge=df.WindSpeed3pm.quantile(0.25)-(IQR*1.5)
upper_bridge=df.WindSpeed3pm.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
-8.5 43.5
```

In [55]:
```
df.loc[df['WindSpeed3pm']>43.5,'WindSpeed3pm']=43.5
df.loc[df['WindSpeed3pm']<=-8.5,'WindSpeed3pm']=-8.5
```

In [56]:
```
IQR=df.Humidity9am.quantile(0.75)-df.Humidity9am.quantile(0.25)
lower_bridge=df.Humidity9am.quantile(0.25)-(IQR*1.5)
upper_bridge=df.Humidity9am.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
20.0 116.0
```

In [57]:
```
df.loc[df['Humidity9am']>=116,'Humidity9am']=116
df.loc[df['Humidity9am']<=20,'Humidity9am']=20
```

In [58]:
```
IQR=df.Pressure9am.quantile(0.75)-df.Pressure9am.quantile(0.25)
lower_bridge=df.Pressure9am.quantile(0.25)-(IQR*1.5)
upper_bridge=df.Pressure9am.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
1003.0500000000001 1032.25
```

In [59]:
```
df.loc[df['Pressure9am']>=1032.25,'Pressure9am']=1032.25
df.loc[df['Pressure9am']<=1003.05,'Pressure9am']=1003.05
```

In [60]:
```
IQR=df.Pressure3pm.quantile(0.75)-df.Pressure3pm.quantile(0.25)
lower_bridge=df.Pressure3pm.quantile(0.25)-(IQR*1.5)
upper_bridge=df.Pressure3pm.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
1000.3 1029.8999999999999
```

In [61]:
```
df.loc[df['Pressure3pm']>=1029.899,'Pressure3pm']=1029.899
df.loc[df['Pressure3pm']<=1000.3,'Pressure3pm']=1000.3
```

In [62]:
```
IQR=df.Temp9am.quantile(0.75)-df.Temp9am.quantile(0.25)
lower_bridge=df.Temp9am.quantile(0.25)-(IQR*1.5)
upper_bridge=df.Temp9am.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
1.650000000000004 34.05
```

In [63]:
```
df.loc[df['Temp9am']>=34.05,'Temp9am']=34.05
df.loc[df['Temp9am']<=1.65,'Temp9am']=1.65
```

In [64]:
```
IQR=df.Temp3pm.quantile(0.75)-df.Temp3pm.quantile(0.25)
lower_bridge=df.Temp3pm.quantile(0.25)-(IQR*1.5)
upper_bridge=df.Temp3pm.quantile(0.75)+(IQR*1.5)
print(lower_bridge, upper_bridge)
```

```
5.8000000000000025 38.6
```

In [65]:
```
df.loc[df['Temp3pm']>=38.6,'Temp3pm']=38.6
df.loc[df['Temp3pm']<=5.8,'Temp3pm']=5.8
```

In [66]:
```
#checking for outliers
from scipy.stats import zscore
z=np.abs(zscore(df))
print(df.shape)
print(z.shape)
threshold=3
print(np.where(z>3))
len(np.where(z>3)[0])
```

Out[66]:
```
(8425, 24)
(8425, 24)
(array([], dtype=int64), array([], dtype=int64))
0
```

The df dataset does not have any outliers

In [67]:
```
df
```

Out[67]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am
0	4	13.4	22.9	0.6	3.0	13.8	4.0	44.0	5.0
1	4	7.4	25.1	0.0	2.2	7.6	2.0	44.0	0.0
2	4	12.9	25.7	0.0	4.6	7.7	5.0	46.0	5.0
3	4	9.2	28.0	0.0	1.8	8.0	11.0	24.0	13.0
4	4	17.5	32.3	1.0	3.8	11.9	4.0	41.0	12.0
...	...	...	...	...	...	...	...	...	...
8420	12	2.8	23.4	0.0	6.4	1.2	15.0	31.0	13.0
8421	12	3.6	25.3	0.0	12.6	7.1	0.0	22.0	13.0
8422	12	5.4	26.9	0.0	4.2	13.0	3.0	37.0	13.0
8423	12	7.8	27.0	0.0	4.0	13.1	12.0	28.0	11.0
8424	12	14.9	23.3	0.0	8.4	4.4	3.0	39.0	14.0

8425 rows × 24 columns

In [68]:
```
#splliting the df dataset into x and y
x=df.drop('RainTomorrow',axis=1)
y=df['RainTomorrow']
print(x)
print(y)
```

```
      Location  MinTemp  MaxTemp  Rainfall  Evaporation  Sunshine  \
0            4     13.4     22.9       0.6          3.0      13.8
1            4      7.4     25.1       0.0          2.2       7.6
2            4     12.9     25.7       0.0          4.6       7.7
3            4      9.2     28.0       0.0          1.8       8.0
4            4     17.5     32.3       1.0          3.8      11.9
...        ...      ...      ...       ...          ...       ...
8420        12      2.8     23.4       0.0          6.4       1.2
8421        12      3.6     25.3       0.0         12.6       7.1
8422        12      5.4     26.9       0.0          4.2      13.0
8423        12      7.8     27.0       0.0          4.0      13.1
8424        12     14.9     23.3       0.0          8.4       4.4
```

```
      WindGustDir  WindGustSpeed  WindDir9am  WindDir3pm  ...  Humidity3pm  \
0             4.0           44.0         5.0         3.0  ...         22.0
1             2.0           44.0         0.0         6.0  ...         25.0
2             5.0           46.0         5.0         6.0  ...         30.0
3            11.0           24.0        13.0        14.0  ...         16.0
4             4.0           41.0        12.0         0.0  ...         33.0
...           ...            ...         ...         ...  ...          ...
8420         15.0           31.0        13.0        13.0  ...         24.0
8421          0.0           22.0        13.0         2.0  ...         21.0
8422          3.0           37.0        13.0         3.0  ...         24.0
8423         12.0           28.0        11.0         2.0  ...         24.0
8424          3.0           39.0        14.0        15.0  ...         36.0

      Pressure9am  Pressure3pm  Cloud9am  Cloud3pm  Temp9am  Temp3pm  \
0          1007.7       1007.1       8.0       8.0     16.9     21.8
1          1010.6       1007.8       1.0       4.0     17.2     24.3
2          1007.6       1008.7       2.0       2.0     21.0     23.2
3          1017.6       1012.8       8.0       7.0     18.1     26.5
4          1010.8       1006.0       7.0       8.0     17.8     29.7
...           ...          ...       ...       ...      ...      ...
8420       1024.6       1020.3       8.0       4.0     10.1     22.4
8421       1023.5       1019.1       1.0       1.0     10.9     24.5
8422       1021.0       1016.8       6.0       5.0     12.5     26.1
8423       1019.4       1016.5       3.0       2.0     15.1     26.0
8424       1020.2       1017.9       8.0       8.0     15.0     20.9

      RainToday  Month  Day
0             0     12    1
1             0     12    2
2             0     12    3
3             0     12    4
4             0     12    5
...         ...    ...  ...
8420          0      6   21
8421          0      6   22
8422          0      6   23
8423          0      6   24
8424          0      6   25

[8425 rows x 23 columns]
0       0
1       0
2       0
3       0
4       0
       ..
8420    0
8421    0
8422    0
8423    0
8424    0
Name: RainTomorrow, Length: 8425, dtype: uint8
```

In [69]:

```
#removing the skewness by yeo johnson method
from sklearn.preprocessing import power_transform
x=power_transform(x,method='yeo-johnson')
x
```

Out[69]:

```
array([[ 0.13620128,  0.04086969, -0.07771816, ..., -0.55609919,
         1.51140913, -1.89336522],
       [ 0.13620128, -1.07647605,  0.28062102, ..., -0.55609919,
         1.51140913, -1.71075845],
       [ 0.13620128, -0.05197004,  0.37494175, ..., -0.55609919,
         1.40913, -1.54413081],
```

```
       ...,
       [ 1.90092379, -1.45095611,  0.55955621, ..., -0.55609919,
        -0.0533197 ,  0.83338099],
       [ 1.90092379, -1.00172743,  0.57470753, ..., -0.55609919,
        -0.0533197 ,  0.93203309],
       [ 1.90092379,  0.3191459 , -0.01102715, ..., -0.55609919,
        -0.0533197 ,  1.02962993]])
```

In [70]:

```
# performing the minmaxscaler tecnique to bring all the values in one range 0 to 1
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
x1=mms.fit_transform(x)
```

In [71]:

```
# importing all the algorithems for checking the accuracy_score  and model perforfance
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split,cross_val_score,GridSearchCV
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,f1_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
```

In [72]:

```
model=[RandomForestClassifier(),DecisionTreeClassifier(),SVC(),KNeighborsClassifier(),Gaus
max_accuracy_score=0
for i_state in range(0,10):
    x_train,x_test,y_train,y_test=train_test_split(x1,y,random_state=i_state,test_size=0.2
    for a in model:
        a.fit(x_train,y_train)
        pred=a.predict(x_test)
        score=accuracy_score(y_test,pred)
        print('score for random_state',i_state,'is',score)
        if score>max_accuracy_score:
            max_accuracy_score=score
            Final_state=i_state
            Final_model= a
print('accuracy_score ',max_accuracy_score,'for random state ',Final_state, 'and model is
```

```
score for random_state 0 is 0.8902077151335311
score for random_state 0 is 0.827893175074184
score for random_state 0 is 0.8445103857566766
score for random_state 0 is 0.8112759643916914
score for random_state 0 is 0.7804154302670623
score for random_state 0 is 0.8314540059347181
score for random_state 1 is 0.8878338278931751
score for random_state 1 is 0.8195845697329377
score for random_state 1 is 0.8338278931750742
score for random_state 1 is 0.8059347181008902
score for random_state 1 is 0.7637982195845697
score for random_state 1 is 0.8338278931750742
score for random_state 2 is 0.8789317507418397
score for random_state 2 is 0.829673590504451
score for random_state 2 is 0.8302670623145401
score for random_state 2 is 0.8154302670623146
score for random_state 2 is 0.7916913946587537
score for random_state 2 is 0.8308605341246291
score for random_state 3 is 0.8818991097922849
score for random_state 3 is 0.8314540059347181
score for random_state 3 is 0.8219584569732937
score for random_state 3 is 0.8041543026706232
score for random_state 3 is 0.772700296735905
```

```
score for random_state 3 is 0.8201780415430268
score for random_state 4 is 0.8913946587537092
score for random_state 4 is 0.827893175074184
score for random_state 4 is 0.8480712166172106
score for random_state 4 is 0.801780415430267
score for random_state 4 is 0.7798219584569733
score for random_state 4 is 0.8356083086053413
score for random_state 5 is 0.884272997032641
score for random_state 5 is 0.8219584569732937
score for random_state 5 is 0.8373887240356083
score for random_state 5 is 0.8029673590504451
score for random_state 5 is 0.771513353115727
score for random_state 5 is 0.829080118694362
score for random_state 6 is 0.8818991097922849
score for random_state 6 is 0.8130563798219584
score for random_state 6 is 0.8367952522255193
score for random_state 6 is 0.8077151335311573
score for random_state 6 is 0.7810089020771513
score for random_state 6 is 0.8320474777448071
score for random_state 7 is 0.8741839762611276
score for random_state 7 is 0.8142433234421365
score for random_state 7 is 0.8243323442136499
score for random_state 7 is 0.8071216617210683
score for random_state 7 is 0.7798219584569733
score for random_state 7 is 0.8267062314540059
score for random_state 8 is 0.8807121661721068
score for random_state 8 is 0.8249258160237388
score for random_state 8 is 0.8207715133531157
score for random_state 8 is 0.7988130563798219
score for random_state 8 is 0.771513353115727
score for random_state 8 is 0.8207715133531157
score for random_state 9 is 0.8896142433234422
score for random_state 9 is 0.8314540059347181
score for random_state 9 is 0.8415430267062315
score for random_state 9 is 0.8225519287833828
score for random_state 9 is 0.7863501483679525
score for random_state 9 is 0.8338278931750742
accuracy_score  0.8913946587537092 for random state  4 and model is  RandomForestClassifie
r()
```

In [73]:

```
# we are training the model with RandomForestClassifier for randomstate 4 and checking the
rfc=RandomForestClassifier()
x_train,x_test,y_train,y_test=train_test_split(x1,y,random_state=4,test_size=0.2)
rfc.fit(x_train,y_train)
rfc.score(x_train,y_train)
pred_y=rfc.predict(x_test)
rfcs=accuracy_score(y_test,pred_y)
print('accuracy_score =',rfcs*100)
print(classification_report(y_test,pred_y))
print(confusion_matrix(y_test,pred_y))
print('F1_score = ',f1_score(y_test,pred_y)*100)
from sklearn.model_selection import cross_val_score
cv_score=cross_val_score(rfc,x1,y,cv=5)
cv_mean=cv_score.mean()
print("cross_val_score=",cv_mean*100)
```

```
accuracy_score = 88.4272997032641
              precision    recall  f1-score   support

           0       0.89      0.97      0.93      1290
           1       0.87      0.59      0.71       395

    accuracy                           0.88      1685
   macro avg       0.88      0.78      0.82      1685
```

```
weighted avg       0.88      0.88      0.88      1685

[[1256   34]
 [ 161  234]]
F1_score =  70.58823529411764
cross_val_score= 85.09198813056379
```

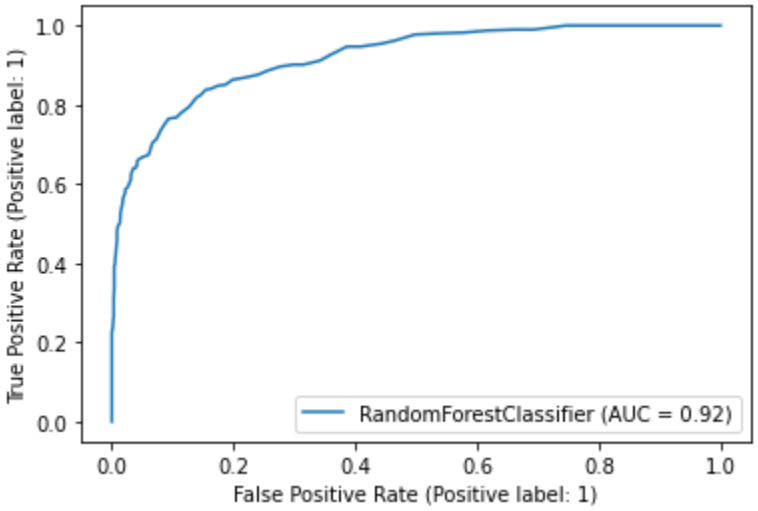
accuracy_score is 88.4 which is good value, f1_score, cross_validation_score,classification_report and confussion_matrix values also calculated

In [79]:

```
from sklearn import metrics
metrics.plot_roc_curve(rfc, x_test, y_test)
metrics.roc_auc_score(y_test, pred_y, average=None)
```

Out[79]:

```
0.7830242370719264
```



By the above graph we can say that area under the curve is 92% which is very good value

CONCLUSION

1) area under the curve is 92percent which is very good value

2) The RandomForestClassifier() giving the best accuracy value

3) accuracy_score, F1_score, Classification_report, Confussion_matrix and also AUC value is also shown in the above table

4) RandomForestClassifier is giving the best accuracy score so we need to save the RandomForestClassifier predicted values by using pickel

In [ ]: