In [1]:
```
# Import required libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df= pd.read_csv("https://raw.githubusercontent.com/dsrscientist/dataset3/main/wea
```

In [3]:
```
df
```

Out[3]:

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

8425 rows × 23 columns

In [4]:
```
# Top 5 records
df.head()
```

Out[4]:

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

5 rows × 23 columns

In [5]:
```
df.tail()
```

Out[5]:

	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 [6]:
```
print("Dataset have \nRows-",df.shape[0],'\nColumns -',df.shape[1])
```

```
Dataset have
Rows- 8425
Columns - 23
```

In [7]:
```
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 [8]:
```
df.describe().T
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
MinTemp	8350.0	13.193305	5.403596	-2.0	9.20	13.3	17.4	28.5
MaxTemp	8365.0	23.859976	6.136408	8.2	19.30	23.3	28.0	45.5
Rainfall	8185.0	2.805913	10.459379	0.0	0.00	0.0	1.0	371.0
Evaporation	4913.0	5.389395	5.044484	0.0	2.60	4.6	7.0	145.0
Sunshine	4431.0	7.632205	3.896235	0.0	4.75	8.7	10.7	13.9
WindGustSpeed	7434.0	40.174469	14.665721	7.0	30.00	39.0	50.0	107.0
WindSpeed9am	8349.0	13.847646	10.174579	0.0	6.00	13.0	20.0	63.0
WindSpeed3pm	8318.0	18.533662	9.766986	0.0	11.00	19.0	24.0	83.0
Humidity9am	8366.0	67.822496	16.833283	10.0	56.00	68.0	80.0	100.0
Humidity3pm	8323.0	51.249790	18.423774	6.0	39.00	51.0	63.0	99.0
Pressure9am	7116.0	1017.640233	6.828699	989.8	1013.00	1017.7	1022.3	1039.0
Pressure3pm	7113.0	1015.236075	6.766681	982.9	1010.40	1015.3	1019.8	1036.0
Cloud9am	6004.0	4.566622	2.877658	0.0	1.00	5.0	7.0	8.0
Cloud3pm	5970.0	4.503183	2.731659	0.0	2.00	5.0	7.0	8.0
Temp9am	8369.0	17.762015	5.627035	1.9	13.80	17.8	21.9	39.4
Temp3pm	8329.0	22.442934	5.980020	7.3	18.00	21.9	26.4	44.1

Missing Values

In [9]:
```
df.isnull().sum()
```

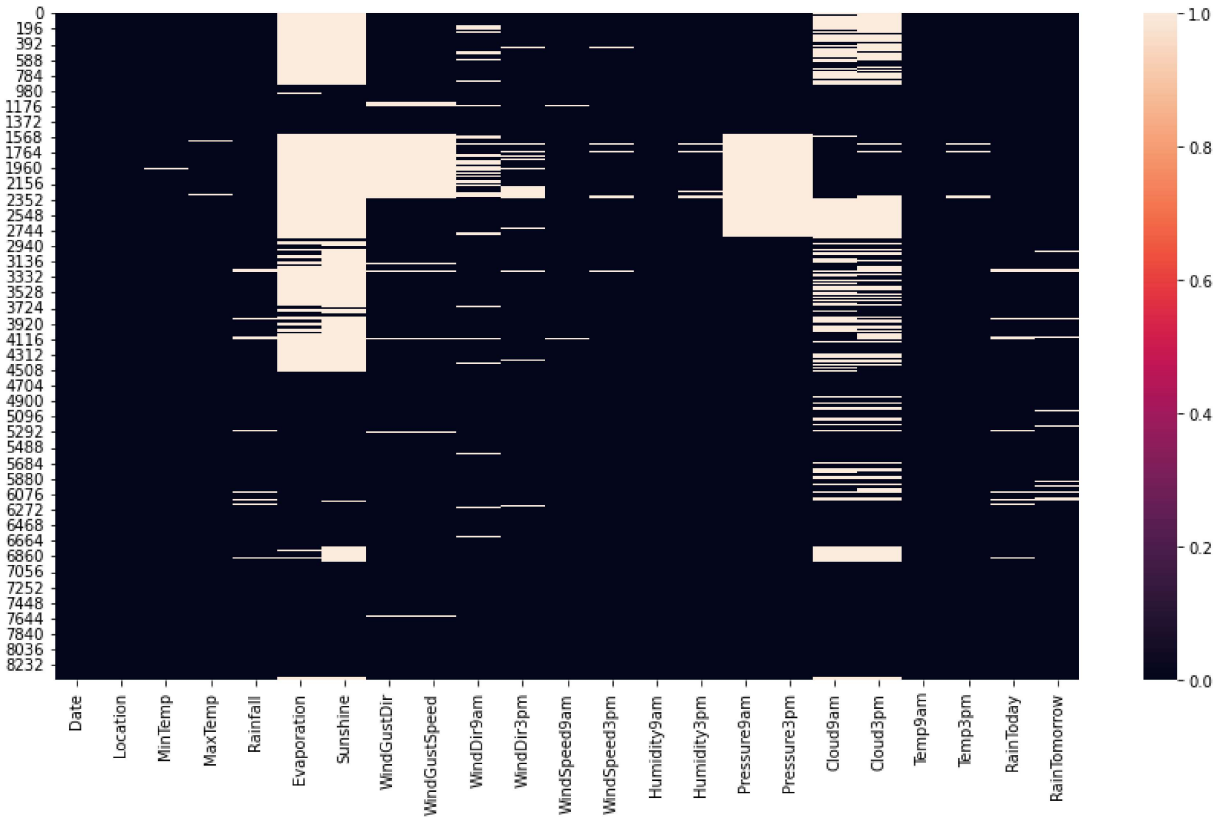
Out[9]:
```
Date              0
Location          0
MinTemp          75
MaxTemp          60
Rainfall        240
Evaporation    3512
Sunshine       3994
WindGustDir     991
WindGustSpeed   991
WindDir9am      829
WindDir3pm      308
WindSpeed9am     76
WindSpeed3pm    107
Humidity9am      59
Humidity3pm     102
Pressure9am    1309
Pressure3pm    1312
Cloud9am       2421
Cloud3pm       2455
Temp9am          56
Temp3pm          96
RainToday       240
RainTomorrow    239
dtype: int64
```

In [10]:
```
round(df.isnull().sum()/df.shape[0]*100,2).sort_values(ascending=False)
```

Out[10]:
```
Sunshine         47.41
Evaporation      41.69
Cloud3pm         29.14
Cloud9am         28.74
Pressure3pm      15.57
Pressure9am      15.54
WindGustDir      11.76
WindGustSpeed    11.76
WindDir9am        9.84
WindDir3pm        3.66
RainToday         2.85
Rainfall          2.85
RainTomorrow      2.84
WindSpeed3pm      1.27
Humidity3pm       1.21
Temp3pm           1.14
WindSpeed9am      0.90
MinTemp           0.89
MaxTemp           0.71
Humidity9am       0.70
Temp9am           0.66
Location          0.00
Date              0.00
dtype: float64
```

In [11]:
```
plt.figure(figsize=(15,8))
sns.heatmap(df.isnull())
```

Out[11]: <AxesSubplot:>



Alot of missing values are present in the dataset

In [12]:
```
df.columns
```

Out[12]:
```
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 [13]:
```
catg_features=[col for col in df.columns if df[col].dtypes=='object']
cont_features=[col for col in df.columns if df[col].dtypes!='object']
```

In [14]:
```
catg_features
```

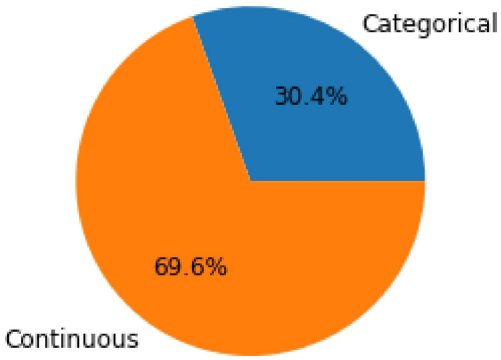
Out[14]:
```
['Date',
 'Location',
 'WindGustDir',
 'WindDir9am',
 'WindDir3pm',
 'RainToday',
 'RainTomorrow']
```

In [15]:
```
cont_features
```

Out[15]:
```
['MinTemp',
 'MaxTemp',
 'Rainfall',
 'Evaporation',
 'Sunshine',
 'WindGustSpeed',
 'WindSpeed9am',
 'WindSpeed3pm',
 'Humidity9am',
 'Humidity3pm',
 'Pressure9am',
 'Pressure3pm',
 'Cloud9am',
 'Cloud3pm',
 'Temp9am',
 'Temp3pm']
```

In [16]:
```
plt.pie([len(catg_features),len(cont_features)],labels=['Categorical','Continuous
```

Out[16]:
```
([<matplotlib.patches.Wedge at 0x1fa85eb8760>,
  <matplotlib.patches.Wedge at 0x1fa85eb8f40>],
 [Text(0.6343483909086429, 0.8986668564888858, 'Categorical'),
  Text(-0.6343483909086428, -0.8986668564888859, 'Continuous')],
 [Text(0.3460082132228961, 0.4901819217212104, '30.4%'),
  Text(-0.34600821322289604, -0.49018192172121045, '69.6%')])
```



Target Feature

In [17]:
```
df['RainTomorrow'].unique()
```

Out[17]:
```
array(['No', 'Yes', nan], dtype=object)
```

In [18]:
```
df['RainTomorrow'].value_counts(normalize=True,dropna=False)*100
```

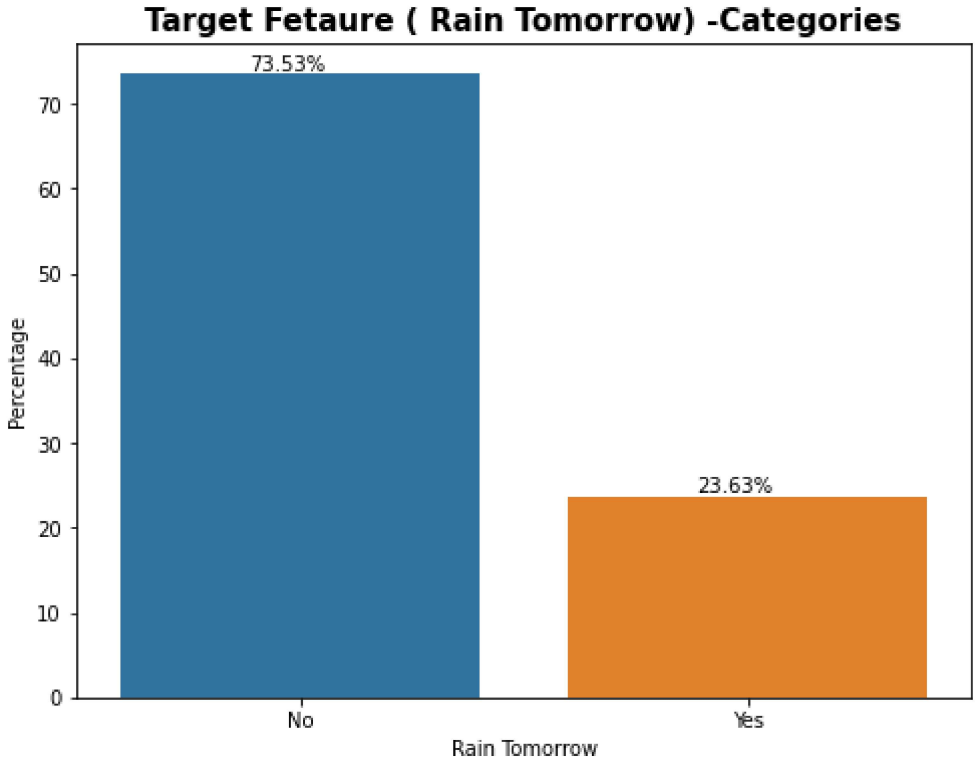
Out[18]:
```
No     73.531157
Yes    23.632047
NaN     2.836795
Name: RainTomorrow, dtype: float64
```

In [19]:
```
target_df=df['RainTomorrow'].value_counts(normalize=True,dropna=False)*100
```

In [20]:
```
plt.figure(figsize=(8,6))
plt.title("Target Fetaure ( Rain Tomorrow) -Categories",fontweight='bold',fontsi
ax=sns.barplot(x=target_df.index,y=target_df.values)
plt.xlabel('Rain Tomorrow')
plt.ylabel('Percentage')



for p in ax.patches:
    height=p.get_height()
    width=p.get_width()
    x,_=p.get_xy()
    ax.text(x +width/2.8,height+.5,f'{height:.2f}%')
```



Date

In [21]:
```
df['Date'].nunique()
```

Out[21]: 3004

Location

In [22]:
```
df['Location'].unique()
```

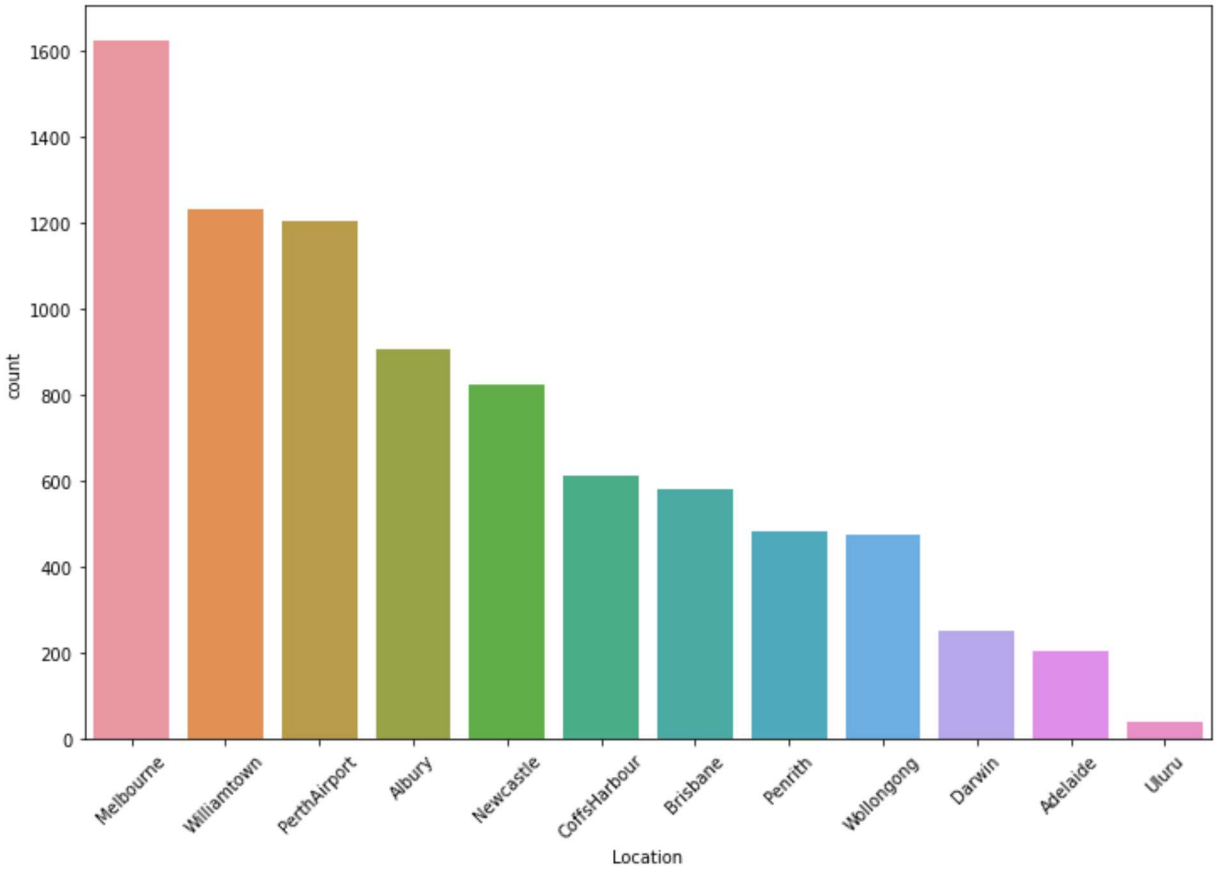
Out[22]:
```
array(['Albury', 'CoffsHarbour', 'Newcastle', 'Penrith', 'Williamtown',
       'Wollongong', 'Melbourne', 'Brisbane', 'Adelaide', 'PerthAirport',
       'Darwin', 'Uluru'], dtype=object)
```

In [23]:
```
df['Location'].value_counts()
```

Out[23]:
```
Melbourne       1622
Williamtown     1230
PerthAirport    1204
Albury           907
Newcastle        822
CoffsHarbour     611
Brisbane         579
Penrith          482
Wollongong       474
Darwin           250
Adelaide         205
Uluru             39
Name: Location, dtype: int64
```

In [24]:
```
plt.figure(figsize=(12,8))
sns.countplot(df['Location'],order=df['Location'].value_counts().index)
plt.xticks(rotation=45)
```

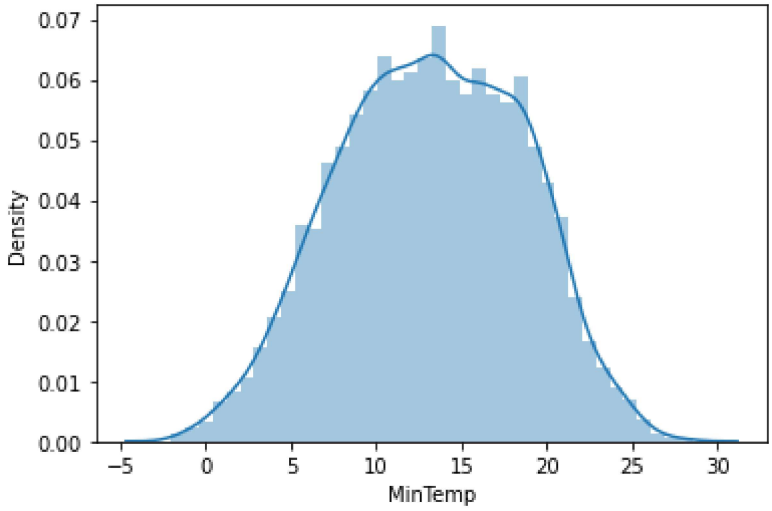
Out[24]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11]),
 [Text(0, 0, 'Melbourne'),
  Text(1, 0, 'Williamtown'),
  Text(2, 0, 'PerthAirport'),
  Text(3, 0, 'Albury'),
  Text(4, 0, 'Newcastle'),
  Text(5, 0, 'CoffsHarbour'),
  Text(6, 0, 'Brisbane'),
  Text(7, 0, 'Penrith'),
  Text(8, 0, 'Wollongong'),
  Text(9, 0, 'Darwin'),
  Text(10, 0, 'Adelaide'),
  Text(11, 0, 'Uluru')])
```



MinTemp

In [25]:
```
sns.distplot(df['MinTemp'])
```

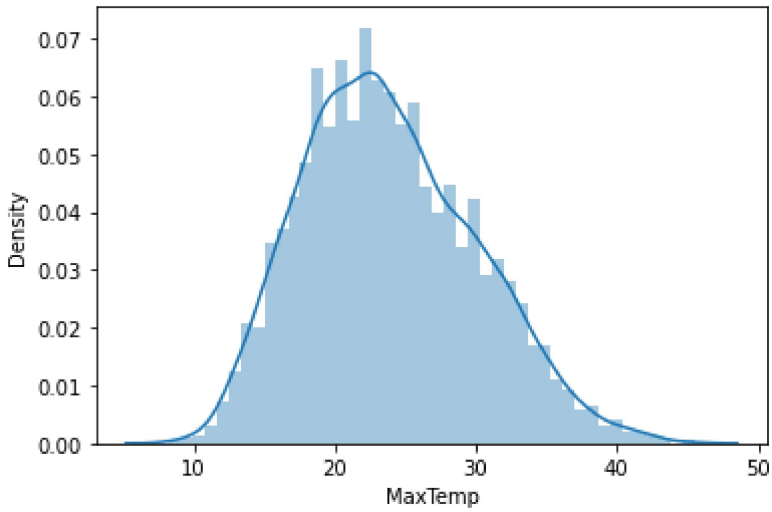
Out[25]:
```
<AxesSubplot:xlabel='MinTemp', ylabel='Density'>
```



MaxTemp

In [26]:
```
sns.distplot(df['MaxTemp'])
```

Out[26]:
```
<AxesSubplot:xlabel='MaxTemp', ylabel='Density'>
```



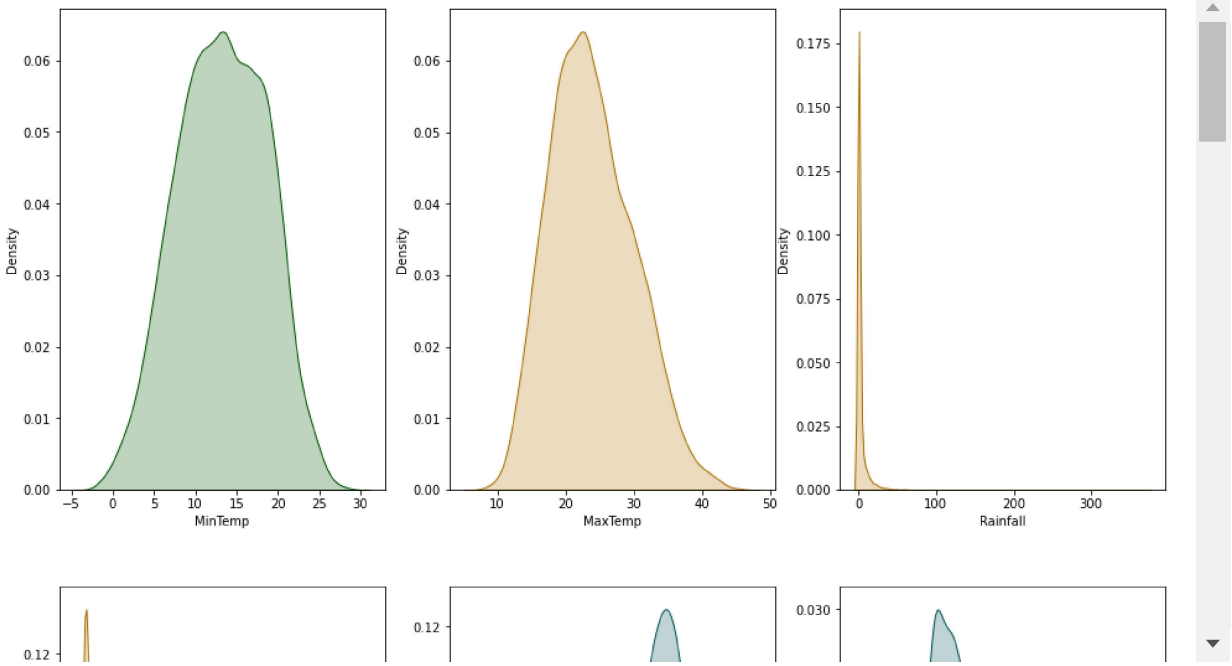
Distribution of Continuous features

In [27]:
```
import random

color_=['#000057','#005757','#005700','#ad7100','#008080','#575757','#003153']
cmap_=['magma','copper','crest']
```

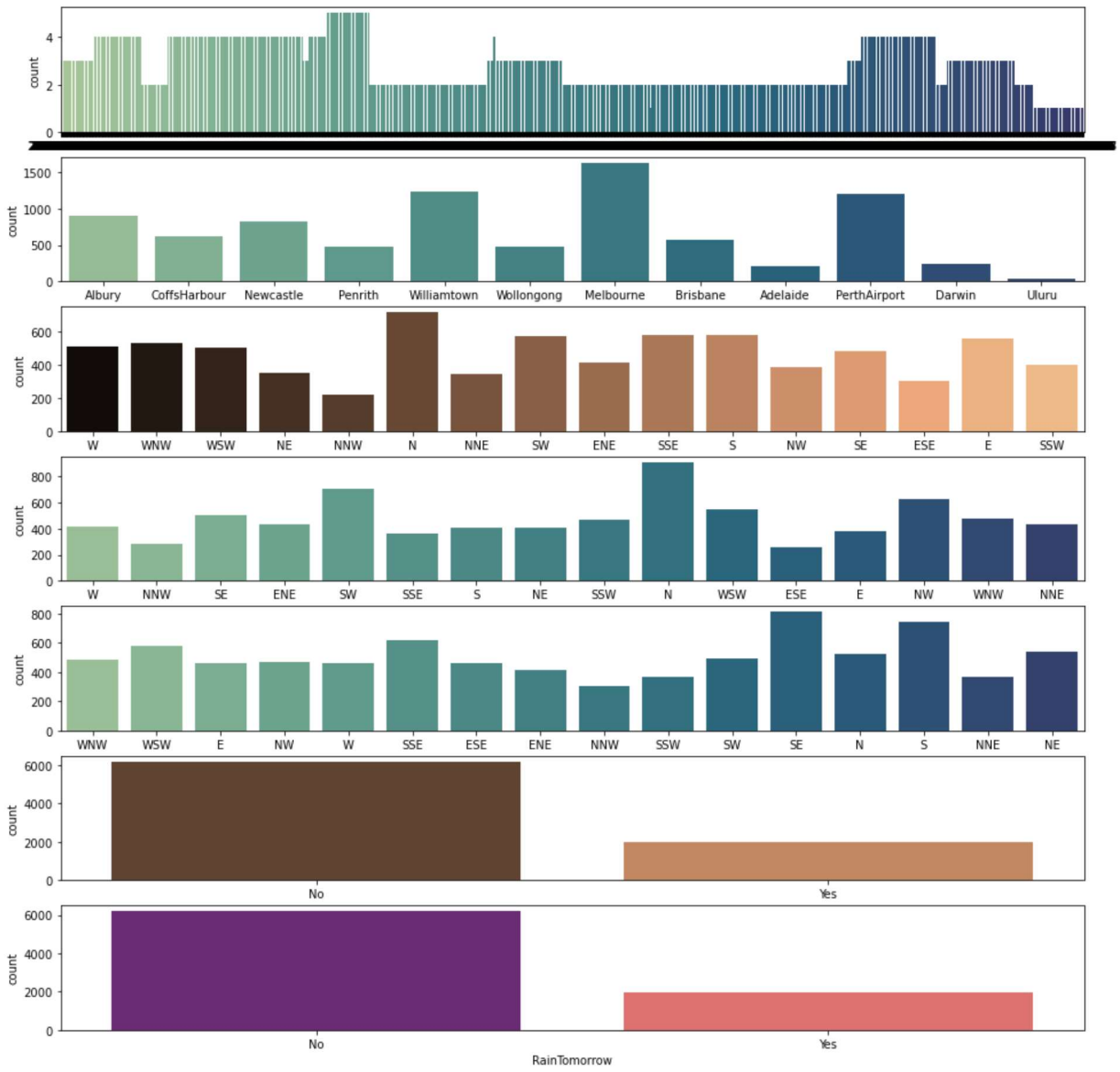
In [28]:
```
plt.figure(figsize=(16,50))
for i,col in enumerate(df[cont_features].columns):
    rand_col=color_[random.sample(range(6),1)[0]]
    plt.subplot(6,3,i+1)

    sns.kdeplot(data=df,x=col,color=rand_col,fill=rand_col,palette=cmap_[random.s
```



In [29]:
```
plt.figure(figsize=(16,50))
for i,col in enumerate(df[catg_features].columns):
    rand_col=color_[random.sample(range(6),1)[0]]
    plt.subplot(21,1,i+1)

    sns.countplot(data=df,x=col,color=rand_col,fill=rand_col,palette=cmap_[randon
```



Bivarient EDA

In [30]:
```
df.columns
```

Out[30]:
```
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 [31]:
```
df.head(2)
```

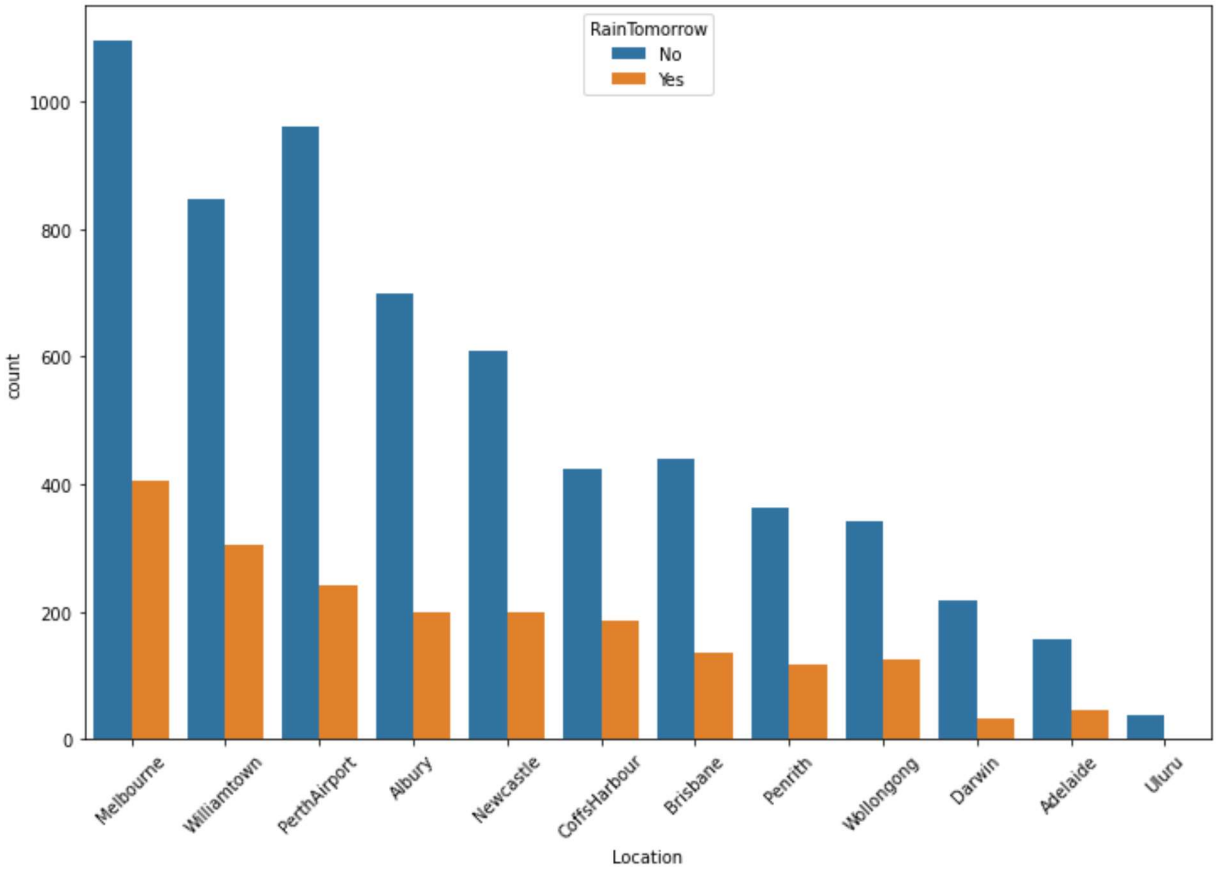
Out[31]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSp
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	

2 rows × 23 columns

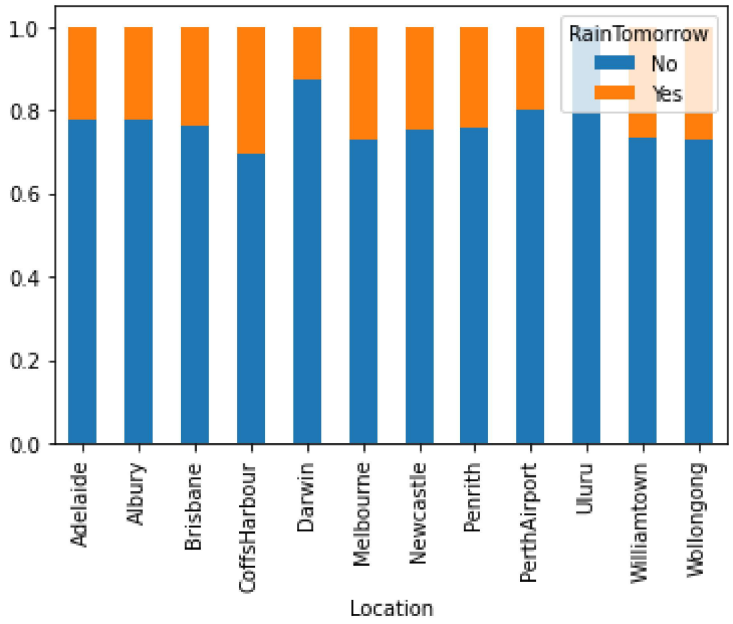
In [32]:
```
plt.figure(figsize=(12,8))
sns.countplot(df['Location'],order=df['Location'].value_counts().index,hue=df['Ra
plt.xticks(rotation=45)
```

Out[32]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11]),
 [Text(0, 0, 'Melbourne'),
  Text(1, 0, 'Williamtown'),
  Text(2, 0, 'PerthAirport'),
  Text(3, 0, 'Albury'),
  Text(4, 0, 'Newcastle'),
  Text(5, 0, 'CoffsHarbour'),
  Text(6, 0, 'Brisbane'),
  Text(7, 0, 'Penrith'),
  Text(8, 0, 'Wollongong'),
  Text(9, 0, 'Darwin'),
  Text(10, 0, 'Adelaide'),
  Text(11, 0, 'Uluru')])
```



In [33]:
```
table=pd.crosstab(df['Location'],df['RainTomorrow'])
table.div(table.sum(1),axis=0).plot(kind='bar',stacked=True)
```

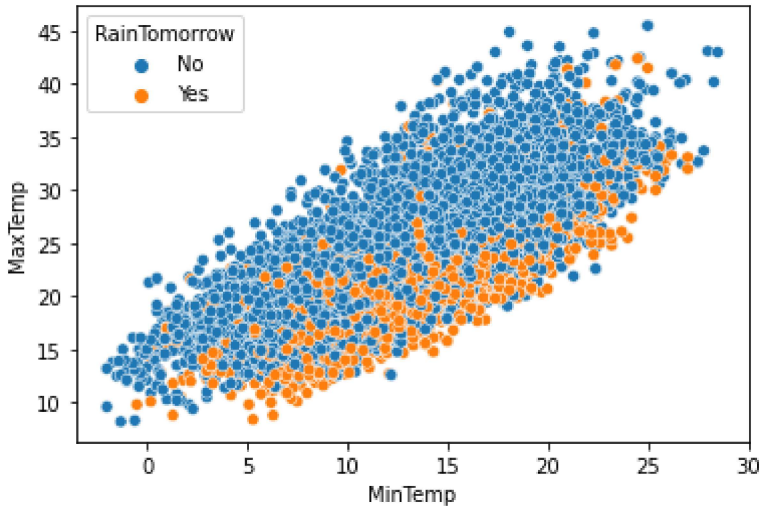
Out[33]: <AxesSubplot:xlabel='Location'>



MinTemp', 'MaxTemp

In [34]:
```
sns.scatterplot(df['MinTemp'],df['MaxTemp'],hue='RainTomorrow',data=df)
```

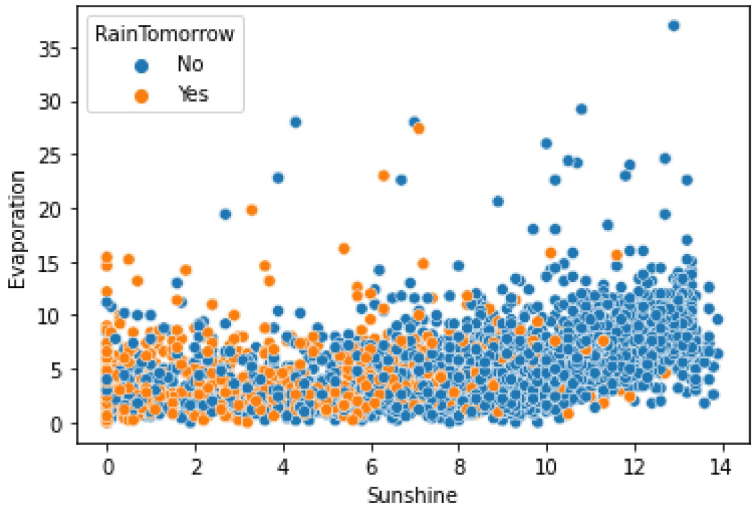
Out[34]: <AxesSubplot:xlabel='MinTemp', ylabel='MaxTemp'>



'Evaporation'

In [35]:
```
sns.scatterplot(df['Sunshine'],df['Evaporation'],hue='RainTomorrow',data=df)
```

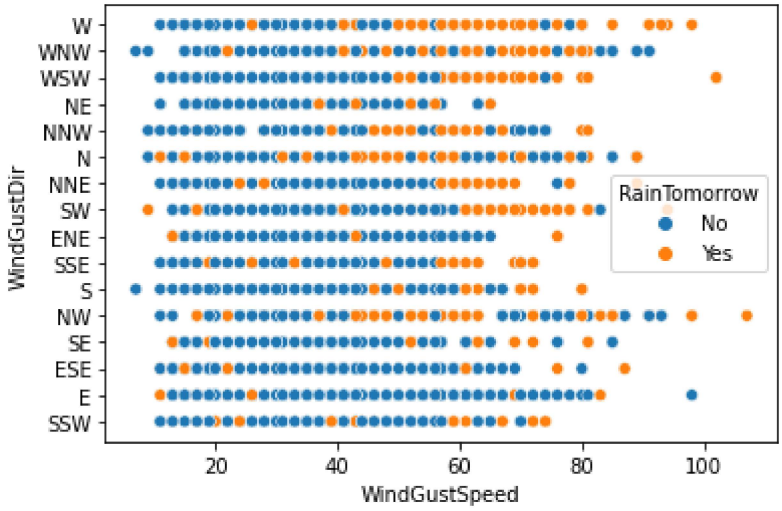
Out[35]: <AxesSubplot:xlabel='Sunshine', ylabel='Evaporation'>



WindGustDir', 'WindGustSpeed

In [36]:
```
sns.scatterplot(df['WindGustSpeed'],df['WindGustDir'],hue='RainTomorrow',data=df)
```

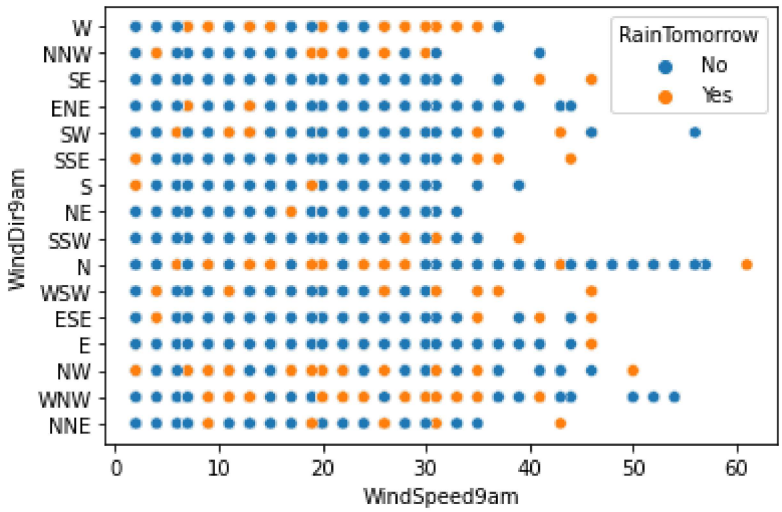
Out[36]: <AxesSubplot:xlabel='WindGustSpeed', ylabel='WindGustDir'>



WindDir9am WindSpeed9am

In [37]:
```
sns.scatterplot(df['WindSpeed9am'],df['WindDir9am'],hue='RainTomorrow',data=df)
```

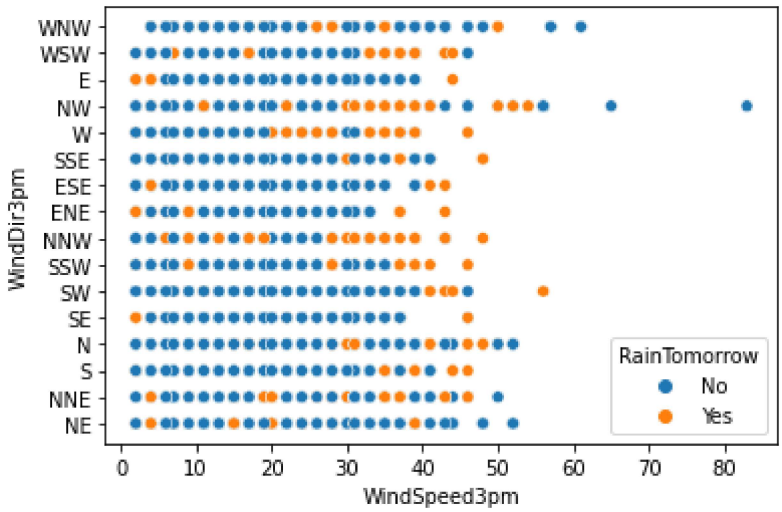
Out[37]:
```
<AxesSubplot:xlabel='WindSpeed9am', ylabel='WindDir9am'>
```



WindDir3pm WindSpeed3pm

In [38]:
```
sns.scatterplot(df['WindSpeed3pm'],df['WindDir3pm'],hue='RainTomorrow',data=df)
```

Out[38]:
```
<AxesSubplot:xlabel='WindSpeed3pm', ylabel='WindDir3pm'>
```



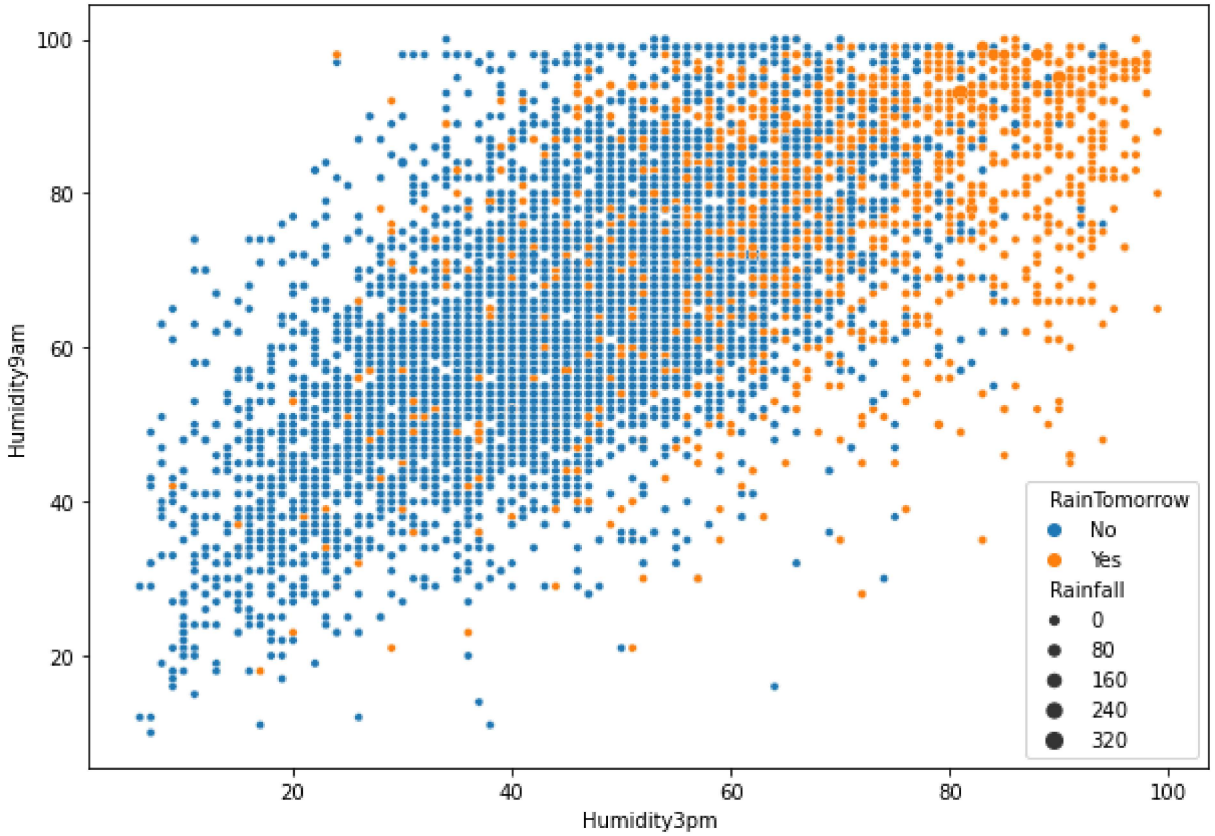
In [39]:
```
df.columns
```

Out[39]:
```
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')
```

'Humidity9am', 'Humidity3pm'

In [40]:
```
plt.figure(figsize=(10,7))
sns.scatterplot(df['Humidity3pm'],df['Humidity9am'],hue='RainTomorrow',size='Rain
```

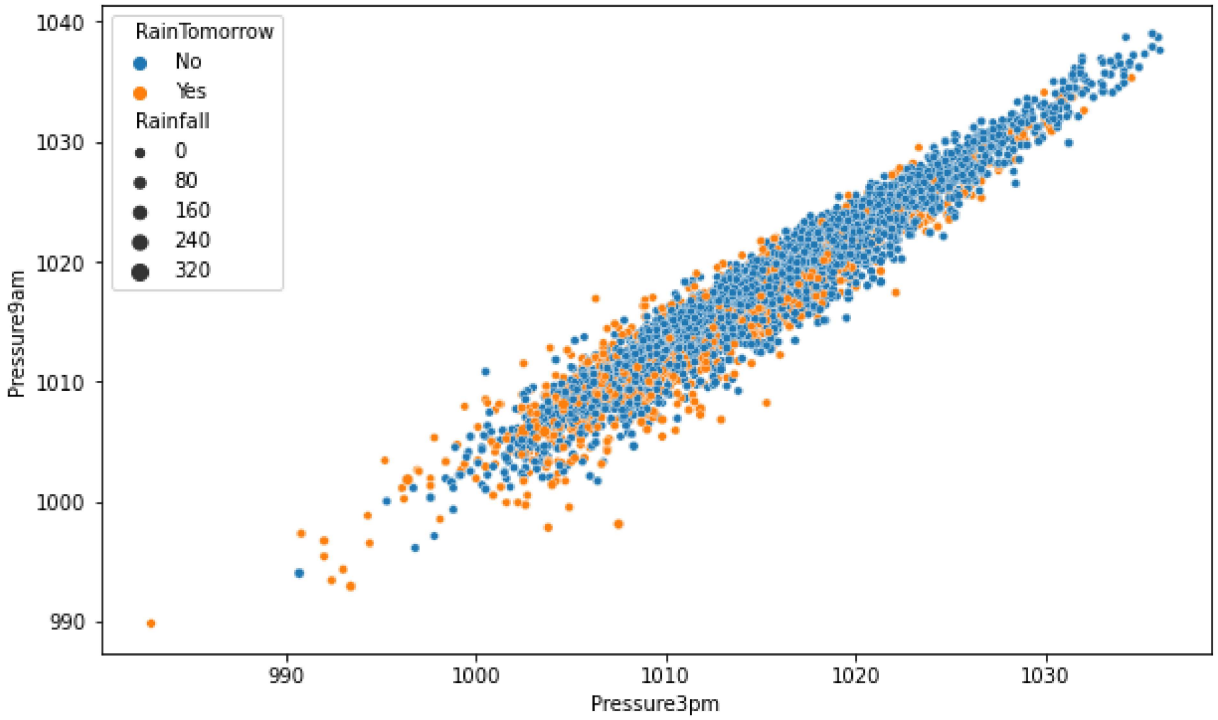
Out[40]:
```
<AxesSubplot:xlabel='Humidity3pm', ylabel='Humidity9am'>
```



Pressure9am', 'Pressure3pm

In [41]:
```
plt.figure(figsize=(10,6))
sns.scatterplot(df['Pressure3pm'],df['Pressure9am'],hue='RainTomorrow',size='Rain
```

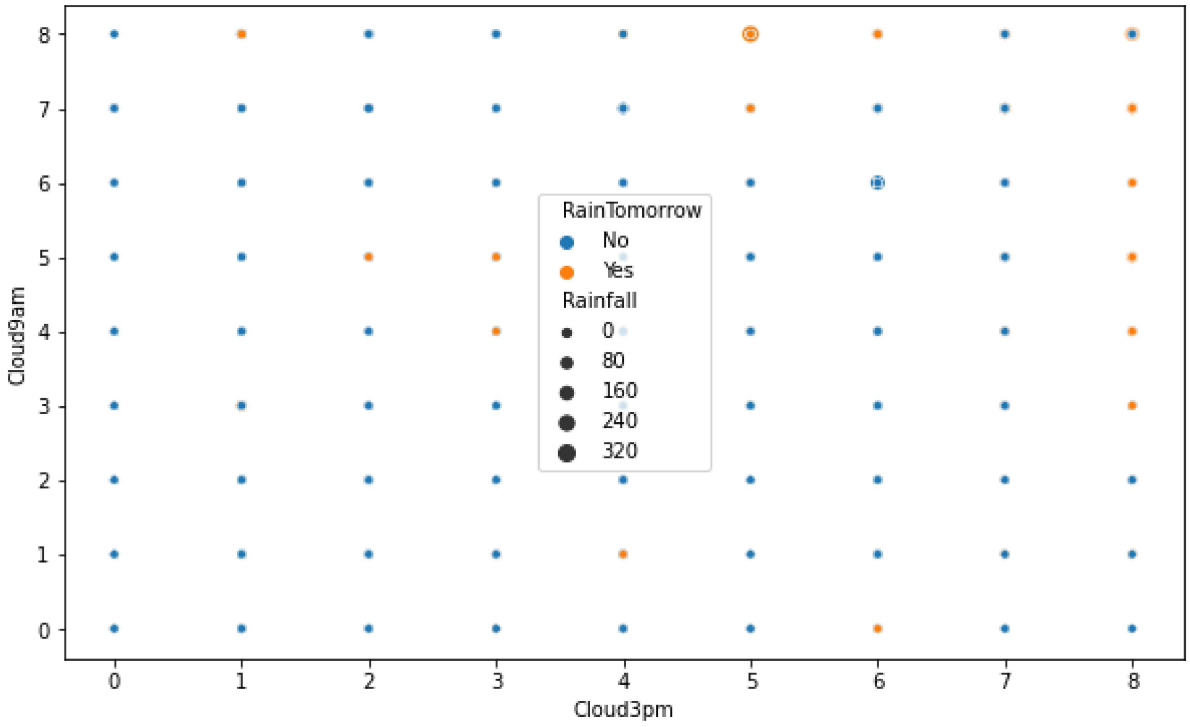
Out[41]:
```
<AxesSubplot:xlabel='Pressure3pm', ylabel='Pressure9am'>
```



'Cloud9am', 'Cloud3pm'

In [42]:
```
plt.figure(figsize=(10,6))
sns.scatterplot(df['Cloud3pm'],df['Cloud9am'],hue='RainTomorrow',size='Rainfall',
```

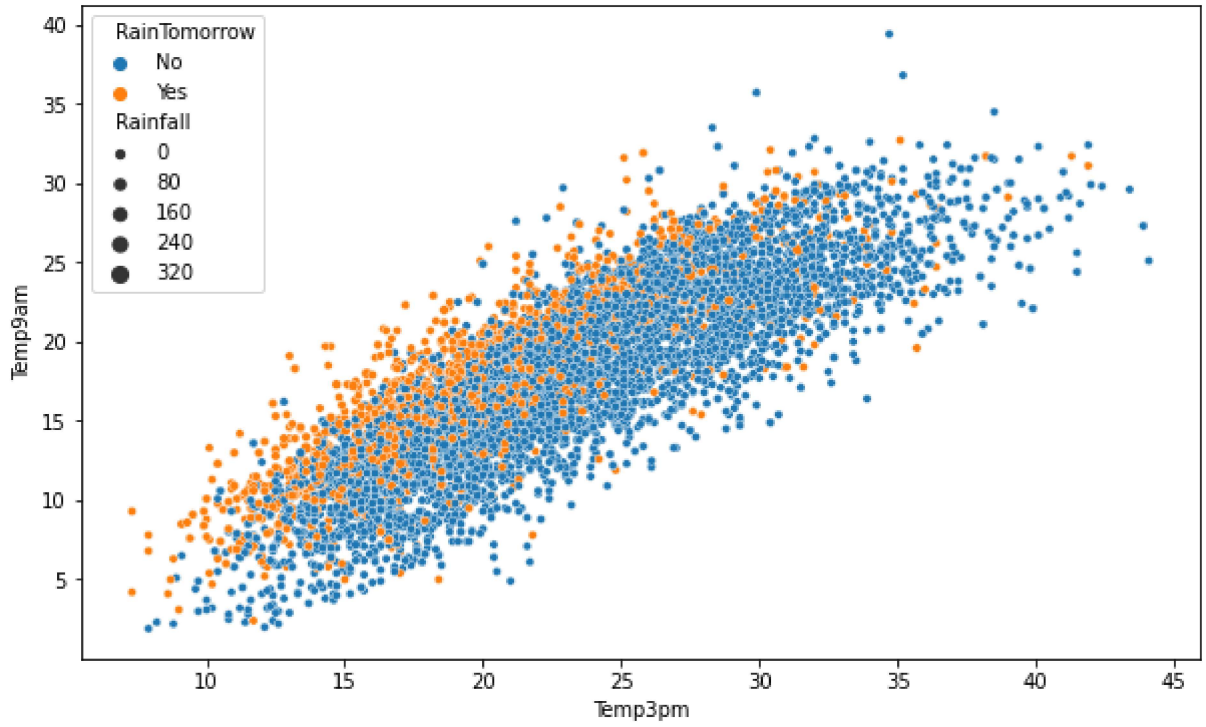
Out[42]:
```
<AxesSubplot:xlabel='Cloud3pm', ylabel='Cloud9am'>
```



Temp9am','Temp3pm

In [43]:
```
plt.figure(figsize=(10,6))
sns.scatterplot(df['Temp3pm'],df['Temp9am'],hue='RainTomorrow',size='Rainfall',da
```

Out[43]:
```
<AxesSubplot:xlabel='Temp3pm', ylabel='Temp9am'>
```



RainToday', 'RainTomorrow'

In [44]:
```
df['RainToday'].value_counts()
```

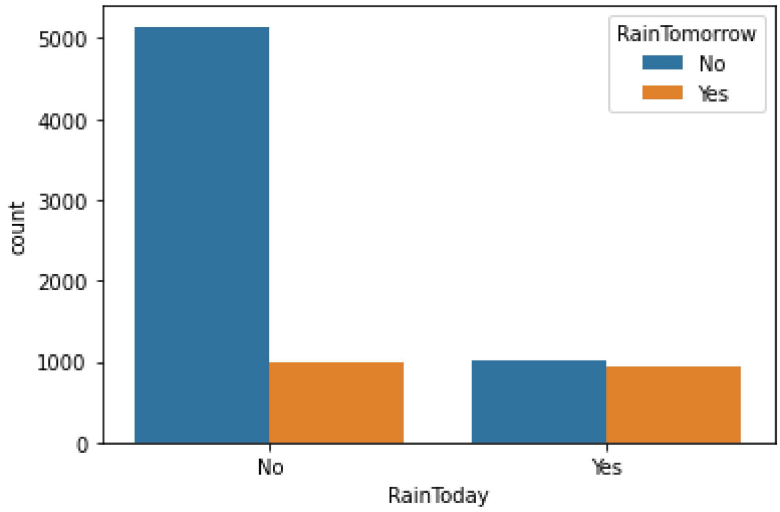
Out[44]:
```
No     6195
Yes    1990
Name: RainToday, dtype: int64
```

In [45]:
```
df.groupby('RainToday')['RainTomorrow'].value_counts()
```

Out[45]:
```
RainToday  RainTomorrow
No         No              5142
           Yes              978
Yes        No              1013
           Yes              946
Name: RainTomorrow, dtype: int64
```

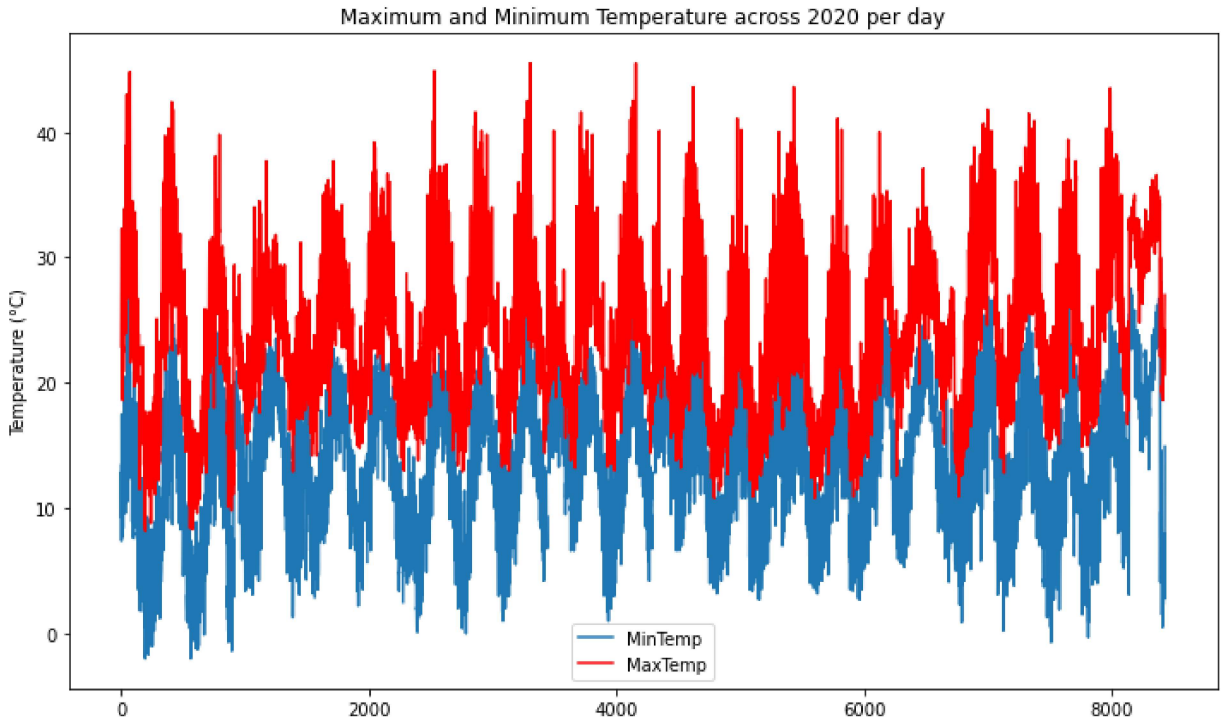
In [46]:
```
sns.countplot(df['RainToday'],hue=df['RainTomorrow'])
```

Out[46]: <AxesSubplot:xlabel='RainToday', ylabel='count'>



In [47]:
```
max_temp = df['MaxTemp']
min_temp = df['MinTemp']

min_temp.plot(figsize=(12,7), legend=True)
max_temp.plot(figsize=(12,7), color='r', legend=True)
plt.title('Maximum and Minimum Temperature across 2020 per day')
plt.ylabel('Temperature (°C)')
plt.show()
```

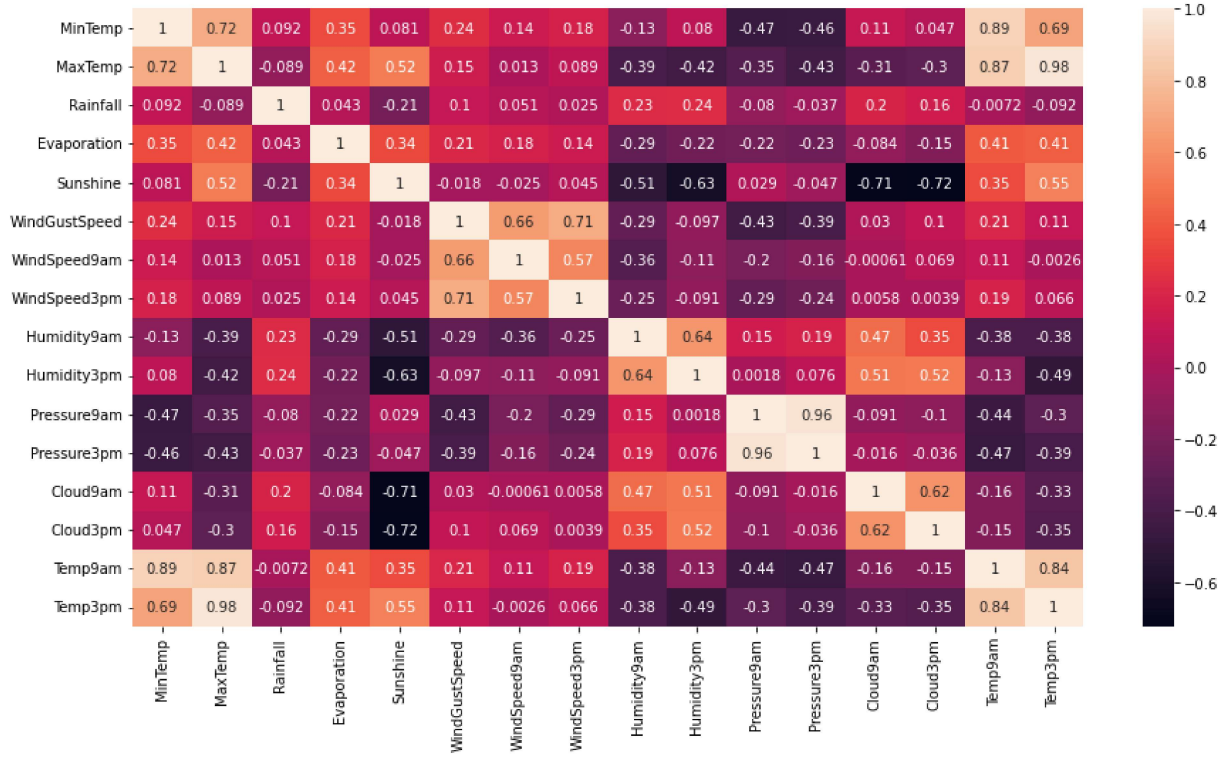


Correlation

In [48]:
```
#df['RainTomorrow'].replace('No',0,inplace=True)
#df['RainTomorrow'].replace('Yes',1,inplace=True)
#df['RainToday'].replace('No',0,inplace=True)
#df['RainToday'].replace('Yes',1,inplace=True)
```

In [49]:
```
plt.figure(figsize=(15,8))
sns.heatmap(df.corr(),annot=True)
```

Out[49]: <AxesSubplot:>



In [ ]:
```
```

Create Model to predict if TrainTomorrow will happen or not

In [50]:
```
round(df.isnull().sum()/df.shape[0]*100,2).sort_values(ascending=False)
```

Out[50]:
```
Sunshine         47.41
Evaporation      41.69
Cloud3pm         29.14
Cloud9am         28.74
Pressure3pm      15.57
Pressure9am      15.54
WindGustDir      11.76
WindGustSpeed    11.76
WindDir9am        9.84
WindDir3pm        3.66
RainToday         2.85
Rainfall          2.85
RainTomorrow      2.84
WindSpeed3pm      1.27
Humidity3pm       1.21
Temp3pm           1.14
WindSpeed9am      0.90
MinTemp           0.89
MaxTemp           0.71
Humidity9am       0.70
Temp9am           0.66
Location          0.00
Date              0.00
dtype: float64
```

In [51]:
```
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 [52]:
```
# seprate database on basis of RainTomorrow

yes_rain = df[df['RainTomorrow']=='Yes']
no_rain = df[df['RainTomorrow']=='No']
```

In [53]:
```
yes_rain.shape , no_rain.shape
```

Out[53]:
```
((1991, 23), (6195, 23))
```

In [54]:
```
# For Temparatures replacing NaN with its respective mode value # Mode is most re
```

In [55]:
```
yes_rain['MinTemp'].fillna(yes_rain['MinTemp'].mode()[0],inplace=True)
no_rain['MinTemp'].fillna(no_rain['MinTemp'].mode()[0],inplace=True)

yes_rain['MaxTemp'].fillna(yes_rain['MaxTemp'].mode()[0],inplace=True)
no_rain['MaxTemp'].fillna(no_rain['MaxTemp'].mode()[0],inplace=True)

yes_rain['Temp9am'].fillna(yes_rain['Temp9am'].mode()[0],inplace=True)
no_rain['Temp9am'].fillna(no_rain['Temp9am'].mode()[0],inplace=True)

yes_rain['Temp3pm'].fillna(yes_rain['Temp3pm'].mode()[0],inplace=True)
no_rain['Temp3pm'].fillna(no_rain['Temp3pm'].mode()[0],inplace=True)


yes_rain['Humidity3pm'].fillna(yes_rain['Humidity3pm'].mode()[0],inplace=True)
no_rain['Humidity3pm'].fillna(no_rain['Humidity3pm'].mode()[0],inplace=True)

yes_rain['Humidity9am'].fillna(yes_rain['Humidity9am'].mode()[0],inplace=True)
no_rain['Humidity9am'].fillna(no_rain['Humidity9am'].mode()[0],inplace=True)
```

In [56]:
```
round(df.isnull().sum()/df.shape[0]*100,2).sort_values(ascending=False)
```

Out[56]:
```
Sunshine         47.41
Evaporation      41.69
Cloud3pm         29.14
Cloud9am         28.74
Pressure3pm      15.57
Pressure9am      15.54
WindGustDir      11.76
WindGustSpeed    11.76
WindDir9am        9.84
WindDir3pm        3.66
RainToday         2.85
Rainfall          2.85
RainTomorrow      2.84
WindSpeed3pm      1.27
Humidity3pm       1.21
Temp3pm           1.14
WindSpeed9am      0.90
MinTemp           0.89
MaxTemp           0.71
Humidity9am       0.70
Temp9am           0.66
Location          0.00
Date              0.00
dtype: float64
```

In [57]:
```
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 [58]:
```
yes_rain['Sunshine'].fillna(yes_rain['Sunshine'].median(),inplace=True)
no_rain['Sunshine'].fillna(no_rain['Sunshine'].median(),inplace=True)

yes_rain['Evaporation'].fillna(yes_rain['Evaporation'].median(),inplace=True)
no_rain['Evaporation'].fillna(no_rain['Evaporation'].median(),inplace=True)

yes_rain['Cloud3pm'].fillna(yes_rain['Cloud3pm'].median(),inplace=True)
no_rain['Cloud3pm'].fillna(no_rain['Cloud3pm'].median(),inplace=True)

yes_rain['Cloud9am'].fillna(yes_rain['Cloud9am'].median(),inplace=True)
no_rain['Cloud9am'].fillna(no_rain['Cloud9am'].median(),inplace=True)

yes_rain['Pressure3pm'].fillna(yes_rain['Pressure3pm'].median(),inplace=True)
no_rain['Pressure3pm'].fillna(no_rain['Pressure3pm'].median(),inplace=True)

yes_rain['Pressure9am'].fillna(yes_rain['Pressure9am'].median(),inplace=True)
no_rain['Pressure9am'].fillna(no_rain['Pressure9am'].median(),inplace=True)

yes_rain['WindGustDir'].fillna(yes_rain['WindGustDir'].mode()[0],inplace=True)
no_rain['WindGustDir'].fillna(no_rain['WindGustDir'].mode()[0],inplace=True)

yes_rain['WindGustSpeed'].fillna(yes_rain['WindGustSpeed'].median(),inplace=True)
no_rain['WindGustSpeed'].fillna(no_rain['WindGustSpeed'].median(),inplace=True)

yes_rain['WindDir9am'].fillna(yes_rain['WindDir9am'].mode()[0],inplace=True)
no_rain['WindDir9am'].fillna(no_rain['WindDir9am'].mode()[0],inplace=True)

yes_rain['WindDir3pm'].fillna(yes_rain['WindDir3pm'].mode()[0],inplace=True)
no_rain['WindDir3pm'].fillna(no_rain['WindDir3pm'].mode()[0],inplace=True)

yes_rain['WindSpeed3pm'].fillna(yes_rain['WindSpeed3pm'].median(),inplace=True)
no_rain['WindSpeed3pm'].fillna(no_rain['WindSpeed3pm'].median(),inplace=True)

yes_rain['WindSpeed9am'].fillna(yes_rain['WindSpeed9am'].median(),inplace=True)
no_rain['WindSpeed9am'].fillna(no_rain['WindSpeed9am'].median(),inplace=True)



yes_rain['Rainfall'].fillna(yes_rain['Rainfall'].median(),inplace=True)
no_rain['Rainfall'].fillna(no_rain['Rainfall'].median(),inplace=True)
```

In [59]:
```
data= yes_rain.append(no_rain, ignore_index=True)
```

In [60]:
```
data
```

Out[60]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGu
0	2008-12-09	Albury	9.7	31.9	0.0	3.8	4.2	NNW	
1	2008-12-11	Albury	13.4	30.4	0.0	3.8	4.2	N	
2	2008-12-12	Albury	15.9	21.7	2.2	3.8	4.2	NNE	
3	2008-12-13	Albury	15.9	18.6	15.6	3.8	4.2	W	
4	2008-12-17	Albury	14.1	20.9	0.0	3.8	4.2	ENE	
...	...	...	...	...	...	...	...	...	
8181	2017-06-20	Uluru	3.5	21.8	0.0	4.8	9.6	E	
8182	2017-06-21	Uluru	2.8	23.4	0.0	4.8	9.6	E	
8183	2017-06-22	Uluru	3.6	25.3	0.0	4.8	9.6	NNW	
8184	2017-06-23	Uluru	5.4	26.9	0.0	4.8	9.6	N	
8185	2017-06-24	Uluru	7.8	27.0	0.0	4.8	9.6	SE	

8186 rows × 23 columns

In [61]:
```
data.shape
```

Out[61]:
```
(8186, 23)
```

In [ ]:
```

```

In [62]:
```
data.dropna(inplace=True)
```

In [63]:
```
#data['RainTomorrow'].replace('No',0,inplace=True)
#data['RainTomorrow'].replace('Yes',1,inplace=True)

#data['RainToday'].replace('No',0,inplace=True)
#data['RainToday'].replace('Yes',1,inplace=True)
```

In [64]:
```
data.head()
```

Out[64]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSp
0	2008-12-09	Albury	9.7	31.9	0.0	3.8	4.2	NNW	
1	2008-12-11	Albury	13.4	30.4	0.0	3.8	4.2	N	
2	2008-12-12	Albury	15.9	21.7	2.2	3.8	4.2	NNE	
3	2008-12-13	Albury	15.9	18.6	15.6	3.8	4.2	W	
4	2008-12-17	Albury	14.1	20.9	0.0	3.8	4.2	ENE	

5 rows × 23 columns

In [65]:
```
# COnvert Date feature into datetime
data['Date']=pd.to_datetime(data['Date'])
data['year']=data['Date'].dt.year
data['month']=data['Date'].dt.month
data['day']=data['Date'].dt.day
data.drop('Date',axis=1,inplace=True)
```

In [66]:
```
data
```

Out[66]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed
0	Albury	9.7	31.9	0.0	3.8	4.2	NNW	80.0
1	Albury	13.4	30.4	0.0	3.8	4.2	N	30.0
2	Albury	15.9	21.7	2.2	3.8	4.2	NNE	31.0
3	Albury	15.9	18.6	15.6	3.8	4.2	W	61.0
4	Albury	14.1	20.9	0.0	3.8	4.2	ENE	22.0
...	...	...	...	...	...	...	...	..
8181	Uluru	3.5	21.8	0.0	4.8	9.6	E	31.0
8182	Uluru	2.8	23.4	0.0	4.8	9.6	E	31.0
8183	Uluru	3.6	25.3	0.0	4.8	9.6	NNW	22.0
8184	Uluru	5.4	26.9	0.0	4.8	9.6	N	37.0
8185	Uluru	7.8	27.0	0.0	4.8	9.6	SE	28.0

8079 rows × 25 columns

In [67]:
```
d1=data  # will be used for regression problem
```

In [68]:
```
num=[col for col in data.columns if data[col].dtypes!='O']
num
```

Out[68]:
```
['MinTemp',
 'MaxTemp',
 'Rainfall',
 'Evaporation',
 'Sunshine',
 'WindGustSpeed',
 'WindSpeed9am',
 'WindSpeed3pm',
 'Humidity9am',
 'Humidity3pm',
 'Pressure9am',
 'Pressure3pm',
 'Cloud9am',
 'Cloud3pm',
 'Temp9am',
 'Temp3pm',
 'year',
 'month',
 'day']
```

VIF to find multicolinearity between continuous independent features

In [69]:
```
v=data[num]
```

In [70]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
scaled=sc.fit_transform(v)
from statsmodels.stats.outliers_influence import variance_inflation_factor
VIF= pd.DataFrame()
VIF['features']=v.columns
VIF['vif']= [variance_inflation_factor(scaled,i) for i in range(len(v.columns))]
```

In [71]:
```
VIF
```

Out[71]:

	features	vif
0	MinTemp	8.536558
1	MaxTemp	25.650652
2	Rainfall	1.131077
3	Evaporation	1.217936
4	Sunshine	2.137066
5	WindGustSpeed	2.353808
6	WindSpeed9am	2.030998
7	WindSpeed3pm	1.946039
8	Humidity9am	3.854017
9	Humidity3pm	5.579726
10	Pressure9am	18.164613
11	Pressure3pm	17.796774
12	Cloud9am	2.022755
13	Cloud3pm	1.969383
14	Temp9am	17.380827
15	Temp3pm	32.596027
16	year	1.112733
17	month	1.171960
18	day	1.003529

In [72]:
```
num.remove('Temp3pm')
```

In [73]:
```
v=data[num]
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
scaled=sc.fit_transform(v)
VIF= pd.DataFrame()
VIF['features']=v.columns
VIF['vif']= [variance_inflation_factor(scaled,i) for i in range(len(v.columns))]
VIF
```

Out[73]:

	features	vif
0	MinTemp	8.466862
1	MaxTemp	9.254373
2	Rainfall	1.129338
3	Evaporation	1.217434
4	Sunshine	2.137021
5	WindGustSpeed	2.342729
6	WindSpeed9am	2.030727
7	WindSpeed3pm	1.941617
8	Humidity9am	3.374870
9	Humidity3pm	3.788957
10	Pressure9am	17.197244
11	Pressure3pm	16.997036
12	Cloud9am	2.022711
13	Cloud3pm	1.961882
14	Temp9am	16.148474
15	year	1.112539
16	month	1.156103
17	day	1.003338

In [74]:
```
num.remove('Pressure9am')
```

In [75]:
```
v=data[num]
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
scaled=sc.fit_transform(v)
VIF= pd.DataFrame()
VIF['features']=v.columns
VIF['vif']= [variance_inflation_factor(scaled,i) for i in range(len(v.columns))]
VIF
```

Out[75]:

	features	vif
0	MinTemp	8.282180
1	MaxTemp	8.650329
2	Rainfall	1.124425
3	Evaporation	1.216592
4	Sunshine	2.134751
5	WindGustSpeed	2.325879
6	WindSpeed9am	2.025740
7	WindSpeed3pm	1.930832
8	Humidity9am	3.374529
9	Humidity3pm	3.783076
10	Pressure3pm	1.439756
11	Cloud9am	2.020075
12	Cloud3pm	1.956756
13	Temp9am	16.107884
14	year	1.112348
15	month	1.155157
16	day	1.002241

In [76]:
```
num.remove('Temp9am')
```

In [77]:
```
v=data[num]
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
scaled=sc.fit_transform(v)
VIF= pd.DataFrame()
VIF['features']=v.columns
VIF['vif']= [variance_inflation_factor(scaled,i) for i in range(len(v.columns))]
VIF
```

Out[77]:

	features	vif
0	MinTemp	4.222621
1	MaxTemp	4.951979
2	Rainfall	1.121856
3	Evaporation	1.215636
4	Sunshine	2.131929
5	WindGustSpeed	2.291850
6	WindSpeed9am	2.006285
7	WindSpeed3pm	1.879912
8	Humidity9am	2.349810
9	Humidity3pm	2.972785
10	Pressure3pm	1.433960
11	Cloud9am	1.995461
12	Cloud3pm	1.952044
13	year	1.085107
14	month	1.142017
15	day	1.002009

In [78]:
```
data.shape
```

Out[78]: (8079, 25)

In [79]:
```
data.drop(['Temp3pm','Temp9am','Pressure9am'],axis=1,inplace=True)
data.shape
```

Out[79]: (8079, 22)

In [80]:
```
data.head(2)
```

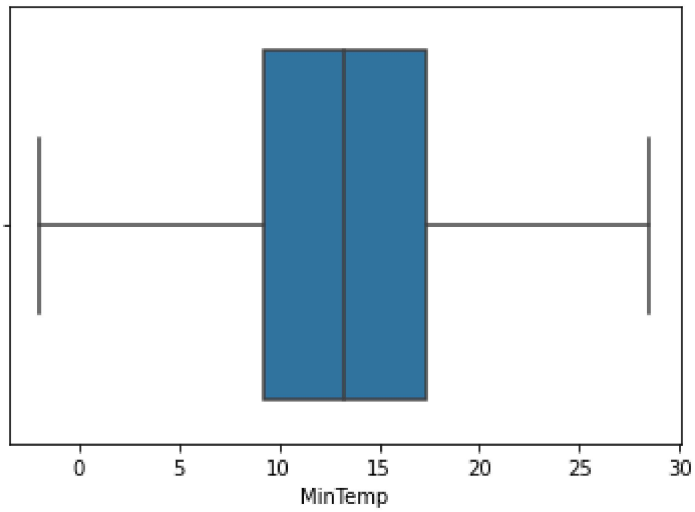
Out[80]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	W
0	Albury	9.7	31.9	0.0	3.8	4.2	NNW	80.0	
1	Albury	13.4	30.4	0.0	3.8	4.2	N	30.0	

2 rows × 22 columns

Outliers

In [81]:
```
for i in num:
    sns.boxplot(data[i])
    plt.show()
```

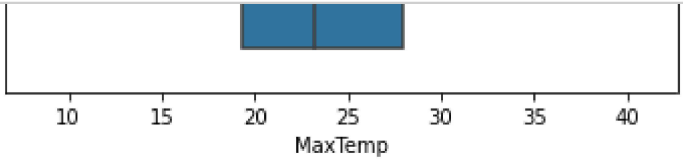


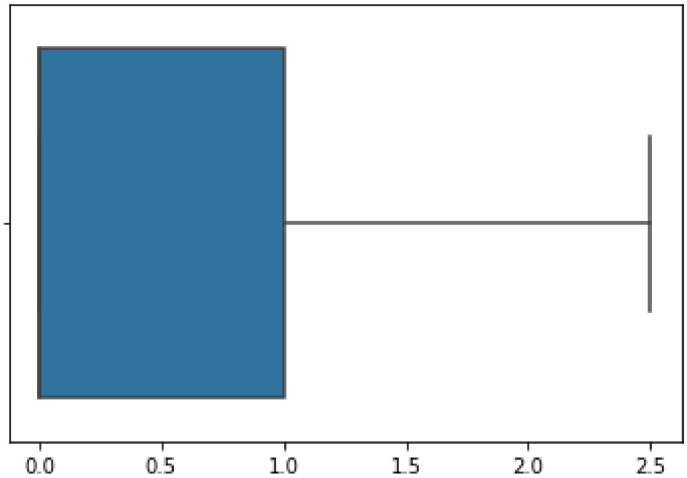
In [82]:
```
data[num].skew()
```

Out[82]:
```
MinTemp          -0.070098
MaxTemp           0.394841
Rainfall         13.073756
Evaporation      13.998671
Sunshine         -0.822526
WindGustSpeed     0.784800
WindSpeed9am      0.949237
WindSpeed3pm      0.509973
Humidity9am      -0.250378
Humidity3pm       0.118933
Pressure3pm      -0.011625
Cloud9am         -0.261462
Cloud3pm         -0.185359
year              0.421639
month             0.054225
day               0.000577
dtype: float64
```

In [83]:
```
for i in num:
    IQR= data[i].quantile(.75)-data[i].quantile(.25)
    lower=data[i].quantile(.25) - (1.5 * IQR)
    upper=data[i].quantile(.75) + (1.5 * IQR)
    data[i]=np.where(data[i]<lower,lower,data[i])
    data[i]=np.where(data[i]>upper,upper,data[i])
```

In [84]:
```
for i in num:
    sns.boxplot(data[i])
    plt.show()
```





In [85]:
```
data[num].skew()
```

Out[85]:
```
MinTemp         -0.070098
MaxTemp          0.372256
Rainfall         1.226519
Evaporation      0.042898
Sunshine        -0.822526
WindGustSpeed    0.556656
WindSpeed9am     0.743355
WindSpeed3pm     0.347722
Humidity9am     -0.235208
Humidity3pm      0.118933
Pressure3pm      0.015165
Cloud9am        -0.261462
Cloud3pm        -0.185359
year             0.421639
month            0.054225
day              0.000577
dtype: float64
```

Transformation

In [86]:
```
from sklearn.preprocessing import power_transform
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
for i in num:
    trans=power_transform(data[num])
    data[i]=sc.fit_transform(trans)
```

In [87]:
```
data.head()
```

Out[87]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed
0	Albury	-0.630013	-0.642992	-0.642992	-0.642992	-0.642992	NNW	-0.642992
1	Albury	0.054150	0.031187	0.031187	0.031187	0.031187	N	0.031187
2	Albury	0.513327	0.498052	0.498052	0.498052	0.498052	NNE	0.498052
3	Albury	0.513327	0.498052	0.498052	0.498052	0.498052	W	0.498052
4	Albury	0.182945	0.161018	0.161018	0.161018	0.161018	ENE	0.161018

5 rows × 22 columns

Encoding

In [88]:
```
data.head()
```

Out[88]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpe
0	Albury	-0.630013	-0.642992	-0.642992	-0.642992	-0.642992	NNW	-0.6429
1	Albury	0.054150	0.031187	0.031187	0.031187	0.031187	N	0.0311
2	Albury	0.513327	0.498052	0.498052	0.498052	0.498052	NNE	0.4980
3	Albury	0.513327	0.498052	0.498052	0.498052	0.498052	W	0.4980
4	Albury	0.182945	0.161018	0.161018	0.161018	0.161018	ENE	0.1610

5 rows × 22 columns

In [89]:
```
chitest=data[['Location','WindGustDir','WindDir9am','WindDir3pm','RainTomorrow']]
chitest
```

Out[89]:

	Location	WindGustDir	WindDir9am	WindDir3pm	RainTomorrow
0	Albury	NNW	SE	NW	Yes
1	Albury	N	SSE	ESE	Yes
2	Albury	NNE	NE	ENE	Yes
3	Albury	W	NNW	NNW	Yes
4	Albury	ENE	SSW	E	Yes
...	...	...	...	...	...
8181	Uluru	E	ESE	E	No
8182	Uluru	E	SE	ENE	No
8183	Uluru	NNW	SE	N	No
8184	Uluru	N	SE	WNW	No
8185	Uluru	SE	SSE	N	No

8079 rows × 5 columns

In [90]:
```
import numpy as np
### let's perform label encoding on WindGustDir
ordinal_label = {k: i for i, k in enumerate(chitest['WindGustDir'].unique(), 0)}
chitest['WindGustDir'] = chitest['WindGustDir'].map(ordinal_label)
ordinal_label = {k: i for i, k in enumerate(chitest['Location'].unique(), 0)}
chitest['Location'] = chitest['Location'].map(ordinal_label)
ordinal_label = {k: i for i, k in enumerate(chitest['WindDir9am'].unique(), 0)}
chitest['WindDir9am'] = chitest['WindDir9am'].map(ordinal_label)
ordinal_label = {k: i for i, k in enumerate(chitest['WindDir3pm'].unique(), 0)}
chitest['WindDir3pm'] = chitest['WindDir3pm'].map(ordinal_label)
chitest['RainTomorrow'].replace('No',0,inplace=True)
chitest['RainTomorrow'].replace('Yes',1,inplace=True)
```

In [91]:
```
chitest
```

Out[91]:

	Location	WindGustDir	WindDir9am	WindDir3pm	RainTomorrow
0	0	0	0	0	1
1	0	1	1	1	1
2	0	2	2	2	1
3	0	3	3	3	1
4	0	4	4	4	1
...	...	...	...	...	...
8181	11	12	7	4	0
8182	11	12	0	2	0
8183	11	0	0	9	0
8184	11	1	0	5	0
8185	11	7	1	9	0

8079 rows × 5 columns

In [92]:
```
inp=chitest.drop('RainTomorrow',axis=1)
out=chitest['RainTomorrow']
## Perform chi2 test
### chi2 returns 2 values
### Fscore and the pvalue
from sklearn.feature_selection import chi2
f_p_values=chi2(inp,out)
```

In [93]:
```
f_p_values
```

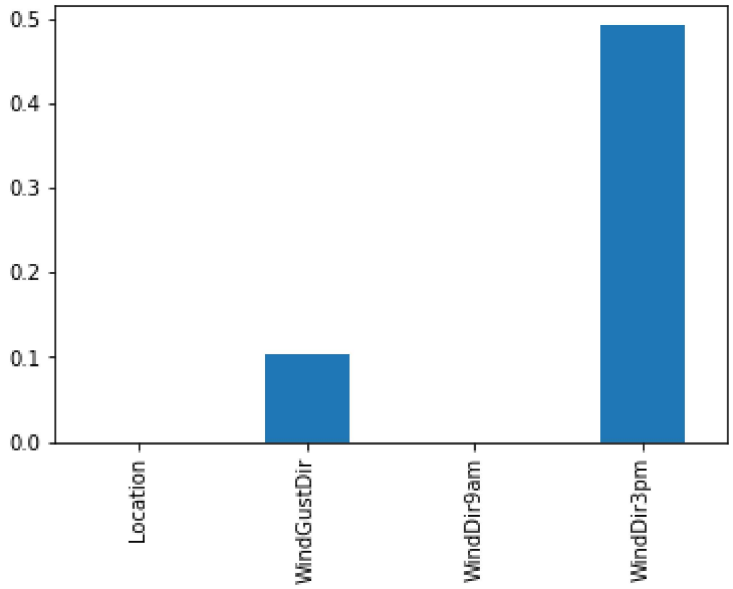
Out[93]:
```
(array([34.02791176,  2.63753342, 20.16354411,  0.47166443]),
 array([5.43271338e-09, 1.04365465e-01, 7.10950475e-06, 4.92222447e-01]))
```

In [94]:
```
p_values=pd.Series(f_p_values[1])
p_values.index=inp.columns
p_values
```

Out[94]:
```
Location       5.432713e-09
WindGustDir    1.043655e-01
WindDir9am     7.109505e-06
WindDir3pm     4.922224e-01
dtype: float64
```

In [95]:
```
p_values.plot.bar()
```

Out[95]: <AxesSubplot:>



In [96]:
```
16-1 * (2-1 )
```

Out[96]: 15

In [97]:
```
data.drop('WindDir3pm',axis=1)
```

Out[97]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpe
0	Albury	-0.630013	-0.642992	-0.642992	-0.642992	-0.642992	NNW	-0.6429
1	Albury	0.054150	0.031187	0.031187	0.031187	0.031187	N	0.0311
2	Albury	0.513327	0.498052	0.498052	0.498052	0.498052	NNE	0.4980
3	Albury	0.513327	0.498052	0.498052	0.498052	0.498052	W	0.4980
4	Albury	0.182945	0.161018	0.161018	0.161018	0.161018	ENE	0.1610
...	...	...	...	...	...	...	...	
8181	Uluru	-1.794801	-1.751381	-1.751381	-1.751381	-1.751381	E	-1.7513
8182	Uluru	-1.928608	-1.876512	-1.876512	-1.876512	-1.876512	E	-1.8765
8183	Uluru	-1.775739	-1.733525	-1.733525	-1.733525	-1.733525	NNW	-1.7335
8184	Uluru	-1.434531	-1.412489	-1.412489	-1.412489	-1.412489	N	-1.4124
8185	Uluru	-0.983996	-0.984025	-0.984025	-0.984025	-0.984025	SE	-0.9840

8079 rows × 21 columns

Encoding

In [98]:
```
data['RainTomorrow'].replace('No',0,inplace=True)
data['RainTomorrow'].replace('Yes',1,inplace=True)

data['RainToday'].replace('No',0,inplace=True)
data['RainToday'].replace('Yes',1,inplace=True)
```

In [99]:
```
data.head()
```

Out[99]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed
0	Albury	-0.630013	-0.642992	-0.642992	-0.642992	-0.642992	NNW	-0.642992
1	Albury	0.054150	0.031187	0.031187	0.031187	0.031187	N	0.031187
2	Albury	0.513327	0.498052	0.498052	0.498052	0.498052	NNE	0.498052
3	Albury	0.513327	0.498052	0.498052	0.498052	0.498052	W	0.498052
4	Albury	0.182945	0.161018	0.161018	0.161018	0.161018	ENE	0.161018

5 rows × 22 columns

In [100]:
```
X=data.drop('RainTomorrow',axis=1)
Y=data['RainTomorrow']
X.shape  , Y.shape
```

Out[100]: ((8079, 21), (8079,))

In [101]:
```
X=pd.get_dummies(X,drop_first=True)
X.shape , Y.shape
```

Out[101]: ((8079, 73), (8079,))

SMOTE- Balancing dataset

In [102]:
```
from imblearn.over_sampling import SMOTE
sm=SMOTE()
x,y=sm.fit_resample(X,Y)
x.shape , y.shape
```

Out[102]: ((12310, 73), (12310,))

Machine Learning

In [103]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
# FInd best Random_state

maxaccu=0
maxRS=0

for i in range(0,200):
    x_train,x_test,y_train,y_test= train_test_split(x,y,random_state=i,test_size=
    LR= LogisticRegression()
    LR.fit(x_train,y_train)
    pred= LR.predict(x_test)
    acc=accuracy_score(y_test,pred)
    if acc>maxaccu:
        maxaccu=acc
        maxRS=i
print("Best accuracy is ",maxaccu,"on Random State =",maxRS)
```

```
Best accuracy is  0.7714595180070404 on Random State = 52
```

In [104]:
```
# Use, Random state= 122
x_train,x_test,y_train,y_test= train_test_split(x,y,random_state=122,test_size=.
```

In [105]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import RidgeClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

In [106]:
```
LR_model= LogisticRegression()
RD_model= RidgeClassifier()
DT_model= DecisionTreeClassifier()
SV_model= SVC()
KNR_model= KNeighborsClassifier()
RFR_model= RandomForestClassifier()
XGB_model= XGBClassifier()
SGH_model= SGDClassifier()
Bag_model=BaggingClassifier()
ADA_model=AdaBoostClassifier()
GB_model= GradientBoostingClassifier()

model=[LR_model,RD_model,DT_model,SV_model,KNR_model,RFR_model,XGB_model,SGH_mode
```

In [107]:
```
accuracy=[]
f1=[]

for m in model:
    m.fit(x_train,y_train)
    m.score(x_train,y_train)
    pred= m.predict(x_test)
    accuracy.append(round(accuracy_score(y_test,pred) * 100, 2))
    f1.append(round(f1_score(y_test,pred) * 100, 2))
    #print('Accuracy_Score of ',m, 'is', accuracy_score(y_test,pred)*100)
    #print("F1 SCore                ", f1_score(y_test,pred)*100)
    #print('Confusion Matrix of ',m,' is \n', confusion_matrix(y_test,pred) )
    #print(classification_report(y_test,pred))
    #print('*'*50)

pd.DataFrame({'Model':model,'Accuracy':accuracy,'F1 Score':f1})
```

Out[107]:

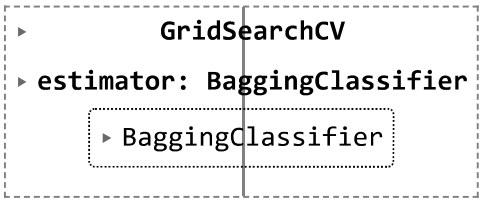
	Model	Accuracy	F1 Score
0	LogisticRegression()	76.63	75.91
1	RidgeClassifier()	76.36	75.02
2	DecisionTreeClassifier()	81.51	81.96
3	SVC()	79.83	79.55
4	KNeighborsClassifier()	76.74	80.14
5	(DecisionTreeClassifier(max_features='sqrt', r...	84.70	84.53
6	XGBClassifier(base_score=0.5, booster='gbtree'...	84.40	84.87
7	SGDClassifier()	74.93	73.00
8	(DecisionTreeClassifier(random_state=186900789...	84.40	84.31
9	(DecisionTreeClassifier(max_depth=1, random_st...	72.92	72.31
10	([DecisionTreeRegressor(criterion='friedman_ms...	74.30	74.01

In [108]:
```
params = {'n_estimators' : [100,150,200,300,500],
    'max_features' : [1, 2, 3, 4, 5],
    'max_samples' : [0.05, 0.1, 0.2, 0.5]
}
```

In [109]:
```
from sklearn.model_selection import GridSearchCV
GCV=GridSearchCV(Bag_model,param_grid=params,cv=5,n_jobs=-1,verbose=2)
GCV.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

Out[109]:



In [111]:
```
GCV.best_estimator_
```

Out[111]:
```
BaggingClassifier(max_features=5, max_samples=0.5, n_estimators=150)
```

In [112]:
```
GCV.best_params_
```

Out[112]:
```
{'max_features': 5, 'max_samples': 0.5, 'n_estimators': 150}
```

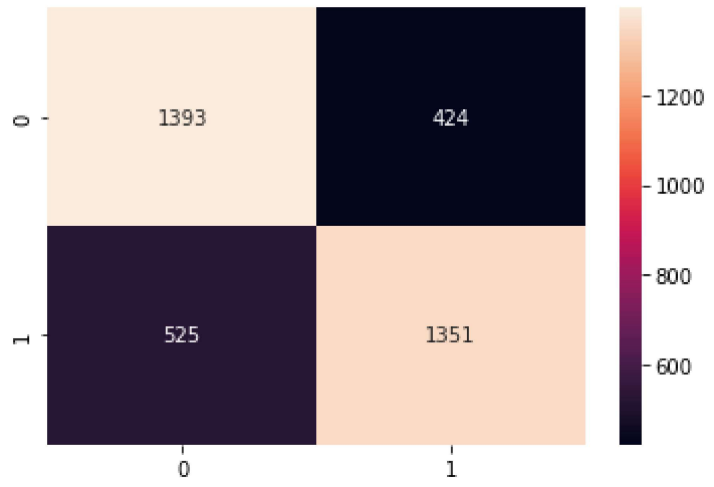
In [113]:
```
GCV_pred=GCV.best_estimator_.predict(x_test)
accuracy_score(y_test,GCV_pred)
```

Out[113]:
```
0.7733549959382616
```

Confusion Matrix

In [114]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,pred)
sns.heatmap(confusion_matrix(y_test,pred),annot=True, fmt='d')
```

Out[114]:
```
<AxesSubplot:>
```

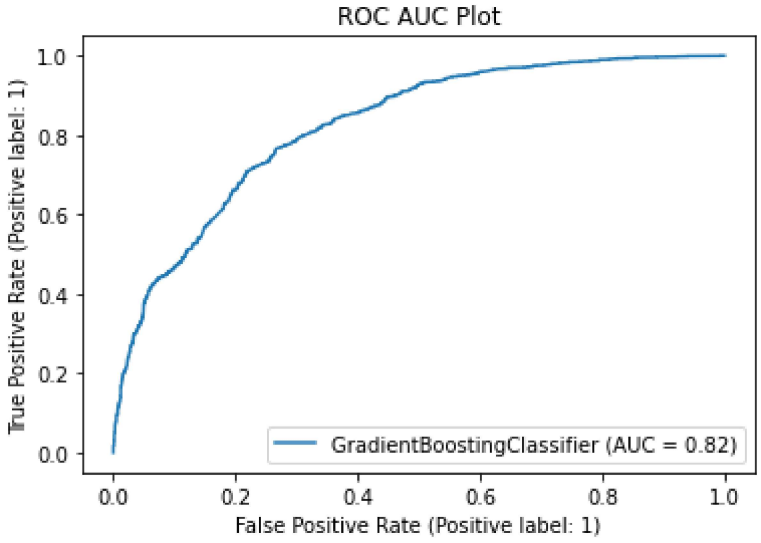


AUC ROC plot

In [115]:
```
from sklearn.metrics import roc_auc_score,roc_curve,plot_roc_curve
```

In [116]:
```
plot_roc_curve(GB_model,x_test,y_test)
plt.title('ROC AUC Plot')
```

Out[116]:
```
Text(0.5, 1.0, 'ROC AUC Plot')
```



Saving Model

In [117]:
```
import joblib
joblib.dump(GB_model,"Rainfall_Prediction.pkl")
```

Out[117]:
```
['Rainfall_Prediction.pkl']
```

Predict the Rainfall in MM. Regression Model

In [118]:
```
d1.head()
```

Out[118]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed
0	Albury	-0.630013	-0.642992	-0.642992	-0.642992	-0.642992	NNW	-0.642992
1	Albury	0.054150	0.031187	0.031187	0.031187	0.031187	N	0.031187
2	Albury	0.513327	0.498052	0.498052	0.498052	0.498052	NNE	0.498052
3	Albury	0.513327	0.498052	0.498052	0.498052	0.498052	W	0.498052
4	Albury	0.182945	0.161018	0.161018	0.161018	0.161018	ENE	0.161018

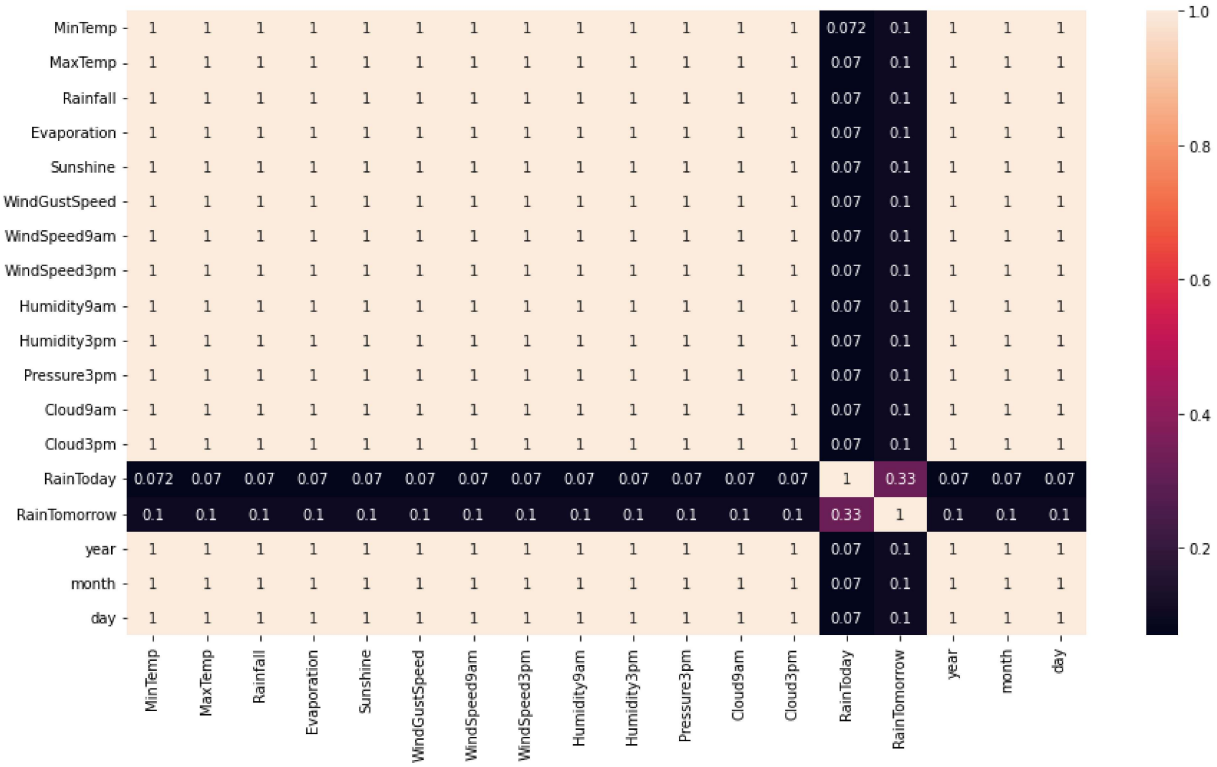
5 rows × 22 columns

In [119]:
```
round(d1.isnull().sum()/d1.shape[0]*100,2).sort_values(ascending=False)
```

Out[119]:
```
Location         0.0
MinTemp          0.0
month            0.0
year             0.0
RainTomorrow     0.0
RainToday        0.0
Cloud3pm         0.0
Cloud9am         0.0
Pressure3pm      0.0
Humidity3pm      0.0
Humidity9am      0.0
WindSpeed3pm     0.0
WindSpeed9am     0.0
WindDir3pm       0.0
WindDir9am       0.0
WindGustSpeed    0.0
WindGustDir      0.0
Sunshine         0.0
Evaporation      0.0
Rainfall         0.0
MaxTemp          0.0
day              0.0
dtype: float64
```

In [120]:
```
plt.figure(figsize=(15,8))
sns.heatmap(d1.corr(),annot=True)
```

Out[120]: <AxesSubplot:>



In [121]:
```
cont= [i for i in d1.columns if d1[i].dtypes!='O']
cont.remove('Rainfall')
cont
```

Out[121]:
```
['MinTemp',
 'MaxTemp',
 'Evaporation',
 'Sunshine',
 'WindGustSpeed',
 'WindSpeed9am',
 'WindSpeed3pm',
 'Humidity9am',
 'Humidity3pm',
 'Pressure3pm',
 'Cloud9am',
 'Cloud3pm',
 'RainToday',
 'RainTomorrow',
 'year',
 'month',
 'day']
```

In [122]:
```
v=d1[cont]
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
scaled=sc.fit_transform(v)
VIF= pd.DataFrame()
VIF['features']=v.columns
VIF['vif']= [variance_inflation_factor(scaled,i) for i in range(len(v.columns))]
VIF
```

Out[122]:

	features	vif
0	MinTemp	1627.956071
1	MaxTemp	inf
2	Evaporation	inf
3	Sunshine	inf
4	WindGustSpeed	inf
5	WindSpeed9am	inf
6	WindSpeed3pm	inf
7	Humidity9am	inf
8	Humidity3pm	inf
9	Pressure3pm	inf
10	Cloud9am	inf
11	Cloud3pm	inf
12	RainToday	1.129154
13	RainTomorrow	1.126224
14	year	inf
15	month	inf
16	day	inf

In [123]:
```
cont.remove('MaxTemp')
v=d1[cont]
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
scaled=sc.fit_transform(v)
VIF= pd.DataFrame()
VIF['features']=v.columns
VIF['vif']= [variance_inflation_factor(scaled,i) for i in range(len(v.columns))]
VIF
```

Out[123]:

	features	vif
0	MinTemp	1627.956071
1	Evaporation	inf
2	Sunshine	inf
3	WindGustSpeed	inf
4	WindSpeed9am	inf
5	WindSpeed3pm	inf
6	Humidity9am	inf
7	Humidity3pm	inf
8	Pressure3pm	inf
9	Cloud9am	inf
10	Cloud3pm	inf
11	RainToday	1.129154
12	RainTomorrow	1.126224
13	year	inf
14	month	inf
15	day	inf

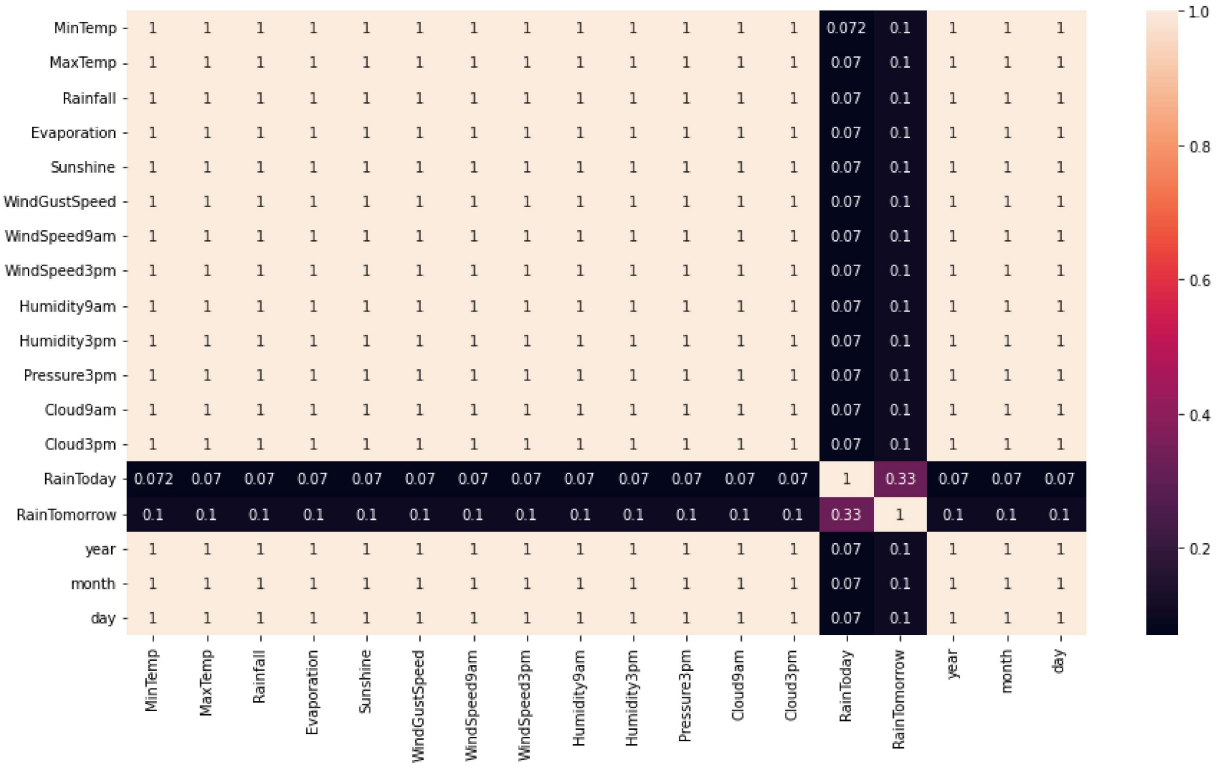
In [126]:
```
v=d1[cont]
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
scaled=sc.fit_transform(v)
VIF= pd.DataFrame()
VIF['features']=v.columns
VIF['vif']= [variance_inflation_factor(scaled,i) for i in range(len(v.columns))]
VIF
```

Out[126]:

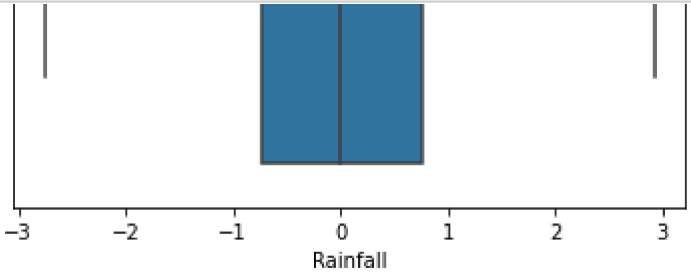
	features	vif
0	MinTemp	1627.956071
1	Evaporation	inf
2	Sunshine	inf
3	WindGustSpeed	inf
4	WindSpeed9am	inf
5	WindSpeed3pm	inf
6	Humidity9am	inf
7	Humidity3pm	inf
8	Pressure3pm	inf
9	Cloud9am	inf
10	Cloud3pm	inf
11	RainToday	1.129154
12	RainTomorrow	1.126224
13	year	inf
14	month	inf
15	day	inf

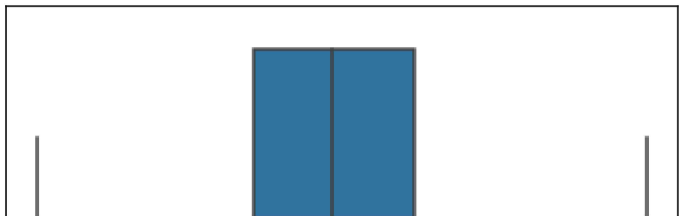
In [129]:
```
plt.figure(figsize=(15,8))
sns.heatmap(d1.corr(),annot=True)
```

Out[129]: <AxesSubplot:>



In [130]:
```
cont= [i for i in X.columns if X[i].dtypes!='O']
for i in cont:
    sns.boxplot(X[i])
    plt.show()
```





In [131]:
```
X[cont].skew()
```

Out[131]:
```
MinTemp          -0.099928
MaxTemp          -0.010778
Rainfall         -0.010778
Evaporation      -0.010778
Sunshine         -0.010778
                    ...
WindDir3pm_SSW    4.472660
WindDir3pm_SW     3.761353
WindDir3pm_W      3.890760
WindDir3pm_WNW    3.790313
WindDir3pm_WSW    3.395764
Length: 73, dtype: float64
```

In [132]:
```
for i in cont:
    IQR= X[i].quantile(.75)-X[i].quantile(.25)
    lower=X[i].quantile(.25) - (1.5 * IQR)
    upper=X[i].quantile(.75) + (1.5 * IQR)
    X[i]=np.where(X[i]<lower,lower,X[i])
    X[i]=np.where(X[i]>upper,upper,X[i])
```

In [133]:
```
X[cont].skew()
```

Out[133]:
```
MinTemp          -0.099928
MaxTemp          -0.010778
Rainfall         -0.010778
Evaporation      -0.010778
Sunshine         -0.010778
                    ...
WindDir3pm_SSW    0.000000
WindDir3pm_SW     0.000000
WindDir3pm_W      0.000000
WindDir3pm_WNW    0.000000
WindDir3pm_WSW    0.000000
Length: 73, dtype: float64
```

In [134]:
```
from sklearn.preprocessing import power_transform
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

In [136]:
```
# Its a Regression problem coz output variable have continuous data, sales predic
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error

LR= LinearRegression()
```

In [137]:
```
# Train Test Split / finding best random_state for my model

for i in range(0,200):
    x_train,x_test,y_train,y_test= train_test_split(X,Y,random_state=i,test_size=
    LR.fit(x_train,y_train)
    train_pred=LR.predict(x_train)
    test_pred=LR.predict(x_test)
    if round(r2_score(y_test,test_pred),2)==round(r2_score(y_train,train_pred),2)
        print("At random state ", i, "The model performance very well")
        print("At random state: ",i)
        print("Test R2 score is: ", round(r2_score(y_test,test_pred),2))
        print('Train R2 score is: ', round(r2_score(y_train,train_pred),2))
        print('X'*50,'\n')
```

```
At random state  1 The model performance very well
At random state:  1
Test R2 score is:  0.01
Train R2 score is:  0.01
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

At random state  3 The model performance very well
At random state:  3
Test R2 score is:  0.01
Train R2 score is:  0.01
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

At random state  4 The model performance very well
At random state:  4
Test R2 score is:  0.01
Train R2 score is:  0.01
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

At random state  5 The model performance very well
```

In [138]:
```
# Select Random state= 23

x_train,x_test,y_train,y_test= train_test_split(X,Y,random_state=23,test_size=.3)
LR.fit(x_train,y_train)
```

Out[138]:
```
LinearRegression
LinearRegression()
```

In [139]:
```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import SGDRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

In [140]:
```
l= LinearRegression()
l= Ridge()
l= Lasso()
l= DecisionTreeRegressor()
l= SVR()
el= KNeighborsRegressor()
el= RandomForestRegressor()
el= XGBRegressor()
_model= ElasticNet()
el= SGDRegressor()
el=BaggingRegressor()
el=AdaBoostRegressor()
l= GradientBoostingRegressor()

LR_model,RD_model,LS_model,DT_model,SV_model,KNR_model,RFR_model,XGB_model,Elasti
```

In [141]:
```
for m in model:
    m.fit(x_train,y_train)
    print('mean_absolute_error of ',m ,'model', mean_absolute_error(y_test,m.pred
    print('mean_square_error of',m,'model' , mean_squared_error(y_test,m.predict(
    print('R2 Score of',m,'model', r2_score(y_test,m.predict(x_test) )*100)
    print('X' * 50, '\n\n')
```

```
mean_absolute_error of  LinearRegression() model 0.3587030923946415
mean_square_error of LinearRegression() model 0.17908789214608228
R2 Score of LinearRegression() model 0.546532809243061
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX


mean_absolute_error of  Ridge() model 0.358640705138443
mean_square_error of Ridge() model 0.17905587906464326
R2 Score of Ridge() model 0.5643107388753621
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX


mean_absolute_error of  Lasso() model 0.3620989486216527
mean_square_error of Lasso() model 0.1800857071297685
R2 Score of Lasso() model -0.007587062029634772
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX


mean_absolute_error of  DecisionTreeRegressor() model 0.353256806077165
mean_square_error of DecisionTreeRegressor() model 0.18562852269983737
R2 Score of DecisionTreeRegressor() model -3.0856970327059363
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX


mean_absolute_error of  SVR() model 0.2884110042481236
mean_square_error of SVR() model 0.19846409740337087
R2 Score of SVR() model -10.213718879158296
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX


mean_absolute_error of  KNeighborsRegressor() model 0.3614686468646865
mean_square_error of KNeighborsRegressor() model 0.21011551155115513
R2 Score of KNeighborsRegressor() model -16.68413695592794
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX


mean_absolute_error of  RandomForestRegressor() model 0.35314829925868246
mean_square_error of RandomForestRegressor() model 0.18494924254598674
R2 Score of RandomForestRegressor() model -2.7084701544130674
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX


mean_absolute_error of  XGBRegressor(base_score=0.5, booster='gbtree', callback
s=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
```

```
              missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=
0,
              num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=
0,
              reg_lambda=1, ...) model 0.3535692013411901
mean_square_error of XGBRegressor(base_score=0.5, booster='gbtree', callbacks=N
one,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=
0,
              num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=
0,
              reg_lambda=1, ...) model 0.18466355087698907
R2 Score of XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=
0,
              num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=
0,
              reg_lambda=1, ...) model -2.5498160617837717
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX


mean_absolute_error of  ElasticNet() model 0.3620989486216527
mean_square_error of ElasticNet() model 0.1800857071297685
R2 Score of ElasticNet() model -0.007587062029634772
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX


mean_absolute_error of  SGDRegressor() model 0.3590360852200712
mean_square_error of SGDRegressor() model 0.18009651546533975
R2 Score of SGDRegressor() model -0.013589290512183716
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX


mean_absolute_error of  BaggingRegressor() model 0.3514100110688451
mean_square_error of BaggingRegressor() model 0.18585931172431067
R2 Score of BaggingRegressor() model -3.213861859475564
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX


mean_absolute_error of  AdaBoostRegressor() model 0.39150746773569917
mean_square_error of AdaBoostRegressor() model 0.18310747771698985
R2 Score of AdaBoostRegressor() model -1.6856768443867187
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
mean_absolute_error of  GradientBoostingRegressor() model 0.3570524346284103
mean_square_error of GradientBoostingRegressor() model 0.17961154133069313
R2 Score of GradientBoostingRegressor() model 0.25573298812126
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Cross Validation

In [142]:
```
from sklearn.model_selection import cross_val_score
for i in model:
    print('Root mean_square of ',i, 'model',np.sqrt(mean_squared_error(y_test,i.p
    score=cross_val_score(i,X,Y,cv=10, scoring='neg_mean_squared_error').mean()
    print("cross Validation score of root mean square ",i ," is ",np.sqrt(-score)
    print('*'*50)
```

```
Root mean_square of  LinearRegression() model 0.42318777409807373
cross Validation score of root mean square  LinearRegression()  is  0.467752803
64971805
**************************************************
Root mean_square of  Ridge() model 0.42314994867616756
cross Validation score of root mean square  Ridge()  is  0.4675678316256257
**************************************************
Root mean_square of  Lasso() model 0.4243650635122648
cross Validation score of root mean square  Lasso()  is  0.4673793959657305
**************************************************
Root mean_square of  DecisionTreeRegressor() model 0.4308462866264921
cross Validation score of root mean square  DecisionTreeRegressor()  is  0.4794
5960415109223
**************************************************
Root mean_square of  SVR() model 0.44549309467529447
cross Validation score of root mean square  SVR()  is  0.44785439670001337
**************************************************
Root mean_square of  KNeighborsRegressor() model 0.4583835856039733
cross Validation score of root mean square  KNeighborsRegressor()  is  0.808159
9955632837
**************************************************
Root mean_square of  RandomForestRegressor() model 0.43005725496262326
cross Validation score of root mean square  RandomForestRegressor()  is  0.4787
2077323203266
**************************************************
Root mean_square of  XGBRegressor(base_score=0.5, booster='gbtree', callbacks=N
one,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
             missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=
0,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=
0,
             reg_lambda=1, ...) model 0.4297249712048266
cross Validation score of root mean square  XGBRegressor(base_score=0.5, booste
r='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
             importance_type=None, interaction_constraints='',
             learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
             missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=
0,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=
0,
```

```
             reg_lambda=1, ...)  is  0.47802622390409
************************************************
Root mean_square of  ElasticNet() model 0.4243650635122648
cross Validation score of root mean square  ElasticNet()  is  0.467379395965730
5
************************************************
Root mean_square of  SGDRegressor() model 0.42437779803535874
cross Validation score of root mean square  SGDRegressor()  is  0.4648564707477
881
************************************************
Root mean_square of  BaggingRegressor() model 0.43111403563826434
cross Validation score of root mean square  BaggingRegressor()  is  0.478712903
96164096
************************************************
Root mean_square of  AdaBoostRegressor() model 0.42791059547175253
cross Validation score of root mean square  AdaBoostRegressor()  is  0.46301671
73064505
************************************************
Root mean_square of  GradientBoostingRegressor() model 0.42380601851636457
cross Validation score of root mean square  GradientBoostingRegressor()  is  0.
4725535138941103
************************************************
```

Hyper tuning of Random Forest

In [143]:
```
n_estimator= [200, 400, 600, 800, 1000]  # no of tree in Random forest, default 1
max_features= ['auto','sqrt','log2']  # mini no of features to create Decission T
max_depth=[10, 64, 118, 173, 227, 282, 336]  #  Max depth of decision tree
min_samples_split= [1,2,3]   # mini no of sample rerquired to split node
min_samples_leaf= [1,3,4,6,7,9]  #mini no of sample required at each leaf node

param_grid= {'n_estimators': n_estimator,
             'max_features':max_features,
             'max_depth':max_depth,
             'min_samples_split':min_samples_split,
             'min_samples_leaf':min_samples_leaf,
             }
param_grid
```

Out[143]:
```
{'n_estimators': [200, 400, 600, 800, 1000],
 'max_features': ['auto', 'sqrt', 'log2'],
 'max_depth': [10, 64, 118, 173, 227, 282, 336],
 'min_samples_split': [1, 2, 3],
 'min_samples_leaf': [1, 3, 4, 6, 7, 9]}
```

In [144]:
```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

In [145]:
```
grid_search=RandomizedSearchCV(estimator=RFR_model,param_distributions=param_gric
grid_search.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[145]:

RandomizedSearchCV
estimator: RandomForestRegressor
RandomForestRegressor

In [146]:
```
grid_search.best_estimator_
```

Out[146]:

RandomForestRegressor

```
RandomForestRegressor(max_depth=10, max_features='auto', min_samples_leaf=7,
                      n_estimators=1000)
```

In [147]:
```
grid_search.best_params_
```

Out[147]:
```
{'n_estimators': 1000,
 'min_samples_split': 2,
 'min_samples_leaf': 7,
 'max_features': 'auto',
 'max_depth': 10}
```

Saving model for future use

In [149]:
```
import joblib
joblib.dump(grid_search.best_estimator_,"Rainfall_Prediction_in_mm.pkl")
```

Out[149]:
```
['Rainfall_Prediction_in_mm.pkl']
```

In [ ]:
```

```