Import Libraries, Dataset and Explore

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('weatherAUS.csv')
df.shape
```

Out[2]: (142193, 24)

In [3]:
```
df.head()
```

Out[3]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity3pm	P
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	...	22.0	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	25.0	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	...	30.0	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	...	16.0	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	...	33.0	

5 rows × 24 columns

In [4]:
```
col_names = df.columns

col_names
```

Out[4]:
```
Index(['Date', 'Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation',
       'Sunshine', 'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm',
       'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
       'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am',
       'Temp3pm', 'RainToday', 'RISK_MM', 'RainTomorrow'],
      dtype='object')
```

In [5]:
```
df.drop(['RISK_MM'], axis=1, inplace=True)
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 142193 entries, 0 to 142192
Data columns (total 23 columns):
Date             142193 non-null object
Location         142193 non-null object
MinTemp          141556 non-null float64
MaxTemp          141871 non-null float64
Rainfall         140787 non-null float64
Evaporation      81350 non-null float64
Sunshine         74377 non-null float64
WindGustDir      132863 non-null object
WindGustSpeed    132923 non-null float64
WindDir9am       132180 non-null object
WindDir3pm       138415 non-null object
WindSpeed9am     140845 non-null float64
WindSpeed3pm     139563 non-null float64
Humidity9am      140419 non-null float64
Humidity3pm      138583 non-null float64
Pressure9am      128179 non-null float64
Pressure3pm      128212 non-null float64
Cloud9am         88536 non-null float64
Cloud3pm         85099 non-null float64
Temp9am          141289 non-null float64
Temp3pm          139467 non-null float64
RainToday        140787 non-null object
RainTomorrow     142193 non-null object
dtypes: float64(16), object(7)
memory usage: 25.0+ MB
```

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

Out[7]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humic
count	141556.000000	141871.000000	140787.000000	81350.000000	74377.000000	132923.000000	140845.000000	139563.000000	14041
mean	12.186400	23.226784	2.349974	5.469824	7.624853	39.984292	14.001988	18.637576	6
std	6.403283	7.117618	8.465173	4.188537	3.781525	13.588801	8.893337	8.803345	1
min	-8.500000	-4.800000	0.000000	0.000000	0.000000	6.000000	0.000000	0.000000	
25%	7.600000	17.900000	0.000000	2.600000	4.900000	31.000000	7.000000	13.000000	5
50%	12.000000	22.600000	0.000000	4.800000	8.500000	39.000000	13.000000	19.000000	7
75%	16.800000	28.200000	0.800000	7.400000	10.600000	48.000000	19.000000	24.000000	8
max	33.900000	48.100000	371.000000	145.000000	14.500000	135.000000	130.000000	87.000000	10

Univariate Analysis

In [8]:
```
df['RainTomorrow'].isnull().sum()
```

Out[8]:
```
0
```

In [9]:
```
df['RainTomorrow'].nunique()
```

Out[9]:
```
2
```

In [10]:
```
df['RainTomorrow'].unique()
```

Out[10]:
```
array(['No', 'Yes'], dtype=object)
```

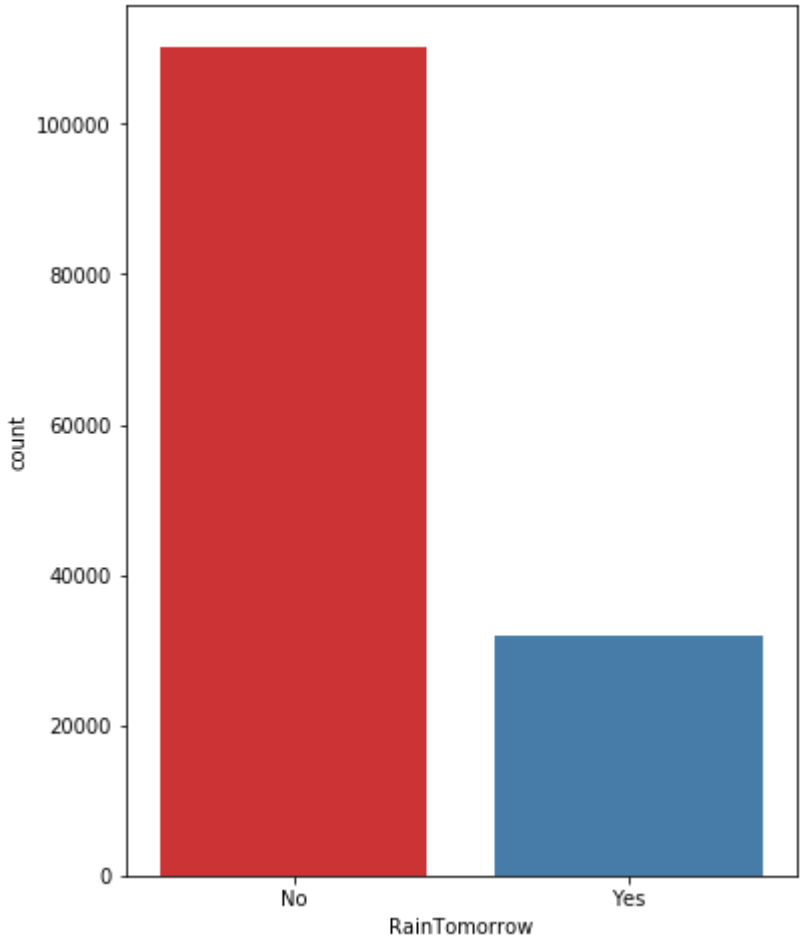
In [11]:
```
df['RainTomorrow'].value_counts()
```

Out[11]:
```
No     110316
Yes     31877
Name: RainTomorrow, dtype: int64
```

In [12]:
```
df['RainTomorrow'].value_counts()/len(df)
```

Out[12]:
```
No     0.775819
Yes    0.224181
Name: RainTomorrow, dtype: float64
```

In [13]:
```
f, ax = plt.subplots(figsize=(6, 8))
ax = sns.countplot(x="RainTomorrow", data=df, palette="Set1")
plt.show()
```



Bivariate Analysis

In [14]:
```
categorical = [var for var in df.columns if df[var].dtype=='O']

print('There are {} categorical variables\n'.format(len(categorical)))

print('The categorical variables are :', categorical)
```

```
There are 7 categorical variables

The categorical variables are : ['Date', 'Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday', 'R
ainTomorrow']
```

In [15]:
```
# view the categorical variables

df[categorical].head()
```

Out[15]:

	Date	Location	WindGustDir	WindDir9am	WindDir3pm	RainToday	RainTomorrow
0	2008-12-01	Albury	W	W	WNW	No	No
1	2008-12-02	Albury	WNW	NNW	WSW	No	No
2	2008-12-03	Albury	WSW	W	WSW	No	No
3	2008-12-04	Albury	NE	SE	E	No	No
4	2008-12-05	Albury	W	ENE	NW	No	No

In [16]:
```
# check missing values in categorical variables

df[categorical].isnull().sum()
```

Out[16]:
```
Date                0
Location            0
WindGustDir      9330
WindDir9am      10013
WindDir3pm       3778
RainToday        1406
RainTomorrow        0
dtype: int64
```

In [17]:
```
# print categorical variables containing missing values

cat1 = [var for var in categorical if df[var].isnull().sum()!=0]

print(df[cat1].isnull().sum())
```

```
WindGustDir     9330
WindDir9am     10013
WindDir3pm      3778
RainToday       1406
dtype: int64
```

In [18]:
```
# view frequency of categorical variables
for var in categorical:

    print(df[var].value_counts())
```

```
2013-04-28    49
2014-04-14    49
2013-11-06    49
2013-06-23    49
2013-12-02    49
2016-05-09    49
2016-09-20    49
2017-01-16    49
2017-01-05    49
2016-06-30    49
2016-10-04    49
2013-08-19    49
2017-01-03    49
2014-02-17    49
2014-10-05    49
2016-05-14    49
2013-11-20    49
2014-02-01    49
2014-01-19    49
```

In [19]:
```
# view frequency distribution of categorical variables

for var in categorical:

    print(df[var].value_counts()/np.float(len(df)))
```

```
2013-04-28    0.000345
2014-04-14    0.000345
2013-11-06    0.000345
2013-06-23    0.000345
2013-12-02    0.000345
2016-05-09    0.000345
2016-09-20    0.000345
2017-01-16    0.000345
2017-01-05    0.000345
2016-06-30    0.000345
2016-10-04    0.000345
2013-08-19    0.000345
2017-01-03    0.000345
2014-02-17    0.000345
2014-10-05    0.000345
2016-05-14    0.000345
2013-11-20    0.000345
2014-02-01    0.000345
2014-01-19    0.000345
```

In [20]:
```
# check for cardinality in categorical variables

for var in categorical:

    print(var, ' contains ', len(df[var].unique()), ' labels')
```

```
Date  contains  3436  labels
Location  contains  49  labels
WindGustDir  contains  17  labels
WindDir9am  contains  17  labels
WindDir3pm  contains  17  labels
RainToday  contains  3  labels
RainTomorrow  contains  2  labels
```

In [21]:
```
df['Date'].dtypes
```

Out[21]:
```
dtype('O')
```

In [22]:
```
# parse the dates, currently coded as strings, into datetime format

df['Date'] = pd.to_datetime(df['Date'])

# extract year from date

df['Year'] = df['Date'].dt.year

df['Year'].head()
```

Out[22]:
```
0    2008
1    2008
2    2008
3    2008
4    2008
Name: Year, dtype: int64
```

In [23]:
```
# extract month from date

df['Month'] = df['Date'].dt.month

df['Month'].head()
```

Out[23]:
```
0    12
1    12
2    12
3    12
4    12
Name: Month, dtype: int64
```

In [24]:
```
# extract day from date

df['Day'] = df['Date'].dt.day

df['Day'].head()
```

Out[24]:
```
0    1
1    2
2    3
3    4
4    5
Name: Day, dtype: int64
```

In [25]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 142193 entries, 0 to 142192
Data columns (total 26 columns):
Date             142193 non-null datetime64[ns]
Location         142193 non-null object
MinTemp          141556 non-null float64
MaxTemp          141871 non-null float64
Rainfall         140787 non-null float64
Evaporation      81350 non-null float64
Sunshine         74377 non-null float64
WindGustDir      132863 non-null object
WindGustSpeed    132923 non-null float64
WindDir9am       132180 non-null object
WindDir3pm       138415 non-null object
WindSpeed9am     140845 non-null float64
WindSpeed3pm     139563 non-null float64
Humidity9am      140419 non-null float64
Humidity3pm      138583 non-null float64
Pressure9am      128179 non-null float64
Pressure3pm      128212 non-null float64
Cloud9am         88536 non-null float64
Cloud3pm         85099 non-null float64
Temp9am          141289 non-null float64
Temp3pm          139467 non-null float64
RainToday        140787 non-null object
RainTomorrow     142193 non-null object
Year             142193 non-null int64
Month            142193 non-null int64
Day              142193 non-null int64
dtypes: datetime64[ns](1), float64(16), int64(3), object(6)
memory usage: 28.2+ MB
```

In [26]:
```
# drop the original Date variable

df.drop('Date', axis=1, inplace = True)
```

In [27]:
```
df.head()
```

Out[27]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	...	Pressu
0	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	WNW	...	
1	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	WSW	...	
2	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	WSW	...	
3	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	E	...	
4	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	NW	...	

5 rows × 25 columns

In [28]:
```
# find numerical variables

numerical = [var for var in df.columns if df[var].dtype!='O']

print('There are {} numerical variables\n'.format(len(numerical)))

print('The numerical variables are :', numerical)
```

```
There are 19 numerical variables

The numerical variables are : ['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'W
indSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3p
m', 'Temp9am', 'Temp3pm', 'Year', 'Month', 'Day']
```

In [29]:
```
# view numerical variables

df[numerical].head()
```

Out[29]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pr
0	13.4	22.9	0.6	NaN	NaN	44.0	20.0	24.0	71.0	22.0	
1	7.4	25.1	0.0	NaN	NaN	44.0	4.0	22.0	44.0	25.0	
2	12.9	25.7	0.0	NaN	NaN	46.0	19.0	26.0	38.0	30.0	
3	9.2	28.0	0.0	NaN	NaN	24.0	11.0	9.0	45.0	16.0	
4	17.5	32.3	1.0	NaN	NaN	41.0	7.0	20.0	82.0	33.0	

In [30]:
```
# check missing values in numerical variables

df[numerical].isnull().sum()
```

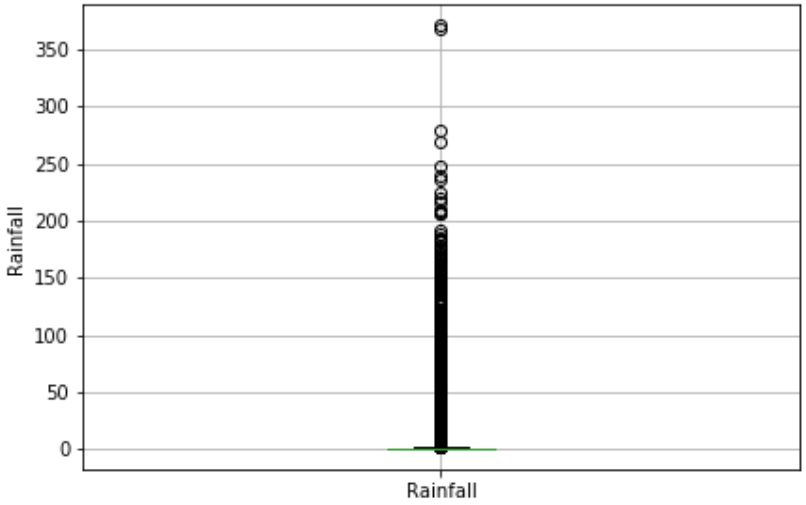
Out[30]:
```
MinTemp           637
MaxTemp           322
Rainfall         1406
Evaporation     60843
Sunshine        67816
WindGustSpeed    9270
WindSpeed9am     1348
WindSpeed3pm     2630
Humidity9am      1774
Humidity3pm      3610
Pressure9am     14014
Pressure3pm     13981
Cloud9am        53657
Cloud3pm        57094
Temp9am           904
Temp3pm          2726
Year                0
Month               0
Day                 0
dtype: int64
```

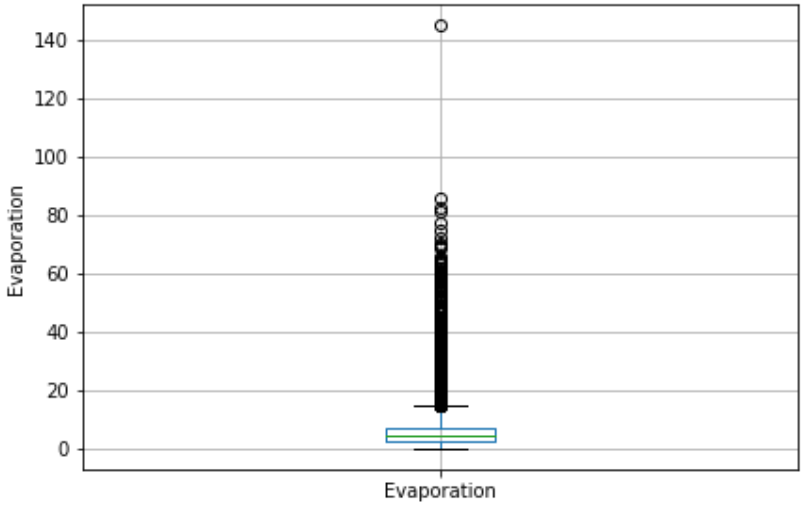
In [31]:
```
# view summary statistics in numerical variables

print(round(df[numerical].describe()),2)
```

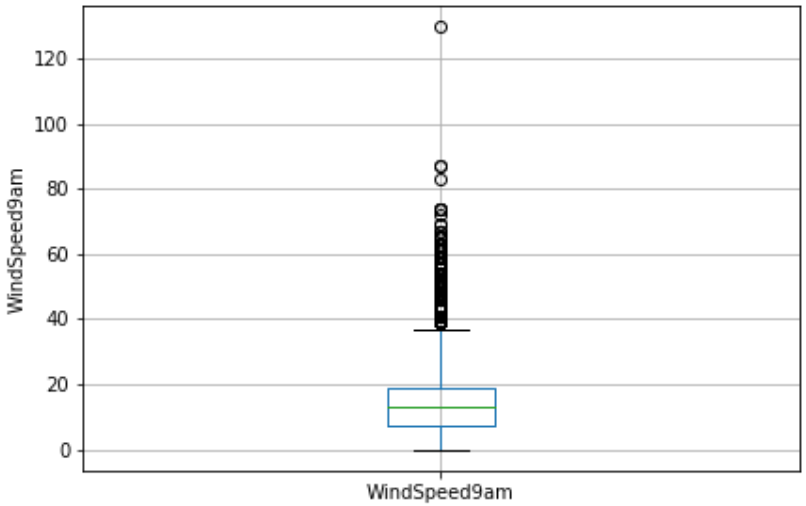
```
        MinTemp    MaxTemp   Rainfall  Evaporation  Sunshine  WindGustSpeed  \
count  141556.0  141871.0  140787.0      81350.0   74377.0       132923.0
mean       12.0      23.0       2.0          5.0       8.0           40.0
std         6.0       7.0       8.0          4.0       4.0           14.0
min        -8.0      -5.0       0.0          0.0       0.0            6.0
25%         8.0      18.0       0.0          3.0       5.0           31.0
50%        12.0      23.0       0.0          5.0       8.0           39.0
75%        17.0      28.0       1.0          7.0      11.0           48.0
max        34.0      48.0     371.0        145.0      14.0          135.0

       WindSpeed9am  WindSpeed3pm  Humidity9am  Humidity3pm  Pressure9am  \
count      140845.0      139563.0     140419.0     138583.0     128179.0
mean           14.0          19.0         69.0         51.0       1018.0
std             9.0           9.0         19.0         21.0          7.0
min             0.0           0.0          0.0          0.0        980.0
25%             7.0          13.0         57.0         37.0       1013.0
50%            13.0          19.0         70.0         52.0       1018.0
75%            19.0          24.0         83.0         66.0       1022.0
max           130.0          87.0        100.0        100.0       1041.0

       Pressure3pm  Cloud9am  Cloud3pm   Temp9am   Temp3pm      Year  \
count     128212.0   88536.0   85099.0  141289.0  139467.0  142193.0
mean        1015.0       4.0       5.0      17.0      22.0    2013.0
std            7.0       3.0       3.0       6.0       7.0       3.0
min          977.0       0.0       0.0      -7.0      -5.0    2007.0
25%         1010.0       1.0       2.0      12.0      17.0    2011.0
50%         1015.0       5.0       5.0      17.0      21.0    2013.0
75%         1020.0       7.0       7.0      22.0      26.0    2015.0
max         1040.0       9.0       9.0      40.0      47.0    2017.0

          Month       Day
count  142193.0  142193.0
mean        6.0      16.0
std         3.0       9.0
min         1.0       1.0
25%         3.0       8.0
50%         6.0      16.0
75%         9.0      23.0
max        12.0      31.0   2
```

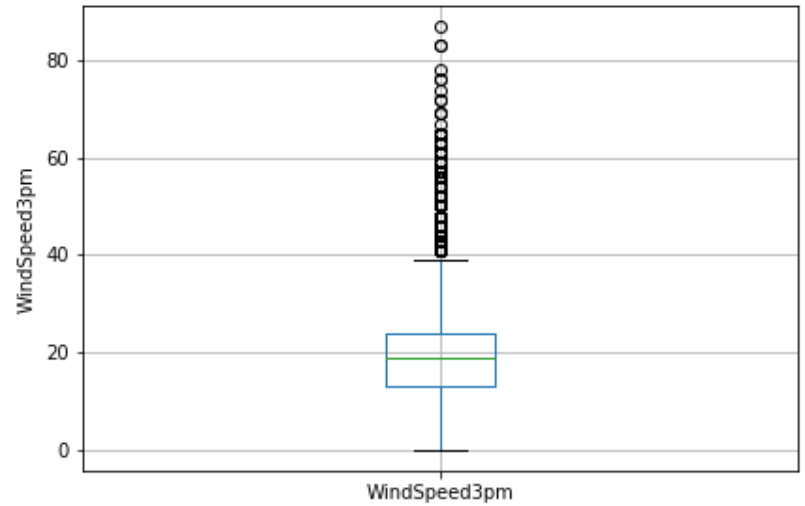
In [32]:
```
# draw boxplots to visualize outliers

plt.figure(figsize=(15,10))


plt.subplot(2, 2, 1)
fig = df.boxplot(column='Rainfall')
fig.set_title('')
fig.set_ylabel('Rainfall')


plt.subplot(2, 2, 2)
fig = df.boxplot(column='Evaporation')
fig.set_title('')
fig.set_ylabel('Evaporation')


plt.subplot(2, 2, 3)
fig = df.boxplot(column='WindSpeed9am')
fig.set_title('')
fig.set_ylabel('WindSpeed9am')


plt.subplot(2, 2, 4)
fig = df.boxplot(column='WindSpeed3pm')
fig.set_title('')
fig.set_ylabel('WindSpeed3pm')
```

Out[32]:
```
Text(0, 0.5, 'WindSpeed3pm')
```


Rainfall
0
50
100
150
200
250
300
350
Rainfall

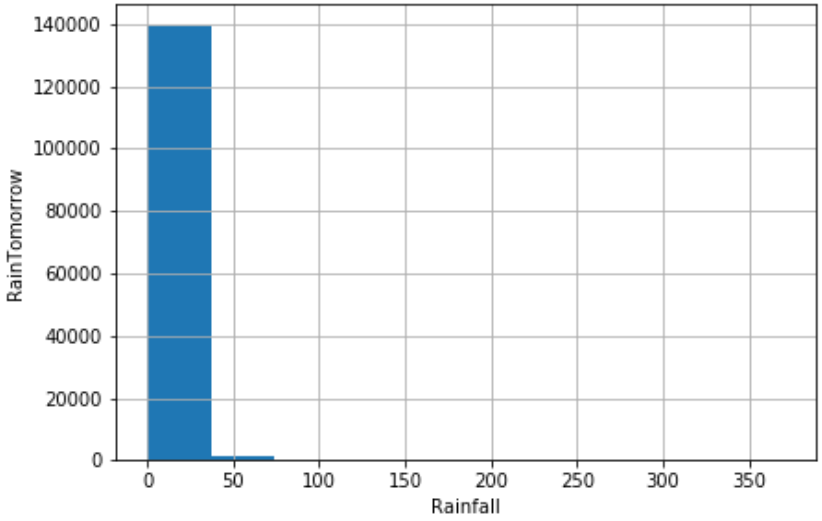

Evaporation
0
20
40
60
80
100
120
140
Evaporation

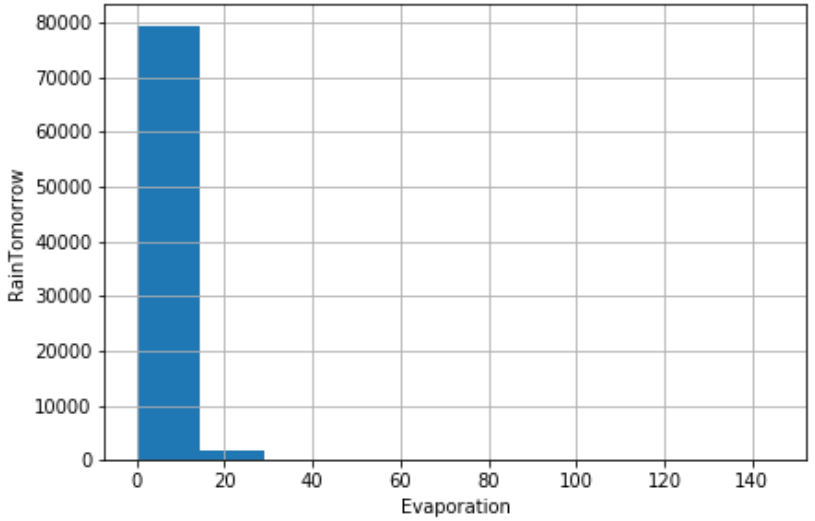

WindSpeed9am
0
20
40
60
80
100
120
WindSpeed9am

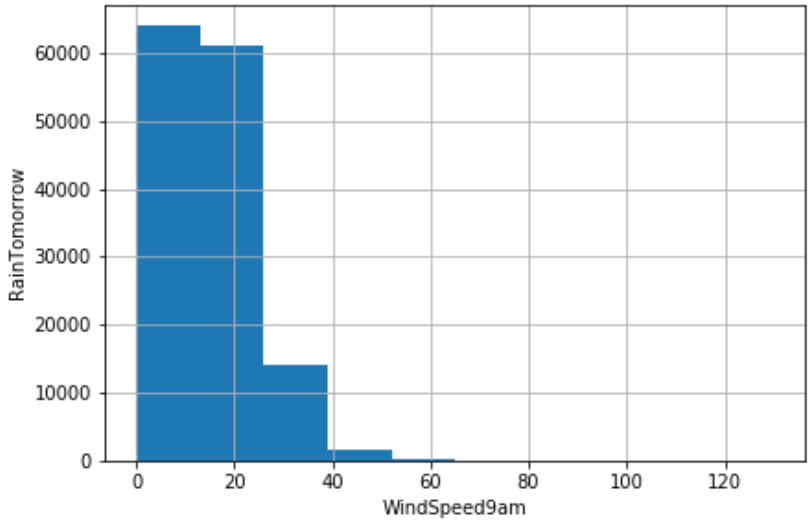

WindSpeed3pm
0
20
40
60
80
WindSpeed3pm

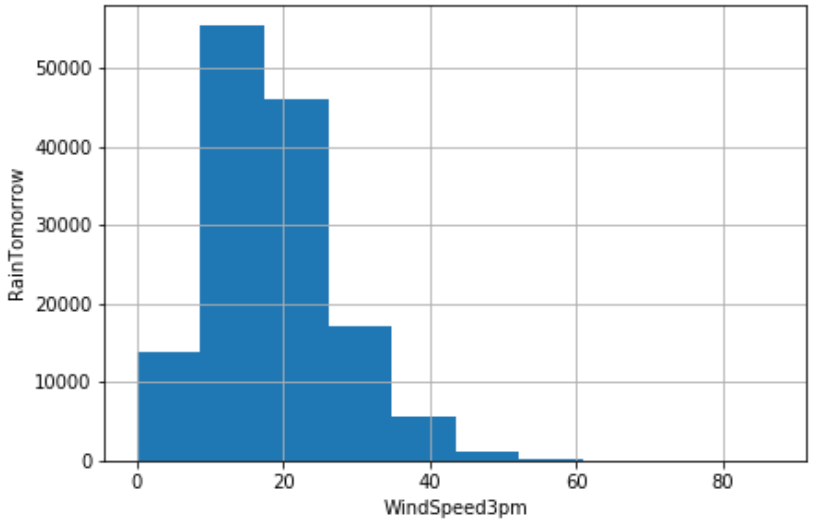
In [33]:
```
# plot histogram to check distribution

plt.figure(figsize=(15,10))


plt.subplot(2, 2, 1)
fig = df.Rainfall.hist(bins=10)
fig.set_xlabel('Rainfall')
fig.set_ylabel('RainTomorrow')


plt.subplot(2, 2, 2)
fig = df.Evaporation.hist(bins=10)
fig.set_xlabel('Evaporation')
fig.set_ylabel('RainTomorrow')


plt.subplot(2, 2, 3)
fig = df.WindSpeed9am.hist(bins=10)
fig.set_xlabel('WindSpeed9am')
fig.set_ylabel('RainTomorrow')


plt.subplot(2, 2, 4)
fig = df.WindSpeed3pm.hist(bins=10)
fig.set_xlabel('WindSpeed3pm')
fig.set_ylabel('RainTomorrow')
```

Out[33]:
```
Text(0, 0.5, 'RainTomorrow')
```









In [34]:
```
# find outliers for Rainfall variable

IQR = df.Rainfall.quantile(0.75) - df.Rainfall.quantile(0.25)
Lower_fence = df.Rainfall.quantile(0.25) - (IQR * 3)
Upper_fence = df.Rainfall.quantile(0.75) + (IQR * 3)
print('Rainfall outliers are values < {lowerboundary} or > {upperboundary}'.format(lowerboundary=Lower_fence, upp
```

```
Rainfall outliers are values < -2.4000000000000004 or > 3.2
```

In [35]:
```
# find outliers for Evaporation variable

IQR = df.Evaporation.quantile(0.75) - df.Evaporation.quantile(0.25)
Lower_fence = df.Evaporation.quantile(0.25) - (IQR * 3)
Upper_fence = df.Evaporation.quantile(0.75) + (IQR * 3)
print('Evaporation outliers are values < {lowerboundary} or > {upperboundary}'.format(lowerboundary=Lower_fence,
```

```
Evaporation outliers are values < -11.800000000000002 or > 21.800000000000004
```

In [36]:
```
# find outliers for WindSpeed9am variable

IQR = df.WindSpeed9am.quantile(0.75) - df.WindSpeed9am.quantile(0.25)
Lower_fence = df.WindSpeed9am.quantile(0.25) - (IQR * 3)
Upper_fence = df.WindSpeed9am.quantile(0.75) + (IQR * 3)
print('WindSpeed9am outliers are values < {lowerboundary} or > {upperboundary}'.format(lowerboundary=Lower_fence
```

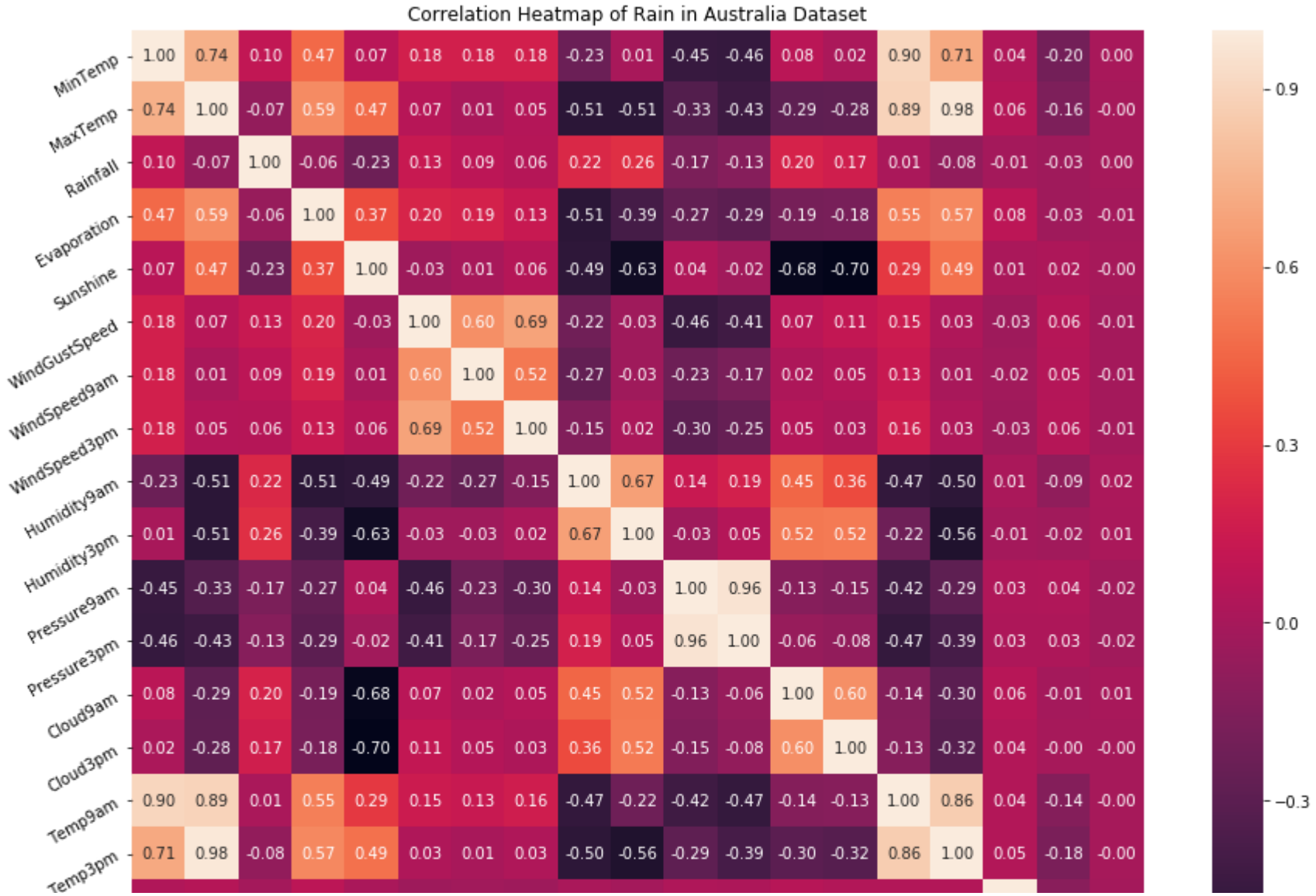
```
WindSpeed9am outliers are values < -29.0 or > 55.0
```

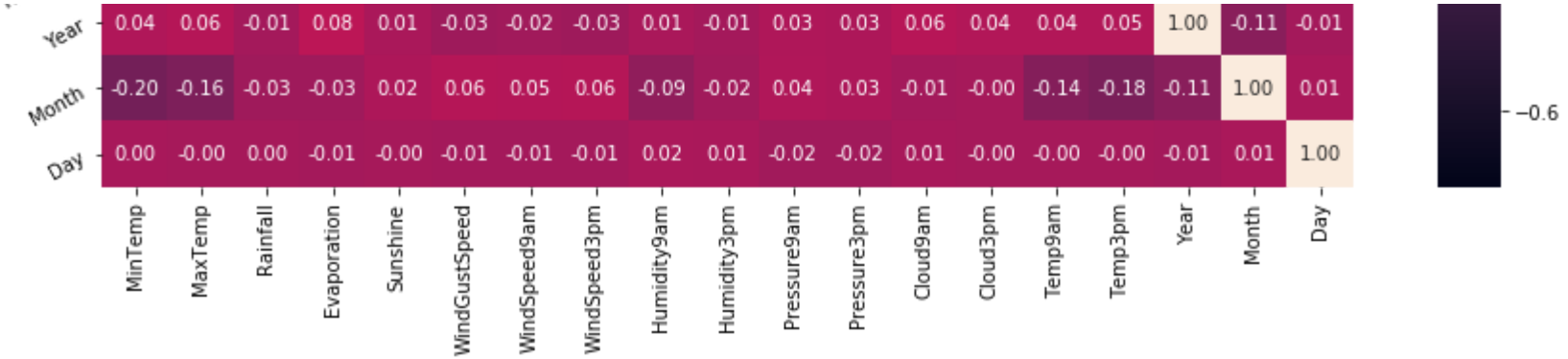
In [37]:
```
# find outliers for WindSpeed3pm variable

IQR = df.WindSpeed3pm.quantile(0.75) - df.WindSpeed3pm.quantile(0.25)
Lower_fence = df.WindSpeed3pm.quantile(0.25) - (IQR * 3)
Upper_fence = df.WindSpeed3pm.quantile(0.75) + (IQR * 3)
print('WindSpeed3pm outliers are values < {lowerboundary} or > {upperboundary}'.format(lowerboundary=Lower_fence
```

```
WindSpeed3pm outliers are values < -20.0 or > 57.0
```

Multivariate Analysis

In [38]:
```
correlation = df.corr()

plt.figure(figsize=(16,12))
plt.title('Correlation Heatmap of Rain in Australia Dataset')
ax = sns.heatmap(correlation, square=True, annot=True, fmt='.2f', linecolor='white')
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
ax.set_yticklabels(ax.get_yticklabels(), rotation=30)
plt.show()
```

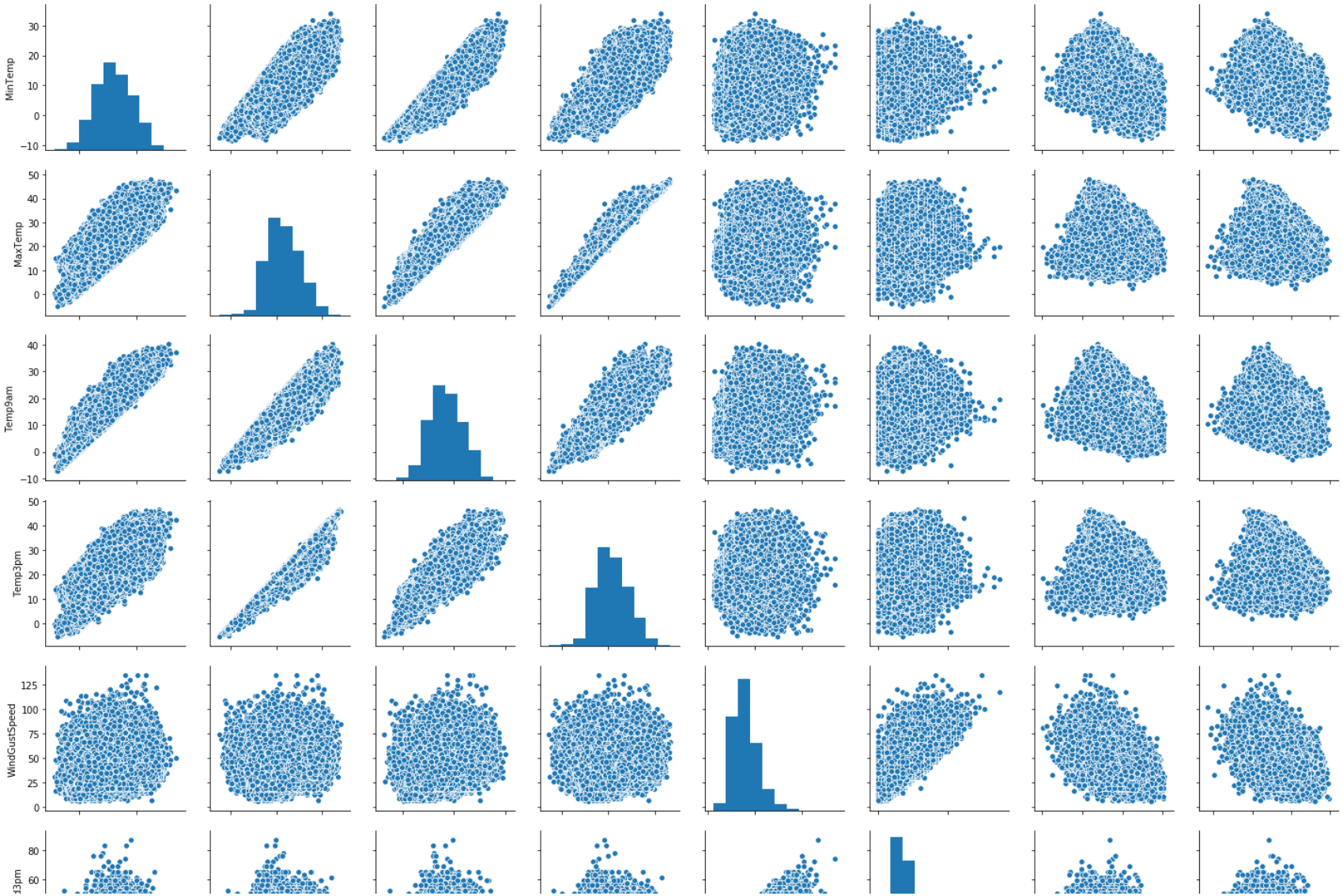


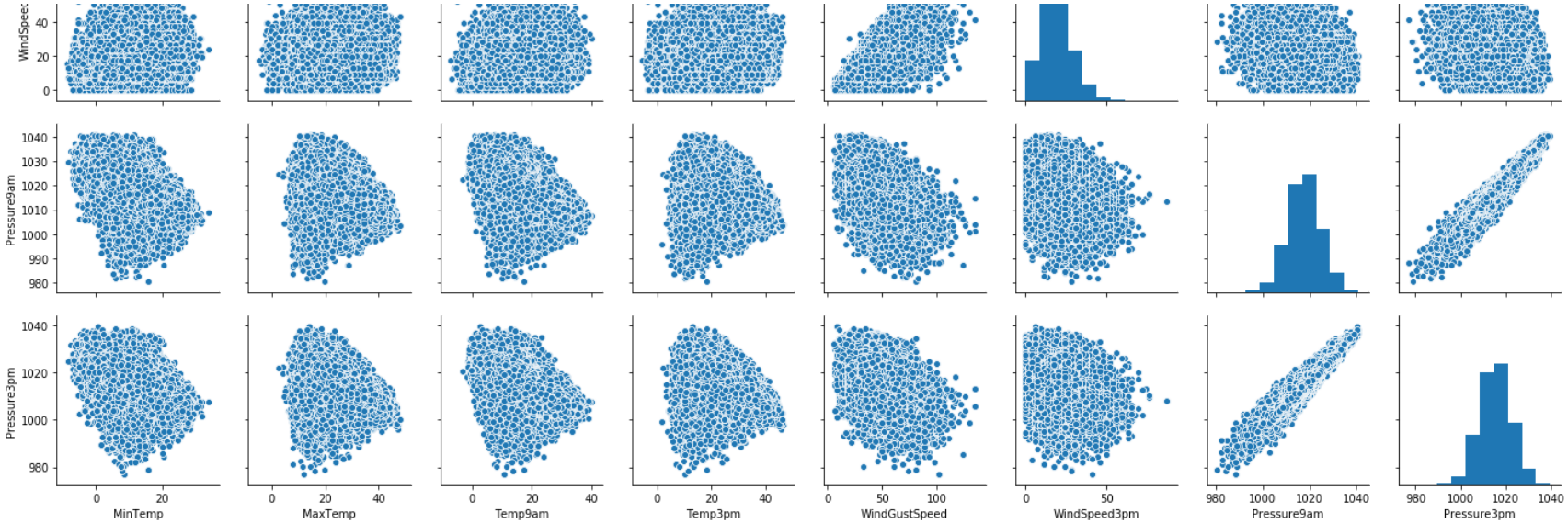


In [39]:
```
num_var = ['MinTemp', 'MaxTemp', 'Temp9am', 'Temp3pm', 'WindGustSpeed', 'WindSpeed3pm', 'Pressure9am', 'Pressure
```

In [40]:
```
sns.pairplot(df[num_var], kind='scatter', diag_kind='hist', palette='Rainbow')
plt.show()
```

```
C:\Users\Tyler\Anaconda3\lib\site-packages\numpy\lib\histograms.py:824: RuntimeWarning: invalid value encountered in greater_equal
  keep = (tmp_a >= first_edge)
C:\Users\Tyler\Anaconda3\lib\site-packages\numpy\lib\histograms.py:825: RuntimeWarning: invalid value encountered in less_equal
  keep &= (tmp_a <= last_edge)
```





Feature Engineering

In [41]:
```
X = df.drop(['RainTomorrow'], axis=1)

y = df['RainTomorrow']
```

In [42]:
```
# split X and y into training and testing sets

from sklearn.model_selection import train_test_split

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

In [43]:
```
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

Out[43]: ((113754, 24), (28439, 24))

In [44]:
```
# check data types in X_train

X_train.dtypes
```

Out[44]:
```
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
Year               int64
Month              int64
Day                int64
dtype: object
```

In [45]:
```
# display categorical variables

categorical = [col for col in X_train.columns if X_train[col].dtypes == 'O']

categorical
```

Out[45]:
```
['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday']
```

In [46]:
```
# display numerical variables

numerical = [col for col in X_train.columns if X_train[col].dtypes != 'O']

numerical
```

Out[46]:
```
['MinTemp',
 'MaxTemp',
 'Rainfall',
 'Evaporation',
 'Sunshine',
 'WindGustSpeed',
 'WindSpeed9am',
 'WindSpeed3pm',
 'Humidity9am',
 'Humidity3pm',
 'Pressure9am',
 'Pressure3pm',
 'Cloud9am',
 'Cloud3pm',
 'Temp9am',
 'Temp3pm',
 'Year',
 'Month',
 'Day']
```

In [47]:
```
# check missing values in numerical variables in X_train

X_train[numerical].isnull().sum()
```

Out[47]:
```
MinTemp           495
MaxTemp           264
Rainfall         1139
Evaporation     48718
Sunshine        54314
WindGustSpeed    7367
WindSpeed9am     1086
WindSpeed3pm     2094
Humidity9am      1449
Humidity3pm      2890
Pressure9am     11212
Pressure3pm     11186
Cloud9am        43137
Cloud3pm        45768
Temp9am           740
Temp3pm          2171
Year                0
Month               0
Day                 0
dtype: int64
```

In [48]:
```
# check missing values in numerical variables in X_test

X_test[numerical].isnull().sum()
```

Out[48]:
```
MinTemp           142
MaxTemp            58
Rainfall          267
Evaporation     12125
Sunshine        13502
WindGustSpeed    1903
WindSpeed9am      262
WindSpeed3pm      536
Humidity9am       325
Humidity3pm       720
Pressure9am      2802
Pressure3pm      2795
Cloud9am        10520
Cloud3pm        11326
Temp9am           164
Temp3pm           555
Year                0
Month               0
Day                 0
dtype: int64
```

In [49]:
```
# print percentage of missing values in the numerical variables in training set

for col in numerical:
    if X_train[col].isnull().mean()>0:
        print(col, round(X_train[col].isnull().mean(),4))
```

```
MinTemp 0.0044
MaxTemp 0.0023
Rainfall 0.01
Evaporation 0.4283
Sunshine 0.4775
WindGustSpeed 0.0648
WindSpeed9am 0.0095
WindSpeed3pm 0.0184
Humidity9am 0.0127
Humidity3pm 0.0254
Pressure9am 0.0986
Pressure3pm 0.0983
Cloud9am 0.3792
Cloud3pm 0.4023
Temp9am 0.0065
Temp3pm 0.0191
```

In [50]:
```
# impute missing values in X_train and X_test with respective column median in X_train

for df1 in [X_train, X_test]:
    for col in numerical:
        col_median=X_train[col].median()
        df1[col].fillna(col_median, inplace=True)
```

```
C:\Users\Tyler\Anaconda3\lib\site-packages\pandas\core\generic.py:6130: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  self._update_inplace(new_data)
```

In [51]:
```
# check again missing values in numerical variables in X_train

X_train[numerical].isnull().sum()
```

Out[51]:
```
MinTemp          0
MaxTemp          0
Rainfall         0
Evaporation      0
Sunshine         0
WindGustSpeed    0
WindSpeed9am     0
WindSpeed3pm     0
Humidity9am      0
Humidity3pm      0
Pressure9am      0
Pressure3pm      0
Cloud9am         0
Cloud3pm         0
Temp9am          0
Temp3pm          0
Year             0
Month            0
Day              0
dtype: int64
```

In [52]:
```
# check missing values in numerical variables in X_test

X_test[numerical].isnull().sum()
```

Out[52]:
```
MinTemp          0
MaxTemp          0
Rainfall         0
Evaporation      0
Sunshine         0
WindGustSpeed    0
WindSpeed9am     0
WindSpeed3pm     0
Humidity9am      0
Humidity3pm      0
Pressure9am      0
Pressure3pm      0
Cloud9am         0
Cloud3pm         0
Temp9am          0
Temp3pm          0
Year             0
Month            0
Day              0
dtype: int64
```

In [53]:
```
# print percentage of missing values in the categorical variables in training set

X_train[categorical].isnull().mean()
```

Out[53]:
```
Location       0.000000
WindGustDir    0.065114
WindDir9am     0.070134
WindDir3pm     0.026443
RainToday      0.010013
dtype: float64
```

In [54]:
```
# print categorical variables with missing data

for col in categorical:
    if X_train[col].isnull().mean()>0:
        print(col, (X_train[col].isnull().mean()))
```

```
WindGustDir 0.06511419378659213
WindDir9am 0.07013379749283542
WindDir3pm 0.026443026179299188
RainToday 0.01001283471350458
```

In [55]:
```
# impute missing categorical variables with most frequent value
for df2 in [X_train, X_test]:
    df2['WindGustDir'].fillna(X_train['WindGustDir'].mode()[0], inplace=True)
    df2['WindDir9am'].fillna(X_train['WindDir9am'].mode()[0], inplace=True)
    df2['WindDir3pm'].fillna(X_train['WindDir3pm'].mode()[0], inplace=True)
    df2['RainToday'].fillna(X_train['RainToday'].mode()[0], inplace=True)
```

```
C:\Users\Tyler\Anaconda3\lib\site-packages\pandas\core\generic.py:6130: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  self._update_inplace(new_data)
```

In [56]:
```
# check missing values in categorical variables in X_train

X_train[categorical].isnull().sum()
```

Out[56]:
```
Location       0
WindGustDir    0
WindDir9am     0
WindDir3pm     0
RainToday      0
dtype: int64
```

In [57]:
```
# check missing values in categorical variables in X_test

X_test[categorical].isnull().sum()
```

Out[57]:
```
Location       0
WindGustDir    0
WindDir9am     0
WindDir3pm     0
RainToday      0
dtype: int64
```

In [58]:
```
# check missing values in X_train

X_train.isnull().sum()
```

Out[58]:
```
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
Year             0
Month            0
Day              0
dtype: int64
```

In [59]:
```
# check missing values in X_test

X_test.isnull().sum()
```

Out[59]:
```
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
Year             0
Month            0
Day              0
dtype: int64
```

In [61]:
```
# cap maximum values and remove outliers from the above variables.

def max_value(df3, variable, top):
    return np.where(df3[variable]>top, top, df3[variable])

for df3 in [X_train, X_test]:
    df3['Rainfall'] = max_value(df3, 'Rainfall', 3.2)
    df3['Evaporation'] = max_value(df3, 'Evaporation', 21.8)
    df3['WindSpeed9am'] = max_value(df3, 'WindSpeed9am', 55)
    df3['WindSpeed3pm'] = max_value(df3, 'WindSpeed3pm', 57)
```

```
C:\Users\Tyler\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  import sys
C:\Users\Tyler\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)

C:\Users\Tyler\Anaconda3\lib\site-packages\ipykernel_launcher.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  if __name__ == '__main__':
C:\Users\Tyler\Anaconda3\lib\site-packages\ipykernel_launcher.py:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  # Remove the CWD from sys.path while we load stuff.
```

In [62]:
```
X_train.Rainfall.max(), X_test.Rainfall.max()
```

Out[62]:
```
(3.2, 3.2)
```

In [63]:
```
X_train.Evaporation.max(), X_test.Evaporation.max()
```

Out[63]:
```
(21.8, 21.8)
```

In [64]:
```
X_train.WindSpeed9am.max(), X_test.WindSpeed9am.max()
```

Out[64]:
```
(55.0, 55.0)
```

In [65]:
```
X_train.WindSpeed3pm.max(), X_test.WindSpeed3pm.max()
```

Out[65]:
```
(57.0, 57.0)
```

In [66]:
```
X_train[numerical].describe()
```

Out[66]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Hum
count	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	1137
mean	12.193497	23.237216	0.675080	5.151606	8.041154	39.884074	13.978155	18.614756	
std	6.388279	7.094149	1.183837	2.823707	2.769480	13.116959	8.806558	8.685862	
min	-8.200000	-4.800000	0.000000	0.000000	0.000000	6.000000	0.000000	0.000000	
25%	7.600000	18.000000	0.000000	4.000000	8.200000	31.000000	7.000000	13.000000	
50%	12.000000	22.600000	0.000000	4.800000	8.500000	39.000000	13.000000	19.000000	
75%	16.800000	28.200000	0.600000	5.400000	8.700000	46.000000	19.000000	24.000000	
max	33.900000	48.100000	3.200000	21.800000	14.500000	135.000000	55.000000	57.000000	1

In [67]:
```
# print categorical variables

categorical
```

Out[67]:
```
['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday']
```

In [68]:
```
X_train[categorical].head()
```

Out[68]:

	Location	WindGustDir	WindDir9am	WindDir3pm	RainToday
110803	Witchcliffe	S	SSE	S	No
87289	Cairns	ENE	SSE	SE	Yes
134949	AliceSprings	E	NE	N	No
85553	Cairns	ESE	SSE	E	No
16110	Newcastle	W	N	SE	No

In [70]:
```
# encode RainToday variable

import category_encoders as ce

encoder = ce.BinaryEncoder(cols=['RainToday'])

X_train = encoder.fit_transform(X_train)

X_test = encoder.transform(X_test)
```

In [71]:
```
X_train.head()
```

Out[71]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	...	P
110803	Witchcliffe	13.9	22.6	0.2	4.8	8.5	S	41.0	SSE	S	...	
87289	Cairns	22.4	29.4	2.0	6.0	6.3	ENE	33.0	SSE	SE	...	
134949	AliceSprings	9.7	36.2	0.0	11.4	12.3	E	31.0	NE	N	...	
85553	Cairns	20.5	30.1	0.0	8.8	11.1	ESE	37.0	SSE	E	...	
16110	Newcastle	16.8	29.2	0.0	4.8	8.5	W	39.0	N	SE	...	

5 rows × 25 columns

In [72]:
```
# We can see that two additional variables RainToday_0 and RainToday_1 are created from RainToday variable.

X_train = pd.concat([X_train[numerical], X_train[['RainToday_0', 'RainToday_1']],
                     pd.get_dummies(X_train.Location),
                     pd.get_dummies(X_train.WindGustDir),
                     pd.get_dummies(X_train.WindDir9am),
                     pd.get_dummies(X_train.WindDir3pm)], axis=1)
```

In [73]:
```
X_train.head()
```

Out[73]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pr
110803	13.9	22.6	0.2	4.8	8.5	41.0	20.0	28.0	65.0	55.
87289	22.4	29.4	2.0	6.0	6.3	33.0	7.0	19.0	71.0	59.
134949	9.7	36.2	0.0	11.4	12.3	31.0	15.0	11.0	6.0	2.
85553	20.5	30.1	0.0	8.8	11.1	37.0	22.0	19.0	59.0	53.
16110	16.8	29.2	0.0	4.8	8.5	39.0	0.0	7.0	72.0	53.

5 rows × 118 columns

In [74]:
```
#  create the X_test testing set.

X_test = pd.concat([X_test[numerical], X_test[['RainToday_0', 'RainToday_1']],
                     pd.get_dummies(X_test.Location),
                     pd.get_dummies(X_test.WindGustDir),
                     pd.get_dummies(X_test.WindDir9am),
                     pd.get_dummies(X_test.WindDir3pm)], axis=1)
```

In [75]:
```
X_test.head()
```

Out[75]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pr
86232	17.4	29.0	0.0	3.6	11.1	33.0	11.0	19.0	63.0	61.
57576	6.8	14.4	0.8	0.8	8.5	46.0	17.0	22.0	80.0	55.
124071	10.1	15.4	3.2	4.8	8.5	31.0	13.0	9.0	70.0	61.
117955	14.4	33.4	0.0	8.0	11.6	41.0	9.0	17.0	40.0	23.
133468	6.8	14.3	3.2	0.2	7.3	28.0	15.0	13.0	92.0	47.

5 rows × 118 columns

Feature Scaling

Map all the feature variables onto the same scale.

In [76]:
```
X_train.describe()
```

Out[76]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Hum
count	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	1137
mean	12.193497	23.237216	0.675080	5.151606	8.041154	39.884074	13.978155	18.614756	
std	6.388279	7.094149	1.183837	2.823707	2.769480	13.116959	8.806558	8.685862	
min	-8.200000	-4.800000	0.000000	0.000000	0.000000	6.000000	0.000000	0.000000	
25%	7.600000	18.000000	0.000000	4.000000	8.200000	31.000000	7.000000	13.000000	
50%	12.000000	22.600000	0.000000	4.800000	8.500000	39.000000	13.000000	19.000000	
75%	16.800000	28.200000	0.600000	5.400000	8.700000	46.000000	19.000000	24.000000	
max	33.900000	48.100000	3.200000	21.800000	14.500000	135.000000	55.000000	57.000000	1

8 rows × 118 columns

In [77]:
```
cols = X_train.columns
```

In [78]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

In [79]:
```
X_train = pd.DataFrame(X_train, columns=[cols])
```

In [80]:
```
X_test = pd.DataFrame(X_test, columns=[cols])
```

In [81]:
```
X_train.describe()
```

Out[81]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Hum
count	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	1137
mean	0.484406	0.530004	0.210962	0.236312	0.554562	0.262667	0.254148	0.326575	
std	0.151741	0.134105	0.369949	0.129528	0.190999	0.101682	0.160119	0.152384	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.375297	0.431002	0.000000	0.183486	0.565517	0.193798	0.127273	0.228070	
50%	0.479810	0.517958	0.000000	0.220183	0.586207	0.255814	0.236364	0.333333	
75%	0.593824	0.623819	0.187500	0.247706	0.600000	0.310078	0.345455	0.421053	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 118 columns

Model training

X_train dataset ready to be fed into the Logistic Regression classifier

In [82]:
```
# train a logistic regression model on the training set
from sklearn.linear_model import LogisticRegression


# instantiate the model
logreg = LogisticRegression(solver='liblinear', random_state=0)


# fit the model
logreg.fit(X_train, y_train)
```

Out[82]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=0, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [84]:
```
y_pred_test = logreg.predict(X_test)

y_pred_test
```

Out[84]:
```
array(['No', 'No', 'No', ..., 'No', 'No', 'Yes'], dtype=object)
```

Predict_proba method gives the probabilities for the target variable(0 and 1) in this case, in array form 0 is for probability of no rain and 1 is for probability of rain.

In [85]:
```
# probability of getting output as 0 - no rain

logreg.predict_proba(X_test)[:,0]
```

Out[85]:
```
array([0.91387283, 0.83563142, 0.82035773, ..., 0.97674028, 0.79853118,
       0.3073425 ])
```

In [86]:
```
# probability of getting output as 1 - rain

logreg.predict_proba(X_test)[:,1]
```

Out[86]:
```
array([0.08612717, 0.16436858, 0.17964227, ..., 0.02325972, 0.20146882,
       0.6926575 ])
```

Check the accuracy

In [87]:
```
from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred_test)))
```

```
Model accuracy score: 0.8501
```

In [88]:
```
y_pred_train = logreg.predict(X_train)

y_pred_train
```

Out[88]:
```
array(['No', 'No', 'No', ..., 'No', 'No', 'No'], dtype=object)
```

In [89]:
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))
```

```
Training-set accuracy score: 0.8476
```

Check for overfitting

In [90]:
```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(logreg.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(logreg.score(X_test, y_test)))
```

```
Training set score: 0.8476
Test set score: 0.8501
```

In [ ]:
```

```