Logistic Regression

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # data visualization
import seaborn as sns # statistical data visualization
%matplotlib inline
```

In [2]:
```
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list all

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [3]:
```
import warnings

warnings.filterwarnings('ignore')
```

Import dataset

In [4]:
```
data = './weatherAUS.csv'

df = pd.read_csv(data)
```

Exploratory data analysis

In [5]:
```
# view dimensions of dataset

df.shape
```

Out[5]: (145460, 23)

In [6]:
```
# preview the dataset

df.head()
```

Out[6]:

	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	

5 rows × 23 columns

In [7]:
```
col_names = df.columns

col_names
```

Out[7]:
```
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 [8]:
```
# view summary of dataset

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 23 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   Date           145460 non-null  object 
 1   Location       145460 non-null  object 
 2   MinTemp        143975 non-null  float64
 3   MaxTemp        144199 non-null  float64
 4   Rainfall       142199 non-null  float64
 5   Evaporation    82670 non-null   float64
 6   Sunshine       75625 non-null   float64
 7   WindGustDir    135134 non-null  object 
 8   WindGustSpeed  135197 non-null  float64
 9   WindDir9am     134894 non-null  object 
 10  WindDir3pm     141232 non-null  object 
 11  WindSpeed9am   143693 non-null  float64
 12  WindSpeed3pm   142398 non-null  float64
 13  Humidity9am    142806 non-null  float64
 14  Humidity3pm    140953 non-null  float64
 15  Pressure9am    130395 non-null  float64
 16  Pressure3pm    130432 non-null  float64
 17  Cloud9am       89572 non-null   float64
 18  Cloud3pm       86102 non-null   float64
 19  Temp9am        143693 non-null  float64
 20  Temp3pm        141851 non-null  float64
 21  RainToday      142199 non-null  object 
 22  RainTomorrow   142193 non-null  object 
dtypes: float64(16), object(7)
memory usage: 25.5+ MB
```

In [9]:

```
# find categorical variables

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', 'RainTomorrow']
```

In [10]:

```
# view the categorical variables

df[categorical].head()
```

Out[10]:

	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

Explore problems within categorical variables

Missing values in categorical variables

In [11]:
```
# check missing values in categorical variables

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

Out[11]:
```
Date                0
Location            0
WindGustDir     10326
WindDir9am      10566
WindDir3pm       4228
RainToday        3261
RainTomorrow     3267
dtype: int64
```

In [12]:
```
# print categorical variables containing missing values

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

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

```
WindGustDir     10326
WindDir9am      10566
WindDir3pm       4228
RainToday        3261
RainTomorrow     3267
dtype: int64
```

In [13]:
```
# view frequency of categorical variables

for var in categorical:

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

```
2013-11-12    49
2014-09-01    49
2014-08-23    49
2014-08-24    49
2014-08-25    49
              ..
2007-11-29     1
2007-11-28     1
2007-11-27     1
2007-11-26     1
2008-01-31     1
Name: Date, Length: 3436, dtype: int64
Canberra            3436
Sydney              3344
Darwin              3193
Melbourne           3193
Brisbane            3193
Adelaide            3193
Perth               3193
Hobart              3193
Albany              3040
MountGambier        3040
Ballarat            3040
Townsville          3040
GoldCoast           3040
Cairns              3040
Launceston          3040
AliceSprings        3040
Bendigo             3040
Albury              3040
MountGinini         3040
Wollongong          3040
Newcastle           3039
Tuggeranong         3039
Penrith             3039
Woomera             3009
Nuriootpa           3009
Cobar               3009
CoffsHarbour        3009
Moree               3009
Sale                3009
PerthAirport        3009
PearceRAAF          3009
Witchcliffe         3009
BadgerysCreek       3009
Mildura             3009
NorfolkIsland       3009
MelbourneAirport    3009
Richmond            3009
SydneyAirport       3009
WaggaWagga          3009
Williamtown         3009
Dartmoor            3009
Watsonia            3009
Portland            3009
Walpole             3006
NorahHead           3004
SalmonGums          3001
Katherine           1578
Nhil                1578
Uluru               1578
```

```
Name: Location, dtype: int64
W      9915
SE     9418
N      9313
SSE    9216
E      9181
S      9168
WSW    9069
SW     8967
SSW    8736
WNW    8252
NW     8122
ENE    8104
ESE    7372
NE     7133
NNW    6620
NNE    6548
Name: WindGustDir, dtype: int64
N      11758
SE      9287
E       9176
SSE     9112
NW      8749
S       8659
W       8459
SW      8423
NNE     8129
NNW     7980
ENE     7836
NE      7671
ESE     7630
SSW     7587
WNW     7414
WSW     7024
Name: WindDir9am, dtype: int64
SE     10838
W      10110
S       9926
WSW     9518
SSE     9399
SW      9354
N       8890
WNW     8874
NW      8610
ESE     8505
E       8472
NE      8263
SSW     8156
NNW     7870
ENE     7857
NNE     6590
Name: WindDir3pm, dtype: int64
No     110319
Yes     31880
Name: RainToday, dtype: int64
No     110316
Yes     31877
Name: RainTomorrow, dtype: int64
```

In [14]:
```
# view frequency distribution of categorical variables
```

```
for var in categorical:

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

```
2013-11-12    0.000337
2014-09-01    0.000337
2014-08-23    0.000337
2014-08-24    0.000337
2014-08-25    0.000337
                ...
2007-11-29    0.000007
2007-11-28    0.000007
2007-11-27    0.000007
2007-11-26    0.000007
2008-01-31    0.000007
Name: Date, Length: 3436, dtype: float64
Canberra            0.023622
Sydney              0.022989
Darwin              0.021951
Melbourne           0.021951
Brisbane            0.021951
Adelaide            0.021951
Perth               0.021951
Hobart              0.021951
Albany              0.020899
MountGambier        0.020899
Ballarat            0.020899
Townsville          0.020899
GoldCoast           0.020899
Cairns              0.020899
Launceston          0.020899
AliceSprings        0.020899
Bendigo             0.020899
Albury              0.020899
MountGinini         0.020899
Wollongong          0.020899
Newcastle           0.020892
Tuggeranong         0.020892
Penrith             0.020892
Woomera             0.020686
Nuriootpa           0.020686
Cobar               0.020686
CoffsHarbour        0.020686
Moree               0.020686
Sale                0.020686
PerthAirport        0.020686
PearceRAAF          0.020686
Witchcliffe         0.020686
BadgerysCreek       0.020686
Mildura             0.020686
NorfolkIsland       0.020686
MelbourneAirport    0.020686
Richmond            0.020686
SydneyAirport       0.020686
WaggaWagga          0.020686
Williamtown         0.020686
Dartmoor            0.020686
Watsonia            0.020686
Portland            0.020686
Walpole             0.020665
NorahHead           0.020652
SalmonGums          0.020631
Katherine           0.010848
Nhil                0.010848
Uluru               0.010848
```

```
Name: Location, dtype: float64
W      0.068163
SE     0.064746
N      0.064024
SSE    0.063358
E      0.063117
S      0.063028
WSW    0.062347
SW     0.061646
SSW    0.060058
WNW    0.056730
NW     0.055837
ENE    0.055713
ESE    0.050681
NE     0.049038
NNW    0.045511
NNE    0.045016
Name: WindGustDir, dtype: float64
N      0.080833
SE     0.063846
E      0.063083
SSE    0.062643
NW     0.060147
S      0.059528
W      0.058153
SW     0.057906
NNE    0.055885
NNW    0.054860
ENE    0.053870
NE     0.052736
ESE    0.052454
SSW    0.052159
WNW    0.050969
WSW    0.048288
Name: WindDir9am, dtype: float64
SE     0.074508
W      0.069504
S      0.068239
WSW    0.065434
SSE    0.064616
SW     0.064306
N      0.061116
WNW    0.061006
NW     0.059192
ESE    0.058470
E      0.058243
NE     0.056806
SSW    0.056070
NNW    0.054104
ENE    0.054015
NNE    0.045305
Name: WindDir3pm, dtype: float64
No     0.758415
Yes    0.219167
Name: RainToday, dtype: float64
No     0.758394
Yes    0.219146
Name: RainTomorrow, dtype: float64
```

Number of labels: cardinality

In [15]:
```
# 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  3  labels
```

Feature Engineering of Date Variable

In [16]:
```
df['Date'].dtypes

#We can see that the data type of Date variable is object. I will parse the date cu
```

Out[16]:
```
dtype('O')
```

In [17]:
```
# parse the dates, currently coded as strings, into datetime format

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

In [18]:
```
# extract year from date

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

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

Out[18]:
```
0    2008
1    2008
2    2008
3    2008
4    2008
Name: Year, dtype: int64
```

In [19]:
```
# extract month from date

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

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

Out[19]:
```
0    12
1    12
2    12
3    12
4    12
Name: Month, dtype: int64
```

In [20]:
```
# extract day from date

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

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

Out[20]:
```
0    1
1    2
2    3
3    4
4    5
Name: Day, dtype: int64
```

In [21]:
```
# again view the summary of dataset

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 26 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Date           145460 non-null  datetime64[ns]
 1   Location       145460 non-null  object
 2   MinTemp        143975 non-null  float64
 3   MaxTemp        144199 non-null  float64
 4   Rainfall       142199 non-null  float64
 5   Evaporation    82670 non-null   float64
 6   Sunshine       75625 non-null   float64
 7   WindGustDir    135134 non-null  object
 8   WindGustSpeed  135197 non-null  float64
 9   WindDir9am     134894 non-null  object
 10  WindDir3pm     141232 non-null  object
 11  WindSpeed9am   143693 non-null  float64
 12  WindSpeed3pm   142398 non-null  float64
 13  Humidity9am    142806 non-null  float64
 14  Humidity3pm    140953 non-null  float64
 15  Pressure9am    130395 non-null  float64
 16  Pressure3pm    130432 non-null  float64
 17  Cloud9am       89572 non-null   float64
 18  Cloud3pm       86102 non-null   float64
 19  Temp9am        143693 non-null  float64
 20  Temp3pm        141851 non-null  float64
 21  RainToday      142199 non-null  object
 22  RainTomorrow   142193 non-null  object
 23  Year           145460 non-null  int64
 24  Month          145460 non-null  int64
 25  Day            145460 non-null  int64
dtypes: datetime64[ns](1), float64(16), int64(3), object(6)
memory usage: 28.9+ MB
```

In [22]:
```
# drop the original Date variable

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

In [23]:
```
# preview the dataset again

df.head()
```

Out[23]:

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

5 rows × 25 columns

Explore Categorical Variables

In [24]:

```
# find categorical variables

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 6 categorical variables

The categorical variables are : ['Location', 'WindGustDir', 'WindDir9am', 'WindDir
3pm', 'RainToday', 'RainTomorrow']
```

In [25]:

```
# check for missing values in categorical variables

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

Out[25]:

```
Location            0
WindGustDir     10326
WindDir9am      10566
WindDir3pm       4228
RainToday        3261
RainTomorrow     3267
dtype: int64
```

Explore Location variable

In [26]:

```
# print number of labels in Location variable

print('Location contains', len(df.Location.unique()), 'labels')
```

```
Location contains 49 labels
```

In [27]:

```
# check labels in location variable

df.Location.unique()
```

Out[27]:
```
array(['Albury', 'BadgerysCreek', 'Cobar', 'CoffsHarbour', 'Moree',
       'Newcastle', 'NorahHead', 'NorfolkIsland', 'Penrith', 'Richmond',
       'Sydney', 'SydneyAirport', 'WaggaWagga', 'Williamtown',
       'Wollongong', 'Canberra', 'Tuggeranong', 'MountGinini', 'Ballarat',
       'Bendigo', 'Sale', 'MelbourneAirport', 'Melbourne', 'Mildura',
       'Nhil', 'Portland', 'Watsonia', 'Dartmoor', 'Brisbane', 'Cairns',
       'GoldCoast', 'Townsville', 'Adelaide', 'MountGambier', 'Nuriootpa',
       'Woomera', 'Albany', 'Witchcliffe', 'PearceRAAF', 'PerthAirport',
       'Perth', 'SalmonGums', 'Walpole', 'Hobart', 'Launceston',
       'AliceSprings', 'Darwin', 'Katherine', 'Uluru'], dtype=object)
```

In [28]:
```
# check frequency distribution of values in Location variable

df.Location.value_counts()
```

Out[28]:
```
Canberra            3436
Sydney              3344
Darwin              3193
Melbourne           3193
Brisbane            3193
Adelaide            3193
Perth               3193
Hobart              3193
Albany              3040
MountGambier        3040
Ballarat            3040
Townsville          3040
GoldCoast           3040
Cairns              3040
Launceston          3040
AliceSprings        3040
Bendigo             3040
Albury              3040
MountGinini         3040
Wollongong          3040
Newcastle           3039
Tuggeranong         3039
Penrith             3039
Woomera             3009
Nuriootpa           3009
Cobar               3009
CoffsHarbour        3009
Moree               3009
Sale                3009
PerthAirport        3009
PearceRAAF          3009
Witchcliffe         3009
BadgerysCreek       3009
Mildura             3009
NorfolkIsland       3009
MelbourneAirport    3009
Richmond            3009
SydneyAirport       3009
WaggaWagga          3009
Williamtown         3009
Dartmoor            3009
Watsonia            3009
Portland            3009
Walpole             3006
NorahHead           3004
SalmonGums          3001
Katherine           1578
Nhil                1578
Uluru               1578
Name: Location, dtype: int64
```

In [29]:
```
# let's do One Hot Encoding of Location variable
# get k-1 dummy variables after One Hot Encoding
# preview the dataset with head() method

pd.get_dummies(df.Location, drop_first=True).head()
```

Out[29]:

	Albany	Albury	AliceSprings	BadgerysCreek	Ballarat	Bendigo	Brisbane	Cairns	Canberra
0	0	1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0	0

5 rows × 48 columns

Explore WindGustDir variable

In [30]:
```
# print number of labels in WindGustDir variable

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

```
WindGustDir contains 17 labels
```

In [31]:
```
# check labels in WindGustDir variable

df['WindGustDir'].unique()
```

Out[31]:
```
array(['W', 'WNW', 'WSW', 'NE', 'NNW', 'N', 'NNE', 'SW', nan, 'ENE',
       'SSE', 'S', 'NW', 'SE', 'ESE', 'E', 'SSW'], dtype=object)
```

In [32]:
```
# check frequency distribution of values in WindGustDir variable

df.WindGustDir.value_counts()
```

Out[32]:
```
W      9915
SE     9418
N      9313
SSE    9216
E      9181
S      9168
WSW    9069
SW     8967
SSW    8736
WNW    8252
NW     8122
ENE    8104
ESE    7372
NE     7133
NNW    6620
NNE    6548
Name: WindGustDir, dtype: int64
```

In [33]:
```
# let's do One Hot Encoding of WindGustDir variable
# get k-1 dummy variables after One Hot Encoding
# also add an additional dummy variable to indicate there was missing data
# preview the dataset with head() method

pd.get_dummies(df.WindGustDir, drop_first=True, dummy_na=True).head()
```

Out[33]:

	ENE	ESE	N	NE	NNE	NNW	NW	S	SE	SSE	SSW	SW	W	WNW	WSW	NaN
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

In [34]:
```
# sum the number of 1s per boolean variable over the rows of the dataset
# it will tell us how many observations we have for each category

pd.get_dummies(df.WindGustDir, drop_first=True, dummy_na=True).sum(axis=0)
```

Out[34]:
```
ENE     8104
ESE     7372
N       9313
NE      7133
NNE     6548
NNW     6620
NW      8122
S       9168
SE      9418
SSE     9216
SSW     8736
SW      8967
W       9915
WNW     8252
WSW     9069
NaN    10326
dtype: int64
```

Explore WindDir9am variable

In [35]:
```
# print number of labels in WindDir9am variable

print('WindDir9am contains', len(df['WindDir9am'].unique()), 'labels')
```

```
WindDir9am contains 17 labels
```

In [36]:
```
# check labels in WindDir9am variable

df['WindDir9am'].unique()
```

Out[36]:
```
array(['W', 'NNW', 'SE', 'ENE', 'SW', 'SSE', 'S', 'NE', nan, 'SSW', 'N',
       'WSW', 'ESE', 'E', 'NW', 'WNW', 'NNE'], dtype=object)
```

In [37]:
```
# check frequency distribution of values in WindDir9am variable

df['WindDir9am'].value_counts()
```

Out[37]:
```
N      11758
SE      9287
E       9176
SSE     9112
NW      8749
S       8659
W       8459
SW      8423
NNE     8129
NNW     7980
ENE     7836
NE      7671
ESE     7630
SSW     7587
WNW     7414
WSW     7024
Name: WindDir9am, dtype: int64
```

In [38]:
```
# let's do One Hot Encoding of WindDir9am variable
# get k-1 dummy variables after One Hot Encoding
# also add an additional dummy variable to indicate there was missing data
# preview the dataset with head() method

pd.get_dummies(df.WindDir9am, drop_first=True, dummy_na=True).head()
```

Out[38]:

	ENE	ESE	N	NE	NNE	NNW	NW	S	SE	SSE	SSW	SW	W	WNW	WSW	NaN
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

In [39]:
```
# sum the number of 1s per boolean variable over the rows of the dataset
# it will tell us how many observations we have for each category

pd.get_dummies(df.WindDir9am, drop_first=True, dummy_na=True).sum(axis=0)
```

Out[39]:
```
ENE     7836
ESE     7630
N      11758
NE      7671
NNE     8129
NNW     7980
NW      8749
S       8659
SE      9287
SSE     9112
SSW     7587
SW      8423
W       8459
WNW     7414
WSW     7024
NaN    10566
dtype: int64
```

Explore WindDir3pm variable

In [40]:
```
# print number of labels in WindDir3pm variable

print('WindDir3pm contains', len(df['WindDir3pm'].unique()), 'labels')
```

```
WindDir3pm contains 17 labels
```

In [41]:
```
# check labels in WindDir3pm variable

df['WindDir3pm'].unique()
```

Out[41]:
```
array(['WNW', 'WSW', 'E', 'NW', 'W', 'SSE', 'ESE', 'ENE', 'NNW', 'SSW',
       'SW', 'SE', 'N', 'S', 'NNE', nan, 'NE'], dtype=object)
```

In [42]:
```
# check frequency distribution of values in WindDir3pm variable

df['WindDir3pm'].value_counts()
```

Out[42]:
```
SE     10838
W      10110
S       9926
WSW     9518
SSE     9399
SW      9354
N       8890
WNW     8874
NW      8610
ESE     8505
E       8472
NE      8263
SSW     8156
NNW     7870
ENE     7857
NNE     6590
Name: WindDir3pm, dtype: int64
```

In [43]:
```
# let's do One Hot Encoding of WindDir3pm variable
# get k-1 dummy variables after One Hot Encoding
# also add an additional dummy variable to indicate there was missing data
# preview the dataset with head() method

pd.get_dummies(df.WindDir3pm, drop_first=True, dummy_na=True).head()
```

Out[43]:

	ENE	ESE	N	NE	NNE	NNW	NW	S	SE	SSE	SSW	SW	W	WNW	WSW	NaN
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

In [44]:
```
# sum the number of 1s per boolean variable over the rows of the dataset
# it will tell us how many observations we have for each category

pd.get_dummies(df.WindDir3pm, drop_first=True, dummy_na=True).sum(axis=0)
```

Out[44]:
```
ENE     7857
ESE     8505
N       8890
NE      8263
NNE     6590
NNW     7870
NW      8610
S       9926
SE     10838
SSE     9399
SSW     8156
SW      9354
W      10110
WNW     8874
WSW     9518
NaN     4228
dtype: int64
```

Explore RainToday variable

In [45]:
```
# print number of labels in RainToday variable

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

```
RainToday contains 3 labels
```

In [46]:
```
# check labels in WindGustDir variable

df['RainToday'].unique()
```

Out[46]:
```
array(['No', 'Yes', nan], dtype=object)
```

In [47]:
```
# check frequency distribution of values in WindGustDir variable

df.RainToday.value_counts()
```

Out[47]:
```
No     110319
Yes     31880
Name: RainToday, dtype: int64
```

In [48]:
```
# let's do One Hot Encoding of RainToday variable
# get k-1 dummy variables after One Hot Encoding
# also add an additional dummy variable to indicate there was missing data
# preview the dataset with head() method

pd.get_dummies(df.RainToday, drop_first=True, dummy_na=True).head()
```

Out[48]:

	Yes	NaN
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0

In [49]:
```
# sum the number of 1s per boolean variable over the rows of the dataset
# it will tell us how many observations we have for each category
```

```
pd.get_dummies(df.RainToday, drop_first=True, dummy_na=True).sum(axis=0)
```

Out[49]:
```
Yes     31880
NaN      3261
dtype: int64
```

Explore Numerical Variables

In [50]:
```
# 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', 'S
unshine', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidit
y3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm',
'Year', 'Month', 'Day']
```

In [51]:
```
# view the numerical variables

df[numerical].head()
```

Out[51]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	Wind!
0	13.4	22.9	0.6	NaN	NaN	44.0	20.0	
1	7.4	25.1	0.0	NaN	NaN	44.0	4.0	
2	12.9	25.7	0.0	NaN	NaN	46.0	19.0	
3	9.2	28.0	0.0	NaN	NaN	24.0	11.0	
4	17.5	32.3	1.0	NaN	NaN	41.0	7.0	

Missing values in numerical variables

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

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

Out[52]:
```
MinTemp           1485
MaxTemp           1261
Rainfall          3261
Evaporation      62790
Sunshine         69835
WindGustSpeed    10263
WindSpeed9am      1767
WindSpeed3pm      3062
Humidity9am       2654
Humidity3pm       4507
Pressure9am      15065
Pressure3pm      15028
Cloud9am         55888
Cloud3pm         59358
Temp9am           1767
Temp3pm           3609
Year                 0
Month                0
Day                  0
dtype: int64
```

Outliers in numerical variables

In [53]:
```
# view summary statistics in numerical variables

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

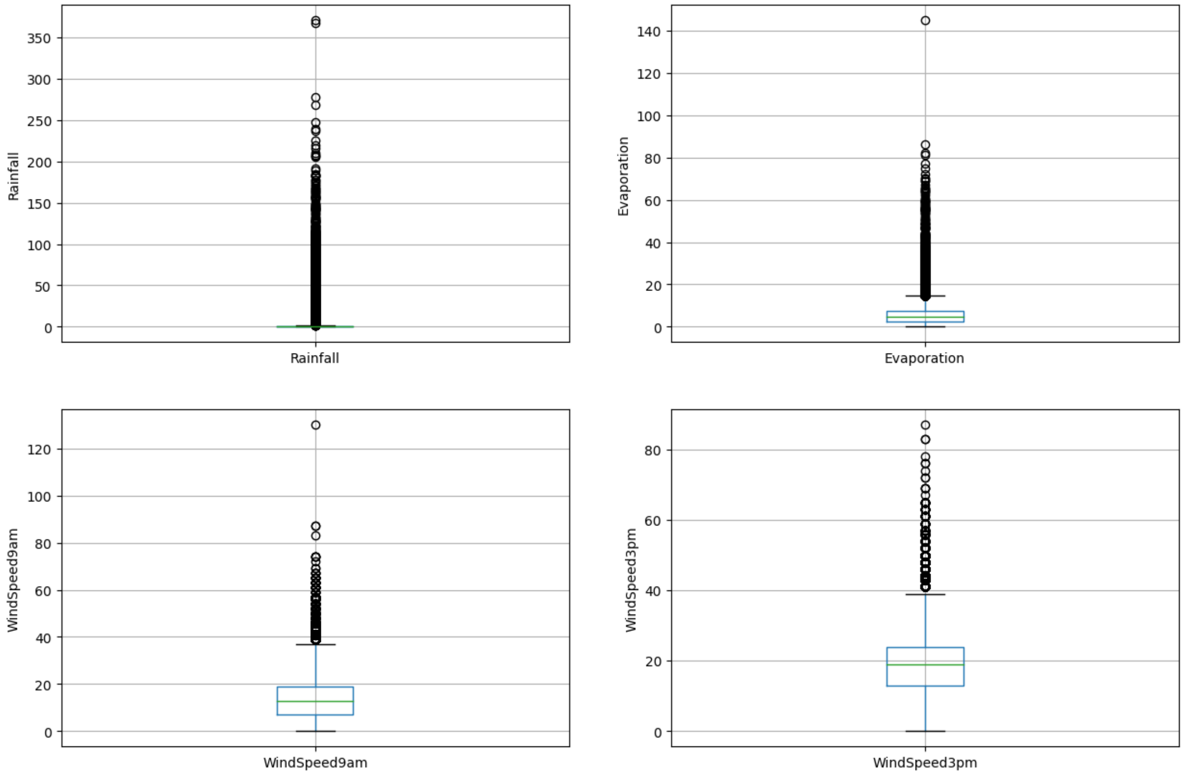
```
        MinTemp   MaxTemp  Rainfall  Evaporation  Sunshine  WindGustSpeed  \
count  143975.0  144199.0  142199.0      82670.0   75625.0       135197.0
mean       12.0      23.0       2.0          5.0       8.0           40.0
std         6.0       7.0       8.0          4.0       4.0           14.0
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      143693.0      142398.0     142806.0     140953.0     130395.0
mean           14.0          19.0         69.0         52.0       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     130432.0   89572.0   86102.0  143693.0  141851.0  145460.0
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  145460.0  145460.0
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 [54]:

```python
# 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')
```

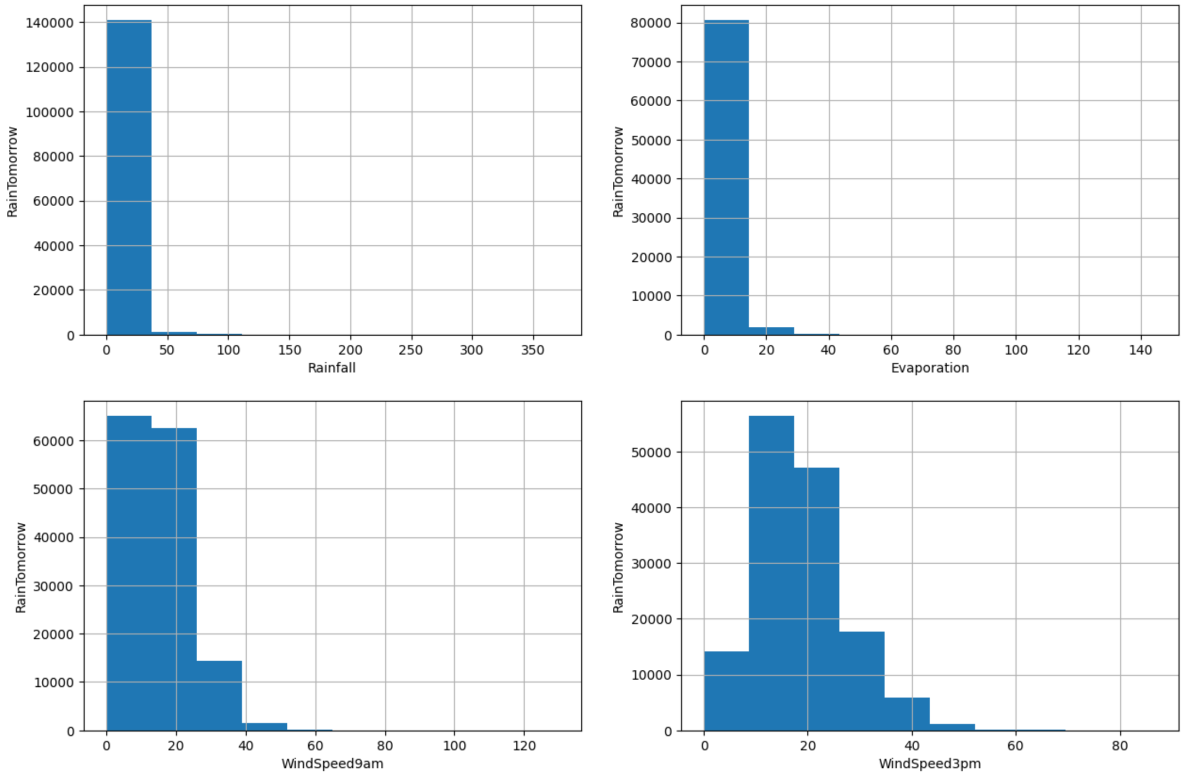
```
plt.subplot(2, 2, 4)
fig = df.boxplot(column='WindSpeed3pm')
fig.set_title('')
fig.set_ylabel('WindSpeed3pm')
```

Out[54]: Text(0, 0.5, 'WindSpeed3pm')



In [55]:
```
# 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[55]: Text(0, 0.5, 'RainTomorrow')



In [56]:
```python
# 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(
```

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

In [57]:
```python
# 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}'.forr
```

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

In [58]:
```python
# 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}'.for
```

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

In [59]:
```python
# 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}'.for
```

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

Declare feature vector and target variable

In [60]:
```
df.dropna(axis=0, subset=['RainTomorrow'], inplace=True)
```

In [61]:
```
X = df.drop(['RainTomorrow'], axis=1)

y = df['RainTomorrow']
```

In [ ]:

Split data into separate training and test set

In [62]:
```
# 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_s
```

In [63]:
```
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

Out[63]:
```
((113754, 24), (28439, 24))
```

Feature Engineering

In [64]:
```
# check data types in X_train

X_train.dtypes
```

Out[64]:
```
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 [65]:
```
# display categorical variables
```

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

categorical
```

Out[65]: ['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday']

In [66]:
```
# display numerical variables

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

numerical
```

Out[66]:
```
['MinTemp',
 'MaxTemp',
 'Rainfall',
 'Evaporation',
 'Sunshine',
 'WindGustSpeed',
 'WindSpeed9am',
 'WindSpeed3pm',
 'Humidity9am',
 'Humidity3pm',
 'Pressure9am',
 'Pressure3pm',
 'Cloud9am',
 'Cloud3pm',
 'Temp9am',
 'Temp3pm',
 'Year',
 'Month',
 'Day']
```

In [67]:
```
# check missing values in numerical variables in X_train

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

Out[67]:
```
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 [68]:
```
# check missing values in numerical variables in X_test

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

Out[68]:
```
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 [69]:
```
# 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 [70]:
```
# impute missing values in X_train and X_test with respective column median in X_tr

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

In [71]:
```
# check again missing values in numerical variables in X_train

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

Out[71]:
```
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 [72]:
```
# check missing values in numerical variables in X_test

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

Out[72]:
```
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
```

Engineering missing values in categorical variables

In [73]:
```
# print percentage of missing values in the categorical variables in training set

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

Out[73]:
```
Location       0.000000
WindGustDir    0.065114
WindDir9am     0.070134
WindDir3pm     0.026443
RainToday      0.010013
dtype: float64
```

In [74]:
```
# 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 [75]:
```
# 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)
```

In [76]:
```
# check missing values in categorical variables in X_train

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

Out[76]:
```
Location       0
WindGustDir    0
WindDir9am     0
WindDir3pm     0
RainToday      0
dtype: int64
```

In [77]:
```
# check missing values in categorical variables in X_test

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

Out[77]:
```
Location       0
WindGustDir    0
WindDir9am     0
WindDir3pm     0
RainToday      0
dtype: int64
```

In [78]:
```
# check missing values in X_train

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

Out[78]:
```
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 [79]:
```
# check missing values in X_test

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

Out[79]:
```
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
```

Engineering outliers in numerical variables

In [80]:
```
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)
```

In [81]:
```
X_train.Rainfall.max(), X_test.Rainfall.max()
```

Out[81]: (3.2, 3.2)

In [82]:
```
X_train.Evaporation.max(), X_test.Evaporation.max()
```

Out[82]: (21.8, 21.8)

In [83]:
```
X_train.WindSpeed9am.max(), X_test.WindSpeed9am.max()
```

Out[83]: (55.0, 55.0)

In [84]:
```
X_train.WindSpeed3pm.max(), X_test.WindSpeed3pm.max()
```

Out[84]: (57.0, 57.0)

In [85]:
```
X_train[numerical].describe()
```

Out[85]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpe
count	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.0000
mean	12.193497	23.237216	0.675080	5.151606	8.041154	39.8840
std	6.388279	7.094149	1.183837	2.823707	2.769480	13.1169
min	-8.200000	-4.800000	0.000000	0.000000	0.000000	6.0000
25%	7.600000	18.000000	0.000000	4.000000	8.200000	31.0000
50%	12.000000	22.600000	0.000000	4.800000	8.500000	39.0000
75%	16.800000	28.200000	0.600000	5.400000	8.700000	46.0000
max	33.900000	48.100000	3.200000	21.800000	14.500000	135.0000

Encode categorical variables

In [86]:
```
categorical
```

Out[86]: ['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday']

In [87]:
```
X_train[categorical].head()
```

Out[87]:

	Location	WindGustDir	WindDir9am	WindDir3pm	RainToday
113462	Witchcliffe	S	SSE	S	No
89638	Cairns	ENE	SSE	SE	Yes
138130	AliceSprings	E	NE	N	No
87898	Cairns	ESE	SSE	E	No
16484	Newcastle	W	N	SE	No

In [88]:
```
# 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 [89]:
```
X_train.head()
```

Out[89]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGı
113462	Witchcliffe	13.9	22.6	0.2	4.8	8.5	S	
89638	Cairns	22.4	29.4	2.0	6.0	6.3	ENE	
138130	AliceSprings	9.7	36.2	0.0	11.4	12.3	E	
87898	Cairns	20.5	30.1	0.0	8.8	11.1	ESE	
16484	Newcastle	16.8	29.2	0.0	4.8	8.5	W	

5 rows × 25 columns

In [90]:
```
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 [91]:
```
X_train.head()
```

Out[91]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am
113462	13.9	22.6	0.2	4.8	8.5	41.0	20.0
89638	22.4	29.4	2.0	6.0	6.3	33.0	7.0
138130	9.7	36.2	0.0	11.4	12.3	31.0	15.0
87898	20.5	30.1	0.0	8.8	11.1	37.0	22.0
16484	16.8	29.2	0.0	4.8	8.5	39.0	0.0

5 rows × 118 columns

In [92]:
```
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 [93]:
```
X_test.head()
```

Out[93]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am
88578	17.4	29.0	0.0	3.6	11.1	33.0	11.0
59016	6.8	14.4	0.8	0.8	8.5	46.0	17.0
127049	10.1	15.4	3.2	4.8	8.5	31.0	13.0
120886	14.4	33.4	0.0	8.0	11.6	41.0	9.0
136649	6.8	14.3	3.2	0.2	7.3	28.0	15.0

5 rows × 118 columns

Feature Scaling

In [94]:
```
X_train.describe()
```

Out[94]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpe
count	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.0000
mean	12.193497	23.237216	0.675080	5.151606	8.041154	39.8840
std	6.388279	7.094149	1.183837	2.823707	2.769480	13.1169
min	-8.200000	-4.800000	0.000000	0.000000	0.000000	6.0000
25%	7.600000	18.000000	0.000000	4.000000	8.200000	31.0000
50%	12.000000	22.600000	0.000000	4.800000	8.500000	39.0000
75%	16.800000	28.200000	0.600000	5.400000	8.700000	46.0000
max	33.900000	48.100000	3.200000	21.800000	14.500000	135.0000

8 rows × 118 columns

In [95]:
```
cols = X_train.columns
```

In [96]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

In [97]:
```
X_train = pd.DataFrame(X_train, columns=[cols])
```

In [98]:
```
X_test = pd.DataFrame(X_test, columns=[cols])
```

In [99]:
```
X_train.describe()
```

Out[99]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpe
count	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.0000
mean	0.484406	0.530004	0.210962	0.236312	0.554562	0.2626
std	0.151741	0.134105	0.369949	0.129528	0.190999	0.1016
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	0.375297	0.431002	0.000000	0.183486	0.565517	0.1937
50%	0.479810	0.517958	0.000000	0.220183	0.586207	0.2558
75%	0.593824	0.623819	0.187500	0.247706	0.600000	0.3100
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.0000

8 rows × 118 columns

LogisticRegression Model training

In [100…
```
# 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[100]:
```
▾ LogisticRegression
LogisticRegression(random_state=0, solver='liblinear')
```

Predict results

In [101…
```
y_pred_test = logreg.predict(X_test)

y_pred_test
```

Out[101]:
```
array(['No', 'No', 'No', ..., 'No', 'No', 'Yes'], dtype=object)
```

predict_proba method

In [102…
```
# probability of getting output as 0 - no rain

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

Out[102]:
```
array([0.91387998, 0.83563444, 0.82035815, ..., 0.97675232, 0.79856396,
       0.30735445])
```

In [103…
```
# probability of getting output as 1 - rain

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

Out[103]:
```
array([0.08612002, 0.16436556, 0.17964185, ..., 0.02324768, 0.20143604,
       0.69264555])
```

Check accuracy score

In [104…
```
from sklearn.metrics import accuracy_score

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

```
Model accuracy score: 0.8502
```

In [105…
```
y_pred_train = logreg.predict(X_train)

y_pred_train
```

Out[105]:
```
array(['No', 'No', 'No', ..., 'No', 'No', 'No'], dtype=object)
```

In [106…
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pre
```

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

Check for overfitting and underfitting

In [107…
```
# 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: 0.8502
```

In [108…
```
# fit the Logsitic Regression model with C=100

# instantiate the model
logreg100 = LogisticRegression(C=100, solver='liblinear', random_state=0)


# fit the model
logreg100.fit(X_train, y_train)
```

Out[108]:
```
LogisticRegression
LogisticRegression(C=100, random_state=0, solver='liblinear')
```

In [109…
```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(logreg100.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(logreg100.score(X_test, y_test)))
```

```
Training set score: 0.8478
Test set score: 0.8505
```

In [110…
```
# fit the Logsitic Regression model with C=001
```

```
# instantiate the model
logreg001 = LogisticRegression(C=0.01, solver='liblinear', random_state=0)


# fit the model
logreg001.fit(X_train, y_train)
```

Out[110]:

```
▾ LogisticRegression
LogisticRegression(C=0.01, random_state=0, solver='liblinear')
```

In [111…

```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(logreg001.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(logreg001.score(X_test, y_test)))
```

```
Training set score: 0.8408
Test set score: 0.8448
```

In [ ]:

Linear Regression

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

In [2]:
```
df1 = pd.read_csv("weatherAUS.csv")
```

In [3]:
```
df1.isnull().sum()
```

Out[3]:
```
Date                 0
Location             0
MinTemp           1485
MaxTemp           1261
Rainfall          3261
Evaporation      62790
Sunshine         69835
WindGustDir      10326
WindGustSpeed    10263
WindDir9am       10566
WindDir3pm        4228
WindSpeed9am      1767
WindSpeed3pm      3062
Humidity9am       2654
Humidity3pm       4507
Pressure9am      15065
Pressure3pm      15028
Cloud9am         55888
Cloud3pm         59358
Temp9am           1767
Temp3pm           3609
RainToday         3261
RainTomorrow      3267
dtype: int64
```

In [4]:
```
df1.dropna(subset = ['RainTomorrow'],inplace =True)
```

In [5]:
```
df1['Date'].dtypes
```

Out[5]:
```
dtype('O')
```

We can see that the data type of `Date` variable is object. I will parse the date currently coded as object into datetime format.

In [6]:
```
df1['Date'] = pd.to_datetime(df1['Date'])
```

In [7]:
```
df1['Year'] = df1['Date'].dt.year

df1['Year'].head()
```

Out[7]:
```
0    2008
1    2008
2    2008
3    2008
4    2008
Name: Year, dtype: int64
```

In [8]:
```
df1['Month'] = df1['Date'].dt.month

df1['Month'].head()
```

Out[8]:
```
0    12
1    12
2    12
3    12
4    12
Name: Month, dtype: int64
```

In [9]:
```
df1['Day'] = df1['Date'].dt.day

df1['Day'].head()
```

Out[9]:
```
0    1
1    2
2    3
3    4
4    5
Name: Day, dtype: int64
```

In [10]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 142193 entries, 0 to 145458
Data columns (total 26 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Date           142193 non-null  datetime64[ns]
 1   Location       142193 non-null  object
 2   MinTemp        141556 non-null  float64
 3   MaxTemp        141871 non-null  float64
 4   Rainfall       140787 non-null  float64
 5   Evaporation    81350 non-null   float64
 6   Sunshine       74377 non-null   float64
 7   WindGustDir    132863 non-null  object
 8   WindGustSpeed  132923 non-null  float64
 9   WindDir9am     132180 non-null  object
 10  WindDir3pm     138415 non-null  object
 11  WindSpeed9am   140845 non-null  float64
 12  WindSpeed3pm   139563 non-null  float64
 13  Humidity9am    140419 non-null  float64
 14  Humidity3pm    138583 non-null  float64
 15  Pressure9am    128179 non-null  float64
 16  Pressure3pm    128212 non-null  float64
 17  Cloud9am       88536 non-null   float64
 18  Cloud3pm       85099 non-null   float64
 19  Temp9am        141289 non-null  float64
 20  Temp3pm        139467 non-null  float64
 21  RainToday      140787 non-null  object
 22  RainTomorrow   142193 non-null  object
 23  Year           142193 non-null  int64
 24  Month          142193 non-null  int64
 25  Day            142193 non-null  int64
dtypes: datetime64[ns](1), float64(16), int64(3), object(6)
memory usage: 29.3+ MB
```

In [11]:
```
df1.drop('Date', axis=1, inplace = True)
```

In [12]:
```
df1.head()
```

Out[12]:

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

5 rows × 25 columns

In [13]:
```
def remove_outlier(i,df1):
    q1=df1[i].quantile(0.25)
    q3=df1[i].quantile(0.75)
    iqr=q3-q1
    ll=q1-3*iqr
    ul=q3+3*iqr
    return df1[~((df1[i]<ll) | (df1[i]>ul))]
```

```
col_list = ['MinTemp' ,'MaxTemp' ,'Rainfall','Evaporation','WindGustSpeed','WindSpe
            'Pressure9am', 'Pressure3pm' , 'Temp9am' ,'Temp3pm']

for i in col_list:
    df1 = remove_outlier(i,df1)
```

In [14]:
```
df1.shape
```

Out[14]: (121051, 25)

In [15]:
```
categorical = [col for col in df1.columns if df1[col].dtypes == 'O']
```

In [16]:
```
import category_encoders as ce
encoder2 = ce.OrdinalEncoder(cols=categorical)
df1 = encoder2.fit_transform(df1)
```

In [17]:
```
df1.head()
```

Out[17]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed
0	1	13.4	22.9	0.6	NaN	NaN	1	44.0
1	1	7.4	25.1	0.0	NaN	NaN	2	44.0
2	1	12.9	25.7	0.0	NaN	NaN	3	46.0
3	1	9.2	28.0	0.0	NaN	NaN	4	24.0
4	1	17.5	32.3	1.0	NaN	NaN	1	41.0

5 rows × 25 columns

In [18]:
```
df1.dtypes
```

Out[18]:
```
Location           int32
MinTemp          float64
MaxTemp          float64
Rainfall         float64
Evaporation      float64
Sunshine         float64
WindGustDir        int32
WindGustSpeed    float64
WindDir9am         int32
WindDir3pm         int32
WindSpeed9am     float64
WindSpeed3pm     float64
Humidity9am      float64
Humidity3pm      float64
Pressure9am      float64
Pressure3pm      float64
Cloud9am         float64
Cloud3pm         float64
Temp9am          float64
Temp3pm          float64
RainToday          int32
RainTomorrow       int32
Year               int64
Month              int64
Day                int64
dtype: object
```

In [19]:
```
df1.corr()
```

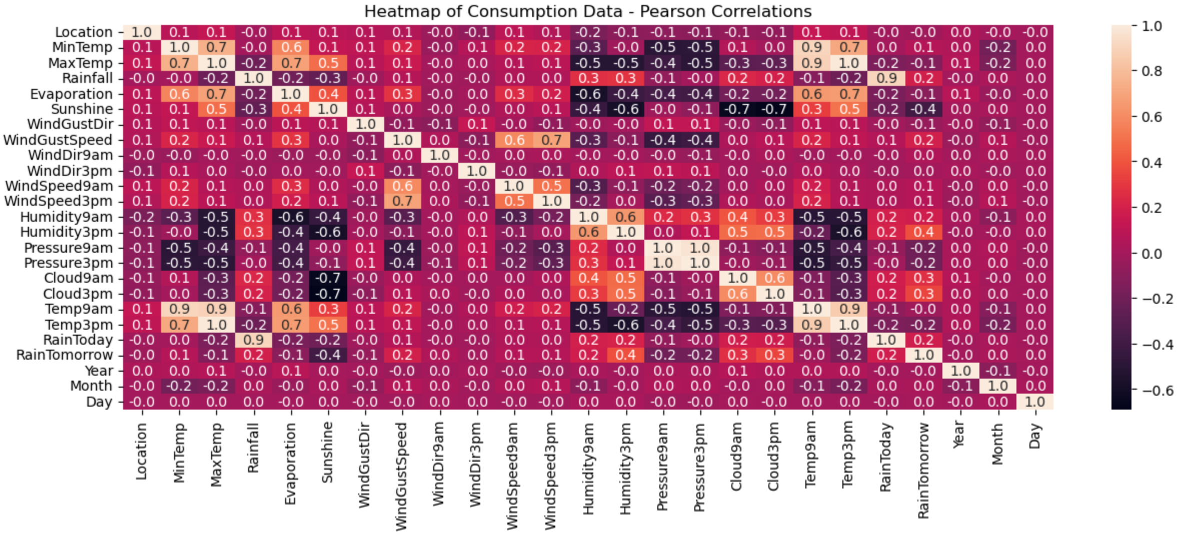
Out[19]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir
Location	1.000000	0.082188	0.117533	-0.004123	0.091788	0.080229	0.067969
MinTemp	0.082188	1.000000	0.743245	-0.008664	0.565739	0.117722	0.106106
MaxTemp	0.117533	0.743245	1.000000	-0.212981	0.679310	0.483600	0.079916
Rainfall	-0.004123	-0.008664	-0.212981	1.000000	-0.206757	-0.254538	-0.023854
Evaporation	0.091788	0.565739	0.679310	-0.206757	1.000000	0.386855	0.083937
Sunshine	0.080229	0.117722	0.483600	-0.254538	0.386855	1.000000	0.068806
WindGustDir	0.067969	0.106106	0.079916	-0.023854	0.083937	0.068806	1.000000
WindGustSpeed	0.050128	0.218491	0.143036	0.065490	0.279760	0.013017	-0.074017
WindDir9am	-0.033991	-0.039051	-0.000636	-0.013690	-0.029226	-0.034535	-0.089680
WindDir3pm	-0.072698	0.056846	0.014997	-0.022975	0.000700	-0.011929	0.123178
WindSpeed9am	0.093215	0.208871	0.063223	0.046736	0.253175	0.040674	-0.008284
WindSpeed3pm	0.054538	0.206275	0.087922	0.045687	0.179118	0.078666	-0.074224
Humidity9am	-0.156558	-0.291315	-0.516474	0.262790	-0.582430	-0.439544	-0.025111
Humidity3pm	-0.094278	-0.024130	-0.517003	0.274678	-0.429953	-0.584378	-0.003455
Pressure9am	-0.088434	-0.495142	-0.420757	-0.063110	-0.381747	-0.025264	0.109496
Pressure3pm	-0.096048	-0.490661	-0.494006	-0.001805	-0.391071	-0.080734	0.113731
Cloud9am	-0.061713	0.056368	-0.282314	0.222417	-0.183739	-0.654960	-0.041749
Cloud3pm	-0.072157	0.000219	-0.268410	0.200801	-0.190198	-0.688850	-0.079204
Temp9am	0.127117	0.903366	0.884857	-0.118867	0.644035	0.318530	0.093154
Temp3pm	0.107554	0.715376	0.984443	-0.216747	0.660576	0.505064	0.082829
RainToday	-0.014824	0.002819	-0.150403	0.903416	-0.161984	-0.198832	-0.015089
RainTomorrow	-0.017184	0.063351	-0.125049	0.184564	-0.105448	-0.397923	-0.052188
Year	0.032071	0.039499	0.059224	-0.008338	0.072619	0.004325	-0.004945
Month	-0.000835	-0.190647	-0.156971	0.020455	-0.037800	0.010696	-0.067340
Day	-0.003642	0.003236	0.002283	0.000408	-0.009165	0.000113	-0.010692

25 rows × 25 columns

In [20]:

```
fig, ax = plt.subplots(figsize=(15, 5))
correlations = df1.corr()
# Rohit Bhimani
# annot=True displays the correlation values

sns.heatmap(correlations, annot=True , fmt=".1f").set(title='Heatmap of Consumptior
```



In [21]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df2 = scaler.fit_transform(df1)
df1 = pd.DataFrame(df2, columns=df1.columns)
```

In [22]:
```
X1 = df1.drop(['RainTomorrow'], axis=1)

y1 = df1['RainTomorrow']
```

Split data into separate training and test set

In [23]:
```
from sklearn.model_selection import train_test_split

X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size = 0.2, ra
```

In [24]:
```
X1_train.shape, X1_test.shape
```

Out[24]:
```
((96840, 24), (24211, 24))
```

Feature Engineering

In [25]:
```
categorical1 = [col for col in X1_train.columns if X1_train[col].dtypes == 'O']
```

In [26]:
```
numerical1 = [col for col in X1_train.columns if X1_train[col].dtypes != 'O']
```

In [27]:
```
# impute missing values in X_train and X_test with respective column median in X_tr

for df2 in [X1_train, X1_test]:
    for col in numerical1:
        col_median=X1_train[col].median()
        df2[col].fillna(col_median, inplace=True)
```

Engineering missing values in categorical variables

In [28]:
```
# impute missing categorical variables with most frequent value
```

```
for df3 in [X1_train, X1_test]:
    df3['WindGustDir'].fillna(X1_train['WindGustDir'].mode()[0], inplace=True)
    df3['WindDir9am'].fillna(X1_train['WindDir9am'].mode()[0], inplace=True)
    df3['WindDir3pm'].fillna(X1_train['WindDir3pm'].mode()[0], inplace=True)
    df3['RainToday'].fillna(X1_train['RainToday'].mode()[0], inplace=True)
```

In [29]:
```
X1_train[numerical1].describe()
```

Out[29]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	Winc
count	96840.000000	96840.000000	96840.000000	96840.000000	96840.000000	96840.000000	9684(
mean	0.492659	0.482982	0.540338	0.078472	0.240927	0.586001	(
std	0.291248	0.151736	0.137494	0.193910	0.122106	0.185015	(
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	(
25%	0.229167	0.375000	0.437743	0.000000	0.200000	0.606897	(
50%	0.479167	0.478774	0.531128	0.000000	0.227273	0.620690	(
75%	0.750000	0.591981	0.636187	0.000000	0.254545	0.634483	(
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	'

8 rows × 24 columns

In [30]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X1_train, y1_train)
```

Out[30]:
```
LinearRegression()
```

In [31]:
```
regressor.intercept_
```

Out[31]:
```
0.08721288900808244
```

In [32]:
```
regressor.coef_
```

Out[32]:
```
array([-1.23849086e-05, -1.11245883e-01,  7.00075166e-02,  1.39751818e-02,
        5.21667485e-02, -2.82094769e-01, -9.82690533e-03,  6.28236452e-01,
        2.42754921e-02, -8.76561506e-03, -2.55242522e-02, -2.79161474e-01,
       -8.11894216e-02,  7.75396520e-01,  9.81618245e-01, -1.51822467e+00,
       -1.89887743e-02,  9.13283491e-02, -1.34434249e-01,  1.68959249e-01,
        1.89300020e-01,  1.40405979e-02,  1.38126912e-02, -1.56348021e-03])
```

In [33]:
```
y1_pred = regressor.predict(X1_test)
```

In [34]:
```
regressor.score(X1_train , y1_train)
```

Out[34]:
```
0.25824604814730534
```

In [35]:
```
regressor.score(X1_test , y1_test)
```

Out[35]:
```
0.25889665229486236
```

In [36]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error

mae = mean_absolute_error(y1_test, y1_pred)
```

```
mse = mean_squared_error(y1_test, y1_pred)
error2 = rmse = np.sqrt(mse)

print(f'Mean absolute error: {mae:.2f}')
print(f'Mean squared error: {mse:.2f}')
print(f'Root mean squared error: {rmse:.2f}')
```

```
Mean absolute error: 0.23
Mean squared error: 0.11
Root mean squared error: 0.33
```

In [36]:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error

mae = mean_absolute_error(y1_test, y1_pred)
mse = mean_squared_error(y1_test, y1_pred)
error2 = rmse = np.sqrt(mse)

print(f'Mean absolute error: {mae:.2f}')
print(f'Mean squared error: {mse:.2f}')
print(f'Root mean squared error: {rmse:.2f}')
```

```
Mean absolute error: 0.23
Mean squared error: 0.11
Root mean squared error: 0.33
```

In [ ]: