

# 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)

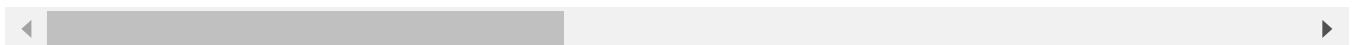
```
In [6]: # preview the dataset

df.head()
```

Out[6]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	10.0
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	10.0
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	10.0
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	10.0
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	10.0

5 rows × 10 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', 'WindDir3pm', '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', 'Sunshine', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', '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	WindSpeed3pm
0	13.4	22.9	0.6	NaN	NaN	44.0	20.0	15.0
1	7.4	25.1	0.0	NaN	NaN	44.0	4.0	15.0
2	12.9	25.7	0.0	NaN	NaN	46.0	19.0	15.0
3	9.2	28.0	0.0	NaN	NaN	24.0	11.0	15.0
4	17.5	32.3	1.0	NaN	NaN	41.0	7.0	15.0

## 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]: *# draw boxplots to visualize outliers*

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

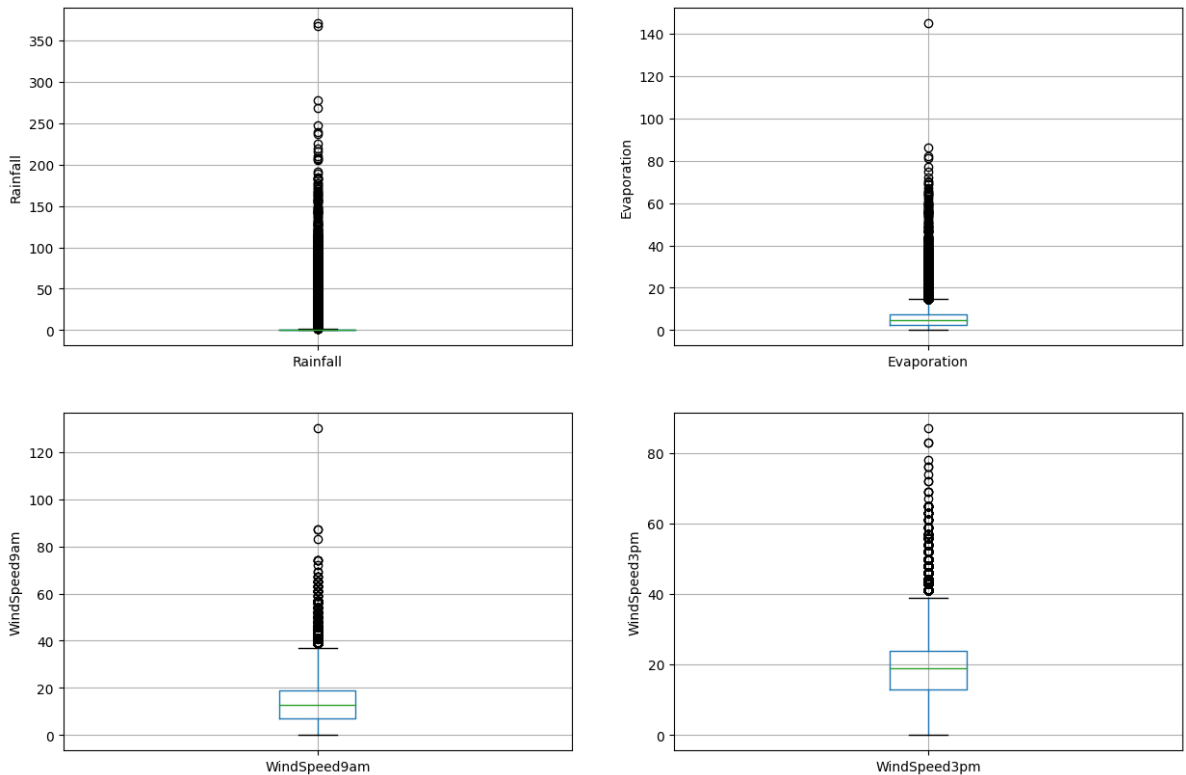
plt.subplot(2, 2, 1)
fig = df.boxplot(column='Rainfall')
fig.set_title('')
fig.set_ylabel('Rainfall')

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))

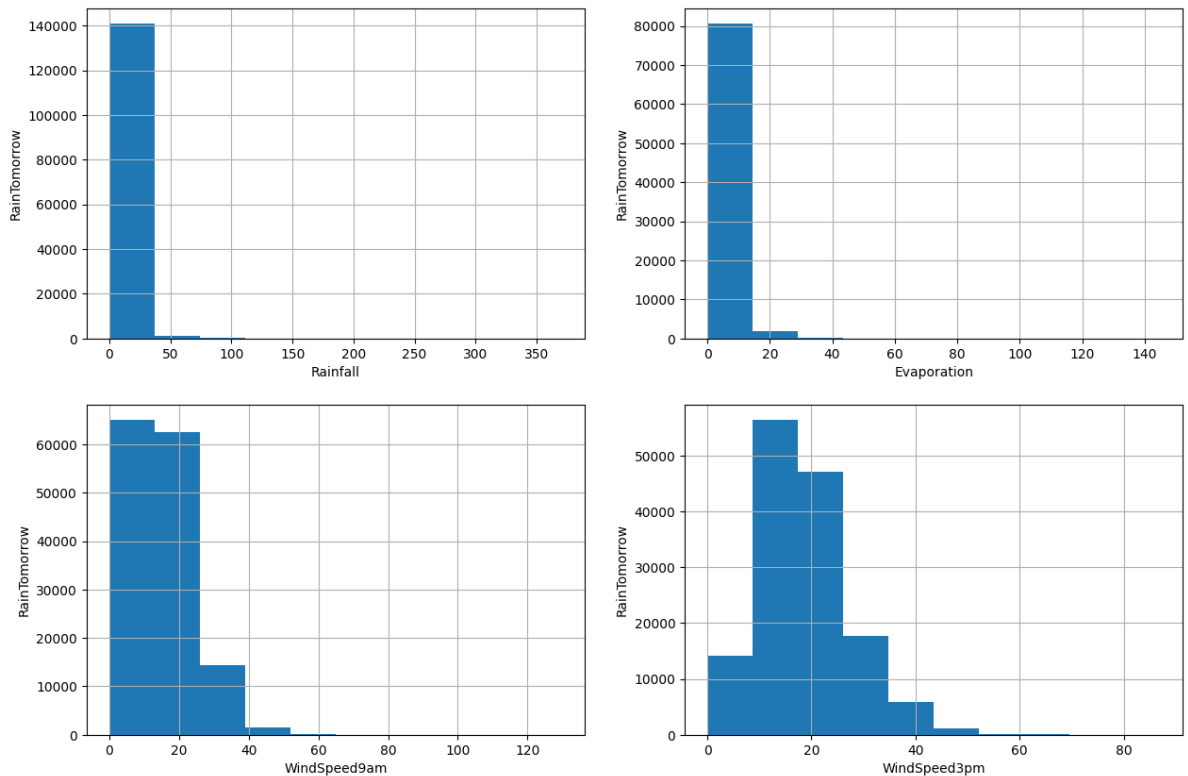
plt.subplot(2, 2, 1)
fig = df.Rainfall.hist(bins=10)
fig.set_xlabel('Rainfall')
fig.set_ylabel('RainTomorrow')

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]: *# 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, upperboundary=Upper_fence))
```

Rainfall outliers are values < -2.4000000000000004 or > 3.2

In [57]: *# 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, upperboundary=Upper_fence))
```

Evaporation outliers are values < -11.800000000000002 or > 21.800000000000004

In [58]: *# 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, upperboundary=Upper_fence))
```

WindSpeed9am outliers are values < -29.0 or > 55.0

In [59]: *# 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, upperboundary=Upper_fence))
```

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_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)
```

```
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	WindGustSpee
<b>count</b>	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000
<b>mean</b>	12.193497	23.237216	0.675080	5.151606	8.041154	39.884000
<b>std</b>	6.388279	7.094149	1.183837	2.823707	2.769480	13.116900
<b>min</b>	-8.200000	-4.800000	0.000000	0.000000	0.000000	6.000000
<b>25%</b>	7.600000	18.000000	0.000000	4.000000	8.200000	31.000000
<b>50%</b>	12.000000	22.600000	0.000000	4.800000	8.500000	39.000000
<b>75%</b>	16.800000	28.200000	0.600000	5.400000	8.700000	46.000000
<b>max</b>	33.900000	48.100000	3.200000	21.800000	14.500000	135.000000

## 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
<b>113462</b>	Witchcliffe	S	SSE	S	No
<b>89638</b>	Cairns	ENE	SSE	SE	Yes
<b>138130</b>	AliceSprings	E	NE	N	No
<b>87898</b>	Cairns	ESE	SSE	E	No
<b>16484</b>	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	WindGustSpeed
<b>113462</b>	Witchcliffe	13.9	22.6	0.2	4.8	8.5	S	41.0
<b>89638</b>	Cairns	22.4	29.4	2.0	6.0	6.3	ENE	33.0
<b>138130</b>	AliceSprings	9.7	36.2	0.0	11.4	12.3	E	31.0
<b>87898</b>	Cairns	20.5	30.1	0.0	8.8	11.1	ESE	37.0
<b>16484</b>	Newcastle	16.8	29.2	0.0	4.8	8.5	W	39.0

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
<b>113462</b>	13.9	22.6	0.2	4.8	8.5	41.0	20.0
<b>89638</b>	22.4	29.4	2.0	6.0	6.3	33.0	7.0
<b>138130</b>	9.7	36.2	0.0	11.4	12.3	31.0	15.0
<b>87898</b>	20.5	30.1	0.0	8.8	11.1	37.0	22.0
<b>16484</b>	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
<b>88578</b>	17.4	29.0	0.0	3.6	11.1	33.0	11.0
<b>59016</b>	6.8	14.4	0.8	0.8	8.5	46.0	17.0
<b>127049</b>	10.1	15.4	3.2	4.8	8.5	31.0	13.0
<b>120886</b>	14.4	33.4	0.0	8.0	11.6	41.0	9.0
<b>136649</b>	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	WindGustSpeed
<b>count</b>	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000
<b>mean</b>	12.193497	23.237216	0.675080	5.151606	8.041154	39.884000
<b>std</b>	6.388279	7.094149	1.183837	2.823707	2.769480	13.116900
<b>min</b>	-8.200000	-4.800000	0.000000	0.000000	0.000000	6.000000
<b>25%</b>	7.600000	18.000000	0.000000	4.000000	8.200000	31.000000
<b>50%</b>	12.000000	22.600000	0.000000	4.800000	8.500000	39.000000
<b>75%</b>	16.800000	28.200000	0.600000	5.400000	8.700000	46.000000
<b>max</b>	33.900000	48.100000	3.200000	21.800000	14.500000	135.000000

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	WindGustSpeed
<b>count</b>	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000
<b>mean</b>	0.484406	0.530004	0.210962	0.236312	0.554562	0.262600
<b>std</b>	0.151741	0.134105	0.369949	0.129528	0.190999	0.101600
<b>min</b>	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
<b>25%</b>	0.375297	0.431002	0.000000	0.183486	0.565517	0.193700
<b>50%</b>	0.479810	0.517958	0.000000	0.220183	0.586207	0.255800
<b>75%</b>	0.593824	0.623819	0.187500	0.247706	0.600000	0.310000
<b>max</b>	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

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

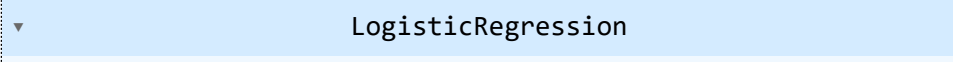
In [108... *# fit the Logsitic Regression model with C=100*

*# instantiate the model*

```
logreg100 = LogisticRegression(C=100, solver='liblinear', random_state=0)
```

*# fit the model*

```
logreg100.fit(X_train, y_train)
```

Out[108]: 

LogisticRegression(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 [ ]: