[14]:
```
!pip install scikit-learn --quiet
```

DOWNLOADING DATA:

[15]:
```
!pip install opendatasets --upgrade --quiet
```

[16]:
```
import opendatasets as od
```

[17]:
```
od.version()
```

[17]:
```
'0.1.22'
```

[18]:
```
dataset_url = 'https://www.kaggle.com/jsphyg/weather-dataset-rattle-package'
```

[19]:
```
od.download(dataset_url)
```

```
Skipping, found downloaded files in ".\weather-dataset-rattle-package" (use force=True to force download)
```

[20]:
```
import os
```

[21]:
```
data_dir = './weather-dataset-rattle-package'
```

[22]:
```
os.listdir(data_dir)
```

[22]:
```
['weatherAUS.csv']
```

[23]:
```
train_csv = data_dir + '/weatherAUS.csv'
```

[24]:
```
train_csv
```

[24]:
```
'./weather-dataset-rattle-package/weatherAUS.csv'
```

[25]:
```
!pip install pandas --quiet
```

[26]:
```
import pandas as pd
```

[27]:
```
raw_df = pd.read_csv(train_csv)
```

[28]:
```
raw_df
```

[28]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity9am	Humidity3pm	Pressure9an
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	...	71.0	22.0	1007.
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	44.0	25.0	1010.
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	...	38.0	30.0	1007.
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	...	45.0	16.0	1017.
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	...	82.0	33.0	1010.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
145455	2017-06-21	Uluru	2.8	23.4	0.0	NaN	NaN	E	31.0	SE	...	51.0	24.0	1024.
145456	2017-06-22	Uluru	3.6	25.3	0.0	NaN	NaN	NNW	22.0	SE	...	56.0	21.0	1023.
145457	2017-06-23	Uluru	5.4	26.9	0.0	NaN	NaN	N	37.0	SE	...	53.0	24.0	1021.
145458	2017-06-24	Uluru	7.8	27.0	0.0	NaN	NaN	SE	28.0	SSE	...	51.0	24.0	1019.
145459	2017-06-25	Uluru	14.9	NaN	0.0	NaN	NaN	NaN	NaN	ESE	...	62.0	36.0	1020.

145460 rows × 23 columns

[29]:
```
raw_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
```

[30]:
```
raw_df.dropna(subset=['RainToday','RainTomorrow'], inplace=True)
```

[31]:
```
raw_df.info()
```

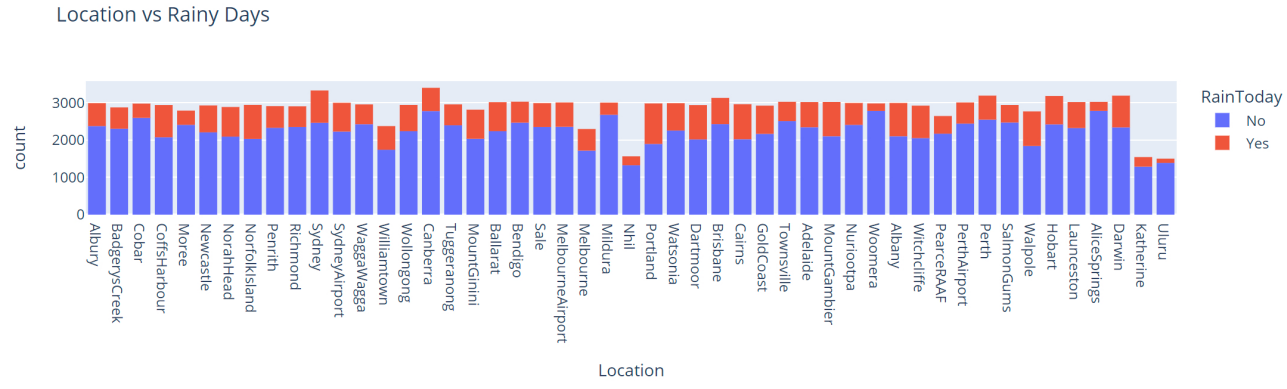
```
<class 'pandas.core.frame.DataFrame'>
Index: 140787 entries, 0 to 145458
Data columns (total 23 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Date           140787 non-null  object
 1   Location       140787 non-null  object
 2   MinTemp        140319 non-null  float64
 3   MaxTemp        140480 non-null  float64
 4   Rainfall       140787 non-null  float64
 5   Evaporation    81093 non-null   float64
 6   Sunshine       73982 non-null   float64
 7   WindGustDir    131624 non-null  object
 8   WindGustSpeed  131682 non-null  float64
 9   WindDir9am     131127 non-null  object
 10  WindDir3pm     137117 non-null  object
 11  WindSpeed9am   139732 non-null  float64
 12  WindSpeed3pm   138256 non-null  float64
 13  Humidity9am    139270 non-null  float64
 14  Humidity3pm    137286 non-null  float64
 15  Pressure9am    127044 non-null  float64
 16  Pressure3pm    127018 non-null  float64
 17  Cloud9am       88162 non-null   float64
 18  Cloud3pm       84693 non-null   float64
 19  Temp9am        140131 non-null  float64
 20  Temp3pm        138163 non-null  float64
 21  RainToday      140787 non-null  object
 22  RainTomorrow   140787 non-null  object
dtypes: float64(16), object(7)
memory usage: 25.8+ MB
```

EXPLORATORY DATA ANALYSIS AND VISUALIZATION:

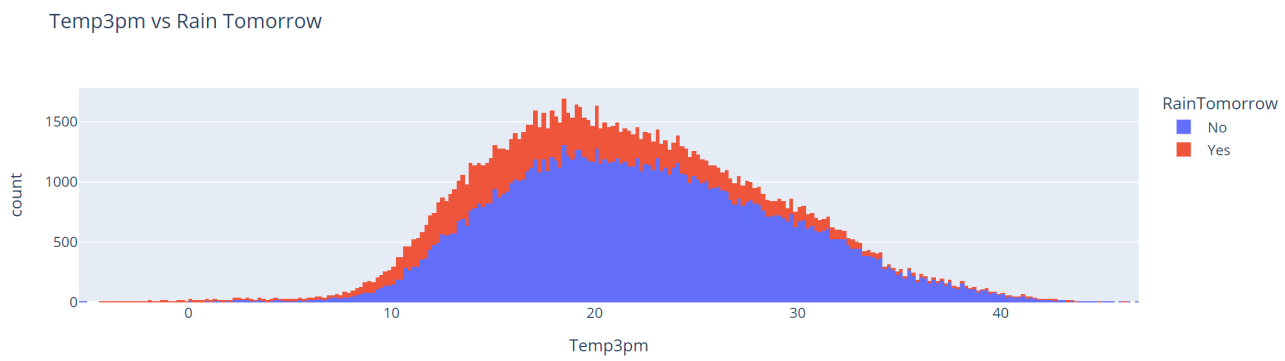
[32]:
```
!pip install plotly matplotlib seaborn --quiet
```

[35]:
```
import plotly.express as px
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

sns.set_style('darkgrid')
matplotlib.rcParams['font.size'] = 14
matplotlib.rcParams['figure.figsize'] = (10, 6)
matplotlib.rcParams['figure.facecolor'] = '#00000000'
```

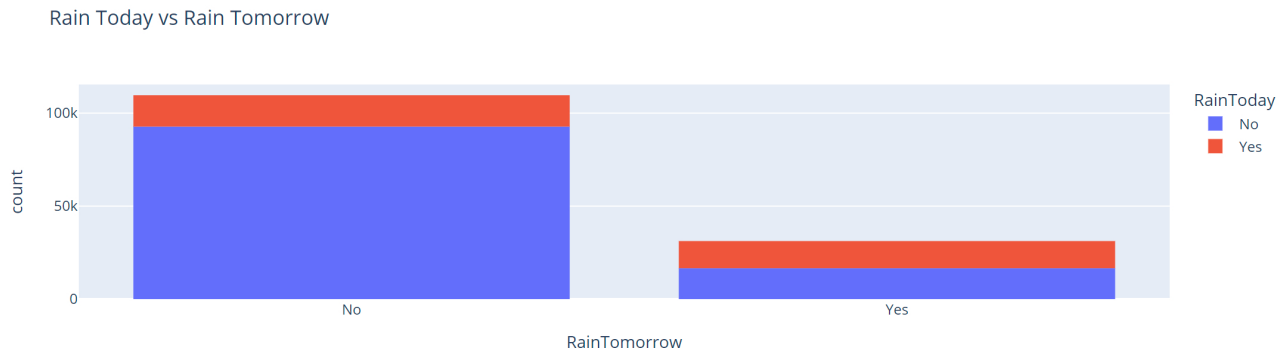
[37]:
```
px.histogram(raw_df, x = 'Location', title = 'Location vs Rainy Days', color= 'RainToday')
```



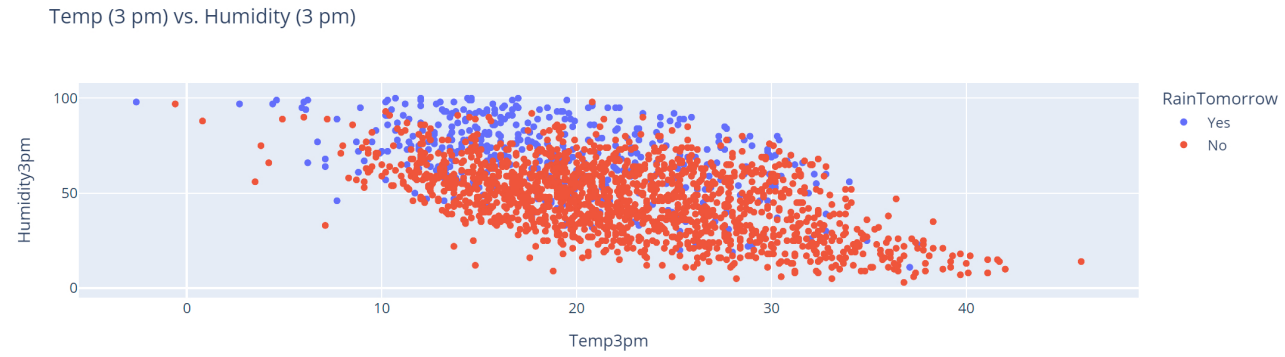
[40]:
```
px.histogram(raw_df, x = 'Temp3pm', title = 'Temp3pm vs Rain Tomorrow', color= 'RainTomorrow')
```



[41]:
```
px.histogram(raw_df, x = 'RainTomorrow', title = 'Rain Today vs Rain Tomorrow', color= 'RainToday')
```



[42]:
```
px.scatter(raw_df.sample(2000),
           title='Temp (3 pm) vs. Humidity (3 pm)',
           x='Temp3pm',
           y='Humidity3pm',
           color='RainTomorrow')
```



[43]:
```
use_sample=False
sample_fraction=0.1
if use_sample:
    raw_df= raw_df.sample(frac = sample_fraction).copy()
```

TRAINING, VALIDATION AND TEST SETS:

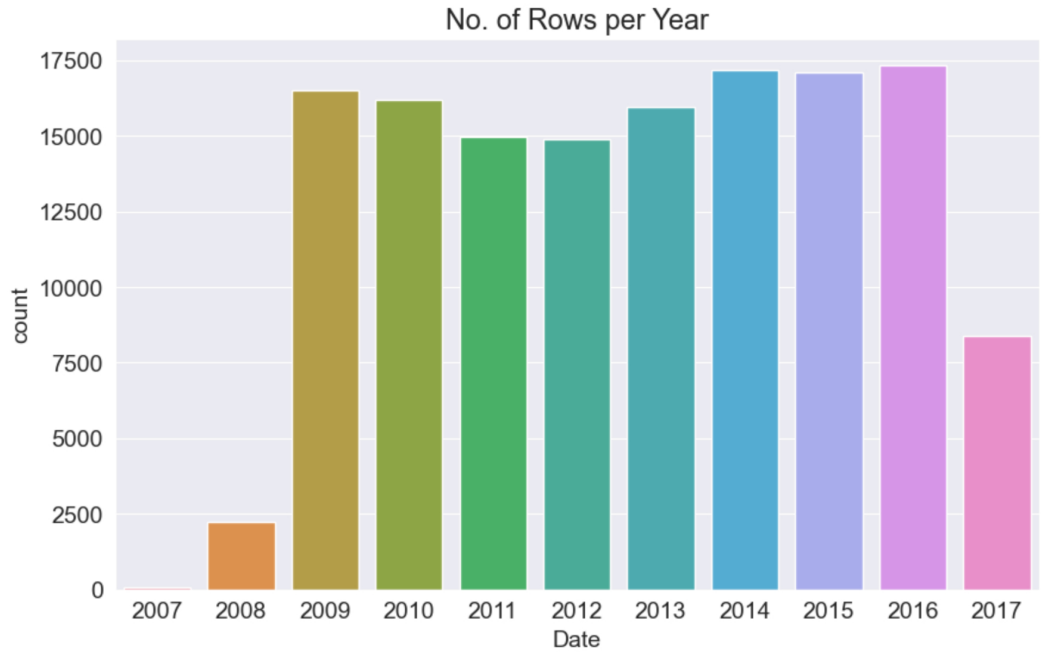
[44]:
```
from sklearn.model_selection import train_test_split
```

[45]:
```
train_val_df, test_df = train_test_split(raw_df, test_size=0.2, random_state = 42)
train_df, val_df = train_test_split(train_val_df, test_size = 0.25, random_state = 42)
```

[46]:
```
print ('train_df.shape:', train_df.shape)
print ('val_df.shape:', val_df.shape)
print ('test_df.shape:', test_df.shape)
```

```
train_df.shape: (84471, 23)
val_df.shape: (28158, 23)
test_df.shape: (28158, 23)
```

[50]:
```
plt.title('No. of Rows per Year')
sns.countplot(x=pd.to_datetime(raw_df.Date).dt.year);
```



[51]:
```
year = pd.to_datetime(raw_df.Date).dt.year

train_df = raw_df[year<2015]
val_df = raw_df[year==2015]
test_df = raw_df[year>2015]
```

[52]:
```
print ('train_df.shape:', train_df.shape)
print ('val_df.shape:', val_df.shape)
print ('test_df.shape:', test_df.shape)
```

```
train_df.shape: (97988, 23)
val_df.shape: (17089, 23)
test_df.shape: (25710, 23)
```

INPUT AND TARGET COLUMNS:

[57]:
```
input_cols = list(train_df.columns)[1:-1]
target_col = 'RainTomorrow'
```

[61]:
```
print (input_cols)
```

```
['Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm', 'RainToday']
```

[62]:
```
target_col
```

[62]:
```
'RainTomorrow'
```

[64]:
```
train_inputs = train_df[input_cols].copy()
train_targets = train_df[target_col].copy()
```

[65]:
```
train_inputs
```

[65]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	...	WindSpeed3pm	Humidity9am	Hu
0	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	WNW	...	24.0	71.0	
1	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	WSW	...	22.0	44.0	
2	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	WSW	...	26.0	38.0	
3	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	E	...	9.0	45.0	
4	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	NW	...	20.0	82.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
144548	Uluru	16.9	33.2	0.0	NaN	NaN	SSE	43.0	ESE	SSE	...	26.0	22.0	
144549	Uluru	15.1	36.8	0.0	NaN	NaN	NE	31.0	ENE	SW	...	20.0	16.0	
144550	Uluru	17.3	37.8	0.0	NaN	NaN	ESE	39.0	ESE	SSE	...	9.0	15.0	
144551	Uluru	20.1	38.5	0.0	NaN	NaN	ESE	43.0	ESE	SSW	...	17.0	22.0	
144552	Uluru	22.5	39.6	0.0	NaN	NaN	WNW	76.0	ENE	SSW	...	13.0	16.0	

97988 rows × 21 columns

[67]:
```
train_targets
```

[67]:
```
0         No
1         No
2         No
3         No
4         No
          ..
144548    No
144549    No
144550    No
144551    No
144552    No
Name: RainTomorrow, Length: 97988, dtype: object
```

[68]:
```
val_inputs = val_df[input_cols].copy()
val_targets = val_df[target_col].copy()
```

[69]:
```
test_inputs = test_df[input_cols].copy()
test_targets = test_df[target_col].copy()
```

[70]:
```
!pip install numpy --quiet
```

[71]:
```
import numpy as np
```

[72]:
```
numerical_cols = train_inputs.select_dtypes(include=np.number).columns.tolist()
categorical_cols = train_inputs.select_dtypes('object').columns.tolist()
```

[73]:
```
numerical_cols
```

[73]:
```
['MinTemp',
 'MaxTemp',
 'Rainfall',
 'Evaporation',
 'Sunshine',
 'WindGustSpeed',
 'WindSpeed9am',
 'WindSpeed3pm',
 'Humidity9am',
 'Humidity3pm',
 'Pressure9am',
 'Pressure3pm',
 'Cloud9am',
 'Cloud3pm',
 'Temp9am',
 'Temp3pm']
```

[74]:
```
categorical_cols
```

[74]:
```
['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday']
```

[76]:
```
train_inputs[numerical_cols].describe()
```

[76]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am
count	97674.000000	97801.000000	97988.000000	61657.000000	57942.000000	91160.000000	97114.000000	96919.000000	96936.000000	96872.000000	88876.00000
mean	12.007831	23.022202	2.372935	5.289991	7.609004	40.215873	14.092263	18.764608	68.628745	51.469547	1017.51373
std	6.347175	6.984397	8.518819	3.952010	3.788813	13.697967	8.984203	8.872398	19.003097	20.756113	7.07251
min	-8.500000	-4.100000	0.000000	0.000000	0.000000	6.000000	0.000000	0.000000	0.000000	0.000000	980.50000
25%	7.500000	17.900000	0.000000	2.600000	4.800000	31.000000	7.000000	13.000000	57.000000	37.000000	1012.80000
50%	11.800000	22.400000	0.000000	4.600000	8.500000	39.000000	13.000000	19.000000	70.000000	52.000000	1017.50000
75%	16.600000	27.900000	0.800000	7.200000	10.600000	48.000000	19.000000	24.000000	83.000000	66.000000	1022.30000
max	33.900000	48.100000	371.000000	82.400000	14.300000	135.000000	87.000000	87.000000	100.000000	100.000000	1041.00000

[77]:
```
train_inputs[categorical_cols].nunique()
```

[77]:
```
Location       49
WindGustDir    16
WindDir9am     16
WindDir3pm     16
RainToday       2
dtype: int64
```

IMPUTING MISSING NUMERIC DATA:

[78]:
```
from sklearn.impute import SimpleImputer
```

[79]:
```
imputer = SimpleImputer(strategy = 'mean')
```

[80]:
```
raw_df[numerical_cols].isna().sum()
```

[80]:
```
MinTemp           468
MaxTemp           307
Rainfall            0
Evaporation     59694
Sunshine        66805
WindGustSpeed    9105
WindSpeed9am     1055
WindSpeed3pm     2531
Humidity9am      1517
Humidity3pm      3501
```

```
Pressure9am      13743
Pressure3pm      13769
Cloud9am         52625
Cloud3pm         56094
Temp9am            656
Temp3pm           2624
dtype: int64
```

[81]:
```
train_df[numerical_cols].isna().sum()
```

[81]:
```
MinTemp            314
MaxTemp            187
Rainfall             0
Evaporation      36331
Sunshine         40046
WindGustSpeed     6828
WindSpeed9am       874
WindSpeed3pm      1069
Humidity9am       1052
Humidity3pm       1116
Pressure9am       9112
Pressure3pm       9131
Cloud9am         34988
Cloud3pm         36022
Temp9am            574
Temp3pm            596
dtype: int64
```

[82]:
```
imputer.fit(raw_df[numerical_cols])
```

[82]:
```
SimpleImputer
SimpleImputer()
```

[83]:
```
list(imputer.statistics_)
```

[83]:
```
[12.18482386562048,
 23.235120301822324,
 2.349974074310839,
 5.472515506887154,
 7.630539861047281,
 39.97051988882308,
 13.990496092519967,
 18.631140782316862,
 68.82683277087672,
 51.44928834695453,
 1017.6545771543717,
 1015.2579625879797,
 4.431160817585808,
 4.499250233195188,
 16.98706638787991,
 21.69318269001107]
```

[84]:
```
train_inputs[numerical_cols] = imputer.transform(train_inputs[numerical_cols])
val_inputs[numerical_cols] = imputer.transform(val_inputs[numerical_cols])
test_inputs[numerical_cols] = imputer.transform(test_inputs[numerical_cols])
```

[85]:
```
train_inputs[numerical_cols].isna().sum()
```

[85]:
```
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
dtype: int64
```

SCALING NUMERIC FEATURES:

[86]:
```
from sklearn.preprocessing import MinMaxScaler
```

[87]:
```
scaler = MinMaxScaler()
```

[88]:
```
scaler.fit(raw_df[numerical_cols])
```

[88]:
```
MinMaxScaler
MinMaxScaler()
```

[90]:
```
print ('Minimum:')
list(scaler.data_min_)
```

Minimum:

[90]:
```
[-8.5,
 -4.8,
 0.0,
```

```
0.0,
0.0,
6.0,
0.0,
0.0,
0.0,
0.0,
980.5,
977.1,
0.0,
0.0,
-7.2,
-5.4]
```

[91]:
```
print('Maximum:')
list(scaler.data_max_)
```

```
Maximum:
```

[91]:
```
[33.9,
 48.1,
 371.0,
 145.0,
 14.5,
 135.0,
 130.0,
 87.0,
 100.0,
 100.0,
 1041.0,
 1039.6,
 9.0,
 9.0,
 40.2,
 46.7]
```

[92]:
```
train_inputs[numerical_cols] = scaler.transform(train_inputs[numerical_cols])
val_inputs[numerical_cols] = scaler.transform(val_inputs[numerical_cols])
test_inputs[numerical_cols] = scaler.transform(test_inputs[numerical_cols])
```

[93]:
```
train_inputs[numerical_cols].describe()
```

[93]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am
count	97988.000000	97988.000000	97988.000000	97988.000000	97988.000000	97988.000000	97988.000000	97988.000000	97988.000000	97988.000000	97988.00000
mean	0.483689	0.525947	0.006396	0.036949	0.525366	0.265107	0.108395	0.215668	0.686309	0.514693	0.61201
std	0.149458	0.131904	0.022962	0.021628	0.200931	0.102420	0.068800	0.101424	0.189008	0.206376	0.11133
min	0.000000	0.013233	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	0.377358	0.429112	0.000000	0.026207	0.517241	0.193798	0.053846	0.149425	0.570000	0.370000	0.54380
50%	0.478774	0.514178	0.000000	0.037741	0.526244	0.255814	0.100000	0.218391	0.690000	0.520000	0.61412
75%	0.591981	0.618147	0.002156	0.038621	0.634483	0.310078	0.146154	0.275862	0.830000	0.650000	0.68264
max	1.000000	1.000000	1.000000	0.568276	0.986207	1.000000	0.669231	1.000000	1.000000	1.000000	1.00000

[99]:
```
train_inputs[numerical_cols]
```

[99]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm
0	0.516509	0.523629	0.001617	0.037741	0.526244	0.294574	0.153846	0.275862	0.71	0.22	0.449587	0.480
1	0.375000	0.565217	0.000000	0.037741	0.526244	0.294574	0.030769	0.252874	0.44	0.25	0.497521	0.491
2	0.504717	0.576560	0.000000	0.037741	0.526244	0.310078	0.146154	0.298851	0.38	0.30	0.447934	0.505
3	0.417453	0.620038	0.000000	0.037741	0.526244	0.139535	0.084615	0.103448	0.45	0.16	0.613223	0.571
4	0.613208	0.701323	0.002695	0.037741	0.526244	0.271318	0.053846	0.229885	0.82	0.33	0.500826	0.462
...	...	...	...	...	...	...	...	...	...	...	...	...
144548	0.599057	0.718336	0.000000	0.037741	0.526244	0.286822	0.184615	0.298851	0.22	0.13	0.555372	0.523
144549	0.556604	0.786389	0.000000	0.037741	0.526244	0.193798	0.146154	0.229885	0.16	0.08	0.530579	0.488
144550	0.608491	0.805293	0.000000	0.037741	0.526244	0.255814	0.200000	0.103448	0.15	0.08	0.519008	0.494
144551	0.674528	0.818526	0.000000	0.037741	0.526244	0.286822	0.215385	0.195402	0.22	0.09	0.553719	0.513
144552	0.731132	0.839319	0.000000	0.037741	0.526244	0.542636	0.230769	0.149425	0.16	0.09	0.522314	0.465

97988 rows × 16 columns

ENCODING CATEGORICAL DATA:

[100]:
```
from sklearn.preprocessing import OneHotEncoder
```

[102]:
```
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
```

[103]:
```
encoder.fit(raw_df[categorical_cols])
```

[103]:
```
OneHotEncoder
```

```
OneHotEncoder(handle_unknown='ignore', sparse_output=False)
```

[104]:
```
categorical_cols
```

[104]:
```
['Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday']
```

[105]:
```
encoder.categories_
```

[105]:
```
[array(['Adelaide', 'Albany', 'Albury', 'AliceSprings', 'BadgerysCreek',
        'Ballarat', 'Bendigo', 'Brisbane', 'Cairns', 'Canberra', 'Cobar',
        'CoffsHarbour', 'Dartmoor', 'Darwin', 'GoldCoast', 'Hobart',
        'Katherine', 'Launceston', 'Melbourne', 'MelbourneAirport',
        'Mildura', 'Moree', 'MountGambier', 'MountGinini', 'Newcastle',
        'Nhil', 'NorahHead', 'NorfolkIsland', 'Nuriootpa', 'PearceRAAF',
        'Penrith', 'Perth', 'PerthAirport', 'Portland', 'Richmond', 'Sale',
        'SalmonGums', 'Sydney', 'SydneyAirport', 'Townsville',
        'Tuggeranong', 'Uluru', 'WaggaWagga', 'Walpole', 'Watsonia',
        'Williamtown', 'Witchcliffe', 'Wollongong', 'Woomera'],
       dtype=object),
 array(['E', 'ENE', 'ESE', 'N', 'NE', 'NNE', 'NNW', 'NW', 'S', 'SE', 'SSE',
        'SSW', 'SW', 'W', 'WNW', 'WSW', nan], dtype=object),
 array(['E', 'ENE', 'ESE', 'N', 'NE', 'NNE', 'NNW', 'NW', 'S', 'SE', 'SSE',
        'SSW', 'SW', 'W', 'WNW', 'WSW', nan], dtype=object),
 array(['E', 'ENE', 'ESE', 'N', 'NE', 'NNE', 'NNW', 'NW', 'S', 'SE', 'SSE',
        'SSW', 'SW', 'W', 'WNW', 'WSW', nan], dtype=object),
 array(['No', 'Yes'], dtype=object)]
```

[106]:
```
encoded_cols = list(encoder.get_feature_names_out(categorical_cols))
print(encoded_cols)
```

```
['Location_Adelaide', 'Location_Albany', 'Location_Albury', 'Location_AliceSprings', 'Location_BadgerysCreek', 'Location_Ballarat', 'Location_Bendigo', 'Lo
cation_Brisbane', 'Location_Cairns', 'Location_Canberra', 'Location_Cobar', 'Location_CoffsHarbour', 'Location_Dartmoor', 'Location_Darwin', 'Location_Gold
Coast', 'Location_Hobart', 'Location_Katherine', 'Location_Launceston', 'Location_Melbourne', 'Location_MelbourneAirport', 'Location_Mildura', 'Location_Mo
ree', 'Location_MountGambier', 'Location_MountGinini', 'Location_Newcastle', 'Location_Nhil', 'Location_NorahHead', 'Location_NorfolkIsland', 'Location_Nur
iootpa', 'Location_PearceRAAF', 'Location_Penrith', 'Location_Perth', 'Location_PerthAirport', 'Location_Portland', 'Location_Richmond', 'Location_Sale',
'Location_SalmonGums', 'Location_Sydney', 'Location_SydneyAirport', 'Location_Townsville', 'Location_Tuggeranong', 'Location_Uluru', 'Location_WaggaWagga',
'Location_Walpole', 'Location_Watsonia', 'Location_Williamtown', 'Location_Witchcliffe', 'Location_Wollongong', 'Location_Woomera', 'WindGustDir_E', 'WindG
ustDir_ENE', 'WindGustDir_ESE', 'WindGustDir_N', 'WindGustDir_NE', 'WindGustDir_NNE', 'WindGustDir_NNW', 'WindGustDir_NW', 'WindGustDir_S', 'WindGustDir_S
E', 'WindGustDir_SSE', 'WindGustDir_SSW', 'WindGustDir_SW', 'WindGustDir_W', 'WindGustDir_WNW', 'WindGustDir_WSW', 'WindGustDir_nan', 'WindDir9am_E', 'Wind
Dir9am_ENE', 'WindDir9am_ESE', 'WindDir9am_N', 'WindDir9am_NE', 'WindDir9am_NNE', 'WindDir9am_NNW', 'WindDir9am_NW', 'WindDir9am_S', 'WindDir9am_SE', 'Wind
Dir9am_SSE', 'WindDir9am_SSW', 'WindDir9am_SW', 'WindDir9am_W', 'WindDir9am_WNW', 'WindDir9am_WSW', 'WindDir9am_nan', 'WindDir3pm_E', 'WindDir3pm_ENE', 'Wi
ndDir3pm_ESE', 'WindDir3pm_N', 'WindDir3pm_NE', 'WindDir3pm_NNE', 'WindDir3pm_NNW', 'WindDir3pm_NW', 'WindDir3pm_S', 'WindDir3pm_SE', 'WindDir3pm_SSE', 'Wi
ndDir3pm_SSW', 'WindDir3pm_SW', 'WindDir3pm_W', 'WindDir3pm_WNW', 'WindDir3pm_WSW', 'WindDir3pm_nan', 'RainToday_No', 'RainToday_Yes']
```

[108]:
```
train_inputs[encoded_cols] = encoder.transform(train_inputs[categorical_cols])
val_inputs[encoded_cols] = encoder.transform(val_inputs[categorical_cols])
test_inputs[encoded_cols] = encoder.transform(test_inputs[categorical_cols])
```

```
C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
```

```
C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:1: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
```

```
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:2: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
```

```
DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\1584174743.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
```

[111]:
```
test_inputs
```

[111]:

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	...	WindDir3pm_SE	WindDir3pm_SSE
2498	Albury	0.681604	0.801512	0.000000	0.037741	0.526244	ENE	0.372093	NaN	ESE	...	0.0	0.0
2499	Albury	0.693396	0.725898	0.001078	0.037741	0.526244	SSE	0.341085	SSE	SE	...	1.0	0.0
2500	Albury	0.634434	0.527410	0.005930	0.037741	0.526244	ENE	0.325581	ESE	ENE	...	0.0	0.0
2501	Albury	0.608491	0.538752	0.042049	0.037741	0.526244	SSE	0.255814	SE	SSE	...	0.0	1.0
2502	Albury	0.566038	0.523629	0.018329	0.037741	0.526244	ENE	0.193798	SE	SSE	...	0.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
145454	Uluru	0.283019	0.502836	0.000000	0.037741	0.526244	E	0.193798	ESE	E	...	0.0	0.0
145455	Uluru	0.266509	0.533081	0.000000	0.037741	0.526244	E	0.193798	SE	ENE	...	0.0	0.0
145456	Uluru	0.285377	0.568998	0.000000	0.037741	0.526244	NNW	0.124031	SE	N	...	0.0	0.0
145457	Uluru	0.327830	0.599244	0.000000	0.037741	0.526244	N	0.240310	SE	WNW	...	0.0	0.0
145458	Uluru	0.384434	0.601134	0.000000	0.037741	0.526244	SE	0.170543	SSE	N	...	0.0	0.0

25710 rows × 123 columns

SAVING PROCESSED DATA TO DISK:

[112]:
```
print('train_inputs:', train_inputs.shape)
print('train_targets:', train_targets.shape)
print('val_inputs:', val_inputs.shape)
print('val_targets:', val_targets.shape)
print('test_inputs:', test_inputs.shape)
print('test_targets:', test_targets.shape)
```

```
train_inputs: (97988, 123)
train_targets: (97988,)
val_inputs: (17089, 123)
val_targets: (17089,)
test_inputs: (25710, 123)
test_targets: (25710,)
```

[113]:
```
!pip install pyarrow --quiet
```

[115]:
```
train_inputs.to_parquet('train_inputs.parquet')
val_inputs.to_parquet('val_inputs.parquet')
test_inputs.to_parquet('test_inputs.parquet')
```

[116]:
```
%%time
pd.DataFrame(train_targets).to_parquet('train_targets.parquet')
pd.DataFrame(val_targets).to_parquet('val_targets.parquet')
pd.DataFrame(test_targets).to_parquet('test_targets.parquet')
```

```
CPU times: total: 0 ns
Wall time: 43.8 ms
```

[117]:
```
%%time
```

```
train_inputs = pd.read_parquet('train_inputs.parquet')
val_inputs = pd.read_parquet('val_inputs.parquet')
test_inputs = pd.read_parquet('test_inputs.parquet')

train_targets = pd.read_parquet('train_targets.parquet')[target_col]
val_targets = pd.read_parquet('val_targets.parquet')[target_col]
test_targets = pd.read_parquet('test_targets.parquet')[target_col]
```

```
CPU times: total: 203 ms
Wall time: 3.44 s
```

TRAINING LOGISTIC REGRESSION MODEL:

[118]:
```
from sklearn.linear_model import LogisticRegression
```

[119]:
```
model = LogisticRegression(solver = 'liblinear')
```

[120]:
```
model.fit(train_inputs[numerical_cols + encoded_cols], train_targets)
```

[120]:
```
LogisticRegression
LogisticRegression(solver='liblinear')
```

[123]:
```
print(numerical_cols + encoded_cols)
```

```
['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressure9am',
'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm', 'Location_Adelaide', 'Location_Albany', 'Location_Albury', 'Location_AliceSprings', 'Location_
BadgerysCreek', 'Location_Ballarat', 'Location_Bendigo', 'Location_Brisbane', 'Location_Cairns', 'Location_Canberra', 'Location_Cobar', 'Location_CoffsHarb
our', 'Location_Dartmoor', 'Location_Darwin', 'Location_GoldCoast', 'Location_Hobart', 'Location_Katherine', 'Location_Launceston', 'Location_Melbourne',
'Location_MelbourneAirport', 'Location_Mildura', 'Location_Moree', 'Location_MountGambier', 'Location_MountGinini', 'Location_Newcastle', 'Location_Nhil',
'Location_NorahHead', 'Location_NorfolkIsland', 'Location_Nuriootpa', 'Location_PearceRAAF', 'Location_Penrith', 'Location_Perth', 'Location_PerthAirport',
'Location_Portland', 'Location_Richmond', 'Location_Sale', 'Location_SalmonGums', 'Location_Sydney', 'Location_SydneyAirport', 'Location_Townsville', 'Loca
tion_Tuggeranong', 'Location_Uluru', 'Location_WaggaWagga', 'Location_Walpole', 'Location_Watsonia', 'Location_Williamtown', 'Location_Witchcliffe', 'Locat
ion_Wollongong', 'Location_Woomera', 'WindGustDir_E', 'WindGustDir_ENE', 'WindGustDir_ESE', 'WindGustDir_N', 'WindGustDir_NE', 'WindGustDir_NNE', 'WindGust
Dir_NNW', 'WindGustDir_NW', 'WindGustDir_S', 'WindGustDir_SE', 'WindGustDir_SSE', 'WindGustDir_SSW', 'WindGustDir_SW', 'WindGustDir_W', 'WindGustDir_WNW',
'WindGustDir_WSW', 'WindGustDir_nan', 'WindDir9am_E', 'WindDir9am_ENE', 'WindDir9am_ESE', 'WindDir9am_N', 'WindDir9am_NE', 'WindDir9am_NNE', 'WindDir9am_NN
W', 'WindDir9am_NW', 'WindDir9am_S', 'WindDir9am_SE', 'WindDir9am_SSE', 'WindDir9am_SSW', 'WindDir9am_SW', 'WindDir9am_W', 'WindDir9am_WNW', 'WindDir9am_WS
W', 'WindDir9am_nan', 'WindDir3pm_E', 'WindDir3pm_ENE', 'WindDir3pm_ESE', 'WindDir3pm_N', 'WindDir3pm_NE', 'WindDir3pm_NNE', 'WindDir3pm_NNW', 'WindDir3pm_
NW', 'WindDir3pm_S', 'WindDir3pm_SE', 'WindDir3pm_SSE', 'WindDir3pm_SSW', 'WindDir3pm_SW', 'WindDir3pm_W', 'WindDir3pm_WNW', 'WindDir3pm_WSW', 'WindDir3pm_
nan', 'RainToday_No', 'RainToday_Yes']
```

[124]:
```
print(model.coef_.tolist())
```

```
[[0.8986296844609359, -2.8799159286158598, 3.1627783078031486, 0.8542472761803896, -1.671394832868807, 6.764404649309685, -0.9423239302288293, -1.428430957
1903335, 0.32288814080650985, 5.995311810260233, 5.463855489018382, -9.176806465043187, -0.16229651827080474, 1.2876591639247392, 0.47471410922966834, 2.02
14274386015316, 0.6016494918198434, -0.5524838093796727, 0.47814227501534884, 0.007670215303817617, 0.3468141038323274, -0.35227648401084205, 0.17971020712
379648, 0.44048598648296355, -0.013982213056392905, 0.028944170557883474, 0.25814806969217374, -0.02120465374093269, -0.042796304899749706, -0.483142260247
8312, -0.13756255135819292, -0.5760584487201934, -0.7875265763826922, -0.25540500948366024, -0.3288828944395345, -0.5690033711989677, 0.08183011085336406,
0.013381563850681389, 0.06412782453816372, -0.9020542552600986, -0.4443309080936066, 0.008516694600742183, -0.46061273490213495, -0.46551772082084514, -0.0
6949483294636523, 0.19115935232541884, 0.4504752332976372, 0.6081210059582741, 0.42731397714945135, -0.0283310544498291, 0.2515458294654211, -0.321605113123
6798, 0.4249561555519268, -0.059037136316173223, -0.1131988415660064, -0.7283761656589695, 0.3664520914752499, 0.18359099914563118, 0.1839752693561043, 0.1
866024426272472, -0.24927020218333118, 0.01794887018421567, 0.7034018061144872, -0.8000198603884683, -0.19234383255353385, -0.16179770422962547, -0.1589853
5259625607, -0.063100142151729, -0.223551079375832 35, -0.23239139744492546, -0.32115423139496135, -0.16557085063341592, -0.15884848364670312, -0.1109766934
4262508, -0.032528519781378995, -0.05062806230467471, -0.09419816448640095, -0.09081497702523972, -0.2231571589761742, -0.2585272751928069, -0.200815688938
55434, 0.09748879624967985, -0.31877423930018967, -0.02935693751089768, -0.32861293907405914, 0.02986468354627196, -0.021071324437948017, 0.144052474965376
32, -0.06176884412454976, -0.05697348896468542, -0.40853238755239507, -0.3081227822033797, -0.4054373155211494, -0.19482654824162582, -0.06029113377254502,
-0.08692094212955662, -0.05711002548213221, -0.016131623584465012, -0.2695436119833161, -0.2398531692687747, -0.15250067289843044, -0.2348350440969778, 0.0
3819171926121875, -0.31141220196456865, -0.08024621762257216, 0.2845424620718202, 0.2218505716645121, -0.26544757964649235, -0.24117476505073643, -0.366880
5248974998, -0.3168629864074375, -0.37002097711988297, -0.18037145950636146, -0.03349950282832776, -0.27597542113298074, 0.07493332407084184, -1.4735184782
235295, -0.9760385071360357]]
```

[125]:
```
print(model.intercept_)
```

```
[-2.44955699]
```

MAKING PREDICTIONS AND EVALUATING THE MODEL:

[127]:
```
X_train = train_inputs[numerical_cols + encoded_cols]
X_val = val_inputs[numerical_cols + encoded_cols]
X_test = test_inputs[numerical_cols + encoded_cols]
```

[128]:
```
train_preds = model.predict(X_train)
```

[129]:
```
train_preds
```

[129]:
```
array(['No', 'No', 'No', ..., 'No', 'No', 'No'], dtype=object)
```

[131]:
```
train_targets
```

[131]:
```
0         No
1         No
2         No
3         No
4         No
          ..
144548    No
144549    No
144550    No
144551    No
144552    No
Name: RainTomorrow, Length: 97988, dtype: object
```

[136]:
```
train_probs = model.predict_proba(X_train)
train_probs
```

[136]:
```
array([[0.94401187, 0.05598813],
```

```
       [0.9407418 , 0.0592582 ],
       [0.96093644, 0.03906356],
       ...,
       [0.98749115, 0.01250885],
       [0.98334684, 0.01665316],
       [0.87453434, 0.12546566]])
```

[139]:
```
model.classes_
```

[139]:
```
array(['No', 'Yes'], dtype=object)
```

[140]:
```
from sklearn.metrics import accuracy_score
```

[141]:
```
accuracy_score(train_targets, train_preds)
```

[141]:
```
0.8519206433440829
```

[142]:
```
from sklearn.metrics import confusion_matrix
```

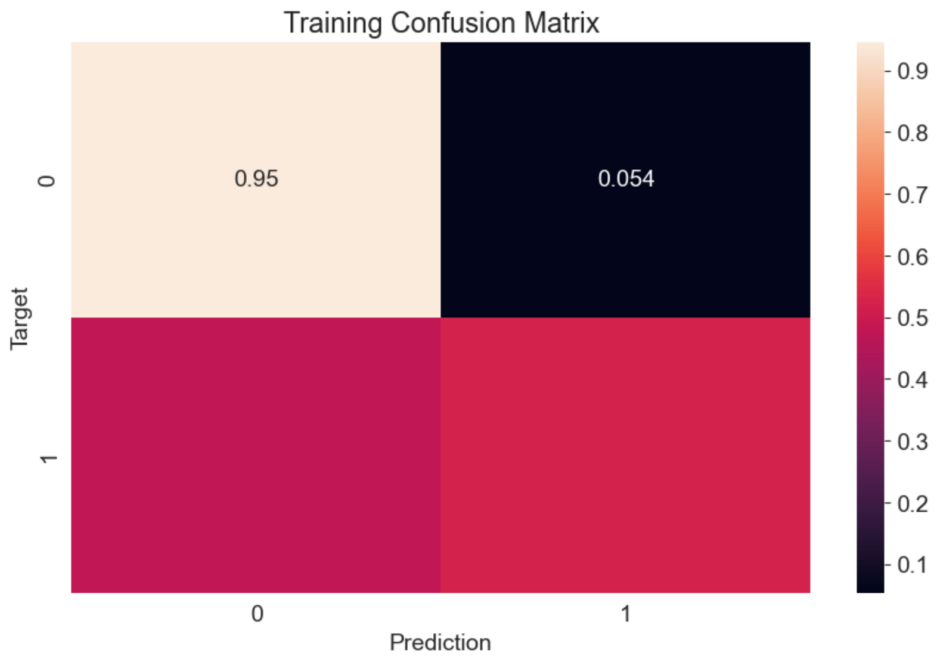
[143]:
```
confusion_matrix(train_targets,train_preds, normalize = 'true')
```

[143]:
```
array([[0.94621341, 0.05378659],
       [0.4776585 , 0.5223415 ]])
```

[144]:
```
def predict_and_plot(inputs, targets, name=''):
    preds = model.predict(inputs)

    accuracy = accuracy_score(targets, preds)
    print("Accuracy: {:.2f}%".format(accuracy * 100))

    cf = confusion_matrix(targets, preds, normalize='true')
    plt.figure()
    sns.heatmap(cf, annot=True)
    plt.xlabel('Prediction')
    plt.ylabel('Target')
    plt.title('{} Confusion Matrix'.format(name));

    return preds
```

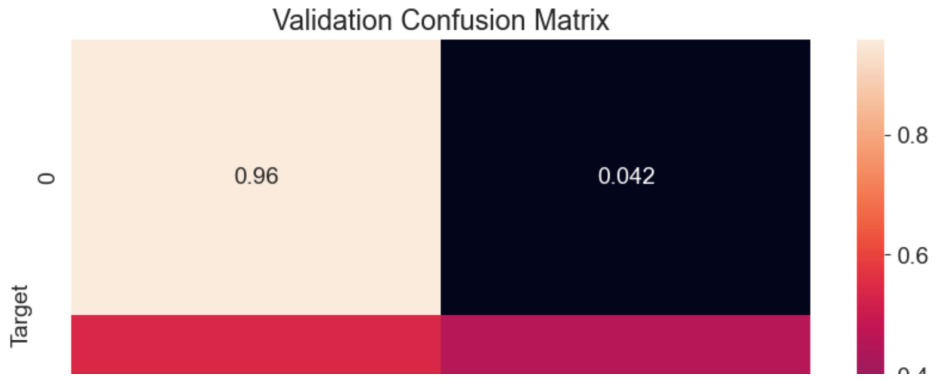
[145]:
```
train_preds = predict_and_plot(X_train, train_targets, 'Training')
```

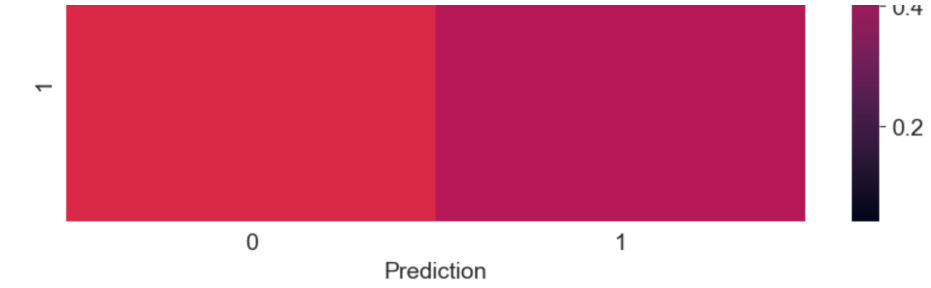
```
Accuracy: 85.19%
```



[146]:
```
val_preds = predict_and_plot(X_val, val_targets, 'Validation')
```

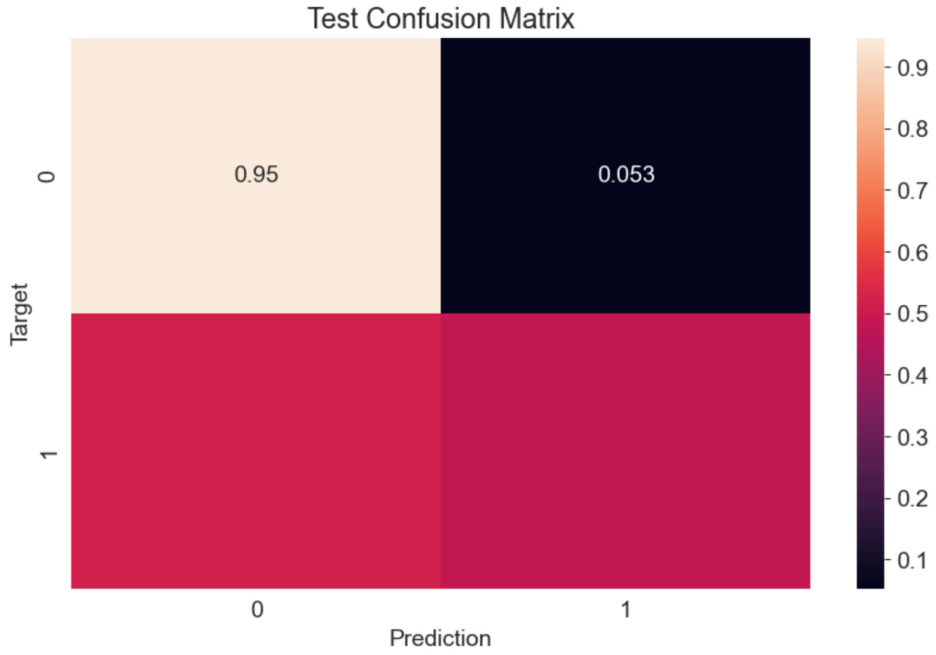
```
Accuracy: 85.40%
```





[147]:
```
test_preds = predict_and_plot(X_test, test_targets, 'Test')
```

```
Accuracy: 84.20%
```



[150]:
```
def random_guess(inputs):
    return np.random.choice(["No","Yes"], len(inputs))
```

[151]:
```
def all_no(inputs):
    return np.full(len(inputs),"No")
```

[152]:
```
random_guess(X_val)
```

[152]:
```
array(['No', 'Yes', 'Yes', ..., 'Yes', 'Yes', 'Yes'], dtype='<U3')
```

[153]:
```
all_no(X_val)
```

[153]:
```
array(['No', 'No', 'No', ..., 'No', 'No', 'No'], dtype='<U2')
```

[154]:
```
accuracy_score(test_targets, random_guess(X_test))
```

[154]:
```
0.503461688059121
```

[155]:
```
accuracy_score(test_targets, all_no(X_test))
```

[155]:
```
0.7734344612991054
```

MAKING PREDICTIONS ON A SINGLE INPUT:

[156]:
```
new_input = {'Date': '2021-06-19',
             'Location': 'Katherine',
             'MinTemp': 23.2,
             'MaxTemp': 33.2,
             'Rainfall': 10.2,
             'Evaporation': 4.2,
             'Sunshine': np.nan,
             'WindGustDir': 'NNW',
             'WindGustSpeed': 52.0,
             'WindDir9am': 'NW',
             'WindDir3pm': 'NNE',
             'WindSpeed9am': 13.0,
             'WindSpeed3pm': 20.0,
             'Humidity9am': 89.0,
             'Humidity3pm': 58.0,
             'Pressure9am': 1004.8,
             'Pressure3pm': 1001.5,
             'Cloud9am': 8.0,
             'Cloud3pm': 5.0,
```

```
          'Temp9am': 25.7,
          'Temp3pm': 33.0,
          'RainToday': 'Yes'}
```

[158]:
```python
new_input_df = pd.DataFrame([new_input])
```

[159]:
```python
new_input_df
```

[159]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	WindSpeed3pm	Humidity9am	Humidity3pm
0	2021-06-19	Katherine	23.2	33.2	10.2	4.2	NaN	NNW	52.0	NW	...	20.0	89.0	58.0

1 rows × 22 columns

[163]:
```python
new_input_df[numerical_cols] = imputer.transform(new_input_df[numerical_cols])
new_input_df[numerical_cols] = scaler.transform(new_input_df[numerical_cols])
new_input_df[encoded_cols] = encoder.transform(new_input_df[categorical_cols])
```

```
C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
```

```
DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2785338023.py:3: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
```

[164]:
```
new_input_df
```

[164]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	WindDir3pm_SE	WindDir3pm_SSE	WindDir3p
0	2021-06-19	Katherine	0.218105	0.104316	0.000074	0.0002	0.036293	NNW	-0.043747	NW	...	0.0	0.0	

1 rows × 124 columns

[166]:
```
X_new_input = new_input_df[numerical_cols + encoded_cols]
X_new_input
```

[166]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	...	WindDir3pm_SE	WindDir3p
0	0.218105	0.104316	0.000074	0.0002	0.036293	-0.043747	0.000769	0.002642	0.0089	0.0058	...	0.0	

1 rows × 118 columns

[169]:
```
prediction = model.predict(X_new_input)[0]
prediction
```

[169]: 'Yes'

[174]:
```
prob = model.predict_proba(X_new_input)[0]
```

[175]:
```
prob
```

[175]: array([0., 1.])

[182]:
```
def predict_input(single_input):
    input_df = pd.DataFrame([single_input])
    input_df[numerical_cols] = imputer.transform(input_df[numerical_cols])
    input_df[numerical_cols] = scaler.transform(input_df[numerical_cols])
    input_df[encoded_cols] = encoder.transform(input_df[categorical_cols])
    X_input = input_df[numerical_cols + encoded_cols]
    pred = model.predict(X_input)[0]
    prob = model.predict_proba(X_input)[0][list(model.classes_).index(pred)]
    return pred, prob
```

[183]:
```
new_input = {'Date': '2021-06-19',
             'Location': 'Launceston',
             'MinTemp': 23.2,
             'MaxTemp': 33.2,
             'Rainfall': 10.2,
             'Evaporation': 4.2,
             'Sunshine': np.nan,
             'WindGustDir': 'NNW',
             'WindGustSpeed': 52.0,
             'WindDir9am': 'NW',
             'WindDir3pm': 'NNE',
             'WindSpeed9am': 13.0,
             'WindSpeed3pm': 20.0,
             'Humidity9am': 89.0,
             'Humidity3pm': 58.0,
             'Pressure9am': 1004.8,
             'Pressure3pm': 1001.5,
             'Cloud9am': 8.0,
             'Cloud3pm': 5.0,
             'Temp9am': 25.7,
             'Temp3pm': 33.0,
             'RainToday': 'Yes'}
```

[184]:
```
predict_input(new_input)
```

```
C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2363771375.py:5: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`

C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2363771375.py:5: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
```

```
C:\Users\adity\AppData\Local\Temp\ipykernel_18516\2363771375.py:5: PerformanceWarning:

DataFrame is highly fragmented.  This is usually the result of calling `frame.insert` many times, which has poor performance.  Consider joining all columns
at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
```

(repeated 20 times)

[184]:
```
('Yes', 0.6474924566585274)
```

[185]:
```
import joblib
```

[187]:
```
aussie_rain = {
    'model': model,
    'imputer': imputer,
    'scaler': scaler,
    'encoder': encoder,
    'input_cols': input_cols,
    'target_col': target_col,
```

```
    'numerical_cols': numerical_cols,
    'categorical_cols': categorical_cols,
    'encoded_cols': encoded_cols
}
```

[188]:
```
joblib.dump(aussie_rain, 'aussie_rain.joblib')
```

[188]:
```
['aussie_rain.joblib']
```

[ ]: