Project: Weather Prediction using Logistic Regression

Predicting whether it will rain tomorrow using today's weather data

Getting Dataset

In [1]:
```
!pip install jovian opendatasets --upgrade --quiet
```

In [2]:
```
url='https://raw.githubusercontent.com/Asad-cuet/Machine-Learning-Code/master/dataset/weatherAUS.csv'
```

In [3]:
```
import opendatasets as od
od.download(url)
```

```
Using downloaded and verified file: ./weatherAUS.csv
```

In [4]:
```
import pandas as pd
```

In [5]:
```
raw_df=pd.read_csv('weatherAUS.csv')
```

In [6]:
```
raw_df
```

Out[6]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	Win
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	
...	...	...	...	...	...	...	...	...	
145455	2017-06-21	Uluru	2.8	23.4	0.0	NaN	NaN	E	
145456	2017-06-22	Uluru	3.6	25.3	0.0	NaN	NaN	NNW	
145457	2017-06-23	Uluru	5.4	26.9	0.0	NaN	NaN	N	
145458	2017-06-24	Uluru	7.8	27.0	0.0	NaN	NaN	SE	
145459	2017-06-25	Uluru	14.9	NaN	0.0	NaN	NaN	NaN	

145460 rows × 23 columns

Identifying input and target columns

In [7]:
```
input_cols=list(raw_df.columns)[1:-1]  # Excluding last column by range [1:-1]. Python range works as like [ , )
input_cols
```

Out[7]:
```
['Location',
 'MinTemp',
 'MaxTemp',
 'Rainfall',
 'Evaporation',
 'Sunshine',
 'WindGustDir',
 'WindGustSpeed',
 'WindDir9am',
 'WindDir3pm',
 'WindSpeed9am',
 'WindSpeed3pm',
 'Humidity9am',
 'Humidity3pm',
 'Pressure9am',
 'Pressure3pm',
 'Cloud9am',
 'Cloud3pm',
 'Temp9am',
 'Temp3pm',
 'RainToday']
```

In [8]:
```
target_cols=list(raw_df.columns)[-1]
target_cols
```

Out[8]:
```
'RainTomorrow'
```

Data Preprocessing

Remove row where target columns is empty

In [9]:
```
raw_df[target_cols].unique()
```

Out[9]:
```
array(['No', 'Yes', nan], dtype=object)
```

See there is nan value

In [10]:
```
raw_df.dropna(subset=['RainToday', 'RainTomorrow'], inplace=True)
```

In [11]:
```
raw_df[target_cols].unique()
```

Out[11]:
```
array(['No', 'Yes'], dtype=object)
```

Now there is no none value

Spliting Dataset

three parts:
Training Set: Train model, compute loss, execute optimization
Validation Set: Pick best verson of model
Test Set: Compare different models

Explaination:

Split raw dataset into **traing validation set** and **test set** in ratio 7:3 .
From traing validation set,split into **training set** and **validation set** in ration 7:3
Split traing set into **traing input set** and **traning target set**

Note: Here,
traing input set is Training Set
validation set is Validation set
test Set is Test Set

df

In [12]:
```
from sklearn.model_selection import train_test_split
```

In [13]:
```
train_val_df, test_df = train_test_split(raw_df,test_size=0.3,random_state=42)
```

In [14]:
```
train_df, val_df = train_test_split(train_val_df,test_size=0.3,random_state=4
2)
```

inputs & targets

In [15]:
```
train_inputs=train_df[input_cols].copy()
train_targets=train_df[target_cols].copy()
```

In [16]:
```
val_inputs=val_df[input_cols].copy()
val_targets=val_df[target_cols].copy()
```

In [17]:
```
test_inputs=test_df[input_cols].copy()
test_targets=test_df[target_cols].copy()
```

Identify Numeric & Categorical Column

In [18]:
```
import numpy as np
```

In [19]:
```
numeric_cols=train_inputs.select_dtypes(include=np.number).columns.tolist()
```

In [20]:
```
categorical_cols=train_inputs.select_dtypes('object').columns.tolist()
```

Observing input columns

In [21]:
```
train_inputs[numeric_cols].describe()
```

Out[21]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	W
count	68740.000000	68846.00000	68985.000000	39555.000000	36105.000000	64493.000000	
mean	12.187416	23.21404	2.405229	5.467337	7.636305	39.942350	
std	6.400621	7.13213	8.757592	4.199693	3.780028	13.572923	
min	-8.200000	-4.10000	0.000000	0.000000	0.000000	6.000000	
25%	7.600000	17.90000	0.000000	2.600000	4.900000	31.000000	
50%	12.000000	22.60000	0.000000	4.800000	8.500000	39.000000	
75%	16.800000	28.20000	0.800000	7.400000	10.700000	48.000000	
max	33.900000	48.10000	367.600000	145.000000	14.500000	135.000000	

In [22]:
```
train_inputs[categorical_cols].nunique()  # always use nunique() in categorica
l column
```

Out[22]:
```
Location       49
WindGustDir    16
WindDir9am     16
WindDir3pm     16
RainToday       2
dtype: int64
```

Cleaning Numeric Columns

Imputation

Model can't work with missing numerical data. The process of filling missing values is called imputation.

In [23]:
```
# Looking is there missing values
train_inputs[numeric_cols].isna().sum()  # isna() shows all missing data
```

Out[23]:
```
MinTemp            245
MaxTemp            139
Rainfall             0
Evaporation      29430
Sunshine         32880
WindGustSpeed     4492
WindSpeed9am       520
WindSpeed3pm      1252
Humidity9am        723
Humidity3pm       1722
Pressure9am       6834
Pressure3pm       6846
Cloud9am         25884
Cloud3pm         27512
Temp9am            316
Temp3pm           1309
dtype: int64
```

Yes. There is missing values

In [24]:
```
from sklearn.impute import SimpleImputer
```

In [25]:
```
imputer=SimpleImputer(strategy='mean') # reaplacing by mean value
```

In [26]:
```
imputer.fit(raw_df[numeric_cols])  # computing mean value from entire dataset.
Beacause traing set, validation set, test set separated now.
                                   # Stored is statistics_
                                   # You can see by imputer.statistics_
```

Out[26]:
```
SimpleImputer()
```

In [27]:
```
train_inputs[numeric_cols]=imputer.transform(train_inputs[numeric_cols])
```

In [28]:
```
val_inputs[numeric_cols]=imputer.transform(val_inputs[numeric_cols])
```

In [29]:
```
test_inputs[numeric_cols]=imputer.transform(test_inputs[numeric_cols])
```

In [30]:
```
## checking again, is there missing value?
train_inputs[numeric_cols].isna().sum()
```

Out[30]:
```
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
```

Now, There is no missing values

Imputation completed

Scaling Values in range 0 to 1

In [31]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [32]:
```
scaler=MinMaxScaler()
```

In [33]:
```
scaler.fit(raw_df[numeric_cols])
```

Out[33]:
```
MinMaxScaler()
```

now you can see min,max value of all columns by scaler.data*min* , scaler.data*max*

list(scaler.data*min*)

In [34]:
```
train_inputs[numeric_cols]=scaler.transform(train_inputs[numeric_cols])
```

In [35]:
```
val_inputs[numeric_cols]=scaler.transform(val_inputs[numeric_cols])
```

In [36]:
```
test_inputs[numeric_cols]=scaler.transform(test_inputs[numeric_cols])
```

Now all valuse scaled.
You can check it by train_inputs[numeric_cols].describe()
Scaling Done

Cleaning Categorical Columns

Converting Categorical data into number using encoder
You can see no. of unique value of all columns by nunique()

In [37]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [38]:
```
encoder=OneHotEncoder(sparse=False,handle_unknown='ignore')
```

In [39]:
```
encoder.fit(raw_df[categorical_cols].fillna('Unknowns')) # categorical_cols].f
illna('Unknowns') replace missing values
```

Out[39]:
```
OneHotEncoder(handle_unknown='ignore', sparse=False)
```

You can see: encoder.categories_

In [40]:
```
encoded_cols=list(encoder.get_feature_names(categorical_cols))  # getting enco
ded column names
```

```
/opt/conda/lib/python3.9/site-packages/sklearn/utils/deprecation.py:87: Futur
eWarning: Function get_feature_names is deprecated; get_feature_names is depr
ecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out in
stead.
  warnings.warn(msg, category=FutureWarning)
```

Now we will create new columns in the dataset

In [41]:
```
train_inputs[encoded_cols]=encoder.transform(train_inputs[categorical_cols])
```

```
/opt/conda/lib/python3.9/site-packages/pandas/core/frame.py:3678: Performance
Warning: DataFrame is highly fragmented.  This is usually the result of calli
ng `frame.insert` many times, which has poor performance.  Consider joining a
ll columns at once using pd.concat(axis=1) instead.  To get a de-fragmented f
rame, use `newframe = frame.copy()`
  self[col] = igetitem(value, i)
```

In [42]:
```
val_inputs[encoded_cols]=encoder.transform(val_inputs[categorical_cols])
```

```
/opt/conda/lib/python3.9/site-packages/pandas/core/frame.py:3678: Performance
Warning: DataFrame is highly fragmented.  This is usually the result of calli
ng `frame.insert` many times, which has poor performance.  Consider joining a
ll columns at once using pd.concat(axis=1) instead.  To get a de-fragmented f
rame, use `newframe = frame.copy()`
  self[col] = igetitem(value, i)
```

In [43]:
```
test_inputs[encoded_cols]=encoder.transform(test_inputs[categorical_cols])
```

```
/opt/conda/lib/python3.9/site-packages/pandas/core/frame.py:3678: Performance
Warning: DataFrame is highly fragmented.  This is usually the result of calli
ng `frame.insert` many times, which has poor performance.  Consider joining a
ll columns at once using pd.concat(axis=1) instead.  To get a de-fragmented f
rame, use `newframe = frame.copy()`
  self[col] = igetitem(value, i)
```

Done
You can see in the dataset

Saving Preprocessing Data. Optional

In [44]:
```
pd.DataFrame(train_inputs).to_csv('train_inputs.csv')
```

In [45]:
```
pd.DataFrame(val_inputs).to_csv('val_inputs.csv')
```

In [46]:
```
pd.DataFrame(test_inputs).to_csv('test_inputs.csv')
```

Saved in file
You can read by pd.read_csv('train_inputs.csv')

Making & Training Model

In [47]:
```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression(solver='liblinear')   #making
```

In [48]:
```
model.fit(train_inputs[numeric_cols+encoded_cols],train_targets)  #  training
```

Out[48]:
```
LogisticRegression(solver='liblinear')
```

Making Prediction

In [49]:
```
X_train=train_inputs[numeric_cols+encoded_cols]
```

In [50]:
```
X_val=val_inputs[numeric_cols+encoded_cols]
```

In [51]:
```
X_test=test_inputs[numeric_cols+encoded_cols]
```

In [52]:
```
train_preds=model.predict(X_train)  # and the train target is train_targets
```

In [53]:
```
val_preds=model.predict(X_val)  # and the train target is  val_targets
```

In [54]:
```
test_preds=model.predict(X_test)  # and the train target is  test_targets
```

Testing: Comparing traning prediction with target values

Accuracy and Confusion Matrix

In [55]:
```
from sklearn.metrics import accuracy_score
```

In [56]:
```
accuracy_score(train_targets,train_preds)
```

Out[56]:
```
0.8524606798579402
```

In [57]:
```
# also you can see prediction matrix
from sklearn.metrics import confusion_matrix
```

In [58]:
```
confusion_matrix(train_targets,train_preds,normalize='true')
```

Out[58]:
```
array([[0.94716574, 0.05283426],
       [0.48174895, 0.51825105]])
```

Do google to know about confusion matrix.

Summering:
Left top value is fraction of 'No' result, which macthed with target value
Right bottom value is fraction of 'Yes' result, which macthed with target value

In [59]:
```
# lets do of others
accuracy_score(val_targets,val_preds)
```

Out[59]:
```
0.8489768307119905
```

In [60]:
```
confusion_matrix(val_targets,val_preds,normalize='true')
```

Out[60]:
```
array([[0.94815974, 0.05184026],
       [0.4993895 , 0.5006105 ]])
```

In [61]:
```
accuracy_score(test_targets,test_preds)
```

Out[61]:
```
0.8470061794161517
```

In [62]:
```
confusion_matrix(test_targets,test_preds,normalize='true')
```

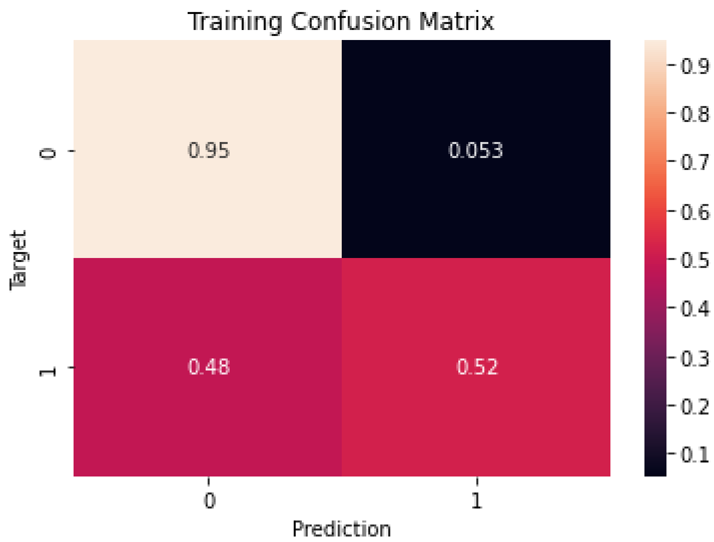
Out[62]:
```
array([[0.94585619, 0.05414381],
       [0.49750451, 0.50249549]])
```

Visualized Confusion Matrix by defined function

In [63]:
```
import matplotlib.pyplot as plt
import seaborn as sns
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
```

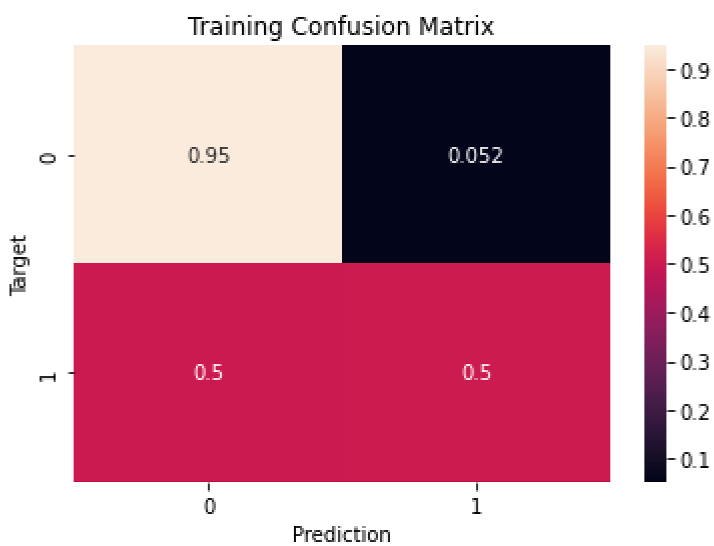
In [64]:
```
train_preds = predict_and_plot(X_train, train_targets, 'Training')
```

```
Accuracy: 85.25%
```



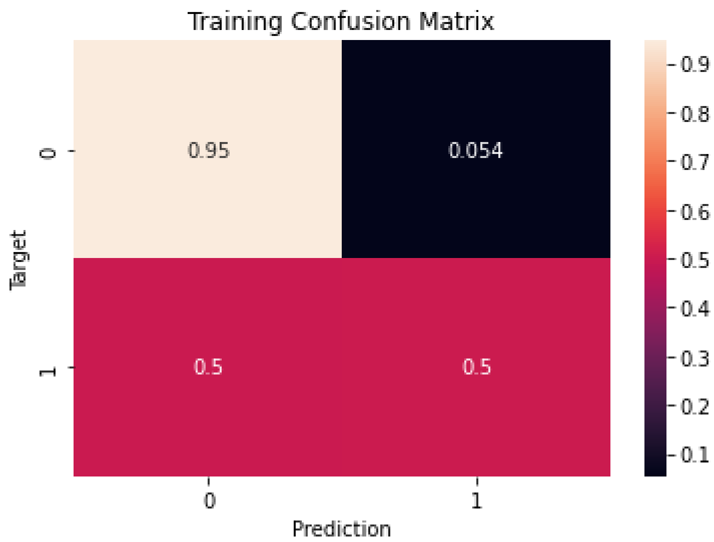
In [65]:
```
train_preds = predict_and_plot(X_val, val_targets, 'Training')
```

```
Accuracy: 84.90%
```



In [66]:
```
train_preds = predict_and_plot(X_test, test_targets, 'Training')
```

```
Accuracy: 84.70%
```



Explaination

Confusion Matrix

Results Explained
The Confusion Matrix created has four different quadrants:

True Negative (Top-Left Quadrant)
False Positive (Top-Right Quadrant)
False Negative (Bottom-Left Quadrant)
True Positive (Bottom-Right Quadrant)
True means that the values were accurately predicted, False means that there was an error or wrong prediction.

Accuracy

Accuracy measures how often the model is correct.

Formula: (True Positive + True Negative) / Total Predictions

Precision

It measures what percentage is truly positive.

Formula: True Positive / (True Positive + False Positive)

Sensitivity (Recall)

It measures what percentage are predicted positive

Formula: True Positive / (True Positive + False Negative)

Specificity

It shows How well the model is at prediciting negative results

Formula: True Negative / (True Negative + False Positive)

F-score

F-score is the "harmonic mean" of precision and sensitivity.

It considers both false positive and false negative cases and is good for imbalanced datasets.

Formula: 2 *((Precision* Sensitivity) / (Precision + Sensitivity))

Prediction on single input

Take Input

In [67]:
```
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'}
```

Preprocess the input

In [68]:
```
new_input_df=pd.DataFrame([new_input])
```

In [69]:
```
new_input_df[numeric_cols]=imputer.transform(new_input_df[numeric_cols]) # imp
uting
```

In [70]:
```
new_input_df[numeric_cols]=scaler.transform(new_input_df[numeric_cols]) # scal
ing
```

In [71]:
```
new_input_df[encoded_cols]=encoder.transform(new_input_df[categorical_cols]) #
encoding
```

```
/opt/conda/lib/python3.9/site-packages/pandas/core/frame.py:3678: Performance
Warning: DataFrame is highly fragmented.  This is usually the result of calli
ng `frame.insert` many times, which has poor performance.  Consider joining a
ll columns at once using pd.concat(axis=1) instead.  To get a de-fragmented f
rame, use `newframe = frame.copy()`
  self[col] = igetitem(value, i)
```

Predicting

In [72]:
```
X_new_input=new_input_df[numeric_cols+encoded_cols]
```

In [73]:
```
preidiction=model.predict(X_new_input)[0]
preidiction
```

Out[73]:
```
'Yes'
```

In [74]:
```
probability=model.predict_proba(X_new_input)[0]
probability
```

Out[74]:
```
array([0.31309278, 0.68690722])
```