import required library

In [1]:

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn import preprocessing
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn.metrics import jaccard_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix, accuracy_score
import sklearn.metrics as metrics
```

Uplode dataset

In [2]:

```
df = pd.read_csv("Weather_Data.csv")
df.head()
```

Out[2]:

	Date	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	Wi
0	2/1/2008	19.5	22.4	15.6	6.2	0.0	W	41	S	
1	2/2/2008	19.5	25.6	6.0	3.4	2.7	W	41	W	
2	2/3/2008	21.6	24.5	6.6	2.4	0.1	W	41	ESE	
3	2/4/2008	20.2	22.8	18.8	2.2	0.0	W	41	NNE	
4	2/5/2008	19.7	25.7	77.4	4.8	0.0	W	41	NNE	

5 rows × 22 columns

In [3]:

```
df.shape
```

Out[3]:

```
(3271, 22)
```

Data Preprocessing

One Hot Encoding

In [5]:

```
df_hot= pd.get_dummies(data=df, columns=['RainToday', 'WindGustDir', 'WindDir9am', 'WindDir3pm'])
```

In [8]:

```
df_hot.shape
```

Out[8]:

```
(3271, 68)
```

In [9]:

```
df_hot.replace(['No', 'Yes'], [0,1], inplace=True)
```

Training Data and Test Data

In [10]:

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

In [11]:

```
df_hot = df_hot.astype(float)
```

In [12]:

```
features = df_hot.drop(columns='RainTomorrow', axis=1)
Y = df_hot['RainTomorrow']
```

In [13]:

```
features.head(2)
```

Out[13]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humid
0	19.5	22.4	15.6	6.2	0.0	41.0	17.0	20.0	
1	19.5	25.6	6.0	3.4	2.7	41.0	9.0	13.0	

2 rows × 66 columns

In [14]:

```
Y.head()
```

. . .

Linear Regression

In [15]:

```
from sklearn.model_selection import train_test_split
```

In [16]:

```
X_train , X_test , y_train , y_test = train_test_split(features , Y , test_size = 0.2 , random_state =10
```

In [17]:

```
Regmodel= LinearRegression()
Regmodel.fit(X_train , y_train)
```

Out[17]:

```
▾ LinearRegression
LinearRegression()
```

In [19]:

```
Prediction = Regmodel.predict(X_test)
Prediction
```

...

In [20]:

```
Regmodel.score(X_test , y_test)
```

Out[20]:

```
0.42713493110579515
```

Using the predictions and the y_test dataframe calculate the value for each metric using the appropriate function.

In [21]:

```
from sklearn.metrics import r2_score
```

In [31]:

```
LinearRegression_MAE = np.mean(np.absolute(Prediction - y_test))*100
LinearRegression_MSE = np.mean((Prediction - y_test)**2)*100
LinearRegression_R2 =  r2_score(y_test , Prediction)*100
```

In [32]:

```
print("Mean Absolute Error is :- ", LinearRegression_MAE)
print("Mean Square Error is :- ", LinearRegression_MSE)
print("R2 score  is :- ", LinearRegression_R2)
```

```
Mean Absolute Error is :-  25.63212270955093
Mean Square Error is :-  11.572001242502735
R2 score  is :-  42.71349311057951
```

Show the MAE, MSE, and R2 in a tabular format using data frame for the linear model

In [33]:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

In [35]:

```
mae = mean_absolute_error(y_test, Prediction)
mse = mean_squared_error(y_test, Prediction)
r2 = r2_score(y_test, Prediction)

# Create a DataFrame to display the results
results = pd.DataFrame({'Metric': ['MAE', 'MSE', 'R2'],
                        'Value': [mae, mse, r2]})

print(results)
```

```
  Metric     Value
0    MAE  0.256321
1    MSE  0.115720
2     R2  0.427135
```

In [ ]:

```

```

KNN

In [39]:

```
Knnmodel = KNeighborsClassifier(n_neighbors=4)
```

In [40]:

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

Out[40]:

```
▾        KNeighborsClassifier
KNeighborsClassifier(n_neighbors=4)
```

In [41]:

```
Prediction = Knnmodel.predict(X_test)
```

In [42]:

```
KNN_Accuracy_Score = accuracy_score(y_test , Prediction)
KNN_JaccardIndex = jaccard_score(y_test , Prediction)
KNN_F1_Score = f1_score(y_test ,Prediction)
```

In [44]:

```
print('KNN Accuracy Score is :-' , KNN_Accuracy_Score*100)
print('KNN_JaccardIndex is :-' , KNN_JaccardIndex)
print('KNN_F1_Scoreis :-' , KNN_F1_Score)
```

```
KNN Accuracy Score is :- 81.83206106870229
KNN_JaccardIndex is :- 0.4251207729468599
KNN_F1_Scoreis :- 0.5966101694915255
```

In [ ]:

```

```

Decision Tree

In [46]:

```
Treemodel = DecisionTreeClassifier()
Treemodel.fit(X_train , y_train)
```

Out[46]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [47]:

```
predictions = Treemodel.predict(X_test)
```

In [49]:

```
print('Tree_Accuracy_Score is :-', accuracy_score(y_test , predictions))
print('Tree_JaccardIndex is :-' , jaccard_score(y_test , predictions))
print('Tree_F1_Score is :-' , f1_score(y_test , predictions))
```

```
Tree_Accuracy_Score is :- 0.7587786259541984
Tree_JaccardIndex is :- 0.3923076923076923
Tree_F1_Score is :- 0.56353591160221
```

In [ ]:

```

```

Logistic Regression

In [55]:

```
x_train, x_test, y_train, y_test = train_test_split(features , Y , test_size = 0.2 , random_state = 1 )
```

In [57]:

```
Logmodel = LogisticRegression()
```

In [58]:

```
Logmodel.fit(x_train , y_train)
```

```
C:\Users\amit7\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/s
table/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (http
s://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[58]:

```
▾ LogisticRegression
LogisticRegression()
```

In [64]:

```
predictions = Logmodel.predict(x_test)
```

In [60]:

```
predict_proba = Logmodel.predict_proba(x_test)
```

In [68]:

```

lr_accuracy_predict = accuracy_score(y_test, predictions)
print('LR Accuracy Score by predict method is:', lr_accuracy_predict)

# Accuracy Score using predict_proba method
lr_accuracy_predict_proba = accuracy_score(y_test, (predict_proba[:, 1] > 0.5).astype(int))
print('LR Accuracy Score by predict_proba method is:', lr_accuracy_predict_proba)

# Jaccard Index using predict method
lr_jaccard_predict = jaccard_score(y_test, predictions)
print('LR Jaccard Index by predict method is:', lr_jaccard_predict)

# Jaccard Index using predict_proba method
lr_jaccard_predict_proba = jaccard_score(y_test, (predict_proba[:, 1] > 0.5).astype(int))
print('LR Jaccard Index by predict_proba method is:', lr_jaccard_predict_proba)

# F1 Score using predict method
lr_f1_predict = f1_score(y_test, predictions)
print('LR F1 Score by predict method is:', lr_f1_predict)

# F1 Score using predict_proba method
lr_f1_predict_proba = f1_score(y_test, (predict_proba[:, 1] > 0.5).astype(int))
print('LR F1 Score by predict_proba method is:', lr_f1_predict_proba)

# Log Loss using predict method (Note: Log Loss is not typically calculated using 'predict' directly)
# lr_log_loss_predict = log_loss(y_test, predictions)
# print('LR Log Loss by predict method is:', lr_log_loss_predict)

# Log Loss using predict_proba method
lr_log_loss_predict_proba = log_loss(y_test, predict_proba)
print('LR Log Loss by predict_proba method is:', lr_log_loss_predict_proba)
```

```
LR Accuracy Score by predict method is: 0.8259541984732824
LR Accuracy Score by predict_proba method is: 0.8259541984732824
LR Jaccard Index by predict method is: 0.47706422018348627
LR Jaccard Index by predict_proba method is: 0.47706422018348627
LR F1 Score by predict method is: 0.6459627329192547
LR F1 Score by predict_proba method is: 0.6459627329192547
LR Log Loss by predict_proba method is: 0.3880358597324609
```

In [ ]:

```

```

SVM

In [73]:

```
from sklearn.svm import SVC
```

In [74]:

```
svm_model = SVC(kernel='linear', C=1.0)
```

In [76]:

```
svm_model.fit(x_train , y_train)
```

Out[76]:

```
▾          SVC
SVC(kernel='linear')
```

In [79]:

```
prediction = svm_model.predict(x_test)
```

In [80]:

```
print('Tree_Accuracy_Score is :-', accuracy_score(y_test , prediction))
print('Tree_JaccardIndex is :-' , jaccard_score(y_test , prediction))
print('Tree_F1_Score is :-' , f1_score(y_test , prediction))
```

```
Tree_Accuracy_Score is :- 0.8381679389312977
Tree_JaccardIndex is :- 0.5069767441860465
Tree_F1_Score is :- 0.6728395061728395
```

Show the Accuracy,Jaccard Index,F1-Score and LogLoss in a tabular format using data frame for all of the above models

In [82]:

```
from sklearn.metrics import accuracy_score , jaccard_score , f1_score , log_loss
```

In [84]:

```
Asc = accuracy_score(y_test , prediction )*100
jscr = jaccard_score(y_test , prediction)
f1scr = f1_score(y_test , prediction)


results = pd.DataFrame({'Metric': ['A_SC', 'J_SCR', 'F1_SCR'],
                        'Value': [Asc, jscr, f1scr]})

print(results)
```

```
   Metric      Value
0    A_SC  83.816794
1   J_SCR   0.506977
2  F1_SCR   0.672840
```

In [ ]:

```

```

In [ ]:

```

```