In [1]:

```python
# Import libraries
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
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

In [2]:

```python
# Read the dataset
df = pd.read_csv("Weather_Data.csv")
df
```

Out[2]:

	Date	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed
0	2/1/2008	19.5	22.4	15.6	6.2	0.0	W	41
1	2/2/2008	19.5	25.6	6.0	3.4	2.7	W	41
2	2/3/2008	21.6	24.5	6.6	2.4	0.1	W	41
3	2/4/2008	20.2	22.8	18.8	2.2	0.0	W	41
4	2/5/2008	19.7	25.7	77.4	4.8	0.0	W	41
...	...	...	...	...	...	...	...	...
3266	6/21/2017	8.6	19.6	0.0	2.0	7.8	SSE	37
3267	6/22/2017	9.3	19.2	0.0	2.0	9.2	W	30
3268	6/23/2017	9.4	17.7	0.0	2.4	2.7	W	24
3269	6/24/2017	10.1	19.3	0.0	1.4	9.3	W	43
3270	6/25/2017	7.6	19.3	0.0	3.4	9.4	W	35

3271 rows × 22 columns

In [3]:

```python
df.head()
```

Out[3]:

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

5 rows × 22 columns

In [4]:
```
df.describe()
```

Out[4]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpee
count	3271.000000	3271.000000	3271.000000	3271.000000	3271.000000	3271.000000	3271.0
mean	14.877102	23.005564	3.342158	5.175787	7.168970	41.476307	15.0
std	4.554710	4.483752	9.917746	2.757684	3.815966	10.806951	7.0
min	4.300000	11.700000	0.000000	0.000000	0.000000	17.000000	0.0
25%	11.000000	19.600000	0.000000	3.200000	4.250000	35.000000	11.0
50%	14.900000	22.800000	0.000000	4.800000	8.300000	41.000000	15.0
75%	18.800000	26.000000	1.400000	7.000000	10.200000	44.000000	20.0
max	27.600000	45.800000	119.400000	18.400000	13.600000	96.000000	54.0

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3271 entries, 0 to 3270
Data columns (total 22 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Date           3271 non-null   object
 1   MinTemp        3271 non-null   float64
 2   MaxTemp        3271 non-null   float64
 3   Rainfall       3271 non-null   float64
 4   Evaporation    3271 non-null   float64
 5   Sunshine       3271 non-null   float64
 6   WindGustDir    3271 non-null   object
 7   WindGustSpeed  3271 non-null   int64
 8   WindDir9am     3271 non-null   object
 9   WindDir3pm     3271 non-null   object
 10  WindSpeed9am   3271 non-null   int64
 11  WindSpeed3pm   3271 non-null   int64
 12  Humidity9am    3271 non-null   int64
 13  Humidity3pm    3271 non-null   int64
 14  Pressure9am    3271 non-null   float64
 15  Pressure3pm    3271 non-null   float64
 16  Cloud9am       3271 non-null   int64
 17  Cloud3pm       3271 non-null   int64
 18  Temp9am        3271 non-null   float64
 19  Temp3pm        3271 non-null   float64
 20  RainToday      3271 non-null   object
 21  RainTomorrow   3271 non-null   object
dtypes: float64(9), int64(7), object(6)
memory usage: 562.3+ KB
```

Data Preprocessing

In [6]:
```
# Convert categorical variables to binary variables
df_dummies = pd.get_dummies(data=df, columns=['RainToday', 'WindGustDir', 'WindDi
df_dummies
```

Out[6]:

	Date	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9a
0	2/1/2008	19.5	22.4	15.6	6.2	0.0	41	
1	2/2/2008	19.5	25.6	6.0	3.4	2.7	41	
2	2/3/2008	21.6	24.5	6.6	2.4	0.1	41	
3	2/4/2008	20.2	22.8	18.8	2.2	0.0	41	
4	2/5/2008	19.7	25.7	77.4	4.8	0.0	41	
...	...	...	...	...	...	...	...	
3266	6/21/2017	8.6	19.6	0.0	2.0	7.8	37	
3267	6/22/2017	9.3	19.2	0.0	2.0	9.2	30	
3268	6/23/2017	9.4	17.7	0.0	2.4	2.7	24	
3269	6/24/2017	10.1	19.3	0.0	1.4	9.3	43	
3270	6/25/2017	7.6	19.3	0.0	3.4	9.4	35	

3271 rows × 68 columns

In [7]:
```
df_dummies.replace(['No', 'Yes'], [0,1], inplace=True)
```

In [8]:
```
df_dummies.drop('Date',axis=1,inplace=True)
```

In [9]:
```
df_dummies = df_dummies.astype(float)
```

In [10]:
```
features = df_dummies.drop(columns='RainTomorrow', axis=1)
Y = df_dummies['RainTomorrow']
```

1. Linear Regression

Q.1 Use the train_test_split function to split the features and Y dataframes with a test_size of 0.2 and the random_state set to 10

In [11]:
```
# Split the data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(features, Y, test_size = 0.2,
```

Q.2 Create and train a Linear Regression model called LinearReg using the training data (x_train, y_train).

In [12]:
```
# Create model Linear regression
LinearReg = LinearRegression()
```

In [13]:
```
LinearReg.fit(x_train, y_train)
```

Out[13]:
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Q.3 Now use the predict method on the testing data (x_test) and save it to the array predictions.

In [57]:

```
# Predict output for the x_test dataset
predictions_Linear = LinearReg.predict(x_test)
predictions_Linear
```

```
Out[57]: array([ 1.31782532e-01,  2.76153564e-01,  9.78088379e-01,  2.87483215e-01,
                 1.32377625e-01,  4.60464478e-01,  3.56773376e-01,  8.56460571e-01,
                 6.75010681e-01,  3.82766724e-02,  4.82940674e-03,  2.81181335e-01,
                 3.39042664e-01,  7.80868530e-02,  6.26449585e-02,  5.64521790e-01,
                -6.15615845e-02,  5.24208069e-01,  1.53785706e-01,  3.59672546e-01,
                 6.05087280e-02,  9.03572083e-01,  4.67338562e-01,  2.03323364e-01,
                -7.10830688e-02,  3.83865356e-01,  5.36071777e-01, -2.28652954e-02,
                 6.40052795e-01, -9.56726074e-02,  3.78089905e-01,  1.20277405e-01,
                -1.81350708e-02,  5.53970337e-02,  5.63514709e-01,  1.06302643e+00,
                -6.73675537e-03,  5.14488220e-01, -8.83865356e-02,  6.92062378e-02,
                 2.44827271e-02,  8.71726990e-01,  2.44621277e-01,  3.94760132e-01,
                 2.67494202e-01,  4.46762085e-01, -4.75540161e-02,  1.89407349e-01,
                 7.76573181e-01,  1.57707214e-01,  3.91387939e-03, -5.19638062e-02,
                 2.07328796e-01, -2.07908630e-01, -7.60879517e-02,  2.49641418e-01,
                 2.79273987e-01,  6.02851868e-01,  6.29592896e-01,  4.90715027e-01,
                 5.63888550e-02,  1.05461121e-01,  6.69990540e-01,  7.64839172e-01,
                 9.84802246e-02, -6.33010864e-02,  4.13795471e-01,  7.26165771e-02,
                 2.49977112e-01,  2.56477356e-01,  3.78570557e-02, -6.88705444e-02,
                 3.48220825e-01,  1.73110962e-01,  4.15390015e-01, -7.24411011e-02,
                 3.41178894e-01,  1.86973572e-01, -1.54838562e-01,  7.99736023e-01,
                 2.32406616e-01,  2.37342834e-01,  2.56103516e-01,  1.03439331e-01,
                 6.20689392e-01,  1.29676819e-01,  3.71398926e-01,  4.83024597e-01,
                 3.08250427e-01,  1.19438171e-01,  5.48324585e-02,  1.12785339e-01,
                -1.62445068e-01, -1.57966614e-01,  7.28530884e-02,  7.76443481e-01,
                 8.42971802e-02,  6.43096924e-01,  1.21810913e-01,  1.78077698e-01,
                 3.04893494e-01,  1.85523987e-01,  1.13146210e+00,  3.44383240e-01,
                 8.26858521e-01,  5.38887024e-01,  2.74291992e-01,  6.29371643e-01,
                -6.72454834e-02,  9.63615417e-01,  5.61912537e-01,  2.34298706e-02,
                 1.05628967e-01,  9.38644409e-02,  8.15734863e-02,  2.92030334e-01,
                 5.55122375e-01,  6.69174194e-02,  4.25491333e-02,  9.15527344e-04,
                 1.37611389e-01, -2.18009949e-01,  7.01370239e-02,  1.19743347e-01,
                 1.92710876e-01, -3.83148193e-02,  1.55982971e-01,  7.12615967e-01,
                 2.62329102e-01,  5.34835815e-01,  3.97834778e-01, -4.65545654e-02,
                 2.43972778e-01,  2.06321716e-01,  2.80395508e-01, -1.28204346e-01,
                 1.19529724e-01,  3.83377075e-01,  1.84379578e-01,  7.28614807e-01,
                 3.31779480e-01, -1.03202820e-01,  2.39532471e-01,  3.05999756e-01,
                 6.63154602e-01,  2.21633911e-02,  1.60064697e-01,  5.21621704e-01,
                -4.34417725e-02,  7.98316956e-01,  2.21687317e-01,  3.89877319e-01,
                -5.63583374e-02,  4.02343750e-01,  2.34130859e-01,  1.20452881e-01,
                 3.05458069e-01,  7.86186218e-01,  3.32870483e-02,  1.22553253e+00,
                 8.94706726e-01,  8.18176270e-02, -1.51977539e-02,  1.71012878e-01,
                -6.85882568e-02, -1.07826233e-01, -5.01937866e-02,  2.98667908e-01,
                 3.57048035e-01,  9.57412720e-02,  8.85307312e-01, -3.33786011e-02,
                 5.99205017e-01,  8.94927979e-02,  9.83657837e-02,  3.54164124e-01,
                 2.35893250e-01, -6.72836304e-02,  1.44309998e-01,  4.17808533e-01,
                 5.87440491e-01,  2.16773987e-01, -3.32717896e-02,  1.00634766e+00,
                 4.25872803e-02, -1.74026489e-02,  2.48222351e-01, -5.93338013e-02,
                 2.22259521e-01, -7.19223022e-02,  8.47396851e-02,  7.15713501e-02,
                 3.49349976e-02,  4.43504333e-01,  1.86744690e-01,  1.83174133e-01,
                 7.56149292e-02,  1.36947632e-02,  1.84654236e-01,  2.32337952e-01,
                 5.30570984e-01,  1.60308838e-01,  3.95248413e-01, -4.48989868e-02,
                 8.39263916e-01,  1.65863037e-02, -1.70288086e-02, -2.14080811e-02,
                 1.46392822e-01,  1.38114929e-01, -3.35006714e-02,  4.14924622e-01,
                 6.57295227e-01, -4.89044189e-03,  2.86331177e-02,  4.61502075e-02,
                -7.21664429e-02,  2.73780823e-01,  5.28648376e-01, -5.90286255e-02,
                 8.43177795e-01,  3.25874329e-01,  3.18893433e-01,  3.63616943e-02,
                -8.51821899e-02,  2.83226013e-01,  3.01261902e-01,  1.37733459e-01,
                 3.35823059e-01,  4.12498474e-01,  3.75244141e-01,  1.24153137e-01,
                 2.87879944e-01,  9.06158447e-01,  4.45510864e-01,  2.74833679e-01,
                -6.66046143e-03,  6.91795349e-01,  6.71966553e-01,  1.15348816e-01,
                 7.74536133e-01,  9.94400024e-01,  5.87608337e-01,  3.14361572e-01,
                 2.26425171e-01, -4.47082520e-02,  2.26020813e-01,  2.30567932e-01,
                 3.54957581e-01, -1.26609802e-01,  3.53012085e-02,  1.82228088e-01,
                 2.15400696e-01,  1.76094055e-01, -2.96478271e-02,  1.00051880e+00,
                 1.88224792e-01,  7.52113342e-01,  6.19873047e-01,  7.17811584e-01,
```

```
 3.15574646e-01,  2.23617554e-02,  3.26393127e-01, -6.88934326e-03,
 2.81578064e-01, -2.27432251e-02,  7.98339844e-02, -2.37274170e-03,
 3.57124329e-01,  1.36497498e-01,  2.65785217e-01,  1.40151978e-02,
 2.21122742e-01,  7.07374573e-01,  1.59515381e-01,  2.04536438e-01,
 5.38299561e-01, -1.52595520e-01,  2.54798889e-01,  2.76359558e-01,
 2.64022827e-01,  6.69326782e-02,  9.74380493e-01,  2.00012207e-01,
 7.55783081e-01, -3.49960327e-02,  4.87709045e-01,  5.89752197e-03,
-7.24411011e-02,  3.05664062e-01, -2.05085754e-01,  1.90986633e-01,
-2.46047974e-02, -7.71484375e-02,  2.78923035e-01,  7.84225464e-02,
 2.33840942e-01,  4.44847107e-01,  6.62460327e-02, -3.31573486e-02,
 5.83091736e-01,  5.78460693e-02,  3.20503235e-01,  7.24166870e-01,
 1.35353088e-01, -2.21786499e-02,  7.33055115e-01, -3.08296204e-01,
 1.09135437e+00,  2.85949707e-01,  8.85025024e-01,  6.96632385e-01,
 5.85266113e-01, -2.22396851e-02,  9.52064514e-01,  7.14958191e-01,
 4.48501587e-01,  5.80467224e-01,  1.87095642e-01,  2.29614258e-01,
-5.77468872e-02,  3.15238953e-01,  8.74656677e-01, -1.67694092e-02,
 2.52609253e-01, -3.34930420e-02, -4.39987183e-02,  4.13589478e-02,
 2.35946655e-01,  2.03071594e-01,  8.81996155e-01,  5.87257385e-01,
 1.20094299e-01, -2.02865601e-02, -1.92718506e-02,  1.03446960e-01,
 8.26263428e-03,  2.29148865e-01, -1.69754028e-02,  7.15484619e-02,
 1.62742615e-01,  1.70097351e-01,  2.46543884e-01,  1.47781372e-01,
 3.91838074e-01, -6.66122437e-02,  1.15684509e-01,  1.84471130e-01,
 9.07058716e-02, -1.80282593e-02,  2.09297180e-01, -1.30462646e-02,
 5.46470642e-01,  4.15779114e-01,  1.78169250e-01,  1.72042847e-02,
 4.93850708e-01,  1.47155762e-01,  5.24894714e-01,  5.98114014e-01,
 3.08280945e-01,  3.09295654e-01,  4.46388245e-01,  1.68220520e-01,
 2.60719299e-01,  7.56401062e-01,  4.06272888e-01,  4.90669250e-01,
 5.14297485e-01, -1.30180359e-01,  7.90176392e-02,  3.50608826e-01,
 1.00466919e+00,  2.73452759e-01,  6.35429382e-01,  5.78605652e-01,
 4.68185425e-01,  3.74603271e-03,  3.59062195e-01,  6.74667358e-02,
-4.23889160e-02,  7.03315735e-01,  1.27159119e-01,  8.00025940e-01,
 4.70840454e-01, -1.74484253e-02,  2.93350220e-02,  5.55259705e-01,
 5.01327515e-02,  1.15348816e-01, -9.69696045e-03,  1.47483826e-01,
 4.71153259e-01, -9.27658081e-02,  8.09402466e-02,  3.10745239e-02,
 3.94844055e-01,  2.23731995e-01,  1.00329590e+00,  7.08076477e-01,
 3.56101990e-01,  2.62931824e-01, -2.83889771e-02, -9.58251953e-03,
 3.09677124e-02,  1.41662598e-01,  7.82546997e-02,  5.12550354e-01,
 2.04780579e-01,  3.11676025e-01,  5.28495789e-01,  4.61898804e-01,
 5.35964966e-02,  8.17260742e-01,  5.33927917e-01, -1.38145447e-01,
 5.09429932e-01,  1.94168091e-01,  4.56214905e-01,  1.85157776e-01,
 2.21946716e-01,  1.15570068e-01,  6.33010864e-02,  7.36923218e-02,
 1.02340698e-01,  5.51345825e-01,  7.02964783e-01,  5.22079468e-01,
 5.35041809e-01,  3.19290161e-02,  1.11122131e-01, -1.18644714e-01,
 5.31913757e-01, -1.74400330e-01,  4.54521179e-01,  9.74418640e-01,
 4.18205261e-01, -1.16004944e-01,  8.70658875e-01,  1.62162781e-01,
 6.13059998e-01,  3.17138672e-01, -7.86209106e-02,  4.93980408e-01,
 7.44834900e-01,  6.05392456e-02,  6.03713989e-02,  2.36778259e-01,
 5.79223633e-01,  1.10740662e-01,  3.81782532e-01,  6.02279663e-01,
 5.84182739e-02,  2.21824646e-01,  3.88992310e-01,  6.63612366e-01,
 1.56120300e-01,  3.97300720e-01,  5.80154419e-01,  5.03753662e-01,
 1.06445312e-01,  1.49841309e-01,  2.33238220e-01, -3.34167480e-03,
-2.85873413e-02,  2.65953064e-01,  2.60597229e-01, -6.16302490e-02,
 1.46217346e-01,  8.24829102e-01,  5.68481445e-01, -9.56268311e-02,
-1.02455139e-01,  8.50067139e-02,  1.61598206e-01, -7.92617798e-02,
 9.43962097e-01,  7.94296265e-02,  3.38417053e-01,  1.57981873e-01,
-6.92901611e-02,  2.15110779e-01,  4.78790283e-01,  2.15377808e-02,
 5.34416199e-01,  1.42539978e-01, -1.58424377e-01,  1.00021362e-02,
 2.26768494e-01,  3.34747314e-01,  6.04705811e-02, -2.08816528e-02,
-1.07841492e-01,  3.34335327e-01,  5.10124207e-01,  3.45932007e-01,
-8.66928101e-02, -4.99572754e-02,  5.05302429e-01,  8.22601318e-02,
 6.74285889e-02,  3.23204041e-01,  3.38134766e-01,  5.37300110e-01,
 2.91786194e-01,  3.30497742e-01, -4.09545898e-02,  8.27468872e-01,
 6.03424072e-01,  2.36755371e-01,  7.02346802e-01,  4.44656372e-01,
 5.81176758e-01,  1.90055847e-01,  5.45578003e-02,  9.38339233e-02,
-6.76956177e-02,  1.05995178e-01, -2.49252319e-02, -2.94113159e-02,
 6.56051636e-02,  9.61074829e-02,  1.03271484e-01,  1.77772522e-01,
```

```
        1.02096558e-01,  6.64031982e-01,  1.97700500e-01,  1.04774475e-01,
        1.13456726e-01, -3.24630737e-02,  1.46598816e-01,  4.64958191e-01,
        7.55508423e-01,  2.03941345e-01,  4.90135193e-01, -1.01310730e-01,
        6.61643982e-01,  7.53471375e-01,  9.01084900e-01,  7.72949219e-01,
        2.73429871e-01,  2.98088074e-01,  1.87126160e-01,  7.19902039e-01,
        2.29797363e-01,  3.34632874e-01,  8.51608276e-01,  3.06320190e-02,
       -3.72314453e-03, -7.81478882e-02,  7.49839783e-01,  2.04597473e-01,
        4.06028748e-01,  3.32359314e-01,  1.81953430e-01,  8.23410034e-01,
        5.03158569e-01,  6.34071350e-01, -5.07736206e-02,  4.25796509e-02,
        1.11099243e-01,  1.89201355e-01,  6.00975037e-01,  4.09095764e-01,
        6.87751770e-01,  2.89863586e-01, -9.21096802e-02,  4.46624756e-02,
        1.98905945e-01,  6.18957520e-01,  1.41899109e-01,  6.98921204e-01,
        3.11897278e-01,  1.26914978e-01,  8.11447144e-01,  6.24542236e-02,
       -1.04484558e-01,  6.60018921e-02,  1.23748779e-01,  5.18707275e-01,
       -5.96847534e-02,  6.39770508e-01, -1.30081177e-02,  3.84140015e-02,
       -1.04942322e-01,  1.96571350e-01,  3.36097717e-01,  4.87998962e-01,
        1.93817139e-01,  7.93266296e-01,  1.69387817e-01,  8.55941772e-02,
        1.68647766e-01,  1.73156738e-01,  3.23867798e-02,  4.69230652e-01,
        2.03071594e-01,  8.46679688e-01,  4.20898438e-01,  2.17918396e-01,
        1.81129456e-01,  1.54930115e-01,  7.26310730e-01,  4.45205688e-01,
        4.87533569e-01,  1.01577759e-01,  4.13154602e-01, -1.13243103e-01,
        1.79405212e-01, -9.75036621e-03,  1.39091492e-01,  1.12022400e-01,
        6.09596252e-01,  2.16674805e-02,  1.88301086e-01, -1.35353088e-01,
        7.06481934e-01,  4.33082581e-01, -5.83572388e-02,  9.28695679e-01,
        5.40626526e-01, -1.65557861e-03,  9.19029236e-01, -2.18505859e-02,
        1.65367126e-01,  1.77780151e-01,  4.72900391e-01, -2.81524658e-03,
        4.00382996e-01,  1.42211914e-02,  1.95396423e-01, -6.79855347e-02,
        5.87219238e-01,  1.92474365e-01,  2.33268738e-01,  2.73651123e-01,
        9.40742493e-01,  7.10136414e-01,  1.33438110e-01,  7.37457275e-02,
        5.58242798e-02, -2.26371765e-01,  4.10919189e-02,  5.16891479e-02,
        3.09066772e-02,  2.93563843e-01,  9.69337463e-01,  7.98675537e-01,
       -2.29568481e-02,  5.26657104e-02,  2.64755249e-01,  1.75148010e-01,
        7.06344604e-01,  1.73805237e-01,  3.44337463e-01])
```

Q.4 Using the predictions and the y_test dataframe calculate the value for each metric using the appropriate function.

In [58]:

```python
# Mean absoulate error of linear regression
LinearRegression_MAE = mean_absolute_error(y_test, predictions_Linear)
print("Mean Absolute Error (MAE):", LinearRegression_MAE)
```

```
Mean Absolute Error (MAE): 0.25631900234076815
```

In [59]:

```python
# Mean absoulate error of linear regression
LinearRegression_MSE = mean_squared_error(y_test, predictions_Linear)
print("Mean Absolute Error (MSE):", LinearRegression_MSE)
```

```
Mean Absolute Error (MSE): 0.11572291759564364
```

In [60]:

```python
# R2 score of linear regression
LinearRegression_R2 = r2_score(y_test, predictions_Linear)
print("R-squared (R2):", LinearRegression_R2)
```

```
R-squared (R2): 0.4271205492306953
```

Q.5 Show the MAE, MSE, and R2 in a tabular format using data frame for the linear model

In [86]:

```
# Create a dictionary with metrics
Report_Linear = {
    'Metric': ['Mean Absolute Error (MAE)', 'Mean Squared Error (MSE)', 'R-square
    'Value': [LinearRegression_MAE, LinearRegression_MSE, LinearRegression_R2]
}

# Create a DataFrame
Report_Linear_df = pd.DataFrame(Report_Linear)

print(Report_Linear_df)
```

```
                      Metric     Value
0  Mean Absolute Error (MAE)  0.256319
1   Mean Squared Error (MSE)  0.115723
2             R-squared (R2)  0.427121
```

2. KNN

In [19]:

```
# Split the data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(features, Y, test_size = 0.2,
```

Q.6 Create and train a KNN model called KNN using the training data (x_train, y_train) with the n_neighbors parameter set to 4

In [62]:

```
k = 4

#Train Model and Predict
KNN = KNeighborsClassifier(n_neighbors = k).fit(x_train, y_train)
KNN
```

Out[62]:

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

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Q.7 Now use the predict method on the testing data (x_test) and save it to the array predictions.

In [63]:
```python
# Predicting the Test set results
predictions_KNN = KNN.predict(x_test)
predictions_KNN
```

Out[63]:
```
array([0., 0., 1., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
       0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0.,
       1., 0., 1., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0.,
       0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0.,
       0., 0., 0., 0., 1., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
       1., 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., 0., 0., 0., 0., 0., 0.,
       1., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 1.,
       1., 0., 1., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       1., 0., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0.,
       1., 0., 1., 0., 1., 1., 0., 0., 1., 0., 0., 0., 1., 0., 0., 1., 1.,
       0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
       0., 1., 1., 1., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0.,
       0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0.,
       0., 1., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       1., 0., 0., 1., 0., 0., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0., 0.,
       0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
       1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
       0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 1., 1., 1., 0., 0., 0., 1.,
       0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0.,
       0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
       0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       1., 0., 0., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 1., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
       1., 1., 0., 0., 0., 0., 1., 0., 1.])
```

Q.8 Using the predictions and the y_test dataframe calculate the value for each metric using the appropriate function.

In [64]:
```python
# KNN accuracy score
KNN_Accuracy_Score  = accuracy_score(y_test, predictions_KNN)
print("KNN accuracy score: ",KNN_Accuracy_Score)
```

```
KNN accuracy score:  0.8183206106870229
```

In [65]:
```python
# KNN jaccard index
jaccard_index_KNN = jaccard_score(y_test, predictions_KNN)
print("KNN Jaccard Index:", jaccard_index_KNN)
```

```
KNN Jaccard Index: 0.4251207729468599
```

In [66]:
```python
# KNN F1 score
f1_KNN = f1_score(y_test, predictions_KNN)
print("KNN F1 Score:", f1_KNN)
```

```
KNN F1 Score: 0.5966101694915255
```

In [67]:
```python
# Create a dictionary with metrics
Report_KNN = {
    'Metric': ['KNN accuracy score', 'KNN Jaccard Index', 'KNN F1 Score'],
    'Value': [KNN_Accuracy_Score, jaccard_index, f1_KNN]
}

# Create a DataFrame
Report_KNN_df = pd.DataFrame(Report_KNN)

print(Report_KNN_df)
```

```
               Metric     Value
0  KNN accuracy score  0.818321
1   KNN Jaccard Index  0.425121
2        KNN F1 Score  0.596610
```

3. Decision Tree

Q.9 Create and train a Decision Tree model called Tree using the training data (x_train, y_train).

In [68]:
```python
# Split the data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(features, Y, test_size = 0.2,
```

In [31]:
```python
Tree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)
Tree
```

Out[31]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [32]:
```python
Tree.fit(x_train, y_train)
```

Out[32]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Q.10 Now use the predict method on the testing data (x_test) and save it to the array predictions.

In [69]:
```python
predictions_Tree = Tree.predict(x_test)
predictions_Tree
```

Out[69]:
```
array([0., 0., 1., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
       1., 0., 0., 0., 1., 0., 0., 0., 1., 1., 0., 1., 0., 1., 0., 0., 0.,
       0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 1., 0., 0.,
       0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 1., 1., 0., 0., 1., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0.,
       0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0.,
       1., 0., 1., 1., 0., 1., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0.,
       0., 0., 0., 0., 1., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       1., 0., 1., 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., 0., 0., 1., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 0., 0., 1.,
       1., 0., 1., 1., 1., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
       1., 0., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
       0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0.,
       1., 0., 1., 0., 1., 1., 1., 0., 1., 1., 1., 1., 0., 0., 0., 0., 1.,
       0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
       0., 1., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0.,
       1., 1., 0., 0., 0., 0., 0., 1., 0., 1., 1., 0., 0., 1., 0., 0., 0.,
       0., 1., 0., 0., 0., 0., 0., 1., 1., 1., 0., 0., 0., 0., 0., 0., 1.,
       0., 0., 1., 1., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
       1., 1., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 1., 0.,
       0., 1., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1.,
       0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
       1., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 1., 1., 0.,
       1., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 1., 1., 0., 0., 0., 1., 1., 1., 1., 0., 0., 0., 1.,
       0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0.,
       0., 1., 1., 1., 0., 0., 0., 0., 1., 0., 1., 0., 0., 1., 0., 0., 0.,
       0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
       1., 0., 1., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
       1., 0., 0., 0., 1., 0., 0., 1., 1., 0., 1., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 1., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
       1., 1., 0., 0., 0., 0., 1., 0., 0.])
```

Q.11 Using the predictions and the y_test dataframe calculate the value for each metric using the appropriate function.

In [37]:
```python
# Tree accuracy score
Tree_Accuracy_Score = accuracy_score(y_test, predictions_Tree)
print("Tree accuracy score:", Tree_Accuracy_Score)
```

```
Tree accuracy score: 0.8183206106870229
```

In [35]:
```python
Tree_JaccardIndex = jaccard_score(y_test, predictions_Tree)
print("Tree jaccard Index:", Tree_JaccardIndex)
```

```
Tree jaccard Index: 0.48034934497816595
```

In [36]:
```python
Tree_F1_Score = f1_score(y_test, predictions_Tree)
print("Tree F1 score:", Tree_F1_Score)
```

```
Tree F1 score: 0.6489675516224188
```

In [38]:
```python
# Create a dictionary with metrics
Report_Tree = {
    'Metric': ['Tree accuracy score', 'Tree jaccard Index', 'Tree F1 score'],
    'Value': [Tree_Accuracy_Score, Tree_JaccardIndex, Tree_F1_Score]
}

# Create a DataFrame
Report_Tree_df = pd.DataFrame(Report_Tree)

print(Report_Tree_df)
```

```
                Metric     Value
0  Tree accuracy score  0.818321
1   Tree jaccard Index  0.480349
2        Tree F1 score  0.648968
```

4. Logistic Regression(LR)

Q.12 Use the train_test_split function to split the features and Y dataframes with a test_size of 0.2 and the random_state set to 1

In [70]:
```python
# Split the data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(features, Y, test_size = 0.2,
```

Q.13 Create and train a LogisticRegression model called LR using the training data (x_train, y_train) with the solver parameter set to liblinear.

In [73]:
```python
# Initialize Logistic Regression model with solver='liblinear'
LR = LogisticRegression(solver='liblinear')
```

In [74]:
```python
# Fit the model on training data
LR.fit(x_train, y_train)
```

Out[74]:
```
LogisticRegression(solver='liblinear')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Q.14 Now, use the predict method on the testing data (x_test) and save it to the array predictions.

In [75]:
```python
# Use the predict method on the testing data
predictions_LR = LR.predict(x_test)
predictions_LR
```

Out[75]:
```
array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 1.,
       0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
       1., 1., 1., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
       0., 1., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
       0., 0., 0., 1., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1.,
       0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 1., 0., 0.,
       0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1.,
       0., 0., 0., 1., 0., 1., 0., 0., 0., 1., 0., 1., 1., 0., 0., 0., 1.,
       0., 0., 1., 0., 0., 1., 0., 1., 0., 1., 1., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 1., 1., 0.,
       0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0., 0.,
       0., 1., 1., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0.,
       1., 0., 1., 1., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0.,
       0., 1., 0., 0., 1., 0., 0., 0., 1., 1., 1., 0., 0., 0., 0., 0., 1.,
       0., 0., 0., 0., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 1., 1., 0., 0., 0., 0.,
       1., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1.,
       1., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 1., 0., 0., 1., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
       1., 1., 0., 0., 1., 1., 1., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0.,
       0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 1., 1.,
       1., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
       0., 1., 0., 0., 1., 1., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0.,
       1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 1., 0., 1.,
       0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 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., 1., 0., 1., 0., 0., 1., 0., 0., 0., 0.,
       0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
       0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
       0., 0., 0., 1., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0.,
       0., 0., 1., 0., 0., 1., 1., 0., 0., 0., 1., 0., 1., 0., 0., 1., 0.,
       0., 0., 1., 0., 1., 0., 0., 0., 1.])
```

Q.15 Using the predictions and the y_test dataframe calculate the value for each metric using the appropriate function.

In [76]:
```python
# Calculate Accuracy Score
LR_accuracy_score = accuracy_score(y_test, predictions_LR)
print("LR Accuracy Score:", LR_accuracy_score)
```

```
LR Accuracy Score: 0.8366412213740458
```

In [77]:
```
# Calculate Jaccard Index
LR_jaccard_index = jaccard_score(y_test, predictions_LR)
print("LR Jaccard Index:", LR_jaccard_index)
```

```
LR Jaccard Index: 0.5091743119266054
```

In [78]:
```
# Calculate F1 Score
LR_f1_score = f1_score(y_test, predictions_LR)
print("LR F1 Score:", LR_f1_score)
```

```
LR F1 Score: 0.6747720364741641
```

In [79]:
```
# Calculate Log Loss
LR_log_loss = log_loss(y_test, LR.predict_proba(x_test))
print("LR Log Loss:", LR_log_loss)
```

```
LR Log Loss: 0.38106374371303714
```

In [80]:
```
# Create a dictionary with metrics
Report_LR = {
    'Metric': ['LR Accuracy Score', 'LR Jaccard Index', 'LR F1 Score', 'LR Log Lo
    'Value': [LR_accuracy_score, LR_jaccard_index, LR_f1_score, LR_log_loss]
}

# Create a DataFrame
Report_LR_df = pd.DataFrame(Report_LR)

print(Report_LR_df)
```

```
              Metric     Value
0  LR Accuracy Score  0.836641
1   LR Jaccard Index  0.509174
2        LR F1 Score  0.674772
3        LR Log Loss  0.381064
```

5. Support Vector Machine(SVM)

Q.16 Create and train a SVM model called SVM using the training data (x_train, y_train).

In [81]:
```
# Split the data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(features, Y, test_size = 0.2,
```

In [82]:
```
svm = SVC(kernel='rbf')
svm.fit(x_train, y_train)
```

Out[82]:
```
SVC()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Q.17 Now use the predict method on the testing data (x_test) and save it to the array predictions.

In [84]:
```python
# Use the predict method on the testing data
predictions_SVM = svm.predict(x_test)
predictions_SVM
```

Out[84]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

Q.18 Using the predictions and the y_test dataframe calculate the value for each metric using the appropriate function.

In [85]:
```python
# Calculate Accuracy Score
SVM_accuracy_score = accuracy_score(y_test, predictions_SVM)
print("SVM Accuracy Score:", SVM_accuracy_score)
```

```
SVM Accuracy Score: 0.7190839694656489
```

In [54]:

```python
# Calculate Jaccard Index
SVM_jaccard_index = jaccard_score(y_test, predictions_SVM)
print("SVM Jaccard Index:", SVM_jaccard_index)
```

```
SVM Jaccard Index: 0.0
```

In [55]:

```python
# Calculate F1 Score
SVM_f1_score = f1_score(y_test, predictions_SVM)
print("SVM F1 Score:", SVM_f1_score)
```

```
SVM F1 Score: 0.0
```

Report

Q.19 Show the Accuracy,Jaccard Index,F1-Score and LogLoss in a tabular format using data frame for all of the above models.

In [93]:

```python
# Create a dictionary with metrics
Report = {
    'Metric': ['Mean Absolute Error(MAE) Linear Regression', 'Mean Squared Error(
    'Value': [LinearRegression_MAE, LinearRegression_MSE, LinearRegression_R2, KN
}

# Create a DataFrame
Report_df = pd.DataFrame(Report)

print(Report_df)
```

```
                                        Metric     Value
0   Mean Absolute Error(MAE) Linear Regression  0.256319
1    Mean Squared Error(MSE) Linear Regression  0.115723
2              R-squared(R2) Linear Regression  0.427121
3                           KNN accuracy score  0.818321
4                            KNN Jaccard Index  0.425121
5                                 KNN F1 Score  0.596610
6                          Tree accuracy score  0.818321
7                           Tree jaccard Index  0.480349
8                                Tree F1 score  0.648968
9           Logistic Regression Accuracy Score  0.836641
10           Logistic Regression Jaccard Index  0.509174
11                Logistic Regression F1 Score  0.674772
12                Logistic Regression Log Loss  0.381064
13                          SVM Accuracy Score  0.719084
14                           SVM Jaccard Index  0.000000
15                                SVM F1 Score  0.000000
```

In [ ]: