Avail Lean Manufacturing to Abate Heart Failure deaths with Machine Learning prognostication

Jupyter Notebook version 7.2.2

[1]:
```
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import csv
```

The original data is being stored in a variable called Original_Data.

[3]:
```
Original_Data = pd.read_csv("heart_failure_clinical_records_dataset.csv")
```

The first five observations are displayed below.

[6]:
```
Original_Data.head()
```

[6]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	DEATH_EVENT
0	75.0	0	582	0	20	1	265000.00	1.9	130	1	0	4	1
1	55.0	0	7861	0	38	0	263358.03	1.1	136	1	0	6	1
2	65.0	0	146	0	20	0	162000.00	1.3	129	1	1	7	1
3	50.0	1	111	0	20	0	210000.00	1.9	137	1	0	7	1
4	65.0	1	160	1	20	0	327000.00	2.7	116	0	0	8	1

This data has 299 rows and 13 columns.

[9]:
```
Original_Data.shape
```

[9]:
```
(299, 13)
```

Some of the scenarios will use all 12 factors without any scaling. This is represented with the variable X_Predictors_12_NoScaling.

[12]:
```
X_Predictors_12_NoScaling = Original_Data.loc[:, Original_Data.columns != "DEATH_EVENT"]
X_Predictors_12_NoScaling.head()
```

[12]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time
0	75.0	0	582	0	20	1	265000.00	1.9	130	1	0	4
1	55.0	0	7861	0	38	0	263358.03	1.1	136	1	0	6
2	65.0	0	146	0	20	0	162000.00	1.3	129	1	1	7
3	50.0	1	111	0	20	0	210000.00	1.9	137	1	0	7
4	65.0	1	160	1	20	0	327000.00	2.7	116	0	0	8

The variable X_Predictors_12_NoScaling only has 12 columns because this variable only represents the predictor variables and not the response variable.

[15]:
```
X_Predictors_12_NoScaling.shape
```

[15]:
```
(299, 12)
```

Some of the scenarios will use all 12 factors with scaling. This is represented with the variable X_Predictors_12_Scaled. The below website provides more information about the type of scaling that is being used.

https://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.StandardScaler.html

[18]:
```
Scaler1 = StandardScaler()
Scaled_Data = Scaler1.fit_transform(X_Predictors_12_NoScaling)
```

[20]:
```
X_Predictors_12_Scaled = pd.DataFrame(Scaled_Data)
X_Predictors_12_Scaled.head()
```

[20]:

	0	1	2	3	4	5	6	7	8	9	10	11
0	1.192945	-0.871105	0.000166	-0.847579	-1.530560	1.359272	1.681648e-02	0.490057	-1.504036	0.735688	-0.687682	-1.629502
1	-0.491279	-0.871105	7.514640	-0.847579	-0.007077	-0.735688	7.535660e-09	-0.284552	-0.141976	0.735688	-0.687682	-1.603691
2	0.350833	-0.871105	-0.449939	-0.847579	-1.530560	-0.735688	-1.038073e+00	-0.090900	-1.731046	0.735688	1.454161	-1.590785
3	-0.912335	1.147968	-0.486071	-0.847579	-1.530560	-0.735688	-5.464741e-01	0.490057	0.085034	0.735688	-0.687682	-1.590785
4	0.350833	1.147968	-0.435486	1.179830	-1.530560	-0.735688	6.517986e-01	1.264666	-4.682176	-1.359272	-0.687682	-1.577879

As shown above, the name of the columns changed after scaling, but the columns are still in the same order.

For example, age = 0, anaemia = 1, creatinine_phosphokinase = 2, diabetes = 3, ejection_fraction = 4, etc.

Using Minitab, 5 predictor variables were shown to have a high correlation with the response variable. Some of the scenarios will consider only these 5 variables.

[24]:
```
X_Predictors_5_NoScaling = Original_Data.loc[:, Original_Data.columns != "DEATH_EVENT"]
X_Predictors_5_NoScaling = X_Predictors_5_NoScaling.drop(columns = ['anaemia','creatinine_phosphokinase',
                                                                    'diabetes','high_blood_pressure','platelets',
                                                                    'sex','smoking'])
X_Predictors_5_NoScaling.head()
#https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html
```

[24]:

	age	ejection_fraction	serum_creatinine	serum_sodium	time
0	75.0	20	1.9	130	4
1	55.0	38	1.1	136	6
2	65.0	20	1.3	129	7
3	50.0	20	1.9	137	7
4	65.0	20	2.7	116	8

As shown above, the variable X_Predictors_5_NoScaling has 5 predictor variables and is not scaled.

Next, those 5 predictor variables will be scaled as shown below.

[27]:
```
Scaler2 = StandardScaler()
Scaled_Data2 = Scaler2.fit_transform(X_Predictors_5_NoScaling)
```

[29]:
```
X_Predictors_5_Scaled = pd.DataFrame(Scaled_Data2)
X_Predictors_5_Scaled.head()
```

[29]:

	0	1	2	3	4
0	1.192945	-1.530560	0.490057	-1.504036	-1.629502
1	-0.491279	-0.007077	-0.284552	-0.141976	-1.603691
2	0.350833	-1.530560	-0.090900	-1.731046	-1.590785
3	-0.912335	-1.530560	0.490057	0.085034	-1.590785
4	0.350833	-1.530560	1.264666	-4.682176	-1.577879

The variable X_Predictors_5_Scaled has the same column names as X_Predictors_5_NoScaling.

For example, age = 0, ejection_fraction = 1, serum_creatinine = 2, serum_sodium = 3, and time = 4.

Next, the correlation between the predictor variables (not including the response variable) will be considered.

```
[33]: Correlation_Between_Factors = X_Predictors_12_NoScaling.corr()
      Correlation_Between_Factors
```

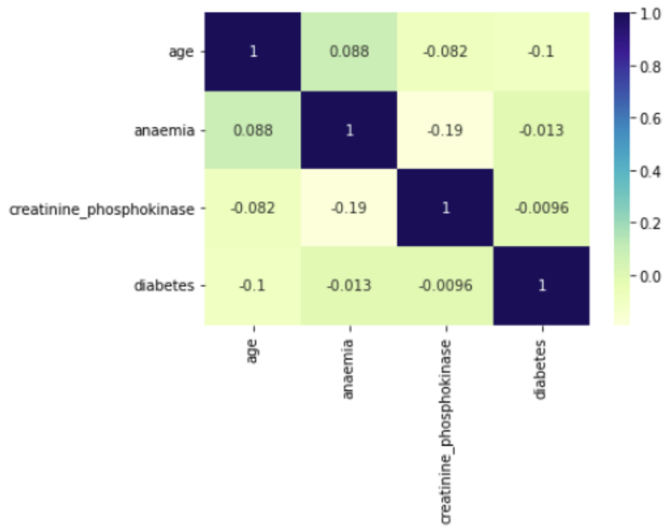
[33]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time
age	1.000000	0.088006	-0.081584	-0.101012	0.060098	0.093289	-0.052354	0.159187	-0.045966	0.065430	0.018668	-0.224068
anaemia	0.088006	1.000000	-0.190741	-0.012729	0.031557	0.038182	-0.043786	0.052174	0.041882	-0.094769	-0.107290	-0.141414
creatinine_phosphokinase	-0.081584	-0.190741	1.000000	-0.009639	-0.044080	-0.070590	0.024463	-0.016408	0.059550	0.079791	0.002421	-0.009346
diabetes	-0.101012	-0.012729	-0.009639	1.000000	-0.004850	-0.012732	0.092193	-0.046975	-0.089551	-0.157730	-0.147173	0.033726
ejection_fraction	0.060098	0.031557	-0.044080	-0.004850	1.000000	0.024445	0.072177	-0.011302	0.175902	-0.148386	-0.067315	0.041729
high_blood_pressure	0.093289	0.038182	-0.070590	-0.012732	0.024445	1.000000	0.049963	-0.004935	0.037109	-0.104615	-0.055711	-0.196439
platelets	-0.052354	-0.043786	0.024463	0.092193	0.072177	0.049963	1.000000	-0.041198	0.062125	-0.125120	0.028234	0.010514
serum_creatinine	0.159187	0.052174	-0.016408	-0.046975	-0.011302	-0.004935	-0.041198	1.000000	-0.189095	0.006970	-0.027414	-0.149315
serum_sodium	-0.045966	0.041882	0.059550	-0.089551	0.175902	0.037109	0.062125	-0.189095	1.000000	-0.027566	0.004813	0.087640
sex	0.065430	-0.094769	0.079791	-0.157730	-0.148386	-0.104615	-0.125120	0.006970	-0.027566	1.000000	0.445892	-0.015608
smoking	0.018668	-0.107290	0.002421	-0.147173	-0.067315	-0.055711	0.028234	-0.027414	0.004813	0.445892	1.000000	-0.022839
time	-0.224068	-0.141414	-0.009346	0.033726	0.041729	-0.196439	0.010514	-0.149315	0.087640	-0.015608	-0.022839	1.000000

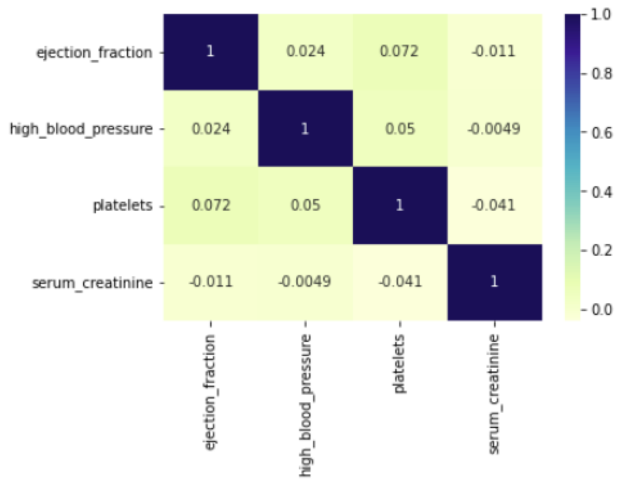
```
[35]: import matplotlib.pyplot as mp
      import pandas as pd
      import seaborn as sb
```

The above correlation table will be represented with a heat map as shown below.

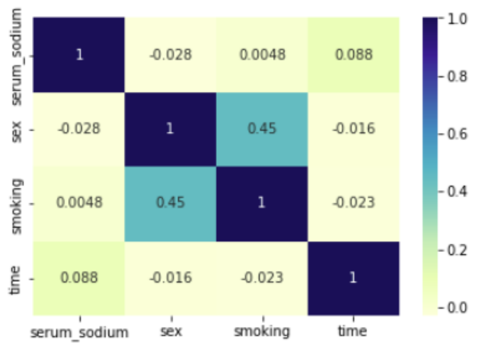
```
[38]: import seaborn as sns
      dataplot = sns.heatmap(X_Predictors_12_NoScaling.iloc[:,:4].corr(), cmap = "YlGnBu", annot = True)

      #https://www.geeksforgeeks.org/enhancing-seaborn-heatmaps-techniques-to-increase-size-for-better-clarity/
      #https://stackoverflow.com/questions/34706845/change-xticklabels-fontsize-of-seaborn-heatmap
      #https://seaborn.pydata.org/tutorial/introduction.html
```



```
[40]: dataplot = sns.heatmap(X_Predictors_12_NoScaling.iloc[:,4:8].corr(), cmap = "YlGnBu", annot = True)
```



```
[42]: dataplot = sns.heatmap(X_Predictors_12_NoScaling.iloc[:,8:12].corr(), cmap = "YlGnBu", annot = True)
```



```
[44]: Y_Response_Variable = Original_Data.loc[:, Original_Data.columns == "DEATH_EVENT"]
      Y_Response_Variable.head()
```

[44]:

	DEATH_EVENT
0	1
1	1
2	1
3	1
4	1

```
[46]: Y_Response_Variable.shape
```

```
[46]: (299, 1)
```

The Death_Event response variable is stored in the variable Y_Response_Variable.

As shown above, the Y_Response_Variable has 299 rows and 1 column.

There are eight scenarios shown below. For each scenario, the data will be split into two groups. The first group is used for training the model. The second group is used for testing the model.

Scenario 1

```
[51]: x_train1, x_test1, y_train1, y_test1 = train_test_split(X_Predictors_12_NoScaling,
                                                                Y_Response_Variable, train_size = 0.80, random_state = 1)
```

```
[53]: x_train1.shape
```

```
[53]: (239, 12)
```

```
[55]: x_test1.shape
```

```
[55]: (60, 12)
```

Scenario 1 has all 12 predictor variables without scaling and is using 80% of the data for training and 20% for testing. Out of 299 data points, 239 are being used for training, while 60 are being used for testing as shown above.

The below code will export scenario 1 from Python to four Excel files. These files have been submitted with this project.

```
[59]: x_train1.to_excel('x_train1.xlsx')
      y_train1.to_excel('y_train1.xlsx')
      x_test1.to_excel('x_test1.xlsx')
      y_test1.to_excel('y_test1.xlsx')
```

Scenario 2

```
[62]: x_train2, x_test2, y_train2, y_test2 = train_test_split(X_Predictors_12_NoScaling,
                                                                Y_Response_Variable, train_size = 0.90, random_state = 1)
```

```
[64]: x_train2.shape
```

```
[64]: (269, 12)
```

```
[66]: x_test2.shape
```

```
[66]: (30, 12)
```

Scenario 2 has all 12 predictor variables without scaling and is using 90% of the data for training and 10% for testing. Out of 299 data points, 269 are being used for training, while 30 are being used for testing as shown above.

```
[69]: x_train2.to_excel('x_train2.xlsx')
      y_train2.to_excel('y_train2.xlsx')
      x_test2.to_excel('x_test2.xlsx')
      y_test2.to_excel('y_test2.xlsx')
```

Scenario 3

```
[72]: x_train3, x_test3, y_train3, y_test3 = train_test_split(X_Predictors_12_Scaled,
                                                                Y_Response_Variable, train_size = 0.80, random_state = 1)
```

Scenario 3 has all 12 predictor variables with scaling and is using 80% of the data for training and 20% for testing.

```
[75]: x_train3.to_excel('x_train3.xlsx')
      y_train3.to_excel('y_train3.xlsx')
      x_test3.to_excel('x_test3.xlsx')
      y_test3.to_excel('y_test3.xlsx')
```

Scenario 4

```
[78]: x_train4, x_test4, y_train4, y_test4 = train_test_split(X_Predictors_12_Scaled,
                                                                Y_Response_Variable, train_size = 0.90, random_state = 1)
```

Scenario 4 has all 12 predictor variables with scaling and is using 90% of the data for training and 10% for testing.

```
[81]: x_train4.to_excel('x_train4.xlsx')
      y_train4.to_excel('y_train4.xlsx')
      x_test4.to_excel('x_test4.xlsx')
      y_test4.to_excel('y_test4.xlsx')
```

Scenario 5

[84]:
```
x_train5, x_test5, y_train5, y_test5 = train_test_split(X_Predictors_5_NoScaling,
                                                        Y_Response_Variable, train_size = 0.80, random_state = 1)
```

Scenario 5 has 5 predictor variables without scaling and is using 80% of the data for training and 20% for testing.

[87]:
```
x_train5.to_excel('x_train5.xlsx')
y_train5.to_excel('y_train5.xlsx')
x_test5.to_excel('x_test5.xlsx')
y_test5.to_excel('y_test5.xlsx')
```

Scenario 6

[90]:
```
x_train6, x_test6, y_train6, y_test6 = train_test_split(X_Predictors_5_NoScaling,
                                                        Y_Response_Variable, train_size = 0.90, random_state = 1)
```

Scenario 6 has 5 predictor variables without scaling and is using 90% of the data for training and 10% for testing.

[93]:
```
x_train6.to_excel('x_train6.xlsx')
y_train6.to_excel('y_train6.xlsx')
x_test6.to_excel('x_test6.xlsx')
y_test6.to_excel('y_test6.xlsx')
```

Scenario 7

[96]:
```
x_train7, x_test7, y_train7, y_test7 = train_test_split(X_Predictors_5_Scaled,
                                                        Y_Response_Variable, train_size = 0.80, random_state = 1)
```

Scenario 7 has 5 predictor variables with scaling and is using 80% of the data for training and 20% for testing.

[99]:
```
x_train7.to_excel('x_train7.xlsx')
y_train7.to_excel('y_train7.xlsx')
x_test7.to_excel('x_test7.xlsx')
y_test7.to_excel('y_test7.xlsx')
```

Scenario 8

[102...
```
x_train8, x_test8, y_train8, y_test8 = train_test_split(X_Predictors_5_Scaled,
                                                        Y_Response_Variable, train_size = 0.90, random_state = 1)
```

Scenario 8 has 5 predictor variables with scaling and is using 90% of the data for training and 10% for testing.

[105...
```
x_train8.to_excel('x_train8.xlsx')
y_train8.to_excel('y_train8.xlsx')
x_test8.to_excel('x_test8.xlsx')
y_test8.to_excel('y_test8.xlsx')
```

The below 8 scenarios are using a Multi-layer Perceptron (MLP)

Scenario 1 (MLP)

[109...
```
from sklearn.neural_network import MLPClassifier
from sklearn import metrics
```

[111...
```
import numpy as np
```

The ravel function is used to change the shape of y_train1 from a column-vector to a 1d array.

https://numpy.org/doc/stable/reference/generated/numpy.ravel.html

[114...
```
y_train1 = np.ravel(y_train1)
```

[116...
```
MLP_Model_1 = MLPClassifier(hidden_layer_sizes=(17,3), max_iter = 1550)
```

MLP_Model_1 is the Multi-layer Perceptron for scenario 1.

The training data is used below to teach this model how to predict the response variable.

[119...
```
MLP_Model_1.fit(x_train1,y_train1)
```

[119...
```
MLPClassifier(hidden_layer_sizes=(17, 3), max_iter=1550)
```

Next, the testing data (x_test1) will be given to the model. The model will attempt to predict the values for y_test1.

[122...
```
X_Test_Prediction1 = MLP_Model_1.predict(x_test1)
```

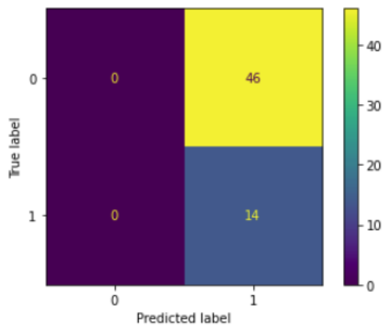
The variable y_test1 represents the target values and X_Test_Prediction1 represents the model predictions.

https://scikit-learn.org/dev/modules/generated/sklearn.metrics.confusion_matrix.html

[125...
```
Confusion_Matrix1 = metrics.confusion_matrix(y_test1, X_Test_Prediction1)
Confusion_Matrix1
```

[125...
```
array([[ 0, 46],
       [ 0, 14]], dtype=int64)
```

[127...
```
CM_Display_MLP_1 = metrics.ConfusionMatrixDisplay(Confusion_Matrix1)
CM_Display_MLP_1.plot()
```

[127...
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361b8deb80>
```


[129...
```
Accuracy1 = metrics.accuracy_score(y_test1, X_Test_Prediction1)
Accuracy1
```
[129...
```
0.23333333333333334
```

The above confusion matrix shows that when the actual value of Death is 0, this model predicts the value one 46 times. Out of 60 data points, 46 are errors. When the actual value of Death is 1, this model predicts the value one 14 times. This model is accurate 14 times out of 60 data points.

The variable Accuracy1 represents the accuracy for scenario 1 with the MLP model.

[133...
```
from sklearn.metrics import r2_score
```
[135...
```
R_Square_1 = r2_score(y_test1, X_Test_Prediction1)
```
[137...
```
print("The R-Squared value for scenario 1 is", R_Square_1)
```
```
The R-Squared value for scenario 1 is -3.2857142857142865
```

Next, the adjusted R square value will be calculated.

https://www.statology.org/adjusted-r-squared-in-python/

https://www.datacamp.com/tutorial/adjusted-r-squared

https://www.ibm.com/docs/en/cognos-analytics/11.1.0?topic=terms-adjusted-r-squared

[140...
```
Adjusted_R_Square_1 = 1 - ((1 - R_Square_1)*(len(y_train1) - 1)/(len(y_train1) - x_train1.shape[1] - 1))
```
[142...
```
print("The Adjusted R-Squared value for scenario 1 is", Adjusted_R_Square_1)
```
```
The Adjusted R-Squared value for scenario 1 is -3.513274336283187
```
[144...
```
from sklearn.metrics import mean_squared_error
import numpy as np
```

Next, the MSE and RMSE values will be calculated.

https://www.geeksforgeeks.org/step-by-step-guide-to-calculating-rmse-using-scikit-learn/#

https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.mean_squared_error.html

[147...
```
MSE_MLP_1 = mean_squared_error(y_test1, X_Test_Prediction1)
```
[149...
```
print("The Mean Square Error (MSE) value for scenario 1 with the MLP model is", MSE_MLP_1)
```
```
The Mean Square Error (MSE) value for scenario 1 with the MLP model is 0.7666666666666667
```
[151...
```
RMSE_MLP_1 = np.sqrt(MSE_MLP_1)
```
[153...
```
print("The Root Mean Square Error (RMSE) value for scenario 1 with the MLP model is", RMSE_MLP_1)
```
```
The Root Mean Square Error (RMSE) value for scenario 1 with the MLP model is 0.8755950357709131
```

Scenario 2 (MLP)

```
[156...  y_train2 = np.ravel(y_train2)
```

```
[158...  MLP_Model_2 = MLPClassifier(hidden_layer_sizes=(17,3), max_iter = 1550)
```

```
[160...  MLP_Model_2.fit(x_train2,y_train2)
```

```
[160...  MLPClassifier(hidden_layer_sizes=(17, 3), max_iter=1550)
```

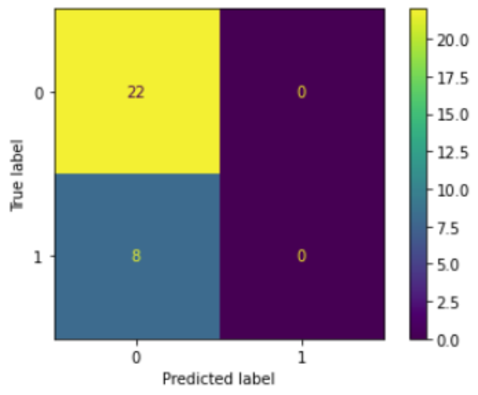
```
[162...  X_Test_Prediction2 = MLP_Model_2.predict(x_test2)
```

```
[164...  Confusion_Matrix2 = metrics.confusion_matrix(y_test2, X_Test_Prediction2)
         Confusion_Matrix2
```

```
[164...  array([[22,  0],
                [ 8,  0]], dtype=int64)
```

```
[166...  CM_Display_MLP_2 = metrics.ConfusionMatrixDisplay(Confusion_Matrix2)
         CM_Display_MLP_2.plot()
```

```
[166...  <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361c971370>
```



```
[168...  Accuracy2 = metrics.accuracy_score(y_test2, X_Test_Prediction2)
         Accuracy2
```

```
[168...  0.7333333333333333
```

[170...
```
R_Square_2 = r2_score(y_test2, X_Test_Prediction2)
```

[172...
```
print("The R-Squared value for scenario 2 is", R_Square_2)
```
```
The R-Squared value for scenario 2 is -0.363636363636364
```

[174...
```
Adjusted_R_Square_2 = 1 - ((1 - R_Square_2)*(len(y_train2) - 1)/(len(y_train2) - x_train2.shape[1] - 1))
```

[176...
```
print("The Adjusted R-Squared value for scenario 2 is", Adjusted_R_Square_2)
```
```
The Adjusted R-Squared value for scenario 2 is -0.4275568181818l857
```

[178...
```
MSE_MLP_2 = mean_squared_error(y_test2, X_Test_Prediction2)
```

[180...
```
print("The Mean Square Error (MSE) value for scenario 2 with the MLP model is", MSE_MLP_2)
```
```
The Mean Square Error (MSE) value for scenario 2 with the MLP model is 0.26666666666666666
```

[182...
```
RMSE_MLP_2 = np.sqrt(MSE_MLP_2)
```

[184...
```
print("The Root Mean Square Error (RMSE) value for scenario 2 with the MLP model is", RMSE_MLP_2)
```
```
The Root Mean Square Error (RMSE) value for scenario 2 with the MLP model is 0.5163977794943222
```

Scenario 3 (MLP)

[187...
```
y_train3 = np.ravel(y_train3)
```

[189...
```
MLP_Model_3 = MLPClassifier(hidden_layer_sizes=(17,3), max_iter = 1550)
```

[191...
```
MLP_Model_3.fit(x_train3,y_train3)
```

[191...
```
MLPClassifier(hidden_layer_sizes=(17, 3), max_iter=1550)
```

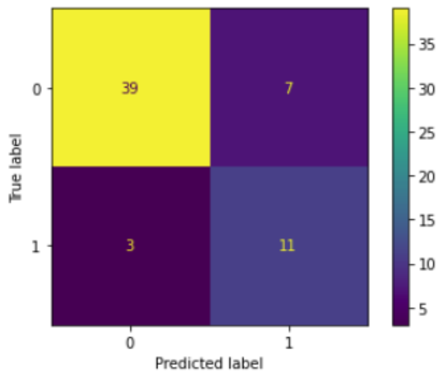
[193...
```
X_Test_Prediction3 = MLP_Model_3.predict(x_test3)
```

[195...
```
Confusion_Matrix3 = metrics.confusion_matrix(y_test3, X_Test_Prediction3)
Confusion_Matrix3
```

[195...
```
array([[39,  7],
       [ 3, 11]], dtype=int64)
```

[197...
```
CM_Display_MLP_3 = metrics.ConfusionMatrixDisplay(Confusion_Matrix3)
CM_Display_MLP_3.plot()
```

[197...
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361cb4dfd0>
```



[199...
```
Accuracy3 = metrics.accuracy_score(y_test3, X_Test_Prediction3)
Accuracy3
```

[199...
```
0.8333333333333334
```

[201...
```
R_Square_3 = r2_score(y_test3, X_Test_Prediction3)
```

[203...
```
print("The R-Squared value for scenario 3 is", R_Square_3)
```

```
The R-Squared value for scenario 3 is 0.06832298136645942
```

[205...
```
Adjusted_R_Square_3 = 1 - ((1 - R_Square_3)*(len(y_train3) - 1)/(len(y_train3) - x_train3.shape[1] - 1))
```

[207...
```
print("The Adjusted R-Squared value for scenario 3 is", Adjusted_R_Square_3)
```

```
The Adjusted R-Squared value for scenario 3 is 0.018853405155829006
```

[209...
```
MSE_MLP_3 = mean_squared_error(y_test3, X_Test_Prediction3)
```

[211...
```
print("The Mean Square Error (MSE) value for scenario 3 with the MLP model is", MSE_MLP_3)
```

```
The Mean Square Error (MSE) value for scenario 3 with the MLP model is 0.16666666666666666
```

[213...
```
RMSE_MLP_3 = np.sqrt(MSE_MLP_3)
```

[215...
```
print("The Root Mean Square Error (RMSE) value for scenario 3 with the MLP model is", RMSE_MLP_3)
```

```
The Root Mean Square Error (RMSE) value for scenario 3 with the MLP model is 0.408248290463863
```

Scenario 4 (MLP)

[218...
```
y_train4 = np.ravel(y_train4)
```

[220...
```
MLP_Model_4 = MLPClassifier(hidden_layer_sizes=(16,3), max_iter = 1500)
```

[222...
```
MLP_Model_4.fit(x_train4,y_train4)
```

[222...
```
MLPClassifier(hidden_layer_sizes=(16, 3), max_iter=1500)
```

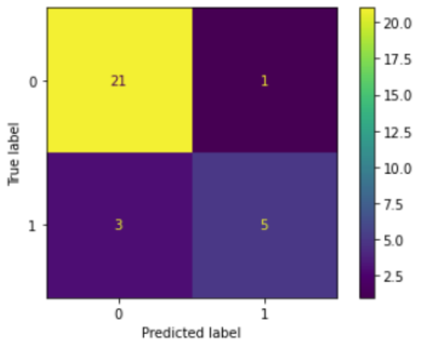
[223...
```
X_Test_Prediction4 = MLP_Model_4.predict(x_test4)
```

[224...
```
Confusion_Matrix4 = metrics.confusion_matrix(y_test4, X_Test_Prediction4)
Confusion_Matrix4
```

[224...
```
array([[21,  1],
       [ 3,  5]], dtype=int64)
```

[228...
```
CM_Display_MLP_4 = metrics.ConfusionMatrixDisplay(Confusion_Matrix4)
CM_Display_MLP_4.plot()
```

[228...
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361bb38c10>
```

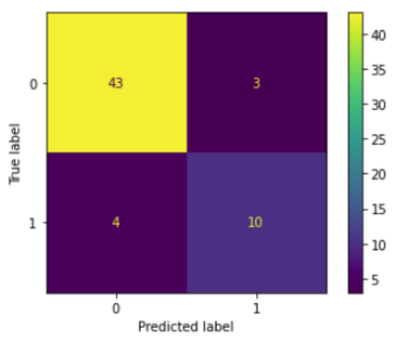


[230...
```
Accuracy4 = metrics.accuracy_score(y_test4, X_Test_Prediction4)
Accuracy4
```

[230...
```
0.8666666666666667
```

[232...
```
R_Square_4 = r2_score(y_test4, X_Test_Prediction4)
```
[234...
```
print("The R-Squared value for scenario 4 is", R_Square_4)
```
```
The R-Squared value for scenario 4 is 0.318181818181818
```
[236...
```
Adjusted_R_Square_4 = 1 - ((1 - R_Square_4)*(len(y_train4) - 1)/(len(y_train4) - x_train4.shape[1] - 1))
```
[238...
```
print("The Adjusted R-Squared value for scenario 4 is", Adjusted_R_Square_4)
```
```
The Adjusted R-Squared value for scenario 4 is 0.2862215909090907
```
[240...
```
MSE_MLP_4 = mean_squared_error(y_test4, X_Test_Prediction4)
```
[242...
```
print("The Mean Square Error (MSE) value for scenario 4 with the MLP model is", MSE_MLP_4)
```
```
The Mean Square Error (MSE) value for scenario 4 with the MLP model is 0.13333333333333333
```
[244...
```
RMSE_MLP_4 = np.sqrt(MSE_MLP_4)
```
[246...
```
print("The Root Mean Square Error (RMSE) value for scenario 4 with the MLP model is", RMSE_MLP_4)
```
```
The Root Mean Square Error (RMSE) value for scenario 4 with the MLP model is 0.3651483716701107
```

Scenario 5 (MLP)

[249...
```
y_train5 = np.ravel(y_train5)
```
[251...
```
MLP_Model_5 = MLPClassifier(hidden_layer_sizes=(17,3), max_iter = 1550)
```
[253...
```
MLP_Model_5.fit(x_train5,y_train5)
```
[253...
```
MLPClassifier(hidden_layer_sizes=(17, 3), max_iter=1550)
```
[255...
```
X_Test_Prediction5 = MLP_Model_5.predict(x_test5)
```
[257...
```
Confusion_Matrix5 = metrics.confusion_matrix(y_test5, X_Test_Prediction5)
Confusion_Matrix5
```
[257...
```
array([[43,  3],
       [ 4, 10]], dtype=int64)
```
[259...
```
CM_Display_MLP_5 = metrics.ConfusionMatrixDisplay(Confusion_Matrix5)
CM_Display_MLP_5.plot()
```
[259...
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361bbab400>
```


[261...
```
Accuracy5 = metrics.accuracy_score(y_test5, X_Test_Prediction5)
Accuracy5
```
[261...
```
0.8833333333333333
```

[263...
```
R_Square_5 = r2_score(y_test5, X_Test_Prediction5)
```

[265...
```
print("The R-Squared value for scenario 5 is", R_Square_5)
```
```
The R-Squared value for scenario 5 is 0.3478260869565216
```

[267...
```
Adjusted_R_Square_5 = 1 - ((1 - R_Square_5)*(len(y_train5) - 1)/(len(y_train5) - x_train5.shape[1] - 1))
```

[269...
```
print("The Adjusted R-Squared value for scenario 5 is", Adjusted_R_Square_5)
```
```
The Adjusted R-Squared value for scenario 5 is 0.3338309386079492
```

[271...
```
MSE_MLP_5 = mean_squared_error(y_test5, X_Test_Prediction5)
```

[273...
```
print("The Mean Square Error (MSE) value for scenario 5 with the MLP model is", MSE_MLP_5)
```
```
The Mean Square Error (MSE) value for scenario 5 with the MLP model is 0.11666666666666667
```

[275...
```
RMSE_MLP_5 = np.sqrt(MSE_MLP_5)
```

[277...
```
print("The Root Mean Square Error (RMSE) value for scenario 5 with the MLP model is", RMSE_MLP_5)
```
```
The Root Mean Square Error (RMSE) value for scenario 5 with the MLP model is 0.3415650255319866
```

Scenario 6 (MLP)

[280...
```
y_train6 = np.ravel(y_train6)
```

[282...
```
MLP_Model_6 = MLPClassifier(hidden_layer_sizes=(17,3), max_iter = 1550)
```

[284...
```
MLP_Model_6.fit(x_train6,y_train6)
```

[284...
```
MLPClassifier(hidden_layer_sizes=(17, 3), max_iter=1550)
```

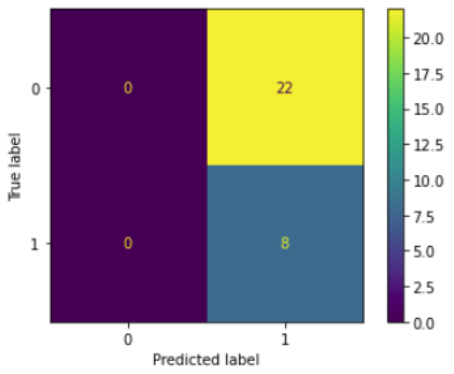
[286...
```
X_Test_Prediction6 = MLP_Model_6.predict(x_test6)
```

[288...
```
Confusion_Matrix6 = metrics.confusion_matrix(y_test6, X_Test_Prediction6)
Confusion_Matrix6
```

[288...
```
array([[ 0, 22],
       [ 0,  8]], dtype=int64)
```

[290...
```
CM_Display_MLP_6 = metrics.ConfusionMatrixDisplay(Confusion_Matrix6)
CM_Display_MLP_6.plot()
```

[290...
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361dbdb070>
```

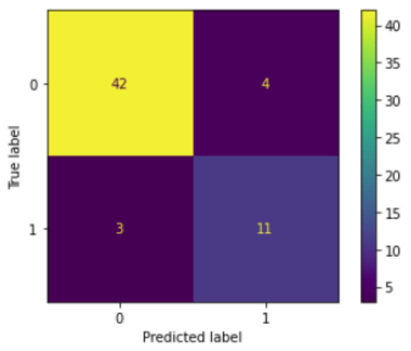


[292...
```
Accuracy6 = metrics.accuracy_score(y_test6, X_Test_Prediction6)
Accuracy6
```

[292...
```
0.26666666666666666
```

[294...
```
R_Square_6 = r2_score(y_test6, X_Test_Prediction6)
```
[296...
```
print("The R-Squared value for scenario 6 is", R_Square_6)
```
```
The R-Squared value for scenario 6 is -2.750000000000001
```
[298...
```
Adjusted_R_Square_6 = 1 - ((1 - R_Square_6)*(len(y_train6) - 1)/(len(y_train6) - x_train6.shape[1] - 1))
```
[300...
```
print("The Adjusted R-Squared value for scenario 6 is", Adjusted_R_Square_6)
```
```
The Adjusted R-Squared value for scenario 6 is -2.8212927756654
```
[302...
```
MSE_MLP_6 = mean_squared_error(y_test6, X_Test_Prediction6)
```
[304...
```
print("The Mean Square Error (MSE) value for scenario 6 with the MLP model is", MSE_MLP_6)
```
```
The Mean Square Error (MSE) value for scenario 6 with the MLP model is 0.7333333333333333
```
[306...
```
RMSE_MLP_6 = np.sqrt(MSE_MLP_6)
```
[308...
```
print("The Root Mean Square Error (RMSE) value for scenario 6 with the MLP model is", RMSE_MLP_6)
```
```
The Root Mean Square Error (RMSE) value for scenario 6 with the MLP model is 0.8563488385776752
```

Scenario 7 (MLP)

[311...
```
y_train7 = np.ravel(y_train7)
```
[313...
```
MLP_Model_7 = MLPClassifier(hidden_layer_sizes=(17,3), max_iter = 1550)
```
[315...
```
MLP_Model_7.fit(x_train7,y_train7)
```
[315...
```
MLPClassifier(hidden_layer_sizes=(17, 3), max_iter=1550)
```
[316...
```
X_Test_Prediction7 = MLP_Model_7.predict(x_test7)
```
[317...
```
Confusion_Matrix7 = metrics.confusion_matrix(y_test7, X_Test_Prediction7)
Confusion_Matrix7
```
[317...
```
array([[42,  4],
       [ 3, 11]], dtype=int64)
```
[321...
```
CM_Display_MLP_7 = metrics.ConfusionMatrixDisplay(Confusion_Matrix7)
CM_Display_MLP_7.plot()
```
[321...
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361dcfb3d0>
```


[323...
```
Accuracy7 = metrics.accuracy_score(y_test7, X_Test_Prediction7)
Accuracy7
```
[323...
```
0.8833333333333333
```

```
[325... R_Square_7 = r2_score(y_test7, X_Test_Prediction7)
```

```
[327... print("The R-Squared value for scenario 7 is", R_Square_7)
```

```
The R-Squared value for scenario 7 is 0.3478260869565216
```

```
[329... Adjusted_R_Square_7 = 1 - ((1 - R_Square_7)*(len(y_train7) - 1)/(len(y_train7) - x_train7.shape[1] - 1))
```

```
[331... print("The Adjusted R-Squared value for scenario 7 is", Adjusted_R_Square_7)
```

```
The Adjusted R-Squared value for scenario 7 is 0.3338309386079492
```

```
[333... MSE_MLP_7 = mean_squared_error(y_test7, X_Test_Prediction7)
```

```
[335... print("The Mean Square Error (MSE) value for scenario 7 with the MLP model is", MSE_MLP_7)
```

```
The Mean Square Error (MSE) value for scenario 7 with the MLP model is 0.11666666666666667
```

```
[337... RMSE_MLP_7 = np.sqrt(MSE_MLP_7)
```

```
[339... print("The Root Mean Square Error (RMSE) value for scenario 7 with the MLP model is", RMSE_MLP_7)
```

```
The Root Mean Square Error (RMSE) value for scenario 7 with the MLP model is 0.3415650255319866
```

Scenario 8

```
[342... y_train8 = np.ravel(y_train8)
```

```
[366... MLP_Model_8 = MLPClassifier(hidden_layer_sizes=(17,3), max_iter = 1700)
```

```
[368... MLP_Model_8.fit(x_train8,y_train8)
```

```
[368... MLPClassifier(hidden_layer_sizes=(17, 3), max_iter=1700)
```

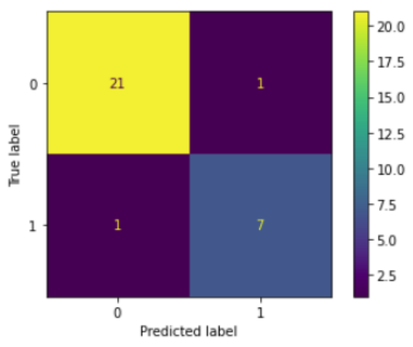
```
[370... X_Test_Prediction8 = MLP_Model_8.predict(x_test8)
```

```
[372... Confusion_Matrix8 = metrics.confusion_matrix(y_test8, X_Test_Prediction8)
Confusion_Matrix8
```

```
[372... array([[21,  1],
        [ 1,  7]], dtype=int64)
```

```
[374... CM_Display_MLP_8 = metrics.ConfusionMatrixDisplay(Confusion_Matrix8)
CM_Display_MLP_8.plot()
```

```
[374... <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361c9fadf0>
```



```
[376... Accuracy8 = metrics.accuracy_score(y_test8, X_Test_Prediction8)
Accuracy8
```

```
[376... 0.9333333333333333
```

[378...
```
R_Square_8 = r2_score(y_test8, X_Test_Prediction8)
```

[380...
```
print("The R-Squared value for scenario 8 is", R_Square_8)
```
```
The R-Squared value for scenario 8 is 0.659090909090909
```

[382...
```
Adjusted_R_Square_8 = 1 - ((1 - R_Square_8)*(len(y_train8) - 1)/(len(y_train8) - x_train8.shape[1] - 1))
```

[384...
```
print("The Adjusted R-Squared value for scenario 8 is", Adjusted_R_Square_8)
```
```
The Adjusted R-Squared value for scenario 8 is 0.6526097476667818
```

[386...
```
MSE_MLP_8 = mean_squared_error(y_test8, X_Test_Prediction8)
```

[388...
```
print("The Mean Square Error (MSE) value for scenario 8 with the MLP model is", MSE_MLP_8)
```
```
The Mean Square Error (MSE) value for scenario 8 with the MLP model is 0.06666666666666667
```

[390...
```
RMSE_MLP_8 = np.sqrt(MSE_MLP_8)
```

[392...
```
print("The Root Mean Square Error (RMSE) value for scenario 8 with the MLP model is", RMSE_MLP_8)
```
```
The Root Mean Square Error (RMSE) value for scenario 8 with the MLP model is 0.2581988897471611
```

The below 8 scenarios are using a Decision Tree (DT) model

Scenario 1

[892...
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn import tree
```

[894...
```
DecisionTreeModel_1 = DecisionTreeClassifier(random_state = 1, max_depth = 9, criterion = 'entropy')
DecisionTreeModel_1.fit(x_train1,y_train1)
```

[894...
```
DecisionTreeClassifier(criterion='entropy', max_depth=9, random_state=1)
```

[896...
```
X_Test_Prediction_DT_1 = DecisionTreeModel_1.predict(x_test1)
X_Test_Prediction_DT_1
```

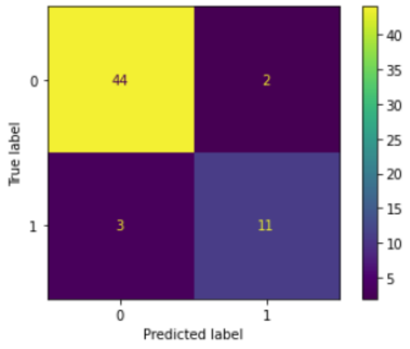
[896...
```
array([0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0], dtype=int64)
```

[898...
```
Confusion_Matrix_DT_1 = metrics.confusion_matrix(y_test1, X_Test_Prediction_DT_1)
Confusion_Matrix_DT_1
```

[898...
```
array([[44,  2],
       [ 3, 11]], dtype=int64)
```

```
[900... CM_Display_DT_1 = metrics.ConfusionMatrixDisplay(Confusion_Matrix_DT_1)
       CM_Display_DT_1.plot()
```

```
[900... <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361ffd9f10>
```



```
[902... Accuracy_DT_1 = metrics.accuracy_score(y_test1, X_Test_Prediction_DT_1)
       Accuracy_DT_1
```

```
[902... 0.9166666666666666
```

```
[904... R_Square_DT_1 = r2_score(y_test1, X_Test_Prediction_DT_1)
```

```
[906... print("The R-Squared value for scenario 1 is", R_Square_DT_1)
       The R-Squared value for scenario 1 is 0.5341614906832297
```

```
[908... Adjusted_R_Square_DT_1 = 1 - ((1 - R_Square_DT_1)*(len(y_train1) - 1)/(len(y_train1) - x_train1.shape[1] - 1))
```

```
[910... print("The Adjusted R-Squared value for scenario 1 is", Adjusted_R_Square_DT_1)
       The Adjusted R-Squared value for scenario 1 is 0.5094267025779144
```

```
[912... MSE_DT_1 = mean_squared_error(y_test1, X_Test_Prediction1)
```

```
[914... print("The Mean Square Error (MSE) value for scenario 1 with the DT model is", MSE_DT_1)
       The Mean Square Error (MSE) value for scenario 1 with the DT model is 0.7666666666666667
```

```
[916... RMSE_DT_1 = np.sqrt(MSE_DT_1)
```

```
[918... print("The Root Mean Square Error (RMSE) value for scenario 1 with the DT model is", RMSE_DT_1)
       The Root Mean Square Error (RMSE) value for scenario 1 with the DT model is 0.8755950357709131
```

The below code will export Decision Tree Scenario 1 into a png file that has been submitted with this project. The name of that file is Decision_Tree_Scenario_1.

```
[921... from sklearn.tree import export_graphviz
```

```
[923... Decision_Tree_Image = tree.export_graphviz(DecisionTreeModel_1,filled=True)
```

```
[925... DT_Image = graphviz.Source(Decision_Tree_Image)
```

```
[927... DT_Image.format = "png"
       DT_Image.render("Decision_Tree_Scenario_1")
```

```
[927... 'Decision_Tree_Scenario_1.png'
```

Scenario 2

```
[462... DecisionTreeModel_2 = DecisionTreeClassifier(random_state = 1, max_depth = 9, criterion = 'entropy')
       DecisionTreeModel_2.fit(x_train2,y_train2)
```

```
[462... DecisionTreeClassifier(criterion='entropy', max_depth=9, random_state=1)
```

```
[464... X_Test_Prediction_DT_2 = DecisionTreeModel_2.predict(x_test2)
       X_Test_Prediction_DT_2
```

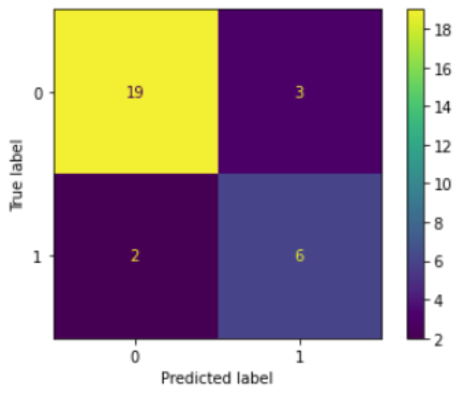
```
[464... array([0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,
              1, 1, 0, 0, 0, 0, 1, 0], dtype=int64)
```

```
[466... Confusion_Matrix_DT_2 = metrics.confusion_matrix(y_test2, X_Test_Prediction_DT_2)
       Confusion_Matrix_DT_2
```

```
[466... array([[19,  3],
              [ 2,  6]], dtype=int64)
```

```
[468... CM_Display_DT_2 = metrics.ConfusionMatrixDisplay(Confusion_Matrix_DT_2)
       CM_Display_DT_2.plot()
```

```
[468... <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361e2f1e20>
```



```
[470... Accuracy_DT_2 = metrics.accuracy_score(y_test2, X_Test_Prediction_DT_2)
       Accuracy_DT_2
```

```
[470... 0.8333333333333334
```

```
[472... R_Square_DT_2 = r2_score(y_test2, X_Test_Prediction_DT_2)
```

```
[474... print("The R-Squared value for scenario 2 is", R_Square_DT_2)
       The R-Squared value for scenario 2 is 0.14772727272727249
```

```
[476... Adjusted_R_Square_DT_2 = 1 - ((1 - R_Square_DT_2)*(len(y_train2) - 1)/(len(y_train2) - x_train2.shape[1] - 1))
```

```
[478... print("The Adjusted R-Squared value for scenario 2 is", Adjusted_R_Square_DT_2)
       The Adjusted R-Squared value for scenario 2 is 0.10777698863636342
```

```
[480... MSE_DT_2 = mean_squared_error(y_test2, X_Test_Prediction_DT_2)
```

```
[482... print("The Mean Square Error (MSE) value for scenario 2 with the DT model is", MSE_DT_2)
       The Mean Square Error (MSE) value for scenario 2 with the DT model is 0.16666666666666666
```

```
[484... RMSE_DT_2 = np.sqrt(MSE_DT_2)
```

```
[486... print("The Root Mean Square Error (RMSE) value for scenario 2 with the DT model is", RMSE_DT_2)
       The Root Mean Square Error (RMSE) value for scenario 2 with the DT model is 0.408248290463863
```

Scenario 3

```
[493... DecisionTreeModel_3 = DecisionTreeClassifier(random_state = 1, max_depth = 9, criterion = 'entropy')
       DecisionTreeModel_3.fit(x_train3,y_train3)
```

```
[493... DecisionTreeClassifier(criterion='entropy', max_depth=9, random_state=1)
```

```
[495... X_Test_Prediction_DT_3 = DecisionTreeModel_3.predict(x_test3)
       X_Test_Prediction_DT_3
```

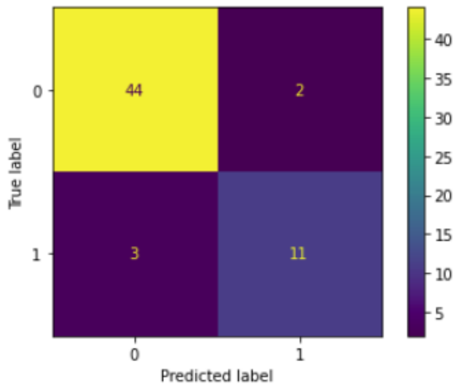
```
[495... array([0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,
              1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
              0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0], dtype=int64)
```

```
[497... Confusion_Matrix_DT_3 = metrics.confusion_matrix(y_test3, X_Test_Prediction_DT_3)
       Confusion_Matrix_DT_3
```

```
[497... array([[44,  2],
              [ 3, 11]], dtype=int64)
```

```
[499... CM_Display_DT_3 = metrics.ConfusionMatrixDisplay(Confusion_Matrix_DT_3)
       CM_Display_DT_3.plot()
```

```
[499... <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361de2c5e0>
```



```
[501... Accuracy_DT_3 = metrics.accuracy_score(y_test3, X_Test_Prediction_DT_3)
       Accuracy_DT_3
```

```
[501... 0.9166666666666666
```

```
[503... R_Square_DT_3 = r2_score(y_test3, X_Test_Prediction_DT_3)
```

```
[505... print("The R-Squared value for scenario 3 is", R_Square_DT_3)
       The R-Squared value for scenario 3 is 0.5341614906832297
```

```
[507... Adjusted_R_Square_DT_3 = 1 - ((1 - R_Square_DT_3)*(len(y_train3) - 1)/(len(y_train3) - x_train3.shape[1] - 1))
```

```
[509... print("The Adjusted R-Squared value for scenario 3 is", Adjusted_R_Square_DT_3)
       The Adjusted R-Squared value for scenario 3 is 0.5094267025779144
```

```
[511... MSE_DT_3 = mean_squared_error(y_test3, X_Test_Prediction_DT_3)
```

```
[513... print("The Mean Square Error (MSE) value for scenario 3 with the DT model is", MSE_DT_3)
       The Mean Square Error (MSE) value for scenario 3 with the DT model is 0.08333333333333333
```

```
[515... RMSE_DT_3 = np.sqrt(MSE_DT_3)
```

```
[517... print("The Root Mean Square Error (RMSE) value for scenario 3 with the DT model is", RMSE_DT_3)
       The Root Mean Square Error (RMSE) value for scenario 3 with the DT model is 0.28867513459481287
```

Scenario 4

```
[520... DecisionTreeModel_4 = DecisionTreeClassifier(random_state = 1, max_depth = 9, criterion = 'entropy')
       DecisionTreeModel_4.fit(x_train4,y_train4)
```

```
[520... DecisionTreeClassifier(criterion='entropy', max_depth=9, random_state=1)
```

```
[522... X_Test_Prediction_DT_4 = DecisionTreeModel_4.predict(x_test4)
       X_Test_Prediction_DT_4
```

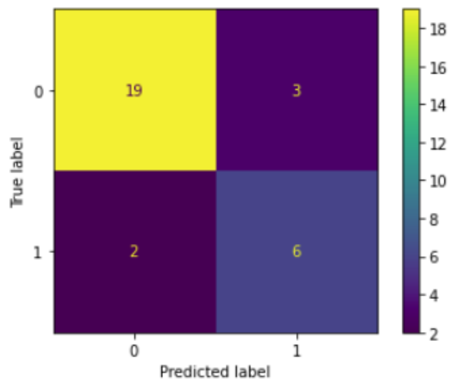
```
[522... array([0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,
              1, 1, 0, 0, 0, 0, 1, 0], dtype=int64)
```

```
[524... Confusion_Matrix_DT_4 = metrics.confusion_matrix(y_test4, X_Test_Prediction_DT_4)
       Confusion_Matrix_DT_4
```

```
[524... array([[19,  3],
              [ 2,  6]], dtype=int64)
```

```
[526... CM_Display_DT_4 = metrics.ConfusionMatrixDisplay(Confusion_Matrix_DT_4)
       CM_Display_DT_4.plot()
```

```
[526... <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361bbab9d0>
```



```
[528... Accuracy_DT_4 = metrics.accuracy_score(y_test4, X_Test_Prediction_DT_4)
       Accuracy_DT_4
```

```
[528... 0.8333333333333334
```

```
[530... R_Square_DT_4 = r2_score(y_test4, X_Test_Prediction_DT_4)
```

```
[532... print("The R-Squared value for scenario 8 is", R_Square_DT_4)
       The R-Squared value for scenario 8 is 0.14772727272727249
```

```
[534... Adjusted_R_Square_DT_4 = 1 - ((1 - R_Square_DT_4)*(len(y_train4) - 1)/(len(y_train4) - x_train4.shape[1] - 1))
```

```
[536... print("The Adjusted R-Squared value for scenario 4 is", Adjusted_R_Square_DT_4)
       The Adjusted R-Squared value for scenario 4 is 0.10777698863636342
```

```
[538... MSE_DT_4 = mean_squared_error(y_test4, X_Test_Prediction_DT_4)
```

```
[540... print("The Mean Square Error (MSE) value for scenario 4 with the DT model is", MSE_DT_4)
       The Mean Square Error (MSE) value for scenario 4 with the DT model is 0.16666666666666666
```

```
[542... RMSE_DT_4 = np.sqrt(MSE_DT_4)
```

```
[544... print("The Root Mean Square Error (RMSE) value for scenario 4 with the DT model is", RMSE_DT_4)
       The Root Mean Square Error (RMSE) value for scenario 4 with the DT model is 0.408248290463863
```

Scenario 5

```
[547... DecisionTreeModel_5 = DecisionTreeClassifier(random_state = 1, max_depth = 9, criterion = 'entropy')
       DecisionTreeModel_5.fit(x_train5,y_train5)
```

```
[547... DecisionTreeClassifier(criterion='entropy', max_depth=9, random_state=1)
```

```
[549... X_Test_Prediction_DT_5 = DecisionTreeModel_5.predict(x_test5)
       X_Test_Prediction_DT_5
```

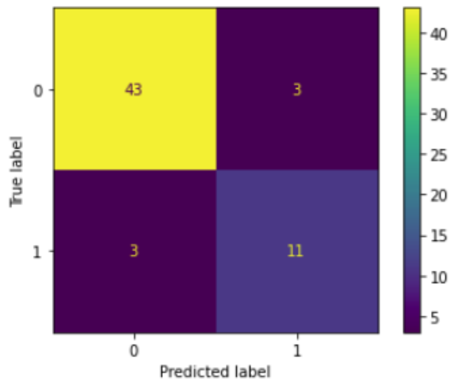
```
[549... array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1,
              1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
              0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0], dtype=int64)
```

```
[551... Confusion_Matrix_DT_5 = metrics.confusion_matrix(y_test5, X_Test_Prediction_DT_5)
       Confusion_Matrix_DT_5
```

```
[551... array([[43,  3],
              [ 3, 11]], dtype=int64)
```

```
[553... CM_Display_DT_5 = metrics.ConfusionMatrixDisplay(Confusion_Matrix_DT_5)
       CM_Display_DT_5.plot()
```

```
[553... <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361e47fa60>
```



```
[555... Accuracy_DT_5 = metrics.accuracy_score(y_test5, X_Test_Prediction_DT_5)
       Accuracy_DT_5
```

```
[555... 0.9
```

```
[557... R_Square_DT_5 = r2_score(y_test5, X_Test_Prediction_DT_5)
```

```
[559... print("The R-Squared value for scenario 5 is", R_Square_DT_5)

       The R-Squared value for scenario 5 is 0.4409937888198756
```

```
[561... Adjusted_R_Square_DT_5 = 1 - ((1 - R_Square_DT_5)*(len(y_train5) - 1)/(len(y_train5) - x_train5.shape[1] - 1))
```

```
[563... print("The Adjusted R-Squared value for scenario 5 is", Adjusted_R_Square_DT_5)

       The Adjusted R-Squared value for scenario 5 is 0.42899794737824215
```

```
[565... MSE_DT_5 = mean_squared_error(y_test5, X_Test_Prediction_DT_5)
```

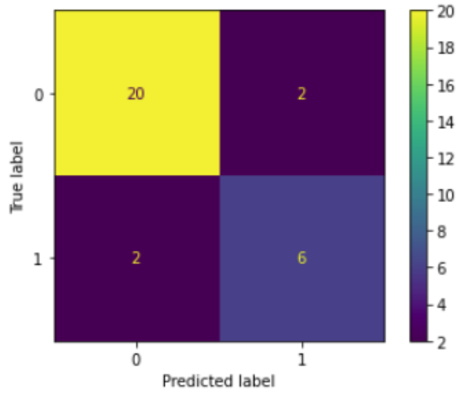
```
[567... print("The Mean Square Error (MSE) value for scenario 5 with the DT model is", MSE_DT_5)

       The Mean Square Error (MSE) value for scenario 5 with the DT model is 0.1
```

```
[569... RMSE_DT_5 = np.sqrt(MSE_DT_5)
```

```
[571... print("The Root Mean Square Error (RMSE) value for scenario 5 with the DT model is", RMSE_DT_5)

       The Root Mean Square Error (RMSE) value for scenario 5 with the DT model is 0.31622776601683794
```

Scenario 6

```
[574...  DecisionTreeModel_6 = DecisionTreeClassifier(random_state = 1, max_depth = 9, criterion = 'entropy')
         DecisionTreeModel_6.fit(x_train6,y_train6)

[574...  DecisionTreeClassifier(criterion='entropy', max_depth=9, random_state=1)

[576...  X_Test_Prediction_DT_6 = DecisionTreeModel_6.predict(x_test6)
         X_Test_Prediction_DT_6

[576...  array([0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1,
                1, 1, 0, 0, 0, 0, 0, 0], dtype=int64)

[578...  Confusion_Matrix_DT_6 = metrics.confusion_matrix(y_test6, X_Test_Prediction_DT_6)
         Confusion_Matrix_DT_6

[578...  array([[20,  2],
                [ 2,  6]], dtype=int64)

[580...  CM_Display_DT_6 = metrics.ConfusionMatrixDisplay(Confusion_Matrix_DT_6)
         CM_Display_DT_6.plot()

[580...  <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361e6f9430>
```



```
[582...  Accuracy_DT_6 = metrics.accuracy_score(y_test6, X_Test_Prediction_DT_6)
         Accuracy_DT_6

[582...  0.8666666666666667

[584...  R_Square_DT_6 = r2_score(y_test6, X_Test_Prediction_DT_6)

[586...  print("The R-Squared value for scenario 6 is", R_Square_DT_6)
         The R-Squared value for scenario 6 is 0.3181818181818181818

[588...  Adjusted_R_Square_DT_6 = 1 - ((1 - R_Square_DT_6)*(len(y_train6) - 1)/(len(y_train6) - x_train6.shape[1] - 1))

[590...  print("The Adjusted R-Squared value for scenario 6 is", Adjusted_R_Square_DT_6)
         The Adjusted R-Squared value for scenario 6 is 0.30521949533356363

[592...  MSE_DT_6 = mean_squared_error(y_test6, X_Test_Prediction_DT_6)

[594...  print("The Mean Square Error (MSE) value for scenario 6 with the DT model is", MSE_DT_6)
         The Mean Square Error (MSE) value for scenario 6 with the DT model is 0.13333333333333333

[596...  RMSE_DT_6 = np.sqrt(MSE_DT_6)

[598...  print("The Root Mean Square Error (RMSE) value for scenario 6 with the DT model is", RMSE_DT_6)
         The Root Mean Square Error (RMSE) value for scenario 6 with the DT model is 0.3651483716701107
```

Scenario 7

```
[603... DecisionTreeModel_7 = DecisionTreeClassifier(random_state = 1, max_depth = 9, criterion = 'entropy')
       DecisionTreeModel_7.fit(x_train7,y_train7)
```

```
[603... DecisionTreeClassifier(criterion='entropy', max_depth=9, random_state=1)
```

```
[605... X_Test_Prediction_DT_7 = DecisionTreeModel_7.predict(x_test7)
       X_Test_Prediction_DT_7
```

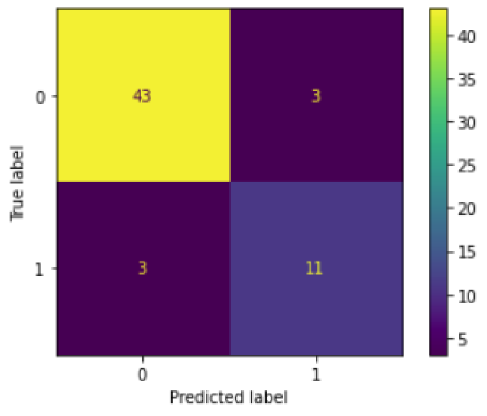
```
[605... array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1,
              1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
              0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0], dtype=int64)
```

```
[607... Confusion_Matrix_DT_7 = metrics.confusion_matrix(y_test7, X_Test_Prediction_DT_7)
       Confusion_Matrix_DT_7
```

```
[607... array([[43,  3],
              [ 3, 11]], dtype=int64)
```

```
[609... cm_display_DT_7 = metrics.ConfusionMatrixDisplay(Confusion_Matrix_DT_7)
       cm_display_DT_7.plot()
```

```
[609... <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361f815f40>
```



```
[611... Accuracy_DT_7 = metrics.accuracy_score(y_test7, X_Test_Prediction_DT_7)
       Accuracy_DT_7
```

```
[611... 0.9
```

```
[613... R_Square_DT_7 = r2_score(y_test7, X_Test_Prediction_DT_7)
```

```
[615... print("The R-Squared value for scenario 7 is", R_Square_DT_7)
       The R-Squared value for scenario 7 is 0.4409937888198756
```

```
[617... Adjusted_R_Square_DT_7 = 1 - ((1 - R_Square_DT_7)*(len(y_train7) - 1)/(len(y_train7) - x_train7.shape[1] - 1))
```

```
[619... print("The Adjusted R-Squared value for scenario 7 is", Adjusted_R_Square_DT_7)
       The Adjusted R-Squared value for scenario 7 is 0.42899794737824215
```

```
[621... MSE_DT_7 = mean_squared_error(y_test7, X_Test_Prediction_DT_7)
```

```
[623... print("The Mean Square Error (MSE) value for scenario 7 with the DT model is", MSE_DT_7)
       The Mean Square Error (MSE) value for scenario 7 with the DT model is 0.1
```

```
[625... RMSE_DT_7 = np.sqrt(MSE_DT_7)
```

```
[627... print("The Root Mean Square Error (RMSE) value for scenario 7 with the DT model is", RMSE_DT_7)
       The Root Mean Square Error (RMSE) value for scenario 7 with the DT model is 0.31622776601683794
```

Scenario 8

```
[630... DecisionTreeModel_8 = DecisionTreeClassifier(random_state = 1, max_depth = 9, criterion = 'entropy')
       DecisionTreeModel_8.fit(x_train8,y_train8)
```

```
[630... DecisionTreeClassifier(criterion='entropy', max_depth=9, random_state=1)
```

```
[632... X_Test_Prediction_DT_8 = DecisionTreeModel_8.predict(x_test8)
       X_Test_Prediction_DT_8
```

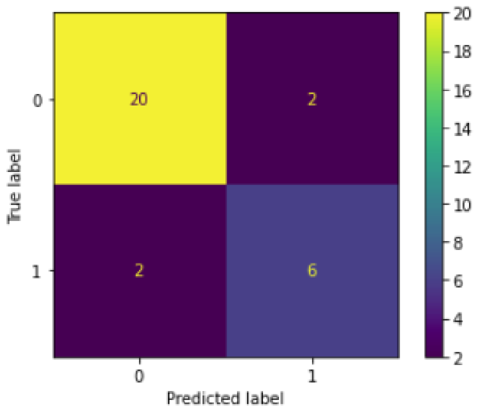
```
[632... array([0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1,
              1, 1, 0, 0, 0, 0, 0, 0], dtype=int64)
```

```
[634... Confusion_Matrix_DT_8 = metrics.confusion_matrix(y_test8, X_Test_Prediction_DT_8)
       Confusion_Matrix_DT_8
```

```
[634... array([[20,  2],
              [ 2,  6]], dtype=int64)
```

```
[636... cm_display_DT_8 = metrics.ConfusionMatrixDisplay(Confusion_Matrix_DT_8)
       cm_display_DT_8.plot()
```

```
[636... <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361f9adca0>
```



```
[638... Accuracy_DT_8 = metrics.accuracy_score(y_test8, X_Test_Prediction_DT_8)
       Accuracy_DT_8
```

```
[638... 0.8666666666666667
```

```
[640... R_Square_DT_8 = r2_score(y_test8, X_Test_Prediction_DT_8)
```

```
[642... print("The R-Squared value for scenario 8 is", R_Square_DT_8)
       The R-Squared value for scenario 8 is 0.31818181818181818
```

```
[644... Adjusted_R_Square_DT_8 = 1 - ((1 - R_Square_DT_8)*(len(y_train8) - 1)/(len(y_train8) - x_train8.shape[1] - 1))
```

```
[646... print("The Adjusted R-Squared value for scenario 8 is", Adjusted_R_Square_DT_8)
       The Adjusted R-Squared value for scenario 8 is 0.30521949533356363
```

```
[648... MSE_DT_8 = mean_squared_error(y_test8, X_Test_Prediction_DT_8)
```

```
[650... print("The Mean Square Error (MSE) value for scenario 8 with the DT model is", MSE_DT_8)
       The Mean Square Error (MSE) value for scenario 8 with the DT model is 0.13333333333333333
```

```
[652... RMSE_DT_8 = np.sqrt(MSE_DT_8)
```

```
[654... print("The Root Mean Square Error (RMSE) value for scenario 8 with the DT model is", RMSE_DT_8)
       The Root Mean Square Error (RMSE) value for scenario 8 with the DT model is 0.3651483716701107
```

The below 8 scenarios are using a Support Vector Machine (SVM) model

Scenario 1

[658...
```
from sklearn import svm
#https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python
#https://www.geeksforgeeks.org/visualizing-support-vector-machines-svm-using-python/
```

[660...
```
y_train1 = np.ravel(y_train1)
```

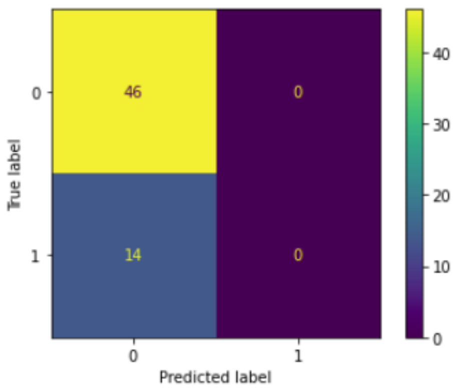
[662...
```
SVM_Model_1 = svm.SVC()
SVM_Model_1.fit(x_train1,y_train1)
```

[662...
```
SVC()
```

[664...
```
X_Test_Prediction_SVM_1 = SVM_Model_1.predict(x_test1)
```

[666...
```
Confusion_Matrix_SVM_1 = metrics.confusion_matrix(y_test1, X_Test_Prediction_SVM_1)
cm_display_SVM_1 = metrics.ConfusionMatrixDisplay(Confusion_Matrix_SVM_1)
cm_display_SVM_1.plot()
```

[666...
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361de2ac40>
```



[668...
```
Accuracy_SVM_1 = metrics.accuracy_score(y_test1, X_Test_Prediction_SVM_1)
Accuracy_SVM_1
```

[668...
```
0.7666666666666667
```

[670...
```
R_Square_SVM_1 = r2_score(y_test1, X_Test_Prediction_SVM_1)
```

[672...
```
print("The R-Squared value for scenario 1 is", R_Square_SVM_1)
```
```
The R-Squared value for scenario 1 is -0.30434782608695676
```

[674...
```
Adjusted_R_Square_SVM_1 = 1 - ((1 - R_Square_SVM_1)*(len(y_train1) - 1)/(len(y_train1) - x_train1.shape[1] - 1))
```

[676...
```
print("The Adjusted R-Squared value for scenario 1 is", Adjusted_R_Square_SVM_1)
```
```
The Adjusted R-Squared value for scenario 1 is -0.3736052327818393
```

[678...
```
MSE_SVM_1 = mean_squared_error(y_test1, X_Test_Prediction_DT_1)
```

[680...
```
print("The Mean Square Error (MSE) value for scenario 1 with the SVM model is", MSE_SVM_1)
```
```
The Mean Square Error (MSE) value for scenario 1 with the SVM model is 0.08333333333333333
```

[682...
```
RMSE_SVM_1 = np.sqrt(MSE_SVM_1)
```

[684...
```
print("The Root Mean Square Error (RMSE) value for scenario 1 with the SVM model is", RMSE_SVM_1)
```
```
The Root Mean Square Error (RMSE) value for scenario 1 with the SVM model is 0.28867513459481287
```

Scenario 2

[687...
```
y_train2 = np.ravel(y_train2)
```

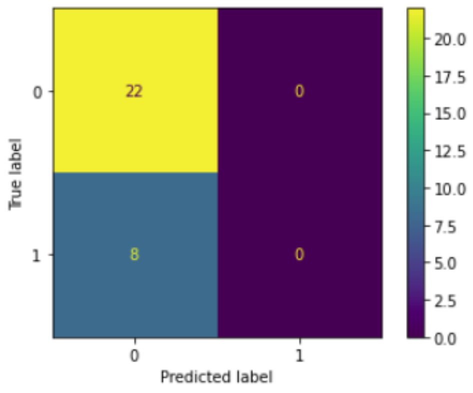
[689...
```
SVM_Model_2 = svm.SVC()
SVM_Model_2.fit(x_train2,y_train2)
```

[689...
```
SVC()
```

[691...
```
X_Test_Prediction_SVM_2 = SVM_Model_2.predict(x_test2)
```

[693...
```
Confusion_Matrix_SVM_2 = metrics.confusion_matrix(y_test2, X_Test_Prediction_SVM_2)
cm_display_SVM_2 = metrics.ConfusionMatrixDisplay(Confusion_Matrix_SVM_2)
cm_display_SVM_2.plot()
```

[693...
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361e2e27c0>
```



[695...
```
Accuracy_SVM_2 = metrics.accuracy_score(y_test2, X_Test_Prediction_SVM_2)
Accuracy_SVM_2
```

[695...
```
0.7333333333333333
```

[697...
```
R_Square_SVM_2 = r2_score(y_test2, X_Test_Prediction_SVM_2)
```

[699...
```
print("The R-Squared value for scenario 2 is", R_Square_SVM_2)
```
```
The R-Squared value for scenario 2 is -0.363636363636364
```

[701...
```
Adjusted_R_Square_SVM_2 = 1 - ((1 - R_Square_SVM_2)*(len(y_train2) - 1)/(len(y_train2) - x_train2.shape[1] - 1))
```

[703...
```
print("The Adjusted R-Squared value for scenario 2 is", Adjusted_R_Square_SVM_2)
```
```
The Adjusted R-Squared value for scenario 2 is -0.42755681818181857
```

[705...
```
MSE_SVM_2 = mean_squared_error(y_test2, X_Test_Prediction_DT_2)
```

[707...
```
print("The Mean Square Error (MSE) value for scenario 2 with the SVM model is", MSE_SVM_2)
```
```
The Mean Square Error (MSE) value for scenario 2 with the SVM model is 0.16666666666666666
```

[709...
```
RMSE_SVM_2 = np.sqrt(MSE_SVM_2)
```

[711...
```
print("The Root Mean Square Error (RMSE) value for scenario 2 with the SVM model is", RMSE_SVM_2)
```
```
The Root Mean Square Error (RMSE) value for scenario 2 with the SVM model is 0.408248290463863
```

Scenario 3

```
[714... y_train3 = np.ravel(y_train3)
```

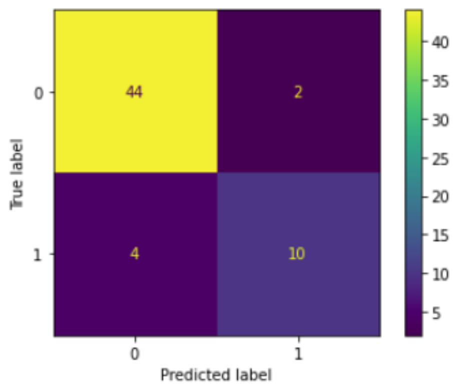
```
[716... SVM_Model_3 = svm.SVC()
       SVM_Model_3.fit(x_train3,y_train3)
```

```
[716... SVC()
```

```
[718... X_Test_Prediction_SVM_3 = SVM_Model_3.predict(x_test3)
```

```
[720... Confusion_Matrix_SVM_3 = metrics.confusion_matrix(y_test3, X_Test_Prediction_SVM_3)
       cm_display_SVM_3 = metrics.ConfusionMatrixDisplay(Confusion_Matrix_SVM_3)
       cm_display_SVM_3.plot()
```

```
[720... <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361e728d90>
```



```
[722... Accuracy_SVM_3 = metrics.accuracy_score(y_test3, X_Test_Prediction_SVM_3)
       Accuracy_SVM_3
```

```
[722... 0.9
```

```
[724... R_Square_SVM_3 = r2_score(y_test3, X_Test_Prediction_SVM_3)
```

```
[726... print("The R-Squared value for scenario 3 is", R_Square_SVM_3)
       The R-Squared value for scenario 3 is 0.4409937888198756
```

```
[728... Adjusted_R_Square_SVM_3 = 1 - ((1 - R_Square_SVM_3)*(len(y_train3) - 1)/(len(y_train3) - x_train3.shape[1] - 1))
```

```
[730... print("The Adjusted R-Squared value for scenario 3 is", Adjusted_R_Square_SVM_3)
       The Adjusted R-Squared value for scenario 3 is 0.41131204309349734
```

```
[732... MSE_SVM_3 = mean_squared_error(y_test3, X_Test_Prediction_DT_3)
```

```
[734... print("The Mean Square Error (MSE) value for scenario 3 with the SVM model is", MSE_SVM_3)
       The Mean Square Error (MSE) value for scenario 3 with the SVM model is 0.08333333333333333
```

```
[736... RMSE_SVM_3 = np.sqrt(MSE_SVM_3)
```

```
[738... print("The Root Mean Square Error (RMSE) value for scenario 3 with the SVM model is", RMSE_SVM_3)
       The Root Mean Square Error (RMSE) value for scenario 3 with the SVM model is 0.28867513459481287
```

Scenario 4

https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python

[742...
```
y_train4 = np.ravel(y_train4)
```

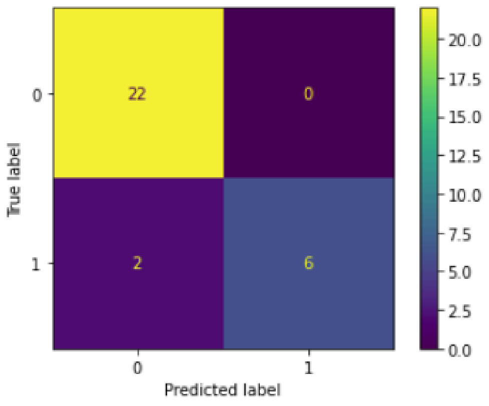
[744...
```
SVM_Model_4 = svm.SVC()
SVM_Model_4.fit(x_train4,y_train4)
```

[744...
```
SVC()
```

[746...
```
X_Test_Prediction_SVM_4 = SVM_Model_4.predict(x_test4)
```

[748...
```
Confusion_Matrix_SVM_4 = metrics.confusion_matrix(y_test4, X_Test_Prediction_SVM_4)
cm_display_SVM_4 = metrics.ConfusionMatrixDisplay(Confusion_Matrix_SVM_4)
cm_display_SVM_4.plot()
```

[748...
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361fb6a400>
```



[750...
```
Accuracy_SVM_4 = metrics.accuracy_score(y_test4, X_Test_Prediction_SVM_4)
Accuracy_SVM_4
```

[750...
```
0.9333333333333333
```

[752...
```
R_Square_SVM_4 = r2_score(y_test4, X_Test_Prediction_SVM_4)
```

[754...
```
print("The R-Squared value for scenario 4 is", R_Square_SVM_4)
```
```
The R-Squared value for scenario 4 is 0.6590909090909
```

[756...
```
Adjusted_R_Square_SVM_4 = 1 - ((1 - R_Square_SVM_4)*(len(y_train4) - 1)/(len(y_train4) - x_train4.shape[1] - 1))
```

[758...
```
print("The Adjusted R-Squared value for scenario 4 is", Adjusted_R_Square_SVM_4)
```
```
The Adjusted R-Squared value for scenario 4 is 0.6431107954545453
```

[760...
```
MSE_SVM_4 = mean_squared_error(y_test4, X_Test_Prediction_DT_4)
```

[762...
```
print("The Mean Square Error (MSE) value for scenario 4 with the SVM model is", MSE_SVM_4)
```
```
The Mean Square Error (MSE) value for scenario 4 with the SVM model is 0.16666666666666666
```

[764...
```
RMSE_SVM_4 = np.sqrt(MSE_SVM_4)
```

[766...
```
print("The Root Mean Square Error (RMSE) value for scenario 4 with the SVM model is", RMSE_SVM_4)
```
```
The Root Mean Square Error (RMSE) value for scenario 4 with the SVM model is 0.408248290463863
```

Scenario 5

```
[769... y_train5 = np.ravel(y_train5)
```

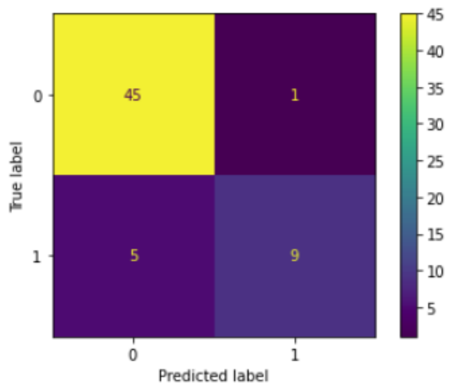
```
[771... SVM_Model_5 = svm.SVC()
       SVM_Model_5.fit(x_train5,y_train5)
```

```
[771... SVC()
```

```
[773... X_Test_Prediction_SVM_5 = SVM_Model_5.predict(x_test5)
```

```
[775... Confusion_Matrix_SVM_5 = metrics.confusion_matrix(y_test5, X_Test_Prediction_SVM_5)
       cm_display_SVM_5 = metrics.ConfusionMatrixDisplay(Confusion_Matrix_SVM_5)
       cm_display_SVM_5.plot()
```

```
[775... <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361fd54df0>
```



```
[777... Accuracy_SVM_5 = metrics.accuracy_score(y_test5, X_Test_Prediction_SVM_5)
       Accuracy_SVM_5
```

```
[777... 0.9
```

```
[779... R_Square_SVM_5 = r2_score(y_test5, X_Test_Prediction_SVM_5)
```

```
[781... print("The R-Squared value for scenario 5 is", R_Square_SVM_5)
       The R-Squared value for scenario 5 is 0.4409937888198756
```

```
[783... Adjusted_R_Square_SVM_5 = 1 - ((1 - R_Square_SVM_5)*(len(y_train5) - 1)/(len(y_train5) - x_train5.shape[1] - 1))
```

```
[785... print("The Adjusted R-Squared value for scenario 5 is", Adjusted_R_Square_SVM_5)
       The Adjusted R-Squared value for scenario 5 is 0.42899794737824215
```

```
[787... MSE_SVM_5 = mean_squared_error(y_test5, X_Test_Prediction_DT_5)
```

```
[789... print("The Mean Square Error (MSE) value for scenario 5 with the SVM model is", MSE_SVM_5)
       The Mean Square Error (MSE) value for scenario 5 with the SVM model is 0.1
```

```
[791... RMSE_SVM_5 = np.sqrt(MSE_SVM_5)
```

```
[793... print("The Root Mean Square Error (RMSE) value for scenario 5 with the SVM model is", RMSE_SVM_5)
       The Root Mean Square Error (RMSE) value for scenario 5 with the SVM model is 0.31622776601683794
```

Scenario 6

```
[796...  y_train6 = np.ravel(y_train6)
```

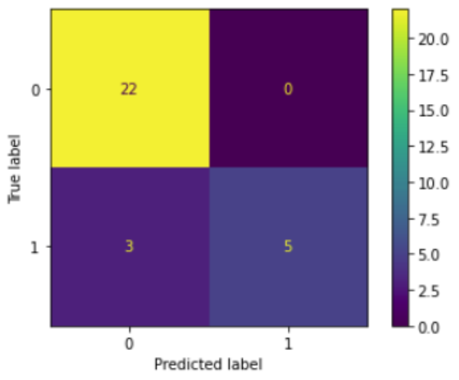
```
[798...  SVM_Model_6 = svm.SVC()
         SVM_Model_6.fit(x_train6,y_train6)
```

```
[798...  SVC()
```

```
[800...  X_Test_Prediction_SVM_6 = SVM_Model_6.predict(x_test6)
```

```
[802...  Confusion_Matrix_SVM_6 = metrics.confusion_matrix(y_test6, X_Test_Prediction_SVM_6)
         cm_display_SVM_6 = metrics.ConfusionMatrixDisplay(Confusion_Matrix_SVM_6)
         cm_display_SVM_6.plot()
```

```
[802...  <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361fe6cd90>
```



```
[804...  Accuracy_SVM_6 = metrics.accuracy_score(y_test6, X_Test_Prediction_SVM_6)
         Accuracy_SVM_6
```

```
[804...  0.9
```

```
[806...  R_Square_SVM_6 = r2_score(y_test6, X_Test_Prediction_SVM_6)
```

```
[808...  print("The R-Squared value for scenario 6 is", R_Square_SVM_6)
         The R-Squared value for scenario 6 is 0.48863636363636354
```

```
[810...  Adjusted_R_Square_SVM_6 = 1 - ((1 - R_Square_SVM_6)*(len(y_train6) - 1)/(len(y_train6) - x_train6.shape[1] - 1))
```

```
[812...  print("The Adjusted R-Squared value for scenario 6 is", Adjusted_R_Square_SVM_6)
         The Adjusted R-Squared value for scenario 6 is 0.4789146215001727
```

```
[814...  MSE_SVM_6 = mean_squared_error(y_test6, X_Test_Prediction_DT_6)
```

```
[816...  print("The Mean Square Error (MSE) value for scenario 6 with the SVM model is", MSE_SVM_6)
         The Mean Square Error (MSE) value for scenario 6 with the SVM model is 0.13333333333333333
```

```
[818...  RMSE_SVM_6 = np.sqrt(MSE_SVM_6)
```

```
[820...  print("The Root Mean Square Error (RMSE) value for scenario 6 with the SVM model is", RMSE_SVM_6)
         The Root Mean Square Error (RMSE) value for scenario 6 with the SVM model is 0.3651483716701107
```

Scenario 7

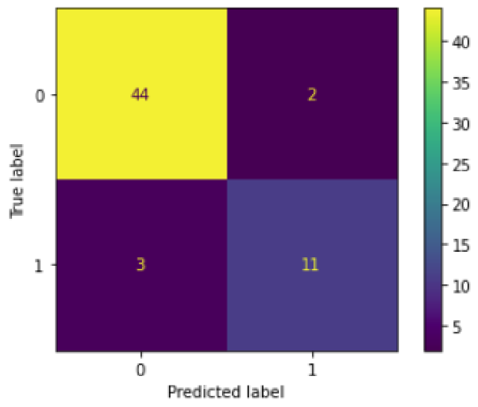
```
[823...  SVM_Model_7 = svm.SVC()
         SVM_Model_7.fit(x_train7,y_train7)
```

```
[823...  SVC()
```

```
[825...  X_Test_Prediction_SVM_7 = SVM_Model_7.predict(x_test7)
```

```
[827...  Confusion_Matrix_SVM_7 = metrics.confusion_matrix(y_test7, X_Test_Prediction_SVM_7)
         cm_display_SVM_7 = metrics.ConfusionMatrixDisplay(Confusion_Matrix_SVM_7)
         cm_display_SVM_7.plot()
```

```
[827...  <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361fd957f0>
```



```
[829...  Accuracy_SVM_7 = metrics.accuracy_score(y_test7, X_Test_Prediction_SVM_7)
         Accuracy_SVM_7
```

```
[829...  0.9166666666666666
```

```
[831...  R_Square_SVM_7 = r2_score(y_test7, X_Test_Prediction_SVM_7)
```

```
[833...  print("The R-Squared value for scenario 7 is", R_Square_SVM_7)
         The R-Squared value for scenario 7 is 0.5341614906832297
```

```
[835...  Adjusted_R_Square_SVM_7 = 1 - ((1 - R_Square_SVM_7)*(len(y_train7) - 1)/(len(y_train7) - x_train7.shape[1] - 1))
```

```
[837...  print("The Adjusted R-Squared value for scenario 7 is", Adjusted_R_Square_SVM_7)
         The Adjusted R-Squared value for scenario 7 is 0.5241649561485351
```

```
[839...  MSE_SVM_7 = mean_squared_error(y_test7, X_Test_Prediction_DT_7)
```

```
[841...  print("The Mean Square Error (MSE) value for scenario 7 with the SVM model is", MSE_SVM_7)
         The Mean Square Error (MSE) value for scenario 7 with the SVM model is 0.1
```

```
[843...  RMSE_SVM_7 = np.sqrt(MSE_SVM_7)
```

```
[845...  print("The Root Mean Square Error (RMSE) value for scenario 7 with the SVM model is", RMSE_SVM_7)
         The Root Mean Square Error (RMSE) value for scenario 7 with the SVM model is 0.31622776601683794
```

Scenario 8

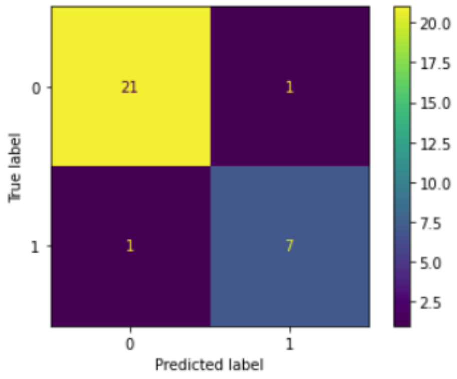
```
[848... SVM_Model_8 = svm.SVC()
        SVM_Model_8.fit(x_train8,y_train8)
```

```
[848... SVC()
```

```
[850... X_Test_Prediction_SVM_8 = SVM_Model_8.predict(x_test8)
```

```
[852... Confusion_Matrix_SVM_8 = metrics.confusion_matrix(y_test8, X_Test_Prediction_SVM_8)
        cm_display_SVM_8 = metrics.ConfusionMatrixDisplay(Confusion_Matrix_SVM_8)
        cm_display_SVM_8.plot()
```

```
[852... <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2361fccf730>
```



```
[854... Accuracy_SVM_8 = metrics.accuracy_score(y_test8, X_Test_Prediction_SVM_8)
        Accuracy_SVM_8
```

```
[854... 0.9333333333333333
```

```
[856... R_Square_SVM_8 = r2_score(y_test8, X_Test_Prediction_SVM_8)
```

```
[858... print("The R-Squared value for scenario 8 is", R_Square_SVM_8)
        The R-Squared value for scenario 8 is 0.659090909090909
```

```
[860... Adjusted_R_Square_SVM_8 = 1 - ((1 - R_Square_SVM_8)*(len(y_train8) - 1)/(len(y_train8) - x_train8.shape[1] - 1))
```

```
[862... print("The Adjusted R-Squared value for scenario 8 is", Adjusted_R_Square_SVM_8)
        The Adjusted R-Squared value for scenario 8 is 0.6526097476667818
```

```
[864... MSE_SVM_8 = mean_squared_error(y_test8, X_Test_Prediction_DT_8)
```

```
[866... print("The Mean Square Error (MSE) value for scenario 8 with the SVM model is", MSE_SVM_8)
        The Mean Square Error (MSE) value for scenario 8 with the SVM model is 0.13333333333333333
```

```
[868... RMSE_SVM_8 = np.sqrt(MSE_SVM_8)
```

```
[870... print("The Root Mean Square Error (RMSE) value for scenario 8 with the SVM model is", RMSE_SVM_8)
        The Root Mean Square Error (RMSE) value for scenario 8 with the SVM model is 0.3651483716701107
```

References

https://www.w3schools.com/python/pandas/pandas_csv.asp

https://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.StandardScaler.html

https://www.kidneyfund.org/all-about-kidneys/tests/serum-creatinine-test#:~:text=Your%20serum%20creatinine%20level%20is,creatinine%20with%20a%20urine%20test.

https://www.kidney.org/kidney-topics/heart-and-kidney-connection#:~:text=Kidney%20disease%20can%20raise%20the,the%20heart%20and%20blood%20vessels.

https://www.geeksforgeeks.org/how-to-create-a-correlation-matrix-using-pandas/

https://atlan.com/standardize-data/

https://nbconvert.readthedocs.io/en/latest/install.html#installing-tex.

https://stackoverflow.com/questions/73847486/jupyter-notebook-download-as-error-nbconvert-failed-no-filter-named-escape-htm

https://pypi.org/project/pandoc/

https://askubuntu.com/questions/1177096/error-saving-to-pdf-from-jupyter-notebook

https://www.geeksforgeeks.org/python-coefficient-of-determination-r2-score/

https://discourse.jupyter.org/t/font-size-for-cells-on-notebooks-in-jupyterlab/11398/3

https://stackoverflow.com/questions/42621190/display-this-decision-tree-with-graphviz