B.S. Abdur Rahman
Crescent
Institute of Science & Technology
Deemed to be University u/s 3 of the UGC Act, 1956

DEPARTMENT OF COMPUTER APPLICATIONS

CADX116 - MACHINE LEARNING ALGORITHMS LABORATORY

B.S. Abdur Rahman
Crescent
Institute of Science & Technology
Deemed to be University u/s 3 of the UGC Act, 1956

LAB RECORD

NAME **:** _____

RRN **:** _____

LAB **:** ***CADX116 - MACHINE LEARNING ALGORITHMS LABORATORY***

B.S. Abdur Rahman Crescent Institute of Science & Technology

Deemed to be University u/s 3 of the UGC Act, 1956

DEPARTMENT OF COMPUTER APPLICATIONS

ACADEMIC YEAR (JULY 2024 - DECEMBER 2024)

COURSE CODE : CADX116

COURSE NAME : MACHINE LEARNING ALGORITHMS LABORATORY

PROGRAMME : BCA (DATA SCIENCE)

SEMESTER : V

B.S. Abdur Rahman
Crescent
Institute of Science & Technology
Deemed to be University u/s 3 of the UGC Act, 1956

BONAFIDE CERTIFICATE

This is a Certified Record Book of _____

RRN: _____ ***submitted for the Semester End Practical Examination held on*** _____, ***for the CADX116 - MACHINE LEARNING ALGORITHMS LABORATORY during 2024 - 2025.***

……..

Signature of Faculty

INDEX

Ex.No.	*Date*	*List of Experiments*	*Page No*	*Signature*
1.		*Implement Naive Bayes Classifier*		
2.		*Implement Linear Regression*		
3.		*Implement Support Vector Machine (SVM) Algorithm*		
4.		*Implement Decision Tree Classifier*		
5.		*Implement Random Forest Algorithm*		
6.		*Implement K-Means Clustering Algorithm*		
7.		*Implement Principal Component Analysis (PCA)*		
8.		*Implement K-Nearest Neighbors (KNN) Algorithm*		
9.		*Demonstrate Weka Tool*		
10.		*Build An Unsupervised Models Using [Scikit-Learn And Pytorch]*		
11.		*Implement Logistic Regression*		
12.		*Implement Neural Network Model*		

EX.NO :

DATE :

IMPLEMENT NAÏVE BAYES CLASSIFIER

SOURCE CODE :

Step 1:

```
[ ]  import pandas as pd
     from sklearn import tree
     from sklearn.preprocessing import LabelEncoder
     from sklearn.naive_bayes import GaussianNB
```

Step 2:

```
[ ]  data=pd.read_csv("Naive_Bayes_PlayTennis.csv")
     print("the First 5 values of data is :\n")
     data.head()
```

Output:

the First 5 values of data is :

	Outlook	***Temperature***	***Humidity***	***Windy***	***Play Tennis***
0	*Sunny*	*Hot*	*High*	*False*	*No*
1	*Sunny*	*Hot*	*High*	*True*	*No*
2	*Overcast*	*Hot*	*High*	*False*	*Yes*
3	*Rainy*	*Mild*	*High*	*False*	*Yes*
4	*Rainy*	*Cool*	*Normal*	*False*	*Yes*

Step 3:

```
[ ]  X=data.iloc[:,:-1]
     print("the First 5 values of data is :\n")
     X.head()
```

Output:

the First 5 values of data is :

	Outlook	***Temperature***	***Humidity***	***Windy***
0	*Sunny*	*Hot*	*High*	*False*
1	*Sunny*	*Hot*	*High*	*True*
2	*Overcast*	*Hot*	*High*	*False*
3	*Rainy*	*Mild*	*High*	*False*
4	*Rainy*	*Cool*	*Normal*	*False*

Step 4:

```
[ ]  y=data.iloc[:,-1]
     print("the First 5 values of data is :\n")
     y.head()
```

Output:

the First 5 values of data is :

```
0    No
1    No
2   Yes
3   Yes
4   Yes
Name: PlayTennis, dtype: object
```

Step 5:

```
[ ]  for column in X.columns:
         X[column]=LabelEncoder().fit_transform(X[column])
     X.head()
```

Output:

	Outlook	***Temperature***	***Humidity***	***Windy***
0	*2*	*1*	*0*	*0*
1	*2*	*1*	*0*	*1*
2	*0*	*1*	*0*	*0*
3	*1*	*2*	*0*	*0*
4	*1*	*0*	*1*	*0*

Step 6:

```
[ ]  y=LabelEncoder().fit_transform(y)
     print(y)
```

Output:

[0 0 1 1 1 0 1 0 1 1 1 1 1 0]

Step 7:

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

Step 8:

```
[ ]  X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20)
```

Step 9:

```
[ ]  from sklearn.naive_bayes import GaussianNB
     classifier = GaussianNB()
     classifier.fit(X_train,y_train)
```

Output:

```
▾ GaussianNB
GaussianNB()
```

Step 10:

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

Step 11:

```
[ ]  print("Accuracy is:",accuracy_score(classifier.predict(X_test),(y_test)))
```

Output:

Accuracy is: 0.6666666666666666

EX.NO :

DATE :

IMPLEMENT LINEAR REGRESSION

SOURCE CODE :

Step 1:

```
[ ]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LinearRegression
     from sklearn.metrics import mean_squared_error, r2_score
```

Step 2:

```
[ ]  data=pd.read_csv("Linear_Regression_Advertising.csv")
     print("the First 5 values of data is :\n")
     data.head()
```

Output:

the First 5 values of data is :

	TV	***Radio***	***Newspaper***	***Sales***
0	*230.1*	*37.8*	*69.2*	*22.1*
1	*44.5*	*39.3*	*45.1*	*10.4*
2	*17.2*	*45.9*	*69.3*	*9.3*
3	*151.5*	*41.3*	*58.5*	*18.5*
4	*180.8*	*10.8*	*58.4*	*12.9*

Step 3:

```
[ ]  X =data[['TV', 'Radio', 'Newspaper']]
     y = data['Sales']
     X.head()
```

Output:

	TV	***Radio***	***Newspaper***
0	*230.1*	*37.8*	*69.2*
1	*44.5*	*39.3*	*45.1*
2	*17.2*	*45.9*	*69.3*
3	*151.5*	*41.3*	*58.5*
4	*180.8*	*10.8*	*58.4*

Step 4:

```
[ ]  y.head()
```

Output:

```
0    22.1
1    10.4
2     9.3
3    18.5
4    12.9
Name: Sales, dtype: float64
```

Step 5:

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
     model = LinearRegression()
     model.fit(X_train, y_train)
```

Output:

```
▾ LinearRegression
LinearRegression()
```

Step 6:

```
[ ]  y_pred = model.predict(X_test)
```

Step 7:

```
[ ]  mse = mean_squared_error(y_test, y_pred)
     r2 = r2_score(y_test, y_pred)
     print(f'Mean Squared Error: {mse}')
     print(f'R^2 Score: {r2}')
```

Output :

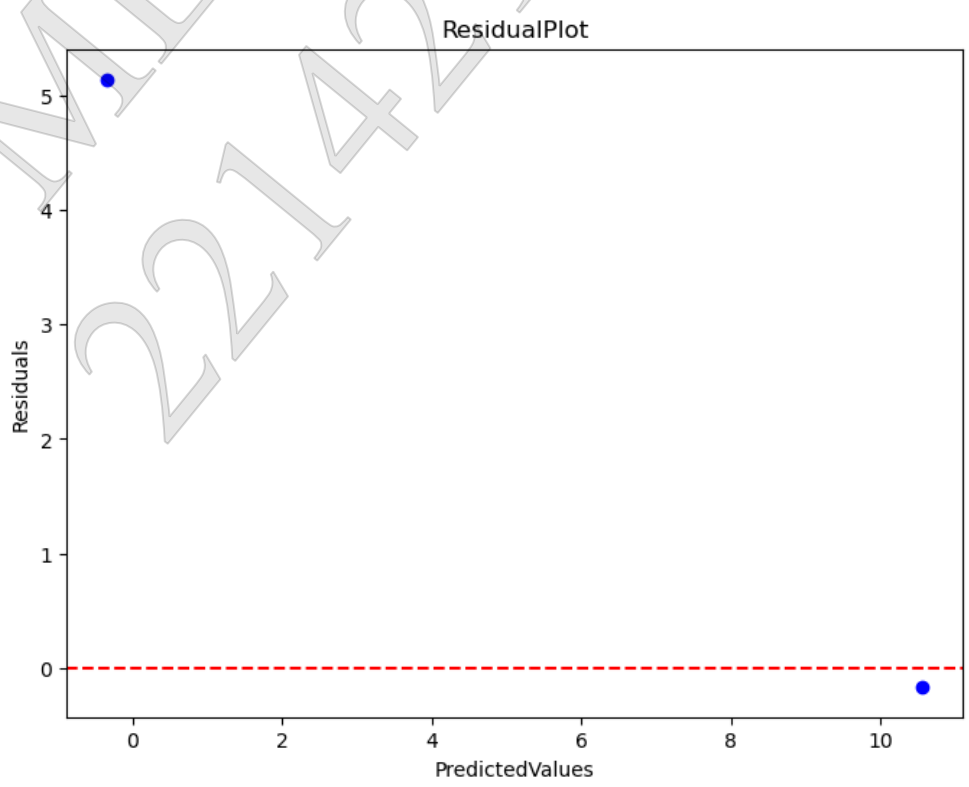
Mean Squared Error: 13.201451790963432

R^2 Score: -0.6838586468065599

Step 8:

```
[ ]  residuals=y_test-y_pred
     plt.figure(figsize=(8,6))
     plt.scatter(y_pred,residuals,color='blue')
     plt.title('ResidualPlot')
     plt.xlabel('PredictedValues')
     plt.ylabel('Residuals')
     plt.axhline(y=0,color='r', linestyle='--')
     plt.show()
```

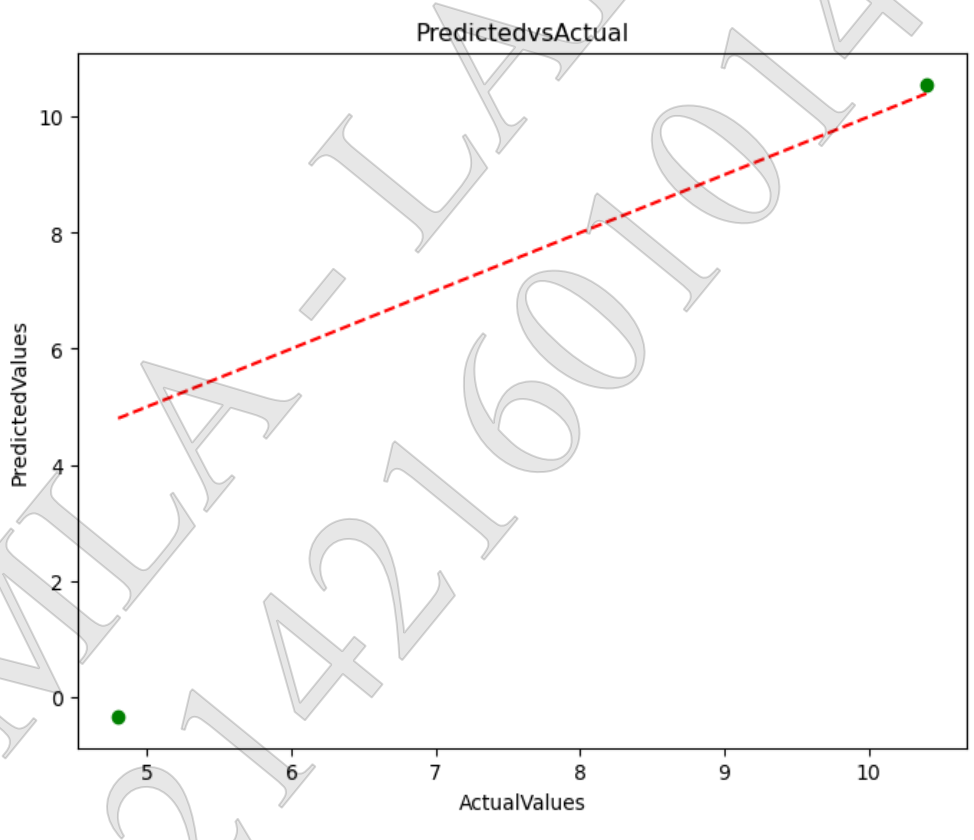
Output:



Step 9:

```
[ ]   plt.figure(figsize=(8,6))
      plt.scatter(y_test,y_pred,color='green')
      plt.title('PredictedvsActual')
      plt.xlabel('ActualValues')
      plt.ylabel('PredictedValues')
      plt.plot([min(y_test),max(y_test)], [min(y_test),max(y_test)],color='red',
      linestyle='--')
      plt.show()
```

Output:



EX.NO :

DATE :

IMPLEMENT SUPPORT VECTOR MACHINE (SVM) ALGORITHM

SOURCE CODE :

Step 1:

```
[ ]  import pandas as pd
     import numpy as np
     from sklearn.model_selection import train_test_split,cross_val_score
     from sklearn.preprocessing import StandardScaler, LabelEncoder
     from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,
     recall_score, f1_score
     from sklearn.svm import SVC
```

Step 2:

```
[ ]  data=pd.read_csv("SVM_PurchasePrediction.csv")
     print("the First 5 values of data is :\n")
     data.head()
```

Output:

the First 5 values of data is :

	Age	***Income***	***Gender***	***Purchase***
0	*25*	*50000*	*Male*	*0*
1	*45*	*64000*	*Female*	*1*
2	*35*	*58000*	*Female*	*0*
3	*50*	*72000*	*Male*	*1*
4	*23*	*48000*	*Male*	*0*

Step 3:

```
[ ]  le=LabelEncoder()
     data['Gender']=le.fit_transform(data['Gender'])
```

Step 4:

```
[ ]  data.fillna(method='ffill',inplace=True)
```

Step 5:

```
[ ]  X =data.drop('Purchase',axis=1)
     y = data['Purchase']
```

Step 6:

```
[ ]  sc=StandardScaler()
     X_scaled=sc.fit_transform(X)
```

Step 7:

```
[ ]  classifiers={
       'Linear SVM': SVC(kernel='linear',random_state=0),
       'Polynomial SVM': SVC(kernel='poly',degree=3,random_state=0),
       'RBF SVM': SVC(kernel='rbf',random_state=0)
     }
```

Step 8:

```
[ ]  for clf_name,clf in classifiers.items():
       scores=cross_val_score(clf,X_scaled,y,cv=5,scoring='accuracy')
       print(f"{clf_name} Cross-validation Accuracy : {scores.mean():.2f}(+/-
     {scores.std()*2:.2f})")
```

Output:

Linear SVM Cross-validation Accuracy : 0.80(+/-0.33)
Polynomial SVM Cross-validation Accuracy : 0.80(+/-0.53)
RBF SVM Cross-validation Accuracy : 0.80(+/-0.53)

Step 9:

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
     classifier=SVC(kernel='linear',random_state=0)
     classifier.fit(X_train, y_train)
```

Output:

```
SVC
SVC(kernel='linear', random_state=0)
```

Step 10:

```
[ ]  y_pred=classifier.predict(X_test)
```

Step 11:

```
[ ]  cm=confusion_matrix(y_test,y_pred)
     accuracy=accuracy_score(y_test,y_pred)
     precision=precision_score(y_test,y_pred)
     recall=recall_score(y_test,y_pred)
     f1=f1_score(y_test,y_pred)
```

Step 12:

```
[ ]  print(f"Confusion Matrix :\n",cm)
     print(f"Accuracy :{accuracy:.2f}")
     print(f"Precision :{precision:.2f}")
     print(f"Recall :{recall:.2f}")
     print(f"F1 Performance score :{f1:.2f}")
```

Output :

Confusion Matrix :
[[4]]
Accuracy :1.00
Precision :1.00
Recall :1.00
F1 Performance score :1.00

EX.NO :

DATE :

IMPLEMENT DECISION TREE CLASSIFIER

SOURCE CODE :

Step 1:

```
[ ]  import pandas as pd
     import numpy as np
     from sklearn.model_selection import train_test_split
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.metrics import
     confusion_matrix,accuracy_score,precision_score,recall_score,f1_score
```

Step 2:

```
[ ]  data=pd.read_csv("DecisionTree_TargetIncome.csv")
     print("the First 5 values of data is :\n")
     data.head()
```

Output:

the First 5 values of data is :

	Age	*Income*	*Gender*	*TargetVariable*
0	*25*	*50000*	*Male*	*0*
1	*45*	*64000*	*Female*	*1*
2	*35*	*58000*	*Female*	*0*
3	*50*	*72000*	*Male*	*1*
4	*23*	*48000*	*Male*	*0*

Step 3:

```
[ ]  data=pd.get_dummies(data, columns=['Gender'],drop_first=True)
```

Step 4:

```
[ ]  print(data.columns)
```

Step 5:

```
[ ]  X =data.drop('TargetVariable',axis=1)
     y = data['TargetVariable']
```

Step 6:

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
     classifier=DecisionTreeClassifier(random_state=0)
     classifier.fit(X_train, y_train)
```

Output:

```
▾        DecisionTreeClassifier
DecisionTreeClassifier(random_state=0)
```

Step 7:

```
[ ]  y_pred=classifier.predict(X_test)
```

Step 8:

```
[ ]  cm=confusion_matrix(y_test,y_pred)
     accuracy=accuracy_score(y_test,y_pred)
     precision=precision_score(y_test,y_pred)
     recall=recall_score(y_test,y_pred)
     f1=f1_score(y_test,y_pred)
```

Step 9:

```
[ ]  print(f"Confusion Matrix :\n",cm)
     print(f"Accuracy :{accuracy:.2f}")
     print(f"Precision :{precision:.2f}")
     print(f"Recall :{recall:.2f}")
     print(f"F1 score :{f1:.2f}")
```

Output :

Confusion Matrix :

[[0 0]

[1 3]]

Accuracy :0.75

Precision :1.00

Recall :0.75

F1 score :0.86

EX.NO :

DATE :

IMPLEMENT RANDOM FOREST ALGORITHM

SOURCE CODE :

Step 1:

```
[ ]  import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.metrics import accuracy_score,classification_report
     import warnings
     warnings.filterwarnings('ignore')
```

Step 2:

```
[ ]  titanic_data=pd.read_csv('RandomForest_TitanicSurvival.csv')
     titanic_data.head()
```

Output:

	Passenger Id	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25	-	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.925	-	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05	-	S

Step 3:

```
[ ]  titanic_data=titanic_data.dropna(subset=['Survived'])
```

Step 4:

```
[ ]  X=titanic_data[['Pclass','Sex','Age','SibSp','Parch','Fare']]
     y=titanic_data['Survived']
```

Step 5:

```
[ ]  X.loc[:,'Sex']=X['Sex'].map({'female':0,'male':1})
     X.loc[:,'Age']=X['Age'].fillna(X['Age'].median())
```

Step 6:

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
     rf_classifier=RandomForestClassifier(n_estimators=100,random_state=42)
     rf_classifier.fit(X_train, y_train)
```

Output:

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Step 7:

```
[ ]  y_pred=rf_classifier.predict(X_test)
```

Step 8:

```
[ ]  accuracy=accuracy_score(y_test,y_pred)
     classification_rep=classification_report(y_test,y_pred)
```

Step 9:

```
[ ]  print(f"Accurancy :{accuracy:.2f}")
     print(f"Classification Report :\n",classification_rep)
```

Output :

Accurancy :0.80

Classification Report :

	precision	*recall*	*f1-score*	*support*
0	*0.82*	*0.85*	*0.83*	*105*
1	*0.77*	*0.73*	*0.75*	*74*
accuracy			*0.80*	*179*
macro avg	*0.79*	*0.79*	*0.79*	*179*
weighted avg	*0.80*	*0.80*	*0.80*	*179*

EX.NO :

DATE :

IMPLEMENT K-MEANS CLUSTERING ALGORITHM

SOURCE CODE :

Step 1:

```
[ ]  import pandas as pd
     import numpy as np
     from sklearn.cluster import KMeans
     import matplotlib.pyplot as plt
```

Step 2:

```
[ ]  data=pd.read_csv("KMeans_FeatureClusters.csv")
     print("the First 5 values of data is :\n")
     data.head()
```

Output:

the First 5 values of data is :

	Feature1	***Feature2***
0	*2.3*	*3.4*
1	*1.5*	*1.8*
2	*7.6*	*6.5*
3	*2.1*	*4.2*
4	*8.0*	*7.0*

Step 3:

```
[ ]  kmeans=KMeans(n_clusters=2,random_state=0)
     kmeans.fit(df)
```

Output:

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the
value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there
are less chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
```

```
▾          KMeans
KMeans(n_clusters=2, random_state=0)
```

Step 4:

```
[ ]  df['cluster']=kmeans.labels_
```

Step 5:

```
[ ]  print(df.head())
```

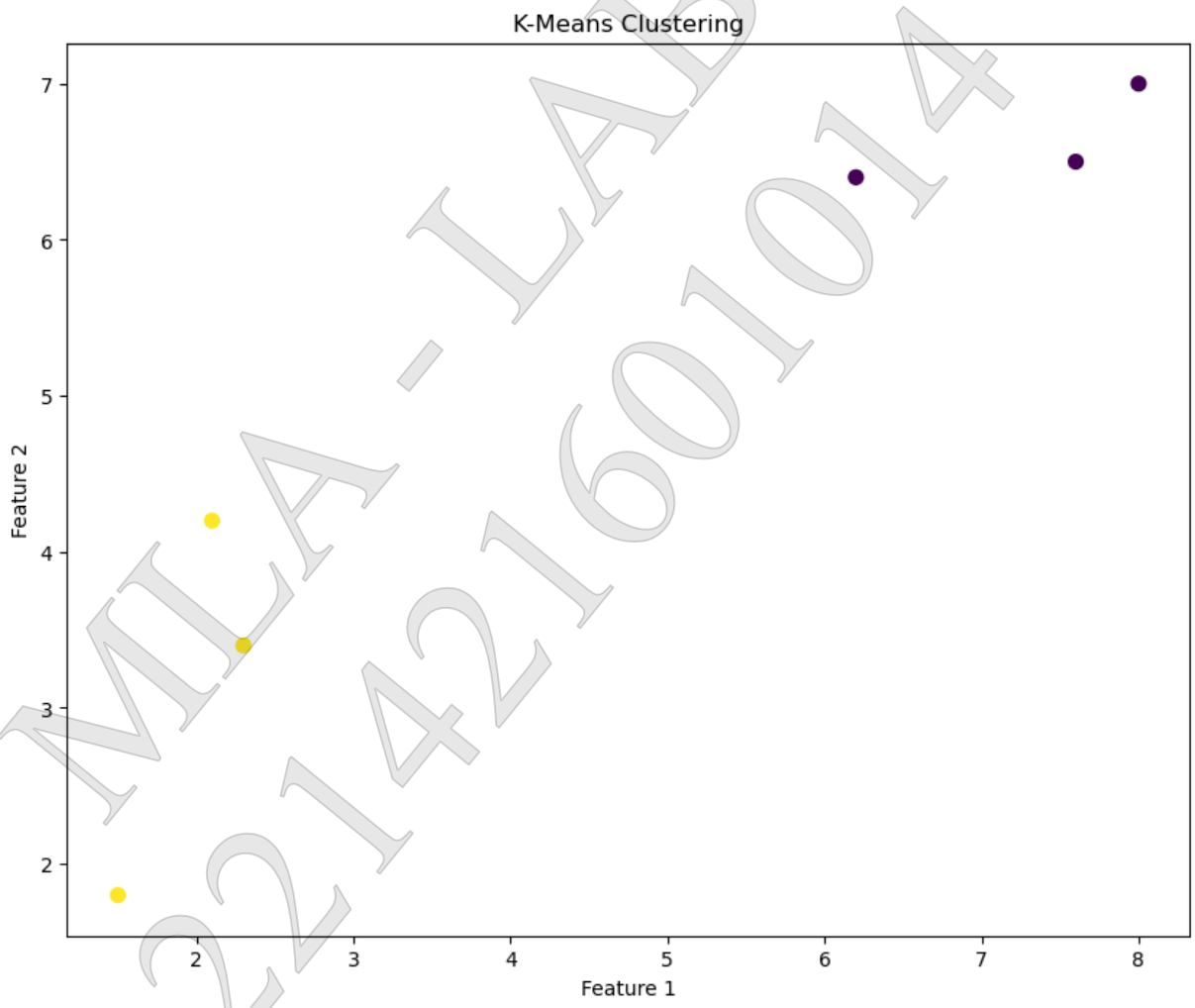
Output:

	Feature1	***Feature2***	***Cluster***
0	*2.3*	*3.4*	*1*
1	*1.5*	*1.8*	*1*
2	*7.6*	*6.5*	*0*
3	*2.1*	*4.2*	*1*
4	*8.0*	*7.0*	*0*

Step 6:

```
[ ]   plt.figure(figsize=(10,8))
      plt.scatter(df['Feature1'],df['Feature2'],c=df['cluster'],cmap='viridis',s=50)
      plt.title('K-Means Clustering')
      plt.xlabel('Feature 1')
      plt.ylabel('Feature 2')
      plt.show()
```

Output:



EX.NO :

DATE :

IMPLEMENT PRINCIPAL COMPONENT ANALYSIS (PCA)

SOURCE CODE :

Step 1:

```
[ ]  import pandas as pd
     import numpy as np
     from sklearn.decomposition import PCA
     from sklearn.preprocessing import StandardScaler
     import matplotlib.pyplot as plt
     import seaborn as sns
```

Step 2:

```
[ ]  data=pd.read_csv("PCA_CustomerSegmentation.csv")
     print("the First 5 values of data is :\n")
     data.head()
```

Output:

the First 5 values of data is :

	Age	*Annual Income*	*Spending Score*	*Number of Purchases*	*Cluster*
0	*22*	*35000*	*45*	*15*	*3*
1	*28*	*45000*	*50*	*18*	*2*
2	*33*	*60000*	*55*	*22*	*1*
3	*38*	*70000*	*60*	*25*	*4*
4	*45*	*80000*	*65*	*30*	*2*

Step 3:

```
[ ]  Features=['Age','Annual Income','Spending Score','Number of Purchases']
     X=df[Features]
```

Step 4:

```
[ ]  Scaler=StandardScaler()
     X_scaled=Scaler.fit_transform(X)
```

Step 5:

```
[ ]  pca=PCA(n_components=2)
     X_pca=pca.fit_transform(X_scaled)
```

Step 6:

```
[ ]  df_pca=pd.DataFrame(data=X_pca,columns=['pc1','pc2'])
     df_pca['Cluster']=df['Cluster']
```

Step 7:

```
[ ]  plt.figure(figsize=(10,6))
     sns.scatterplot(x='pc1',y='pc2',hue='Cluster',data=df_pca,palette='viridis')
     plt.title('PCA Result - Customer Segmentation')
     plt.xlabel('principal component 1')
     plt.ylabel('principal component 2')
     plt.show()
```

Output :

Step 8:

```
[ ]  explained_variance=pca.explained_variance_ratio_
     print(f'Explained Variance by each Component: {explained_variance}')
     print(f'Total Explained Variance : {np.sum(explained_variance)}')
```

Output :

Explained Variance by each Component: [0.87276806 0.1144451]

Total Explained Variance : 0.9872131621734076

EX.NO :

DATE :

IMPLEMENT K-NEAREST NEIGHBORS (KNN) ALGORITHM

SOURCE CODE :

Step 1:

```
[ ]  import pandas as pd
     import plotly.graph_objects as go
     import plotly.offline as pyoff
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

Step 2:

```
[ ]  data=pd.read_csv("KNN_CellClassification.csv")
     data.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 698 entries, 0 to 697
Data columns (total 11 columns):
 #   Column   Non-Null Count  Dtype
---  ------  --------------  -----
 0   1000025  698 non-null    int64
 1   5        698 non-null    int64
 2   1        698 non-null    int64
 3   1.1      698 non-null    int64
 4   1.2      698 non-null    int64
 5   2        698 non-null    int64
 6   1.3      698 non-null    object
 7   3        698 non-null    int64
 8   1.4      698 non-null    int64
 9   1.5      698 non-null    int64
 10  2.1      698 non-null    int64
dtypes: int64(10), object(1)
memory usage: 60.1+ KB
```

Step 3:

```
[ ]  data.head()
```

Output:

	1000025	5	1	1.1	1.2	2	1.3	3	1.4	1.5	2.1
0	1002945	5	4	4	5	7	10	3	2	1	2
1	1015425	3	1	1	1	2	2	3	1	1	2
2	1016277	6	8	8	1	3	4	3	7	1	2
3	1017023	4	1	1	3	2	1	3	1	1	2
4	1017122	8	10	10	8	7	10	9	7	1	4

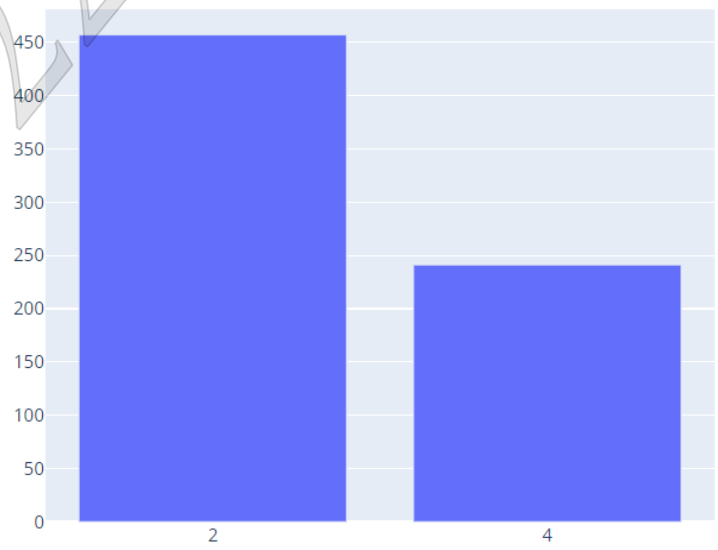
Step 4:

```
[ ]  data.columns=['Id','Clump Thickness','uniformity of cell size','uniformity of cell
     shape','Marginal Adhesion','Single Epithelial cell size','Bare Nuclei','Bland
     Chromatin','Normal Nucleoli','Mitoses','Class']
```

Step 5:

```
[ ]  target_balance=data['Class'].value_counts().reset_index()
     target_balance.columns=['Class','Count']
     target_class=go.Bar(
        name="Target Balance",
        x=target_balance['Class'].astype(str),
        y=target_balance['Count']
        )
     fig=go.Figure(target_class)
     pyoff.iplot(fig)
```

Output:



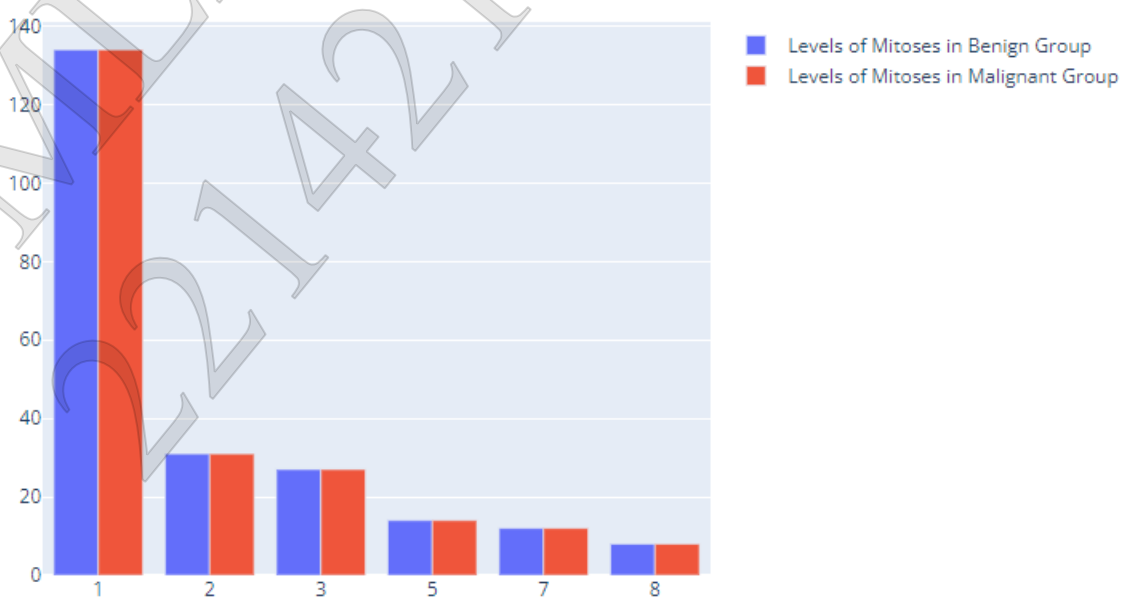
Step 6:

```
[ ]  beg_class_pat=data.loc[data['Class']==2]
     mal_class_pat=data.loc[data['Class']==4]
     Mith_10_beg=beg_class_pat['Mitoses'].value_counts().reset_index()
     Mith_10_beg.columns=['Mitoses','Count']
     Mith_10_mal=mal_class_pat['Mitoses'].value_counts().reset_index()
     Mith_10_mal.columns=['Mitoses','Count']
```

Step 7:

```
[ ]  fig=go.Figure(data=[
       go.Bar(name='Levels of Mitoses in Benign
     Group',x=Mith_10_beg['Mitoses'].astype(str),y=Mith_10_mal['Count']),
       go.Bar(name='Levels of Mitoses in Malignant
     Group',x=Mith_10_beg['Mitoses'].astype(str),y=Mith_10_mal['Count'])])
     fig.update_layout(barmode='group')
     fig.show()
```

Output:



Step 8:

```
[ ]  x=data.drop(columns=['Id','Class'])
     y=data['Class']
```

Step 9:

```
[ ]  print("Unique Values in 'Bara Nuclei':",x['Bare Nuclei'].unique())
```

Output :

Unique Values in 'Bara Nuclei': ['10' '2' '4' '1' '3' '9' '7' '?' '5' '8' '6']

Step 10:

```
[ ]  x['Bare Nuclei']=pd.to_numeric(x['Bare Nuclei'],errors='coerce')
```

Step 11:

```
[ ]   x=x.fillna(x.median())
```

Step 12:

```
[ ]  x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=42)
```

Step 13:

```
[ ]  scaler=StandardScaler()
     x_train_scaled=scaler.fit_transform(x_train)
     x_test_scaled=scaler.transform(x_test)
```

Step 14:

```
[ ]  knn=KNeighborsClassifier(n_neighbors=5)
     knn.fit(x_train_scaled,y_train)
```

Output :

▾ KNeighborsClassifier
KNeighborsClassifier()

Step 15:

```
[ ]  y_pred=knn.predict(x_test_scaled)
```

Step 16:

```
[ ]  print("Accuracy: ",accuracy_score(y_test,y_pred))
     print()
     print("Confusion Matrix: \n",confusion_matrix(y_test,y_pred))
     print("Classification Report: \n",classification_report(y_test,y_pred))
```

Output :

```
Accuracy:  0.9714285714285714


Confusion Matrix:
 [[131   4]
 [  2  73]]
Classification Report:
                precision    recall  f1-score   support

           2       0.98      0.97      0.98       135
           4       0.95      0.97      0.96        75

    accuracy                           0.97       210
   macro avg       0.97      0.97      0.97       210
weighted avg       0.97      0.97      0.97       210
```

EX.NO :

DATE :

DEMONSTRATE WEKA TOOL

SOURCE CODE :

Step 1:

```
[ ]  import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
```

Step 2:

```
[ ]  data=pd.read_csv("Weka_CustomerChurn.csv")
     print("the First 5 values of data is :\n")
     data.head()
```

Output:

the First 5 values of data is :

	Customer ID	Gender	Senior Citizen	Partner	Dependents	Tenure	Phone Service	Monthly Charges	Total Charges	Churn
0	1	Male	0	Yes	No	1	No	29.85	29.85	No
1	2	Female	1	No	No	34	Yes	56.95	1889.50	No
2	3	Female	0	Yes	Yes	2	Yes	53.85	108.15	Yes
3	4	Male	0	No	Yes	45	Yes	42.30	1840.75	No
4	5	Female	1	No	No	2	No	70.70	151.65	Yes

Step 3:

```
[ ]  X = data.drop(columns=['Churn'])
     y = data['Churn'].apply(lambda x: 1 if x == 'Yes' else 0)
```

Step 4:

```
[ ]  X = pd.get_dummies(X, drop_first=True)
```

Step 5:

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 6:

```
[ ]  scaler = StandardScaler()
     X_train = scaler.fit_transform(X_train)
     X_test = scaler.transform(X_test)
```

Step 7:

```
[ ]  model = RandomForestClassifier(n_estimators=100, random_state=42)
```

Step 8:

```
[ ]  model.fit(X_train, y_train)
```

Output:

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Step 9:

```
[ ]  y_pred = model.predict(X_test)
```

Step 10:

```
[ ]  accuracy = accuracy_score(y_test, y_pred)
     conf_matrix = confusion_matrix(y_test, y_pred)
     report = classification_report(y_test, y_pred)
```

Output :

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Step 11:

```
[ ]  print(f'Accuracy: {accuracy}')
     print('Confusion Matrix:')
     print(conf_matrix)
     print('Classification Report:')
     print(report)
```

Output :

```
Accuracy: 0.5
Confusion Matrix:
[[0 1]
 [0 1]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2
```

EX.NO :

DATE :

BUILD AN UNSUPERVISED MODEL USING [Scikit-Learn and PyTorch]

SOURCE CODE :

Step 1:

```
[ ]  import pandas as pd
     from sklearn.datasets import load_iris
     from sklearn.preprocessing import StandardScaler
```

Step 2:

```
[ ]  iris=load_iris()
     X=pd.DataFrame(iris.data,columns=iris.feature_names)
```

Step 3:

```
[ ]  Scaler=StandardScaler()
     X_Scaled=Scaler.fit_transform(X)
```

Step 4:

```
[ ]  from sklearn.cluster import KMeans
     from sklearn.metrics import silhouette_score
     kmeans=KMeans(n_clusters=3,random_state=42)
     kmeans_labels=kmeans.fit_predict(X_Scaled)
```

Output :

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the
value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there
are less chunks than available threads. You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(
```

Step 5:

```
[ ]  kmeans_silhouette=silhouette_score(X_Scaled, kmeans_labels)
     print(f'K-Means Silhouette Score: {kmeans_silhouette}')
```

Output :

K-Means Silhouette Score: 0.45994823920518635

Step 6:

```
[ ]  import torch
     import torch.nn as nn
     import torch.optim as optim
     from torch.utils.data import DataLoader, TensorDataset
     X_tensor=torch.tensor(X_Scaled, dtype=torch.float32)
```

Step 7:

```
[ ]  data_loader=DataLoader(TensorDataset(X_tensor),batch_size=16,shuffle=True)
```

Step 8:

```
[ ]  class Autoencoder(nn.Module):
       def __init__(self):
         super(Autoencoder, self).__init__()
         self.encoder=nn.Sequential(
           nn.Linear(4,2),
           nn.ReLU()
         )
         self.decoder=nn.Sequential(
           nn.Linear(2,4),
           nn.ReLU()
         )
       def forward(self,x):
         encoded=self.encoder(x)
         decoded=self.decoder(encoded)
         return decoded
```

Step 9:

```
[ ]  autoencoder=Autoencoder()
     criterion=nn.MSELoss()
     optimizer=optim.Adam(autoencoder.parameters(),lr=0.01)
```

Step 10:

```
[ ]  num_epochs=100
     for epoch in range(num_epochs):
        for data in data_loader:
            inputs, =data
            optimizer.zero_grad()
            outputs=autoencoder(inputs)
            loss=criterion(outputs, inputs)
            loss.backward()
            optimizer.step()
```

Step 11:

```
[ ]  with torch.no_grad():
       encoded_data=autoencoder.encoder(X_tensor).numpy()
```

Step 12:

```
[ ]  encoded_kmeans=KMeans(n_clusters=3,random_state=42)
     encoded_kmeans_labels=encoded_kmeans.fit_predict(encoded_data)
```

Output :

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set
the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1382:
UserWarning: KMeans is known to have a memory leak on Windows with MKL,
when there are less chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

Step 13:

```
[ ]  encoded_silhouette = silhouette_score(X_Scaled, encoded_kmeans_labels)
     print(f'Autoencoder + kmeans silhouette score : {encoded_silhouette}')
```

Output :

Autoencoder + kmeans silhouette score : 0.302958422921628

EX.NO :

DATE :

IMPLEMENT LOGISTIC REGRESSION

SOURCE CODE :

Step 1:

```
[ ]  import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler, LabelEncoder
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

Step 2:

```
[ ]  data=pd.read_csv("LogisticRegression_HeartDisease.csv")
     print("the First 5 values of data is :\n")
     data.head()
```

Output:

the First 5 values of data is :

	Age	***CholesterolLevel***	***BloodPressure***	***SmokingStatus***	***HeartDisease***
0	*68*	*213.510222*	*119.336301*	*No*	*No*
1	*58*	*209.232228*	*109.821694*	*Yes*	*No*
2	*44*	*194.023226*	*154.003458*	*No*	*No*
3	*72*	*152.111973*	*162.855094*	*Yes*	*No*
4	*37*	*179.826361*	*121.379939*	*No*	*Yes*

Step 3:

```
[ ]  label_encoder = LabelEncoder()
     data['SmokingStatus'] = label_encoder.fit_transform(data['SmokingStatus'])
     data['HeartDisease'] = label_encoder.fit_transform(data['HeartDisease'])
```

Step 4:

```
[ ]  X = data[['Age', 'CholesterolLevel', 'BloodPressure', 'SmokingStatus']]
     X.head()
```

Output:

	Age	*CholesterolLevel*	*BloodPressure*	*SmokingStatus*
0	*68*	*213.510222*	*119.336301*	*0*
1	*58*	*209.232228*	*109.821694*	*1*
2	*44*	*194.023226*	*154.003458*	*0*
3	*72*	*152.111973*	*162.855094*	*1*
4	*37*	*179.826361*	*121.379939*	*0*

Step 5:

```
[ ]  y = data[['HeartDisease']]
     y.head()
```

Output:

	HeartDisease
0	***0***
1	***0***
2	***0***
3	***0***
4	***1***

Step 6:

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
     random_state=42)
```

Step 7:

```
[ ]  scaler = StandardScaler()
     X_train = scaler.fit_transform(X_train)
     X_test = scaler.transform(X_test)
```

Step 8:

```
[ ]  model = LogisticRegression()
     model.fit(X_train, y_train)
```

Output:

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1143: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for
example using ravel().
  y = column_or_1d(y, warn=True)
```

```
▾ LogisticRegression
LogisticRegression()
```

Step 9:

```
[ ]  y_pred = model.predict(X_test)
```

Step 10:

```
[ ]  accuracy = accuracy_score(y_test, y_pred)
     conf_matrix = confusion_matrix(y_test, y_pred)
     class_report = classification_report(y_test, y_pred)
```

Step 11:

```
[ ]  print(f'Accuracy: {accuracy:.2f}')
     print('Confusion Matrix:')
     print(conf_matrix)
     print('Classification Report:')
     print(class_report)
```

Output:

```
Accuracy: 0.60
Confusion Matrix:
[[7 3]
 [5 5]]
Classification Report:
              precision    recall  f1-score   support

           0       0.58      0.70      0.64        10
           1       0.62      0.50      0.56        10

    accuracy                           0.60        20
   macro avg       0.60      0.60      0.60        20
weighted avg       0.60      0.60      0.60        20
```

EX.NO :

DATE :

IMPLEMENT NEURAL NETWORK MODEL

SOURCE CODE :

Step 1:

```
[ ]  import numpy as np
     import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense
     from sklearn.metrics import classification_report, confusion_matrix
     import matplotlib.pyplot as plt
     import seaborn as sns
```

Step 2:

```
[ ]  df=pd.read_csv("NeuralNetwork_PredictiveMaintenance.csv")
     print("the First 5 values of data is :\n")
     df.head()
```

Output:

The First 5 values of data is :

	sensor_1	sensor_2	sensor_3	sensor_4	sensor_5	sensor_6	sensor_7	sensor_8	sensor_9	sensor_10	failure
0	0.374540	0.950714	0.731994	0.598658	0.156019	0.155995	0.058084	0.866176	0.601115	0.708073	0
1	0.020584	0.969910	0.832443	0.212339	0.181825	0.183405	0.304242	0.524756	0.431945	0.291229	1
2	0.611853	0.139494	0.292145	0.366362	0.456070	0.785176	0.199674	0.514234	0.592415	0.046450	1
3	0.607545	0.170524	0.065052	0.948886	0.965632	0.808397	0.304614	0.097672	0.684233	0.440152	1
4	0.122038	0.495177	0.034389	0.909320	0.258780	0.662522	0.311711	0.520068	0.546710	0.184854	0

Step 3:

```
[ ]  X = df.drop('failure', axis=1).values  # Features (sensor data)
     y = df['failure'].values  # Target (failure labels)
```

Step 4:

```
[ ]  scaler = StandardScaler()
     X_scaled = scaler.fit_transform(X)
```

Step 5:

```
[ ]  X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.15,
     random_state=42)

     X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.15,
     random_state=42)
```

Step 6:

```
[ ]  model = Sequential()
     model.add(Dense(64, input_dim=X.shape[1], activation='relu'))
     model.add(Dense(32, activation='relu'))
     model.add(Dense(1, activation='sigmoid'))
```

Output:

```
C:\ProgramData \anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning:
Do not  pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Step 7:

```
[ ]  model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Step 8:

```
[ ]  # Train the model with fewer epochs
     history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=10,
     batch_size=32)

     # Evaluate the model on the test set
     test_loss, test_accuracy = model.evaluate(X_test, y_test)
     print(f"Test Loss: {test_loss}")
     print(f"Test Accuracy: {test_accuracy}")

     # Predict and generate classification report
     y_pred = (model.predict(X_test) > 0.5).astype("int32")
     print(classification_report(y_test, y_pred))
```

Output:

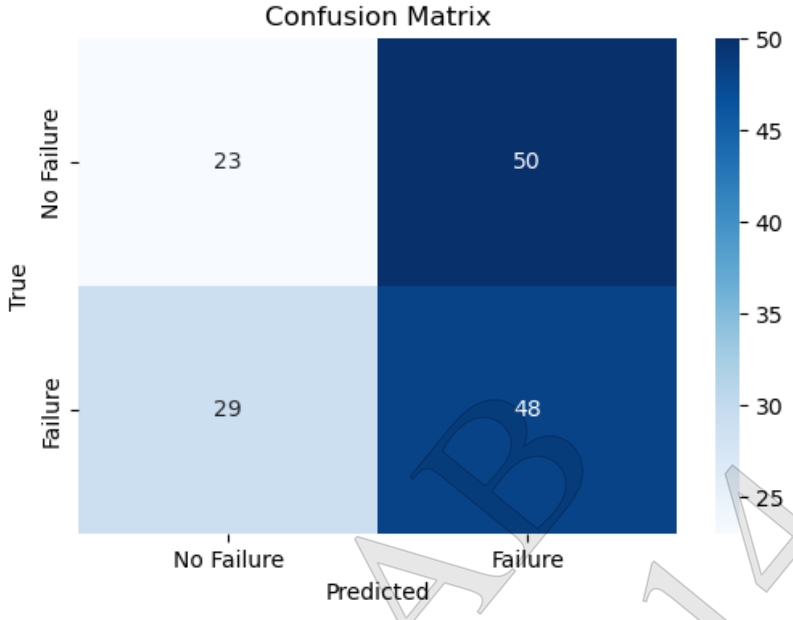
```
Epoch 1/10
23/23 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7976 - loss: 0.5117 -
val_accuracy: 0.5156 - val_loss: 0.7832
Epoch 2/10
23/23 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7784 - loss: 0.5235 -
val_accuracy: 0.5156 - val_loss: 0.7902
Epoch 3/10
23/23 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7891 - loss: 0.5068 -
val_accuracy: 0.5000 - val_loss: 0.7948
Epoch 4/10
23/23 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7687 - loss: 0.5194 -
val_accuracy: 0.4922 - val_loss: 0.8020
Epoch 5/10
23/23 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7619 - loss: 0.5247 -
val_accuracy: 0.5312 - val_loss: 0.8121
Epoch 6/10
23/23 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7967 - loss: 0.5169 -
val_accuracy: 0.5234 - val_loss: 0.8118
Epoch 7/10
23/23 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7831 - loss: 0.5106 -
val_accuracy: 0.5000 - val_loss: 0.8141
Epoch 8/10
23/23 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7767 - loss: 0.5078 -
val_accuracy: 0.5391 - val_loss: 0.8291
Epoch 9/10
23/23 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7946 - loss: 0.4805 -
val_accuracy: 0.4922 - val_loss: 0.8261
Epoch 10/10
23/23 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8199 - loss: 0.4780 -
val_accuracy: 0.5156 - val_loss: 0.8287
5/5 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4935 - loss: 0.7497
Test Loss: 0.7296823263168335
Test Accuracy: 0.5
5/5 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

	precision	recall	f1-score	support
0	0.48	0.44	0.46	73
1	0.51	0.56	0.53	77
accuracy			0.50	150
macro avg	0.50	0.50	0.50	150
weighted avg	0.50	0.50	0.50	150

Step 9:

```
[ ]  # Confusion Matrix
     cm = confusion_matrix(y_test, y_pred)
     plt.figure(figsize=(6, 4))
     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Failure',
     'Failure'], yticklabels=['No Failure', 'Failure'])
     plt.title('Confusion Matrix')
     plt.xlabel('Predicted')
     plt.ylabel('True')
     plt.show()
```

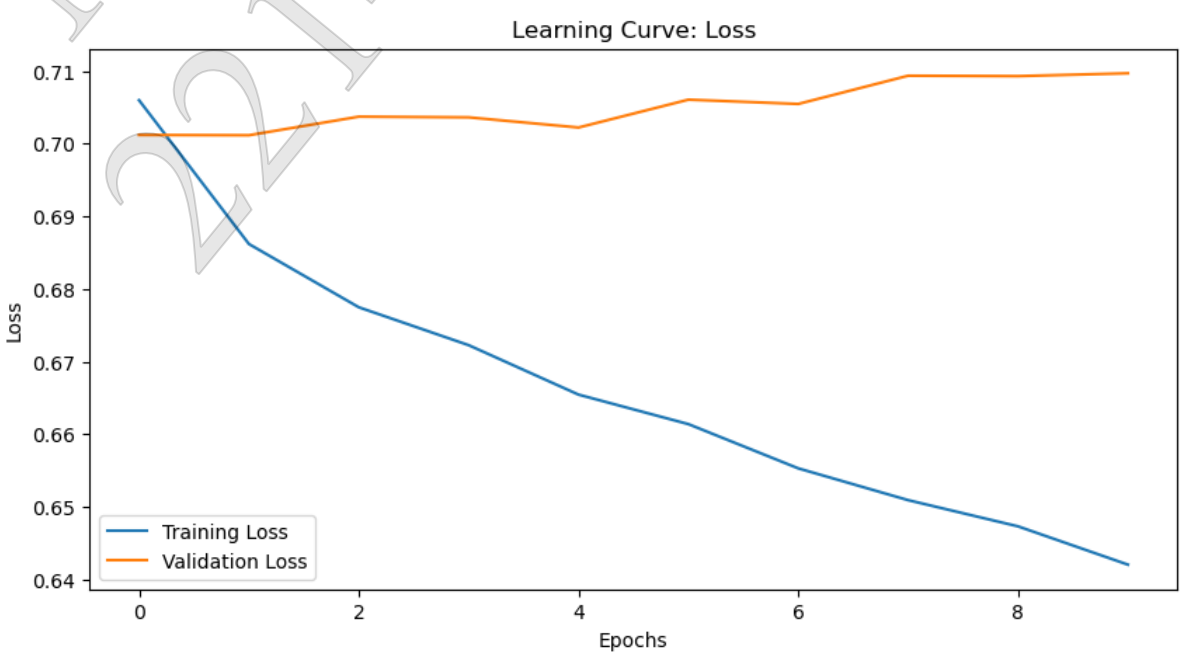
Output:



Step 10:

```
[ ]  # Plot learning curves for loss
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Learning Curve: Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Output:



Step 11:

```
[ ]  # Plot learning curves for accuracy
     plt.figure(figsize=(10, 5))
     plt.plot(history.history['accuracy'], label='Training Accuracy')
     plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
     plt.title('Learning Curve: Accuracy')
     plt.xlabel('Epochs')
     plt.ylabel('Accuracy')
     plt.legend()
     plt.show()
```

Output: