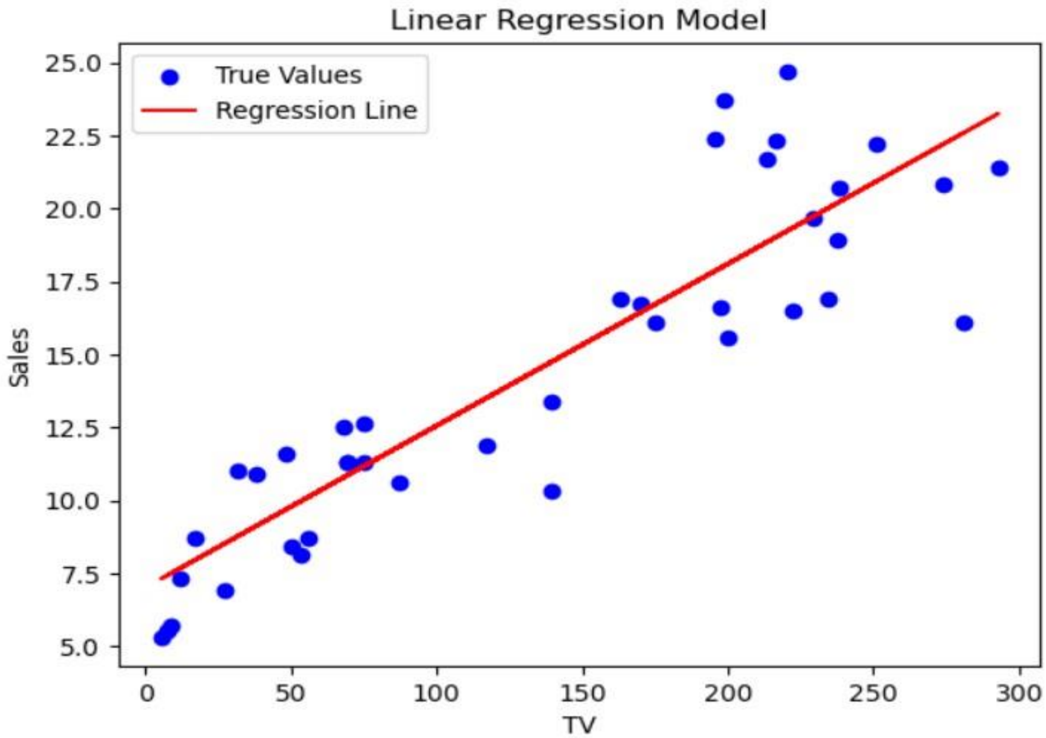
Assignment 1: Write a program to implementing and evaluating a Linear Regression model

```
# Import necessary libraries import numpy as np import
pandas as pd import matplotlib.pyplot as plt from
sklearn.linear_model import LinearRegression from
sklearn.model_selection import train_test_split from
sklearn.metrics import mean_squared_error, r2_score

# Load the dataset from a CSV file
data = pd.read_csv('Data science II/advertising.csv')
# Check the first few rows of the dataset to understand its structure print(data)

# Define the independent variable (feature) and dependent variable (target) X
= data[['TV']]  # Independent variable (1D array, needs to be 2D for sklearn) y
= data['Sales']    # Dependent variable (target)

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model's performance mse =
mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Output the evaluation metrics print(f"Mean
Squared Error (MSE): {mse}")
print(f"R-squared (R2) Score: {r2}")

# Visualize the results
plt.scatter(X_test, y_test, color='blue', label='True Values')
plt.plot(X_test, y_pred, color='red', label='Regression Line')
plt.xlabel('TV') plt.ylabel('Sales')
plt.title('Linear Regression Model') plt.legend()
plt.show()
```

Output:

```
Mean Squared Error (MSE): 6.101072906773964
R-squared (R2) Score: 0.802561303423698
```



Assignment 2: Write a program to implementing and evaluating a Logistic Regression model.

```
import numpy as np import
pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler from
sklearn.linear_model import LogisticRegression from
sklearn.metrics import accuracy_score,
classification_report, confusion_matrix

# Load dataset from CSV file
df = pd.read_csv('log.csv')

# Assuming the last column is the target variable
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# Split dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features scaler
= StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train Logistic Regression model model
= LogisticRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model accuracy =
accuracy_score(y_test, y_pred) conf_matrix =
confusion_matrix(y_test, y_pred) report =
classification_report(y_test, y_pred)
```

# Print evaluation results
print(f'Accuracy: {accuracy:.4f}')
print('Confusion Matrix:')
print(conf_matrix) print('Classification
Report:') print(report)

Output:

```
Accuracy: 0.4650
Confusion Matrix:
[[46 50]
 [57 47]]
Classification Report:
              precision    recall  f1-score   support

           0       0.45      0.48      0.46        96
           1       0.48      0.45      0.47       104

    accuracy                           0.47       200
   macro avg       0.47      0.47      0.46       200
weighted avg       0.47      0.47      0.47       200
```

Assignment 3: Write a program to implementing and evaluating a Decision Tree classifier.

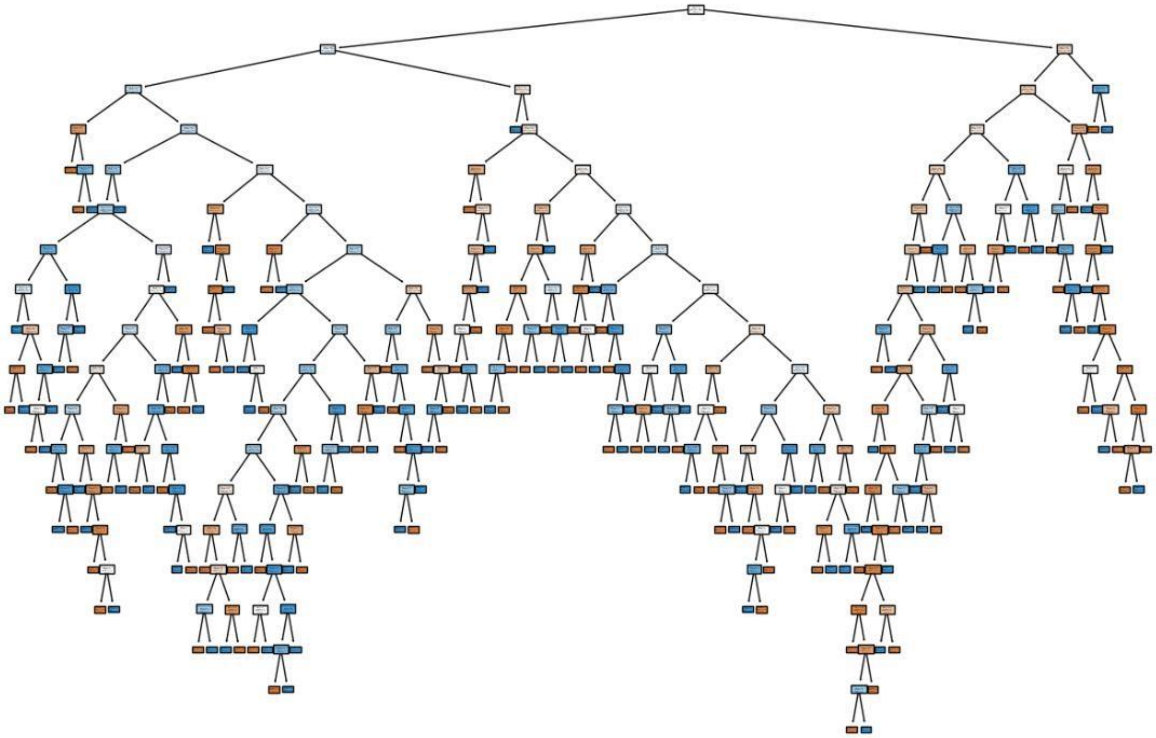
```
import numpy as np import
pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split from
sklearn.preprocessing import StandardScaler from
sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load dataset from CSV file df
= pd.read_csv('log.csv')

# Assuming the last column is the target variable
X = df.iloc[:, :-1] y
= df.iloc[:, -1]

# Split dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features (optional for Decision Tree, but can help with performance) scaler
= StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train Decision Tree model
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model accuracy =
accuracy_score(y_test, y_pred) conf_matrix =
confusion_matrix(y_test, y_pred) report =
classification_report(y_test, y_pred)

# Print evaluation results
print(f'Accuracy: {accuracy:.4f}')
print('Confusion Matrix:')
```

print(conf_matrix) print('Classification
Report:') print(report)

Visualize the Decision Tree plt.figure(figsize=(15,
10))
plot_tree(model, filled=True, feature_names=df.columns[:-1], class_names=str(np.unique(y)))
plt.show()

Output:

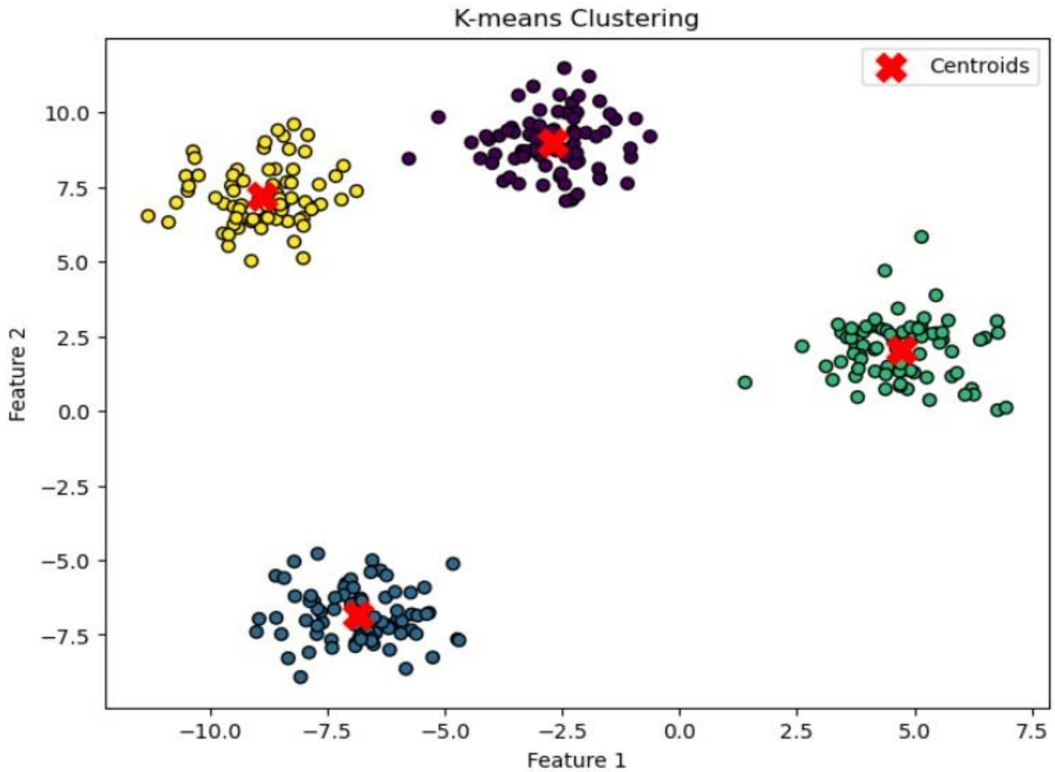
```
Accuracy: 0.4650
Confusion Matrix:
[[45 51]
 [56 48]]
Classification Report:
              precision    recall  f1-score   support

           0       0.45      0.47      0.46        96
           1       0.48      0.46      0.47       104

    accuracy                           0.47       200
   macro avg       0.47      0.47      0.46       200
weighted avg       0.47      0.47      0.47       200
```



Assignment 4: Write a program to implementing Clustering using the K-means algorithm

```
import numpy as np import
matplotlib.pyplot as plt from
sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Step 1: Generate synthetic data (for demonstration)
# Generating 300 data points with 4 centers (clusters)
X, y = make_blobs(n_samples=300, centers=4, random_state=42)

# Step 2: Apply the K-means algorithm
# Set the number of clusters to 4 (since we generated data with 4 centers)
kmeans = KMeans(n_clusters=4, random_state=42) kmeans.fit(X)

# Step 3: Get the centroids and labels for the clusters
centroids = kmeans.cluster_centers_ labels
= kmeans.labels_

# Step 4: Visualize the clusters plt.figure(figsize=(8,
6))

# Scatter plot of data points with colors corresponding to cluster labels plt.scatter(X[:,
0], X[:, 1], c=labels, cmap='viridis', marker='o', edgecolor='k')

# Mark the centroids with a red 'X'
plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', s=200, c='red', label='Centroids')

# Add titles and labels plt.title('K-
means Clustering') plt.xlabel('Feature
1') plt.ylabel('Feature 2')

# Show the legend
plt.legend()

# Display the plot plt.show()
```

Output:

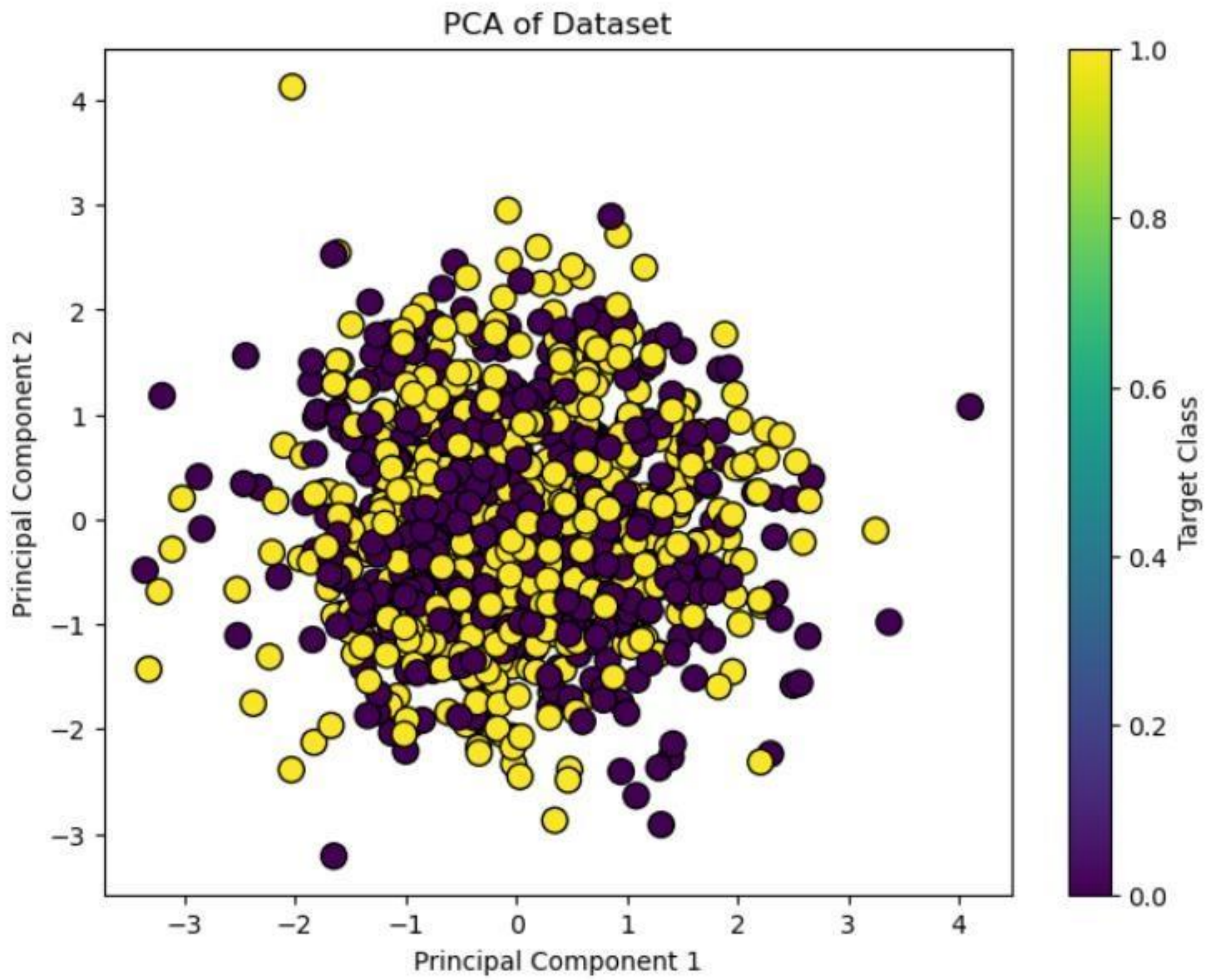

K-means Clustering
Centroids
Feature 2
Feature 1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5

Assignment 5: Write a program to implementing Dimensionality reduction using PCA.

```
# Import necessary libraries import numpy as np
import pandas as pd import matplotlib.pyplot as plt
from sklearn.decomposition import PCA from
sklearn.preprocessing import StandardScaler from
sklearn.model_selection import train_test_split

# Load CSV file (replace 'your_dataset.csv' with your actual dataset file path) df
= pd.read_csv('log.csv')

# Check the first few rows of the dataset print(df.head())

# Step 1: Separate features (X) and target (y) if applicable
# Assuming the last column is the target variable
X = df.iloc[:, :-1].values  # All rows, all columns except the last one y
= df.iloc[:, -1].values   # Last column is the target

# Step 2: Standardize the dataset (important for PCA)
scaler = StandardScaler() X_scaled
= scaler.fit_transform(X)

# Step 3: Apply PCA to reduce to 2 dimensions for visualization pca =
PCA(n_components=2)  # Reduce to 2 components for visualization
X_pca = pca.fit_transform(X_scaled)

# Step 4: Explained variance ratio (how much variance is captured by each component)
print("Explained variance ratio:", pca.explained_variance_ratio_)

# Step 5: Plot the 2D PCA result plt.figure(figsize=(8,6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolor='k', s=100)
plt.title("PCA of Dataset") plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2") plt.colorbar(label='Target Class')
plt.show()
```

Output:

```
   feature_0  feature_1  feature_2  feature_3  feature_4  feature_5  \
0   0.496714  -0.138264   0.647689   1.523030  -0.234153  -0.234137
1  -0.463418  -0.465730   0.241962  -1.913280  -1.724918  -0.562288
2   1.465649  -0.225776   0.067528  -1.424748  -0.544383   0.110923
3  -0.601707   1.852278  -0.013497  -1.057711   0.822545  -1.220844
4   0.738467   0.171368  -0.115648  -0.301104  -1.478522  -0.719844

   feature_6  feature_7  feature_8  feature_9  target
0   1.579213   0.767435  -0.469474   0.542560       1
1  -1.012831   0.314247  -0.908024  -1.412304       1
2  -1.150994   0.375698  -0.600639  -0.291694       0
3   0.208864  -1.959670  -1.328186   0.196861       0
4  -0.460639   1.057122   0.343618  -1.763040       0
Explained variance ratio: [0.11348263 0.10726962]
```



Assignment 6: Write a program to implementing Bagging using Random Forest.

Import necessary libraries import pandas as pd from
sklearn.model_selection import train_test_split from
sklearn.ensemble import RandomForestClassifier from
sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

Load CSV file (replace 'your_dataset.csv' with your actual dataset file path) df
= pd.read_csv('log.csv')

Check the first few rows of the dataset print(df.head())

Step 1: Handle missing values (if any)
Example: Drop rows with missing values (you can also fill with the mean or median) df
= df.dropna()

# Step 2: Separate features (X) and target (y)
# Assuming the last column is the target variable
X = df.iloc[:, :-1].values # All rows, all columns except the last one (features) y
= df.iloc[:, -1].values   # Last column is the target

# Step 3: Encode the target variable if it's categorical
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

Step 4: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 5: Initialize and train the Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)

# Step 6: Make predictions on the test set
y_pred = rf_classifier.predict(X_test)

Step 7: Evaluate the model's performance accuracy
= accuracy_score(y_test, y_pred) print(f'Accuracy of
Random Forest on test data: {accuracy * 100:.2f}%')

Output :

```
   feature_0  feature_1  feature_2  feature_3  feature_4  feature_5  \
0   0.496714  -0.138264   0.647689   1.523030  -0.234153  -0.234137   
1  -0.463418  -0.465730   0.241962  -1.913280  -1.724918  -0.562288   
2   1.465649  -0.225776   0.067528  -1.424748  -0.544383   0.110923   
3  -0.601707   1.852278  -0.013497  -1.057711   0.822545  -1.220844   
4   0.738467   0.171368  -0.115648  -0.301104  -1.478522  -0.719844   

   feature_6  feature_7  feature_8  feature_9  target  
0   1.579213   0.767435  -0.469474   0.542560       1  
1  -1.012831   0.314247  -0.908024  -1.412304       1  
2  -1.150994   0.375698  -0.600639  -0.291694       0  
3   0.208864  -1.959670  -1.328186   0.196861       0  
4  -0.460639   1.057122   0.343618  -1.763040       0  
Accuracy of Random Forest on test data: 53.00%
```

Assignment 7: Write a program to implementing Boosting using AdaBoost

```
# Import necessary libraries import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

# Load CSV file (replace 'your_dataset.csv' with your actual dataset file path) df
= pd.read_csv('log.csv')

# Check the first few rows of the dataset print(df.head())


# Step 1: Handle missing values (if any)
# Example: Drop rows with missing values (you can also fill with the mean or median) df
= df.dropna()

# Step 2: Separate features (X) and target (y)
# Assuming the last column is the target variable
X = df.iloc[:, :-1].values  # All rows, all columns except the last one (features) y
= df.iloc[:, -1].values   # Last column is the target

# Step 3: Encode the target variable if it's categorical
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Step 4: Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 5: Initialize and train the AdaBoost classifier with a DecisionTree as the base estimator #
DecisionTreeClassifier with max_depth=1 is used to create a weak learner (stump)
base_estimator = DecisionTreeClassifier(max_depth=1)

# Update: Use 'estimator' instead of 'base_estimator'
adaboost_classifier = AdaBoostClassifier(estimator=base_estimator, n_estimators=50,
random_state=42)
adaboost_classifier.fit(X_train, y_train)

# Step 6: Make predictions on the test set
```

```
y_pred = adaboost_classifier.predict(X_test)

# Step 7: Evaluate the model's performance accuracy
= accuracy_score(y_test, y_pred)
print(f'Accuracy of AdaBoost on test data: {accuracy * 100:.2f}%')
```

Output:

```
   feature_0  feature_1  feature_2  feature_3  feature_4  feature_5  \
0   0.496714  -0.138264   0.647689   1.523030  -0.234153  -0.234137
1  -0.463418  -0.465730   0.241962  -1.913280  -1.724918  -0.562288
2   1.465649  -0.225776   0.067528  -1.424748  -0.544383   0.110923
3  -0.601707   1.852278  -0.013497  -1.057711   0.822545  -1.220844
4   0.738467   0.171368  -0.115648  -0.301104  -1.478522  -0.719844

   feature_6  feature_7  feature_8  feature_9  target
0   1.579213   0.767435  -0.469474   0.542560       1
1  -1.012831   0.314247  -0.908024  -1.412304       1
2  -1.150994   0.375698  -0.600639  -0.291694       0
3   0.208864  -1.959670  -1.328186   0.196861       0
4  -0.460639   1.057122   0.343618  -1.763040       0
Accuracy of AdaBoost on test data: 49.00%
```

Assignment 8: Write a program to implementing SVM for classification tasks.

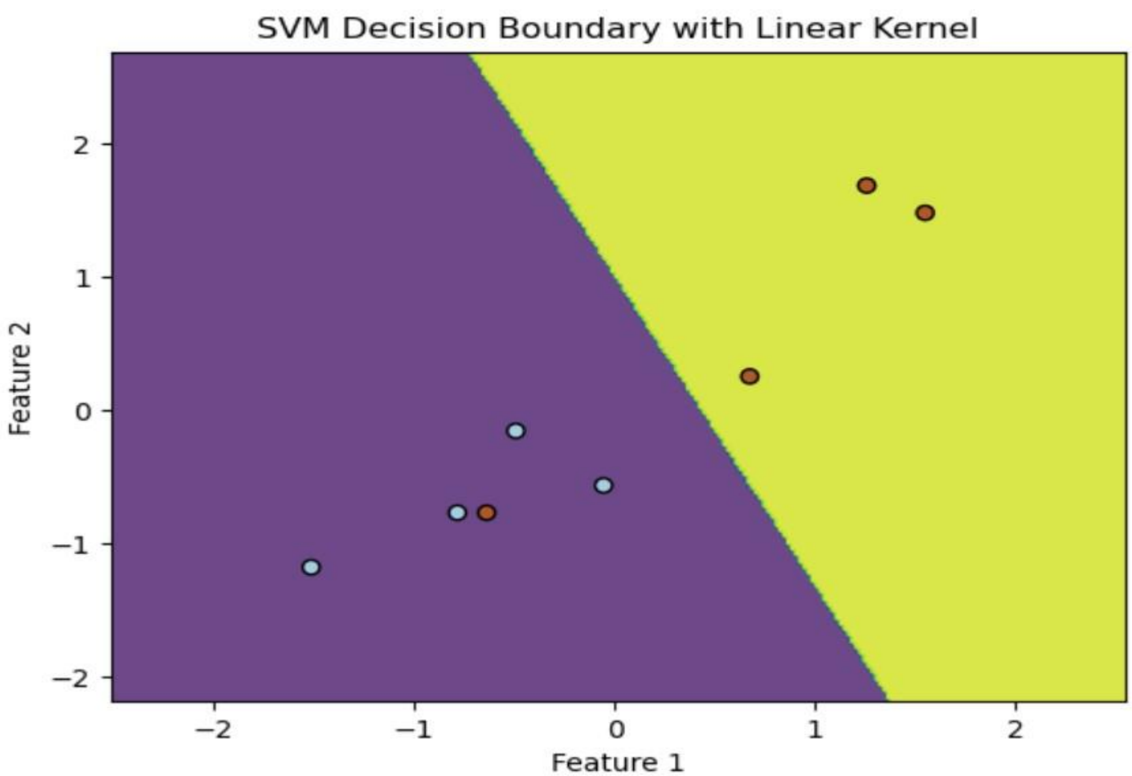
```
import numpy as np import pandas as pd import
matplotlib.pyplot as plt from
sklearn.model_selection import train_test_split from
sklearn.preprocessing import StandardScaler from
sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load dataset from CSV file (replace 'your_dataset.csv' with your actual file path) df
= pd.read_csv('pca.csv')

# Check the first few rows of the dataset print(df.head())

# Assuming the last column is the target variable (classification labels)
X = df.iloc[:, :-1]  # Select all rows and all columns except the last one for features y
= df.iloc[:, -1]   # Select the last column for the target (labels)

# Step 1: Split the dataset into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 2: Standardize features using StandardScaler (important for SVM) scaler
= StandardScaler()
X_train = scaler.fit_transform(X_train)  # Fit on training data and transform it
X_test = scaler.transform(X_test)        # Use the same scaler to transform test data

# Step 3: Train the Support Vector Machine (SVM) model
svm_model = SVC(kernel='linear', random_state=42)  # Using a linear kernel for simplicity
svm_model.fit(X_train, y_train)  # Train the model on the training set

# Step 4: Make predictions using the trained SVM model y_pred
= svm_model.predict(X_test)

# Step 5: Evaluate the model's performance accuracy
= accuracy_score(y_test, y_pred) conf_matrix =
confusion_matrix(y_test, y_pred) report =
classification_report(y_test, y_pred)

# Print evaluation results
print(f'Accuracy: {accuracy:.4f}')
print('Confusion Matrix:')
```

```
print(conf_matrix) print('Classification
Report:') print(report)

# Step 6: Optional - Visualize the decision boundaries (only works for 2D features)
# This is just a visualization example for datasets with two features if
X.shape[1] == 2:  # Check if we have only two features for visualization
    # Create a mesh grid for plotting decision boundaries
h = .02
    x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    # Predict over the mesh grid
    Z = svm_model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot decision boundary
plt.contourf(xx, yy, Z, alpha=0.8)
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, edgecolors='k', marker='o',
cmap=plt.cm.Paired)
    plt.title('SVM Decision Boundary with Linear Kernel')
plt.xlabel('Feature 1')     plt.ylabel('Feature 2')
plt.show()
```

Output:

	Feature1	Feature2	Target
0	2.5	3.1	0
1	1.2	2.3	0
2	3.4	4.2	1
3	2.1	3.0	1
4	3.0	3.5	1

```
Accuracy: 1.0000
Confusion Matrix:
[[1 0]
 [0 1]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         1

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2
```



Assignment 9: Write a program to implementing a simple neural network using TensorFlow/Keras.

```
import pandas as pd import
numpy as np import
tensorflow as tf import
tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input import
matplotlib.pyplot as plt

df = pd.read_csv('diabetes.csv') df.head()

print ('Number of Rows :', df.shape[0]) print ('Number of
Columns :', df.shape[1]) print ('Number of Patients with
outcome 1 :', df.Outcome.sum()) print ('Event Rate :',
round(df.Outcome.mean()*100,2) ,'%') df.describe()

from sklearn.model_selection import train_test_split
X = df.to_numpy()[:,0:8]  Y
= df.to_numpy()[:,8]
seed = 42
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.25, random_state = seed)
print (f'Shape of Train Data : {X_train.shape}') print (f'Shape of Test Data : {X_test.shape}')

model = Sequential([
   Input(shape=(8,)),  # Define the input shape using the new `shape` argument
   Dense(24, activation='relu'),
   Dense(12, activation='relu'),
   Dense(1, activation='sigmoid'),
])

# Compile the model (optional, but necessary for training)
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

# Summary of the model
#model.summary() model.summary()
history = model.fit(X_train, y_train, epochs=150, batch_size=32, verbose = 1)

# Plotting loss
```

plt.plot(history.history['loss'])
plt.title('Binary Cross Entropy Loss on Train dataset')
plt.ylabel('loss') plt.xlabel('epoch')
plt.show()

# Plotting accuracy metric
plt.plot(history.history['accuracy'])
plt.title('Accuracy on the train dataset')
plt.ylabel('accuracy') plt.xlabel('epoch')
plt.show()

Output:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
Number of Rows : 768
Number of Columns : 9
Number of Patients with outcome 1 : 268
Event Rate : 34.9 %
Shape of Train Data : (576, 8)
Shape of Test Data : (192, 8)
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
dense_18 (Dense)	(None, 24)	216
dense_19 (Dense)	(None, 12)	300
dense_20 (Dense)	(None, 1)	13

Total params: 529 (2.07 KB)
Trainable params: 529 (2.07 KB)
Non-trainable params: 0 (0.00 B)
Epoch 1/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5895 - loss: 4.9180

Epoch 2/150

18/18 - - loss:

- - loss:

- - loss:
0s 3ms/step accuracy: 0.5057
1.5014
Epoch 3/150
18/18 0s 3ms/step accuracy: 0.6015
1.1479
Epoch 4/150
18/18 0s 3ms/step accuracy: 0.6004
1.0280
Epoch 5/150
18/18 0s 2ms/step - accuracy: 0.6393 - loss:
0.8963
Epoch 6/150
18/18 0s 2ms/step - accuracy: 0.6242 - loss:
0.8435
Epoch 7/150
18/18 0s 2ms/step - accuracy: 0.6523 - loss:
0.7196
Epoch 8/150
18/18 0s 2ms/step - accuracy: 0.6367 - loss:
0.7654
Epoch 9/150
18/18 0s 2ms/step - accuracy: 0.6541 - loss:
0.7686
Epoch 10/150
18/18 0s 2ms/step - accuracy: 0.6621 - loss:
0.6817
Epoch 11/150

0s 2ms/step - - loss:

18/18 0s 2ms/step - - loss:

- - loss:

- - loss:
18/18 0s 5ms/step - accuracy: 0.6425 - loss:
0.7043
Epoch 12/150
18/18 0s 2ms/step - accuracy: 0.7117 - loss:
0.6386
Epoch 13/150
18/18 0s 3ms/step - accuracy: 0.6487 - loss:
0.6587
Epoch 14/150
18/18 accuracy: 0.6393
0.6852
Epoch 15/150
0s 2ms/step accuracy: 0.6932
0.6562
Epoch 16/150
18/18 0s 2ms/step accuracy: 0.6278
0.6890
Epoch 17/150
18/18 0s 2ms/step accuracy: 0.6813
0.6323
Epoch 18/150
18/18 0s 3ms/step - accuracy: 0.6853 - loss:
0.6206
Epoch 19/150
18/18 0s 2ms/step - accuracy: 0.7143 - loss:
0.5713
Epoch 20/150

0s 2ms/step - - loss:

18/18 ━━━━━━━━━━━━━━━━━━━━ - - loss:

━━━━━━━━━━━━━━━━━━━━ - - loss:

━━━━━━━━━━━━━━━━━━━━ - - loss:
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6995 - loss: 0.5817
Epoch 21/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6969 - loss: 0.6234
Epoch 22/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7152 - loss: 0.5890
Epoch 23/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7173 - loss: 0.5726
Epoch 24/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6975 - loss: 0.5961
Epoch 25/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6997 - loss: 0.6367
Epoch 26/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7037 - loss: 0.6078
Epoch 27/150
18/18 accuracy: 0.7107
0.5834
Epoch 28/150
accuracy: 0.7045
0.5672
Epoch 29/150

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

━━━━━━━━━━━━━━━━━━━━ - - loss:

━━━━━━━━━━━━━━━━━━━━ - - loss:
18/18 0s 2ms/step accuracy: 0.7128
0.5608
Epoch 30/150
18/18 0s 3ms/step accuracy: 0.7039
0.5908
Epoch 31/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7340 - loss:
0.5645
Epoch 32/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7118 - loss:
0.5859
Epoch 33/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7159 - loss:
0.5662
Epoch 34/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6892 - loss:
0.5822
Epoch 35/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7272 - loss:
0.5440
Epoch 36/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7180 - loss:
0.5606
Epoch 37/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7144 - loss:
0.5587
Epoch 38/150

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

18/18 ━━━━━━━━━━━━━━━━━━━━ - - loss:

━━━━━━━━━━━━━━━━━━━━ - - loss:

━━━━━━━━━━━━━━━━━━━━ - - loss:
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7395 - loss:
0.5620
Epoch 39/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7001 - loss:
0.5776
Epoch 40/150
18/18 accuracy: 0.6902
0.5962
Epoch 41/150
0s 3ms/step accuracy: 0.7072
0.5891
Epoch 42/150
18/18 0s 3ms/step accuracy: 0.7389
0.5666
Epoch 43/150
18/18 0s 3ms/step accuracy: 0.6933
0.6019
Epoch 44/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7406 - loss:
0.5502
Epoch 45/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7194 - loss:
0.5679
Epoch 46/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7061 - loss:
0.5431
Epoch 47/150

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

━━━━━━━━━━━━━━━━━━━━ - - loss:

━━━━━━━━━━━━━━━━━━━━ - - loss:

18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7438 - loss:
0.5202
Epoch 48/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7346 - loss:
0.5445
Epoch 49/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6963 - loss:
0.5831
Epoch 50/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7246 - loss:
0.5752
Epoch 51/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7290 - loss:
0.5586
Epoch 52/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6933 - loss:
0.5897
Epoch 53/150
18/18 accuracy: 0.7588
0.5280
Epoch 54/150
accuracy: 0.7153
0.5653
Epoch 55/150
18/18 0s 2ms/step accuracy: 0.7419
0.5464
Epoch 56/150

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

18/18 ━━━━━━━━━━━━━━━━━━━━ - - loss:

━━━━━━━━━━━━━━━━━━━━ - - loss:

━━━━━━━━━━━━━━━━━━━━ - - loss:
18/18 0s 2ms/step accuracy: 0.7117
0.5628
Epoch 57/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7576 - loss:
0.5011
Epoch 58/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7324 - loss:
0.5320
Epoch 59/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7703 - loss:
0.4985
Epoch 60/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7558 - loss:
0.5444
Epoch 61/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7404 - loss:
0.5359
Epoch 62/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7296 - loss:
0.5656
Epoch 63/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7643 - loss:
0.5179
Epoch 64/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7075 - loss:
0.5770
Epoch 65/150

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

━━━━━━━━━━━━━━━━━━━━ - - loss:

━━━━━━━━━━━━━━━━━━━━ - - loss:
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7425 - loss:
0.5180
Epoch 66/150
18/18 accuracy: 0.7389
0.5451
Epoch 67/150

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

18/18 0s 2ms/step - - loss:

0s 2ms/step - - loss:

- - loss:
accuracy: 0.7635
0.5084
Epoch 68/150
18/18 accuracy: 0.7268
0.5460
Epoch 69/150
18/18 0s 2ms/step accuracy: 0.7467
0.5410
Epoch 70/150
18/18 0s 2ms/step - accuracy: 0.7459 - loss:
0.5315
Epoch 71/150
18/18 0s 2ms/step - accuracy: 0.7889 - loss:
0.5101
Epoch 72/150
18/18 0s 2ms/step - accuracy: 0.7333 - loss:
0.5205
Epoch 73/150
18/18 0s 2ms/step - accuracy: 0.7744 - loss:
0.5036
Epoch 74/150
18/18 0s 2ms/step - accuracy: 0.7172 - loss:
0.5641
Epoch 75/150
18/18 0s 2ms/step - accuracy: 0.7345 - loss:
0.5297
Epoch 76/150

0s 2ms/step - - loss:

18/18 0s 2ms/step - - loss:

\- - loss:

\- - loss:
18/18 0s 2ms/step - accuracy: 0.7540 - loss:
0.5003
Epoch 77/150
18/18 0s 2ms/step - accuracy: 0.7523 - loss:
0.5178
Epoch 78/150
18/18 0s 2ms/step - accuracy: 0.7509 - loss:
0.5055
Epoch 79/150
18/18 accuracy: 0.7389
0.5510
Epoch 80/150
accuracy: 0.7464
0.5330
Epoch 81/150
18/18 0s 4ms/step accuracy: 0.7178
0.5377
Epoch 82/150
18/18 0s 2ms/step accuracy: 0.7065
0.5838
Epoch 83/150
18/18 0s 2ms/step - accuracy: 0.7326 - loss:
0.5747
Epoch 84/150
18/18 0s 2ms/step - accuracy: 0.7411 - loss:
0.5718
Epoch 85/150

0s 2ms/step - - loss:

18/18 0s 2ms/step - - loss:

0s 2ms/step - - loss:

- - loss:
18/18 0s 2ms/step - accuracy: 0.7313 - loss:
0.6302
Epoch 86/150
18/18 0s 2ms/step - accuracy: 0.7352 - loss:
0.5402
Epoch 87/150
18/18 0s 2ms/step - accuracy: 0.7502 - loss:
0.5175
Epoch 88/150
18/18 0s 2ms/step - accuracy: 0.7232 - loss:
0.5524
Epoch 89/150
18/18 0s 2ms/step - accuracy: 0.7249 - loss:
0.5583
Epoch 90/150
18/18 0s 2ms/step - accuracy: 0.7564 - loss:
0.5251
Epoch 91/150
18/18 0s 2ms/step - accuracy: 0.7525 - loss:
0.4848
Epoch 92/150
18/18 accuracy: 0.7200
0.5333
Epoch 93/150
accuracy: 0.7700
0.4992
Epoch 94/150

0s 2ms/step - - loss:

18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

━━━━━━━━━━━━━━━━━━━━ - - loss:

━━━━━━━━━━━━━━━━━━━━ - - loss:
18/18 accuracy: 0.7563
0.5124
Epoch 95/150
18/18 0s 5ms/step accuracy: 0.7918
0.4565
Epoch 96/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7253 - loss:
0.5327
Epoch 97/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7681 - loss:
0.4731
Epoch 98/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7563 - loss:
0.5068
Epoch 99/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7770 - loss:
0.4929
Epoch 100/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7342 - loss:
0.5462
Epoch 101/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7210 - loss:
0.5485
Epoch 102/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7474 - loss:
0.5126
Epoch 103/150

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

18/18 0s 2ms/step - - loss:

0s 2ms/step - - loss:

\- - loss:
18/18 0s 2ms/step - accuracy: 0.7680 - loss:
0.5199
Epoch 104/150
18/18 0s 2ms/step - accuracy: 0.7530 - loss:
0.5064
Epoch 105/150
18/18 accuracy: 0.7746
0.5026
Epoch 106/150

0s 2ms/step - - loss:

18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:
accuracy: 0.7992
0.4694
Epoch 107/150
18/18 accuracy: 0.7494
0.5140
Epoch 108/150
18/18 accuracy: 0.7442
0.5407
Epoch 109/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7394 - loss:
0.5266
Epoch 110/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7719 - loss:
0.4795
Epoch 111/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7351 - loss:
0.5200
Epoch 112/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7625 - loss:
0.4970
Epoch 113/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7815 - loss:
0.4912
Epoch 114/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7550 - loss:
0.4981
Epoch 115/150

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7752 - loss:
0.4759
Epoch 116/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7665 - loss:
0.4901
Epoch 117/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7712 - loss:
0.4704
Epoch 118/150
18/18 accuracy: 0.7220
0.5439
Epoch 119/150
accuracy: 0.7663
0.4747
Epoch 120/150
18/18 accuracy: 0.7419
0.5171
Epoch 121/150
18/18 accuracy: 0.7474
0.5315
Epoch 122/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7596 - loss:
0.5100
Epoch 123/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7611 - loss:
0.4954
Epoch 124/150

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7780 - loss: 0.4855
Epoch 125/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7708 - loss: 0.4956
Epoch 126/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7892 - loss: 0.4946
Epoch 127/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7564 - loss: 0.4991
Epoch 128/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7740 - loss: 0.5154
Epoch 129/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7924 - loss: 0.4877
Epoch 130/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7559 - loss: 0.4971
Epoch 131/150
18/18 accuracy: 0.7745
0.4818
Epoch 132/150
accuracy: 0.7583
0.4932
Epoch 133/150

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:
18/18 accuracy: 0.7677
0.4927
Epoch 134/150
18/18 accuracy: 0.7872
0.4741
Epoch 135/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7663 - loss:
0.4600
Epoch 136/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7785 - loss:
0.4928
Epoch 137/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7636 - loss:
0.5001
Epoch 138/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7424 - loss:
0.5203
Epoch 139/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7641 - loss:
0.4943
Epoch 140/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7559 - loss:
0.4887
Epoch 141/150
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7780 - loss:
0.4835
Epoch 142/150

━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - - loss:

18/18 0s 2ms/step - - loss:

0s 2ms/step - - loss:

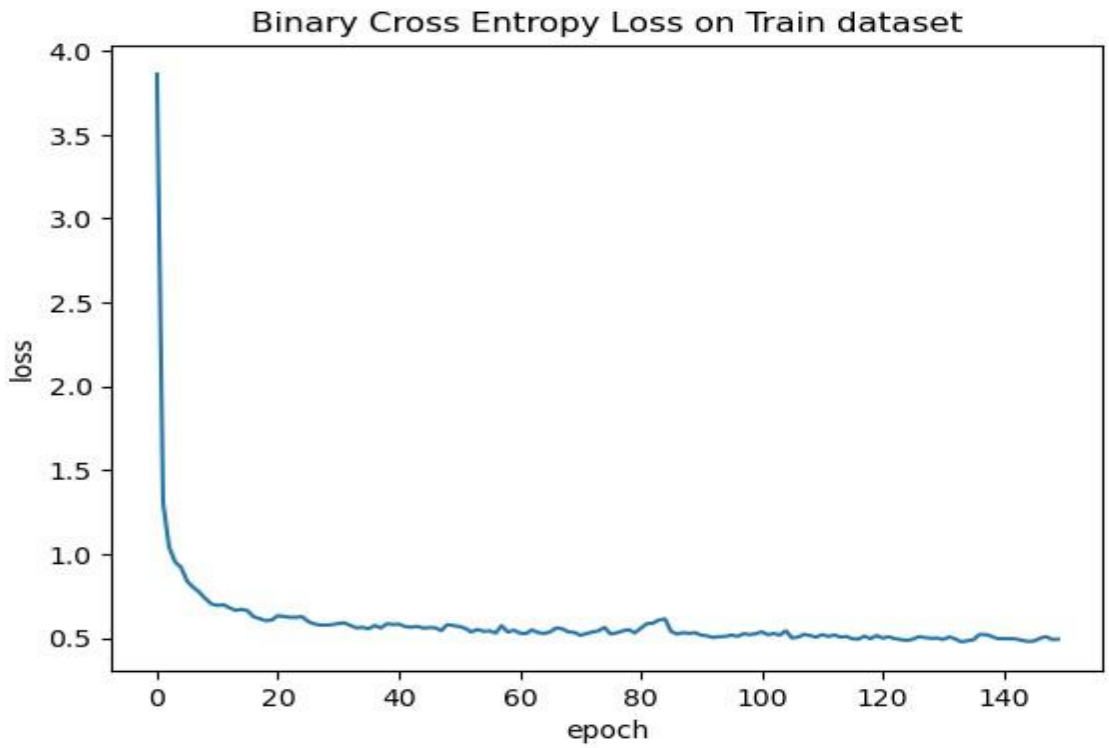
0s 2ms/step - - loss:
18/18 0s 2ms/step - accuracy: 0.7715 - loss:
0.5056
Epoch 143/150
18/18 0s 2ms/step - accuracy: 0.7542 - loss:
0.5170
Epoch 144/150
18/18 accuracy: 0.7890
0.4876
Epoch 145/150

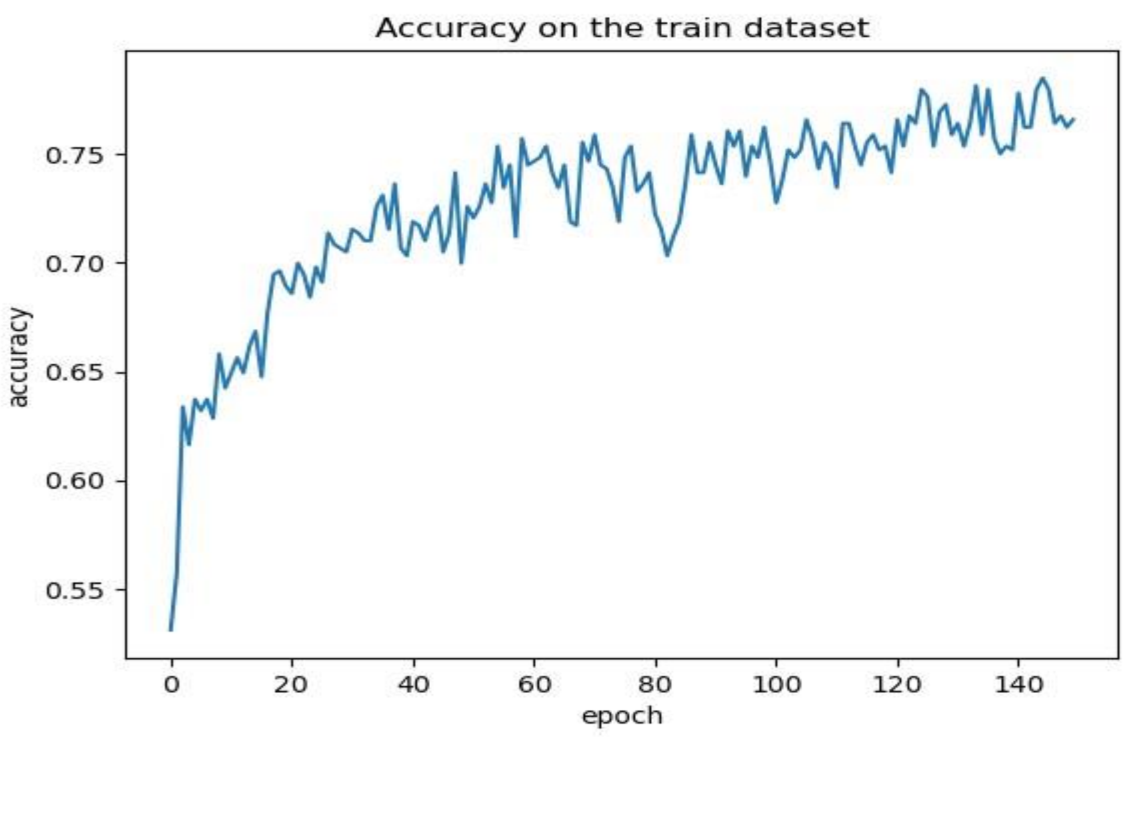
0s 2ms/step - - loss:

18/18 0s 2ms/step - - loss:

0s 2ms/step - - loss:

0s 2ms/step - - loss:
accuracy: 0.8027
0.4567
Epoch 146/150
18/18 accuracy: 0.7647
0.5335
Epoch 147/150
18/18 accuracy: 0.7290
0.5359
Epoch 148/150
18/18 0s 3ms/step - accuracy: 0.7641 - loss:
0.5116
Epoch 149/150
18/18 0s 2ms/step - accuracy: 0.7540 - loss:
0.5059
Epoch 150/150
18/18 0s 2ms/step - accuracy: 0.7521 - loss:
0.5285


Binary Cross Entropy Loss on Train dataset
loss
epoch
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
20
40
60
80
100
120
140


Accuracy on the train dataset
accuracy
epoch
0.75
0.70
0.65
0.60
0.55
0
20
40
60
80
100
120
140

Assignment 10: Write a program to implementing with big data concepts using sample datasets & Setting up a Hadoop environment.

```
# Install Java
!sudo apt update
!sudo apt install openjdk-8-jdk

# Download and extract Hadoop
!wget http://apache.mirrors.lucidnetworks.net/hadoop/common/hadoop-3.3.1/hadoop-3.3.1.tar.gz
!tar -xzvf hadoop-3.3.1.tar.gz
!mv hadoop-3.3.1 /usr/local/hadoop

# Sample dataset (you can imagine it as a text file with large data) dataset
= """
Hadoop is a framework for processing large datasets.
It is used for distributed storage and distributed computing.
Hadoop is part of the Big Data ecosystem.
Hadoop helps process Big Data.
"""
```

```
# Save dataset to a file (simulating a big text file) with
open('/content/dataset.txt', 'w') as f:     f.write(dataset)

from pyspark.sql import SparkSession

# Initialize Spark session
spark = SparkSession.builder.appName('WordCount').getOrCreate()

# Load the dataset into an RDD (Resilient Distributed Dataset) rdd
= spark.sparkContext.textFile('/content/dataset.txt')

# Perform word count word_counts =
rdd.flatMap(lambda line: line.split()) \
                .map(lambda word: (word.lower(), 1)) \
.reduceByKey(lambda x, y: x + y)

# Collect and print the results for word,
count in word_counts.collect():
print(f'{word}: {count}')

# Stop the Spark session spark.stop()
```

Output:

```
hadoop: 3
framework: 1
for: 2
large: 1
it: 1
used: 1
distributed: 2
storage: 1
and: 1
part: 1
of: 1
big: 2
ecosystem.: 1
helps: 1
data.: 1
is: 3
a: 1
processing: 1
datasets.: 1
computing.: 1
the: 1
data: 1
process: 1
```

Assignment 11: Write a program to implementing CRUD operations in MongoDB

pip install pymongo

from pymongo import MongoClient

Connect to MongoDB server (default localhost:27017) client
= MongoClient("mongodb://localhost:27017/")

Use the 'mydatabase' database and 'users' collection db
= client['mydatabase']
collection = db['users']

# CREATE operation: Insert a document
user_data = {
'name': 'John Doe',
    'age': 30,
    'email': 'john.doe@example.com'
}
result = collection.insert_one(user_data)
print(f"Document inserted with ID: {result.inserted_id}")

READ operation: Find a single document by name user
= collection.find_one({"name": "John Doe"})
print("Found user:", user)

UPDATE operation: Update the user's age update_result
= collection.update_one(
    {"name": "John Doe"},
    {"$set": {"age": 31}}
)
print(f"Documents matched: {update_result.matched_count}, Documents modified:
{update_result.modified_count}")

# DELETE operation: Delete a user by name
delete_result = collection.delete_one({"name": "John Doe"}) print(f"Documents
deleted: {delete_result.deleted_count}") **Output:**

```
Collecting pymongo
  Downloading pymongo-4.11.2-cp312-cp312-win_amd64.whl.metadata (22 kB)
Collecting dnspython<3.0.0,>=1.16.0 (from pymongo)
  Downloading dnspython-2.7.0-py3-none-any.whl.metadata (5.8 kB)
Downloading pymongo-4.11.2-cp312-cp312-win_amd64.whl (882 kB)
   ---------------------------------------- 0.0/882.2 kB ? eta -:--:--
   ---------------------------------------- 882.2/882.2 kB 19.4 MB/s eta 0:00:00
Downloading dnspython-2.7.0-py3-none-any.whl (313 kB)
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.7.0 pymongo-4.11.2
Note: you may need to restart the kernel to use updated packages.
```

Document inserted with ID: 67d270cb7c6068e82f03a444
Found user: {'_id': ObjectId('67d270cb7c6068e82f03a444'), 'name': 'John Doe', 'age': 30, 'email': 'john.doe@example.com'}
Documents matched: 1, Documents modified: 1
Documents deleted: 1

Assignment 12: Write a program to implementing with NLTK: Tokenization, stemming, and lemmatization

```
pip install nltk import
nltk

# Download the 'punkt' tokenizer nltk.download('punkt')

import nltk

# Download the 'punkt_tab' resource, which is required for tokenization
nltk.download('punkt_tab')

# Download other necessary resources for lemmatization and stop words
nltk.download('wordnet')   nltk.download('stopwords')

from nltk.tokenize import word_tokenize from
nltk.stem import PorterStemmer from
nltk.stem import WordNetLemmatizer

# Sample text for demonstration
text = "NLTK is a great toolkit for Natural Language Processing. Tokenization, Stemming, and
Lemmatization are important tasks."

# Tokenization: Split text into words tokens
= word_tokenize(text)
print("Tokens:", tokens)

# Stemming: Reduce words to their root form using Porter Stemmer stemmer
= PorterStemmer()
stemmed_words = [stemmer.stem(word) for word in tokens] print("Stemmed
words:", stemmed_words)

# Lemmatization: Reduce words to their base form using WordNet Lemmatizer
lemmatizer = WordNetLemmatizer() lemmatized_words =
[lemmatizer.lemmatize(word) for word in tokens] print("Lemmatized words
(default pos=noun):", lemmatized_words) # Optional: Lemmatization with
POS tagging (verbs, adjectives, etc.) lemmatized_verbs =
[lemmatizer.lemmatize(word, pos='v') for word in tokens] print("Lemmatized
words (as verbs):", lemmatized_verbs)
```

Output:

```
Requirement already satisfied: nltk in c:\users\imrd\anaconda3\lib\site-packages (3.9.1)
Requirement already satisfied: click in c:\users\imrd\anaconda3\lib\site-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in c:\users\imrd\anaconda3\lib\site-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in c:\users\imrd\anaconda3\lib\site-packages (from nltk) (2024.9.11)
Requirement already satisfied: tqdm in c:\users\imrd\anaconda3\lib\site-packages (from nltk) (4.66.5)
Requirement already satisfied: colorama in c:\users\imrd\anaconda3\lib\site-packages (from click->nltk) (0.4.6)
Note: you may need to restart the kernel to use updated packages.
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\IMRD\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
True
[nltk_data] Downloading package punkt_tab to
[nltk_data]     C:\Users\IMRD\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt_tab.zip.
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\IMRD\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\IMRD\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Tokens: ['NLTK', 'is', 'a', 'great', 'toolkit', 'for', 'Natural', 'Language', 'Processing', '.', 'Tokenization', ',', 'Stemming', ',', 'and', 'Lemmatization', 'are', 'important', 'tasks', '.']
Stemmed words: ['nltk', 'is', 'a', 'great', 'toolkit', 'for', 'natur', 'languag', 'process', '.', 'token', ',', 'stem', ',', 'and', 'lemmat', 'are', 'import', 'task', '.']
Lemmatized words (default pos=noun): ['NLTK', 'is', 'a', 'great', 'toolkit', 'for', 'Natural', 'Language', 'Processing', '.', 'Tokenization', ',', 'Stemming', ',', 'and', 'Lemmatization', 'are', 'important', 'task', '.']
Lemmatized words (as verbs): ['NLTK', 'be', 'a', 'great', 'toolkit', 'for', 'Natural', 'Language', 'Processing', '.', 'Tokenization', ',', 'Stemming', ',', 'and', 'Lemmatization', 'be', 'important', 'task', '.']