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

```
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

data = pd.read_csv('Data science II/advertising.csv')
print(data)

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

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)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2) Score: {r2}")

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()
```

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

df = pd.read_csv('log.csv')

X = df.iloc[:, :-1]
y = df.iloc[:, -1]

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

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = LogisticRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

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

print(f'Accuracy: {accuracy:.4f}')
print('Confusion Matrix:')
print(conf_matrix)
print('Classification Report:')
print(report)
```

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

df = pd.read_csv('log.csv')

X = df.iloc[:, :-1]
y = df.iloc[:, -1]

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

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = DecisionTreeClassifier()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

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

print(f'Accuracy: {accuracy:.4f}')
print('Confusion Matrix:')
print(conf_matrix)
print('Classification Report:')
print(report)
```

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

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

X, y = make_blobs(n_samples=300, centers=4, random_state=42)

kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(X)

centroids = kmeans.cluster_centers_
labels = kmeans.labels_

plt.figure(figsize=(8, 6))

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o', edgecolor='k')

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

plt.title('K-means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

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

```
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

df = pd.read_csv('log.csv')
print(df.head())

X = df.iloc[:, :-1].values  # All rows, all columns except the last one
y = df.iloc[:, -1].values   # Last column is the target

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=2)  # Reduce to 2 components for visualization
X_pca = pca.fit_transform(X_scaled)

print("Explained variance ratio:", pca.explained_variance_ratio_)
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()
```

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

```
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

df = pd.read_csv('log.csv')

print(df.head())

df = df.dropna()

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

label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train, y_train)
y_pred = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of Random Forest on test data: {accuracy * 100:.2f}%')
```

Assignment 7: Write a program to implementing Boosting using AdaBoost

```
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

df = pd.read_csv('log.csv')

print(df.head())

df = df.dropna()

X = df.iloc[:, :-1].values  # All rows, all columns except the last one (features)
y = df.iloc[:, -1].values  # Last column is the target
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

base_estimator = DecisionTreeClassifier(max_depth=1)

adaboost_classifier = AdaBoostClassifier(estimator=base_estimator, n_estimators=50,
random_state=42)
adaboost_classifier.fit(X_train, y_train)

y_pred = adaboost_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy of AdaBoost on test data: {accuracy * 100:.2f}%')
```

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

df = pd.read_csv('pca.csv')
print(df.head())

X = df.iloc[:, :-1]
y = df.iloc[:, -1]

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

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)

y_pred = svm_model.predict(X_test)

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


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

if X.shape[1] == 2:
    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))

    Z = svm_model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    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()
```

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,)),  Dense(24, activation='relu'),
   Dense(12, activation='relu'),
   Dense(1, activation='sigmoid'),
])

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

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

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


plt.show()

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

```
from nltk.stem import WordNetLemmatizer

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

tokens = word_tokenize(text)
print("Tokens:", tokens)

stemmer = PorterStemmer()
stemmed_words = [stemmer.stem(word) for word in tokens]
print("Stemmed words:", stemmed_words)

lemmatizer = WordNetLemmatizer()
lemmatized_words = [lemmatizer.lemmatize(word) for word in tokens]
print("Lemmatized words (default pos=noun):", lemmatized_words)
lemmatized_verbs = [lemmatizer.lemmatize(word, pos='v') for word in tokens]
print("Lemmatized words (as verbs):", lemmatized_verbs)
```

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()
```

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

```
pip install pymongo
```

```
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")

db = client['mydatabase']
collection = db['users']

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}")

user = collection.find_one({"name": "John Doe"})
print("Found user:", user)

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_result = collection.delete_one({"name": "John Doe"})
print(f"Documents deleted: {delete_result.deleted_count}")
```

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

```
pip install nltk
import nltk

nltk.download('punkt')

import nltk

nltk.download('punkt_tab')

nltk.download('wordnet')
nltk.download('stopwords')

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