Q1] Introduction to various Python libraries for machine learning a. NumPy b. Pandas c. Matplotlib d. Seaborn e. Scikit learn

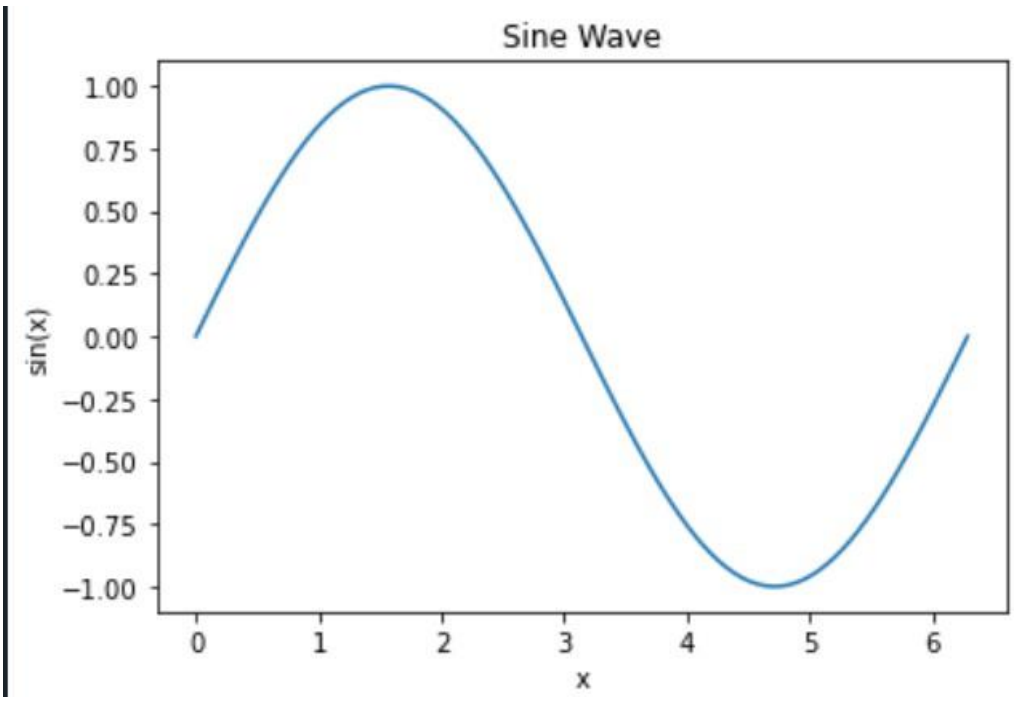
```
# Import the quired libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

# NumPy: Introduction

print("a. NumPy - Numerical Python")

arr = np.array([1, 2, 3, 4, 5])

print("Sample NumPy array:", arr)

print("Array Shape:", arr.shape)

print("Array Sum:", np.sum(arr))

print("\n")

# Pandas: Introduction

print("b. Pandas - Data Analysis Library")

data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],

        'Age': [25, 30, 22, 35]} df =

pd.DataFrame(data) print("Sample

Pandas DataFrame:") print(df)

print("Summary Statistics:")

print(df.describe())

print("\n")

# Matplotlib: Introduction

print("c. Matplotlib - Data Visualization Library")

x = np.linspace(0, 2 * np.pi, 100)
```

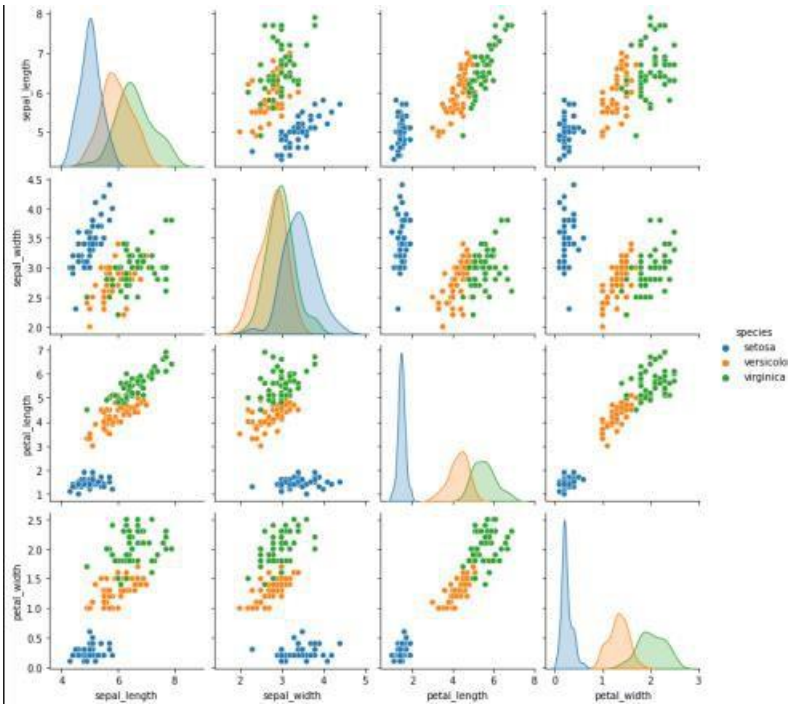
```
y = np.sin(x)
plt.figure()
plt.plot(x, y)
plt.title("Sine Wave")
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.show()
print("\n")
# Seaborn: Introduction
print("d. Seaborn - Statistical Data Visualization Library")
iris = sns.load_dataset("iris")
sns.pairplot(iris, hue="species")
plt.show()
print("\n")
# Scikit-Learn: Introduction
print("e. Scikit-Learn - Machine Learning Library")
# Load a sample dataset (Iris dataset)
iris = load_iris()
X, y = iris.data, iris.target
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train a Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
# Make predictions and calculate accuracy
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Random Forest Classifier Accuracy:", accuracy)
# Generate a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
```

print("Confusion Matrix:")

print(conf_matrix)

```
In [1]: runfile('D:/python/krai1.py', wdir='D:/pyt
a. NumPy - Numerical Python
Sample NumPy array: [1 2 3 4 5]
Array Shape: (5,)
Array Sum: 15


b. Pandas - Data Analysis Library
Sample Pandas DataFrame:
      Name  Age
0    Alice   25
1      Bob   30
2  Charlie   22
3    David   35
Summary Statistics:
             Age
count   4.000000
mean   28.000000
std     5.715476
min    22.000000
25%    24.250000
50%    27.500000
75%    31.250000
max    35.000000
```






```
e. Scikit-Learn - Machine Learning Library
Random Forest Classifier Accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

In [2]:
```

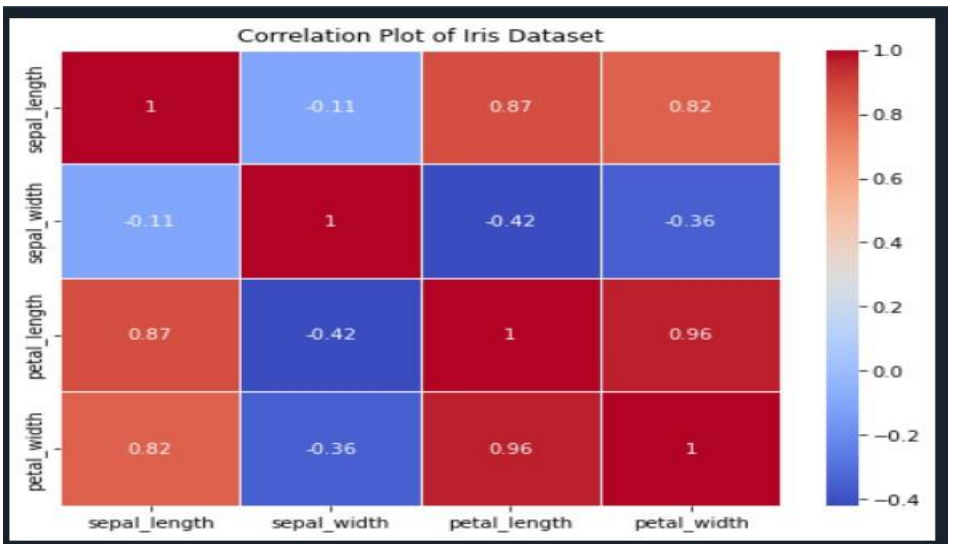
Q2] Write a program to find the correlation matrix

```
import pandas as pd
data = {
    'A': [1, 2, 3, 4, 5],
    'B': [5, 4, 3, 2, 1],
    'C': [2, 3, 1, 4, 5],
}
df = pd.DataFrame(data)
correlation_matrix = df.corr()
print("Correlation Matrix:")
print(correlation_matrix)
```

```
 [ 0  0 11]]

In [2]: runfile('D:/python/krai2.py', wdir='D:/python')
Correlation Matrix:
     A    B    C
A  1.0 -1.0  0.7
B -1.0  1.0 -0.7
C  0.7 -0.7  1.0
```

Q3] Write a program to Plot the correlation plot on dataset and visualize giving an overview of relationships among data on iris data.

```
import seaborn as sns
import matplotlib.pyplot as plt
iris = sns.load_dataset("iris")
correlation_matrix = iris.corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", linewidths=0.5)
plt.title("Correlation Plot of Iris Dataset")
plt.show()
```



Q4] Write a program to implement Analysis of covariance: variance (ANOVA) on IRIS dataset

```
import pandas as pd

from scipy.stats import f_oneway

iris = pd.read_csv("https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv")

grouped_data = [group["sepal_length"] for name, group in iris.groupby("species")]

f_statistic, p_value = f_oneway(*grouped_data)

alpha = 0.05

print("One-way ANOVA Results:")

print(f"F-statistic: {f_statistic:.2f}")

print(f"P-value: {p_value:.4f}")

if p_value < alpha:
    print("Reject the null hypothesis. There are significant differences in means.")

else:
    print("Fail to reject the null hypothesis. There are no significant differences in means.")
```

```
In [4]: runfile('D:/python/krai4.py', wdir='D:/python')
One-way ANOVA Results:
F-statistic: 119.26
P-value: 0.0000
Reject the null hypothesis. There are significant differences in means.

In [5]:
```

Q5] Write a program to implement linear regression algorithm to create and evaluate a model on a given dataset

```
import numpy as np

import pandas as pd

from sklearn.datasets import fetch_openml

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

import matplotlib.pyplot as plt

california_housing = fetch_openml(name="california_housing", as_frame=True)

data = california_housing.frame

target = data["target"]

X_train, X_test, y_train, y_test = train_test_split(data, target, 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("Mean Squared Error: {:.2f}".format(mse))

print("R-squared (Coefficient of Determination): {:.2f}".format(r2))

plt.scatter(y_test, y_pred)

plt.xlabel("Actual Values")

plt.ylabel("Predicted Values")

plt.title("Actual vs. Predicted Values in Linear Regression")

plt.show()
```

Q6] Write a program to classify the given dataset using logistic regression and evaluate the model

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
data = pd.DataFrame({
    'Feature1': [1.2, 2.4, 1.5, 3.5, 2.7, 4.8, 3.2, 5.1, 4.0, 6.2],
    'Feature2': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
    'Target': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
})
X = data[['Feature1', 'Feature2']]
y = data['Target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
classification_rep = classification_report(y_test, y_pred)
print("Classification Report:")
print(classification_rep)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
In [8]: runfile('D:/python/krai7.py', wdir='D:/python')
Accuracy: 0.0
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       1.0
           1       0.00      0.00      0.00       1.0

    accuracy                           0.00       2.0
   macro avg       0.00      0.00      0.00       2.0
weighted avg       0.00      0.00      0.00       2.0

Confusion Matrix:
[[0 1]
 [1 0]]
```

Q7] Write a program to implement support vector machine algorithm

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
data = pd.DataFrame({
    'Feature1': [1.2, 2.4, 1.5, 3.5, 2.7, 4.8, 3.2, 5.1, 4.0, 6.2],
    'Feature2': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
    'Target': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
})
X = data[['Feature1', 'Feature2']]
y = data['Target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = SVC(kernel='linear', C=1.0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

```
print("Accuracy:", accuracy)
classification_rep = classification_report(y_test, y_pred)
print("Classification Report:")
print(classification_rep)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

Q8] Write a program to implement Decision Tree model on the given dataset

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
data = pd.DataFrame({
    'Feature1': [1.2, 2.4, 1.5, 3.5, 2.7, 4.8, 3.2, 5.1, 4.0, 6.2],
    'Feature2': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
    'Target': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
})
X = data[['Feature1', 'Feature2']]
y = data['Target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
classification_rep = classification_report(y_test, y_pred)
print("Classification Report:")
print(classification_rep)
```

conf_matrix = confusion_matrix(y_test, y_pred)

print("Confusion Matrix:")

print(conf_matrix)

```
In [8]: runfile('D:/python/krai7.py', wdir='D:/python')
Accuracy: 0.0
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       1.0
           1       0.00      0.00      0.00       1.0

    accuracy                           0.00       2.0
   macro avg       0.00      0.00      0.00       2.0
weighted avg       0.00      0.00      0.00       2.0

Confusion Matrix:
[[0 1]
 [1 0]]
```

Q9] Write a program to implement Bayesian classification on given dataset.

import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

data = pd.DataFrame({

    'Feature1': [1.2, 2.4, 1.5, 3.5, 2.7, 4.8, 3.2, 5.1, 4.0, 6.2],

    'Feature2': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1],

    'Target': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]

})

X = data[['Feature1', 'Feature2']]

y = data['Target']

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

model = GaussianNB()

model.fit(X_train, y_train)

```
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
classification_rep = classification_report(y_test, y_pred)
print("Classification Report:")
print(classification_rep)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
In [8]: runfile('D:/python/krai7.py', wdir='D:/python')
Accuracy: 0.0
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       1.0
           1       0.00      0.00      0.00       1.0

    accuracy                           0.00       2.0
   macro avg       0.00      0.00      0.00       2.0
weighted avg       0.00      0.00      0.00       2.0

Confusion Matrix:
[[0 1]
 [1 0]]
```

Q10] Write a program to implement K-Nearest Neighbor algorithm on given dataset.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
data = pd.DataFrame({
    'Feature1': [1.2, 2.4, 1.5, 3.5, 2.7, 4.8, 3.2, 5.1, 4.0, 6.2],
    'Feature2': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
    'Target': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
})
X = data[['Feature1', 'Feature2']]
```

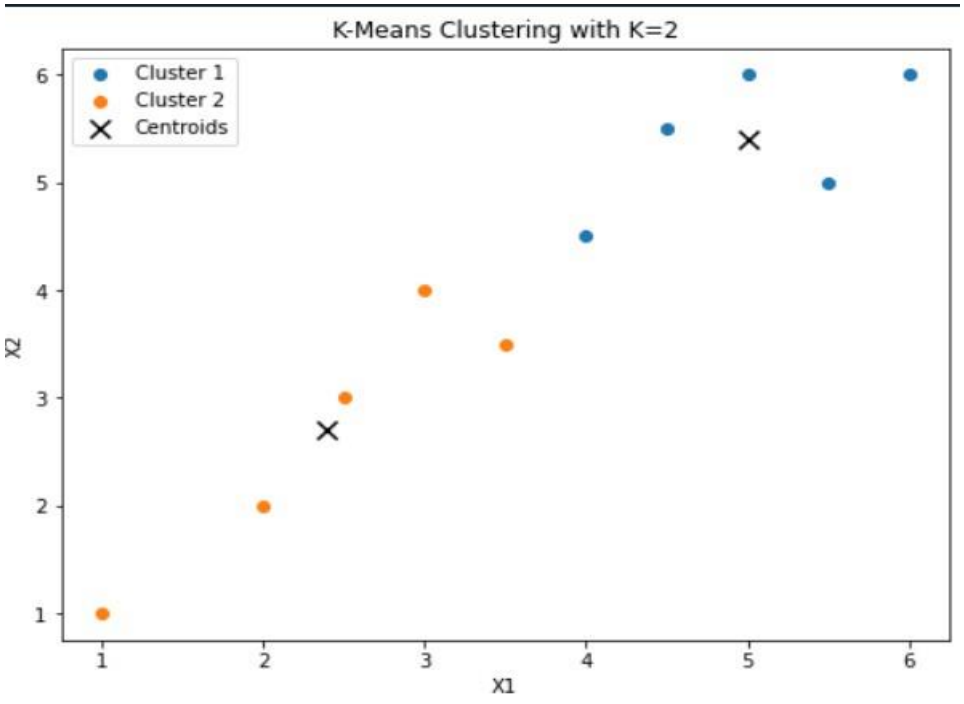
```
y = data['Target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
classification_rep = classification_report(y_test, y_pred)
print("Classification Report:")
print(classification_rep)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
In [8]: runfile('D:/python/krai7.py', wdir='D:/python')
Accuracy: 0.0
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       1.0
           1       0.00      0.00      0.00       1.0

    accuracy                           0.00       2.0
   macro avg       0.00      0.00      0.00       2.0
weighted avg       0.00      0.00      0.00       2.0

Confusion Matrix:
[[0 1]
 [1 0]]
```

Q11] Write a program to implement K-Means algorithm on given dataset and visualize the clusters.

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
data = pd.DataFrame({
    'X1': [1, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6],
```

```
    'X2': [1, 2, 3, 4, 3.5, 4.5, 5.5, 6, 5, 6]
})
K = 2
model = KMeans(n_clusters=K, random_state=42)
model.fit(data)
cluster_centers = model.cluster_centers_
labels = model.labels_
plt.figure(figsize=(8, 6))
for k in range(K):
    cluster_data = data[labels == k]
    plt.scatter(cluster_data['X1'], cluster_data['X2'], label=f'Cluster {k + 1}')
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], c='black', marker='x', s=100,
label='Centroids')
plt.title(f'K-Means Clustering with K={K}')
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()
plt.show()
```



Q12] Write a program to implement deep learning algorithm using ANN

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
model = keras.models.Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
print(f"Test accuracy: {test_accuracy}")
predictions = model.predict(x_test)
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_test[i],  cmap=plt.cm.binary)
    predicted_label = tf.argmax(predictions[i])
    true_label = y_test[i]
    if predicted_label == true_label:
```

```
        color = 'green'
    else:
        color = 'red'
    plt.xlabel(f"{predicted_label} ({true_label})", color=color)
plt.show()
```

Q13] Write a program to implement deep learning algorithm using CNN

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
model = keras.Sequential([
    layers.Input(shape=(28, 28, 1)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
```

```
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=2)
print(f"Test accuracy: {test_accuracy}")
predictions = model.predict(x_test)
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_test[i].reshape(28, 28), cmap=plt.cm.binary)
    predicted_label = tf.argmax(predictions[i])
    true_label = y_test[i]
    if predicted_label == true_label:
        color = 'green'
    else:
        color = 'red'
    plt.xlabel(f"{predicted_label} ({true_label})", color=color)
plt.show()
```

Q14] Write a program to implement deep learning algorithm using GAN

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
def build_generator(latent_dim):
    model = keras.Sequential()
    model.add(layers.Dense(7 * 7 * 128, input_dim=latent_dim))
    model.add(layers.Reshape((7, 7, 128)))
    model.add(layers.Conv2DTranspose(128, kernel_size=4, strides=2, padding="same"))
    model.add(layers.BatchNormalization(momentum=0.8))
```

```
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Conv2DTranspose(64, kernel_size=4, strides=2, padding="same"))
    model.add(layers.BatchNormalization(momentum=0.8))
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Conv2D(1, kernel_size=4, padding="same", activation="tanh"))
    return model
def build_discriminator(img_shape):
    model = keras.Sequential()
    model.add(layers.Conv2D(32, kernel_size=4, strides=2, input_shape=img_shape,
padding="same"))
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Conv2D(64, kernel_size=4, strides=2, padding="same"))
    model.add(layers.BatchNormalization(momentum=0.8))
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Conv2D(128, kernel_size=4, strides=2, padding="same"))
    model.add(layers.BatchNormalization(momentum=0.8))
    model.add(layers.LeakyReLU(alpha=0.2))
    model.add(layers.Flatten())
    model.add(layers.Dense(1, activation="sigmoid"))
    return model
discriminator = build_discriminator((28, 28, 1))
discriminator.compile(loss="binary_crossentropy",
optimizer=keras.optimizers.Adam(0.0002, 0.5), metrics=["accuracy"])
latent_dim = 100
generator = build_generator(latent_dim)
discriminator.trainable = False
gan_input = keras.Input(shape=(latent_dim,))
x = generator(gan_input)
gan_output = discriminator(x)
gan = keras.models.Model(gan_input, gan_output)
gan.compile(loss="binary_crossentropy", optimizer=keras.optimizers.Adam(0.0002, 0.5))
```

```
def train_gan(generator, discriminator, gan, batch_size, latent_dim, num_epochs):
    for epoch in range(num_epochs):
        for _ in range(int(60000 / batch_size)):
            noise = tf.random.normal((batch_size, latent_dim))
            generated_images = generator.predict(noise)
            real_images = x_train[np.random.randint(0, x_train.shape[0], batch_size)]
            labels_real = np.ones((batch_size, 1))
            labels_fake = np.zeros((batch_size, 1))
            d_loss_real = discriminator.train_on_batch(real_images, labels_real)
            d_loss_fake = discriminator.train_on_batch(generated_images, labels_fake)
            d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
            noise = tf.random.normal((batch_size, latent_dim))
            labels_gan = np.ones((batch_size, 1))
            g_loss = gan.train_on_batch(noise, labels_gan)
            print(f"Epoch {epoch}/{num_epochs}, D Loss: {d_loss[0]}, G Loss: {g_loss}")
        plot_generated_images(generator, epoch, latent_dim)
def plot_generated_images(generator, epoch, latent_dim, examples=10, dim=(1, 10),
figsize=(10, 1)):
    noise = tf.random.normal((examples, latent_dim))
    generated_images = generator.predict(noise)
    generated_images = 0.5 * generated_images + 0.5
    plt.figure(figsize=figsize)
    for i in range(examples): plt.subplot(dim[0], dim[1], i +
        1) plt.imshow(generated_images[i, :, :, 0],
        cmap="gray") plt.axis("off")
    plt.tight_layout()
    plt.savefig(f"gan_generated_image_epoch_{epoch}.png")
(x_train, _), (_, _) = keras.datasets.mnist.load_data()
x_train = x_train / 127.5 - 1.0
x_train = np.expand_dims(x_train, axis=3)
```

train_gan(generator, discriminator, gan, batch_size=64, latent_dim=latent_dim, num_epochs=100)

```
Tokenized words: ['natural', 'language', 'processing', 'nlp', 'is', 'a',
'subfield', 'of', 'artificial', 'intelligence', 'that', 'focuses', 'on',
'the', 'interaction', 'between', 'computers', 'and', 'humans', 'through',
'natural', 'language', 'nlp', 'enables', 'computers', 'to', 'understand',
'interpret', 'and', 'generate', 'human', 'language', 'in', 'a', 'valuable',
'way']
Filtered words (after removing punctuation and stop words): ['natural',
'language', 'processing', 'nlp', 'subfield', 'artificial', 'intelligence',
'focuses', 'interaction', 'computers', 'humans', 'natural', 'language',
'nlp', 'enables', 'computers', 'understand', 'interpret', 'generate',
'human', 'language', 'valuable', 'way']
Word frequency: Counter({'language': 3, 'natural': 2, 'nlp': 2,
'computers': 2, 'processing': 1, 'subfield': 1, 'artificial': 1,
'intelligence': 1, 'focuses': 1, 'interaction': 1, 'humans': 1, 'enables':
1, 'understand': 1, 'interpret': 1, 'generate': 1, 'human': 1, 'valuable':
1, 'way': 1})
Sentiment Analysis - Polarity: -0.1
Sentiment Analysis - Subjectivity: 0.475
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\LENOVO\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\LENOVO\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Q16] Write a program to implement web scrapping on the given URL

```
import requests
from bs4 import BeautifulSoup
url = "https://example.com"
response = requests.get(url)
if response.status_code == 200:
    soup = BeautifulSoup(response.text, "html.parser")
    links = soup.find_all("a")
    for link in links:
        print(link.get("href"))
else:
    print("Failed to retrieve the webpage. Status code:", response.status_code)
```

```
[nltk_data]   Package punkt is already up-to-date!

In [27]: runfile('D:/python/krai16.py', wdir='D:/python')
https://www.iana.org/domains/example

In [28]:
```