Q1. Find the correlation matrix.

Ans:

```
import numpy as np

data = np.array([[1, 2, 3], [4, 5, 6],[7, 8, 9]])

# Calculate the correlation matrix

correlation_matrix = np.corrcoef(data, rowvar=False)
print("Correlation Matrix:")
print(correlation_matrix)
```

Output:

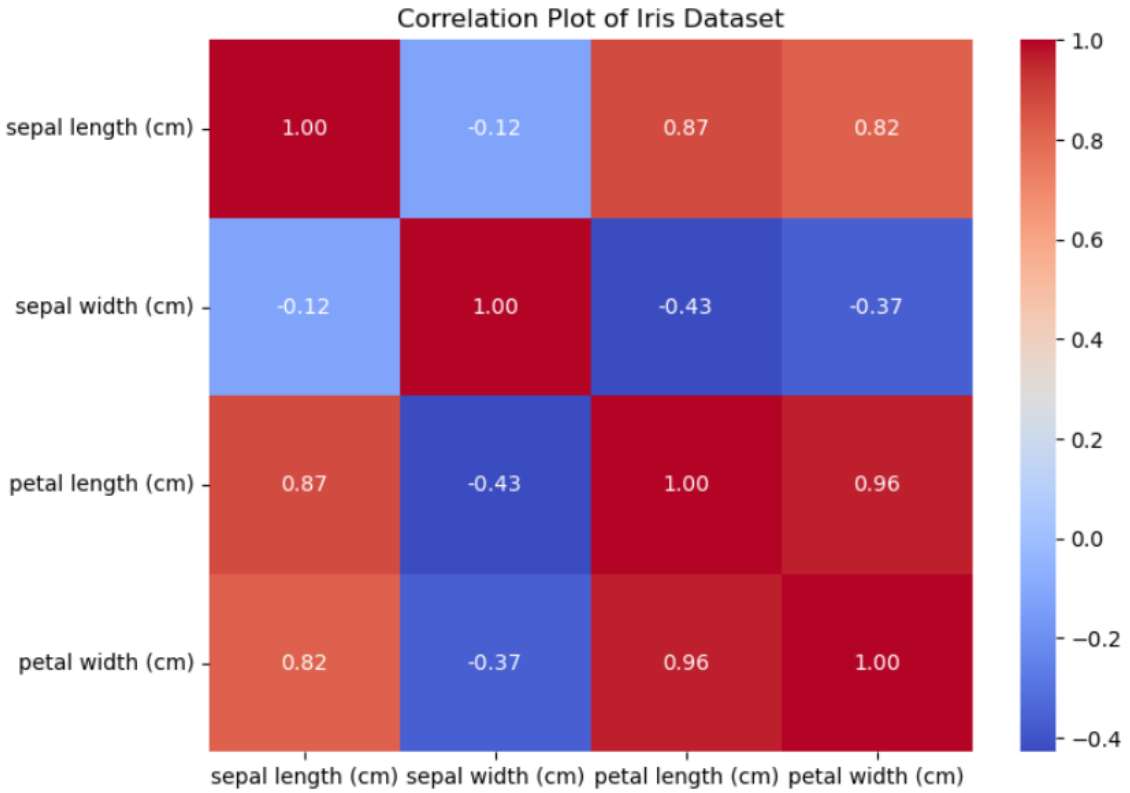
```
Correlation Matrix:
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

Q2. Plot the correlation plot on dataset and visualize giving an overview of relationships among data on iris data.

Ans:

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import pandas as pd

# Load Iris dataset
iris = load_iris()
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
# Calculate the correlation matrix
correlation_matrix = iris_df.corr()
# Create a heatmap of the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', annot_kws={"size":
10})
plt.title('Correlation Plot of Iris Dataset')
plt.show()
```

Output:



3.Analysis of covariance: variance (ANOVA), if data have categorical variables on iris data.

Ans:

```
import numpy as np
import pandas as pd
from scipy.stats import f_oneway
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()
iris_data = iris.data
iris_feature_names = iris.feature_names
iris_target = iris.target

# Create a DataFrame from the Iris data
iris_df = pd.DataFrame(data=iris_data, columns=iris_feature_names)
iris_df['Species'] = iris.target_names[iris_target]

# Perform one-way ANOVA for each feature
for feature in iris_feature_names:
    groups = [iris_df[feature][iris_df['Species'] == species] for species in iris.target_names]
    f_statistic, p_value = f_oneway(*groups)
    print(f"ANOVA for {feature}: F-statistic={f_statistic:.2f}, p-value={p_value:.4f}")
```

Output:

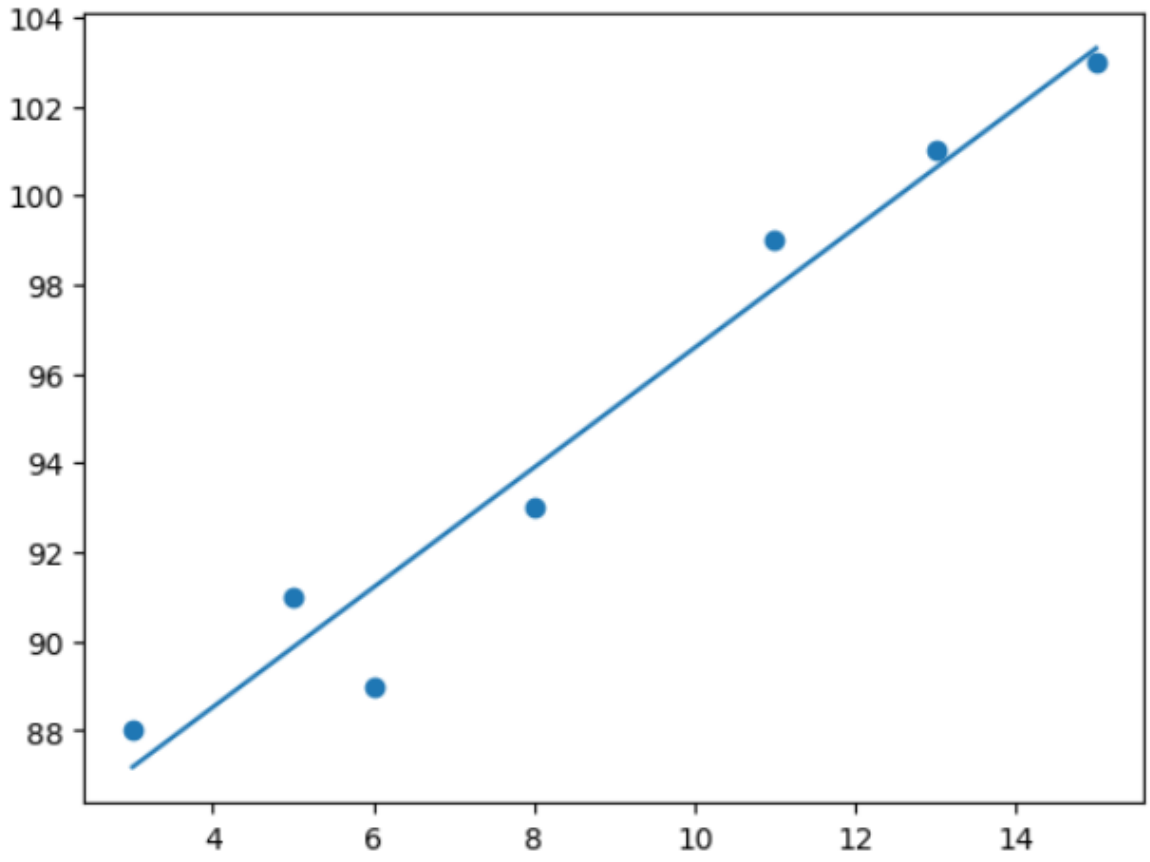
```
ANOVA for sepal length (cm): F-statistic=119.26, p-value=0.0000
ANOVA for sepal width (cm): F-statistic=49.16, p-value=0.0000
ANOVA for petal length (cm): F-statistic=1180.16, p-value=0.0000
ANOVA for petal width (cm): F-statistic=960.01, p-value=0.0000
```

Q4. Apply linear regression Model techniques to predict the data on any dataset.

Ans:

```
import matplotlib.pyplot as plt
from scipy import stats
# New input values
x = [3, 5, 6, 8, 11, 13, 15]
y = [88, 91, 89, 93, 99, 101, 103]

# Calculate linear regression
slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
    return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

Output:



Q5 Apply logical regression model techniques to predict the data on any dataset.

Ans:

```
import numpy as np
from sklearn import linear_model
# Input data (tumor sizes)
X = np.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1, 1)
# Output labels (0 for non-cancerous, 1 for cancerous)
y = np.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
# Create and fit the logistic regression model
logr = linear_model.LogisticRegression()
logr.fit(X, y)

# Predict if tumor is cancerous where the size is 3.46mm
size_to_predict = np.array([3.46]).reshape(-1, 1)
predicted = logr.predict(size_to_predict)
print ("Prediction for tumor size 3.46mm:", predicted)
```

Output:

```
Prediction for tumor size 3.46mm: [0]
```

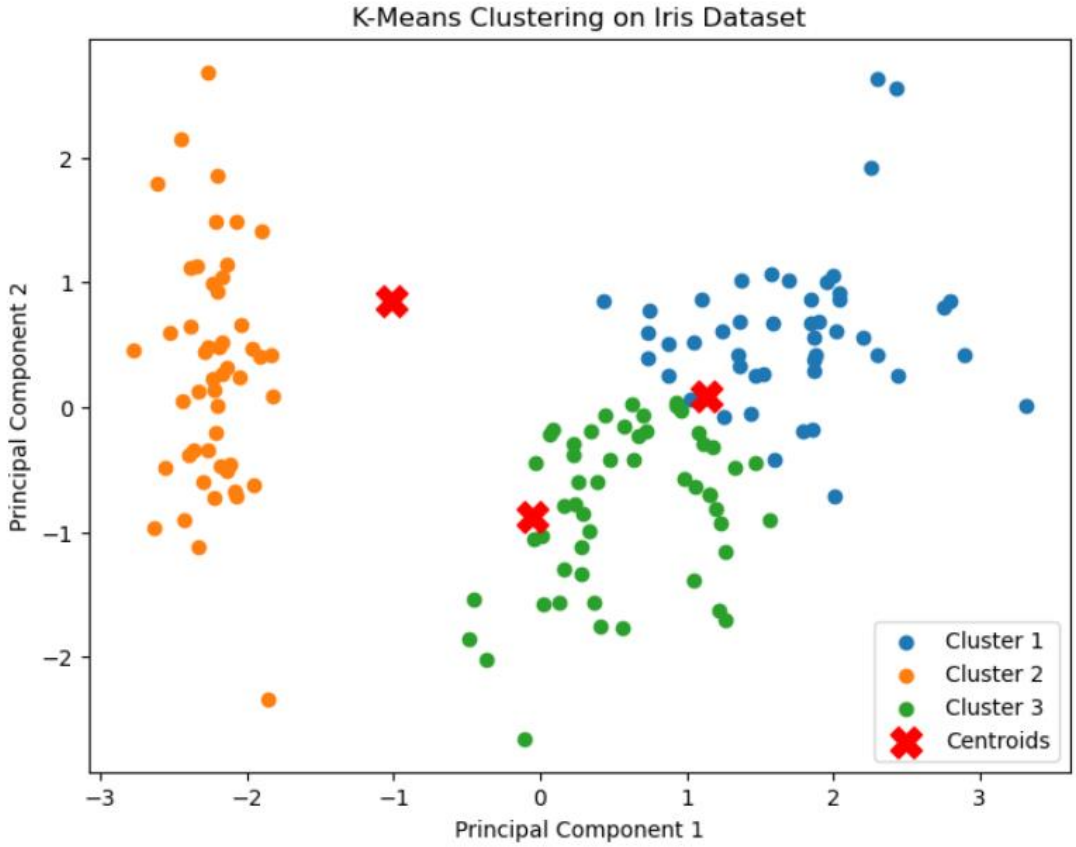
Q6. Clustering algorithms for unsupervised classification.

Ans:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris = load_iris()
iris_data = iris.data
iris_feature_names = iris.feature_names

# Standardize the features (important for K-Means)
scaler = StandardScaler()
iris_data_standardized = scaler.fit_transform(iris_data)

# Apply K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(iris_data_standardized)

# Add cluster labels to the original dataset for visualization
iris_df = pd.DataFrame(data=iris_data_standardized, columns=iris_feature_names)
iris_df['Cluster'] = labels

# Visualize the clusters using PCA (for simplicity, using only the first two principal components)
pca = PCA(n_components=2)
iris_pca = pca.fit_transform(iris_data_standardized)

# Plot the clusters
plt.figure(figsize=(8, 6))
for cluster in range(3):
    plt.scatter(
        iris_pca[labels == cluster, 0],
        iris_pca[labels == cluster, 1],
        label=f'Cluster {cluster + 1}'
    )

plt.scatter(
    kmeans.cluster_centers_[:, 0],
    kmeans.cluster_centers_[:, 1],
    s=200,
```

```
    c='red',
    marker='X',
    label='Centroids'
)
plt.title('K-Means Clustering on Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```

Output:



Q7. Association algorithms for supervised classification on any dataset.

Ans:

```
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
from sklearn.datasets import load_breast_cancer
# Load the breast cancer dataset
cancer = load_breast_cancer()
cancer_data = cancer.data
cancer_feature_names = cancer.feature_names
cancer_target = cancer.target
# Create a DataFrame from the breast cancer data
cancer_df = pd.DataFrame(data=cancer_data, columns=cancer_feature_names)
cancer_df['Target'] = cancer_target

# Select features and target variable
X = cancer_df[cancer_feature_names]
y = cancer_df['Target']

# Split the data 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)

# Create and train the Decision Tree model
decision_tree_model = DecisionTreeClassifier(random_state=42)
decision_tree_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = decision_tree_model.predict(X_test)
\
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Display classification report
print('Classification Report:')
print(classification_report(y_test, y_pred))
```

Output:

```
Accuracy: 0.95
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        43
           1       0.96      0.96      0.96        71

    accuracy                           0.95       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114
```

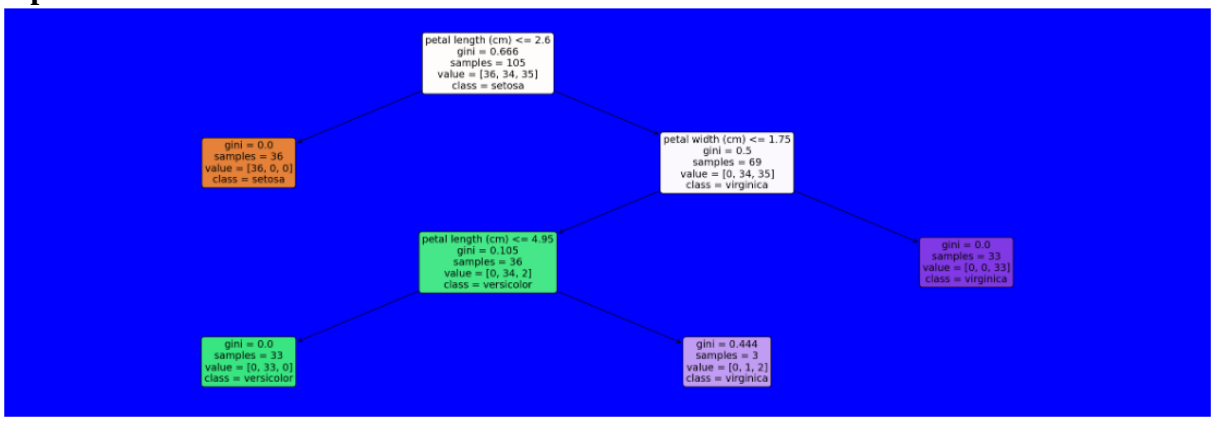
Q8. Developing and implementing Decision Tree model on the dataset.

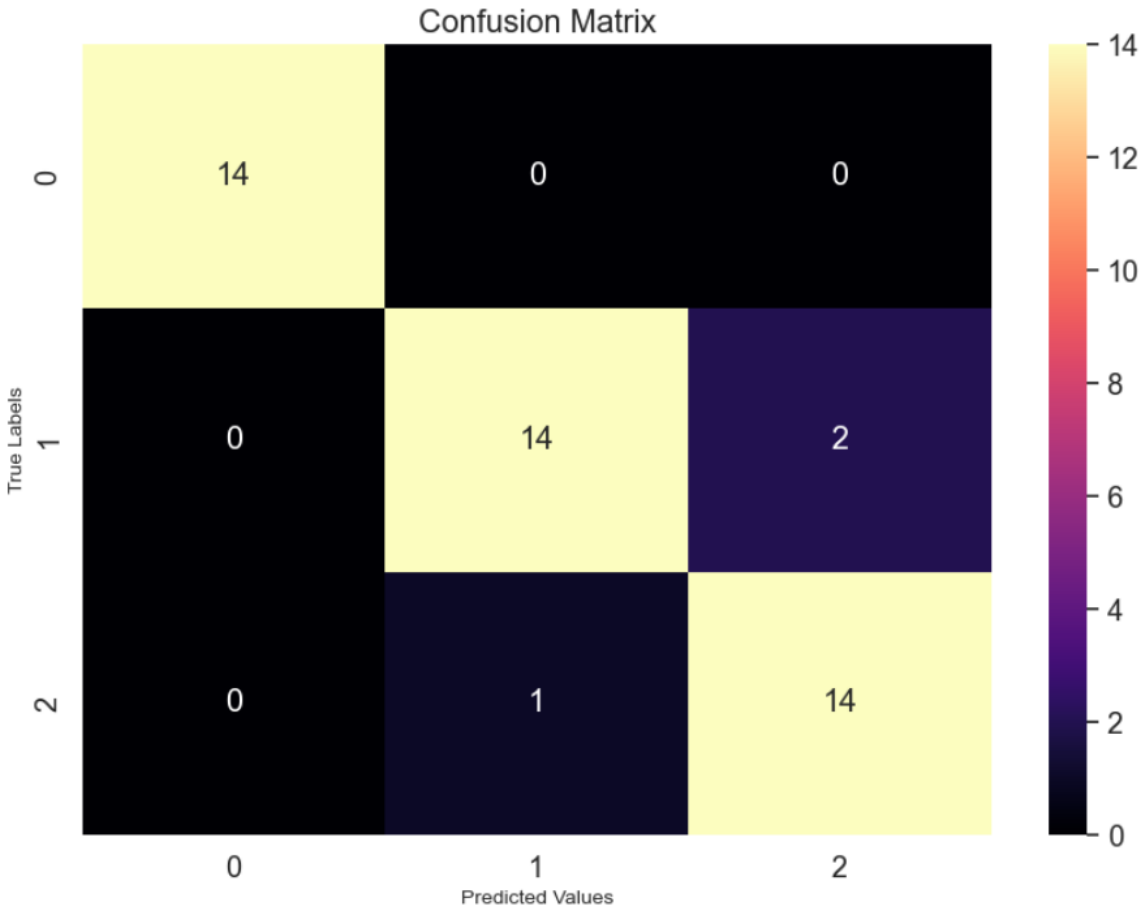
Ans:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree  # Import plot_tree directly
# Load the Iris dataset
iris = load_iris()
# Create DataFrame
data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
data['Species'] = iris.target
target = np.unique(iris.target)
target_n = np.unique(iris.target_names)
target_dict = dict(zip(target, target_n))
data['Species'] = data['Species'].replace(target_dict)
x = data.drop(columns="Species")
y = data["Species"]
names_features = x.columns
target_labels = y.unique()
# Split data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=93)
# Decision Tree Classifier
dtc = DecisionTreeClassifier(max_depth=3, random_state=93)
dtc.fit(x_train, y_train)
# Visualize the Decision Tree
plt.figure(figsize=(30, 10), facecolor='b')
plot_tree(dtc, feature_names=names_features, class_names=target_labels,
          rounded=True, filled=True, fontsize=14)
plt.show()

# Predictions and Confusion Matrix
y_pred = dtc.predict(x_test)
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
matrix = pd.DataFrame(confusion_matrix)

# Plot Confusion Matrix
plt.figure(figsize=(10, 7))
sns.set(font_scale=1.3)
sns.heatmap(matrix, annot=True, fmt="g", cmap="magma")
plt.title('Confusion Matrix')
```

```
plt.xlabel("Predicted Values", fontsize=10)
plt.ylabel("True Labels", fontsize=10)
plt.show()
```

Output:





Q9. Bayesian classification on any dataset.

Ans:

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import datasets
from sklearn import metrics

# Load the Iris dataset
iris = datasets.load_iris()

X = iris.data  # Features
y = iris.target  # Target variable

# Split the dataset 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)

# Create a Gaussian Naive Bayes classifier
model = GaussianNB()

# Train the model
model.fit(X_train, y_train)

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

# Evaluate the model
accuracy = metrics.accuracy_score(y_test, y_pred)
precision = metrics.precision_score(y_test, y_pred, average='weighted')
recall = metrics.recall_score(y_test, y_pred, average='weighted')
f1 = metrics.f1_score(y_test, y_pred, average='weighted')

# Display the evaluation metrics
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
```

Output:

```
Accuracy: 0.98
Precision: 0.98
Recall: 0.98
F1 Score: 0.98
```

Q10. SVM Classification on any dataset.

Ans:

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import datasets
from sklearn import metrics

# Load the Iris dataset
iris = datasets.load_iris()

X = iris.data  # Features
y = iris.target  # Target variable

# Split the dataset 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)

# Create an SVM classifier with a linear kernel
model = SVC(kernel='linear')

# Train the model
model.fit(X_train, y_train)

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

# Evaluate the model
accuracy = metrics.accuracy_score(y_test, y_pred)
precision = metrics.precision_score(y_test, y_pred, average='weighted')
recall = metrics.recall_score(y_test, y_pred, average='weighted')
f1 = metrics.f1_score(y_test, y_pred, average='weighted')

# Display the evaluation metrics
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
```

Output:

```
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1 Score: 1.00
```

Q11. Text Mining algorithms on unstructured dataset.

Ans:

```
import pandas as pd
import numpy as np
import nltk
import os

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

nltk.download('punkt')

text = "In Brazil they drive on the right-hand side of the road. Brazil has a large coastline on the eastern side of South America"

token = word_tokenize(text)
print(token)
```

Output:

```
['In', 'Brazil', 'they', 'drive', 'on', 'the', 'right-hand', 'side', 'of', 'the', 'road', '.', 'Brazil', 'has', 'a', 'large', 'coastline', 'on', 'the', 'eastern', 'side', 'of', 'South', 'America']
```

```
from nltk.probability import FreqDist
fdist = FreqDist(token)
print(fdist)
```

Output:

```
FreqDist({'the': 3, 'Brazil': 2, 'on': 2, 'side': 2, 'of': 2, 'In': 1, 'they': 1, 'drive': 1, 'right-hand': 1, 'road': 1, ...})
```

```
fdist1 = fdist.most_common(10)
print(fdist1)
```

Output:

```
[('the', 3), ('Brazil', 2), ('on', 2), ('side', 2), ('of', 2), ('In', 1), ('they', 1), ('drive', 1), ('right-hand', 1), ('road', 1)]
```

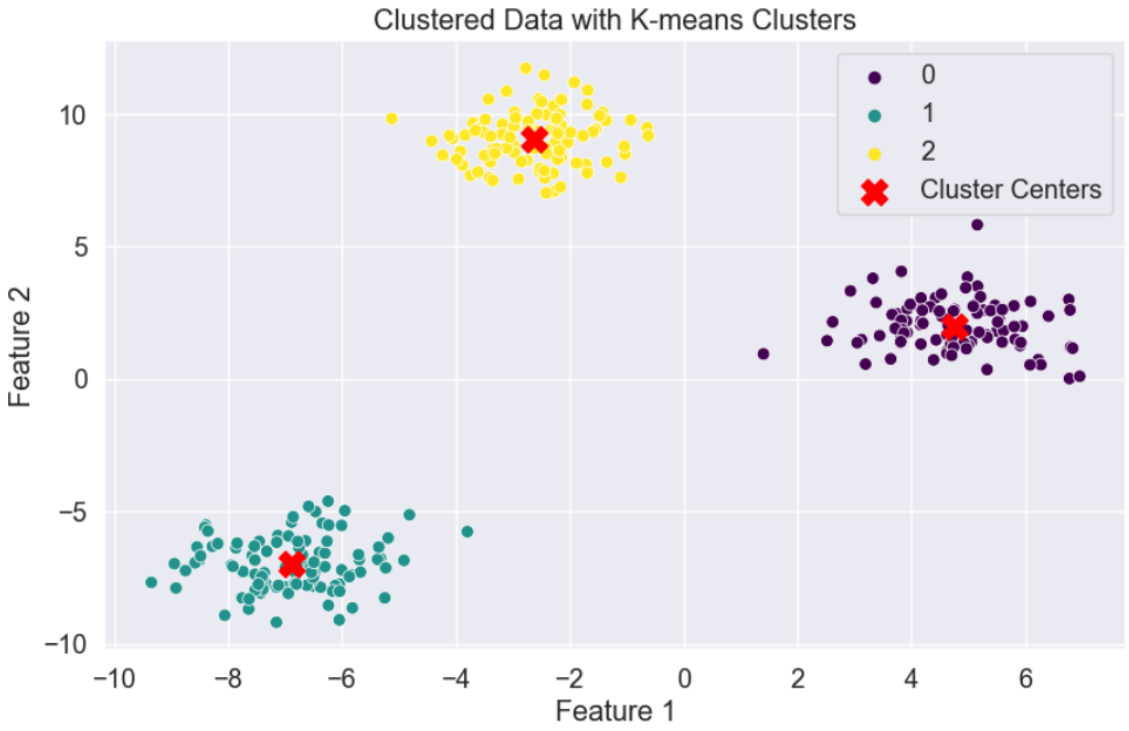
Q12. Plot cluster data using python visualization.

Ans:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate some sample clustered data
data, labels = make_blobs(n_samples=300, centers=3, random_state=42)

# Convert data to a Pandas DataFrame for easier handling
df = pd.DataFrame(data, columns=['Feature 1', 'Feature 2'])
df['True Labels'] = labels

# Apply K-means clustering
kmeans = KMeans(n_clusters=3)
kmeans.fit(data)
df['Predicted Labels'] = kmeans.predict(data)

# Plot the original data with colors representing true clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Feature 1', y='Feature 2', hue='True Labels', palette='viridis', legend='full',
s=50, data=df)
plt.title("Original Data with True Clusters")
plt.show()

# Plot the clustered data with colors representing clusters assigned by K-means
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Feature 1', y='Feature 2', hue='Predicted Labels', palette='viridis', legend='full',
s=50, data=df)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], marker='X', s=200,
c='red', label='Cluster Centers')
plt.title("Clustered Data with K-means Clusters")
plt.legend()
plt.show()
```

Output:


Original Data with True Clusters
True Labels
0
1
2
Feature 2
Feature 1
10
5
0
−5
−10
−10
−8
−6
−4
−2
0
2
4
6


Clustered Data with K-means Clusters
0
1
2
Cluster Centers
Feature 2
Feature 1
10
5
0
−5
−10
−10
−8
−6
−4
−2
0
2
4
6

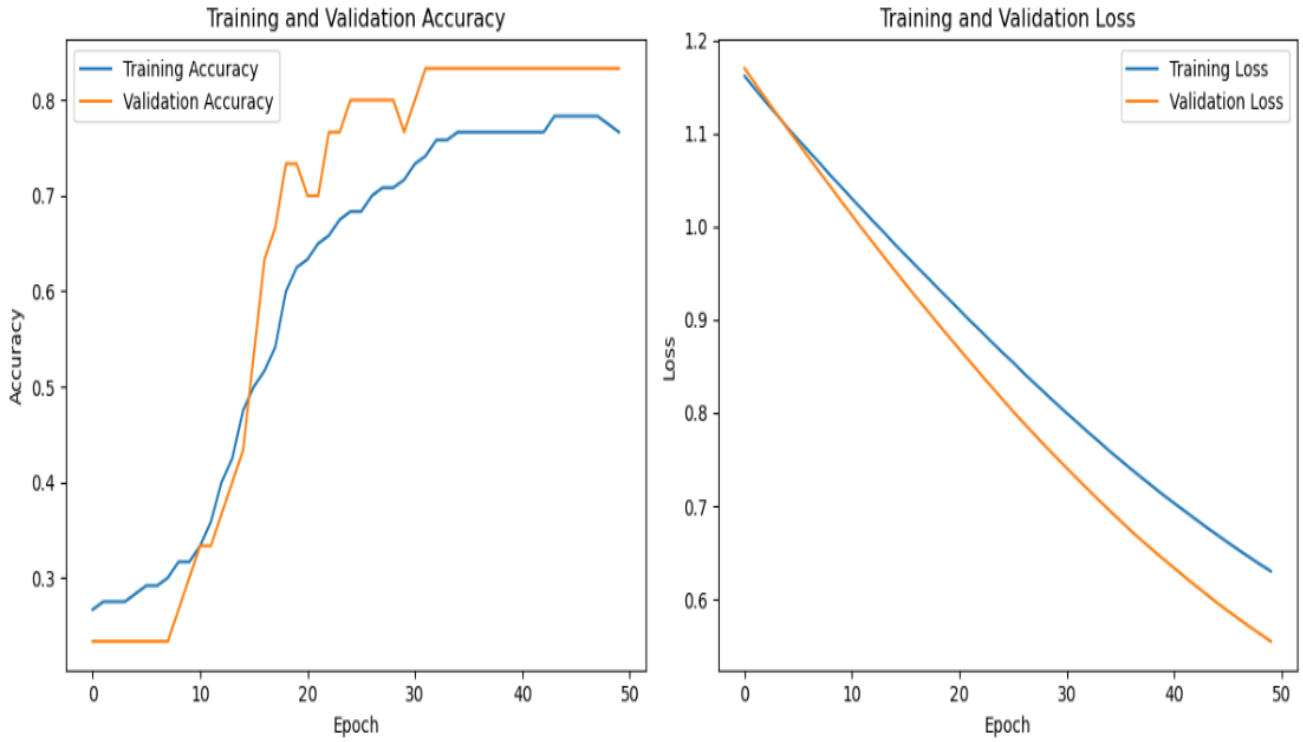
Q13. Creating and visualizing neural network for the given data.

Ans:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import plot_model

# Load the Iris dataset
iris = load_iris()

X = iris.data
y = iris.target

# Split the data 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)

# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a simple neural network model
model = Sequential([
    Dense(8, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(3, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train_scaled, y_train, epochs=50, validation_data=(X_test_scaled, y_test))

# Visualize the model architecture
plot_model(model, to_file='neural_network.png', show_shapes=True, show_layer_names=True)

# Visualize the training history (accuracy and loss over epochs)
plt.figure(figsize=(12, 5))

# Accuracy
plt.subplot(1, 2, 1)
```

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

Output:



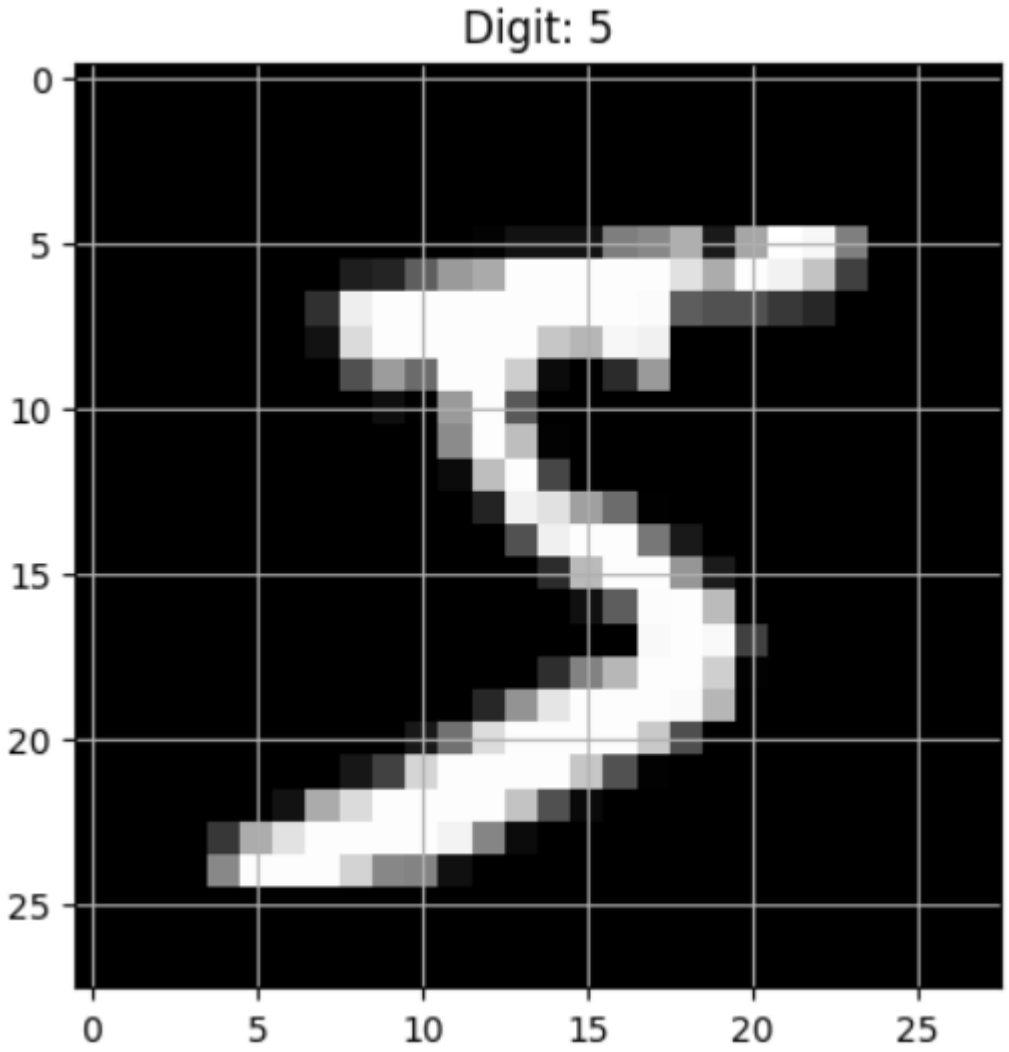
Q14. Recognize optical character using ANN.

Ans:

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalize pixel values to be between 0 and 1
x_train, x_test = x_train / 255.0, x_test / 255.0

# Display an example digit
plt.imshow(x_train[0], cmap='gray')
plt.title(f"Digit: {y_train[0]}")
plt.grid(True)  # Add gridlines to both x and y axes
plt.show()

# Flatten the images and one-hot encode the labels
x_train_flat = x_train.reshape((x_train.shape[0], -1))
x_test_flat = x_test.reshape((x_test.shape[0], -1))
y_train_one_hot = to_categorical(y_train)
y_test_one_hot = to_categorical(y_test)

# Create a simple ANN model
model = Sequential([
    Flatten(input_shape=(28, 28)),  # Flatten 28x28 images to a 1D array
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')  # Output layer with 10 neurons for 10 digits
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train_one_hot, epochs=5, batch_size=32, validation_data=(x_test,
y_test_one_hot))

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test, y_test_one_hot)
print(f"Test Accuracy: {test_accuracy}")
```

```
# Predictions on a few test samples
predictions = model.predict(x_test[:5])
predicted_labels = np.argmax(predictions, axis=1)
print(f"Predicted Labels: {predicted_labels}")
```

Output:




```
Predicted Labels: [7 2 1 0 4]
```

Q15. Write a program to implement CNN.

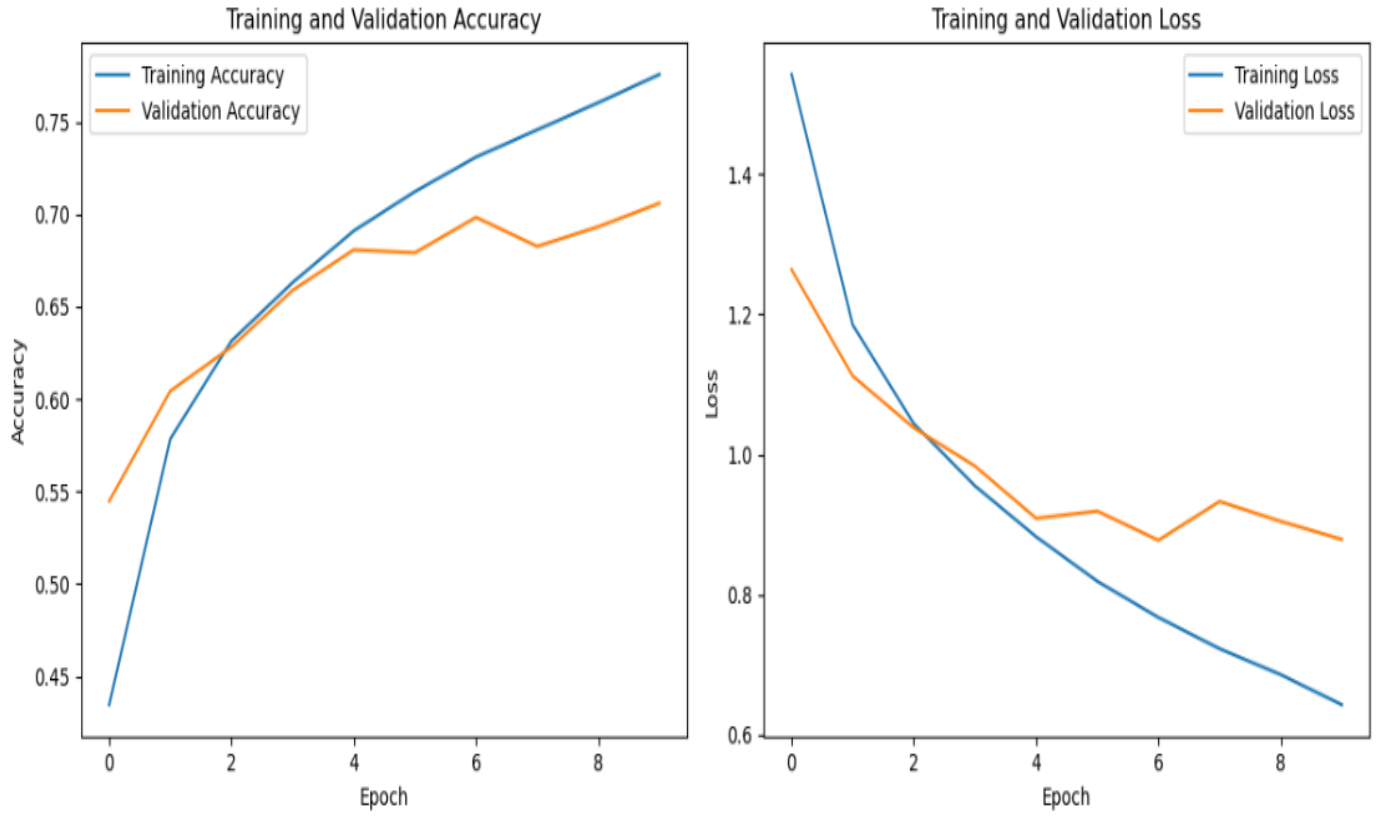
Ans:

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
import matplotlib.pyplot as plt

# Load and preprocess the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  # Normalize pixel values to be between 0 and 1
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Define the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Display the model architecture
model.summary()

# Train the model
history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"\nTest Accuracy: {test_accuracy}")
# Plot training history (accuracy and loss over epochs)
plt.figure(figsize=(12, 5))
# Accuracy
plt.subplot(1, 2, 1)
```

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

Output:

```
Test Accuracy: 0.7059999704360962
```



Q16. Write a program to implement RNN.

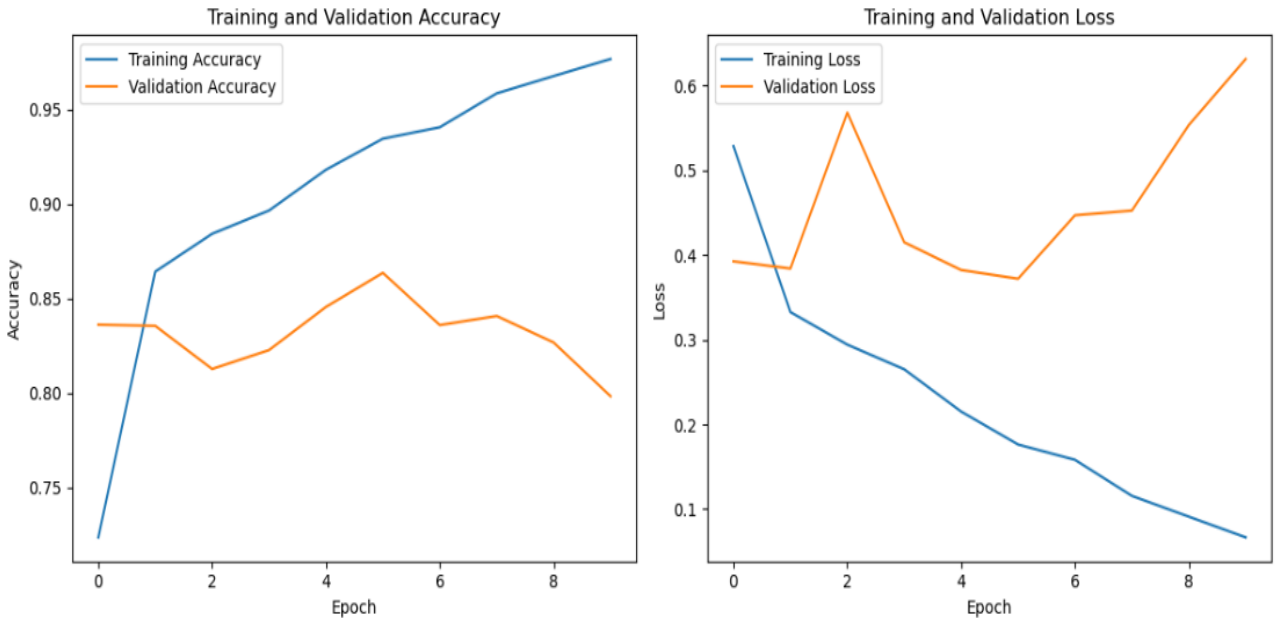
Ans:

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.preprocessing import sequence
import matplotlib.pyplot as plt

# Load and preprocess the IMDB dataset
max_features = 10000  # Number of most frequent words to consider
maxlen = 500  # Cuts off reviews after 500 words
batch_size = 32

print("Loading data...")
(input_train, y_train), (input_test, y_test) = imdb.load_data(num_words=max_features)
print(len(input_train), "train sequences")
print(len(input_test), "test sequences")

print("Pad sequences (samples x time)")
input_train = sequence.pad_sequences(input_train, maxlen=maxlen)
input_test = sequence.pad_sequences(input_test, maxlen=maxlen)
print("input_train shape:", input_train.shape)
print("input_test shape:", input_test.shape)

# Define the RNN model
model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])

# Display the model architecture
```

```
model.summary()

# Train the model
history = model.fit(input_train, y_train, epochs=10, batch_size=batch_size,
validation_split=0.2)

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(input_test, y_test)
print(f"\nTest Accuracy: {test_accuracy}")

# Plot training history (accuracy and loss over epochs)
plt.figure(figsize=(12, 5))

# Accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['acc'], label='Training Accuracy')
plt.plot(history.history['val_acc'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

Output:

```
Test Accuracy: 0.7980800271034241
```



Q17. Write a program to implement GAN.

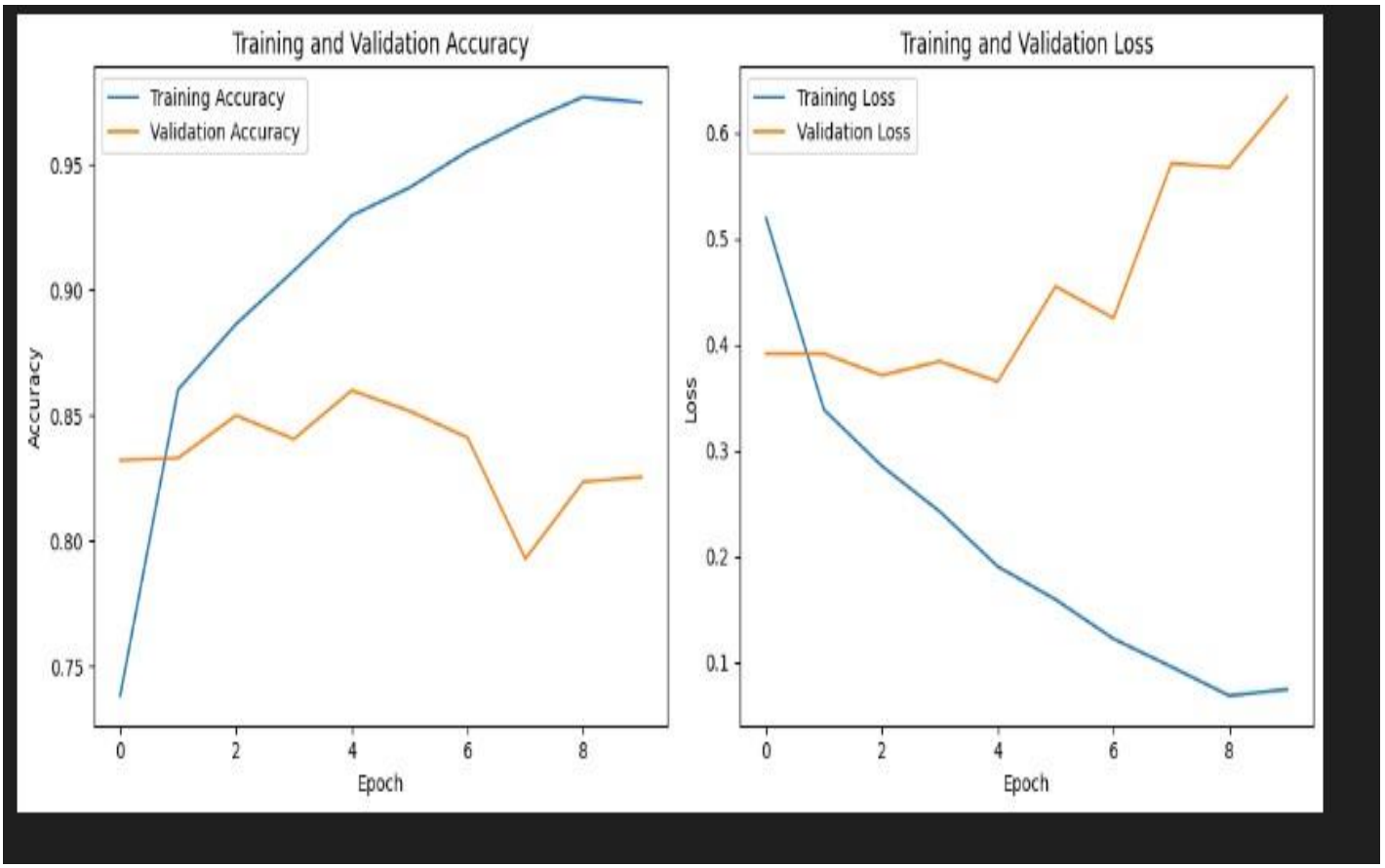
Ans:

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense, LeakyReLU, BatchNormalization, Reshape, Flatten,
Conv2DTranspose, Conv2D, Dropout, ZeroPadding2D, Input
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.datasets import mnist

# Load and preprocess the MNIST dataset
(x_train, _), (_, _) = mnist.load_data()
x_train = x_train / 127.5 - 1.0
x_train = x_train.reshape((x_train.shape[0], 28, 28, 1))

# Define the generator model
def build_generator(latent_dim):
    model = Sequential()
    model.add(Dense(128 * 7 * 7, input_dim=latent_dim))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Reshape((7, 7, 128)))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Conv2DTranspose(1, (4,4), strides=(2,2), padding='same', activation='tanh'))
    return model

# Define the discriminator model
def build_discriminator(img_shape):
    model = Sequential()
    model.add(Conv2D(32, (3,3), strides=(2,2), padding='same', input_shape=img_shape))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.25))
    model.add(Conv2D(64, (3,3), strides=(2,2), padding='same'))
    model.add(ZeroPadding2D(padding=((0,1),(0,1))))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.25))
    model.add(Conv2D(128, (3,3), strides=(2,2), padding='same'))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))
```

```
    model.add(Dropout(0.25))
    model.add(Conv2D(256, (3,3), strides=(1,1), padding='same'))
    model.add(BatchNormalization(momentum=0.8))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    return model


generator = build_generator(latent_dim)
generator.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5)) # Compile the
discriminator
discriminator = build_discriminator(img_shape)

discriminator.compile(loss='binary_crossentropy',    optimizer=Adam(0.0002,       0.5),
metrics=['accuracy'])
# Build and compile the combined model (stacked generator and discriminator)
discriminator.trainable = False
z = Input(shape=(latent_dim,))
img = generator(z)
validity = discriminator(img)
combined = Model(z, validity)
combined.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5)) # Training
parameters
epochs = 30000

batch_size = 64

half_batch = int(batch_size / 2)


for epoch in range(epochs):
   #
# Train Discriminator #
# Select a random half batch of images

  idx = np.random.randint(0, x_train.shape[0], half_batch)
imgs = x_train[idx]
noise = np.random.normal(0, 1, (half_batch, latent_dim)) # Generate a half batch of new images
gen_imgs = generator.predict(noise) # Train the discriminator

d_loss_real = discriminator.train_on_batch(imgs, np.ones((half_batch, 1)))
d_loss_fake = discriminator.train_on_batch(gen_imgs, np.zeros((half_batch, 1)))
d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
```

```
# Train Generator

noise = np.random.normal(0, 1, (batch_size, latent_dim))

# Train the generator (to have the discriminator label samples as valid) valid_y = np.array([1] *
batch_size)
g_loss = combined.train_on_batch(noise, valid_y) # Print the progress
if epoch % 1000 == 0:

    print(f"{epoch} [D loss: {d_loss[0]} | D accuracy: {100 * d_loss[1]}] [G loss: {g_loss}]") #
Save generated images at specified intervals
if epoch % save_interval == 0:
    save_generated_images(epoch)
```

Output:



Q18. Web scraping experiments(by using tools).
Ans:

```
from urllib.request import urlopen

html = urlopen("http://geeksforgeeks.org")
print(html.read())
```

Output:

```
b'\n<!DOCTYPE html>\r\n<!--[if IE 7]>\r\n<html class="ie ie7" lang="en-US" prefix="og: http://ogp.me/ns#">\r\n<![endif]-->\r
\n<!--[if IE 8]>\r\n<html class="ie ie8" lang="en-US" prefix="og: http://ogp.me/ns#">\r\n<![endif]-->\r\n<!--[if !(IE 7) | !
(IE 8)  ]><!-->\r\n<html lang="en-US" prefix="og: http://ogp.me/ns#" >\r\n\r\n<!--<![endif]-->\r\n<head>\r\n<meta charset="UT
F-8" />\r\n<meta name="keywords" content="Data Structures, Algorithms, Python, Java, C, C++, JavaScript, Android Development,
SQL, Data Science, Machine Learning, PHP, Web Development, System Design, Tutorial, Technical Blogs, Interview Experience, In
terview Preparation, Programming, Competitive Programming, SDE Sheet, Job-a-thon, Coding Contests, GATE CSE, HTML, CSS, Reac
t, NodeJS, Placement, Aptitude, Quiz, Computer Science, Programming Examples, GeeksforGeeks Courses, Puzzles">\r\n<meta name
="viewport" content="width=device-width, initial-scale=1.0, minimum-scale=0.5, maximum-scale=3.0"> \r\n<meta property="og:des
cription" name="description" content="A Computer Science portal for geeks. It contains well written, well thought and well ex
plained computer science and programming articles, quizzes and practice/competitive programming/company interview Question
s."><meta property="og:url" content="https://www.geeksforgeeks.org/" /><meta name="verify-admitad" content="d656723ea9" />\r
\n<link rel="shortcut icon" href="https://media.geeksforgeeks.org/wp-content/cdn-uploads/gfg_favicon.png" type="image/x-icon"
/>\r\n<link rel="preconnect" href="https://fonts.googleapis.com">\r\n<link rel="preconnect" href="https://fonts.gstatic.com"
crossorigin>\r\n\r\n<meta name="theme-color" content="#308D46" />\r\n\r\n<meta name="image" property="og:image" content="http
s://media.geeksforgeeks.org/wp-content/cdn-uploads/gfg_200x200-min.png">\r\n<meta property="og:image:type" content="image/pn
g">\r\n<meta property="og:image:width" content="200">\r\n<meta property="og:image:height" content="200">\r\n<meta name="faceb
ook-domain-verification" content="xo7t4ve2wn3ywfkjdvwbrk01pvdond" />\r\n\r\n<script defer src="https://apis.google.com/js/pla
tform.js"></script>\r\n<script async src="//cdnjs.cloudflare.com/ajax/libs/require.js/2.1.14/require.min.js"></script>\r\n<!-
```

```
# importing BeautifulSoup from bs4 module
from bs4 import BeautifulSoup
# importing requests
import requests

# get URL
r = requests.get("https://www.geeksforgeeks.org")
data = r.text
soup = BeautifulSoup(data, 'html.parser')

# Find and print all the links on the page
for link in soup.find_all('a'):
    print(link.get('href'))
```

Output: