Since 2011

RIG
Realistic Infotech Group

Our outcomes are over 5000 trainees.

Get IT Right from RIG

Artificial Intelligence Engineering (Level-1)

Learn, Create, and Shine

Digital Space

Level-1

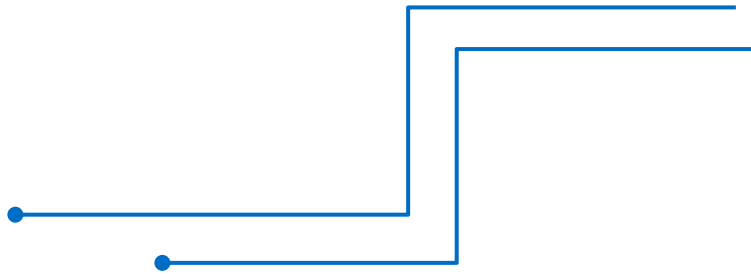
- Module 1: Introduction to AI and Machine Learning
- Module 2: Linear Algebra, Statistics and Probability for AI
- Module 3: Neural Network Architecture
- Module 4: Building Machine Learning Models
- Module 5: Deep Learning Concepts
- Module 6: Python Data Structure
- Module 7: Data Handling with Pandas and NumPy
- Module 8: Python for AI
- Module 9: Classification AI Project
- Module 10: Prediction AI Project

Artificial Intelligence Engineering (Level-1)

Module 8: Python for AI

Content

- scikit-learn Package
- tensorflow Package
- Matplotlib Package
- How to implement Neural Network Classifier?



Learning Outcomes

- Understand and apply the **Scikit-learn package** for machine learning tasks.
- Utilize the **TensorFlow package** to build and train deep learning models.
- Leverage the **Matplotlib package** for data visualization and analysis.
- Implement a **Neural Network Classifier** for classification problems using appropriate frameworks.

Scikit-learn Package

Powerful open-source machine learning library in Python

scikit-learn Package (sklearn)

- ✓ Scikit-learn is a widely used open-source machine learning library in Python.
- ✓ Provide simple and efficient tools for data mining and machine learning.
- ✓ Scikit-learn is built on top of foundational Python libraries like NumPy, SciPy, and matplotlib.
- ✓ It is essential for tasks like classification, regression, clustering, and dimensionality reduction.

https://scikit-learn.org/

Why use scikit-learn?

- ✓ **Comprehensive Functionality**
  - Includes a wide range of supervised and unsupervised learning algorithms.
  - Offers tools for feature extraction, model selection, and evaluation.
- ✓ **Ease of Use**
  - Simple and consistent API, making it beginner-friendly.
  - Extensive documentation and community support.

Why use scikit-learn?

- **Integration**
  - Works seamlessly with Python's data science stack (pandas, NumPy).
  - Can integrate with other machine learning frameworks or libraries like TensorFlow or PyTorch.

- **Efficiency**
  - Optimized for performance, especially when using NumPy and SciPy for numerical operations.
- **Flexibility**
  - Allows customization and extension for advanced users.

Features of scikit-learn

Supervised Learning

- Algorithms for classification (SVM, Random Forest, Logistic Regression).
- Algorithms for regression (Linear Regression, Ridge Regression).
- Use case: Predicting house prices or spam email classification.

Unsupervised Learning

- Algorithms for clustering (K-Means, DBSCAN).
- Techniques for dimensionality reduction (PCA, t-SNE).
- Use case: Customer segmentation or reducing dataset dimensionality.

Model Evaluation

- Metrics like accuracy, precision, recall, F1 score, ROC-AUC.
- Cross-validation for better model performance insights.
- Use case: Validating a machine learning model before deployment.

Features of scikit-learn

Preprocessing Tools

- Tools for scaling (StandardScaler, MinMaxScaler).
- Handling missing values and encoding categorical variables.
- Use case: Preparing raw data for machine learning.

Pipeline Creation

- Automates workflows by chaining preprocessing and model steps.
- Use case: Building a robust pipeline for production-ready ML.

Feature Selection

- Methods to identify important features.
- Use case: Reducing the number of features for faster computation.

Dependencies

Python (>= 3.9)

NumPy (>= 1.19.5)

SciPy (>= 1.6.0)

joblib (>= 1.2.0)

threadpoolctl (>= 3.1.0)

How to install scikit-learn?

```
pip install scikit-learn
```

```
C:\Users\ECS>pip install scikit-learn
Defaulting to user installation because normal site-packages is not writeable
Collecting scikit-learn
  Downloading scikit_learn-1.6.0-cp312-cp312-win_amd64.whl.metadata (15 kB)
Collecting numpy>=1.19.5 (from scikit-learn)
  Downloading numpy-2.2.0-cp312-cp312-win_amd64.whl.metadata (60 kB)
Collecting scipy>=1.6.0 (from scikit-learn)
  Downloading scipy-1.14.1-cp312-cp312-win_amd64.whl.metadata (60 kB)
Collecting joblib>=1.2.0 (from scikit-learn)
  Downloading joblib-1.4.2-py3-none-any.whl.metadata (5.4 kB)
Collecting threadpoolctl>=3.1.0 (from scikit-learn)
  Downloading threadpoolctl-3.5.0-py3-none-any.whl.metadata (13 kB)
Downloading scikit_learn-1.6.0-cp312-cp312-win_amd64.whl (11.1 MB)
   ---------------------------------------- 11.1/11.1 MB 34.6 MB/s eta 0:00:00
Downloading joblib-1.4.2-py3-none-any.whl (301 kB)
Downloading numpy-2.2.0-cp312-cp312-win_amd64.whl (12.6 MB)
   ---------------------------------------- 12.6/12.6 MB 49.4 MB/s eta 0:00:00
Downloading scipy-1.14.1-cp312-cp312-win_amd64.whl (44.5 MB)
   ---------------------------------------- 44.5/44.5 MB 56.7 MB/s eta 0:00:00
Downloading threadpoolctl-3.5.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, numpy, joblib, scipy, scikit-learn
  WARNING: The scripts f2py.exe and numpy-config.exe are installed in 'C:\Users\ECS\AppData\Local\Packages\PythonSoftwar
eFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed joblib-1.4.2 numpy-2.2.0 scikit-learn-1.6.0 scipy-1.14.1 threadpoolctl-3.5.0
```

Scikit-learn Examples

Iris Dataset

- ✓ **Predicted attribute: class of iris plant.**
- ✓ **Features(columns)**
  - ✓ **sepal length (cm)**
  - ✓ **sepal width (cm)**
  - ✓ **petal length (cm)**
  - ✓ **petal width (cm)**
- ✓ **The target column contains the class labels for the flower species:**
  - ✓ **0: Iris-Setosa**
  - ✓ **1: Iris-Versicolour**
  - ✓ **2: Iris-Virginica**

1. Loading a Built-in Dataset (Iris Dataset)

```
from sklearn.datasets import load_iris
import pandas as pd

# Load the Iris dataset
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# Display the first few rows
print(df.head())

# Save as CSV
df.to_csv('iris_dataset.csv', index=False)
print("Dataset saved as 'iris_dataset.csv'")
```

1. Output

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0       0
1       0
2       0
3       0
4       0
```

2. Splitting Data into Training and Testing Sets

```
from sklearn.model_selection import train_test_split

# Features (X) and target (y)
X = df.drop('target', axis=1)
y = df['target']

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

print("Training set shape:", X_train.shape)
print("Testing set shape:", X_test.shape)
```

2. Output

```
Training set shape: (120, 4)
Testing set shape: (30, 4)
```

Feature (X) , Target (y)

- ✓ **Feature (X)**
  - ✓ **Creates the feature matrix X by dropping the target column from the DataFrame.**
  - ✓ **Input Variables**
- ✓ **Target (y)**
  - ✓ **Creates the target vector y by extracting the target column from the DataFrame.**
  - ✓ **y contains the class labels.**

Split Training & Testing Dataset

- ✓ **Splits the dataset into training and testing subsets.**
- ✓ **test_size=0.2**
  - ✓ **Specifies that 20% of the data should be used for testing, and the remaining 80% for training.**
- ✓ **random_state=42**
  - ✓ **Sets a random seed for reproducibility, ensuring that the split is the same every time the code runs**

Logistic Regression

```
class LogisticRegression(
    penalty: Literal['l1', 'l2', 'elasticnet'] | None = "l2",
    *,
    dual: bool = False,
    tol: Float = 0.0001,
    C: Float = 1,
    fit_intercept: bool = True,
    intercept_scaling: Float = 1,
    class_weight: Mapping | str | None = None,
    random_state: Int | RandomState | None = None,
    solver: Literal['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga'] = "lbfgs",
    max_iter: Int = 100,
    multi_class: Literal['auto', 'ovr', 'multinomial'] = "auto",
    verbose: Int = 0,
    warm_start: bool = False,
    n_jobs: Int | None = None,
    l1_ratio: Float | None = None
)
```

3. Building a Simple Classifier (Logistic Regression)

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix

# Create and train the model
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

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

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test,
y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

3. Output

```
Accuracy: 1.0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30


Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

3.1 Create and Train the Model

- ✓ **Creates a logistic regression model from sklearn.linear_model.**
  - ✓ **The max_iter=200 parameter**
  - ✓ **sets the maximum number of iterations for the solver to converge (find the optimal parameters)**
  - ✓ **Default value -100**
- ✓ **Trains the logistic regression model on the training data:**
  - ✓ **X_train: The training feature matrix.**
  - ✓ **y_train: The training target vector (labels).**

3.2 Make Predictions

- ✓ **Uses the trained model to make predictions on the test feature set (X_test).**
  - ✓ **The result (y_pred) is a vector of predicted labels for the test set.**

3.3 Evaluate the Model

- ✓ **Accuracy**
  - ✓ **accuracy_score(y_test, y_pred):**
  - ✓ **Calculates the accuracy of the model, which is the ratio of correctly predicted instances to the total number of instances**
  - ✓ **Accuracy = Number of Correct Predictions/Total Number of Predictions**
- ✓ **Classification Report**
  - ✓ **y_test: The true labels of the test set.**
  - ✓ **y_pred: The predicted labels from the model.**
  - ✓ **Precision: Proportion of true positives out of all predicted positives for each class.**
  - ✓ **Recall: Proportion of true positives out of all actual positives for each class.**
  - ✓ **F1-Score: Harmonic mean of precision and recall for each class.**
  - ✓ **Support: Number of actual instances for each class in the test set.**

3.3 Evaluate the Model

- **Confusion_matrix(y_test, y_pred)**
  - **Creates a matrix that compares true labels (y_test) to predicted labels (y_pred).**
  - **Each cell (i,j) indicates the count of instances where the true label was class**
  - **i and the predicted label was class j.**

(class) StandardScaler

- ✓ Standardize features by removing the mean and scaling to unit variance.
- ✓ The standard score of a sample x is calculated as:
- ✓ z = (x - u) / s
  - ✓ `u` is the mean of the training samples or zero if `with_mean=False`,
  - ✓ `s` is the standard deviation of the training samples or one if `with_std=False`.

- ✓ Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using :meth:`transform`.
- ✓ 'StandardScaler' is sensitive to outliers, and the features may scale differently from each other in the presence of outliers.

4. Feature Scaling (Standardization)

```
from sklearn.preprocessing import StandardScaler

# Initialize the scaler
scaler = StandardScaler()

# Fit and transform the training data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Print the first 5 rows of the scaled training data
print(X_train_scaled[:5])
```

4. Output

```
[[-1.47393679  1.20365799 -1.56253475 -1.31260282]
 [-0.13307079  2.99237573 -1.27600637 -1.04563275]
 [ 1.08589829  0.08570939  0.38585821  0.28921757]
 [-1.23014297  0.75647855 -1.2187007  -1.31260282]
 [-1.7177306   0.30929911 -1.39061772 -1.31260282]]
```

4.1 Initialize the Scaler

- **StandardScaler()**
  - **Creates an instance of the StandardScaler.**
  - **Standardizes the features by removing the mean and scaling them to unit variance:**
- $Z = (X - \mu) / \sigma$
  - X: **Original feature value.**
  - μ: **Mean of the feature values in the training set.**
  - σ: **Standard deviation of the feature values in the training set.**

4.2 Fit and Transform the Training Data

- ✓ **scaler.fit_transform(X_train)**
  - ✓ **fit: Computes the mean (μ) and standard deviation (σ) of each feature in the training set.**
  - ✓ **transform: Applies the computed scaling to transform the training data**
- ✓ **X_scaled =(X_original $-\mu$)/ σ**

4.3 Transform the Testing Data

- ✓ **The transformed training data (X_train_scaled) is now standardized.**
- ✓ **scaler.transform(X_test)**
  - ✓ **to scale the testing data (X_test).**
  - ✓ **Why not fit again? To avoid data leakage. The testing data must not influence the scaling parameters.**
  - ✓ **Displays the first 5 rows of the standardized training data. Each value represents the standardized version of a feature in the training set.**

class GridSearchCV

```
class GridSearchCV(
    estimator: LogisticRegression,
    param_grid: Mapping | Sequence[dict],
    *,
    scoring: ArrayLike | tuple | ((...) -> Any) | Mapping | None = None,
    n_jobs: Int | None = None,
    refit: str | ((...) -> Any) | bool = True,
    cv: int | BaseCrossValidator | Iterable | BaseShuffleSplit | None = None,
    verbose: Int = 0,
    pre_dispatch: str | int = "2*n_jobs",
    error_score: Float | str = ...,
    return_train_score: bool = False
)
```

5. Hyperparameter Tuning using Grid Search

```
from sklearn.model_selection import GridSearchCV

# Define hyperparameters for tuning
param_grid = {'C': [0.1, 1, 10], 'solver': ['liblinear', 'lbfgs']}
grid_search = GridSearchCV(LogisticRegression(max_iter=200),
param_grid, cv=5)

# Train with different hyperparameters
grid_search.fit(X_train_scaled, y_train)

print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Score:", grid_search.best_score_)
```

5. Output

```
Best Parameters: {'C': 1, 'solver': 'lbfgs'}
Best Cross-Validation Score: 0.9583333333333334
```

5.1 Define Hyperparameters for Tuning

- ✓ **param_grid**
  - ✓ **Specifies the hyperparameters to test during the grid search.**

- ✓ **C**
  - ✓ **Regularization strength parameter for logistic regression.**
  - ✓ **Lower values (e.g., 0.1) mean stronger regularization, which helps prevent overfitting but can underfit the model.**
  - ✓ **Higher values (e.g., 10) reduce regularization, allowing the model to fit the data more closely.**

- ✓ **solver**
  - ✓ **Optimization algorithms for logistic regression.**
  - ✓ **liblinear: Suitable for small datasets and supports**
  - ✓ **$L1$ regularization.**
  - ✓ **lbfgs: Recommended for larger datasets and supports**
  - ✓ **$L2$ regularization.**

5.2 Initialize Grid Search

- **GridSearchCV**
  - **Performs an exhaustive search over the specified hyperparameter grid (param_grid).**
  - **Trains the model using each combination of hyperparameters and evaluates performance using cross-validation.**
- **Parameters**

  - **LogisticRegression(max_iter=200):** The logistic regression model to tune.
  - **param_grid:** Hyperparameter grid to search over.
  - **cv=5: Performs 5-fold cross-validation, splitting the training data into 5 subsets (folds):**
  - The model is trained on 4 folds and validated on the 5th.
  - This process is repeated 5 times, using a different fold for validation each time.

5.3 Train the Model with Grid Search

- ✓ **Trains the logistic regression model for every combination of hyperparameters in param_grid.**
- ✓ **For each combination, it performs 5-fold cross-validation to evaluate the model's performance.**
- ✓ **Selects the combination of hyperparameters that achieves the best cross-validation score.**

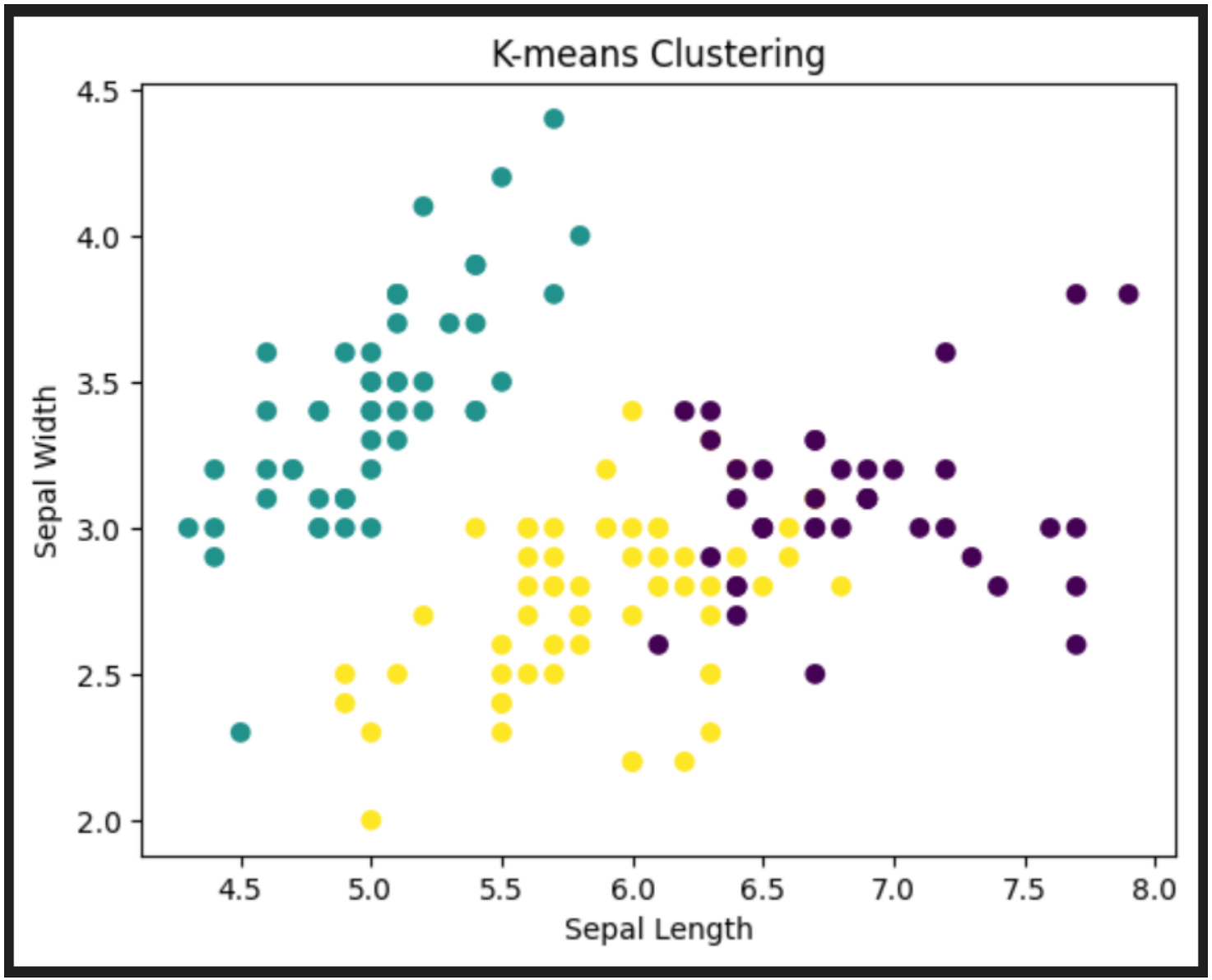
5.4 Print Best Hyperparameters and Cross-Validation Score

- ✓ **grid_search.best_params_**
  - ✓ **Returns the hyperparameters that achieved the best cross-validation score.**
- ✓ **grid_search.best_score_**
  - ✓ **The highest cross-validation score obtained during grid search.**
  - ✓ **Indicates how well the model performs with the best combination of hyperparameters.**

6. Clustering with K-means (Unsupervised Learning)

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Apply K-means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Add cluster labels to the original data
df['cluster'] = kmeans.labels_
# Plot the clusters
plt.scatter(df['sepal length (cm)'], df['sepal width (cm)'],
c=df['cluster'])
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('K-means Clustering')
plt.show()
```

6. Output


K-means Clustering
Sepal Width
Sepal Length
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

6.1 Apply K-means Clustering

- KMeans
  - A clustering algorithm that groups data points into a specified number of clusters (k).
  - It minimizes the sum of squared distances between data points and their cluster centroids.
- Parameters
  - n_clusters=3:
  - Specifies the number of clusters (groups) to create. For the Iris dataset, there are 3 natural groups (species), so k =3.
- random_state=42
  - Ensures reproducibility by fixing the random seed for centroid initialization.
- kmeans.fit(X)
  - Runs the K-means algorithm on the feature matrix X.
  - Finds the cluster centroids and assigns each data point to a cluster.

6.2 Add Cluster Labels to the Original Data

- ✓ **kmeans.labels_**
  - ✓ **An array containing the cluster assignments for each data point.**
  - ✓ **For example, if the dataset has 150 samples, this array will contain 150 cluster labels (e.g., 0, 1, or 2).**
- ✓ **df['cluster']**
  - ✓ **Adds a new column, cluster, to the DataFrame df, storing the cluster assignment for each sample.**

6.3 Plot the Clusters

- plt.scatter
  - Creates a scatter plot of the data points.
  - X-axis: Sepal length.
  - Y-axis: Sepal width.
  - c=df['cluster']: Colors the points based on their cluster assignment (cluster column).

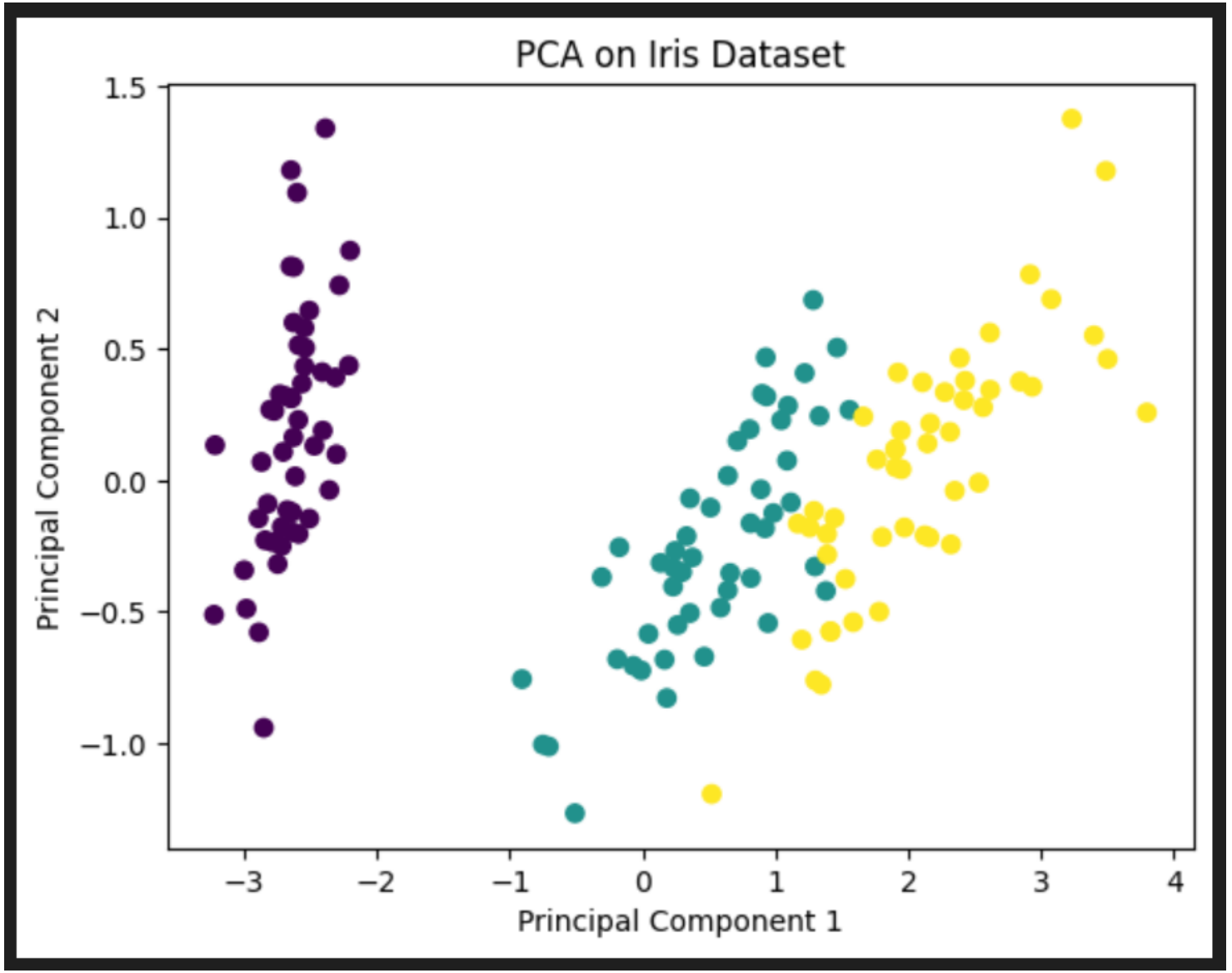
6.4 Add Labels and Title

- ✓ **Adds axis labels (Sepal Length, Sepal Width) and a title for context.**
- ✓ **plt.show(): Displays the scatter plot.**

7. Principal Component Analysis (PCA) for Dimensionality Reduction

```
from sklearn.decomposition import PCA

# Initialize PCA and reduce to 2 components

pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

# Plot the reduced data
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=y)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA on Iris Dataset')
plt.show()
```

7. Output


PCA on Iris Dataset
Principal Component 2
Principal Component 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−3
−2
−1
0
1
2
3
4

7.1 PCA Initialization

- **PCA**
  - **PCA (Principal Component Analysis) is a technique for reducing the dimensionality of a dataset while retaining as much variance as possible.**
  - **It identifies the principal components (directions of maximum variance) in the data.**
- **n_components=2**
  - **Specifies that the dataset should be reduced to 2 principal components (from its original number of features).**
  - **This is useful for visualizing high-dimensional data in a 2D space.**

7.2 Fit and Transform the Data

- ✓ **pca.fit_transform(X)**
  - ✓ **fit: Learns the principal components from the feature matrix X.**
  - ✓ **transform: Projects the original data into the 2D space defined by the first two principal components.**
- ✓ **X_reduced**
  - ✓ **A 2D array where each row is a data point, and the two columns are the new coordinates in the reduced 2D space.**

7.3 Plot the Reduced Data

- ✓ **X_reduced[:, 0] and X_reduced[:, 1]**
  - ✓ **Access the first and second principal components, which serve as the x and y coordinates in the plot.**
- ✓ **c=y**
  - ✓ **Colors the points based on their class labels (y), so different classes are visually distinguishable.**

7.4 Label and Display the Plot

- ✓ **Adds axis labels (Principal Component 1, Principal Component 2) and a title for context.**
- ✓ **plt.show(): Displays the scatter plot.**

8. Model Evaluation Metrics

```
from sklearn.metrics import precision_score, recall_score, f1_score

# Calculate evaluation metrics
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Precision: {precision:.4f}, Recall: {recall:.4f}, F1 Score: {f1:.4f}")
```

8. Output

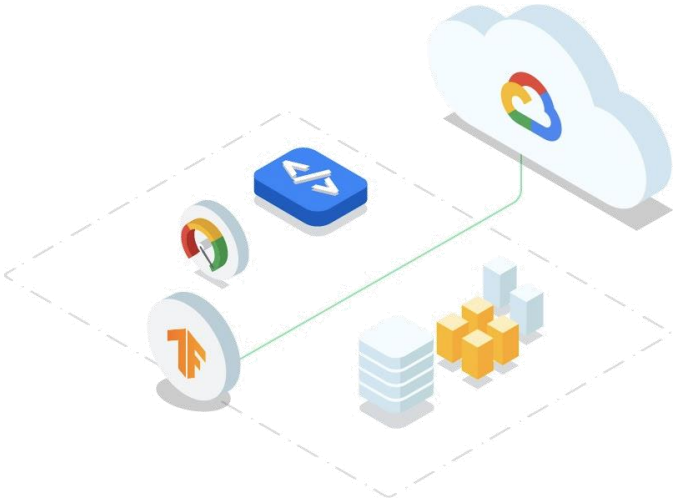
```
Precision: 1.0000, Recall: 1.0000, F1 Score: 1.0000
```

Tensorflow Package

Popular open-source deep learning framework developed by Google

tensorflow Package

- The TensorFlow package is a widely used open-source library developed by **Google for machine learning** and deep learning applications.
- It is known for its versatility and scalability, allowing developers to build and deploy machine learning models across various platforms, from mobile devices to distributed cloud computing.



Why use Tensorflow?

- **Comprehensive Machine Learning Platform**
  - TensorFlow supports the entire machine learning workflow
  - Data preparation and preprocessing.
  - Model building and training.
  - Deployment on various platforms, including cloud, edge devices, and web browsers.
- **Flexibility**
  - TensorFlow offers both high-level APIs (Keras) for quick prototyping and low-level APIs for custom machine learning tasks.
  - Supports advanced use cases like custom loss functions, layers, and operations.

Why use Tensorflow?

- **Scalability**
  - TensorFlow allows training on CPUs, GPUs, TPUs, and distributed systems with ease.
  - Optimized for large-scale data and computationally intensive tasks.
- **Versatility**
  - Deep learning: Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformers.
  - Traditional machine learning: Gradient Boosting, Random Forest, etc., through its integrated tools.
  - Reinforcement learning and unsupervised learning.

CPUs, GPUs, TPUs

Feature	CPU	GPU	TPU
Core Count	Few cores (4–64)	Hundreds to thousands	Fixed matrix units
Optimization	General-purpose tasks	Parallel processing	Tensor/matrix operations
Clock Speed	High	Moderate	Moderate
Use Cases	General computing, OS	Graphics, ML training/inference	ML training/inference (TensorFlow)
Efficiency	Low for ML	High for ML	Highest for ML (specific models)

Why use Tensorflow?

- **Deployment**
  - TensorFlow models can be deployed across:
  - Mobile and Edge Devices: TensorFlow Lite.
  - Web Browsers: TensorFlow.js.
  - Cloud: TensorFlow Serving and TensorFlow Extended (TFX).
- **Active Ecosystem**
  - Extensive libraries like TensorFlow Datasets, TensorFlow Hub, and TensorFlow Probability for specialized needs.
  - Vibrant community and regular updates.

Features of Tensorflow

High-Level Model Building with Keras

- Keras is integrated into TensorFlow for building and training models in a modular and user-friendly manner.
- Quickly create and train a neural network using tf.keras.Sequential.

Deep Learning Support

- State-of-the-art deep learning techniques like CNNs, RNNs, and Transformers.
- **Example**: Image recognition, natural language processing (NLP), and speech recognition.

Tensor Operations

- Low-level operations for matrix manipulations and custom computations.
- Implementing advanced research models.

Features of Tensorflow

Scalability

- Distributed training support for scaling to massive datasets.
- Training large models on multiple GPUs or TPUs.

Pre-trained Models

- Use pre-trained models from TensorFlow Hub for tasks like object detection, image classification, and text embeddings.
- Transfer learning for faster training and better results.

Data Handling

- Utilities for efficient data loading and preprocessing.
- TensorFlow Datasets for accessing standardized datasets.
- Handling large-scale image or text datasets.

Features of Tensorflow

TensorFlow Lite

- For deploying machine learning models on mobile and IoT devices.
- Real-time object detection on Android.

TensorFlow.js

- Run machine learning models directly in a browser or Node.js.
- Interactive web-based ML applications.

TensorFlow Extended (TFX)

- A complete platform for end-to-end ML pipelines, including data validation, model analysis, and serving.
- Enterprise-level ML workflows.

How to install tensorflow?

```
pip install tensorflow
```

```
pip install tensorflow
[1]  ✓ 1m 22.7s                                                                    Python
...  Collecting tensorflow
       Downloading tensorflow-2.18.0-cp310-cp310-win_amd64.whl (7.5 kB)
     Collecting tensorflow-intel==2.18.0
       Downloading tensorflow_intel-2.18.0-cp310-cp310-win_amd64.whl (390.0 MB)
          ---------------------------------------- 0.0/390.0 MB ? eta -:--:--
          ---------------------------------------- 0.7/390.0 MB 15.5 MB/s eta 0:00:26
          ---------------------------------------- 3.2/390.0 MB 28.9 MB/s eta 0:00:14
           ---------------------------------------- 6.6/390.0 MB 42.5 MB/s eta 0:00:10
           ---------------------------------------- 9.9/390.0 MB 48.8 MB/s eta 0:00:08
          - -------------------------------------- 13.1/390.0 MB 59.5 MB/s eta 0:00:07
          - -------------------------------------- 16.4/390.0 MB 65.2 MB/s eta 0:00:06
          - -------------------------------------- 18.8/390.0 MB 59.5 MB/s eta 0:00:07
          -- ------------------------------------- 22.0/390.0 MB 59.8 MB/s eta 0:00:07
          -- ------------------------------------- 25.6/390.0 MB 65.6 MB/s eta 0:00:06
          -- ------------------------------------- 29.3/390.0 MB 73.1 MB/s eta 0:00:05
          --- ------------------------------------ 32.7/390.0 MB 72.6 MB/s eta 0:00:05
          --- ------------------------------------ 35.8/390.0 MB 72.6 MB/s eta 0:00:05
          --- ------------------------------------ 39.4/390.0 MB 59.8 MB/s eta 0:00:06
          ---- ----------------------------------- 42.6/390.0 MB 59.5 MB/s eta 0:00:06
          ---- ----------------------------------- 46.3/390.0 MB 65.6 MB/s eta 0:00:06
          ---- ----------------------------------- 49.5/390.0 MB 59.5 MB/s eta 0:00:06
          ----- ---------------------------------- 53.1/390.0 MB 65.6 MB/s eta 0:00:06
          ----- ---------------------------------- 56.3/390.0 MB 65.6 MB/s eta 0:00:06
          ----- ---------------------------------- 57.3/390.0 MB 59.5 MB/s eta 0:00:06
          ----- ---------------------------------- 59.1/390.0 MB 50.4 MB/s eta 0:00:07
     ...
       Downloading mdurl-0.1.2-py3-none-any.whl (10.0 kB)
     Installing collected packages: namex, libclang, flatbuffers, wrapt, wheel, urllib3, termcolor, tensorflow-io-gcs-filesystem, tensorboard-data-server, protobuf, optree, opt-e
     Successfully installed MarkupSafe-3.0.2 absl-py-2.1.0 astunparse-1.6.3 certifi-2024.12.14 charset-normalizer-3.4.0 flatbuffers-24.3.25 gast-0.6.0 google-pasta-0.2.0 grpcio-1
     Note: you may need to restart the kernel to use updated packages.
```

Note: TensorFlow does not support Python versions earlier than **3.8** or later than **3.11** (as of December 2024).

Tensorflow Examples

1. Importing TensorFlow

```
from sklearn.metrics import precision_score, recall_score, f1_score
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
```

Output

```
TensorFlow version: 2.18.0
```

2. Building a Simple Neural Network (MNIST Dataset)

Step 1: Load the Dataset

```
from tensorflow.keras.datasets import mnist

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

# Normalize the pixel values (0-255) to (0-1)
x_train, x_test = x_train / 255.0, x_test / 255.0

# Check the shapes of the datasets
print("Training data shape:", x_train.shape)
print("Test data shape:", x_test.shape)
```

Step 1: Load the Dataset

Output

```
Training data shape: (60000, 28, 28)
Test data shape: (10000, 28, 28)
```

Step 2: Build the Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense

# Define the model architecture
model = Sequential([
    Flatten(input_shape=(28, 28)),       # Flatten the 28x28 images to a 1D array
    Dense(128, activation='relu'),       # Hidden layer with 128 neurons
    Dense(10, activation='softmax')      # Output layer for 10 classes (digits 0-9)
])

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

Step 2: Build the Model
Output

```
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
flatten_2 (Flatten)	(None, 784)	0
dense_4 (Dense)	(None, 128)	100,480
dense_5 (Dense)	(None, 10)	1,290

```
 Total params: 101,770 (397.54 KB)

 Trainable params: 101,770 (397.54 KB)

 Non-trainable params: 0 (0.00 B)
```

Step 3: Train the Model

```
# Train the model
history = model.fit(x_train, y_train, epochs=5, validation_split=0.1)
```

Step 4: Evaluate the Model

```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_accuracy:.4f}")
```

Step 3: Train the Model

Output

```
Epoch 1/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 14s 5ms/step - accuracy: 0.8679 - loss: 0.4667 - val_accuracy: 0.9583 - val_loss: 0.1469
Epoch 2/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9603 - loss: 0.1347 - val_accuracy: 0.9700 - val_loss: 0.0987
Epoch 3/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.9743 - loss: 0.0873 - val_accuracy: 0.9735 - val_loss: 0.0918
Epoch 4/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9811 - loss: 0.0639 - val_accuracy: 0.9755 - val_loss: 0.0841
Epoch 5/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9871 - loss: 0.0440 - val_accuracy: 0.9768 - val_loss: 0.0815
```

Step 4: Evaluate the Model

Output

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9711 - loss: 0.0894
Test Accuracy: 0.9755
```

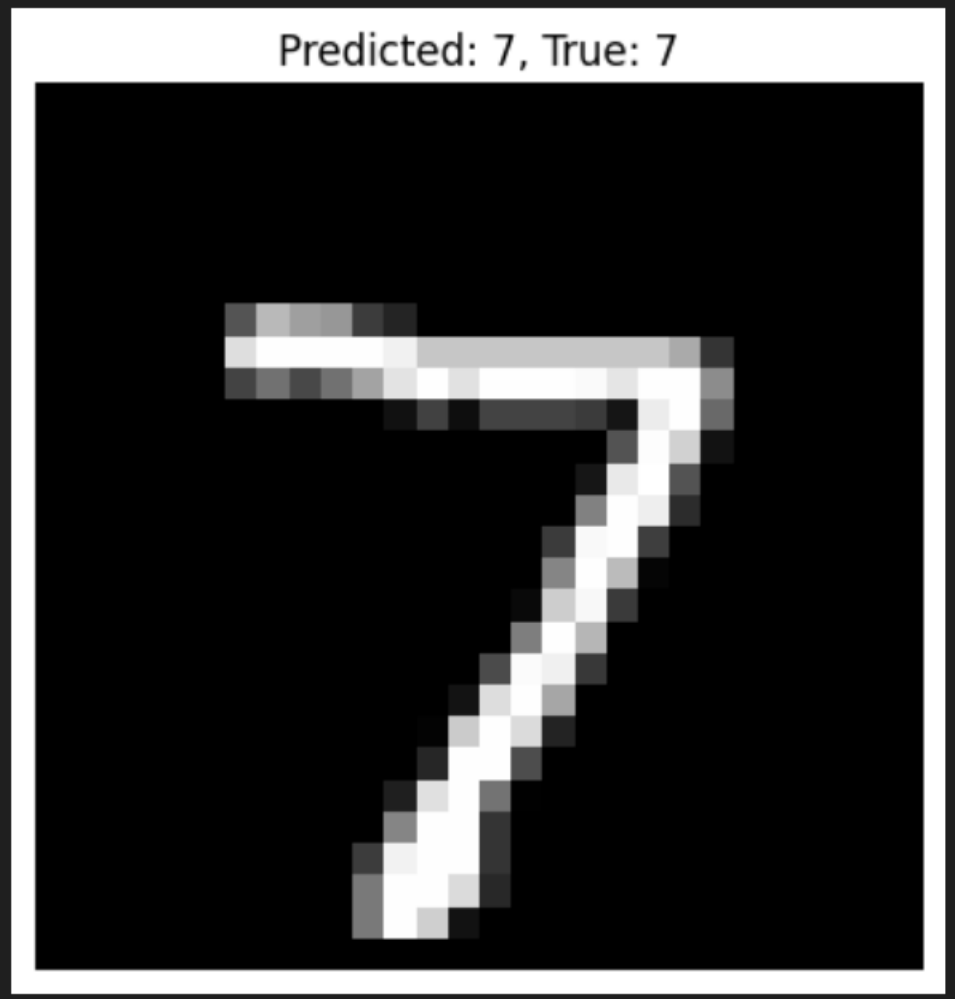
Step 5: Make Predictions

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

# Make predictions on the test data
predictions = model.predict(x_test)

# Predict the label for the first test image
predicted_label = np.argmax(predictions[0])
print(f"Predicted label: {predicted_label}, True label: {y_test[0]}")

# Visualize the first test image and its predicted label
plt.imshow(x_test[0], cmap='gray')
plt.title(f"Predicted: {predicted_label}, True: {y_test[0]}")
plt.axis('off')
plt.show()
```

Step 5: Make Predictions

Output

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Predicted label: 7, True label: 7
```



3. Building a Convolutional Neural Network (CNN) for Image Classification

Step 1: Build a CNN Model

```
import numpy as np
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Reshape the data to include a channel dimension
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)
```

Output

```
import numpy as np
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Reshape the data to include a channel dimension
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)
✓ 0.0s
```

Step 1: Build a CNN Model

```
# Define the CNN architecture
cnn_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])
# Compile the CNN model
cnn_model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

# Display model summary
cnn_model.summary()
```

Step 1: Build a CNN Model
Output

```
c:\Users\ECS\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using S
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten_3 (Flatten)	(None, 1600)	0
dense_6 (Dense)	(None, 64)	102,464
dense_7 (Dense)	(None, 10)	650

```
Total params: 121,930 (476.29 KB)

Trainable params: 121,930 (476.29 KB)

Non-trainable params: 0 (0.00 B)
```

Step 2: Train and Evaluate the CNN Model

```
# Train the CNN model
cnn_model.fit(x_train, y_train, epochs=5, validation_split=0.1)

# Evaluate the CNN model
cnn_test_loss, cnn_test_accuracy = cnn_model.evaluate(x_test, y_test)
print(f"CNN Test Accuracy: {cnn_test_accuracy:.4f}")
```

Step 2: Train and Evaluate the CNN Model Output

```
Epoch 1/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 29s 15ms/step - accuracy: 0.8977 - loss: 0.3324 - val_accuracy: 0.9857 - val_loss: 0.0513
Epoch 2/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 23s 14ms/step - accuracy: 0.9830 - loss: 0.0532 - val_accuracy: 0.9900 - val_loss: 0.0373
Epoch 3/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 44s 16ms/step - accuracy: 0.9891 - loss: 0.0329 - val_accuracy: 0.9907 - val_loss: 0.0367
Epoch 4/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 32s 19ms/step - accuracy: 0.9930 - loss: 0.0223 - val_accuracy: 0.9902 - val_loss: 0.0364
Epoch 5/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 37s 16ms/step - accuracy: 0.9945 - loss: 0.0181 - val_accuracy: 0.9880 - val_loss: 0.0470
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9855 - loss: 0.0429
CNN Test Accuracy: 0.9889
```

4. Saving and Loading a Model

```
# Save the trained model
model.save('mnist_classifier.h5')
print("Model saved to disk.")

# Load the saved model
loaded_model = tf.keras.models.load_model('mnist_classifier.h5')
print("Model loaded successfully.")
```

4. Saving and Loading a Model

Output

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g.
                                                
```

5. Transfer Learning using Pre-trained Models

```
from tensorflow.keras.applications import VGG16

# Load a pre-trained VGG16 model (without the top layer)
pretrained_model = VGG16(include_top=False, input_shape=(224, 224, 3))
pretrained_model.summary()
```

5. Transfer Learning using Pre-trained Models

Output

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step

Model: "vgg16"
```

Layer (type)	Output Shape	Param #
input_layer_4 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0

```
Total params: 14,714,688 (56.13 MB)

Trainable params: 14,714,688 (56.13 MB)

Non-trainable params: 0 (0.00 B)
```

6. TensorFlow for Natural Language Processing (NLP)

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Sample text data
texts = ["I love AI", "AI is the future", "I dislike spam emails"]
labels = [1, 1, 0]

# Tokenize the text data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
padded_sequences = pad_sequences(sequences, padding='post')

print("Tokenized and Padded Sequences:")
print(padded_sequences)
```

6. TensorFlow for Natural Language Processing (NLP)

Output

```
Tokenized and Padded Sequences:
[[1 3 2 0]
 [2 4 5 6]
 [1 7 8 9]]
```

Matplotlib Package

A powerful Python library used for creating static, interactive, and animated visualizations

Matplotlib Package

- Matplotlib is a powerful and widely used Python library for creating visualizations.
- It is particularly useful for data visualization, allowing users to create static, interactive, and animated plots.
- Matplotlib is an essential tool for data analysis, machine learning, and scientific research because it helps in understanding data and communicating insights effectively.

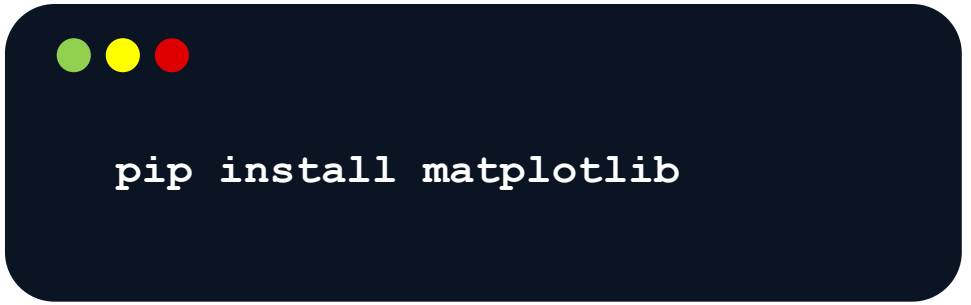
Why use Matplotlib?

- **Versatility**
  - Supports a wide range of visualizations:
  - Line plots
  - Scatter plots
  - Bar charts
  - Histograms
  - Heatmaps
  - 3D plots
  - Customizable to fit specific visualization needs.

Why use Matplotlib?

- **Integration**
  - Works seamlessly with NumPy, pandas, and other data science libraries.
  - Can be used with Jupyter notebooks for interactive data exploration.
- **Customization**
  - Offers fine-grained control over every aspect of a plot (e.g., labels, titles, colors, line styles).
  - Supports custom styles for professional-quality visualizations.
- **Accessibility**
  - Easy to use for beginners while providing advanced features for experienced users.

How to install matplotlib?




```
C:\Users\ECS>pip install matplotlib
Defaulting to user installation because normal site-packages is not writeable
Collecting matplotlib
  Downloading matplotlib-3.10.0-cp312-cp312-win_amd64.whl.metadata (11 kB)
Collecting contourpy>=1.0.1 (from matplotlib)
  Downloading contourpy-1.3.1-cp312-cp312-win_amd64.whl.metadata (5.4 kB)
Collecting cycler>=0.10 (from matplotlib)
  Using cached cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB)
Collecting fonttools>=4.22.0 (from matplotlib)
  Downloading fonttools-4.55.3-cp312-cp312-win_amd64.whl.metadata (168 kB)
Collecting kiwisolver>=1.3.1 (from matplotlib)
  Downloading kiwisolver-1.4.7-cp312-cp312-win_amd64.whl.metadata (6.4 kB)
Requirement already satisfied: numpy>=1.23 in c:\users\ecs\appdata\local\packages\pythonsoftwarefoundation.python.3.12_qbz5n2kfra8p0\localcache\local-packages\python312\site-packages (from matplotlib) (2.0.2)
Requirement already satisfied: packaging>=20.0 in c:\users\ecs\appdata\local\packages\pythonsoftwarefoundation.python.3.12_qbz5n2kfra8p0\localcache\local-packages\python312\site-packages (from matplotlib) (24.2)
Collecting pillow>=8 (from matplotlib)
  Downloading pillow-11.0.0-cp312-cp312-win_amd64.whl.metadata (9.3 kB)
Collecting pyparsing>=2.3.1 (from matplotlib)
  Using cached pyparsing-3.2.0-py3-none-any.whl.metadata (5.0 kB)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\ecs\appdata\local\packages\pythonsoftwarefoundation.python.3.12_qbz5n2kfra8p0\localcache\local-packages\python312\site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\ecs\appdata\local\packages\pythonsoftwarefoundation.python.3.12_qbz5n2kfra8p0\localcache\local-packages\python312\site-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
Downloading matplotlib-3.10.0-cp312-cp312-win_amd64.whl (8.0 MB)
   ---------------------------------------- 8.0/8.0 MB 38.3 MB/s eta 0:00:00
Downloading contourpy-1.3.1-cp312-cp312-win_amd64.whl (220 kB)
Using cached cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.55.3-cp312-cp312-win_amd64.whl (2.2 MB)
   ---------------------------------------- 2.2/2.2 MB 31.4 MB/s eta 0:00:00
Downloading kiwisolver-1.4.7-cp312-cp312-win_amd64.whl (55 kB)
Downloading pillow-11.0.0-cp312-cp312-win_amd64.whl (2.6 MB)
   ---------------------------------------- 2.6/2.6 MB 37.2 MB/s eta 0:00:00
Using cached pyparsing-3.2.0-py3-none-any.whl (106 kB)
Installing collected packages: pyparsing, pillow, kiwisolver, fonttools, cycler, contourpy, matplotlib
  WARNING: The scripts fonttools.exe, pyftmerge.exe, pyftsubset.exe and ttx.exe are installed in 'C:\Users\ECS\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\Scripts'
which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed contourpy-1.3.1 cycler-0.12.1 fonttools-4.55.3 kiwisolver-1.4.7 matplotlib-3.10.0 pillow-11.0.0 pyparsing-3.2.0
```

Matplotlib Examples

1. Importing Matplotlib

```
import matplotlib.pyplot as plt
```

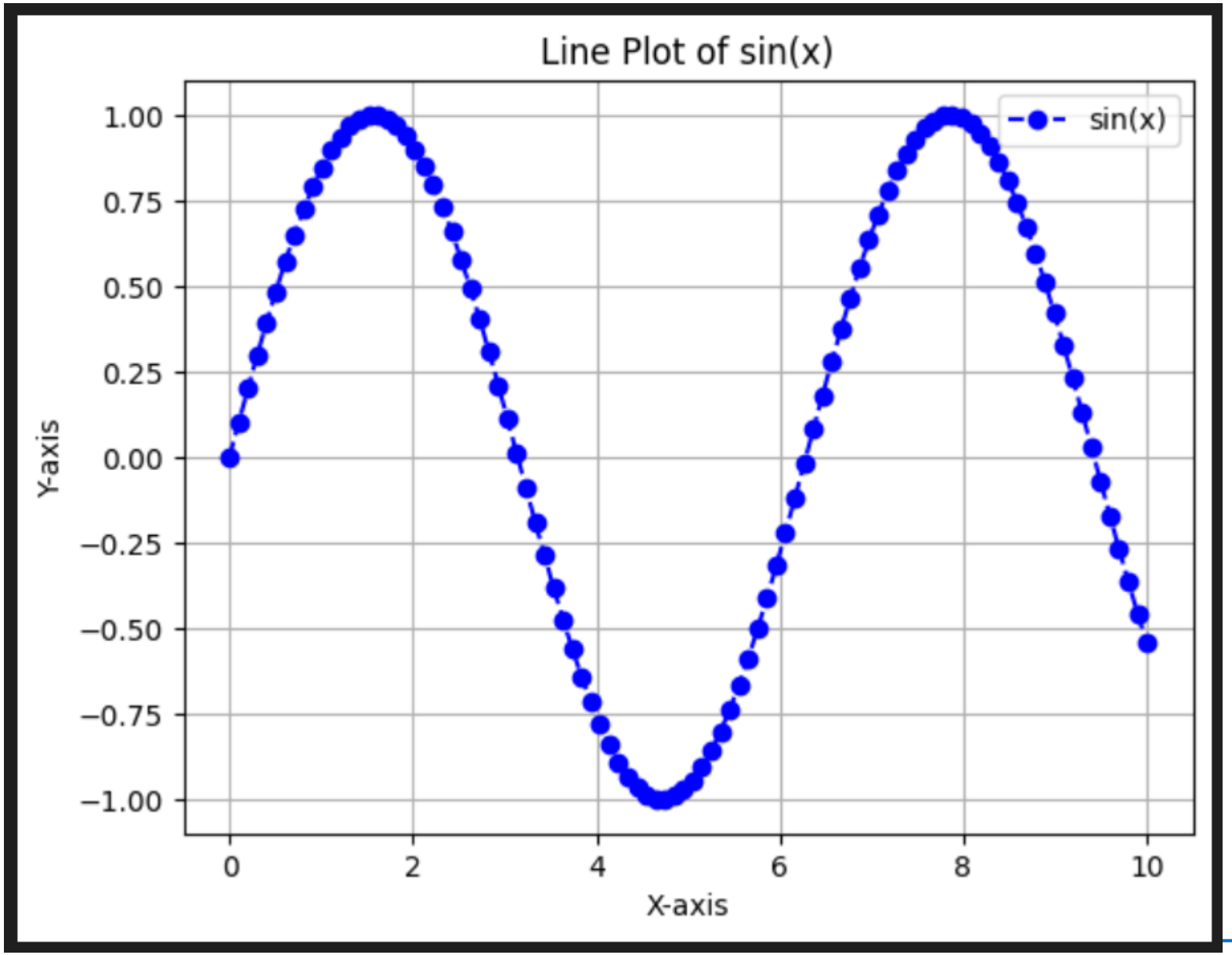
Output

```
import matplotlib.pyplot as plt
✓ 0.0s
```

2. Line Plot

```
import numpy as np

# Sample data
x = np.linspace(0, 10, 100)
y = np.sin(x)

# Create a line plot
plt.plot(x, y, label='sin(x)', color='blue', linestyle='--', marker='o')
plt.title("Line Plot of sin(x)")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.grid(True)
plt.show()
```

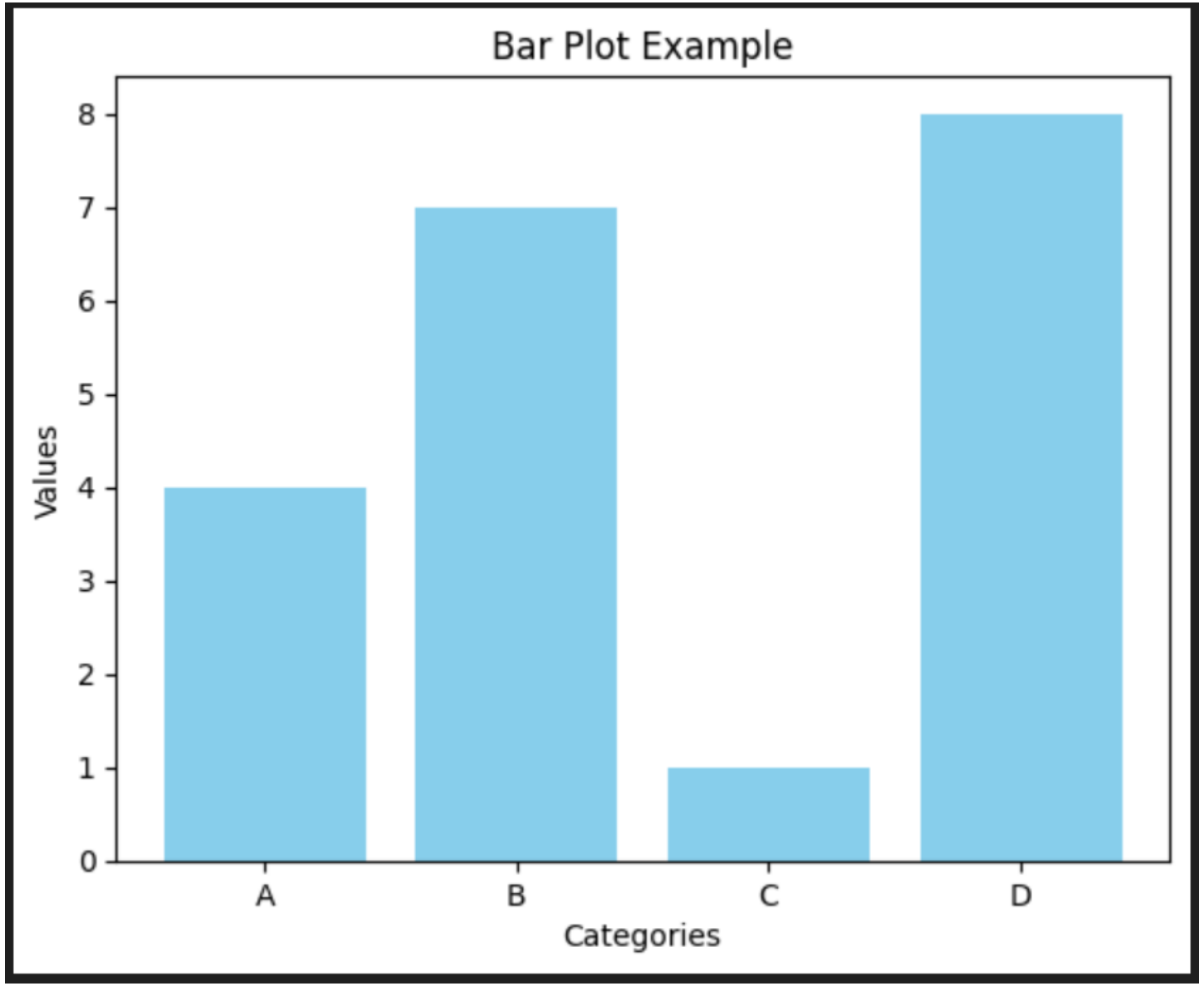
2. Line Plot Output



3. Bar Plot

```
categories = ['A', 'B', 'C', 'D']
values = [4, 7, 1, 8]

# Create a bar plot
plt.bar(categories, values, color='skyblue')
plt.title("Bar Plot Example")
plt.xlabel("Categories")
plt.ylabel("Values")
plt.show()
```

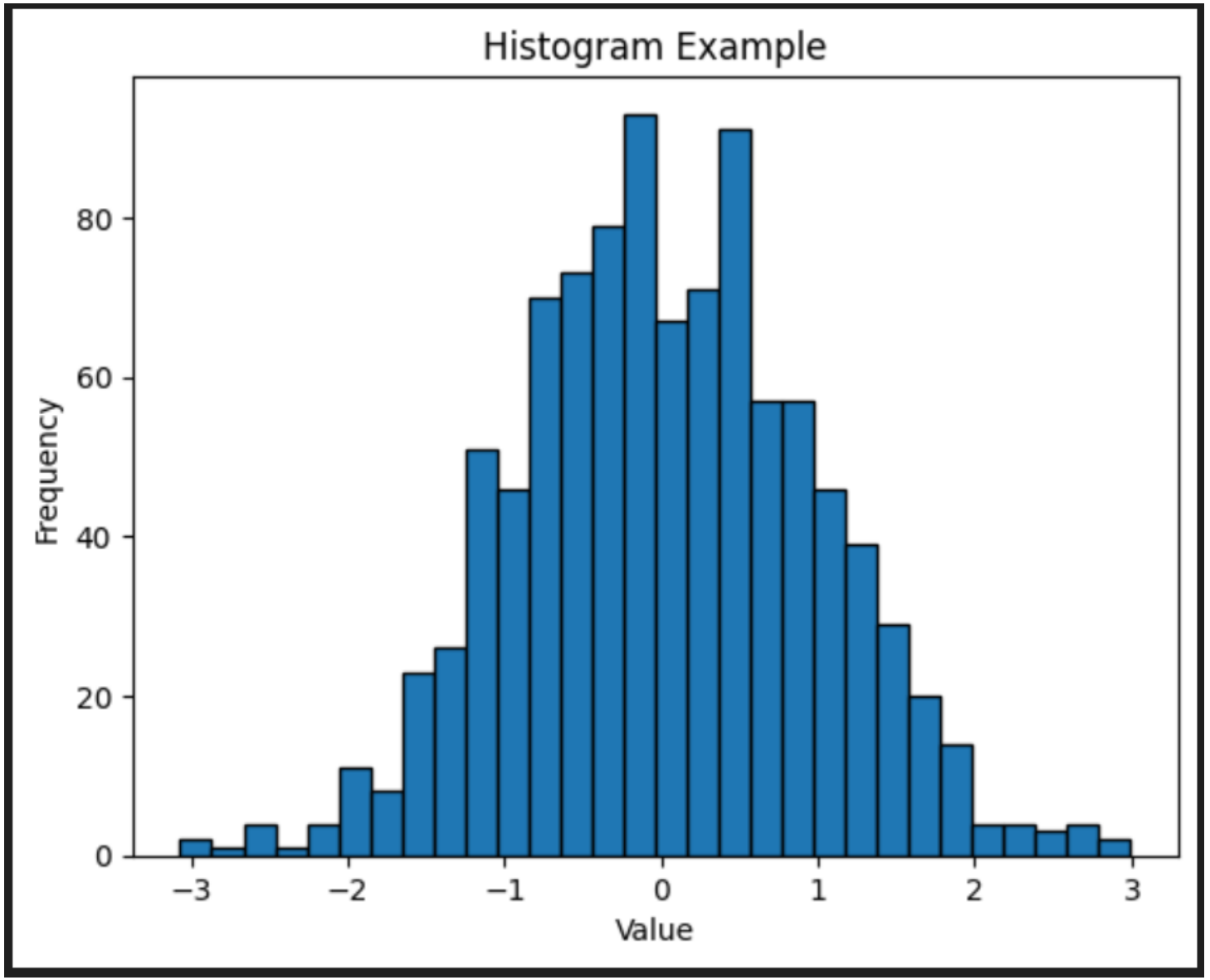
3. Bar Plot Output



4. Histogram

```
data = np.random.randn(1000)  # Generate 1000 random numbers

# Create a histogram
plt.hist(data, bins=30, edgecolor='black')
plt.title("Histogram Example")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()
```

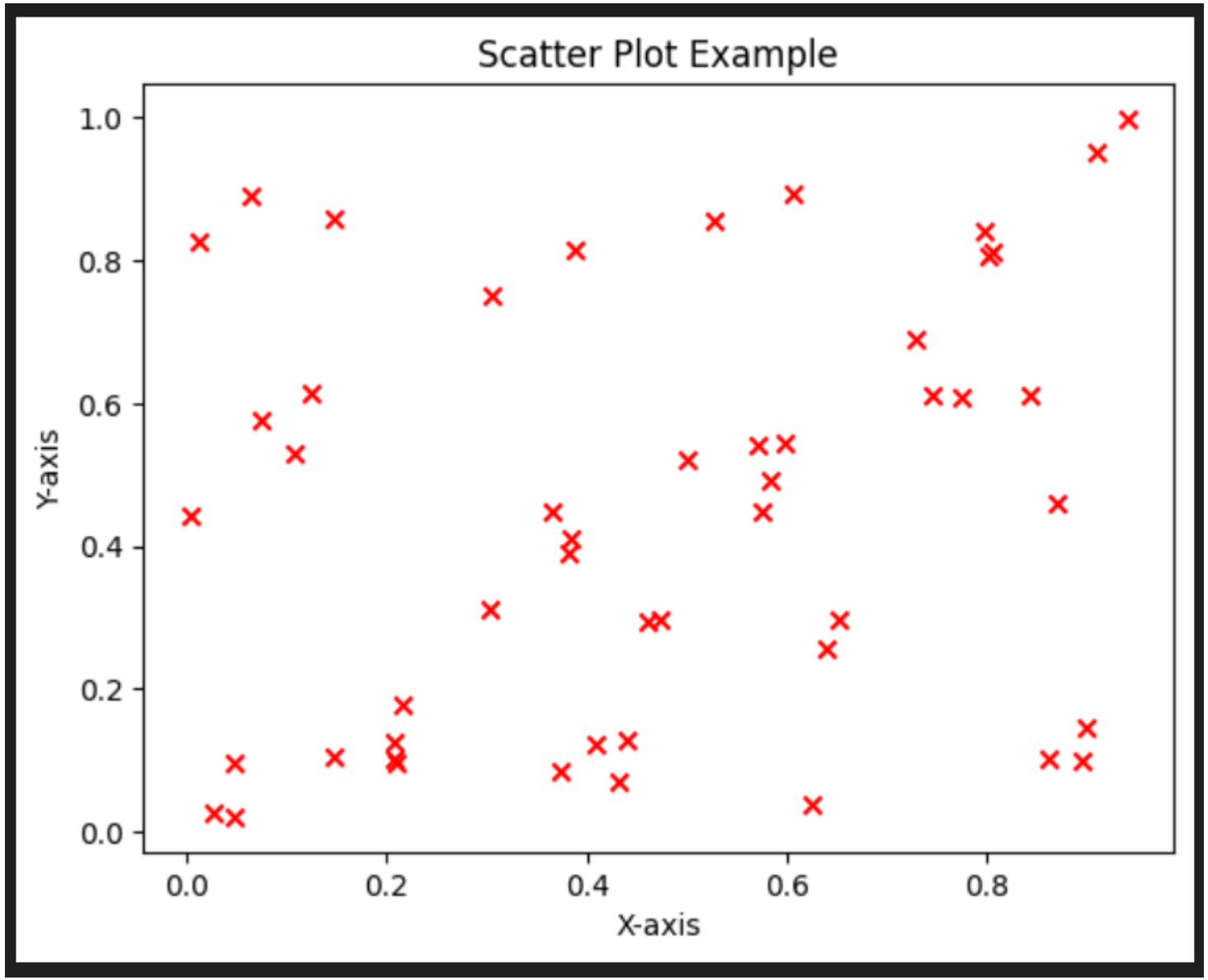
4. Histogram Output



5. Scatter Plot

```
x = np.random.rand(50)
y = np.random.rand(50)

# Create a scatter plot
plt.scatter(x, y, color='red', marker='x')
plt.title("Scatter Plot Example")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

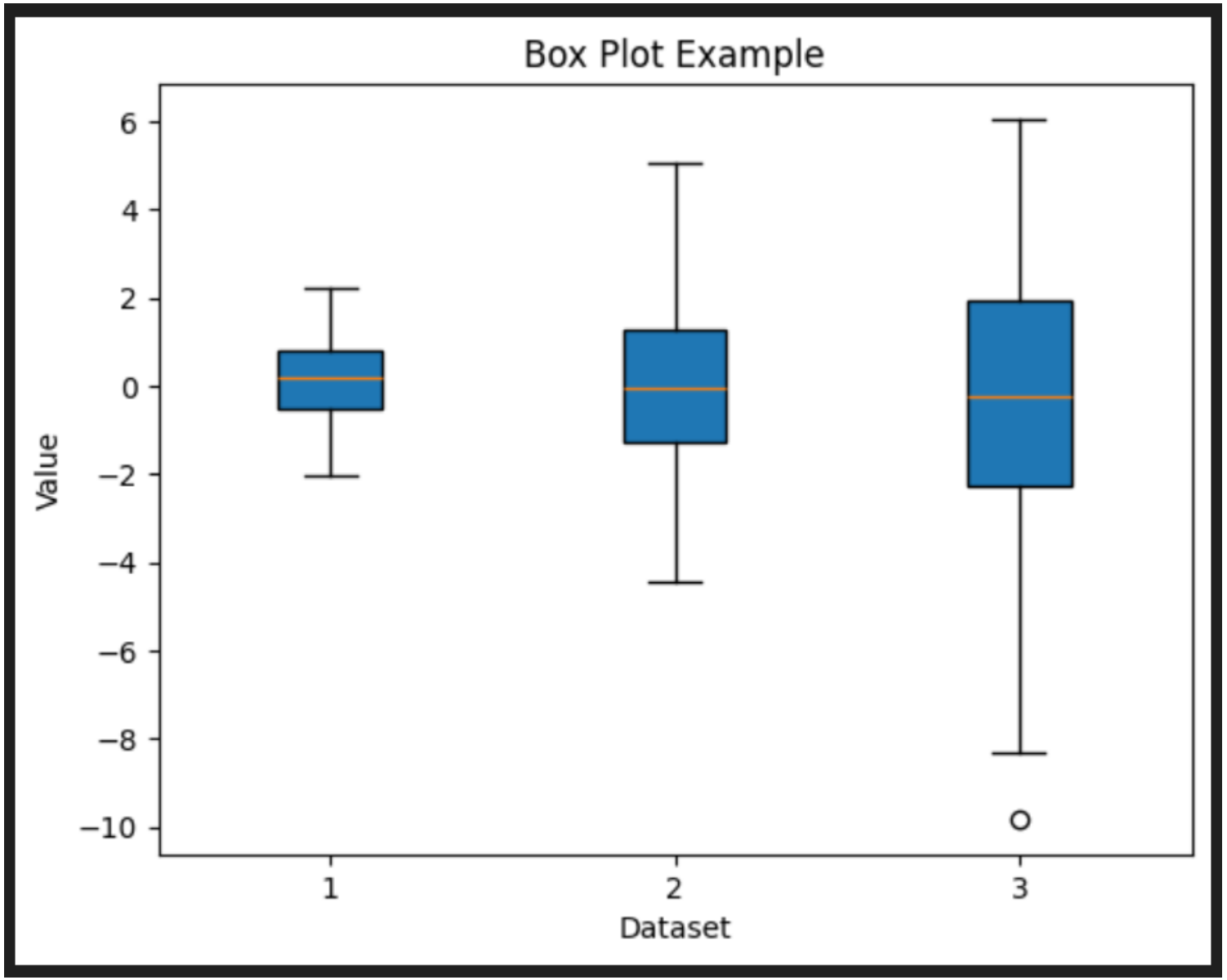
5. Scatter Plot Output



6. Box Plot

```
data = [np.random.normal(0, std, 100) for std in range(1, 4)]

# Create a box plot
plt.boxplot(data, patch_artist=True)
plt.title("Box Plot Example")
plt.xlabel("Dataset")
plt.ylabel("Value")
plt.show()
```

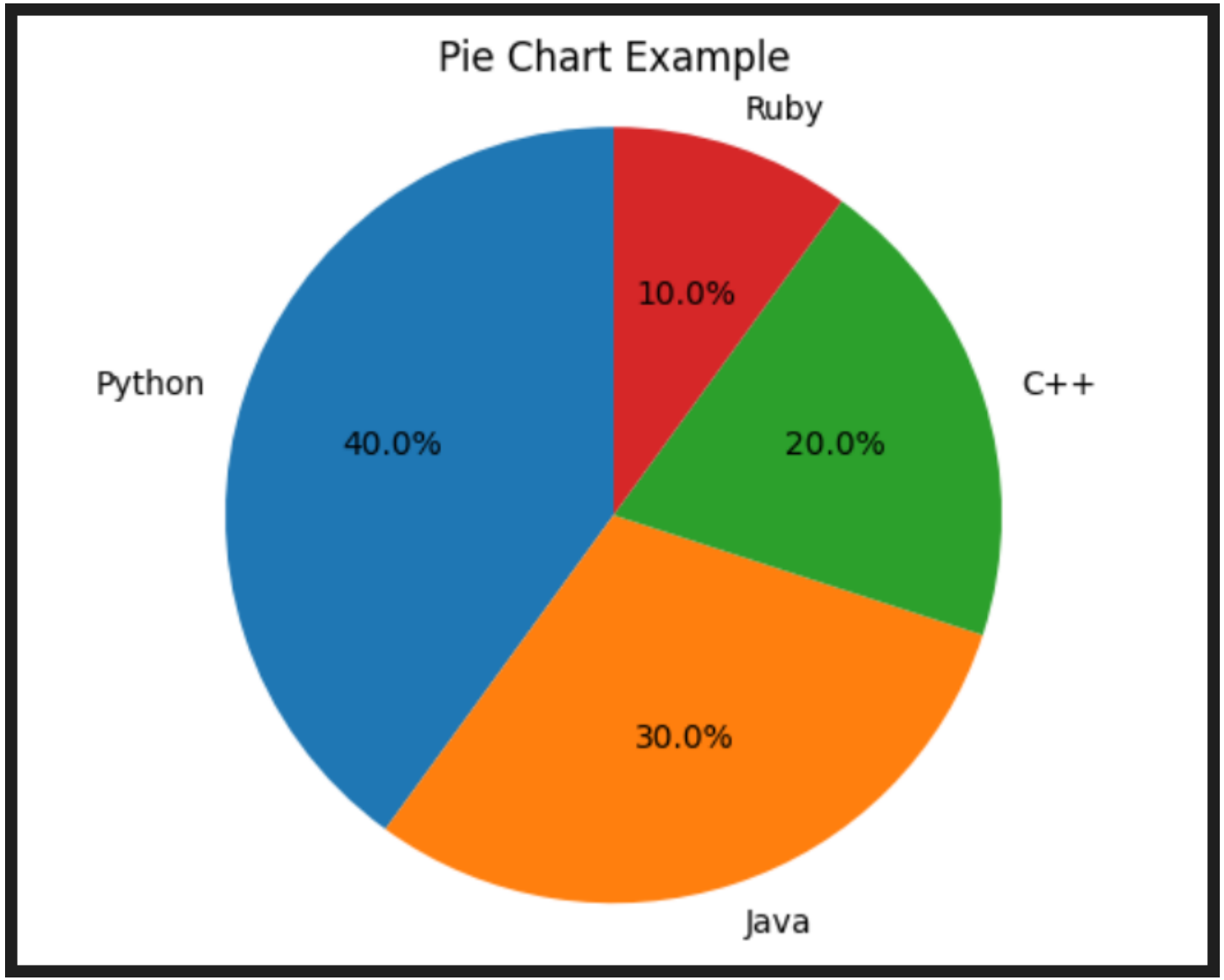
6. Box Plot Output



7. Pie Chart

```
labels = ['Python', 'Java', 'C++', 'Ruby']
sizes = [40, 30, 20, 10]

# Create a pie chart
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
plt.title("Pie Chart Example")
plt.axis('equal')  # Equal aspect ratio ensures the pie is circular
plt.show()
```

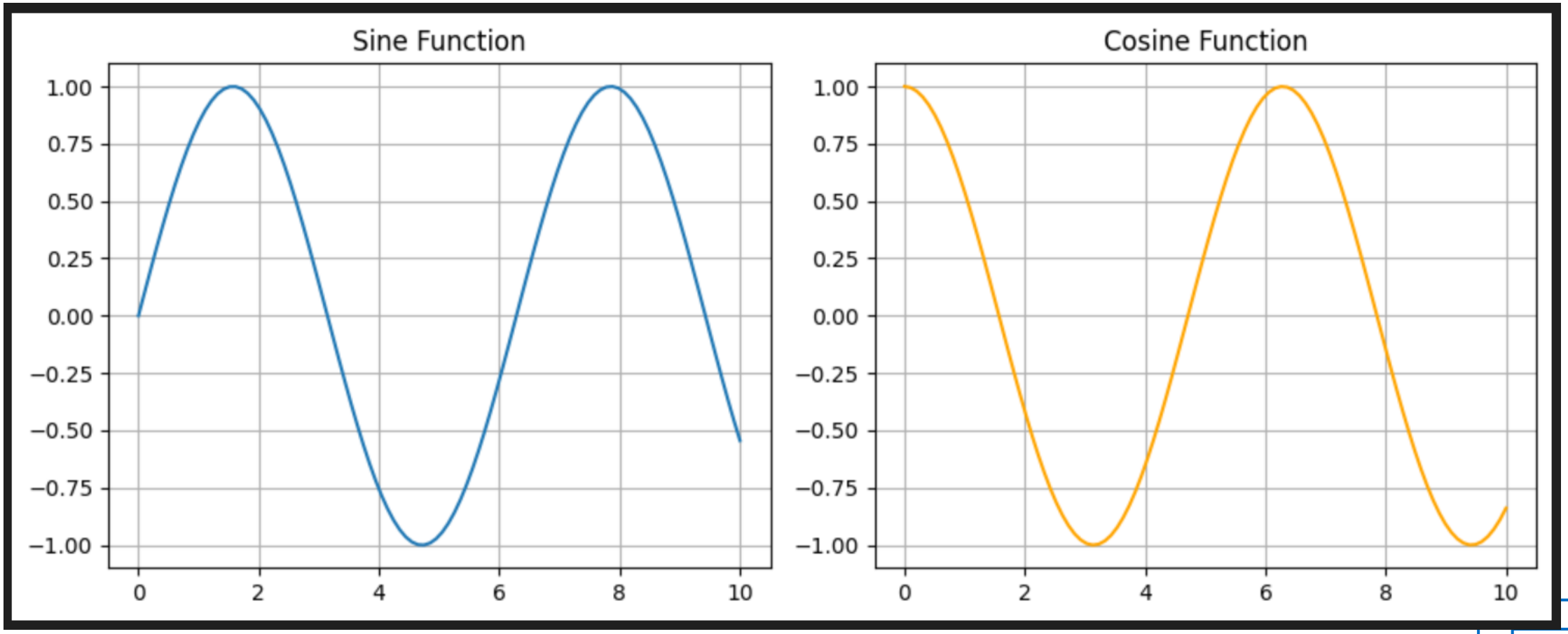
7. Pie Chart Output



8. Subplots

```
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Create subplots
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(x, y1, label='sin(x)')
plt.title("Sine Function")
plt.grid(True)
plt.subplot(1, 2, 2)
plt.plot(x, y2, label='cos(x)', color='orange')
plt.title("Cosine Function")
plt.grid(True)
plt.tight_layout()
plt.show()
```

8. Subplots Output

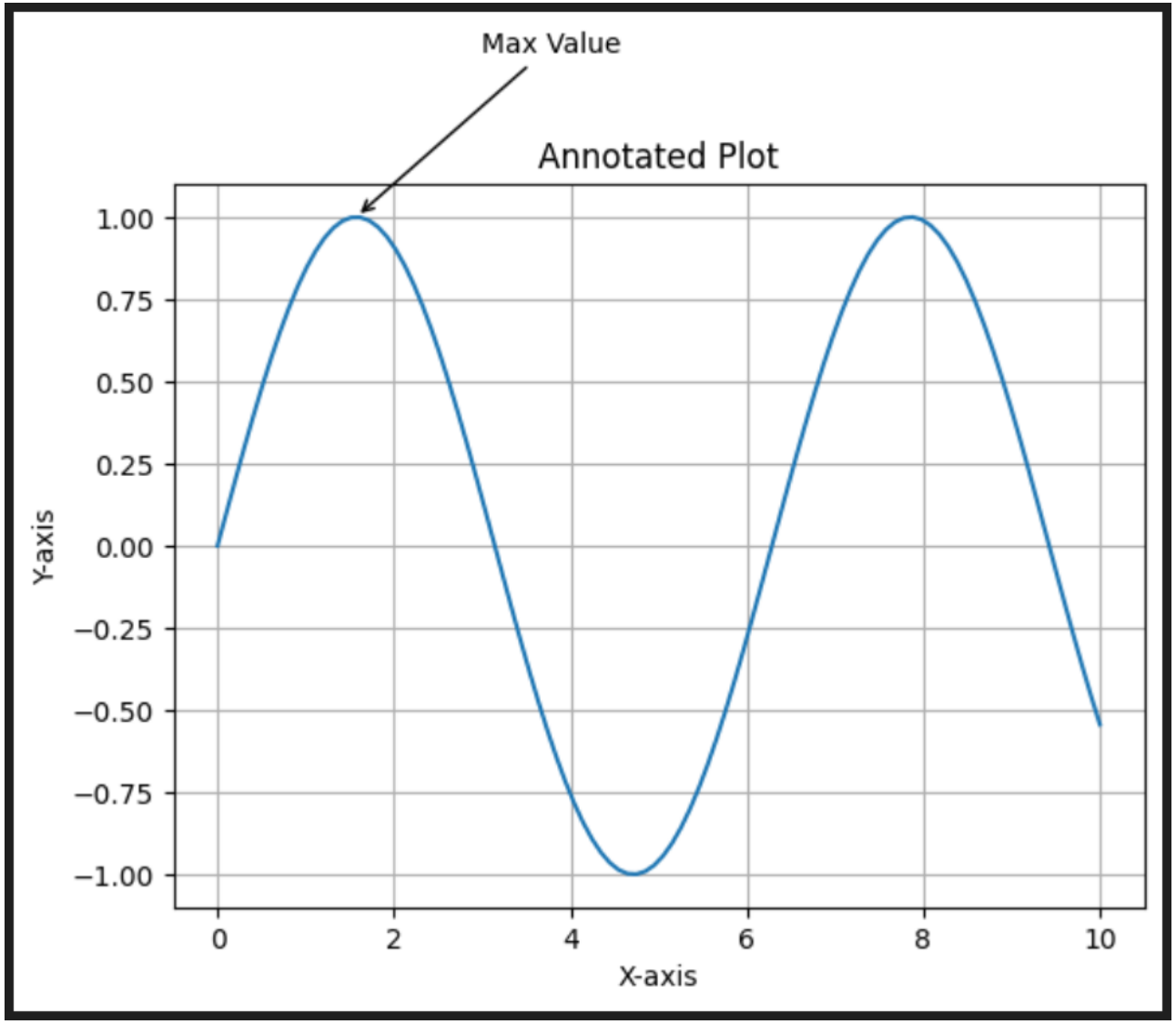


9. Adding Annotations

```
x = np.linspace(0, 10, 100)
y = np.sin(x)

plt.plot(x, y)
plt.title("Annotated Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")

# Annotate a point
plt.annotate('Max Value', xy=(1.57, 1), xytext=(3, 1.5),
             arrowprops=dict(facecolor='black', arrowstyle='->'))
plt.grid(True)
plt.show()
```

9. Adding Annotations

Output



10. Saving Plots

```
x = np.linspace(0, 10, 100)
y = np.sin(x)

plt.plot(x, y)
plt.title("Save Plot Example")
plt.savefig('plot.png', dpi=300)
plt.show()
```

10. Saving Plots

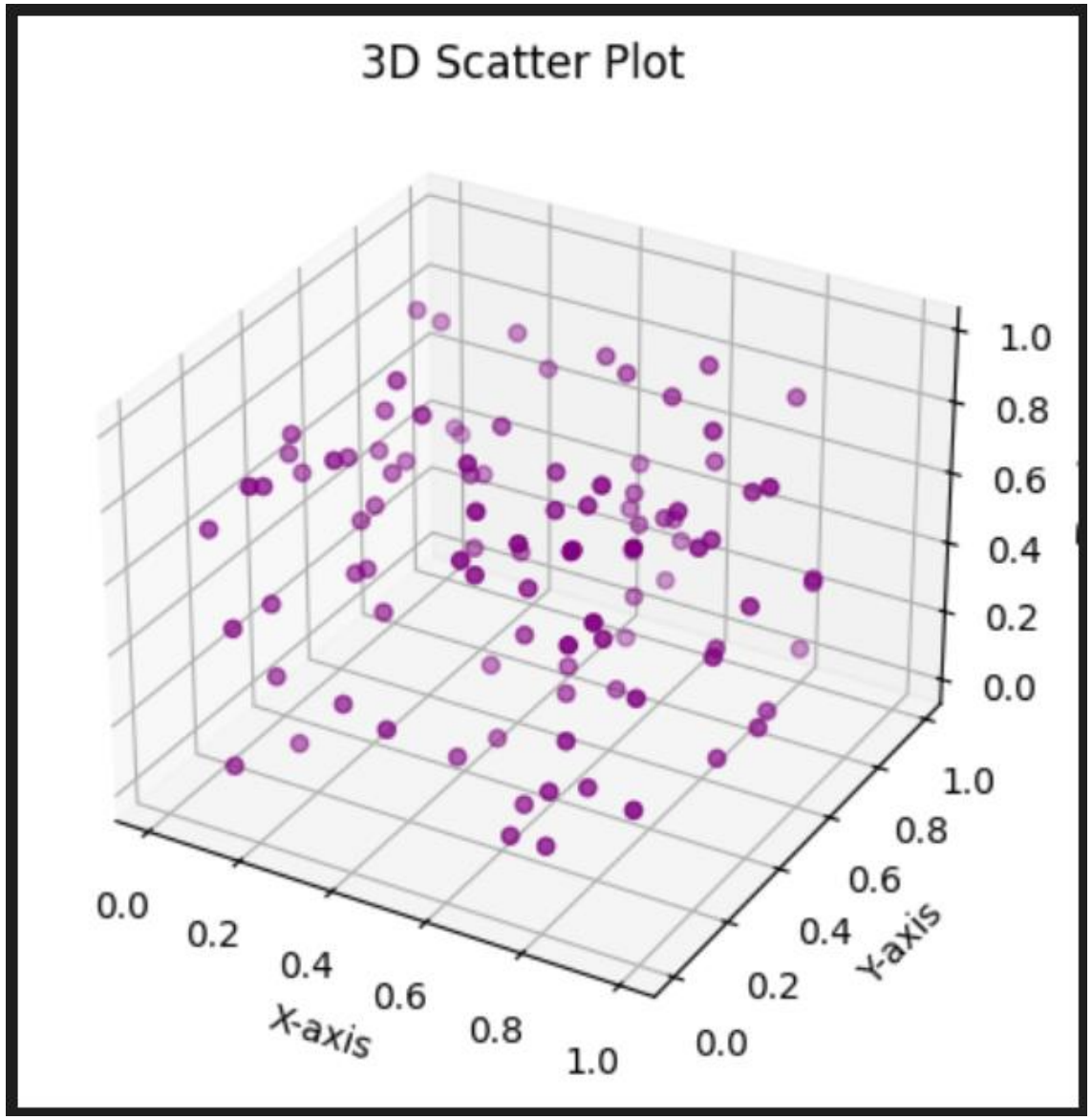
Output

11. 3D Plotting (using mpl_toolkits.mplot3d)

```
from mpl_toolkits.mplot3d import Axes3D

x = np.random.rand(100)
y = np.random.rand(100)
z = np.random.rand(100)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(x, y, z, color='purple')
ax.set_title("3D Scatter Plot")
ax.set_xlabel("X-axis")
ax.set_ylabel("Y-axis")
ax.set_zlabel("Z-axis")
plt.show()
```

11. 3D Plotting (using mpl_toolkits.mplot3d)

Output



How to implement Neural Network Classifier?

- Step 1: Import Libraries
- Step 2: Load the Dataset
- Step 3: Preprocess the Data
- Step 4: Build the Neural Network Model
- Step 5: Train the Model
- Step 6: Evaluate the Model
- Step 7: Make Predictions
- Step 8: Visualize Training Accuracy and Loss

Step 1: Import Libraries

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
```

Step 2: Load the Dataset

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

# Check the shapes of the dataset
print(f"Training data shape: {x_train.shape}")
print(f"Test data shape: {x_test.shape}")
```

Step 2: Load the Dataset

Output

```
Training data shape: (60000, 28, 28)
Test data shape: (10000, 28, 28)
```

Step 3: Preprocess the Data

```
# Normalize pixel values to range 0-1
x_train = x_train / 255.0
x_test = x_test / 255.0

# Convert labels to one-hot encoding
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

Step 4: Build the Neural Network Model

```
# Initialize the model
model = Sequential([
    Flatten(input_shape=(28, 28)),  # Flatten the 28x28 images into 1D vectors
    Dense(128, activation='relu'),  # First hidden layer
    Dense(64, activation='relu'),   # Second hidden layer
    Dense(10, activation='softmax') # Output layer for 10 classes
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Display the model summary
model.summary()
```

Step 4: Build the Neural Network Model

Output

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 784)	0
dense_3 (Dense)	(None, 128)	100,480
dense_4 (Dense)	(None, 64)	8,256
dense_5 (Dense)	(None, 10)	650

```
Total params: 109,386 (427.29 KB)

Trainable params: 109,386 (427.29 KB)

Non-trainable params: 0 (0.00 B)
```

Step 5: Train the Model

```
# Train the model
history = model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=0.2,
verbose=1)
```

Step 5: Train the Model

Output

```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 12s 6ms/step - accuracy: 0.8689 - loss: 0.4527 - val_accuracy: 0.9582 - val_loss: 0.1379
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9643 - loss: 0.1191 - val_accuracy: 0.9678 - val_loss: 0.1113
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 9s 6ms/step - accuracy: 0.9754 - loss: 0.0794 - val_accuracy: 0.9730 - val_loss: 0.0961
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9810 - loss: 0.0574 - val_accuracy: 0.9732 - val_loss: 0.0913
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9870 - loss: 0.0429 - val_accuracy: 0.9724 - val_loss: 0.0962
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9895 - loss: 0.0346 - val_accuracy: 0.9761 - val_loss: 0.0901
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9920 - loss: 0.0251 - val_accuracy: 0.9733 - val_loss: 0.1081
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9928 - loss: 0.0224 - val_accuracy: 0.9762 - val_loss: 0.1003
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9937 - loss: 0.0193 - val_accuracy: 0.9758 - val_loss: 0.1071
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9956 - loss: 0.0134 - val_accuracy: 0.9729 - val_loss: 0.1216
```

Step 6: Evaluate the Model

```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=1)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

Output

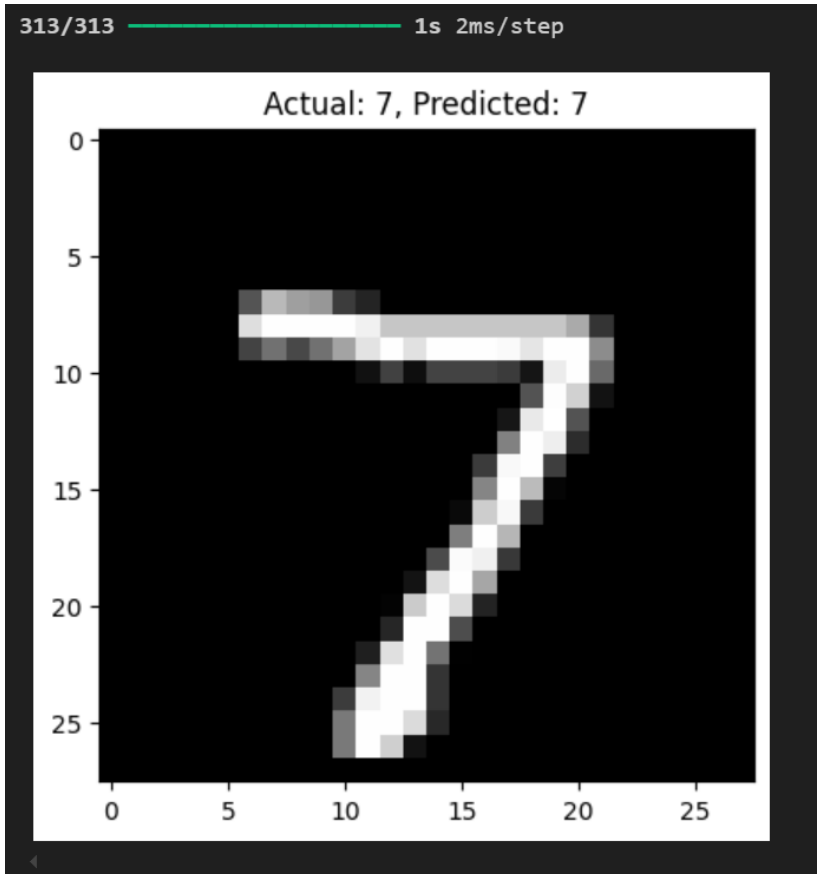
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9690 - loss: 0.1326
Test Loss: 0.11119746416807175
Test Accuracy: 0.9724000096321106
```

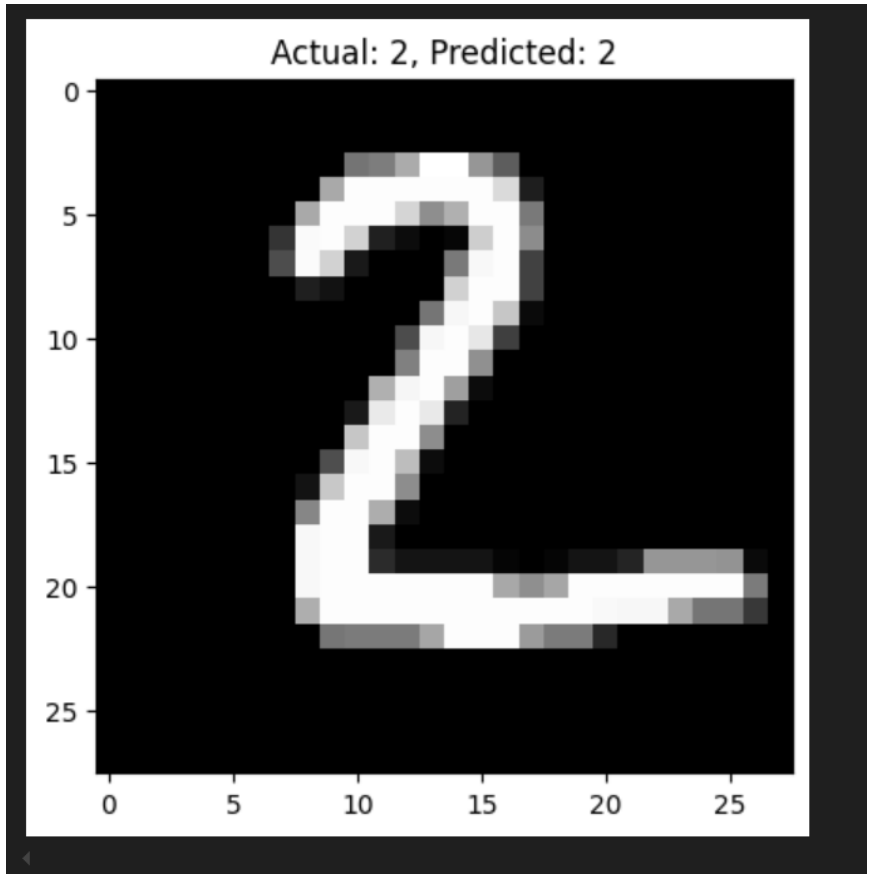
Step 7: Make Predictions

```
# Predict on the test dataset
predictions = model.predict(x_test)

# Display a few predictions along with the actual labels
import numpy as np
for i in range(5):
    plt.imshow(x_test[i], cmap='gray')
    plt.title(f"Actual: {np.argmax(y_test[i])}, Predicted: {np.argmax(predictions[i])}")
    plt.show()
```

Step 7: Make Predictions

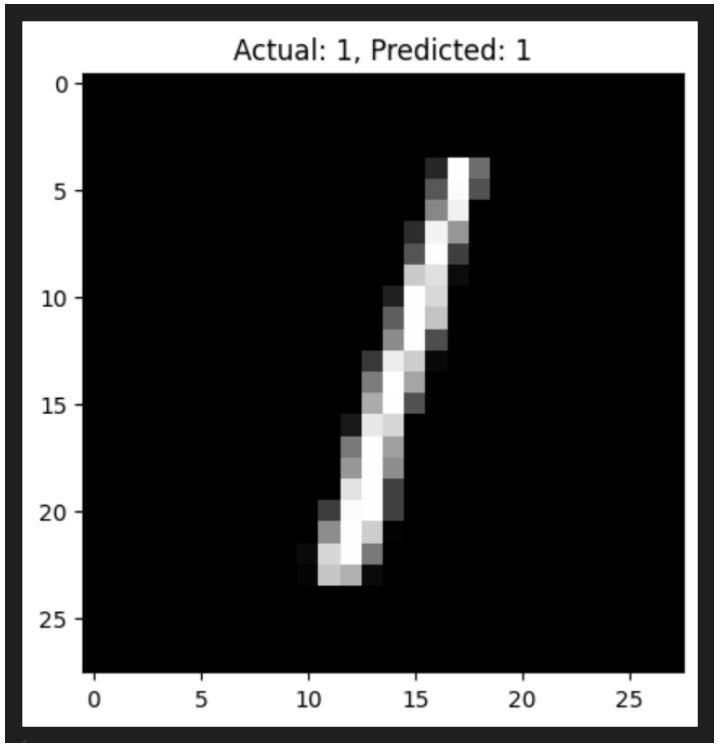
Output

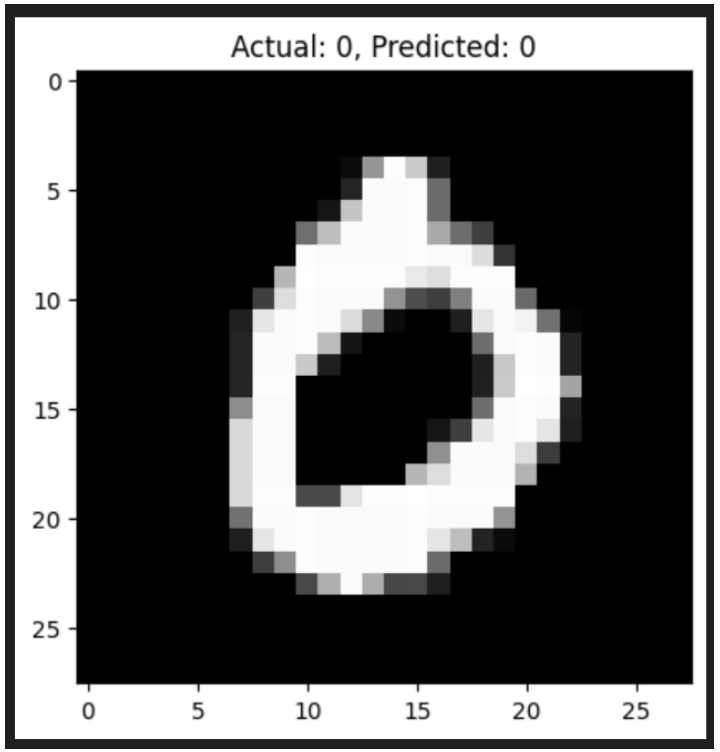


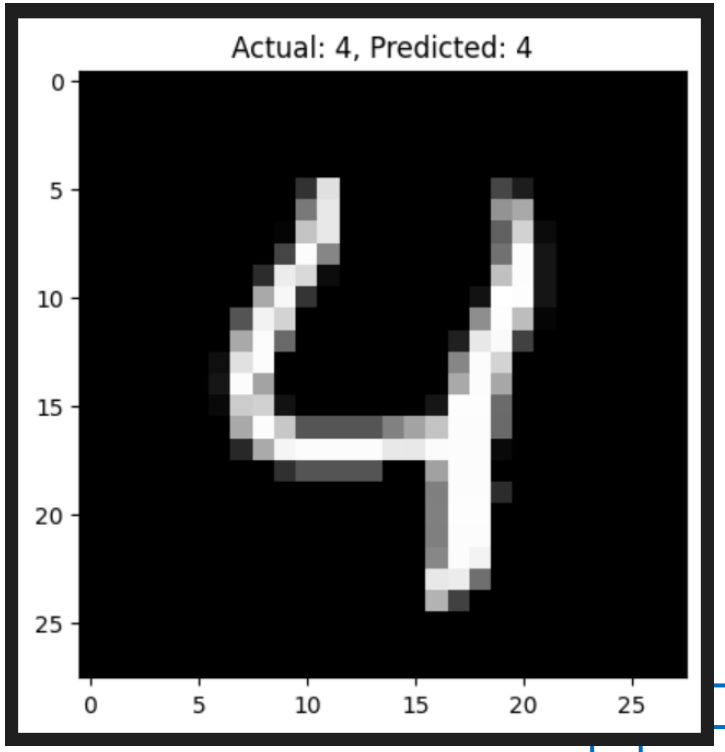


Step 7: Make Predictions

Output







Step 8: Visualize Training Accuracy and Loss

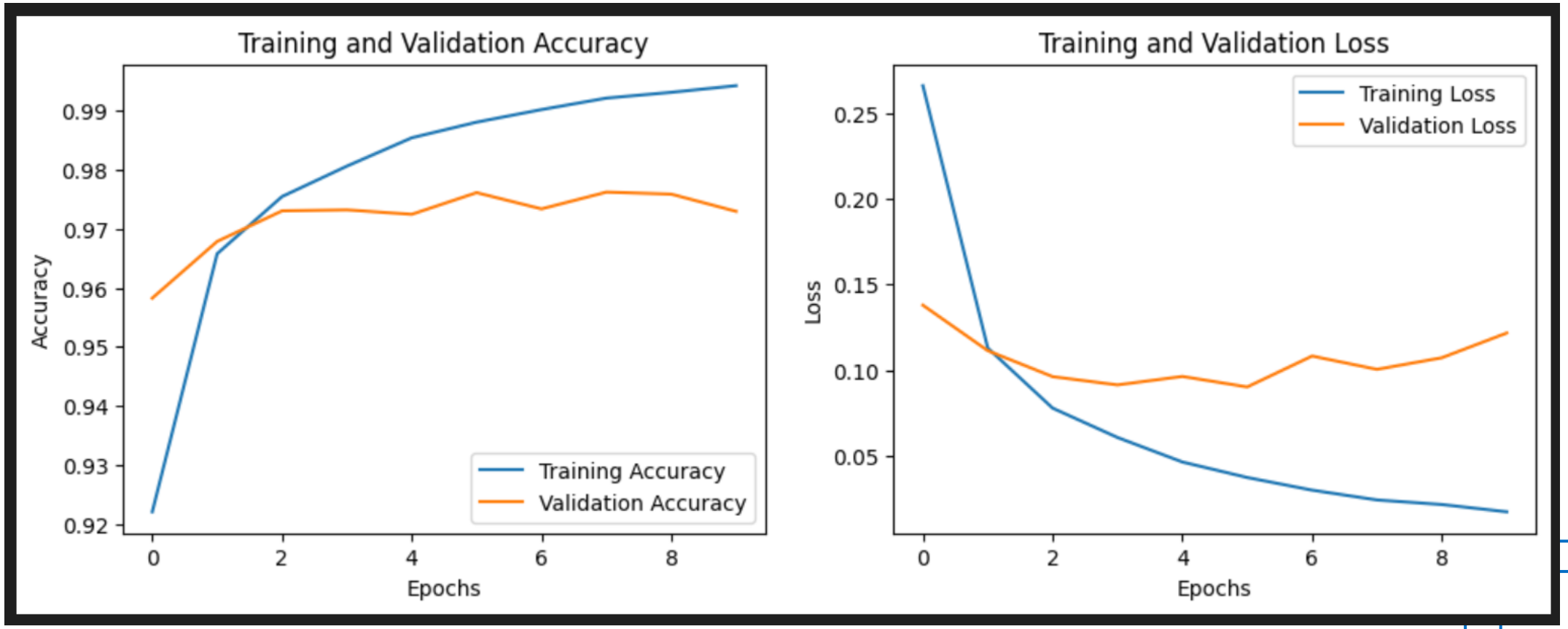
```
# Plot training accuracy and loss
plt.figure(figsize=(12, 4))

# Accuracy plot
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('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

Step 8: Visualize Training Accuracy and Loss

```
# Loss plot
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('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

Step 8: Visualize Training Accuracy and Loss

Output



Realistic Infotech Group
IT Training & Services
No.79/A, First Floor
Corner of Insein Road and
Damaryon Street
Quarter (9), Hlaing Township
Near Thukha Bus Station
09256675642, 09953933826
http://www.rig-info.com