# Appendix

This tutorial is designed to help engineers, regardless of their coding background, to effectively use a versatile Python script for various machine learning tasks, including regression, classification, and clustering. This script will allow you to:

* Set up Python environment
* Understand and configure the script
* Run the script with either synthetic (or your own data)
* Interpret the results and visualizations

**Table of Contents**

1. Prerequisites
2. Setting Up Your Python Environment
3. Obtaining the Script
4. Understanding the Script Structure
5. Configuring the Script
6. Preparing Your Data
7. Running the Script
8. Interpreting Results and Visualizations
9. Troubleshooting Common Issues
10. Additional Tips

# 1. Prerequisites

First, ensure you have the following:

* **A Computer**: Windows, macOS, or Linux.
* **Internet Connection**: To download necessary software and libraries.
* **Basic Computer (not coding) Skills**: Familiarity with navigating files and folders.

# 2. Setting Up Your Python Environment

To run the script, install Python on your computer, along with several libraries. Here's how to set everything up:

## a. Install Python

1. Download Python:
   * Visit the official Python website
   * Download the latest **Python 3.x** installer for your operating system.
2. Install Python:
   * Windows:
     + Run the downloaded .exe installer.
     + Important: During installation, check the box that says **"Add Python to PATH"**.
   * macOS:
     + Run the downloaded .pkg installer and follow the prompts.
   * Linux:
     + Python is usually pre-installed. To check, open the terminal and type:
     + python3 --version
     + If not installed, follow your distribution's instructions to install Python 3.

## b. Install Required Python Libraries

1. Open Command Prompt or Terminal:
   * Windows: Search for Command Prompt.
   * macOS/Linux: Open the Terminal.
2. Install Libraries via pip:
   * Type the following command and press Enter:
   * pip install numpy pandas matplotlib scikit-learn shap openpyxl
   * Note:
     + openpyxl is required to read Excel files (.xlsx).
     + If you plan to use XGBoost models, install it with:
       1. pip install xgboost
3. Verify Installation:
   * To ensure libraries are installed, you can try importing them in Python:

Then, in the Python prompt:

import numpy

import pandas

import matplotlib

import sklearn

import shap

* + - If no errors appear, installations were successful.
    - Exit Python by typing exit() and pressing **Enter**.

# 3. Obtaining the Script

You need to save the provided Python script to your computer. Here's how:

1. Open a Text Editor:
   * Windows: Notepad or any preferred editor like Notepad++.
   * macOS: TextEdit (ensure it's in plain text mode) or editors like Sublime Text.
   * Linux: Gedit, Nano, or any preferred editor.
2. Copy the Script:
   * Copy the entire Python script provided in your question.
3. Paste and Save:
   * Paste the script into the text editor.
   * Save the file with a .py extension, e.g., ml\_script.py.

# 4. Understanding the Script Structure

While you don't need to modify the code, understanding its structure will help you effectively use it. Here's a breakdown of the script sections:

1. Import Libraries: Loads necessary Python libraries for data manipulation, visualization, and machine learning.
2. Task Selection: Choose the type of machine learning task:
   * Regression: Predict continuous values.
   * Classification: Predict categorical labels.
   * Clustering: Group similar data points.
3. Toggle Mechanism:
   * USE\_USER\_DATA: Decide whether to use your own dataset or synthetic (generated) data.
   * MODEL\_TYPE: Select the machine learning model to use (e.g., Random Forest, XGBoost).
   * COMPARE\_ALL: Whether to compare multiple models simultaneously.
4. Synthetic Data Definition: Generates sample data if not using your own.
5. Data Loading Functionality: Function to load data from Excel or CSV files.
6. Data Preparation: Loads data based on your selection and prepares it for modeling.
7. Models for Each Task: Defines available models based on the selected task.
8. Model Selection Logic: Chooses which model(s) to train based on your settings.
9. Training: Trains the selected model(s) using the prepared data.
10. Metrics (Single Model): Calculates performance metrics for the trained model.
11. Results and Visualization: Displays metrics and generates visual plots of results.
12. Additional Plots for Multi-Model: Generates plots when comparing multiple models.
13. SHAP Analysis (Single Model): Provides model explainability using SHAP.
14. Clustering Evaluation (Optional): Additional evaluation metrics for clustering tasks.

# 5. Configuring the Script

Before running the script, you'll need to configure a few settings to tailor it to your needs.

a. Open the Script for Editing

1. Locate the Script File:
   * Navigate to the folder where you saved ml\_script.py.
2. Open with Text Editor:
   * Right-click the file and choose Open with → your preferred text editor.

b. Configure Task Selection

At the beginning of the script, you'll find the Task Selection section:

# 1. Task Selection

# -------------------------------

# Set a seed to ensure reproducible results every time the code runs

np.random.seed(42) # Seed for randomness

# Choose the ML task you want to perform: 'regression', 'classification', or 'clustering'

TASK = 'regression' # You can change this to 'classification' or 'clustering' as needed

* **TASK Options**:
  + 'regression': For predicting continuous values (e.g., house prices).
  + 'classification': For predicting categories (e.g., spam detection).
  + 'clustering': For grouping similar data points without predefined labels.

**Action**: Set the TASK variable to your desired task by replacing 'regression' with 'classification' or 'clustering' if needed.

c. Configure Toggle Mechanism

# 2. Toggle Mechanism

# -------------------------------

# Decide whether to use your own dataset or use synthetic (generated) data

USE\_USER\_DATA = False # Set to True if you want to load your own data files

# Choose a single model type (if COMPARE\_ALL=False). Options:

# For regression/classification: 'random\_forest', 'xgboost', 'knn'

# For clustering: 'kmeans', 'birch', 'gaussian\_mixture'

MODEL\_TYPE = 'random\_forest'

# Compare all algorithms for the selected TASK if True.

# If True, we loop over all relevant models and produce multi-model metrics & plots (but skip SHAP).

COMPARE\_ALL = False

* USE\_USER\_DATA:
  + False: The script will use synthetic data.
  + True: You'll provide your own dataset.
* MODEL\_TYPE:
  + For Regression/Classification:
    - 'random\_forest'
    - 'xgboost' (requires XGBoost library)
    - 'knn'
  + For Clustering:
    - 'kmeans'
    - 'birch'
    - 'gaussian\_mixture'
* COMPARE\_ALL:
  + False: Use a single model as specified by MODEL\_TYPE.
  + True: Compare multiple models relevant to the selected task.

Action:

1. Using Your Own Data:
   * If you have your own dataset, set USE\_USER\_DATA = True.
   * Ensure your data files are in the same directory as the script.
2. Selecting Model Type:
   * Choose your preferred model by setting MODEL\_TYPE accordingly.
   * If unsure, 'random\_forest' is a good default choice.
3. Comparing All Models:
   * If you want to see how different models perform on your task, set COMPARE\_ALL = True.
   * This will train and evaluate multiple models automatically.

d. Save the Configuration

After making your changes, save the script (Ctrl + S or Cmd + S).

# 6. Preparing Your Data

Depending on whether you're using synthetic data or your own dataset, follow the appropriate steps.

a. Using Synthetic Data (Default)

If USE\_USER\_DATA = False, the script will automatically generate synthetic data based on the selected task. No additional steps are needed.

b. Using Your Own Dataset

If you prefer to use your own data, follow these steps:

i. Prepare Your Data File

* File Formats Supported:
  + Excel Files: .xlsx
  + CSV Files: .csv
* Data Structure:
  + Regression & Classification:
    - Features: All columns except the last one.
    - Target: The last column.
    - Ensure there are at least two columns: one for features and one for the target.
  + Clustering:
    - Features: All columns (no target variable).
* Example:
  + Regression:

|  |  |  |
| --- | --- | --- |
| Feature1 | Feature2 | Target |
| 5.1 | 3.5 | 10.2 |
| 4.9 | 3.0 | 9.8 |
| ... | ... | ... |

* + Classification:

|  |  |  |
| --- | --- | --- |
| Feature1 | Feature2 | Label |
| 2.5 | 3.6 | 0 |
| 1.3 | 3.2 | 1 |
| ... | ... | ... |

* + Clustering:

|  |  |
| --- | --- |
| Feature1 | Feature2 |
| 5.1 | 3.5 |
| 4.9 | 3.0 |
| ... | ... |

ii. Place the Data File

1. Location:
   * Ensure your data file is in the same folder as ml\_script.py.
2. Naming Conventions:
   * Regression Data: Name it regression\_dataset.xlsx or regression\_dataset.csv.
   * Classification Data: Name it classification\_dataset.xlsx or classification\_dataset.csv.
   * Clustering Data: Name it clustering\_dataset.xlsx or clustering\_dataset.csv.

Example:

* + If performing a regression task, save your file as regression\_dataset.csv.

iii. Verify Data Integrity

Ensure your data:

* Contains No Missing Values: The script may not handle missing data.
* Is Properly Formatted: Numerical data should be in numerical format.
* Consistent: All rows should have the same number of columns.

Tip: You can use Excel or a spreadsheet application to clean and verify your data.

# 7. Running the Script

Now that your environment is set up and the script is configured, it's time to run it.

a. Open Command Prompt or Terminal

* Windows:
  + Open Command Prompt.
* macOS/Linux:
  + Open the Terminal application.

b. Navigate to the Script Directory

Use the cd command to change directories to where your script is saved.

Example:

* Windows:
* cd C:\ML\_Scripts\
* macOS/Linux:
* cd ~/ML\_Scripts/

Action: Replace the path with your actual script location.

c. Run the Script

Execute the script using Python.

python ml\_script.py

Note:

* If you have multiple Python versions installed, you might need to specify python3 instead of python:
* python3 ml\_script.py

Action: Type the above command and press Enter.

d. Wait for Execution

The script will perform the following:

1. Data Loading: Loads either synthetic or your provided data.
2. Model Training: Trains the selected machine learning model(s).
3. Evaluation: Calculates performance metrics.
4. Visualization: Displays plots and metrics.

# 8. Interpreting Results and Visualizations

After running the script, you'll receive both textual metrics and visual plots. Here's how to understand them:

a. Performance Metrics

Depending on the task, different metrics will be displayed.

i. Regression

|  |  |  |  |
| --- | --- | --- | --- |
| Metric | Training | Cross-Validation | Test |
| R² | 0.85 | 0.80 | 0.78 |
| MAE (Mean Absolute Error) | 2.1 | 2.5 | 2.7 |

* R² (R-squared): Measures how well the model explains the variability of the target variable. Closer to 1 is better.
* MAE: Average absolute difference between predicted and actual values. Lower is better.

ii. Classification

|  |  |  |  |
| --- | --- | --- | --- |
| Metric | Training | Cross-Validation | Test |
| Accuracy | 0.90 | 0.85 | 0.83 |
| F1-Score | 0.89 | 0.84 | 0.82 |

* Accuracy: Percentage of correct predictions. Higher is better.
* F1-Score: Balances precision and recall. Higher is better.

iii. Clustering

|  |  |
| --- | --- |
| Metric | Score |
| Silhouette Score | 0.65 |
| Davies-Bouldin Index | 1.2 |

* Silhouette Score: Measures how similar an object is to its own cluster compared to other clusters. Higher is better (range: -1 to 1).
* Davies-Bouldin Index: Measures cluster separation. Lower is better.

b. Visual Plots

The script generates various plots based on the task and configuration.

i. Regression Plots

* Actual vs. Predicted Scatter Plots:
  + Training Data: Compares actual vs. predicted values on training data.
  + Cross-Validation: Compares actual vs. predicted values during cross-validation.
  + Test Data: Compares actual vs. predicted values on test data.
* SHAP Summary Plots:
  + Bar Plot: Shows the importance of each feature.
  + Summary Plot: Visualizes the impact of features on predictions.

ii. Classification Plots

* Confusion Matrices:
  + Visual representations showing true vs. predicted labels for training, cross-validation, and test data.
* SHAP Summary Plots:
  + Similar to regression, it is used to understand feature impacts.

iii. Clustering Plots

* PCA-Reduced Data Scatter Plot:
  + Visualizes clusters in a 2D space using Principal Component Analysis (PCA).

c. Multi-Model Comparisons

If COMPARE\_ALL = True, the script will display metrics and plots for each model evaluated, allowing you to compare their performances side by side.

# 9. Troubleshooting Common Issues

Even with careful setup, you might encounter some issues. Here's how to resolve common problems:

a. Python Not Recognized

Error:

'python' is not recognized as an internal or external command...

Solution:

* Ensure Python is installed and added to your system's PATH.
* Reinstall Python and check the "Add Python to PATH" option during installation.

b. Missing Libraries

Error:

ModuleNotFoundError: No module named 'numpy'

Solution:

* Install the missing library using pip.
* pip install numpy
* Repeat for other missing libraries.

c. File Not Found

Error:

FileNotFoundError: File 'regression\_dataset.xlsx' does not exist...

Solution:

* Ensure your data file is named correctly and placed in the same directory as the script.
* Check the file extension (.xlsx or .csv).

d. SHAP Library Not Installed

Error:

ModuleNotFoundError: No module named 'shap'

Solution:

* Install SHAP using pip:
* pip install shap

e. Unsupported File Format

Error:

ValueError: Unsupported file format. Please provide an Excel (.xlsx) or CSV (.csv) file.

Solution:

* Ensure your data file is either .xlsx or .csv.
* Rename the file if necessary.

f. XGBoost Import Error

Error:

ImportError: No module named 'xgboost'

Solution:

* Install XGBoost if you intend to use it:
* pip install xgboost
* If not needed, ensure MODEL\_TYPE isn't set to 'xgboost'.

g. Indentation Errors in Script

Error:

IndentationError: expected an indented block

Solution:

* Ensure the script's indentation is correct. Python relies on indentation to define code blocks.
* If you edited the script, double-check that all code blocks are properly indented (usually by 4 spaces).

h. Other Errors

If you encounter other errors:

1. Read the Error Message Carefully: It often points to the issue.
2. Google the Error: Many others might have faced similar issues.

# 10. Additional Tips

For better understanding, consider running the script in smaller sections, especially in environments like Jupyter Notebook. This allows you to:

* Execute Code Blocks Individually: Understand each step's output.
* Modify Parameters on the Fly: Experiment with different settings without rerunning the entire script.

A complete script for the case studies used in this short paper. A companion commentary is also provided.

# Import necessary libraries for numerical operations, data visualization, and ML tasks

import numpy as np  # NumPy for numerical computations

import matplotlib.pyplot as plt  # Matplotlib for plotting graphs

import pandas as pd  # Pandas for data manipulation and analysis

import os  # OS module for interacting with the operating system

import shap  # SHAP for model explainability

from sklearn.model\_selection import train\_test\_split, cross\_val\_predict  # Functions for splitting data and cross-validation

from sklearn.metrics import (

    mean\_absolute\_error, r2\_score,  # Metrics for regression tasks

    accuracy\_score, f1\_score,  # Metrics for classification tasks

    silhouette\_score, davies\_bouldin\_score  # Metrics for clustering tasks

)

from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier  # ML models for regression and classification

from sklearn.cluster import KMeans  # ML model for clustering

from sklearn.decomposition import PCA  # Principal Component Analysis for dimensionality reduction

from sklearn.metrics import ConfusionMatrixDisplay  # Function to display confusion matrices

from matplotlib.gridspec import GridSpec  # Grid specification for complex layouts in Matplotlib

# -------------------------------

# 1. Task Selection

# -------------------------------

# Set a seed to ensure reproducible results every time the code runs

np.random.seed(42)  # Seed for randomness

# Choose the ML task you want to perform: 'regression', 'classification', or 'clustering'

TASK = 'regression'  # You can change this to 'classification' or 'clustering' as needed

# -------------------------------

# 2. Toggle Mechanism

# -------------------------------

# Decide whether to use your own dataset or use synthetic (generated) data

USE\_USER\_DATA = False  # Set to True if you want to load your own data files

# Choose a single model type (if COMPARE\_ALL=False). Options:

# For regression/classification: 'random\_forest', 'xgboost', 'knn'

# For clustering: 'kmeans', 'birch', 'gaussian\_mixture'

MODEL\_TYPE = 'random\_forest'

# Compare all algorithms for the selected TASK if True.

# If True, we loop over all relevant models and produce multi-model metrics & plots (but skip SHAP).

COMPARE\_ALL = False

# -------------------------------

# 3. Synthetic Data Definition

# -------------------------------

# Define synthetic datasets for each ML task

# Synthetic data for Regression

X\_reg\_synthetic = np.random.rand(200, 1) \* 10  # 200 samples, 1 feature scaled by 10

y\_reg\_synthetic = 3.5 \* X\_reg\_synthetic.squeeze() + np.random.randn(200) \* 2  # Linear relationship with noise

# Synthetic data for Classification

X\_class\_synthetic = np.random.rand(200, 2)  # 200 samples, 2 features

y\_class\_synthetic = (X\_class\_synthetic[:, 0] + X\_class\_synthetic[:, 1] > 1).astype(int)  # Binary target based on feature sum

# Synthetic data for Clustering

X\_cluster\_synthetic = np.random.rand(200, 2) \* 10  # 200 samples, 2 features scaled by 10

# -------------------------------

# 4. Data Loading Functionality

# -------------------------------

def load\_data(file\_path, task):

    """

    Load data from an Excel or CSV file.

    Parameters:

        file\_path (str): Path to the data file.

        task (str): The task type ('regression', 'classification', 'clustering').

    Returns:

        X (np.ndarray): Features.

        y (np.ndarray or None): Target. None for clustering.

    """

    # Check if the file exists in the specified path

    if not os.path.exists(file\_path):

        raise FileNotFoundError(f"File '{file\_path}' does not exist in the current directory.")

    # Load data based on file extension

    if file\_path.endswith('.xlsx'):

        data = pd.read\_excel(file\_path)  # Read Excel file

    elif file\_path.endswith('.csv'):

        data = pd.read\_csv(file\_path)  # Read CSV file

    else:

        raise ValueError("Unsupported file format. Please provide an Excel (.xlsx) or CSV (.csv) file.")

    # Process data based on the selected task

    if task in ['regression', 'classification']:

        if data.shape[1] < 2:

            raise ValueError("Data must contain at least one feature column and one target column.")

        X = data.iloc[:, :-1].values  # All columns except the last are input features

        y = data.iloc[:, -1].values   # Last column is the target variable

    elif task == 'clustering':

        X = data.values  # All columns are features for clustering

        y = None  # No target variable for clustering

    else:

        raise ValueError("Unsupported task. Choose from 'regression', 'classification', or 'clustering'.")

    return X, y  # Return features and target

# -------------------------------

# 5. Data Preparation

# -------------------------------

if USE\_USER\_DATA:

    # Define file paths based on the selected task

    if TASK == 'regression':

        data\_file = 'regression\_dataset.xlsx'       # Replace with your actual regression data file

    elif TASK == 'classification':

        data\_file = 'classification\_dataset.xlsx'   # Replace with your actual classification data file

    elif TASK == 'clustering':

        data\_file = 'clustering\_dataset.csv'        # Replace with your actual clustering data file

    else:

        raise ValueError("Unsupported task. Choose from 'regression', 'classification', or 'clustering'.")

    try:

        X, y = load\_data(data\_file, TASK)  # Load your own data

        print(f"Loaded {TASK} data from '{data\_file}'.")  # Confirm successful loading

    except Exception as e:

        print(f"Error loading {TASK} data: {e}")  # Print error if loading fails

        exit(1)  # Exit the program if data loading fails

else:

    # Use synthetic data based on the selected task

    if TASK == 'regression':

        X = X\_reg\_synthetic

        y = y\_reg\_synthetic

        print("Using synthetic regression data.")  # Inform the user

    elif TASK == 'classification':

        X = X\_class\_synthetic

        y = y\_class\_synthetic

        print("Using synthetic classification data.")  # Inform the user

    elif TASK == 'clustering':

        X = X\_cluster\_synthetic

        y = None

        print("Using synthetic clustering data.")  # Inform the user

    else:

        raise ValueError("Unsupported task. Choose from 'regression', 'classification', or 'clustering'.")

# -------------------------------

# 6. Split Data into Training and Test Sets (Supervised Tasks)

# -------------------------------

if TASK in ['regression', 'classification']:

    # For regression and classification, split 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  # 80% training, 20% testing

    )

else:

    # For clustering, use all data for training (no target variable)

    X\_train = X

    X\_test = X

    y\_train = None

    y\_test = None

# -------------------------------

# 7. Models for Each Task

# -------------------------------

def get\_all\_models(task):

    """

    Returns a dictionary of {model\_name: model\_object} for the given task.

    """

    models = {}

    if task == 'regression':

        # RandomForest, XGBoost, KNN

        from sklearn.ensemble import RandomForestRegressor

        from sklearn.neighbors import KNeighborsRegressor

        random\_forest = RandomForestRegressor(n\_estimators=100, random\_state=42)

        models['random\_forest'] = random\_forest

        try:

            from xgboost import XGBRegressor

            xgboost\_model = XGBRegressor(random\_state=42)

            models['xgboost'] = xgboost\_model

        except ImportError:

            pass  # if xgboost not installed, skip

        knn = KNeighborsRegressor(n\_neighbors=5)

        models['knn'] = knn

    elif task == 'classification':

        # RandomForest, XGBoost, KNN

        from sklearn.ensemble import RandomForestClassifier

        from sklearn.neighbors import KNeighborsClassifier

        random\_forest = RandomForestClassifier(n\_estimators=100, random\_state=42)

        models['random\_forest'] = random\_forest

        try:

            from xgboost import XGBClassifier

            xgboost\_model = XGBClassifier(random\_state=42)

            models['xgboost'] = xgboost\_model

        except ImportError:

            pass

        knn = KNeighborsClassifier(n\_neighbors=5)

        models['knn'] = knn

    elif task == 'clustering':

        # KMeans, Birch, GaussianMixture

        from sklearn.cluster import KMeans, Birch

        from sklearn.mixture import GaussianMixture

        kmeans = KMeans(n\_clusters=2, random\_state=42)

        models['kmeans'] = kmeans

        birch = Birch(n\_clusters=2)

        models['birch'] = birch

        gmm = GaussianMixture(n\_components=2, random\_state=42)

        models['gaussian\_mixture'] = gmm

    else:

        raise ValueError("Unsupported task. Choose from 'regression', 'classification', or 'clustering'.")

    return models

# -------------------------------

# 7. Initialize a Default Model

# -------------------------------

if TASK == 'regression':

    model = RandomForestRegressor(n\_estimators=100, random\_state=42)  # Default

elif TASK == 'classification':

    model = RandomForestClassifier(n\_estimators=100, random\_state=42)  # Default

elif TASK == 'clustering':

    model = KMeans(n\_clusters=2, random\_state=42)  # Default

else:

    raise ValueError("Unsupported task. Choose from 'regression', 'classification', or 'clustering'.")

# -------------------------------

# 8. Model Selection Logic

# -------------------------------

if not COMPARE\_ALL:

    if TASK == 'regression':

        if MODEL\_TYPE == 'xgboost':

            from xgboost import XGBRegressor

            model = XGBRegressor(random\_state=42)

        elif MODEL\_TYPE == 'knn':

            from sklearn.neighbors import KNeighborsRegressor

            model = KNeighborsRegressor(n\_neighbors=5)

        elif MODEL\_TYPE == 'random\_forest':

            pass  # Use default

    elif TASK == 'classification':

        if MODEL\_TYPE == 'xgboost':

            from xgboost import XGBClassifier

            model = XGBClassifier(random\_state=42)

        elif MODEL\_TYPE == 'knn':

            from sklearn.neighbors import KNeighborsClassifier

            model = KNeighborsClassifier(n\_neighbors=5)

        elif MODEL\_TYPE == 'random\_forest':

            pass  # Use default

    elif TASK == 'clustering':

        if MODEL\_TYPE == 'birch':

            from sklearn.cluster import Birch

            model = Birch(n\_clusters=2)

        elif MODEL\_TYPE == 'gaussian\_mixture':

            from sklearn.mixture import GaussianMixture

            model = GaussianMixture(n\_components=2, random\_state=42)

        elif MODEL\_TYPE == 'kmeans':

            pass  # Use default

# For multi-model compare

multi\_model\_metrics = []

all\_models = None

# -------------------------------

# 9. Training: Single vs Multi

# -------------------------------

if not COMPARE\_ALL:

    # Single-model workflow

    if TASK in ['regression', 'classification']:

        y\_cv\_pred = cross\_val\_predict(model, X\_train, y\_train, cv=5)

        model.fit(X\_train, y\_train)

        y\_train\_pred = model.predict(X\_train)

        y\_test\_pred = model.predict(X\_test)

    else:  # clustering

        model.fit(X\_train)

        # For KMeans/Birch/GaussianMixture .predict(...) works

        cluster\_labels = model.predict(X\_test)

else:

    # Multi-model workflow

    all\_models = get\_all\_models(TASK)

    for name, this\_model in all\_models.items():

        print(f"Training model: {name} ...")

        if TASK in ['regression', 'classification']:

            y\_cv\_pred = cross\_val\_predict(this\_model, X\_train, y\_train, cv=5)

            this\_model.fit(X\_train, y\_train)

            y\_train\_pred = this\_model.predict(X\_train)

            y\_test\_pred = this\_model.predict(X\_test)

            if TASK == 'regression':

                data\_metrics = {

                    "Model": name,

                    "Train R2": r2\_score(y\_train, y\_train\_pred),

                    "Train MAE": mean\_absolute\_error(y\_train, y\_train\_pred),

                    "CV R2": r2\_score(y\_train, y\_cv\_pred),

                    "CV MAE": mean\_absolute\_error(y\_train, y\_cv\_pred),

                    "Test R2": r2\_score(y\_test, y\_test\_pred),

                    "Test MAE": mean\_absolute\_error(y\_test, y\_test\_pred),

                    "\_preds\_": {

                        "train\_pred": y\_train\_pred,

                        "test\_pred": y\_test\_pred,

                        "cv\_pred": y\_cv\_pred

                    }

                }

            else:

                data\_metrics = {

                    "Model": name,

                    "Train Accuracy": accuracy\_score(y\_train, this\_model.predict(X\_train)),

                    "Train F1": f1\_score(y\_train, this\_model.predict(X\_train), average='weighted'),

                    "CV Accuracy": accuracy\_score(y\_train, y\_cv\_pred),

                    "CV F1": f1\_score(y\_train, y\_cv\_pred, average='weighted'),

                    "Test Accuracy": accuracy\_score(y\_test, this\_model.predict(X\_test)),

                    "Test F1": f1\_score(y\_test, this\_model.predict(X\_test), average='weighted'),

                    "\_preds\_": {

                        "train\_pred": y\_train\_pred,

                        "test\_pred": y\_test\_pred,

                        "cv\_pred": y\_cv\_pred

                    }

                }

            multi\_model\_metrics.append(data\_metrics)

        else:

            # For kmeans, birch, gaussian\_mixture

            this\_model.fit(X\_train)

            cluster\_labels = this\_model.predict(X\_test)

            data\_metrics = {

                "Model": name,

                "Silhouette": silhouette\_score(X\_test, cluster\_labels),

                "Davies-Bouldin": davies\_bouldin\_score(X\_test, cluster\_labels),

                "\_labels\_": cluster\_labels

            }

            multi\_model\_metrics.append(data\_metrics)

# -------------------------------

# 10. Metrics (Single Model)

# -------------------------------

if not COMPARE\_ALL:

    if TASK == 'regression':

        metrics = {

            "Metric": ["R²", "MAE"],

            "Training": [

                r2\_score(y\_train, y\_train\_pred),

                mean\_absolute\_error(y\_train, y\_train\_pred)

            ],

            "Cross-Validation": [

                r2\_score(y\_train, y\_cv\_pred),

                mean\_absolute\_error(y\_train, y\_cv\_pred)

            ],

            "Test": [

                r2\_score(y\_test, y\_test\_pred),

                mean\_absolute\_error(y\_test, y\_test\_pred)

            ],

        }

        metrics\_df = pd.DataFrame(metrics)

    elif TASK == 'classification':

        metrics = {

            "Metric": ["Accuracy", "F1-Score"],

            "Training": [

                accuracy\_score(y\_train, model.predict(X\_train)),

                f1\_score(y\_train, model.predict(X\_train), average='weighted')

            ],

            "Cross-Validation": [

                accuracy\_score(y\_train, y\_cv\_pred),

                f1\_score(y\_train, y\_cv\_pred, average='weighted')

            ],

            "Test": [

                accuracy\_score(y\_test, model.predict(X\_test)),

                f1\_score(y\_test, model.predict(X\_test), average='weighted')

            ],

        }

        metrics\_df = pd.DataFrame(metrics)

    elif TASK == 'clustering':

        metrics = {

            "Metric": ["Silhouette Score", "Davies-Bouldin Index"],

            "Score": [

                silhouette\_score(X\_test, cluster\_labels),

                davies\_bouldin\_score(X\_test, cluster\_labels)

            ],

        }

        metrics\_df = pd.DataFrame(metrics)

    else:

        raise ValueError("Unsupported task. Choose from 'regression', 'classification', or 'clustering'.")

# -------------------------------

# 11. Results and Visualization

# -------------------------------

if not COMPARE\_ALL:

    styled\_metrics = metrics\_df.style.set\_properties(\*\*{'text-align': 'center'})\

                                    .set\_table\_styles([

                                        {'selector': 'th', 'props': [('text-align', 'center')]},

                                        {'selector': 'td', 'props': [('text-align', 'center')]}

                                    ])\

                                    .set\_caption("Model Performance Metrics (Single Model)")

    try:

        from IPython.display import display

        display(styled\_metrics)

    except ImportError:

        print(metrics\_df)

else:

    # For multi-model comparison, exclude the "\_preds\_" or "\_labels\_" columns:

    comparison\_list = []

    for d in multi\_model\_metrics:

        newd = {k: v for k, v in d.items() if not k.startswith("\_")}

        comparison\_list.append(newd)

    comparison\_df = pd.DataFrame(comparison\_list)

    try:

        from IPython.display import display

        display(comparison\_df)

    except ImportError:

        print(comparison\_df)

# Plots

if not COMPARE\_ALL:

    if TASK in ['regression', 'classification']:

        def plot\_results\_supervised(ax, y\_actual, y\_pred, title, color, metric1, metric2):

            if TASK == 'regression':

                ax.scatter(y\_actual, y\_actual, color=color, label='Actual', marker='o', s=60, alpha=0.7, edgecolors='none')

                ax.scatter(y\_actual, y\_pred, facecolors='none', edgecolors=color, label='Predicted', marker='o', s=60, alpha=0.7)

                min\_val = min(np.min(y\_actual), np.min(y\_pred))

                max\_val = max(np.max(y\_actual), np.max(y\_pred))

                ax.plot([min\_val, max\_val], [min\_val, max\_val], 'k--', linewidth=1)

                ax.set\_title(title, fontsize=14)

                ax.set\_xlabel('Actual Values', fontsize=12)

                ax.set\_ylabel('Predicted Values', fontsize=12)

                ax.set\_aspect('equal', adjustable='box')

                legend = ax.legend(title=f'R²={metric1:.2f}, MAE={metric2:.2f}')

            elif TASK == 'classification':

                ConfusionMatrixDisplay.from\_predictions(y\_actual, y\_pred, ax=ax, cmap='Blues', colorbar=False)

                ax.set\_title(title, fontsize=14)

                ax.text(0.95, 0.05, f'Accuracy={metric1:.2f}\nF1-Score={metric2:.2f}',

                        verticalalignment='bottom', horizontalalignment='right',

                        transform=ax.transAxes,

                        color='black', fontsize=12,

                        bbox=dict(facecolor='white', alpha=0.5, boxstyle='round,pad=0.5'))

            ax.grid(True, linestyle='--', linewidth=0.5)

        if TASK == 'regression':

            fig = plt.figure(constrained\_layout=True, figsize=(20, 6))

            gs = GridSpec(1, 3, figure=fig, wspace=0.3)

            ax1 = fig.add\_subplot(gs[0, 0])

            ax2 = fig.add\_subplot(gs[0, 1])

            ax3 = fig.add\_subplot(gs[0, 2])

            train\_r2 = metrics\_df.loc[metrics\_df['Metric'] == 'R²', 'Training'].values[0]

            train\_mae = metrics\_df.loc[metrics\_df['Metric'] == 'MAE', 'Training'].values[0]

            cv\_r2 = metrics\_df.loc[metrics\_df['Metric'] == 'R²', 'Cross-Validation'].values[0]

            cv\_mae = metrics\_df.loc[metrics\_df['Metric'] == 'MAE', 'Cross-Validation'].values[0]

            test\_r2 = metrics\_df.loc[metrics\_df['Metric'] == 'R²', 'Test'].values[0]

            test\_mae = metrics\_df.loc[metrics\_df['Metric'] == 'MAE', 'Test'].values[0]

            plot\_results\_supervised(ax1, y\_train, y\_train\_pred, 'Training Data', color='blue', metric1=train\_r2, metric2=train\_mae)

            plot\_results\_supervised(ax2, y\_train, y\_cv\_pred, 'Cross-Validation', color='red', metric1=cv\_r2, metric2=cv\_mae)

            plot\_results\_supervised(ax3, y\_test, y\_test\_pred, 'Test Data', color='green', metric1=test\_r2, metric2=test\_mae)

            fig.suptitle('Regression Results: Actual vs. Predicted', fontsize=16)

            plt.show()

        elif TASK == 'classification':

            fig = plt.figure(constrained\_layout=True, figsize=(20, 6))

            gs = GridSpec(1, 3, figure=fig, wspace=0.3)

            ax1 = fig.add\_subplot(gs[0, 0])

            ax2 = fig.add\_subplot(gs[0, 1])

            ax3 = fig.add\_subplot(gs[0, 2])

            train\_acc = metrics\_df.loc[metrics\_df['Metric'] == 'Accuracy', 'Training'].values[0]

            train\_f1 = metrics\_df.loc[metrics\_df['Metric'] == 'F1-Score', 'Training'].values[0]

            cv\_acc = metrics\_df.loc[metrics\_df['Metric'] == 'Accuracy', 'Cross-Validation'].values[0]

            cv\_f1 = metrics\_df.loc[metrics\_df['Metric'] == 'F1-Score', 'Cross-Validation'].values[0]

            test\_acc = metrics\_df.loc[metrics\_df['Metric'] == 'Accuracy', 'Test'].values[0]

            test\_f1 = metrics\_df.loc[metrics\_df['Metric'] == 'F1-Score', 'Test'].values[0]

            plot\_results\_supervised(ax1, y\_train, model.predict(X\_train), 'Training Data', color='blue', metric1=train\_acc, metric2=train\_f1)

            plot\_results\_supervised(ax2, y\_train, y\_cv\_pred, 'Cross-Validation', color='red', metric1=cv\_acc, metric2=cv\_f1)

            plot\_results\_supervised(ax3, y\_test, y\_test\_pred, 'Test Data', color='green', metric1=test\_acc, metric2=test\_f1)

            fig.suptitle('Classification Results: Confusion Matrices', fontsize=16)

            plt.show()

    elif TASK == 'clustering':

        pca = PCA(n\_components=2)

        X\_pca = pca.fit\_transform(X\_test)

        fig = plt.figure(constrained\_layout=True, figsize=(10, 6))

        gs = GridSpec(1, 1, figure=fig)

        ax = fig.add\_subplot(gs[0, 0])

        scatter = ax.scatter(X\_pca[:, 0], X\_pca[:, 1], c=cluster\_labels, cmap='viridis', alpha=0.7, edgecolors='k', s=60)

        ax.set\_title('Clustering Results (PCA-Reduced Data)', fontsize=16)

        ax.set\_xlabel('PCA Component 1', fontsize=12)

        ax.set\_ylabel('PCA Component 2', fontsize=12)

        legend1 = ax.legend(\*scatter.legend\_elements(), title="Clusters")

        ax.add\_artist(legend1)

        ax.grid(True, linestyle='--', linewidth=0.5)

        plt.show()

# -------------------------------

# 12. Additional Plots for Multi-Model

# -------------------------------

elif COMPARE\_ALL:

    if TASK in ['regression', 'classification']:

        def plot\_results\_supervised(ax, y\_actual, y\_pred, title, color, metric1, metric2):

            if TASK == 'regression':

                ax.scatter(y\_actual, y\_actual, color=color, label='Actual', marker='o', s=60, alpha=0.7, edgecolors='none')

                ax.scatter(y\_actual, y\_pred, facecolors='none', edgecolors=color, label='Predicted', marker='o', s=60, alpha=0.7)

                min\_val = min(np.min(y\_actual), np.min(y\_pred))

                max\_val = max(np.max(y\_actual), np.max(y\_pred))

                ax.plot([min\_val, max\_val], [min\_val, max\_val], 'k--', linewidth=1)

                ax.set\_title(title, fontsize=14)

                ax.set\_xlabel('Actual Values', fontsize=12)

                ax.set\_ylabel('Predicted Values', fontsize=12)

                ax.set\_aspect('equal', adjustable='box')

                legend = ax.legend(title=f'R²={metric1:.2f}, MAE={metric2:.2f}')

            elif TASK == 'classification':

                ConfusionMatrixDisplay.from\_predictions(y\_actual, y\_pred, ax=ax, cmap='Blues', colorbar=False)

                ax.set\_title(title, fontsize=14)

                ax.text(0.95, 0.05, f'Accuracy={metric1:.2f}\nF1-Score={metric2:.2f}',

                        verticalalignment='bottom', horizontalalignment='right',

                        transform=ax.transAxes,

                        color='black', fontsize=12,

                        bbox=dict(facecolor='white', alpha=0.5, boxstyle='round,pad=0.5'))

            ax.grid(True, linestyle='--', linewidth=0.5)

        rows = len(multi\_model\_metrics)

        fig = plt.figure(constrained\_layout=True, figsize=(20, 4 \* rows))

        gs = GridSpec(rows, 3, figure=fig, wspace=0.3, hspace=0.3)

        for i, data\_metrics in enumerate(multi\_model\_metrics):

            name = data\_metrics["Model"]

            \_preds\_ = data\_metrics["\_preds\_"]

            train\_pred = \_preds\_["train\_pred"]

            cv\_pred = \_preds\_["cv\_pred"]

            test\_pred = \_preds\_["test\_pred"]

            ax1 = fig.add\_subplot(gs[i, 0])

            ax2 = fig.add\_subplot(gs[i, 1])

            ax3 = fig.add\_subplot(gs[i, 2])

            if TASK == 'regression':

                train\_r2 = data\_metrics["Train R2"]

                train\_mae = data\_metrics["Train MAE"]

                cv\_r2 = data\_metrics["CV R2"]

                cv\_mae = data\_metrics["CV MAE"]

                test\_r2 = data\_metrics["Test R2"]

                test\_mae = data\_metrics["Test MAE"]

                plot\_results\_supervised(ax1, y\_train, train\_pred, f'{name} - Train', 'blue', train\_r2, train\_mae)

                plot\_results\_supervised(ax2, y\_train, cv\_pred, f'{name} - CV', 'red', cv\_r2, cv\_mae)

                plot\_results\_supervised(ax3, y\_test, test\_pred, f'{name} - Test', 'green', test\_r2, test\_mae)

            else:  # classification

                train\_acc = data\_metrics["Train Accuracy"]

                train\_f1 = data\_metrics["Train F1"]

                cv\_acc = data\_metrics["CV Accuracy"]

                cv\_f1 = data\_metrics["CV F1"]

                test\_acc = data\_metrics["Test Accuracy"]

                test\_f1 = data\_metrics["Test F1"]

                plot\_results\_supervised(ax1, y\_train, train\_pred, f'{name} - Train', 'blue', train\_acc, train\_f1)

                plot\_results\_supervised(ax2, y\_train, cv\_pred, f'{name} - CV', 'red', cv\_acc, cv\_f1)

                plot\_results\_supervised(ax3, y\_test, test\_pred, f'{name} - Test', 'green', test\_acc, test\_f1)

        title\_txt = 'Multi-Model Results: Regression' if TASK == 'regression' else 'Multi-Model Results: Classification'

        fig.suptitle(title\_txt, fontsize=16)

        plt.show()

    elif TASK == 'clustering':

        rows = len(multi\_model\_metrics)

        fig = plt.figure(constrained\_layout=True, figsize=(10, 5 \* rows))

        gs = GridSpec(rows, 1, figure=fig, wspace=0.3, hspace=0.3)

        for i, data\_metrics in enumerate(multi\_model\_metrics):

            name = data\_metrics["Model"]

            cluster\_labels = data\_metrics["\_labels\_"]

            pca = PCA(n\_components=2)

            X\_pca = pca.fit\_transform(X\_test)

            ax = fig.add\_subplot(gs[i, 0])

            scatter = ax.scatter(X\_pca[:, 0], X\_pca[:, 1], c=cluster\_labels, cmap='viridis', alpha=0.7, edgecolors='k', s=60)

            ax.set\_title(f'Clustering Results ({name})', fontsize=14)

            ax.set\_xlabel('PCA Component 1', fontsize=12)

            ax.set\_ylabel('PCA Component 2', fontsize=12)

            legend1 = ax.legend(\*scatter.legend\_elements(), title="Clusters")

            ax.add\_artist(legend1)

            ax.grid(True, linestyle='--', linewidth=0.5)

        plt.show()

# -------------------------------

# 13. SHAP Analysis (Single Model)

# -------------------------------

if TASK in ['regression', 'classification'] and not COMPARE\_ALL:

    try:

        import shap

    except ImportError:

        print("SHAP library is not installed. You can install it using 'pip install shap'")

        exit(1)

    def is\_tree\_model\_fn(m):

        return any(x in str(type(m)).lower() for x in ['forest', 'xgb', 'gradientboost'])

    if is\_tree\_model\_fn(model):

        explainer = shap.TreeExplainer(model)

        shap\_values = explainer.shap\_values(X\_test)

    else:

        explainer = shap.KernelExplainer(model.predict, X\_train[:50])

        shap\_values = explainer.shap\_values(X\_test[:50], nsamples=50)

    if USE\_USER\_DATA:

        if TASK in ['regression', 'classification']:

            if data\_file.endswith('.xlsx'):

                feature\_names = pd.read\_excel(data\_file).columns[:-1].tolist()

            else:

                feature\_names = pd.read\_csv(data\_file).columns[:-1].tolist()

    else:

        feature\_names = [f"Feature {i+1}" for i in range(X.shape[1])]

        print(f"Feature names: {feature\_names}")

    if not is\_tree\_model\_fn(model):

        X\_test\_small = X\_test[:50]

        X\_test\_df = pd.DataFrame(X\_test\_small, columns=feature\_names)

    else:

        X\_test\_df = pd.DataFrame(X\_test, columns=feature\_names)

    if TASK == 'classification' and is\_tree\_model\_fn(model):

        if isinstance(shap\_values, list):

            if len(shap\_values) == 2:

                shap\_values\_to\_plot = shap\_values[1]

                print("Using SHAP values for the positive class (class 1).")

            else:

                raise ValueError(f"Expected 2 classes for binary classification, but got {len(shap\_values)} classes.")

        else:

            shap\_values\_to\_plot = shap\_values

            print("SHAP values are not in a list; using directly.")

    else:

        shap\_values\_to\_plot = shap\_values

    print(f"SHAP values type: {type(shap\_values\_to\_plot)}")

    if isinstance(shap\_values\_to\_plot, np.ndarray):

        print(f"SHAP values shape: {shap\_values\_to\_plot.shape}")

    print(f"X\_test\_df shape: {X\_test\_df.shape}")

    print(f"Number of feature names: {len(feature\_names)}")

    print(f"Feature names: {feature\_names}")

    if isinstance(shap\_values\_to\_plot, np.ndarray):

        if shap\_values\_to\_plot.ndim == 2 and shap\_values\_to\_plot.shape[1] != X\_test\_df.shape[1]:

            raise ValueError(

                f"Number of features in shap\_values\_to\_plot ({shap\_values\_to\_plot.shape[1]}) "

                f"does not match number of features in X\_test\_df ({X\_test\_df.shape[1]})"

            )

        assert not np.isnan(shap\_values\_to\_plot).any(), "Error: shap\_values\_to\_plot contains NaN values."

        assert not np.isinf(shap\_values\_to\_plot).any(), "Error: shap\_values\_to\_plot contains infinite values."

    assert not X\_test\_df.isnull().values.any(), "Error: X\_test\_df contains NaN values."

    assert not np.isinf(X\_test\_df.values).any(), "Error: X\_test\_df contains infinite values."

    print("No NaN or infinite values detected in X\_test\_df and shap\_values\_to\_plot.")

    if isinstance(shap\_values\_to\_plot, np.ndarray):

        plt.figure(figsize=(10, 6))

        shap.summary\_plot(shap\_values\_to\_plot, X\_test\_df, plot\_type="bar", show=False)

        plt.title('SHAP Summary Plot (Bar)')

        plt.tight\_layout()

        plt.show()

        plt.figure(figsize=(10, 6))

        shap.summary\_plot(shap\_values\_to\_plot, X\_test\_df, show=False)

        plt.title('SHAP Summary Plot')

        plt.tight\_layout()

        plt.show()

    else:

        print("SHAP values are not a single NumPy array (e.g., might be a list for multiple classes). Skipping summary plots.")

# -------------------------------

# 14. Clustering Evaluation (Optional)

# -------------------------------

if TASK == 'clustering':

    """

    if y\_test is not None:

        from sklearn.metrics import adjusted\_rand\_score, normalized\_mutual\_info\_score

        ari = adjusted\_rand\_score(y\_test, cluster\_labels)

        nmi = normalized\_mutual\_info\_score(y\_test, cluster\_labels)

        print(f"Adjusted Rand Index: {ari:.2f}")

        print(f"Normalized Mutual Information: {nmi:.2f}")

    """

    pass