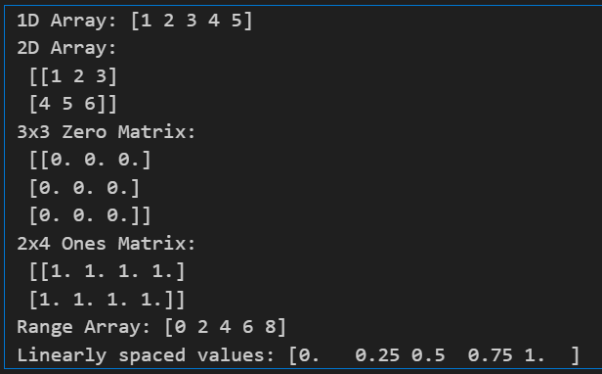
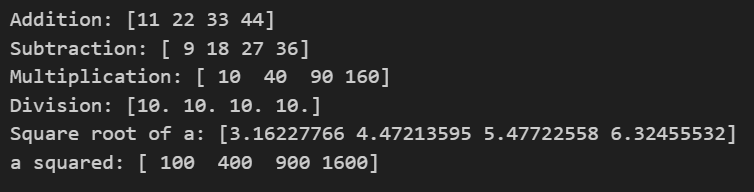
**CODE 1.py**

Numpy

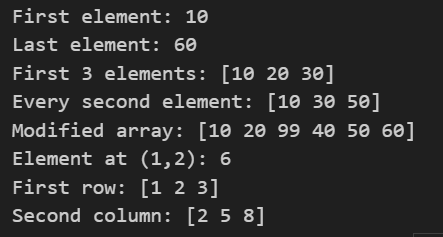
Machine Learning

* Creates a 1D array ([1, 2, 3, 4, 5]).
* Creates a 2D array (matrix).
* Generates a 3x3 zero matrix with np. zeros
* Generates a 2x4 ones matrix with np. ones .
* Creates a range array using np. arange with step size 2.
* Creates an array of linearly spaced values between 0 and 1 using np.linspace.
* It shows different ways to initialize and generate arrays in NumPy.

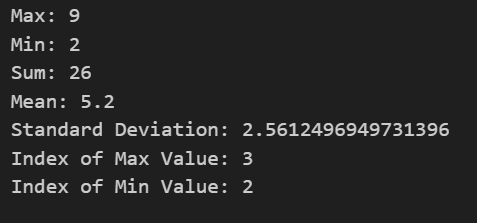
**CODE 2.py**



* Element-wise addition, subtraction, multiplication, and division between two arrays a and b.
* Square root of all elements in array a using np. sqrt.
* Exponentiation: squares each element of a using np.power(a, 2).
* In short: It shows how NumPy supports vectorized mathematical operations directly on arrays.

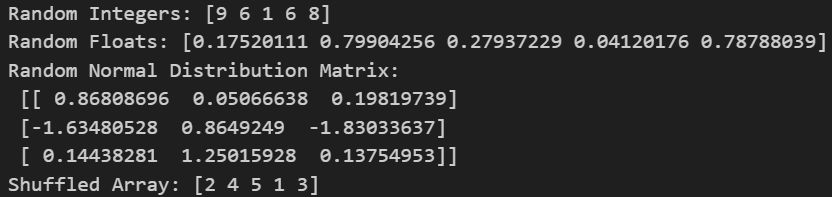
**CODE 3.py**

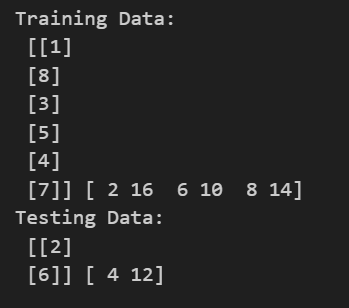
* arr[0] → first element, arr[-1] → last element
* arr[0:3] → first 3, arr[::2] → every 2nd
* arr[2] = 99 → modify element
* mat[1,2] → single element in 2D
* mat[0,:] → row, mat[:,1] → column

**CODE 4.py**

* np.max(arr) —Y finds the maximum value.
* np.min(arr) -4 finds the minimum value.
* np. sum(arr) -4 computes the sum of elements.
* np.mean(arr) —Y computes the average (mean).
* np. std(arr) -4 calculates the standard deviation.
* np.argmax(arr) -4 returns the index of the maximum value.
* np.argmin(arr) -4 returns the index of the minimum value.
* In short: It shows how NumPy makes it easy to perform summary statistics on arrays.

**CODE 5.py**

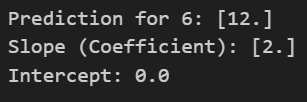


* np.random.randint → random integers in a range
* np.random.rand → random floats [0,1)
* np.random.randn → random normal distribution
* np.random.shuffle → randomly rearranges array
* Random functions help in simulations and testing

**CODE 6.py**

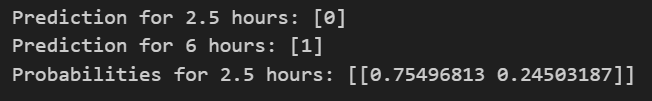
* X → features, y → labels
* train\_test\_split → splits data into training & testing sets
* test\_size=0.2 → 20% data for testing
* random\_state=42 → ensures reproducible split
* X\_train, X\_test, y\_train, y\_test → four output arrays

**CODE 7.py**



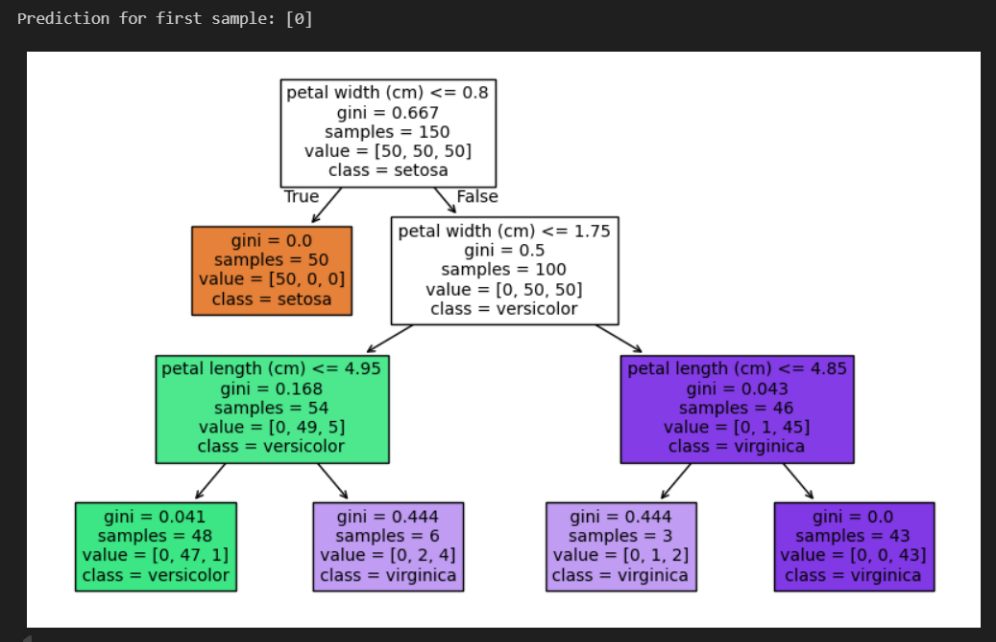
* LinearRegression() → creates a linear model
* model.fit(X, y) → trains model on data
* model.predict([[6]]) → predicts output for new input
* model.coef\_ → slope of the line
* model.intercept\_ → y-intercept

**CODE 8.py**

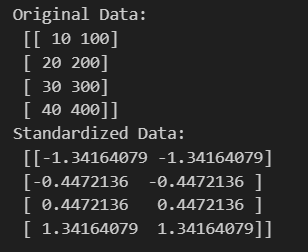
****

* LogisticRegression() → creates a classification model
* model.fit(X, y) → trains on study hours vs pass/fail
* model.predict([[x]]) → predicts class (0 or 1)
* model.predict\_proba([[x]]) → gives probability of each class
* More study hours → higher probability of passing

**CODE 9.py**

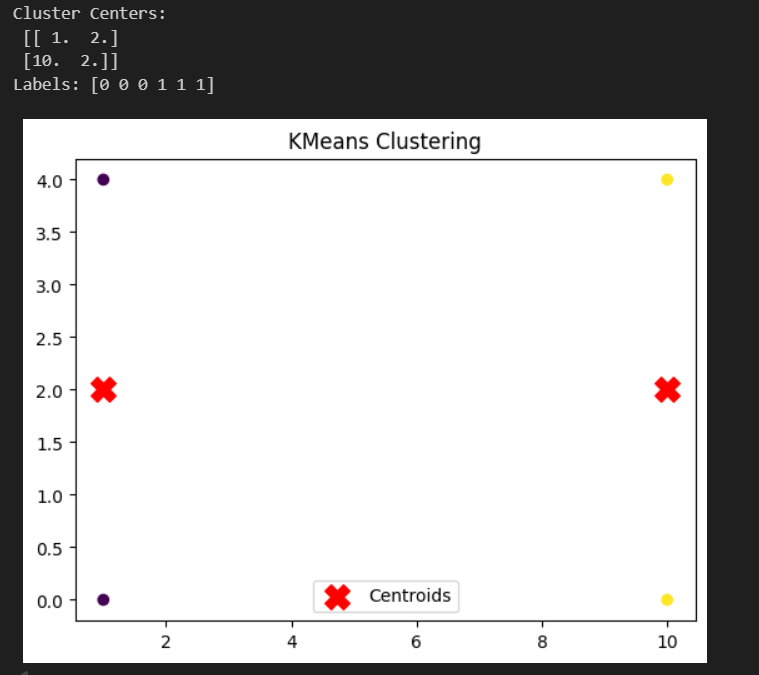
****

* load\_iris() → loads iris dataset (features + labels)
* DecisionTreeClassifier(max\_depth=3) → builds a decision tree with limited depth
* clf.fit(X, y) → trains model on iris data
* clf.predict([X[0]]) → predicts class for first sample
* tree.plot\_tree() → visualizes decision tree with features & classes

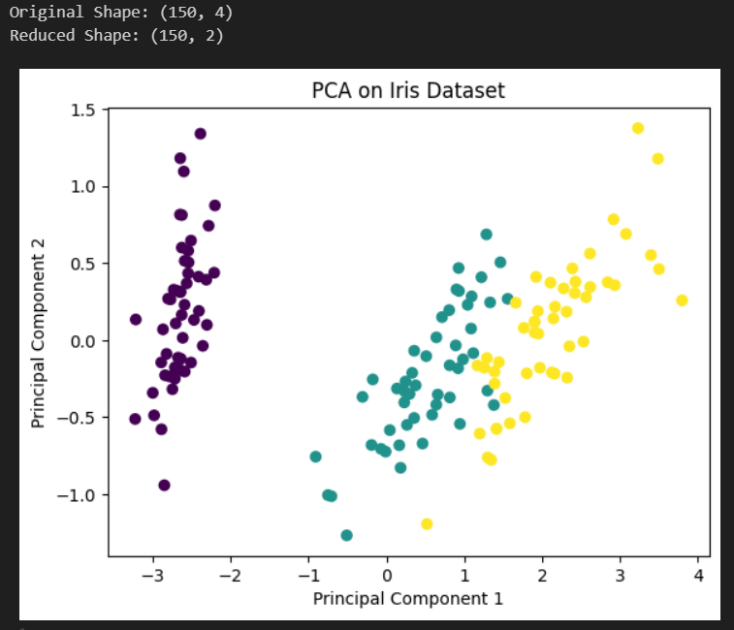
**CODE 10.py**

* StandardScaler() → standardizes features
* fit\_transform(X) → applies scaling to data
* Standardization → mean = 0, std = 1
* Helps models work better with different scales
* Original vs scaled data → different ranges but same info

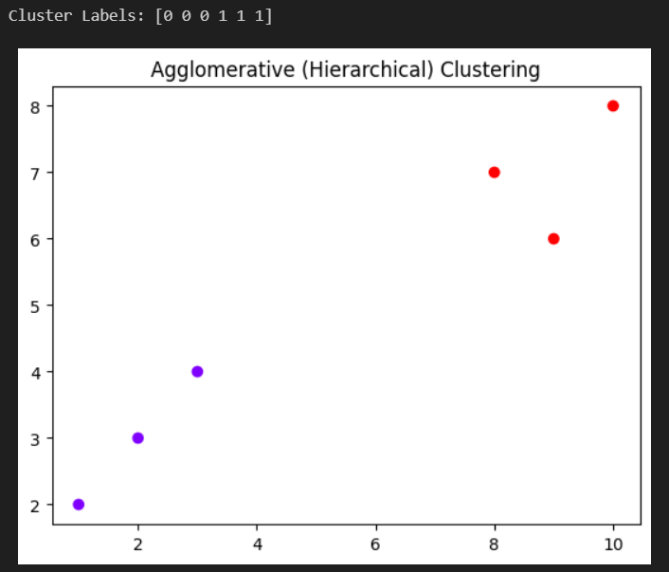
**CODE 11.py**



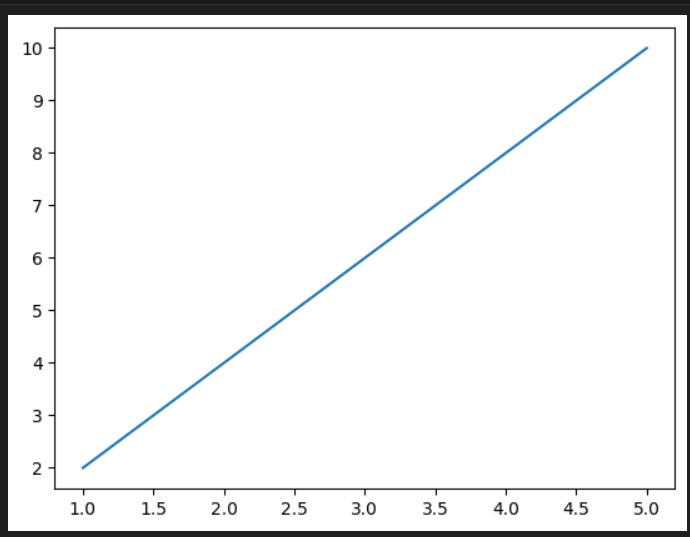
* KMeans(n\_clusters=2) → groups data into 2 clusters
* fit(X) → assigns cluster labels to data points
* cluster\_centers\_ → coordinates of cluster centroids
* labels\_ → shows cluster membership of each point
* Plot → points colored by cluster, red "X" = centroids

**CODE 12.py**

* PCA(n\_components=2) → reduces data to 2 dimensions
* fit\_transform(X) → applies PCA to dataset
* Shape change: 4 features → 2 components
* Scatter plot shows separation of classes
* PCA helps visualize high-dimensional data

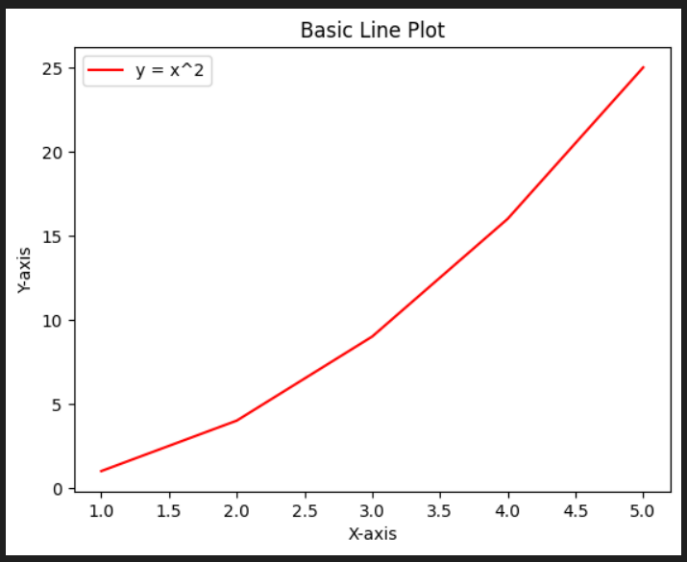
**CODE 13.py**

* AgglomerativeClustering(n\_clusters=2) → groups data hierarchically into 2 clusters
* fit\_predict(X) → assigns cluster labels to points
* labels → shows which cluster each point belongs to
* Plot → colors indicate different clusters
* Hierarchical clustering builds clusters step by step (bottom-up)

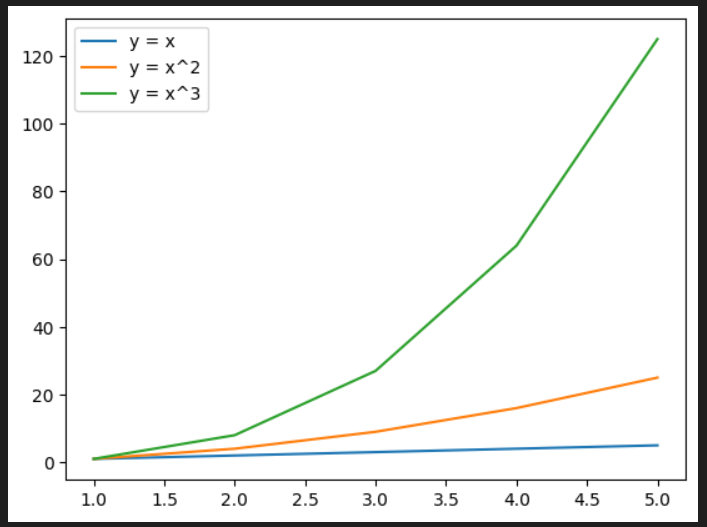
**CODE 1.py**

Matplot

* **\*x and y store the data points.**
* **plt.plot(x, y) draws a line graph of y versus x.**
* **plt.show() displays the plot window.**

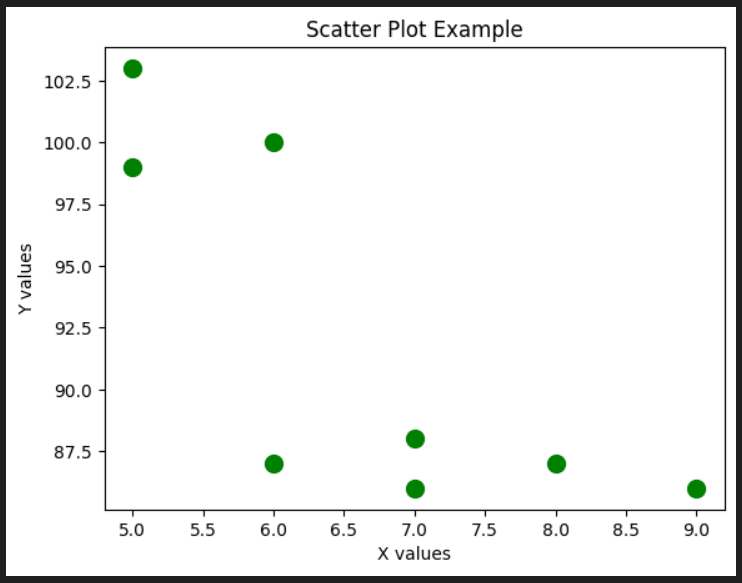
**CODE 2.py**

* The code plots a simple line graph of y=x2y = x^2 with x values from 1 to 5.
* The line is displayed in red because of color="red".
* Labels are added to both the X-axis and Y-axis for clarity.
* A legend is included to describe the line ("y = x^2").
* The graph has a title ("Basic Line Plot") shown at the top.

**CODE 3.py**

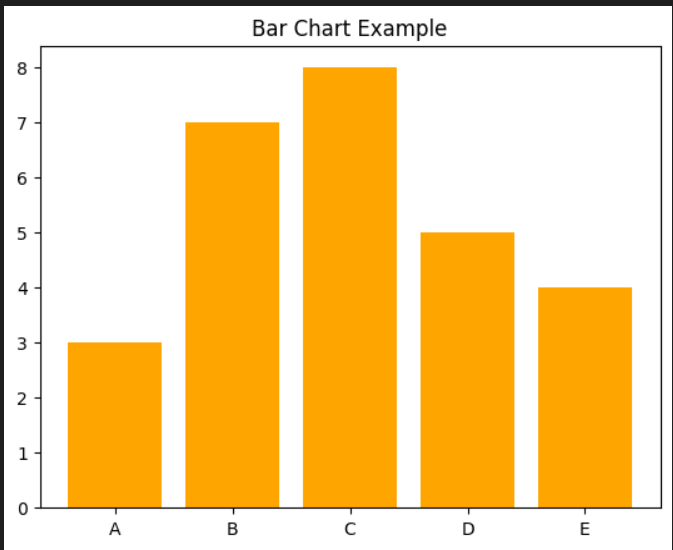
* Three mathematical functions are plotted: linear (y=x), quadratic (y=x²), and cubic (y=x³).
* The same x-values (1–5) are used for all three functions.
* Since no colors are specified, matplotlib assigns default colors (blue, orange, green).
* The legend distinguishes each curve clearly.
* The plot visually demonstrates the increasing growth rate: line < parabola < cubic curve.

**CODE 4.py**

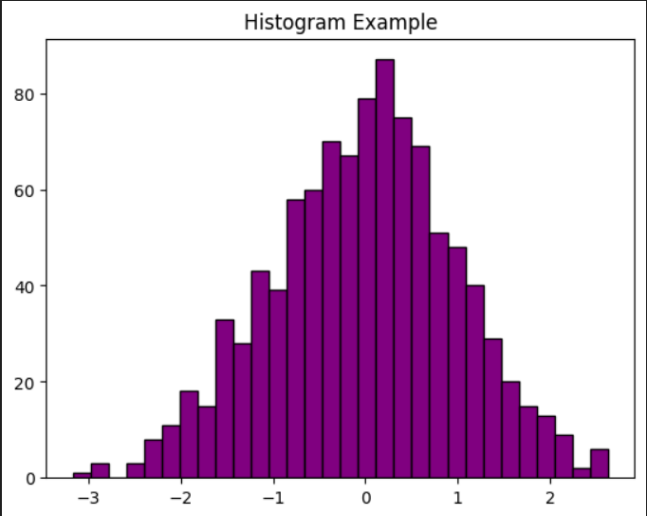
****

* A scatter plot is created using the given x and y values.
* The points are plotted in green with circular markers ("o").
* Each point has a size of 100 because of s=100.
* Axis labels ("X values", "Y values") and a title ("Scatter Plot Example") are added.
* The plot shows how data points are distributed, not connected like a line plot.

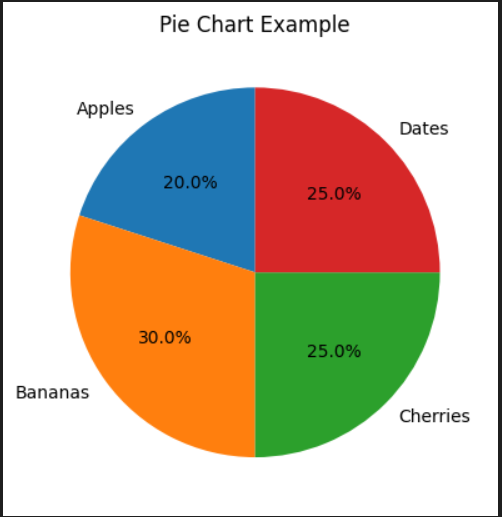
**CODE 5.py**

****

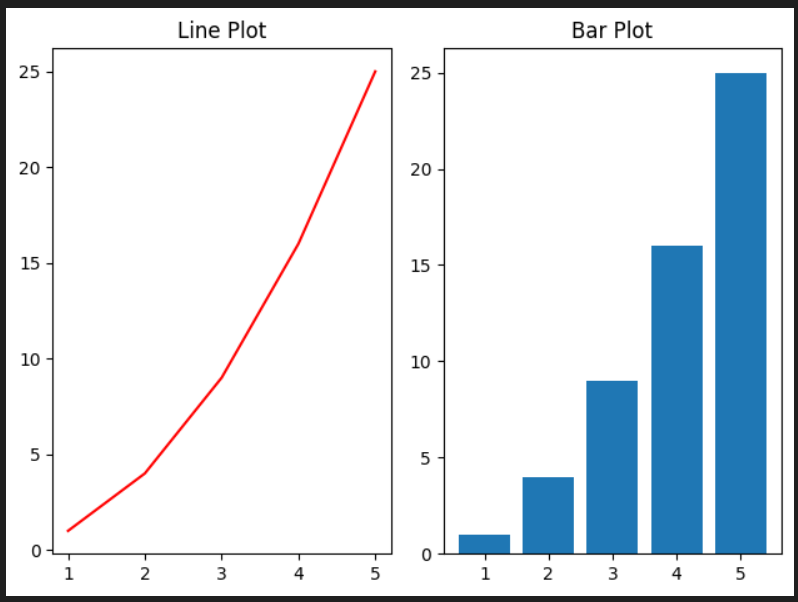
* A bar chart is created with categories "A" to "E" on the x-axis.
* The bar heights are defined by the list y = [3, 7, 8, 5, 4].
* All bars are displayed in orange.
* A title ("Bar Chart Example") is added to the chart.
* No labels are added for the axes, so only the bars and title are shown.

**CODE 6.py**

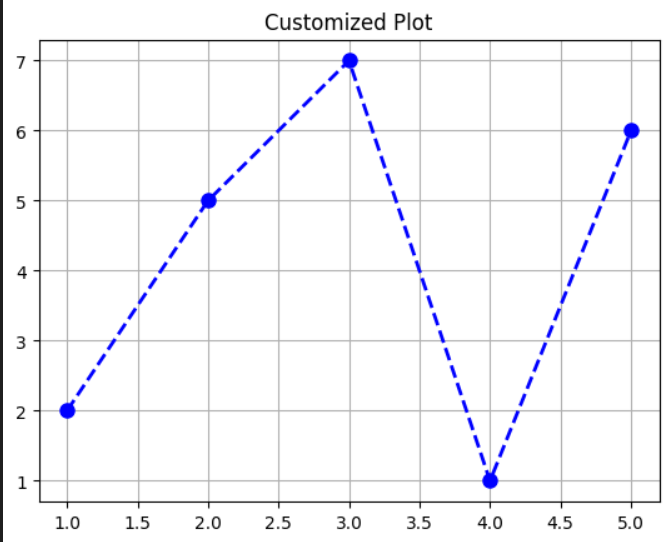
* A histogram is plotted using 1000 randomly generated numbers (np.random.randn).
* The data follows a normal distribution (mean ≈ 0, std ≈ 1).
* The histogram uses 30 bins to group the values.
* Bars are filled in purple with a black edge for clarity.
* A title ("Histogram Example") is displayed, but no axis labels are given.

**CODE 7.py**

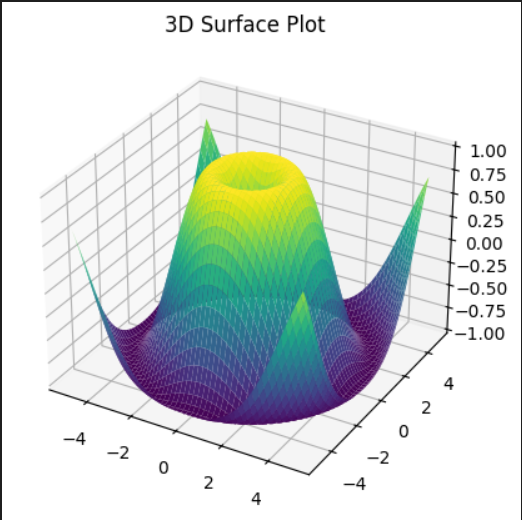
* A pie chart is created with four categories: Apples, Bananas, Cherries, and Dates.
* The sizes list [20, 30, 25, 25] defines the proportions of each slice.
* Percentages are displayed inside slices using autopct="%1.1f%%".
* The chart starts at a 90° angle (startangle=90), so slices are rotated.
* A title ("Pie Chart Example") is added for clarity.

**CODE 8.py**

* The code creates two subplots side by side in a single figure.
* The first subplot (left) is a red line plot of y=x2y = x^2.
* The second subplot (right) is a bar chart of the same data.
* plt.tight\_layout() is used to prevent overlap between titles and plots.
* Both plots share the same x and y values, but are visualized differently.

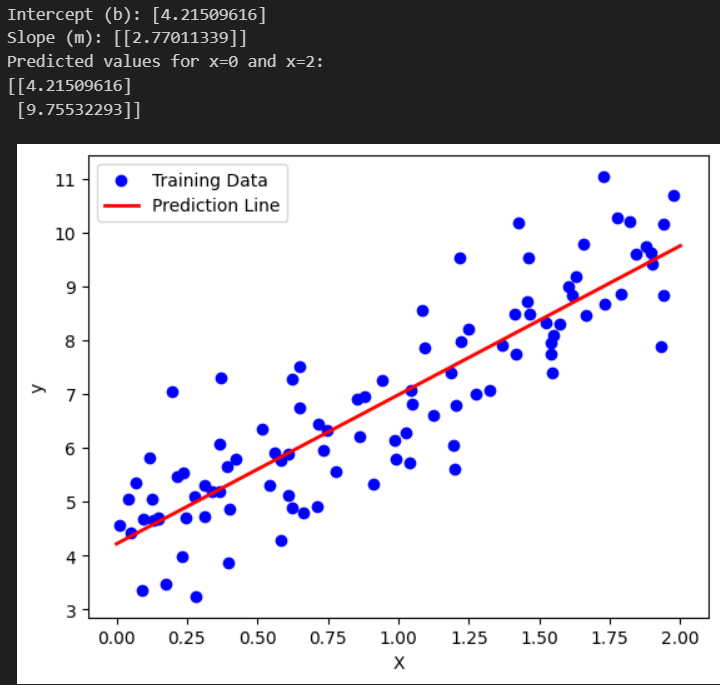
**CODE 9.py**

* A line plot is created with points (x, y) connected by a blue dashed line.
* Each data point is marked with a circle ("o") marker.
* The line width is 2 and marker size is 8, making them clearly visible.
* A grid is enabled (plt.grid(True)) for better readability.
* The plot is given a title: "Customized Plot".

**CODE 10.py**

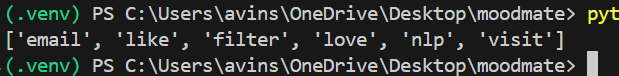
* The code generates a 3D surface plot using Matplotlib’s Axes3D.
* The surface is based on the function Z=sin⁡(X2+Y2)Z = \sin(\sqrt{X^2 + Y^2}).
* np.meshgrid creates a grid of X and Y values ranging from -5 to 5.
* The surface is colored using the "viridis" colormap, which smoothly transitions colors.
* A title ("3D Surface Plot") is added above the 3D figure.

**CODE 1.PY Matplotlib.**

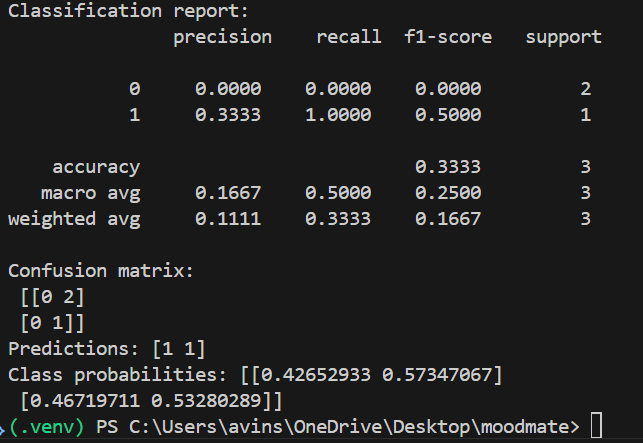
****

* The code generates synthetic linear data with noise using y=4+3x+ϵy = 4 + 3x + \epsilon.
* A Linear Regression model from scikit-learn is trained on the data.
* The model learns the intercept (b) and slope (m), which are printed.
* Predictions are made for x=0x = 0 and x=2x = 2, showing expected outputs near 4 and 10.
* The plot displays training data as blue scatter points and the prediction line in red.

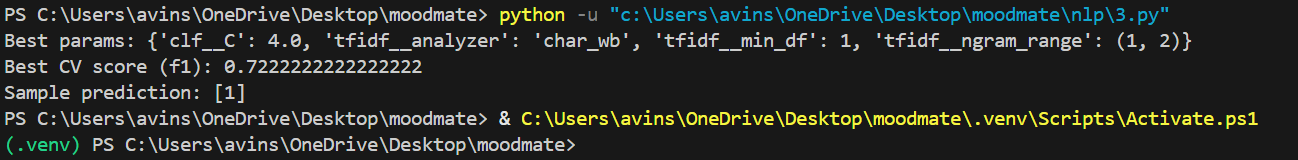
NLP

**file: 1\_preprocess.py**

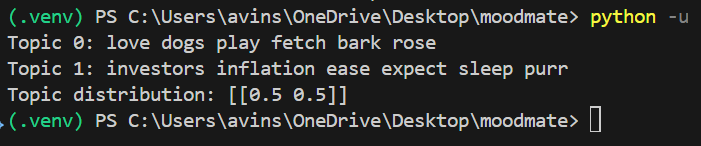
* Cleaning step: The basic\_clean() function lowercases text, removes URLs, emails, mentions, hashtags, and keeps only letters, numbers, spaces, and apostrophes.
* Tokenization & lemmatization: It uses spaCy (en\_core\_web\_sm) to tokenize and get lemmatized forms of words.
* Stopword removal: Stopwords are filtered out using scikit-learn’s built-in stopword list (ENGLISH\_STOP\_WORDS).
* Additional filtering: Tokens that are punctuation, whitespace, or shorter than 3 characters are excluded.
* Pipeline usage: The preprocess() function applies both cleaning and tokenization, returning a list of useful lemmas.

**file: 2\_classify\_tfidf.py**

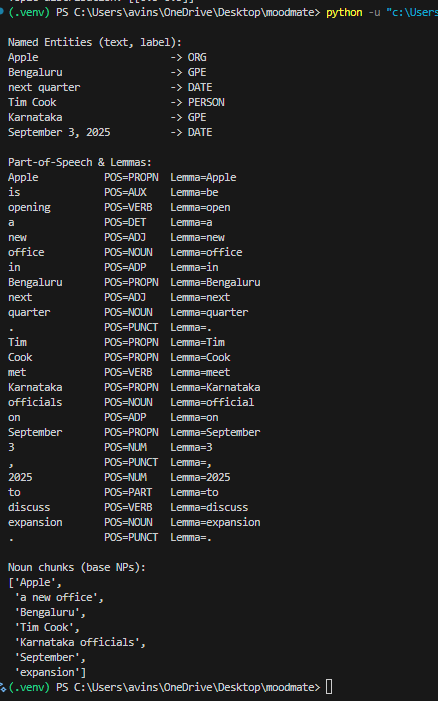
* Dataset: A small demo dataset of 10 movie reviews is created, labeled as positive (1) or negative (0).
* Vectorization: Text is transformed into features using TF–IDF with unigrams and bigrams (ngram\_range=(1,2)).
* Model: A Logistic Regression classifier is trained with up to 1000 iterations.
* Evaluation: The script prints a classification report (precision, recall, F1) and a confusion matrix on the test set.
* Inference: It demonstrates predictions and class probabilities for new sample sentences.

**file: 3\_tune\_grid.py**

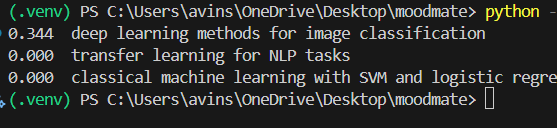
* Pipeline: Combines TfidfVectorizer with LogisticRegression for end-to-end text classification.
* Hyperparameters tuned: It explores n-gram ranges, minimum document frequency, analyzer type (word vs char\_wb), and logistic regression regularization strength (C).
* Search method: Uses GridSearchCV with 3-fold cross validation and F1 score as the evaluation metric.
* Output: Prints the best parameter set, the best CV F1 score, and a sample prediction using the optimized model.
* Flexibility: With character-level features and different C values, the model can adapt to small datasets with varying levels of granularity.

**file: 4\_topic\_lda.py**

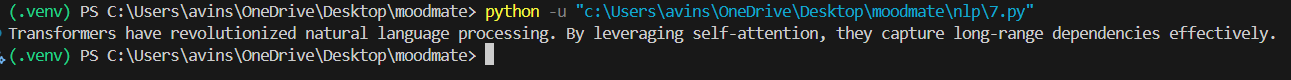
* Takes cv\_results\_ (all scores + params).
* Wraps it in a DataFrame for readability.
* Sorts by mean test score (higher is better).
* Prints the top N parameter sets with their average F1 score.

**file: 5\_spacy\_ner\_pos.py**

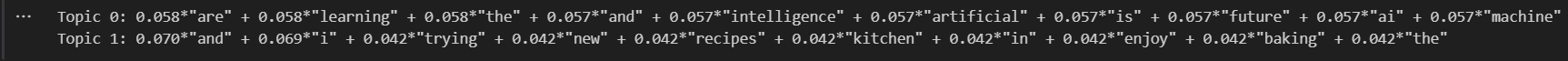
* Model: Loads the small English model en\_core\_web\_sm for tokenization, POS tagging, dependency parsing, and named entity recognition.
* NER output: Extracts named entities like *Apple* (ORG), *Bengaluru* (GPE), *Tim Cook* (PERSON), *September 3, 2025* (DATE).
* POS & Lemma: Prints each token with its part of speech (NOUN, VERB, PROPN, etc.) and lemma (base form).
* Noun chunks: Identifies base noun phrases (e.g., "Apple", "a new office", "Tim Cook", "Karnataka officials").
* Formatting: Uses string alignment and pprint for clearer, structured output.

**file: 6\_semantic\_search.py**

* Corpus: Contains 5 short ML/NLP-related documents for testing.
* Vectorization: Uses TF–IDF with unigrams and bigrams, ignoring English stopwords.
* Similarity: Computes cosine similarity between the query and all documents.
* Ranking: Results are sorted by similarity score, and the top-k (default 3) are returned.
* Example query: Searching "best models for text classification" will rank NLP-related docs (like transformers, transfer learning) higher than vision ones.

**file: 7\_summarize\_extractive.py** 

* Sentence splitting: Uses a regex-based splitter to divide text into sentences.
* Word preprocessing: Removes stopwords, keeps words longer than 2 characters, and lowercases them.
* Word scoring: Counts word frequencies and normalizes them to weigh important words more.
* Sentence scoring: Each sentence is scored by the sum of normalized word frequencies, favoring sentences with more frequent keywords.
* Summary selection: Returns the top max\_sentences sentences in their original order to create an extractive summary.

 **file: 8\_topic\_modeling.py**

* Dataset: Defines a small list of text documents (docs).
* Preprocessing: Converts each document to lowercase and tokenizes it into words (texts).
* Dictionary & Corpus: Creates a dictionary mapping words to IDs and converts documents into a bag-of-words format (corpus).
* LDA Model: Trains an LDA model with 2 topics on the corpus.
* Output: Prints the discovered topics with their top words.
* Essentially, it extracts 2 main topics from the given text data.