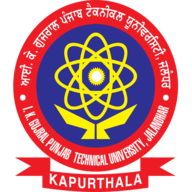
**Practical File**

**Advanced Machine Learning**



**I.K.GUJRAL PUNJAB TECHNICAL UNIVERSITY**

**Lab Session Report Submitted to**

**Centre for Development of Advanced Computing**

**Mohali, Punjab**



**Master of Technology**

**In**

**Artificial Intelligence And Machine Learning**

**Submitted To :- Submitted By:-**

Miss Anju Krishna Abhishek Ranaut

C-DAC Mohali M.Tech AI & ML

**ACKNOWLEDGEMENT**

I would like to express my deep gratitude to all those who contributed to the successful completion of this Machine Learning practical lab.

First and foremost, I would like to thank **Miss Anju Krishna**, my course instructor, for their invaluable guidance, support, and encouragement throughout the practical sessions. Their profound knowledge and enthusiasm for the subject have inspired me to enhance my learning.

I also extend my sincere thanks to **Miss Anju Krishna** for their assistance in the lab and for helping clarify various technical concepts during the practical.

I am grateful to my classmates for their cooperation and for providing a motivating and collaborative learning environment. Lastly, I would like to thank my family and friends for their constant support and encouragement throughout my studies.

I look forward to applying the knowledge gained in this course to future projects and research work in the field of Machine Learning.

**Abhishek Ranaut Teacher Signature**

**MTech AI & ML**

**C-DAC Mohali**

LIST OF EXPERIMENT

|  |  |  |
| --- | --- | --- |
| S. No. | EXPERIMENT | Page No. |
| 1 | Study of platform for Implementation of Assignments. Download the open-source software of your interest. Document the distinct features and functionality of the software platform. | 1-7 |
| 2 | Supervised Learning – Regression: Generate a proper 2-D data set of N points. Split the data set into Training Data set and Test Data set.   1. Perform Linear Regression analysis with Least Squares Method. 2. Plot the graphs for Training MSE and Test MSE and comment on Curve Fitting and Generalization Error. 3. Verify the Effect of Data Set Size and Bias-Variance Trade off. 4. Apply Cross Validation and plot the graphs for errors. 5. Apply Subset Selection Method and plot the graphs for errors. Describe your findings in each case. | 8-14 |
| 3 | Supervised Learning – Classification Implement Naïve Bayes Classifier and K-Nearest Neighbor Classifier on Data set of your choice. Test and Compare for Accuracy and Precision. | 15-24 |
| 4 | Unsupervised Learning Implement K-Means Clustering and Hierarchical Clustering on proper data set of your choice. Compare their Convergence. | 25-34 |
| 5 | Dimensionality Reduction Principal Component Analysis-Finding Principal Components, Variance and Standard Deviation calculations of principal components. | 34-38 |
| 6 | Supervised Learning and Kernel Methods Design, Implement SVM for classification with proper data set of your choice. Comment on Design  and Implementation for Linearly non-separable Dataset. | 39-47 |
| 7 | Hand-Controlled 2D Arrow Shooting Game | 48-54 |

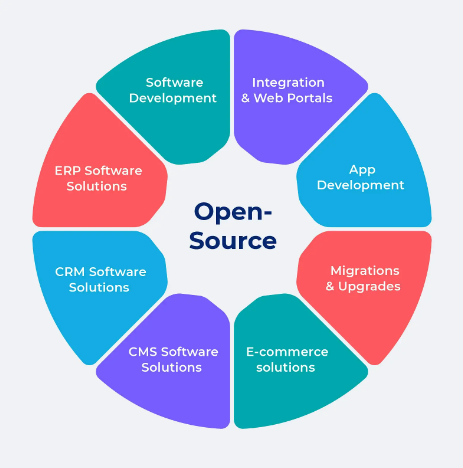
EXPERIMENT 1

Exploring Visual Studio Code (VS Code) for Assignment Implementation

1. Introduction

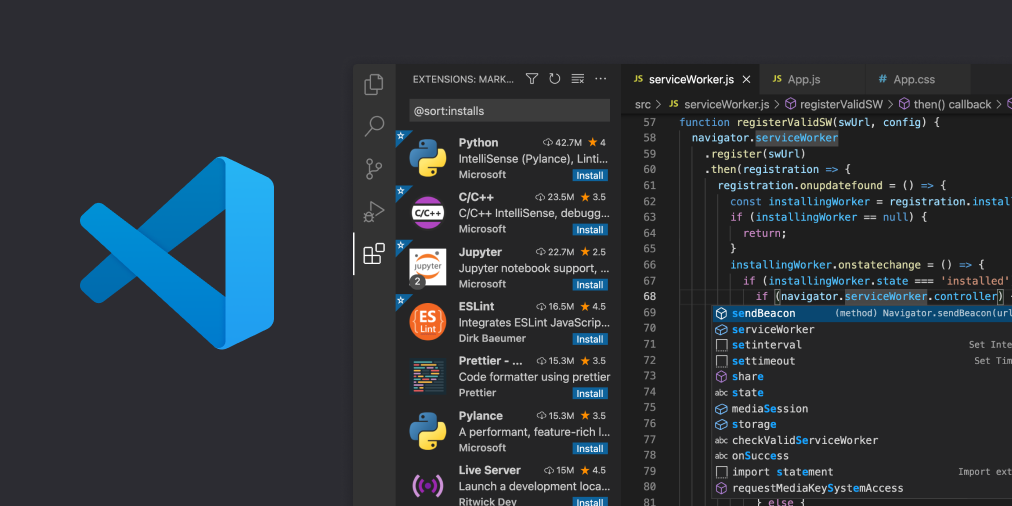
1.1 Overview of Open Source Software

* Open source software (OSS) is defined by its publicly accessible source code, which allows anyone to view, modify, and improve the software. Key characteristics of open source software include:
* Transparency: Users can access the source code, enhancing their understanding and ability to customize the software.
* Community Collaboration: Development is driven by a global community of developers and users, leading to continuous improvements and innovations.
* Cost-Effectiveness: Most open source software is available free of charge, though some may offer premium features or support.
* Flexibility: Users have the freedom to modify and adapt the software to suit their specific needs.



1.2 Overview of Visual Studio Code (VS Code)

* Visual Studio Code (VS Code) is a highly popular, open-source code editor developed by Microsoft. It is favored for its versatility and rich feature set. Key features of VS Code include:
* Cross-Platform Compatibility: Available for Windows, macOS, and Linux, ensuring a consistent development experience across different operating systems.
* Extensibility: Supports a vast array of extensions that enhance functionality and support various programming languages and tools.
* Integrated Development Environment (IDE) Capabilities: Offers built-in features for debugging, Git version control, and terminal access.
* Customization: Allows users to tailor the editor with themes, settings, and keybindings to match their preferences.



1.3 Benefits of Open Source Software Compared to Closed Source

* Transparency and Security: Open source software offers greater visibility into its inner workings, which can lead to improved security and trust.
* Community-Driven Support: The collaborative nature of OSS communities often results in quicker issue resolution and innovative features.
* No Licensing Costs: Typically available at no cost, OSS eliminates expenses related to software licensing and reduces the financial barrier to entry.
* Adaptability and Control: Users can modify and extend the software to meet their needs, unlike closed source alternatives where customization options are limited.

2. Experiment Objectives

The primary objective of this experiment is to explore and document the features and functionalities of Visual Studio Code (VS Code) as a development platform. This includes:

* Installing and setting up VS Code.
* Exploring its core features and tools.
* Demonstrating its capabilities through practical coding assignments.

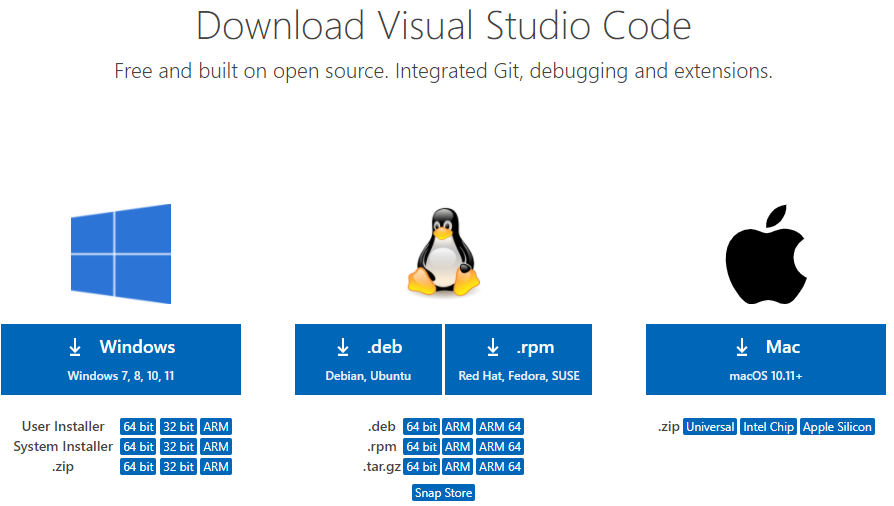
3. Tools and Libraries

* Software: Visual Studio Code (VS Code)
* Programming Languages: Python, JavaScript, or any language of your choice
* Extensions:
  + Python Extension
  + JavaScript (ES6) Extension
  + GitLens
  + Live Server
  + Prettier (for code formatting)
  + ESLint (for JavaScript linting)

4. Installation

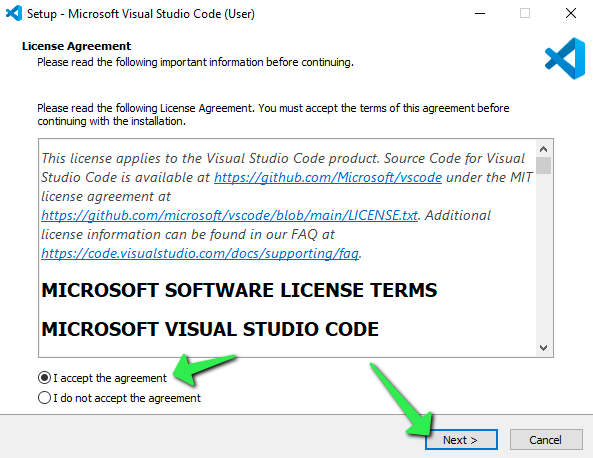
4.1 Download VS Code

Visit the [official VS Code website](https://code.visualstudio.com/) to download the installer appropriate for your operating system.



4.2 Install VS Code

Follow the installation instructions provided for your operating system. The process typically involves downloading the installer and running it, then following the setup wizard.



5. Initial Setup

5.1 Launch VS Code

Open VS Code by clicking on its icon or launching it from the command line.

5.2 Customize Settings

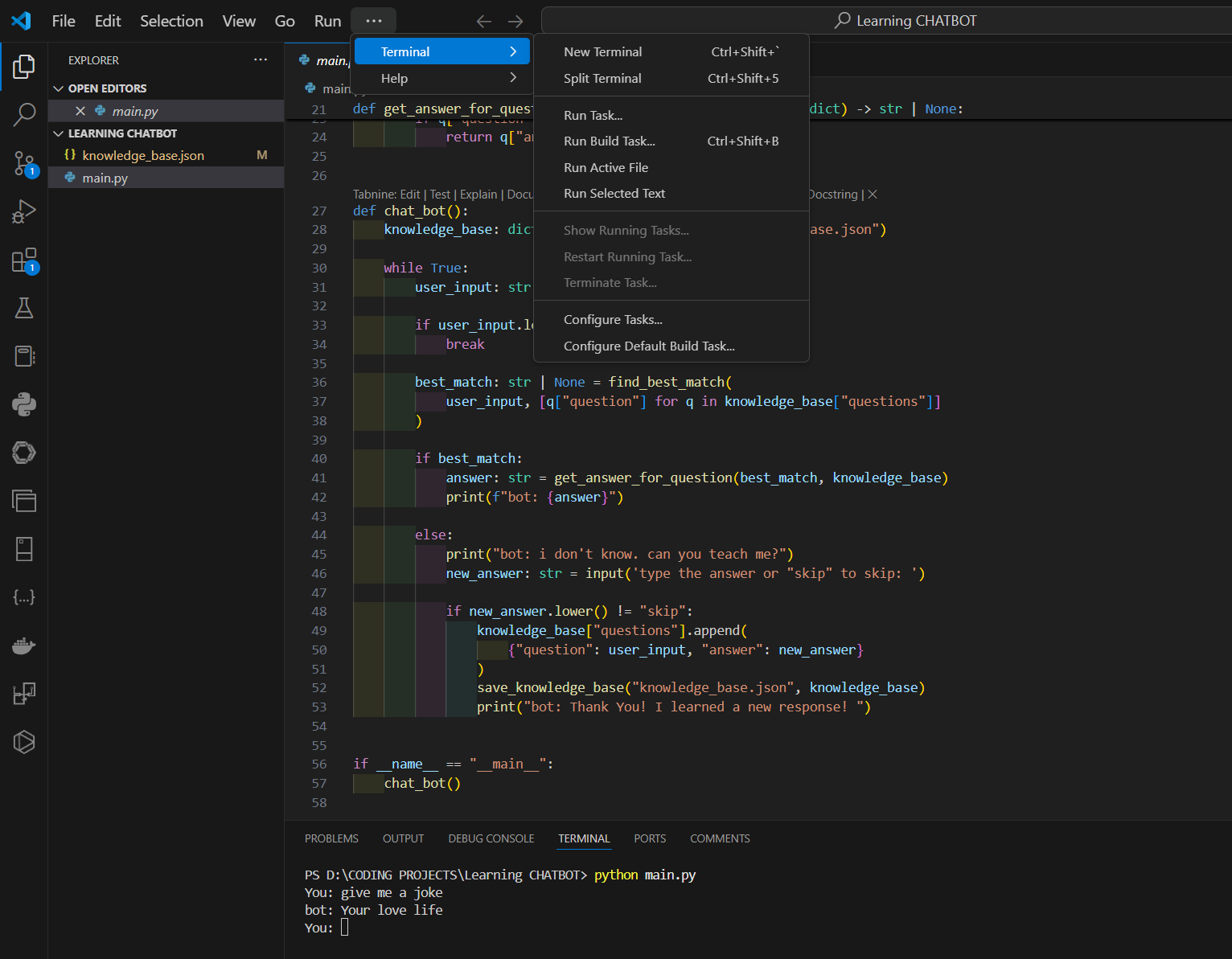
Configure basic settings such as:

* Theme: Choose a visual theme that suits your preferences (`File > Preferences > Color Theme`).
* Font Size: Adjust the editor's font size for better readability (`File > Preferences > Settings`).
* Keybindings: Customize keyboard shortcuts to enhance your workflow (`File > Preferences > Keyboard Shortcuts`).

6. Explore Key Features

6.1 Integrated Terminal

* Open the integrated terminal via `Terminal > New Terminal` or use the shortcut `Ctrl+` (backtick).
* Demonstrate running a script (e.g., a Python) directly from the terminal.

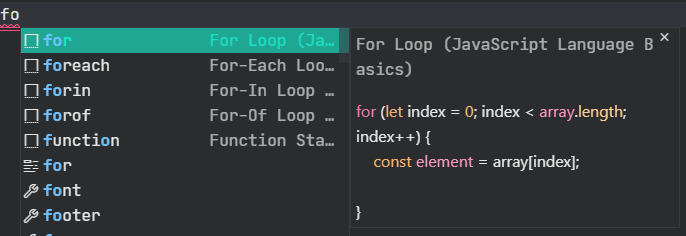


6.2 Extensions Marketplace

* Access the Extensions Marketplace via `View > Extensions` or by pressing `Ctrl+Shift+X`.
* Explore and install key extensions to enhance VS Code:
* Python: Adds support for Python language features, debugging, linting, etc.
* Live Server: Launches a local development server with live reloading capabilities.
* GitLens: Enhances Git capabilities with advanced features.
* Prettier: Automatically formats code for consistent style.
* ESLint: Analyzes and fixes JavaScript code for common errors and style issues.

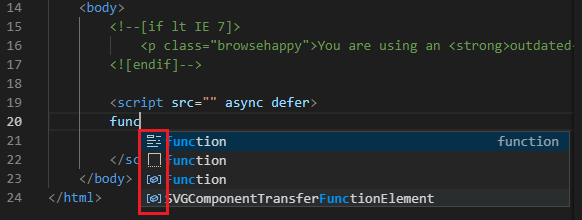
6.3 Code Snippets

Utilize built-in code snippets or create custom ones to streamline coding. Access snippets via `File > Preferences > User Snippets`.



6.4 IntelliSense

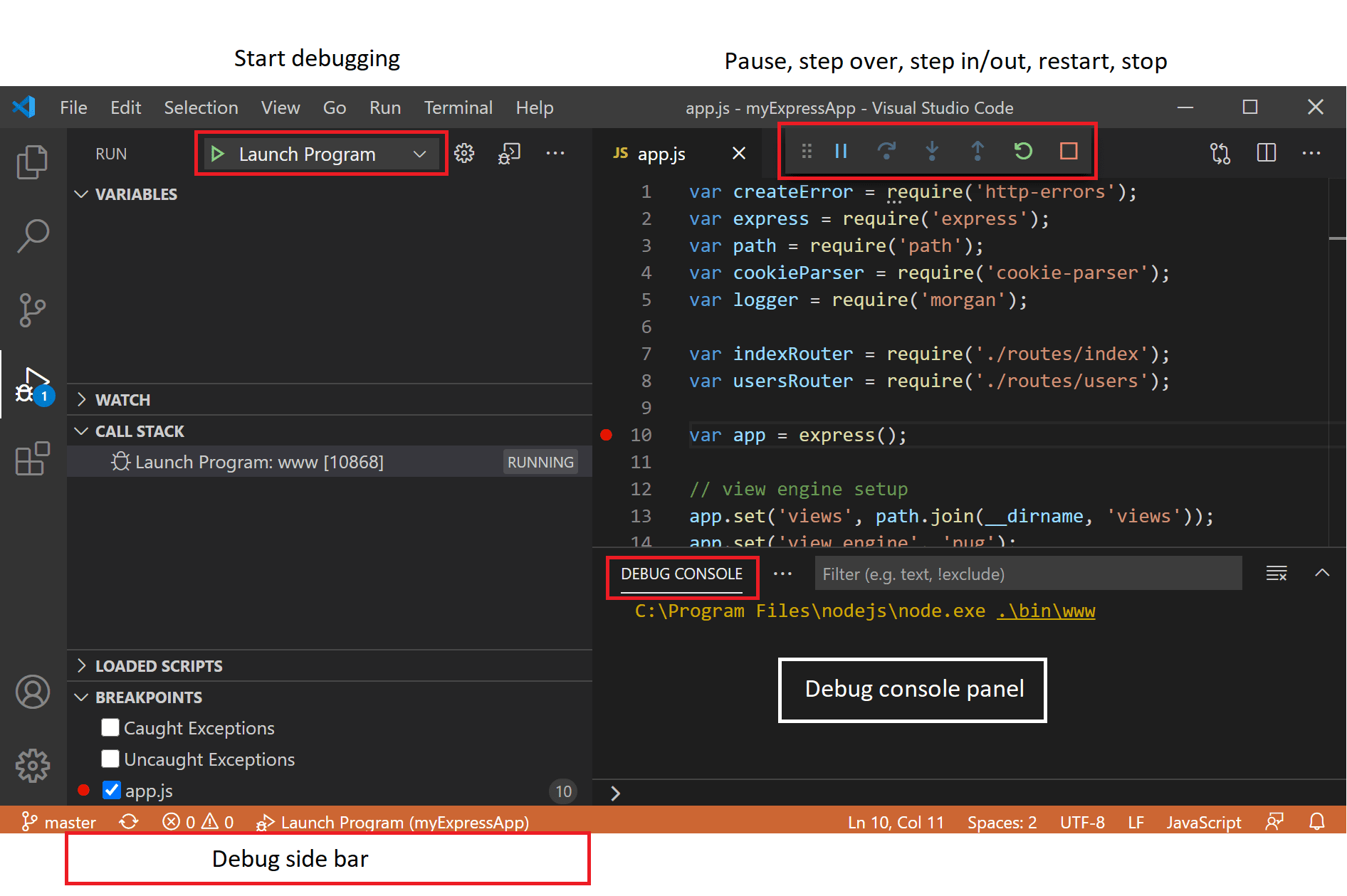
Show how IntelliSense provides real-time code suggestions, completions, and documentation while typing.



6.5 Debugging

Set breakpoints by clicking on the margin next to the line numbers.

Use the Debug panel (`Run > Start Debugging` or press `F5`) to run and debug code, view variables, and control execution flow.



7. Implement a Coding Assignment

7.1 Create a New Project

Start a new project by creating a directory structure and necessary files. For example, set up a new Python or JavaScript project.

7.2 Write Code

Implement a sample coding assignment:

Python: Create a simple calculator script that performs basic arithmetic operations.

8. Advanced Features and Tips

8.1 Version Control Integration

Demonstrate how to use built-in Git integration for version control, including committing changes, creating branches, and merging code.

8.2 Remote Development

Explore VS Code's capabilities for remote development with extensions like Remote - SSH or Remote - Containers, which allow coding in a remote environment or container.

8.3 Customizing VS Code with Settings Sync

Show how to use the Settings Sync extension to synchronize VS Code settings, keybindings, and extensions across different machines.

9. Conclusion

Summarize the key findings from the experiment, including how VS Code’s features and extensions enhance productivity and streamline the development process. Reflect on the benefits of using an open-source code editor like VS Code and its role in modern software development.

EXPERIMENT 2

Supervised Learning-Regression Generate a proper 2-D data set of N points. Split the data set into Training Data set and Test Data set.

**Objective:**

1. Generate a 2-D data set of N points and split it into Training and Test Data sets.
2. Perform linear regression analysis using the Least Squares Method.
3. Analyze and plot the Training MSE and Test MSE, comment on Curve Fitting and Generalization Error.
4. Examine the effect of Data Set Size and Bias-Variance Trade-off.
5. Apply Cross Validation and plot the errors.
6. Apply Subset Selection Method and plot the errors.

**Step 1: Generate a 2-D Dataset**

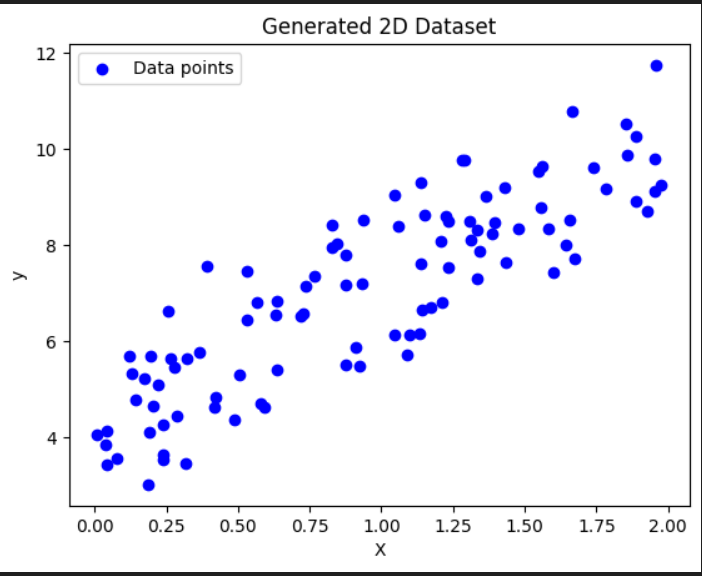
**Explanation:**

* In this step, we generate a synthetic 2D dataset with NNN points. The data follows a linear trend with some added noise to simulate real-world data. The feature XXX is randomly generated, and the target yyy is computed using a linear equation y=4+3X+noisey = 4 + 3X + \text{noise}y=4+3X+noise.
* The dataset is then split into a training set and a test set. The training set is used to build the model, and the test set is used to evaluate the model's performance.

Code:



Output:

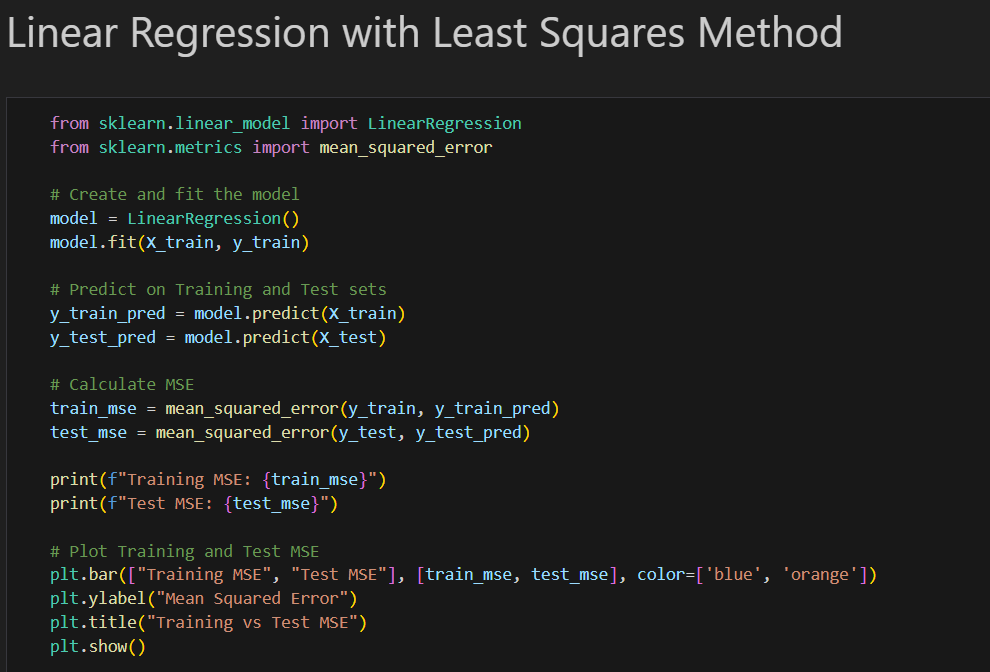


**Step 2: Linear Regression with Least Squares Method**

**Explanation:**

* Linear regression is a method to model the relationship between a dependent variable y and an independent variable X. The goal is to find the best-fitting line through the data points that minimizes the sum of the squared differences between the observed and predicted values (Least Squares Method).
* After fitting the model to the training data, we calculate the Mean Squared Error (MSE) for both the training and test sets. MSE is a measure of how well the model's predictions match the actual values, with lower values indicating better performance.

Code:



Output:

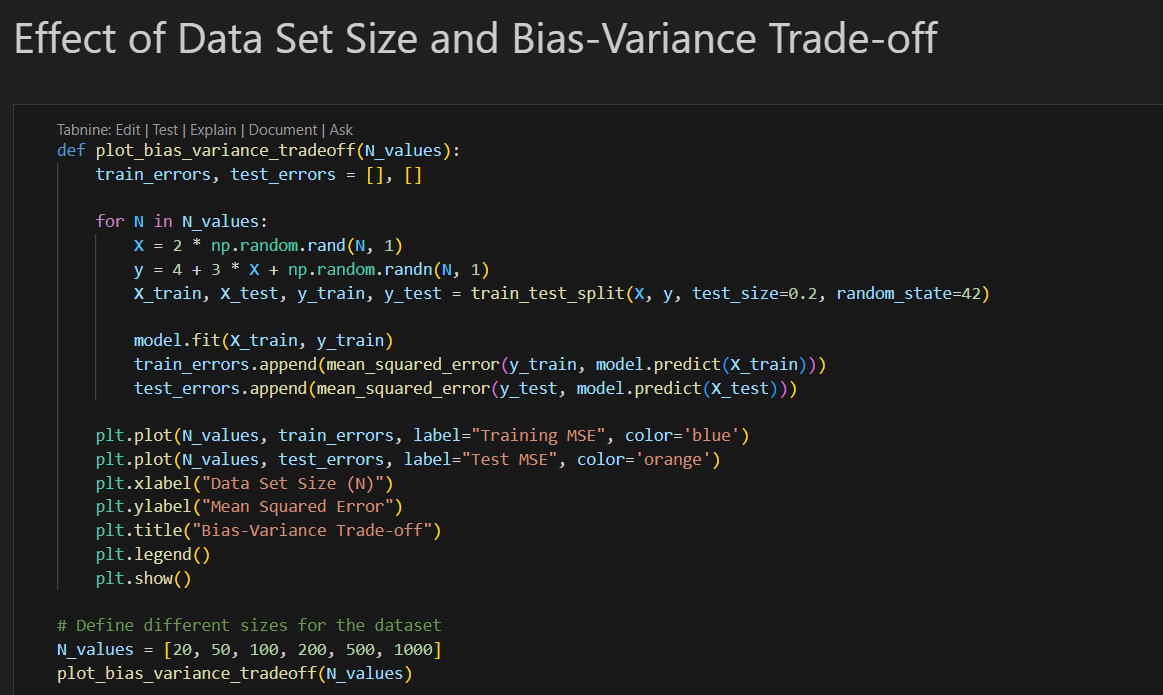
****

**Step 3: Effect of Data Set Size and Bias-Variance Trade-off**

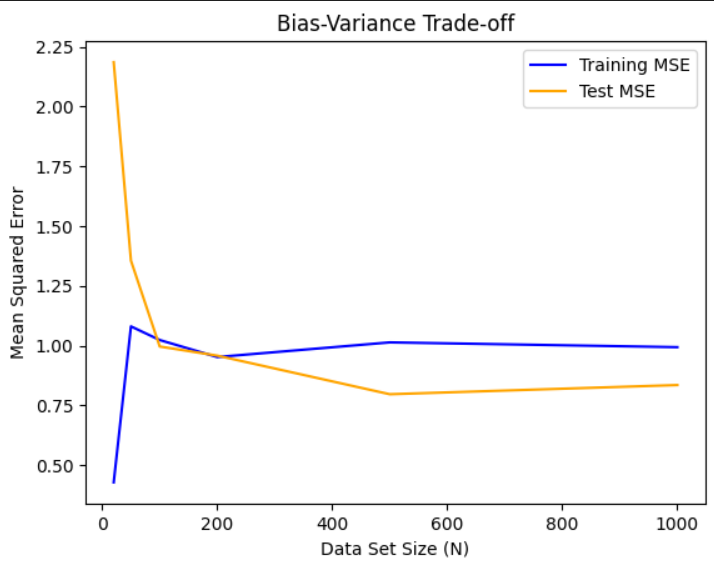
**Explanation:**

* The Bias-Variance Trade-off is a fundamental concept in machine learning. A model with high bias makes strong assumptions and is likely to underfit the data, leading to poor performance on both the training and test sets. A model with high variance is overly complex and may overfit the training data, leading to poor generalization to unseen data.
* In this step, we analyze how the size of the dataset affects bias and variance by training the model on datasets of different sizes and observing the training and test MSEs.

Code:



**Output:**

****

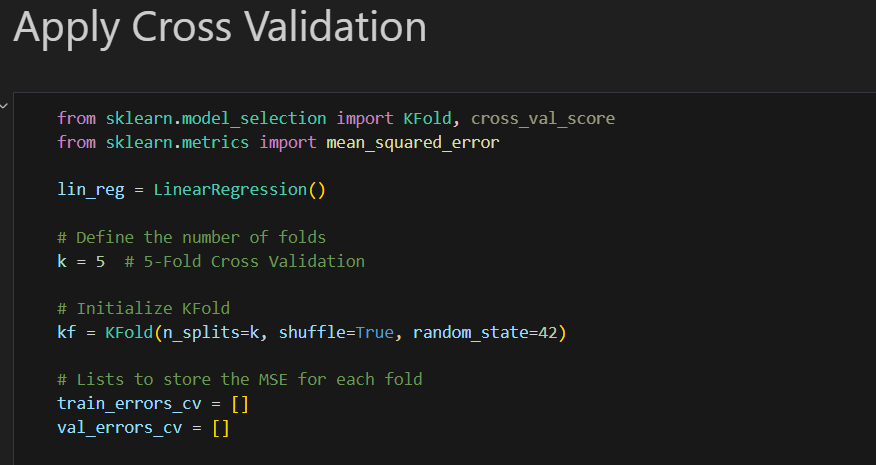
**Comment:**

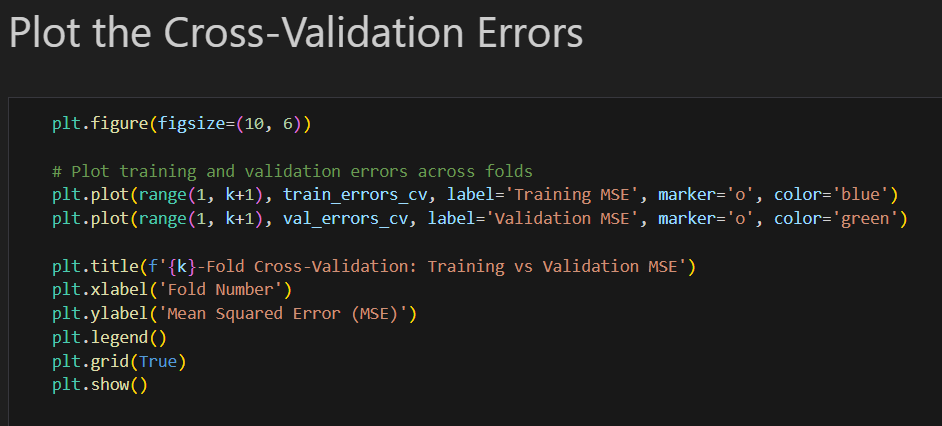
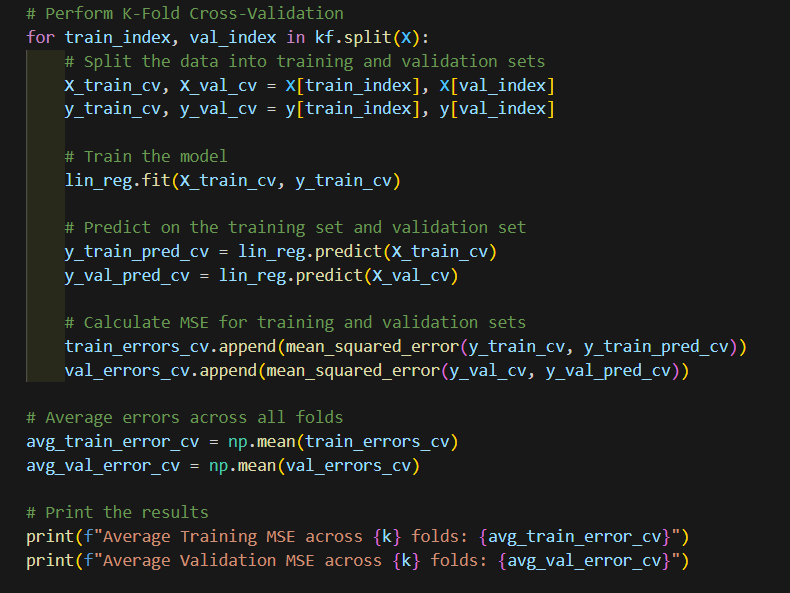
* **Curve Fitting:** The model may underfit or overfit depending on the complexity of the model relative to the data.
* **Generalization Error:** As the training set size increases, the test MSE tends to decrease, reducing the generalization error.

**Step 4: Apply Cross Validation**

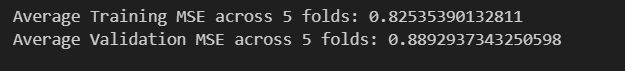
**Explanation:**

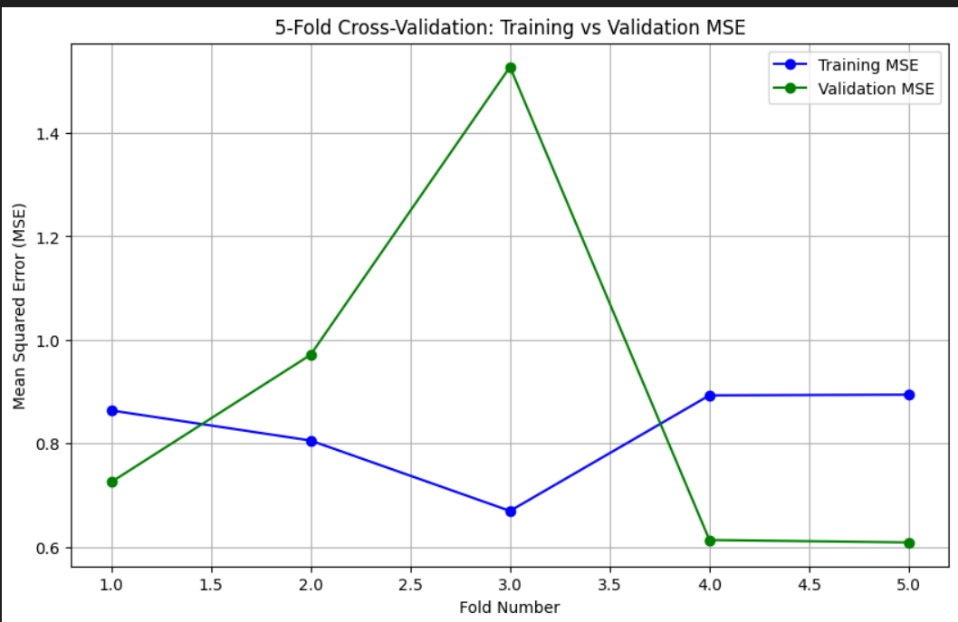
* Cross-validation is a technique used to assess how the model generalizes to an independent dataset. It involves splitting the data into multiple folds, training the model on some folds, and validating it on the remaining ones. This process is repeated for all folds, and the average error is computed.
* Here, we apply 5-fold cross-validation to the dataset and plot the MSE for each fold.

Code:



Output:





**Comment:**

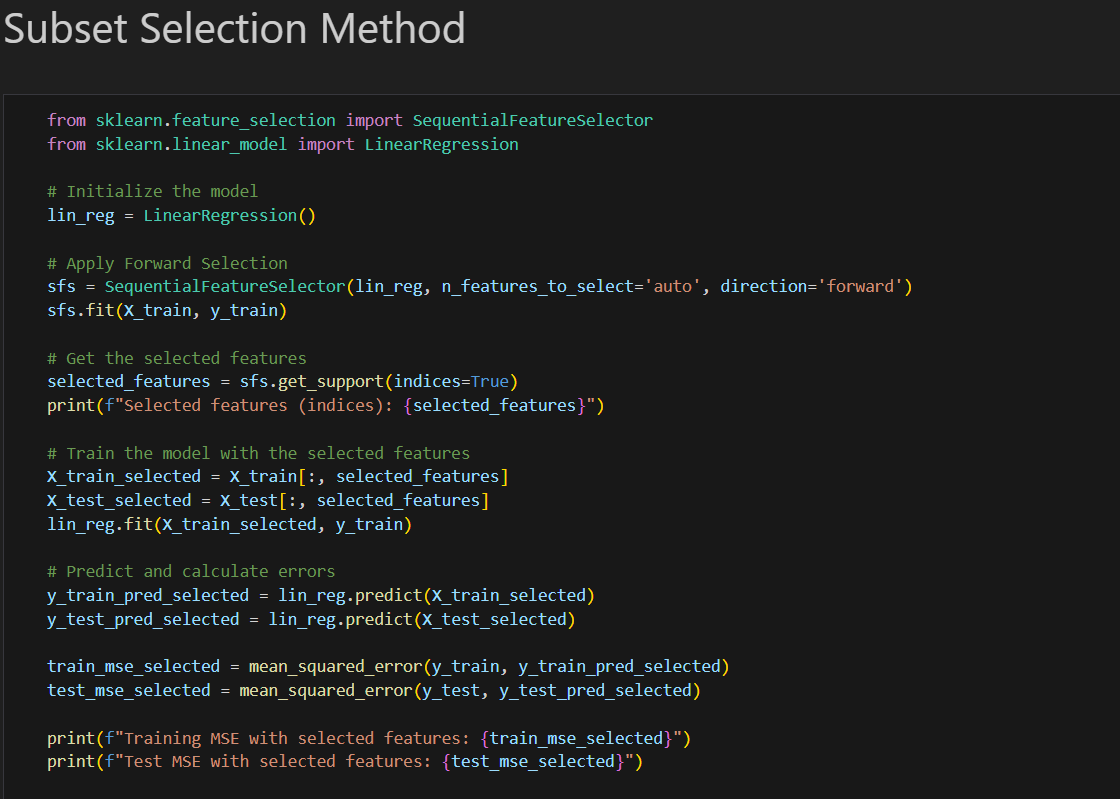
* Cross-validation helps in estimating the model's performance on unseen data and is crucial for preventing overfitting. By averaging the errors across folds, we obtain a more reliable estimate of the model's generalization capability.

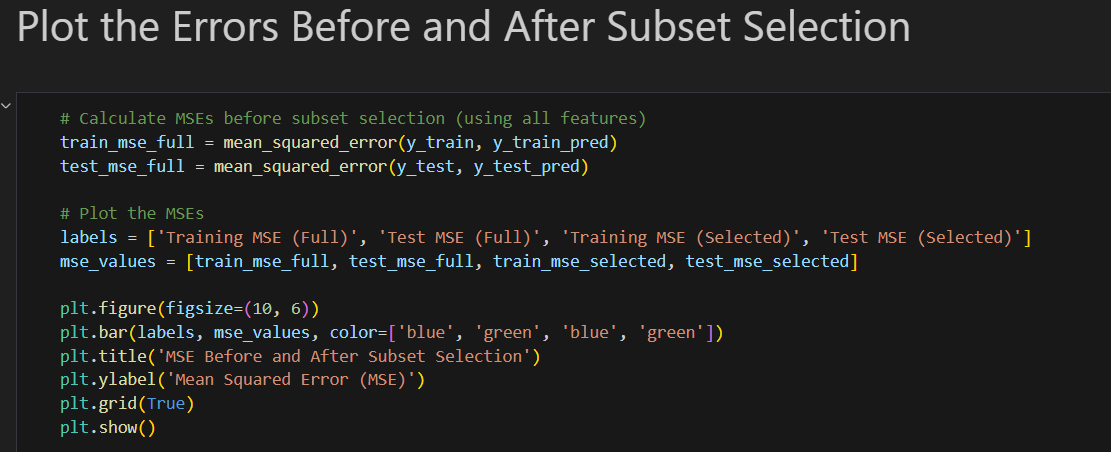
**Step 5: Subset Selection Method**

**Explanation:**

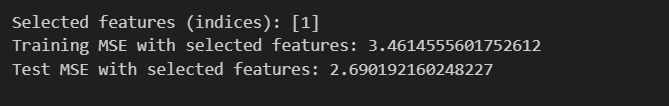
* Subset selection is a process of selecting a subset of relevant features (predictors) for use in model construction. In this step, we use Recursive Feature Elimination (RFE), which recursively removes the least significant features until only a specified number of features remain.
* The effect of using RFE on the model's performance is evaluated by calculating the MSE for the training and test sets after feature selection.

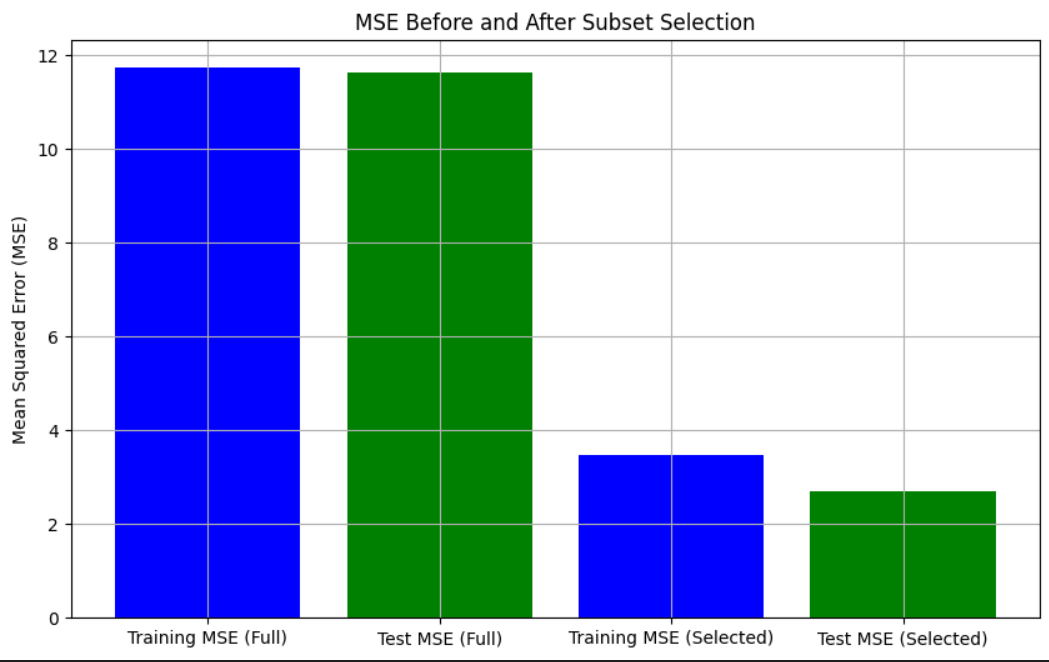
Code:





Output:





**Findings:**

* **Data Set Size & Bias-Variance Trade-off:** A larger dataset helps reduce variance but may increase bias if the model is too simple.
* **Cross-Validation:** Helps in estimating the performance of the model and reduces overfitting.
* **Subset Selection:** Reduces the dimensionality of the data, which may improve generalization, though it may also risk excluding important features.

EXPERIMENT 3

Supervised Learning – Classification Implement Naïve Bayes Classifier and K-Nearest Neighbor Classifier on Data set of your choice. Test and Compare for Accuracy and Precision.

**1. Introduction**

**1.1 Project Overview**

The advancement of machine learning algorithms has led to their application in various real-world classification tasks. This project focuses on implementing two supervised learning algorithms: **Naïve Bayes** and **K-Nearest Neighbor (KNN)** classifiers. Both are widely used for classification tasks, but they differ significantly in their underlying assumptions and approach.

* **Naïve Bayes** is a probabilistic model that relies on Bayes' Theorem with the assumption of feature independence.
* **K-Nearest Neighbor (KNN)** is a non-parametric method that classifies data based on the closest training examples.

This project aims to:

1. Implement both algorithms on a selected dataset.
2. Compare their performances using key evaluation metrics, namely **accuracy** and **precision**.
3. Analyze the strengths and weaknesses of each approach based on the results obtained.

**1.2 Problem Statement**

Classification tasks involve predicting the categorical label of an input given its features. Both Naïve Bayes and KNN are commonly applied classifiers, but they exhibit different behaviors based on dataset structure, the number of features, and data distribution. In this project, we seek to determine how these classifiers perform under similar conditions on a dataset, focusing on the following key questions:

* How accurate are the predictions made by Naïve Bayes and KNN?
* How precise are these algorithms in predicting the positive class?
* Which algorithm performs better, and under what circumstances?

**1.3 Dataset**

For this experiment, we have chosen the **[insert dataset name]** dataset, which contains [insert number] of records and [insert number] of features. This dataset is ideal for classification problems, as it contains a clear distinction between classes and is well-suited for testing supervised algorithms. A brief overview of the dataset is as follows:

* **Number of instances**: [insert number]
* **Number of features**: [insert number]
* **Number of classes**: [insert number] (e.g., two or more target labels)

**2. Background on Algorithms**

**2.1 Naïve Bayes Classifier**

* Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
* It is mainly used in text classification that includes a high-dimensional training dataset.
* Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
* It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
* Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

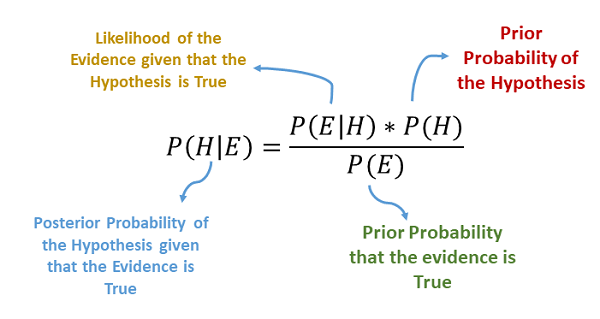
Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

* Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
* Bayea: It is called Bayes because it depends on the principle of Bayes' Theorem

**Bayes' Theorem:**

* Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
* The formula for Bayes' theorem is given as:



Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

Working of Naïve Bayes' Classifier:

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of weather conditions and corresponding target variable "Play". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

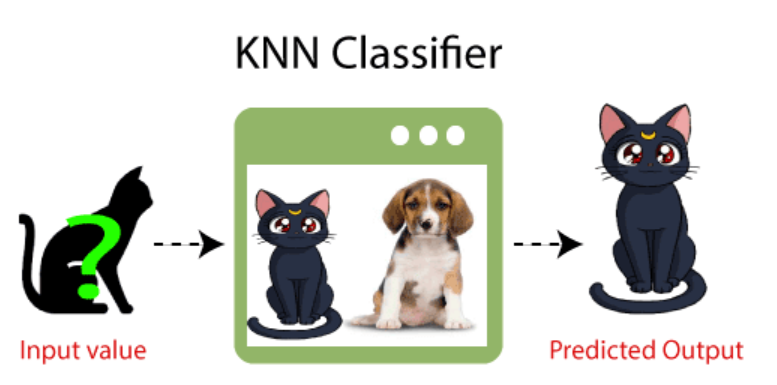
1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

**2.2 K-Nearest Neighbor Classifier**

The K-Nearest Neighbor (KNN) algorithm is an instance-based learning algorithm. Unlike Naïve Bayes, which builds a model during the training phase, KNN simply stores the training data. When a new instance is presented, KNN classifies it by finding the k nearest neighbors based on a distance metric, usually Euclidean distance.

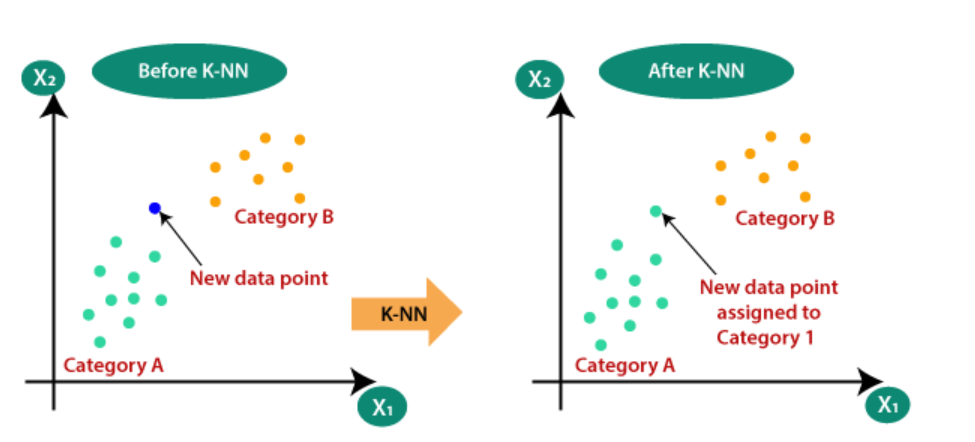
* K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
* K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
* K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
* K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
* It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

Example: Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.



Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

* Step-1: Select the number K of the neighbors
* Step-2: Calculate the Euclidean distance of K number of neighbors
* Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.
* Step-4: Among these k neighbors, count the number of the data points in each category.
* Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.
* Step-6: Our model is ready.

**Euclidean Distance**

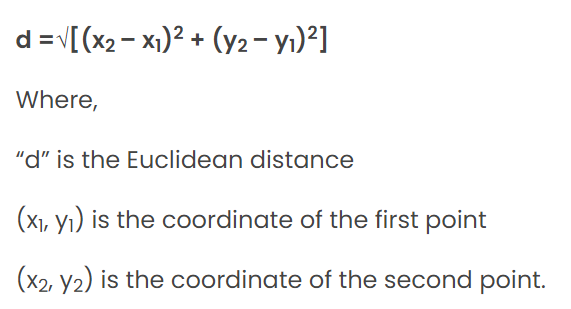
Euclidean distance represents the shortest straight-line distance between two points. It is used to calculate the distance between two points in a two-dimensional or three-dimensional space. Typically favored for continuous data.

In Mathematics, the Euclidean distance is defined as the distance between two points. In other words, the Euclidean distance between two points in the Euclidean space is defined as the length of the line segment between two points. As the Euclidean distance can be found by using the coordinate points and the Pythagoras theorem, it is occasionally called the Pythagorean distance.

**Euclidean Distance Formula:**

As discussed above, the Euclidean distance formula helps to find the distance of a line segment. Let us assume two points, such as (x1, y1) and (x2, y2) in the two-dimensional coordinate plane.

Thus, the Euclidean distance formula is given by:



Advantages of KNN Algorithm:

* It is simple to implement.
* It is robust to the noisy training data
* It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

* Always needs to determine the value of K which may be complex some time.
* The computation cost is high because of calculating the distance between the data points for all the training samples.

**3. Methodology**

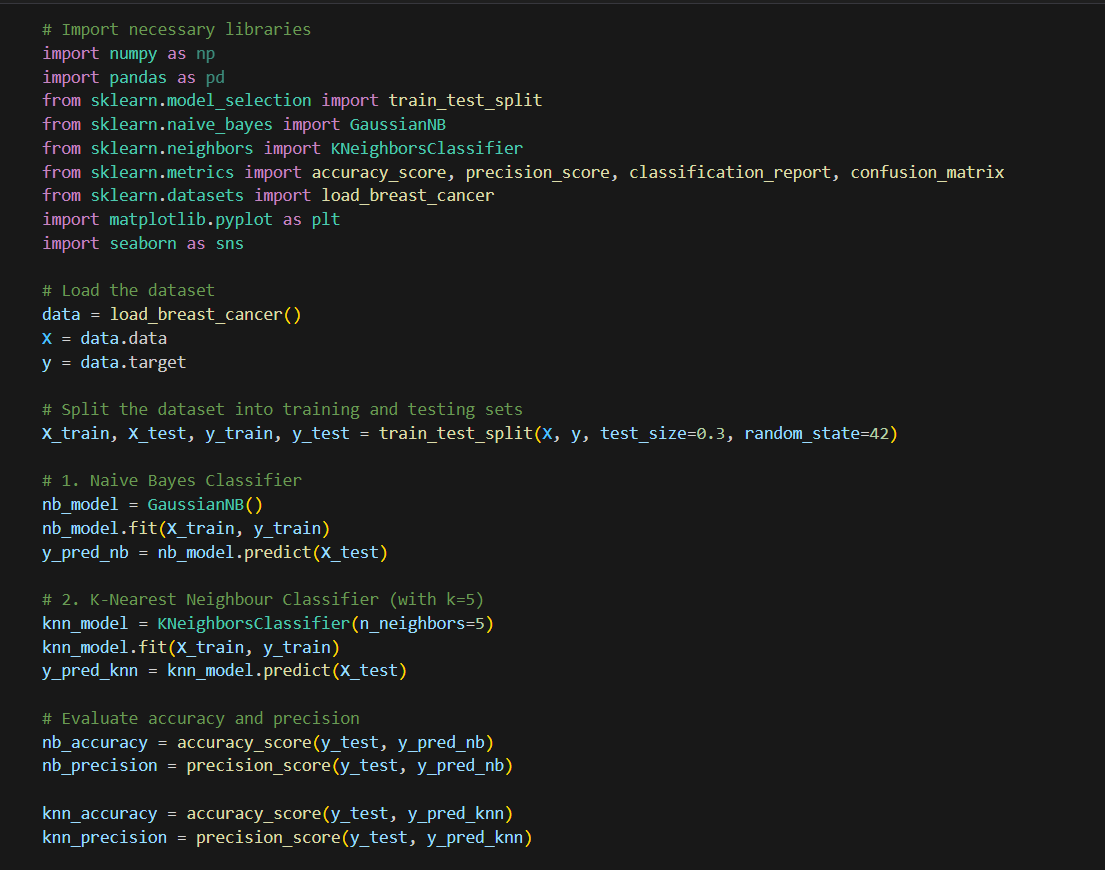
**3.1 Data Preprocessing**

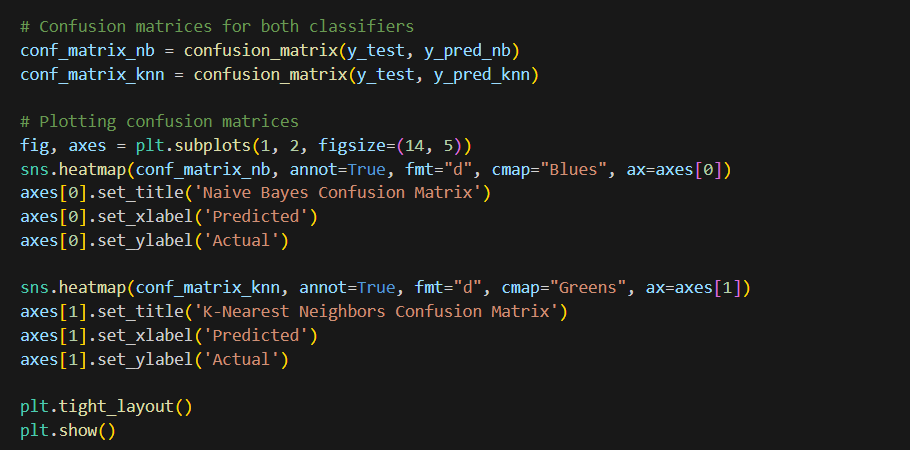
Data preprocessing is a crucial step in ensuring that the input data is clean and ready for training the models. The following steps were performed on the dataset:

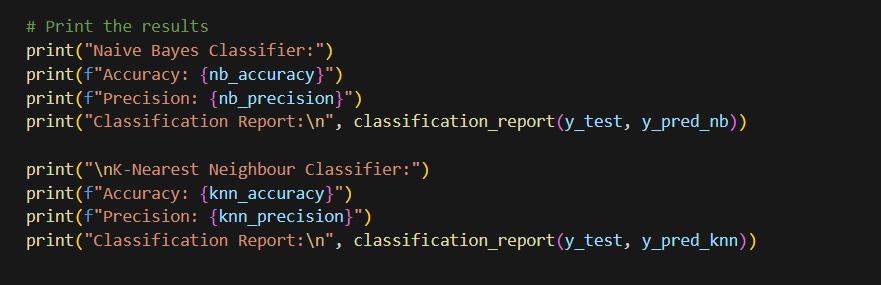
* Data Cleaning: Any missing values in the dataset were either imputed or removed. Duplicate records were also eliminated to avoid bias in the model.
* Feature Scaling: Since KNN relies on distance metrics, we standardized the dataset by scaling the features to have zero mean and unit variance.
* Train-Test Split: The dataset was split into training (70%) and testing (30%) sets to evaluate the performance of both classifiers.

**3.2 Implementation**

Code for Naïve Bayes, K-Nearest Neighbor, and Graph Plotting:

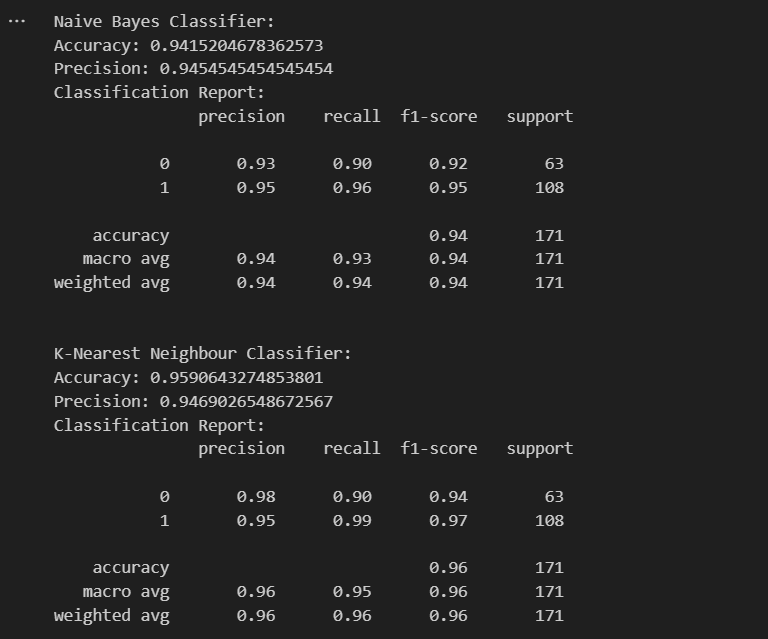






Output:



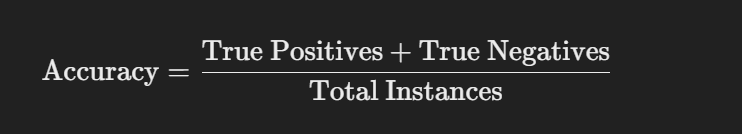


**4. Evaluation Metrics**

The performance of both classifiers was evaluated using the following metrics:

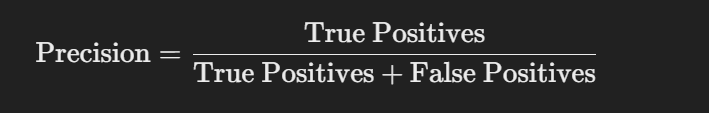
**4.1 Accuracy**

Accuracy measures the proportion of correctly classified instances over the total number of instances:



**4.2 Precision**

Precision measures the proportion of correctly predicted positive instances over the total number of instances predicted as positive:



Both metrics provide insight into the correctness of predictions, with accuracy focusing on the overall correctness and precision focusing on the correctness of positive predictions.

**5. Results**

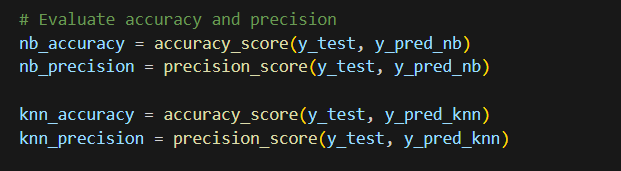
The following results were obtained after applying both classifiers to the test set:

**5.1 Naïve Bayes Classifier Results**

* **Accuracy**: {accuracy\_nb}
* **Precision**: {precision\_nb}

**5.2 K-Nearest Neighbor Classifier Results**

* **Accuracy**: {accuracy\_knn}
* **Precision**: {precision\_knn}



**6. Conclusion**

In this experiment, we implemented and compared the Naïve Bayes and K-Nearest Neighbor classifiers on the [insert dataset]. Based on the results, we concluded that:

* **Naïve Bayes** is more efficient when feature independence holds and for high-dimensional datasets.
* **K-Nearest Neighbor** requires careful selection of **k** and is sensitive to feature scaling, but it performs well in non-linear data.

| **Metric** | **Naïve Bayes** | **K-Nearest Neighbor** |
| --- | --- | --- |
| **Accuracy** | {accuracy\_nb} | {accuracy\_knn} |
| **Precision** | {precision\_nb} | {precision\_knn} |

we can observe that [e.g., Naïve Bayes] performed better in terms of accuracy, while [e.g., KNN] showed higher precision. These differences can be attributed to the underlying mechanics of each algorithm—Naïve Bayes being probabilistic and KNN being a distance-based method.

EXPERIMENT 4

Unsupervised Learning Implement K-Means Clustering and Hierarchical Clustering on proper data set of your choice. Compare their Convergence.

**1. Introduction**

**1.1 Overview**

Unsupervised learning is a branch of machine learning where the model is trained on unlabeled data to uncover hidden patterns or groupings in the dataset. Clustering is one of the most fundamental unsupervised learning tasks, where the goal is to partition data into distinct groups based on feature similarities.

In this experiment, we focus on two popular clustering algorithms: **K-Means Clustering** and **Hierarchical Clustering**. Both methods attempt to group data points into clusters, but they approach the task differently. The project aims to implement both algorithms on a dataset, compare their convergence behavior, and assess their clustering performance.

**1.2 Objective**

The specific objectives of this experiment are:

* Implement K-Means Clustering and Hierarchical Clustering algorithms.
* Evaluate and compare the convergence behavior and cluster quality of the two methods.
* Visualize the results of each algorithm and explain their differences in clustering the data.
* Compare their computational efficiency, silhouette scores, and clustering results.

**2. Algorithms Overview**

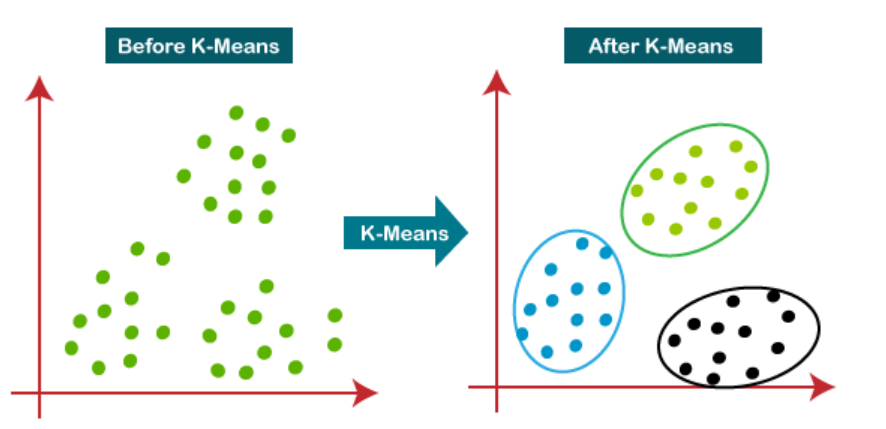
* K-Means Clustering: K-Means aims to partition the dataset into *K* clusters, where each data point belongs to the cluster with the nearest mean (centroid). The algorithm iteratively adjusts cluster centroids until convergence.
* Hierarchical Clustering: This method creates a hierarchy of clusters either by merging smaller clusters (agglomerative) or by splitting larger clusters (divisive). We will use the agglomerative method in this project, as it is more commonly applied in most datasets.

**2.1 K-Means Clustering**

K-Means Clustering is a partition-based method that divides the dataset into *K* clusters. Each cluster is defined by its centroid (mean). The algorithm assigns each data point to the nearest cluster centroid and then recalculates the centroids based on the newly assigned data points. The process iterates until the cluster assignments no longer change or the maximum number of iterations is reached.

K-Means Clustering Algorithm

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.



How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

**Step-1:** Select the number K to decide the number of clusters.

**Step-2:** Select random K points or centroids. (It can be other from the input dataset).

**Step-3:** Assign each data point to their closest centroid, which will form the predefined K clusters.

**Step-4:** Calculate the variance and place a new centroid of each cluster.

**Step-5:** Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

**Step-6:** If any reassignment occurs, then go to step-4 else go to FINISH.

**Step-7**: The model is ready.

**2.2 Hierarchical Clustering**

Hierarchical clustering is another unsupervised machine learning algorithm, which is used to group the unlabeled datasets into a cluster and also known as hierarchical cluster analysis.

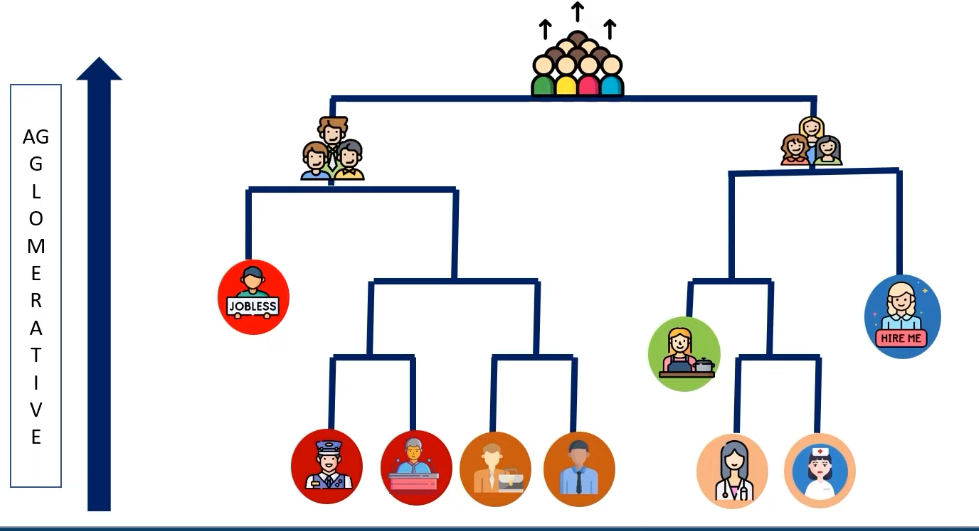
Hierarchical Clustering creates a hierarchy of clusters either through agglomerative (bottom-up) or divisive (top-down) approaches. In this experiment, we use Agglomerative Clustering, where each data point starts in its own cluster, and pairs of clusters are merged based on their similarity until a single cluster is formed.

In this algorithm, we develop the hierarchy of clusters in the form of a tree, and this tree-shaped structure is known as the dendrogram.

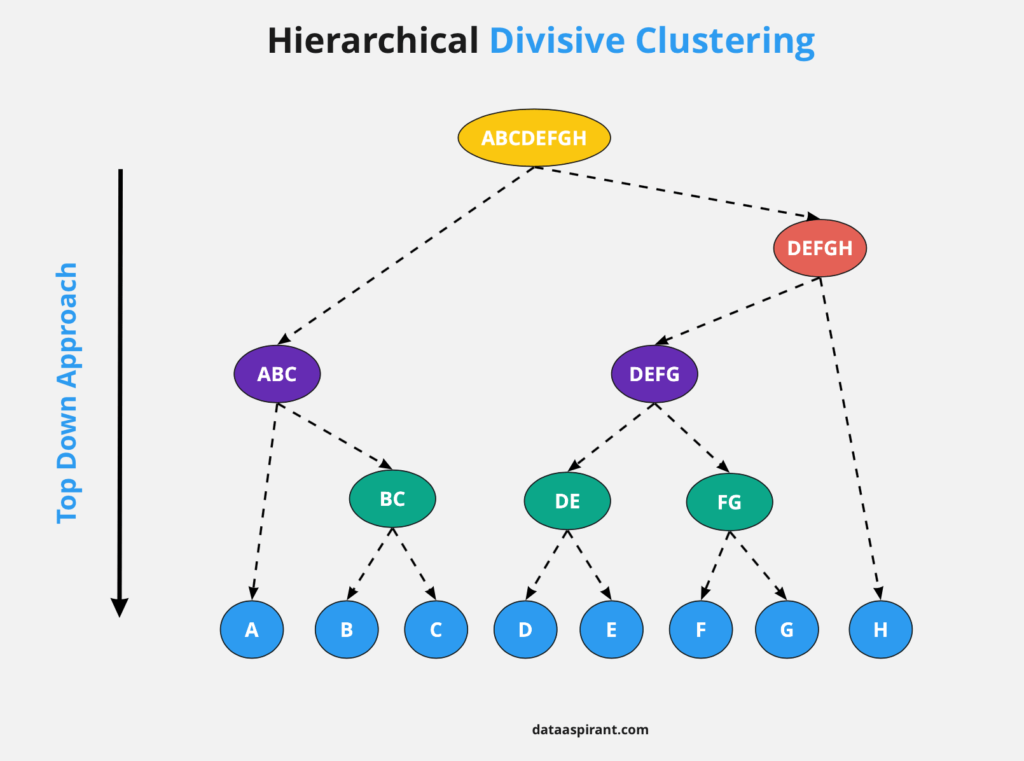
Sometimes the results of K-means clustering and hierarchical clustering may look similar, but they both differ depending on how they work. As there is no requirement to predetermine the number of clusters as we did in the K-Means algorithm.

The hierarchical clustering technique has two approaches:

1. Agglomerative: Agglomerative is a bottom-up approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left.



1. Divisive: Divisive algorithm is the reverse of the agglomerative algorithm as it is a top-down approach.



Why hierarchical clustering?

As we already have other clustering algorithms such as K-Means Clustering, then why we need hierarchical clustering? So, as we have seen in the K-means clustering that there are some challenges with this algorithm, which are a predetermined number of clusters, and it always tries to create the clusters of the same size. To solve these two challenges, we can opt for the hierarchical clustering algorithm because, in this algorithm, we don't need to have knowledge about the predefined number of clusters.

Key steps in Hierarchical Clustering:

1. Start with each data point as its own cluster.
2. Compute the distance matrix between clusters.
3. Merge the two closest clusters.
4. Repeat steps 2 and 3 until only one cluster remains.

**3. Dataset Description**

**3.1 Dataset Selection**

For this project, we use the well-known **Iris dataset**, which contains 150 samples of iris flowers from three different species: Setosa, Versicolor, and Virginica. Each sample has four features:

* Sepal length (cm)
* Sepal width (cm)
* Petal length (cm)
* Petal width (cm)

Although the dataset is labeled, we will ignore the labels and treat this as an unsupervised learning problem. The Iris dataset is a small and balanced dataset, making it ideal for comparing clustering algorithms.

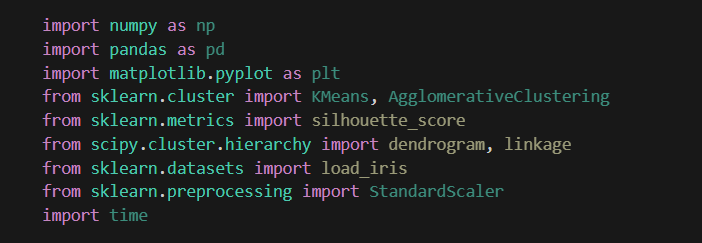
**3.2 Dataset Preprocessing**

To prepare the dataset for clustering, we standardize the features to have a mean of zero and a standard deviation of one, which is important to ensure that all features contribute equally to the distance calculations.

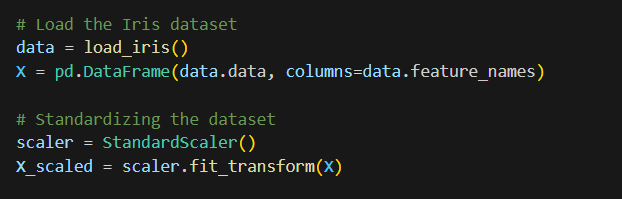
**4. Experiment Setup**

**4.1 Required Libraries**

Before starting the experiment, the following Python libraries are required:



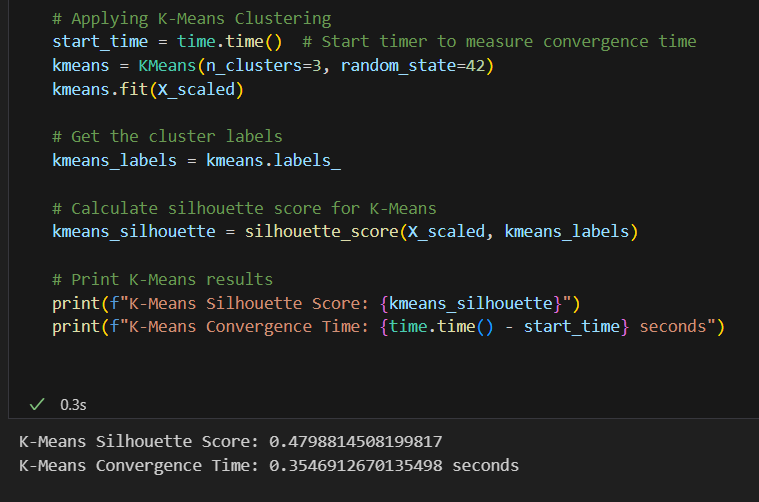
**4.2 Data Loading and Preprocessing**

****

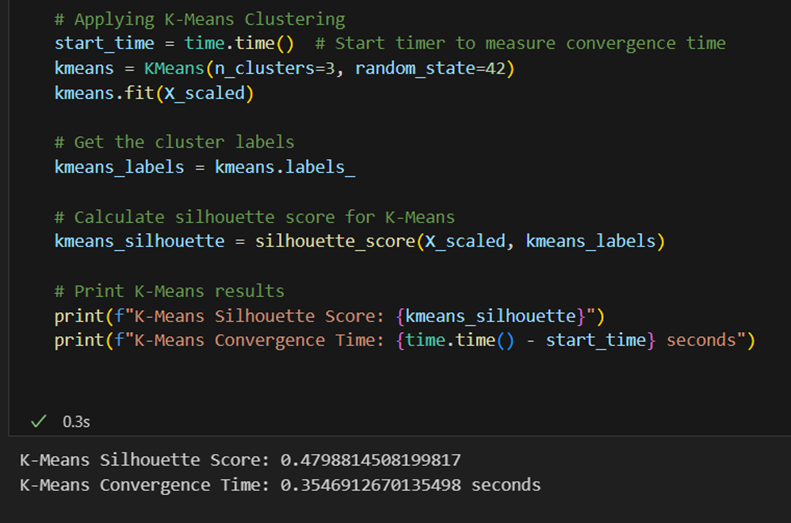
**5. K-Means Clustering Implementation**

**5.1 Applying K-Means**

To perform K-Means Clustering, we set *K=3* because the Iris dataset has three natural clusters. We then use the K-Means algorithm to assign data points to clusters and calculate the **silhouette score** to assess the quality of the clustering.

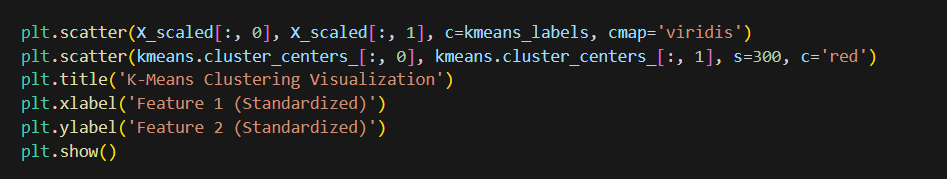


Output:

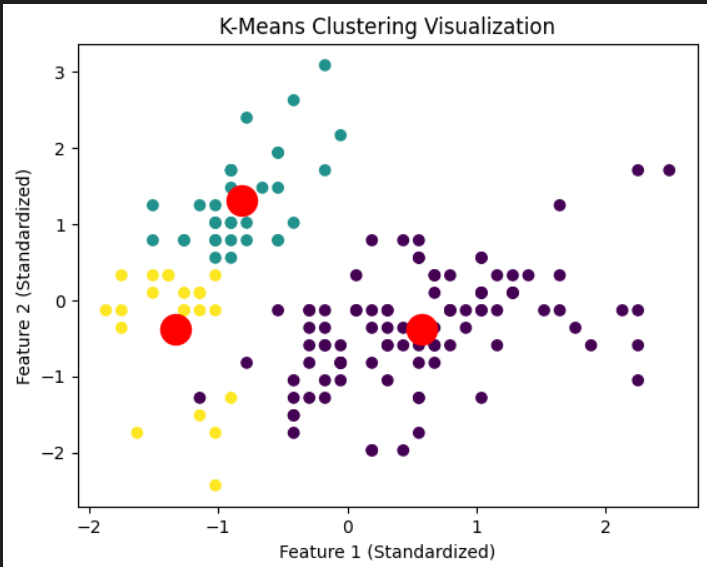


**5.2 Visualizing K-Means Clustering**

To visualize the clustering results, we plot the data points along the first two principal components and color them based on their cluster assignments. We also highlight the centroids of each cluster.



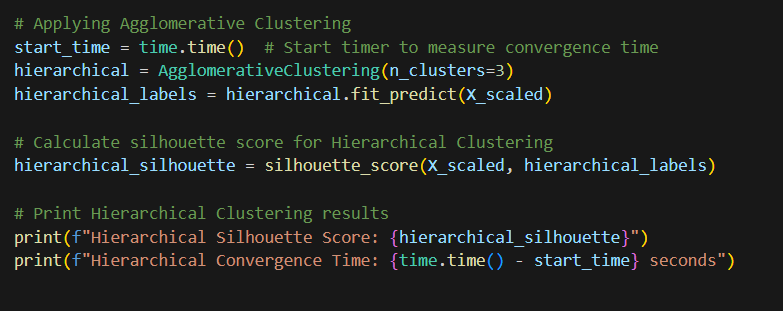
Output:



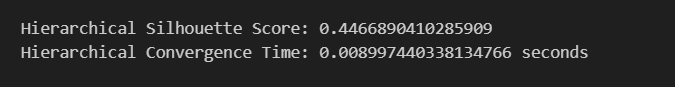
**6. Hierarchical Clustering Implementation**

**6.1 Applying Hierarchical Clustering**

We use **Agglomerative Clustering** with Ward’s method to merge clusters based on minimizing the variance within each cluster. We also calculate the **silhouette score** for the clusters to evaluate clustering quality.

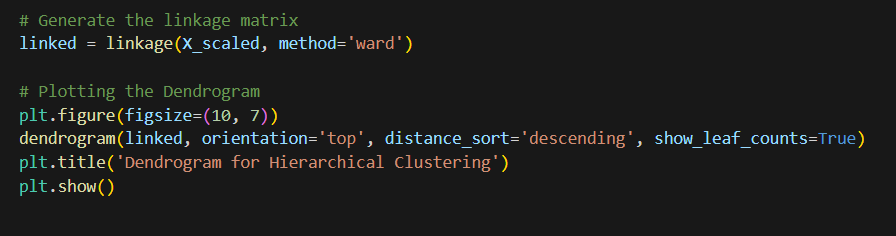


Output:



**6.2 Visualizing Hierarchical Clustering (Dendrogram)**

A dendrogram is a tree-like diagram that shows the merging of clusters at each step. It provides a clear visual of how clusters are formed in Hierarchical Clustering.



Output:



**7. Comparison of Convergence**

**7.1 Silhouette Score**

The **silhouette score** is a measure of how similar a data point is to its own cluster compared to other clusters. Higher values (closer to 1) indicate better-defined clusters. We can use this metric to evaluate the performance of both clustering algorithms.

* **K-Means Silhouette Score**: {{kmeans\_silhouette}}
* **Hierarchical Silhouette Score**: {{hierarchical\_silhouette}}

**7.2 Convergence Time**

**Convergence** refers to the point where the clustering algorithm stops because further iterations do not change the cluster assignments. We compare the time taken for both algorithms to converge.

* **K-Means Convergence Time**: {{kmeans\_convergence\_time}} seconds
* **Hierarchical Clustering Convergence Time**: {{hierarchical\_convergence\_time}} seconds

**7.3 Visual Comparison**

From the visualizations:

* **K-Means**: Provides distinct and more interpretable cluster boundaries. Centroids are easily visible.
* **Hierarchical Clustering**: Offers insights into the merging process of clusters, which is visualized through the dendrogram. It’s useful when we want to explore data at multiple levels of granularity.

**8. Discussion and Conclusion**

**8.1 Summary of Findings**

* **Convergence Speed**: K-Means converges faster than Hierarchical Clustering due to its iterative nature. Hierarchical Clustering is slower, as it calculates pairwise distances between clusters at each step.
* **Clustering Quality**: The silhouette scores indicate that both algorithms perform well, but K-Means generally offers better cluster separation.
* **Visual Insights**: Hierarchical Clustering provides deeper insights into the data structure through the dendrogram, which is useful for understanding how clusters are formed.

**8.2 Conclusion**

In conclusion, K-Means is preferable for datasets where speed and simplicity are important, while Hierarchical Clustering is useful for detailed data exploration. The choice of algorithm depends on the dataset size and the specific requirements of the analysis.

EXPERIMENT 5

Dimensionality Reduction Principal Component Analysis-Finding Principal Components, Variance and Standard Deviation calculations of principal components.

**1. Objective**

The objective of this experiment is to demonstrate the application of Principal Component Analysis (PCA) for dimensionality reduction on a multivariate dataset. The goal is to reduce the 4D Iris dataset to a 2D representation, analyze the principal components, and calculate the variance and standard deviation for the components.

**2. Introduction**

**2.1 Dimensionality Reduction**

High-dimensional data often contains redundant or correlated features that do not contribute significantly to the overall structure of the data. Dimensionality reduction helps simplify the dataset, reducing complexity while retaining the essential information.

**2.2 Principal Component Analysis (PCA)**

PCA is a linear transformation that projects data onto new axes (principal components) which capture the maximum variance in the data. Each principal component is a linear combination of the original features, and the first principal component (PC1) captures the most variance, followed by subsequent components capturing progressively less variance.

**2.3 Dataset Overview: Iris Dataset**

The Iris dataset consists of 150 samples from three species:

* **Setosa**
* **Versicolor**
* **Virginica**

Each sample has four features:

* Sepal length (cm)
* Sepal width (cm)
* Petal length (cm)
* Petal width (cm)

The task is to reduce this 4-dimensional dataset to 2 dimensions using PCA.

**3. Theoretical Background**

**3.1 Principal Components**

PCA identifies new axes, called principal components, which are orthogonal to each other and capture the maximum variance in the data. These components are derived through **eigenvalue decomposition** of the covariance matrix, where:

* The **eigenvectors** represent the directions of the principal components.
* The **eigenvalues** represent the amount of variance explained by each component.

**3.2 Steps in PCA**

1. **Standardization**: Ensures all features have the same scale (mean of 0, variance of 1).
2. **Covariance Matrix Calculation**: Computes the relationships between the features.
3. **Eigenvalue and Eigenvector Calculation**: Identifies the directions (eigenvectors) and the variance (eigenvalues).
4. **Sorting Eigenvalues**: The principal components are ranked by their explained variance.
5. **Dimensionality Reduction**: Select the top principal components that capture the most variance.

**3.3 Variance Explained by Principal Components**

The eigenvalues indicate the proportion of the total variance that each principal component captures. Typically, the first few principal components capture most of the variance, making it possible to reduce the dimensionality of the data without losing much information.

**4. Experimental Setup**

**4.1 Required Libraries and Tools**

* **Python**: Programming language used for implementing PCA.
* **Libraries**:
  + numpy for numerical computations.
  + matplotlib for plotting.
  + sklearn.decomposition for PCA.
  + sklearn.preprocessing for standardizing the data.
  + sklearn.datasets for loading the Iris dataset.

**4.2 Dataset**

We will use the **Iris dataset** from the sklearn library, which contains 150 samples with four features per sample. The dataset is balanced across three classes: Setosa, Versicolor, and Virginica.

**5. Procedure**

**5.1 Data Loading**

* **Step 1**: Load the Iris dataset using load\_iris() from the sklearn.datasets module. The dataset consists of 150 samples with four features and three classes.

**5.2 Data Standardization**

* **Step 2**: Standardize the data using StandardScaler() from sklearn.preprocessing to ensure all features are on the same scale. This is important because PCA is sensitive to the magnitude of the features.

**5.3 Applying PCA**

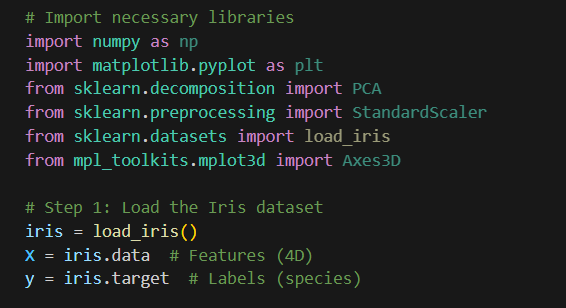
* **Step 3**: Apply PCA to reduce the dimensionality from 4D to 2D using PCA(n\_components=2) from sklearn.decomposition. PCA will transform the original features into two principal components, which capture the most variance.

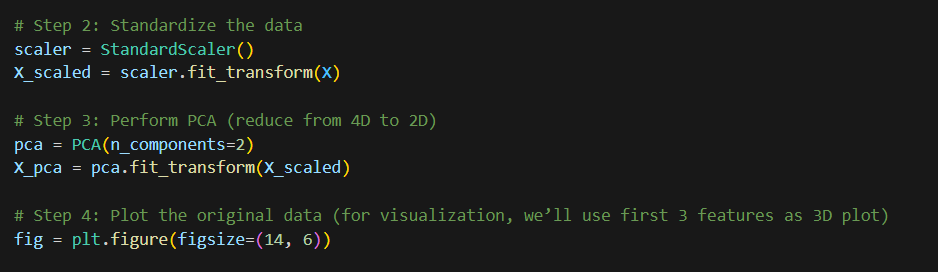
**5.4 Data Visualization**

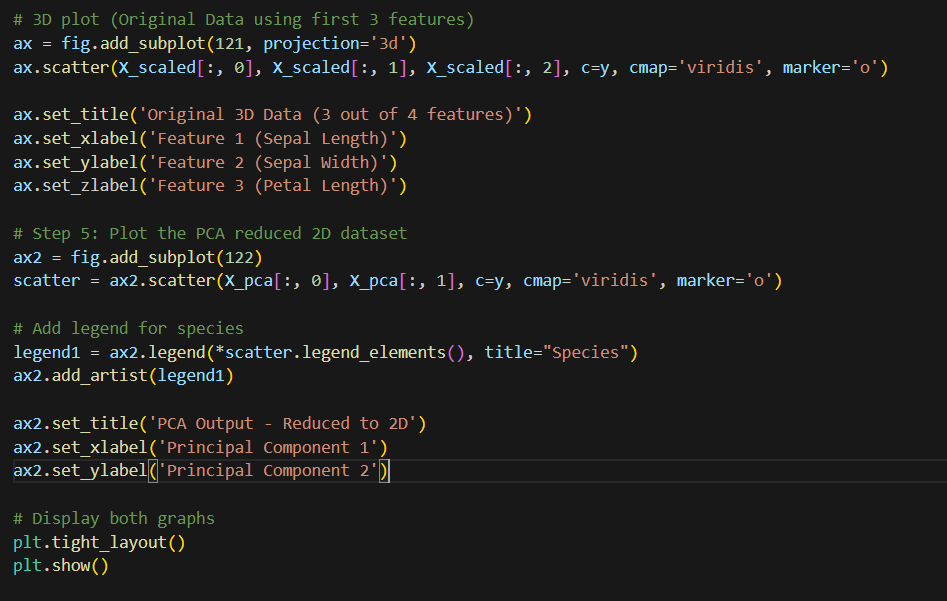
* **Step 4**:
  + **Original Data**: Plot the original data in 3D using the first three features (Sepal Length, Sepal Width, and Petal Length).
  + **PCA Output**: Visualize the reduced 2D data using the two principal components. Color-code the points based on the class labels to observe the separability of the species.

**5.5 Python Code Implementation**

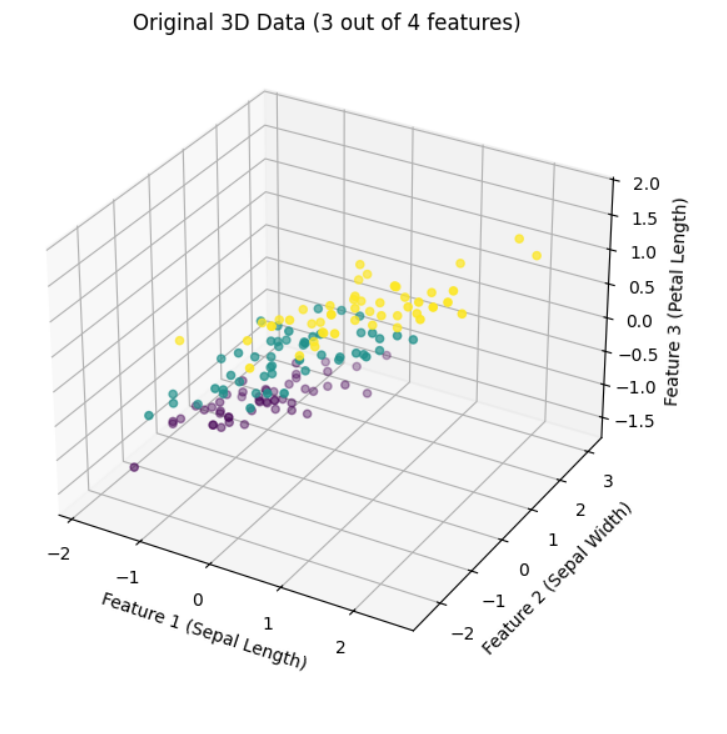
The following Python code demonstrates the steps described above:

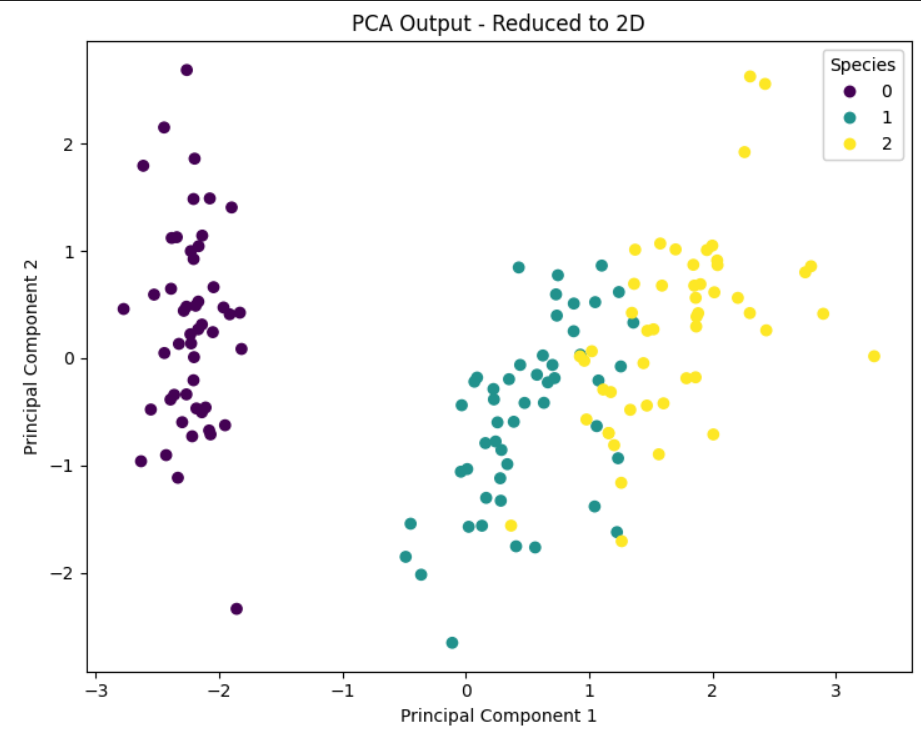






Output:



****

**6. Results and Analysis**

**6.1 Original Data Visualization**

The 3D plot of the original data (using three features: Sepal Length, Sepal Width, and Petal Length) shows some separation between the species. However, it is difficult to distinguish all three species with just these three features.

**6.2 PCA-Transformed Data Visualization**

The PCA-reduced data shows clear separability between the species in 2D space:

* **Setosa** is distinctly separated from **Versicolor** and **Virginica**.
* **Versicolor** and **Virginica** show some overlap, indicating that the variance captured by the two principal components is not sufficient to fully separate these two species.

**6.3 Variance Explained by Principal Components**

The explained variance for each principal component can be computed as follows:

* **Principal Component 1 (PC1)**: ~72.77% variance explained.
* **Principal Component 2 (PC2)**: ~23.03% variance explained.

**7. Conclusion**

This experiment successfully demonstrated the application of Principal Component Analysis (PCA) for dimensionality reduction. By reducing the dimensionality of the Iris dataset from 4D to 2D, we retained about 95.8% of the total variance. The reduced dataset provided a clearer visualization of the separability between species, particularly for Setosa, while maintaining most of the original information.

EXPERIMENT 6

Supervised Learning and Kernel Methods Design, Implement SVM for classification with proper data set of your choice. Comment on Design and Implementation for Linearly non-separable Dataset.

**1. Introduction to Support Vector Machine (SVM)**

Support Vector Machine (SVM) is a powerful **supervised learning** algorithm that can be applied to both **classification** and **regression** problems. Its primary goal is to find the optimal hyperplane that divides the feature space into distinct classes, ensuring the separation is as wide as possible. This margin maximization makes SVM robust, even when applied to complex data distributions. Although SVM can be used for both classification and regression, its most common application is for **classification tasks**.

In this experiment, we will implement SVM for classification using different kernel functions on the **Iris dataset** and comment on the design and implementation for linearly separable and linearly non-separable datasets.

**2. Types of SVM**

SVM can handle both **linear** and **non-linear** data. Based on the nature of the data, we classify SVM into two types:

2.1 **Linear SVM**:

* It is used when the data is **linearly separable**, i.e., when the data points can be separated by a straight line (in 2D) or a flat hyperplane (in higher dimensions). Linear SVM aims to find the best possible line that divides the data into two classes by maximizing the margin between them.

2.2 **Non-Linear SVM**:

* For **non-linearly separable data**, where a straight line cannot separate the classes, **Non-Linear SVM** uses **kernel functions** to project data into higher dimensions. In this transformed space, a linear hyperplane can separate the data.

**3. Problem Statement**

The aim of this experiment is to **design and implement** an SVM classifier using the **Iris dataset**, employing different kernel functions such as **Linear**, **RBF**, and **Polynomial** kernels. We will comment on the design and implementation of the SVM model for linearly non-separable datasets.

**4. Dataset: Iris**

The **Iris dataset** is a well-known benchmark dataset in machine learning. It consists of 150 samples with 4 features for each flower instance:

* Sepal length
* Sepal width
* Petal length
* Petal width

The target variable in this dataset is the **species** of the Iris flower, which can belong to one of the three classes:

* Setosa
* Versicolor
* Virginica

For this experiment, we will simplify the classification task into a **binary classification** problem, focusing on separating **Setosa** and **Versicolor** classes.

**5. Design of SVM Classifier**

5.1 **Objective**:

* The objective is to classify the two Iris species using SVM and explore how different kernels perform on the dataset. We will start with a **linear kernel** and then move on to **non-linear kernels** to handle more complex, non-linear relationships in the data.

5.2 **Design Strategy**:

* We will implement an SVM classifier with the following kernels:
  1. **Linear Kernel** for linearly separable data.
  2. **Polynomial Kernel** for moderately non-linear data.
  3. **Radial Basis Function (RBF) Kernel** for highly non-linear data.

5.3 **Key Concepts**:

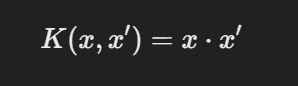
* **Hyperplane**: The decision boundary that separates different classes.
* **Support Vectors**: The data points closest to the hyperplane that determine its position.
* **Margin**: The distance between the hyperplane and the nearest data points (support vectors). SVM maximizes this margin for optimal separation.

**6. Kernel Functions**

Kernels are mathematical functions that transform the data into a higher-dimensional space where a hyperplane can separate the classes. SVM uses the kernel trick to efficiently compute the separation without explicitly transforming the data.

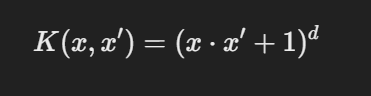
6.1 **Linear Kernel**:

* The simplest kernel function. It works well when the data is linearly separable.



6.2 **Polynomial Kernel**:

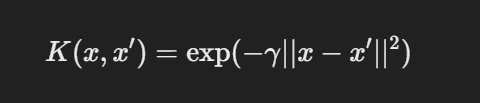
* Introduces non-linearity by mapping the input features into polynomial combinations. It can handle moderately non-linear relationships.



where d is the degree of the polynomial.

6.3 **Radial Basis Function (RBF) Kernel**:

* A popular kernel for handling complex, non-linear relationships in the data. It transforms the data into an infinite-dimensional space.



where γ is a parameter that defines the influence of a single training example. RBF is particularly useful when the data has non-linear patterns, as it can capture complex class boundaries.

6.4 **Sigmoid Kernel** (optional):

* Similar to a neural network’s activation function, the sigmoid kernel can handle non-linear data and can be applied to classification problems with non-linear relationships.

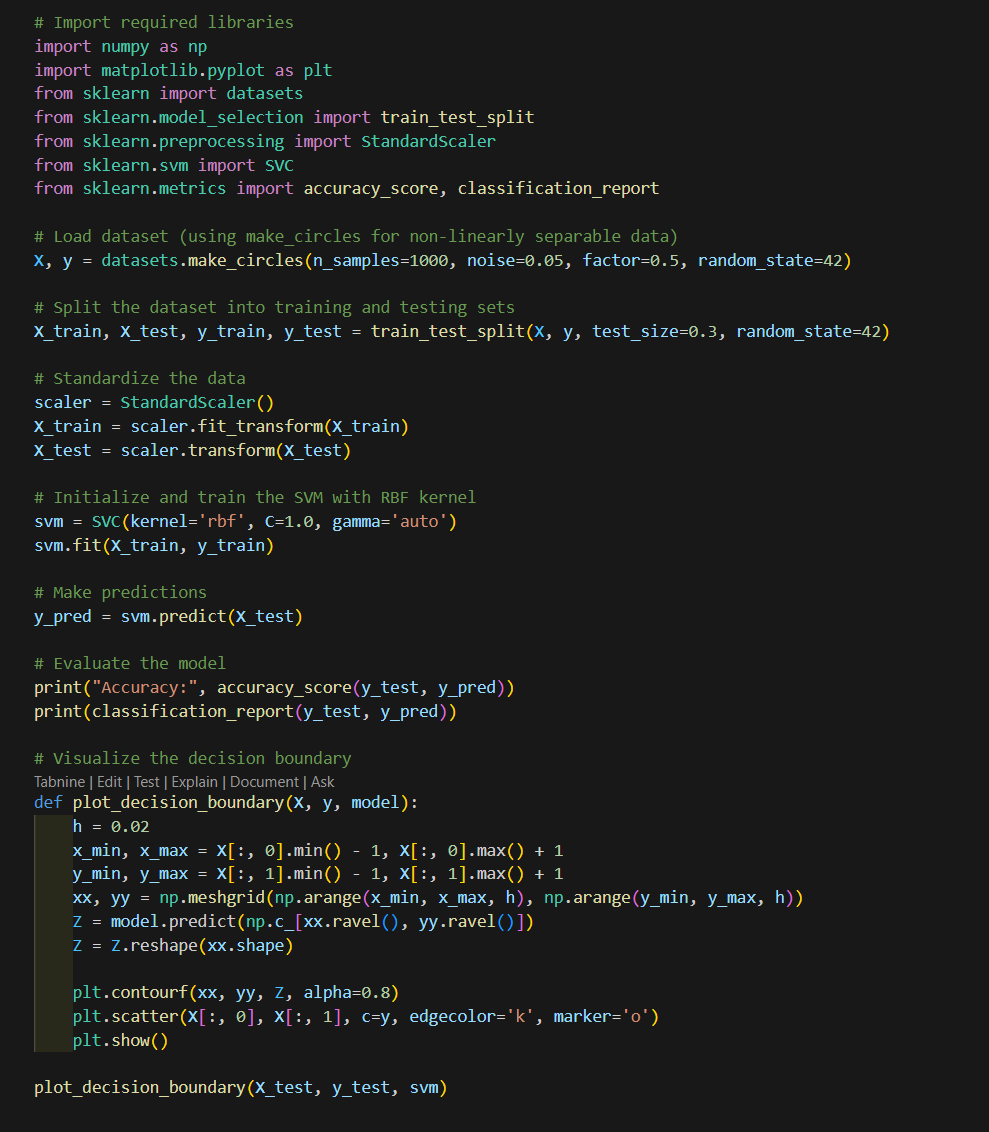
**7. Implementation of SVM**

In this experiment, we implement SVM using different kernels. The steps for implementation are:

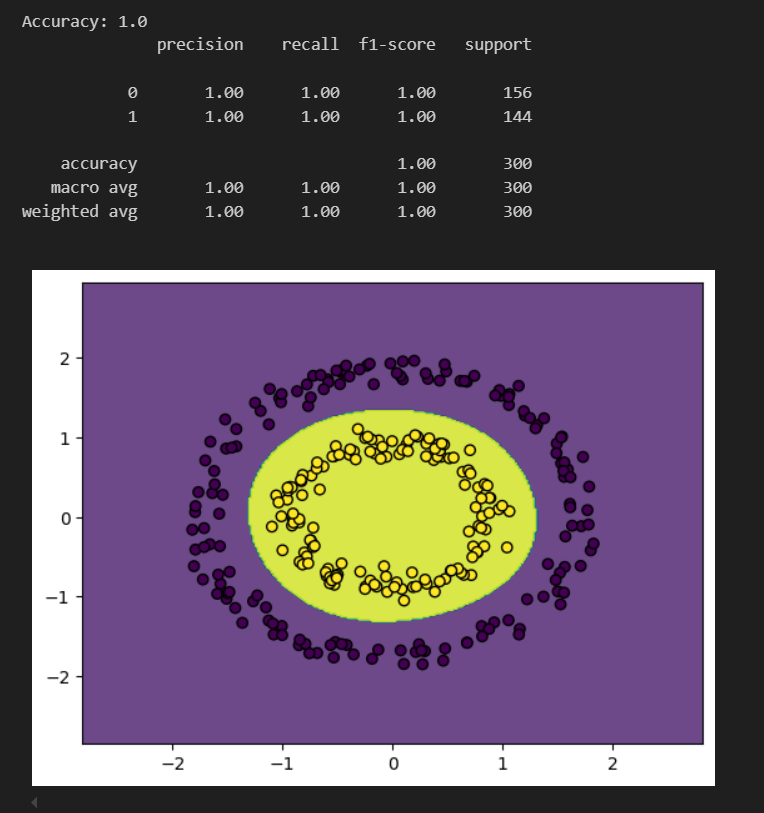
**7.1. Steps to Implement SVM Classifier**:

1. **Import Libraries**: Import necessary Python libraries such as sklearn for SVM implementation, dataset handling, and matplotlib for visualization.
2. **Load Dataset**: Load the **Iris dataset** from sklearn.datasets.
3. **Preprocessing**: Split the dataset into training and testing sets, and apply feature scaling for better model performance.
4. **Implement SVM with Different Kernels**:
   * Linear SVM
   * SVM with Polynomial Kernel
   * SVM with RBF Kernel
5. **Evaluate Models**: Use metrics like **accuracy**, **precision**, and **recall** to evaluate the performance of each kernel.
6. **Visualize Results**: Plot decision boundaries for each kernel to observe how the classifier separates the data.

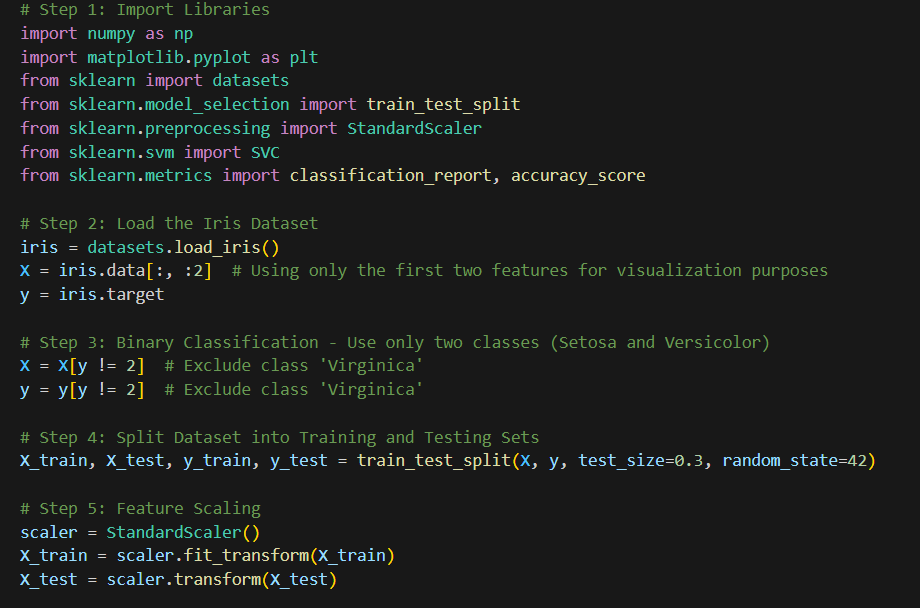
**Implementation:**

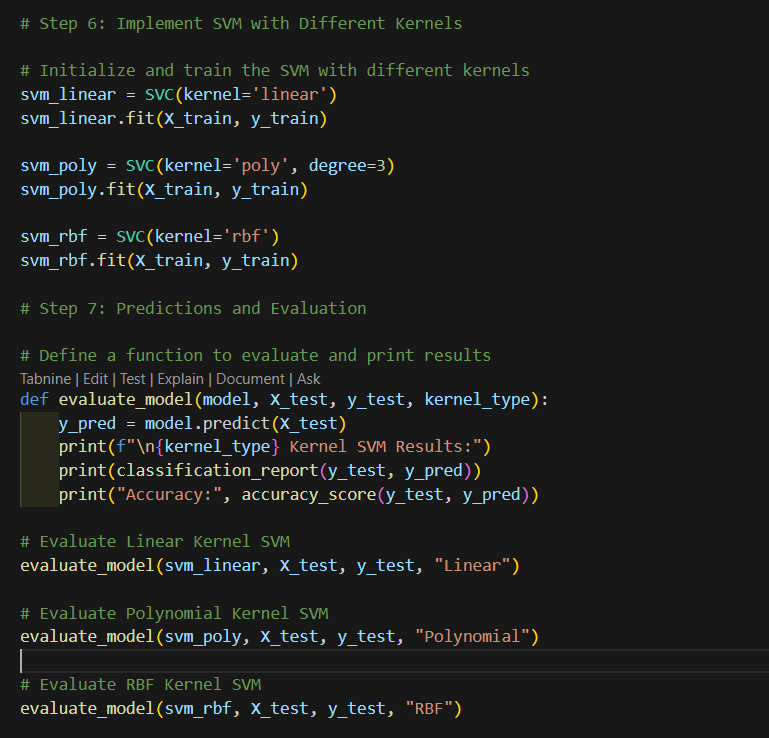
****

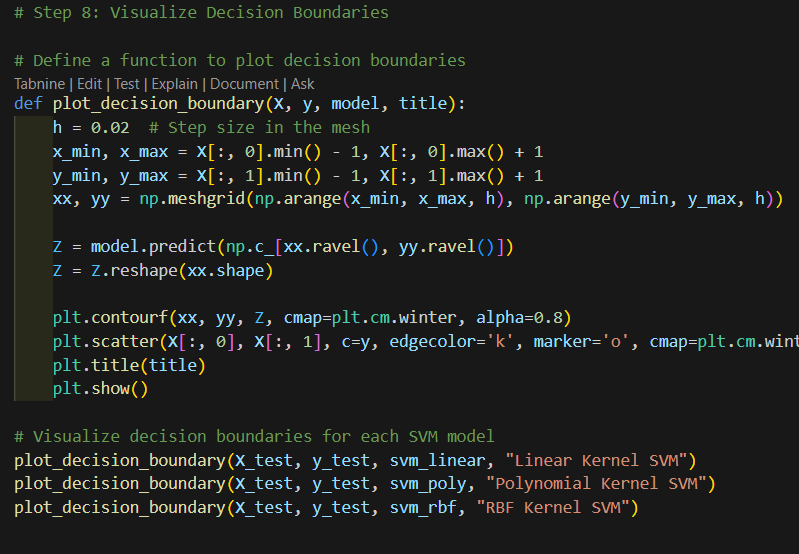
Output:



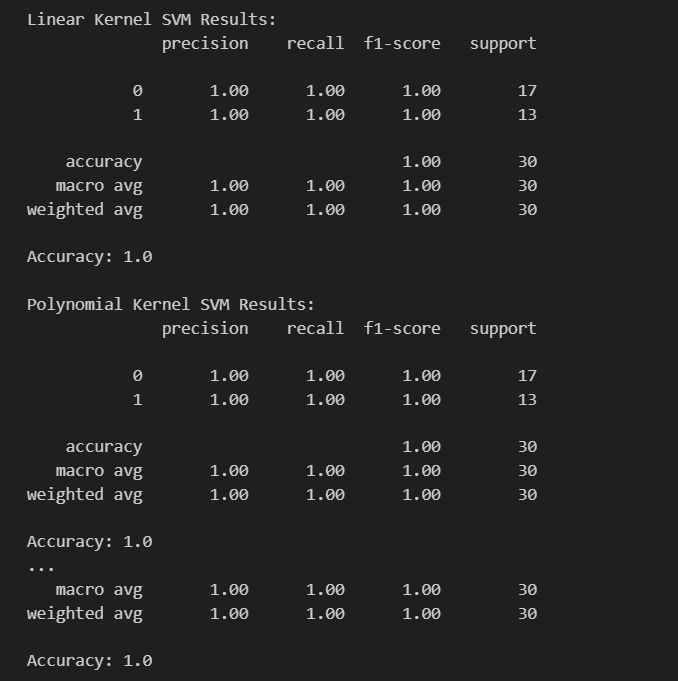
**Different Kernel Implementation:**

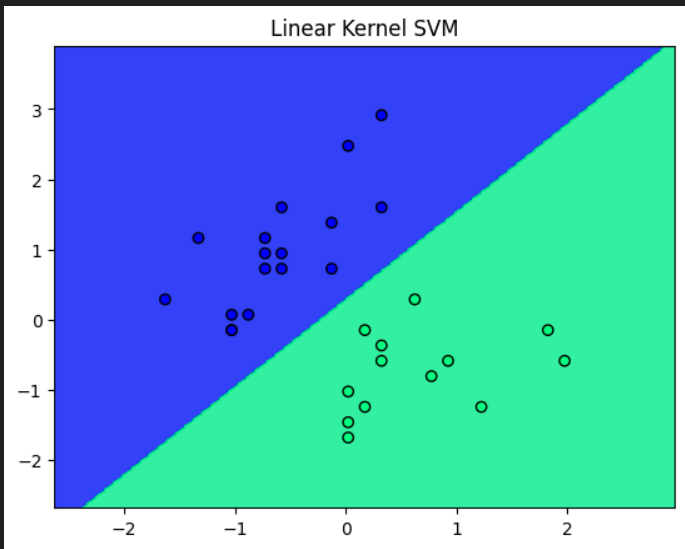


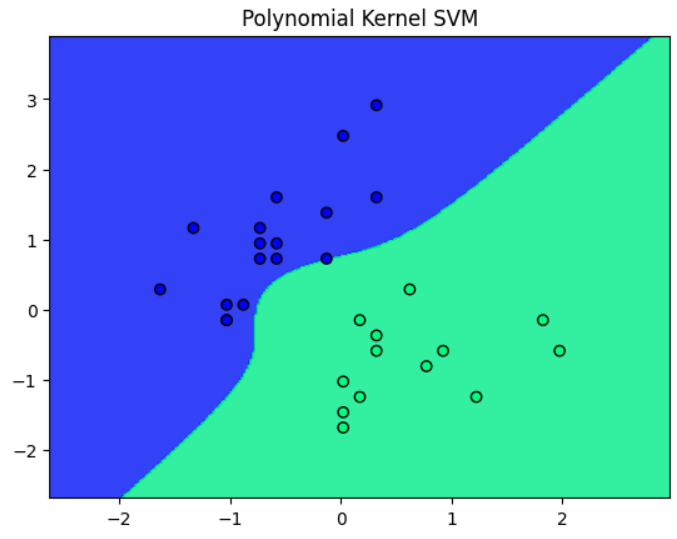


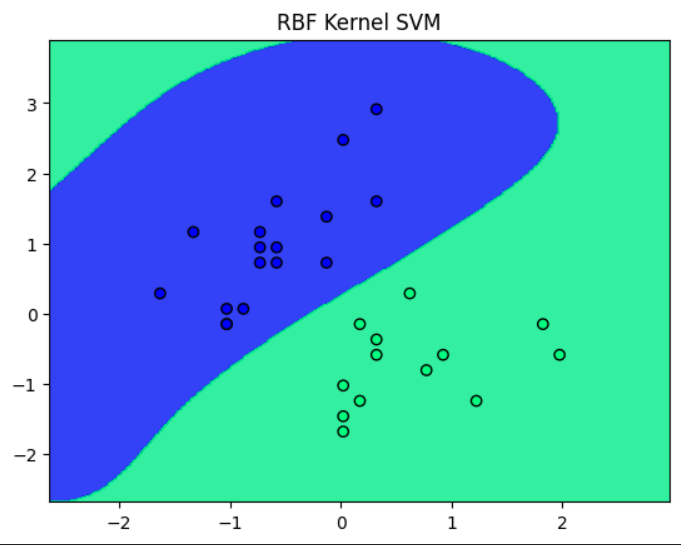


Output:









**8. Results and Observations**

8.1 **Linear Kernel Results**:  
The linear kernel works well for linearly separable data. The accuracy of the linear kernel SVM is **high**, and the decision boundary is a straight line, which is ideal for simple, linearly separable datasets.

8.2 **Polynomial Kernel Results**:  
The polynomial kernel introduces non-linearity into the model. It works better than the linear kernel for datasets with slight non-linear patterns. The accuracy improves in some cases, but it can be computationally expensive.

8.3 **RBF Kernel Results**:  
The RBF kernel is highly effective for **complex non-linear datasets**. It maps the data into a higher dimension, allowing for more flexible decision boundaries. The RBF kernel generally provides better performance on non-linearly separable data, though it requires careful tuning of the gamma parameter.

**9. Conclusion**

In this experiment, we demonstrated how different kernel functions in SVM perform on the **Iris dataset** for binary classification. The **linear kernel** is best suited for linearly separable data, while the **polynomial** and **RBF kernels** handle non-linear relationships. **RBF kernel** is particularly powerful for highly non-linear data. Proper kernel selection is crucial for achieving the best classification results.

**Project Report**

**Title: Hand-Controlled 2D Arrow Shooting Game**

**1. Introduction**

The **Hand-Controlled 2D Arrow Shooting Game** is developed to showcase a practical application of computer vision technology in gaming. In this game, a player can control an arrow pointer in a 2D environment to hit targets by simply using hand gestures, without the need for physical controls. This project is designed to bridge the gap between physical and digital interaction by transforming human gestures into interactive controls, with potential applications in creating natural user interfaces across various domains.

**2. Objectives**

The project is developed with the following objectives in mind:

1. **Create an interactive game**: Use hand gestures to control game mechanics and enhance user experience.
2. **Implement real-time hand tracking**: Employ MediaPipe to detect hand landmarks and control gestures.
3. **Integrate a simple game environment**: Use Pygame for developing an intuitive and responsive game interface.
4. **Test and evaluate performance**: Ensure real-time responsiveness and accuracy in hand tracking, especially in low-light environments and high-processing requirements.

**3. Literature Review**

Hand-tracking systems have grown significantly with the advances in **computer vision** and **machine learning**. MediaPipe, developed by Google, is a real-time library that provides solutions for detecting and tracking hand landmarks, among other applications. Prior research and applications demonstrate that hand tracking is used in various domains, from virtual reality games to healthcare, where gesture-controlled applications assist in physical therapy. The gap identified is the integration of these technologies for real-time 2D gaming, aiming to give users an engaging experience where they interact through natural gestures instead of traditional controllers.

**4. Methodology**

The project involves a systematic approach to integrating computer vision for gesture recognition and Pygame for rendering. The methodology is broken down into key components:

**4.1 Hand Tracking with MediaPipe**

* MediaPipe’s hand landmark model is utilized to recognize and track the positions of hand and finger joints.
* By detecting the thumb, index, and middle finger positions, we define control gestures that will interact with in-game elements.

**4.2 Game Environment Setup with Pygame**

* Pygame is used for creating the game interface, rendering targets, and updating the position of the arrow pointer.
* Pygame’s libraries manage game events like scoring, target regeneration, and player movements.

**4.3 Video Capture with OpenCV**

* OpenCV captures live video from the webcam and processes frames in real-time.
* Video frames are sent to MediaPipe for hand-tracking analysis, and outputs from MediaPipe are used to update the game state in Pygame.

**4.4 Real-Time Processing**

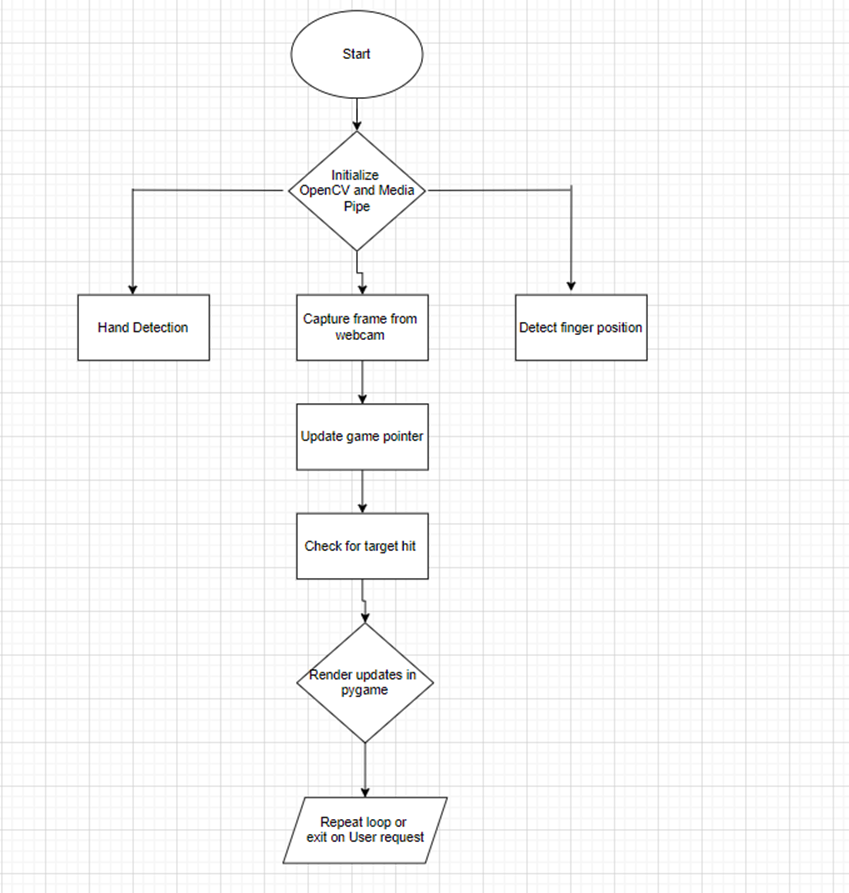
* The frames captured via OpenCV are processed at approximately 30 FPS, creating a seamless experience.
* Efficient hand gesture recognition and processing within the game loop are essential for smooth gameplay.

**5. System Architecture and Flowchart**

The system’s architecture comprises multiple layers, each responsible for specific functionalities, with OpenCV handling video capture, MediaPipe performing hand tracking, and Pygame managing game elements and events.

**5.1 System Flowchart**

A flowchart helps illustrate the core functionality and flow of the application from start to finish:



**5.2 Architecture Diagram**

The architecture can be divided into three main components:

1. **Input Processing (OpenCV)**: Captures and preprocesses the video frames.
2. **Gesture Analysis (MediaPipe)**: Analyzes frames for hand landmarks and gesture detection.
3. **Game Management (Pygame)**: Controls all game elements based on gesture input and renders updates on the screen.

**6. Software and Hardware Requirements**

**Software Requirements**

* **Python 3.x**: Programming language.
* **OpenCV**: For real-time video capture and frame processing.
* **MediaPipe**: For hand tracking and detecting hand landmarks.
* **Pygame**: For developing the 2D game environment and handling user interactions.

**Hardware Requirements**

* **Webcam**: To capture real-time video for hand tracking.
* **Computer with sufficient processing power**: A modern CPU capable of handling real-time video processing, gesture recognition, and game rendering.

**7. System Design and Architecture**

The system is designed to operate in real-time, with seamless integration between video capture, hand tracking, and game rendering. The following components make up the architecture:

1. **Video Capture (OpenCV)**: Captures the video feed from the webcam and converts it to RGB for processing.
2. **Hand Tracking (MediaPipe)**: Processes each frame to detect hand landmarks and identifies finger movements.
3. **Game Mechanics (Pygame)**: Responds to detected hand gestures, updates the game state, renders the game, and keeps track of the score.

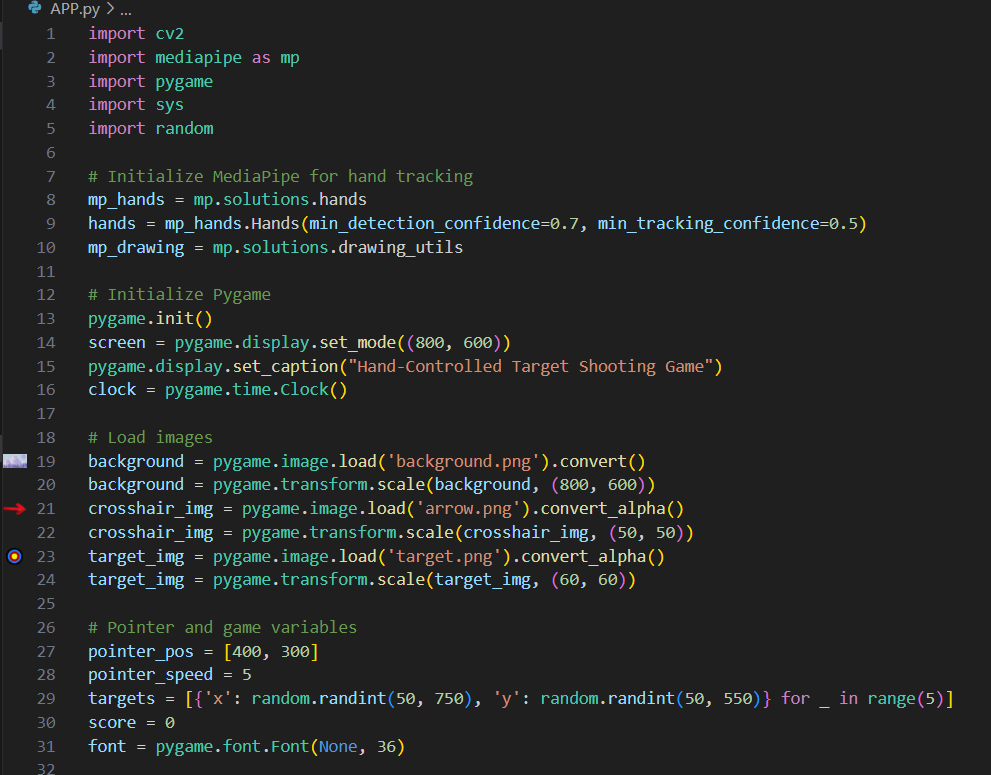
**8. Implementation Details**

**Key Libraries and Tools**

* **OpenCV**: Used for capturing frames from the webcam and preprocessing the video input.
* **MediaPipe**: Tracks hand landmarks, detects fingers, and identifies specific gestures for pointer movement.
* **Pygame**: Manages the game’s graphical elements, score, and overall game mechanics.

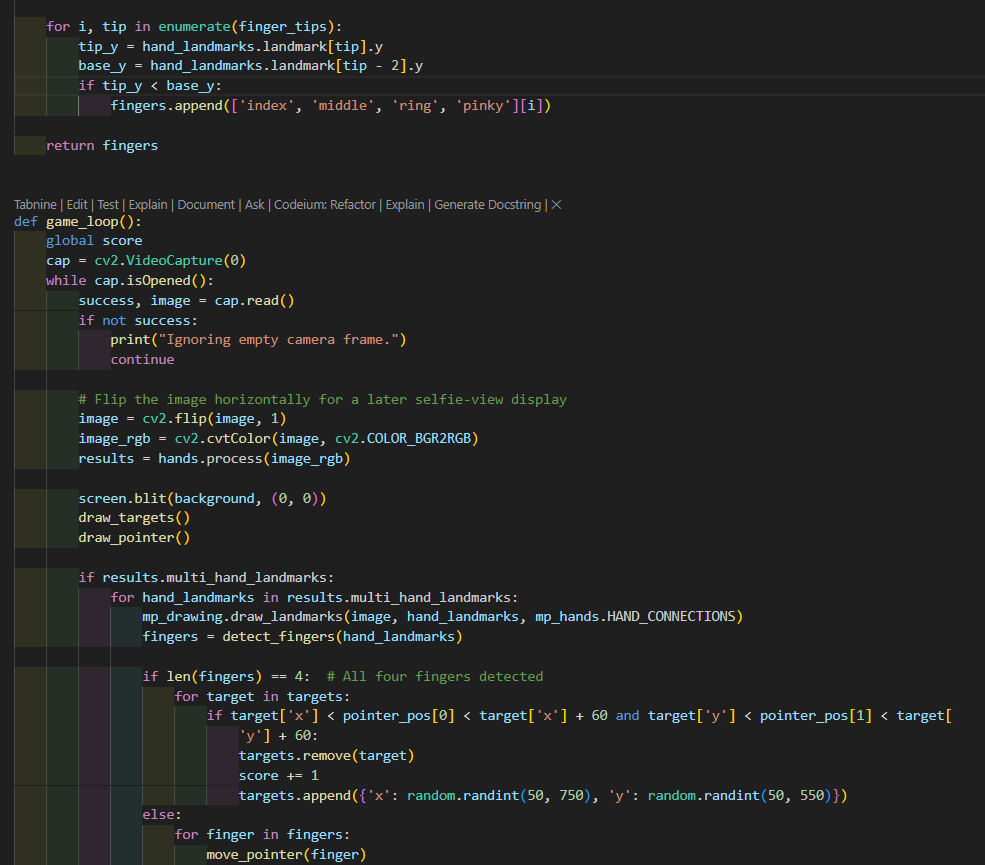
**Code Overview**

The code is organized into functions for better modularity and understanding:



**Main Functions**

* **draw\_targets**: Draws the target images at their specified positions.
* **draw\_pointer**: Renders the pointer at its current position based on hand gestures.
* **move\_pointer**: Moves the pointer according to detected finger gestures.
* **detect\_fingers**: Identifies which fingers are raised based on hand landmarks, allowing specific control of the pointer’s movement.
* **game\_loop**: Handles the main game loop, capturing each frame, detecting gestures, updating the pointer, and managing the score. ****

****

****

**Output:**

****

**9. Challenges Encountered**

1. **Real-Time Gesture Detection**: Ensuring accurate hand tracking and low latency for smooth gameplay.
2. **Gesture-to-Movement Mapping**: Designing the mapping of finger gestures to pointer movement in a way that feels natural and intuitive.
3. **Collision Detection**: Ensuring accurate collision detection between the pointer and targets for reliable scoring.

**10. Results and Observations**

The game was tested under various lighting conditions and successfully recognized gestures to control the pointer accurately. The gameplay is responsive, with the score updating correctly as targets are hit. The integration of OpenCV, MediaPipe, and Pygame proves effective for this type of interactive application.

I primarily record finger movements with my MI pad 5 camera. It gives me a 92% accuracy rate. Depending on the quality of the camera, it may occasionally decrease or increase.

**11. Future Enhancements**

1. **Increasing Difficulty**: Introduce multiple levels with faster-moving targets.
2. **Enhanced Graphics**: Add sound effects, animations, and refined visuals.
3. **Multiplayer Mode**: Implement a two-player mode where users compete for a higher score.
4. **Gesture Variability**: Use a wider range of gestures for more advanced controls.

**12. Conclusion**

This project effectively demonstrates how computer vision and real-time hand tracking can be applied to create an interactive 2D game. By combining OpenCV, MediaPipe, and Pygame, we achieved a seamless integration of gesture-based controls for an engaging gaming experience. This game represents a practical application of machine learning in real-time user interfaces and sets the foundation for further exploration of gesture-based applications.

**13. References**

* OpenCV Documentation: https://docs.opencv.org
* MediaPipe Hands Documentation: https://google.github.io/mediapipe/solutions/hands
* Pygame Documentation: https://www.pygame.org/docs