1. **Study of PROLOG Programming Language and its Function. Write simple facts for the statements using PROLOG.**

#### Logic and Functional Programming

Prolog is a logic-based programming language, a distinct paradigm compared to imperative or object-oriented programming languages. In logic programming, we specify what needs to be done without specifying how it should be done. The focus is on defining facts and relationships, allowing the Prolog engine to infer new facts and answer queries based on logical inference.

Functional programming, on the other hand, involves creating functions and mathematical computations to solve problems. Prolog’s approach differs because it uses a declarative style to define the logical structure of problems.

#### What is Prolog?

Prolog, short for **PROgramming in LOGic**, was developed in the early 1970s as a language for solving problems involving symbolic reasoning and non-numeric computation, especially in artificial intelligence. Its main components are:

* **Facts**: Simple assertions about objects or relationships.
* **Rules**: Inferences or conditions that derive new facts from existing ones.
* **Queries**: Questions posed to the system to obtain answers based on the defined facts and rules.

Prolog is useful for tasks such as:

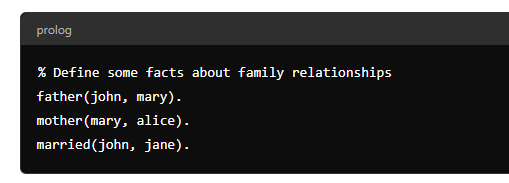
* Knowledge representation
* Natural language processing
* Automated reasoning
* Expert systems

### 2. Prolog Basics

#### Facts

In Prolog, facts represent basic assertions about objects or relationships. These are simple statements that declare a relationship between entities, without any conditions.

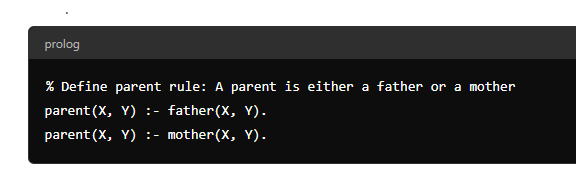
Example of Facts:



#### Rules

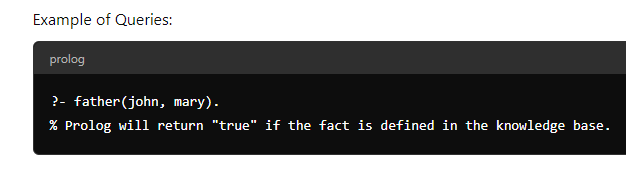
Rules allow Prolog to make inferences from existing facts. They define conditions under which certain relationships or facts hold.

Example of a Rule:



#### Queries

Queries are questions that can be asked to Prolog to check whether a certain relationship or fact is true. The Prolog engine tries to answer queries based on the available facts and rules.



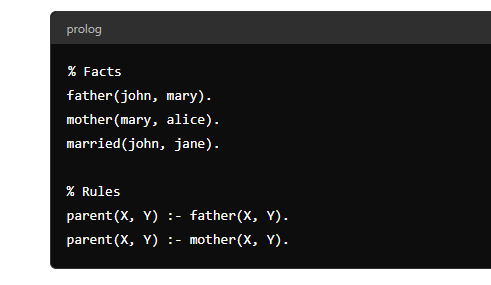
#### Knowledge Base in Logic Programming

A **knowledge base** in Prolog is a collection of facts and rules that represent the logical structure of a problem domain. Prolog works by performing logical inference over this knowledge base to answer queries or deduce new information.

Creating a knowledge base involves:

1. Declaring facts (e.g., relationships or properties).
2. Defining rules (logical inferences based on facts).
3. Interacting with the knowledge base via queries.

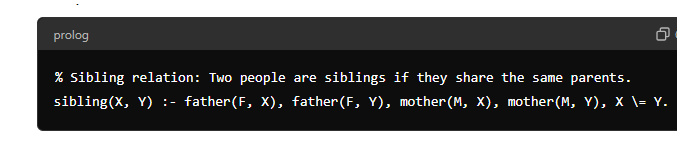
Example Knowledge Base:



#### Relations in Prolog

Prolog can represent various kinds of relations between entities. Relations are expressed in terms of facts and rules, which can be used to infer new relationships through logical queries. These relations can involve family relationships, business connections, or other types of associations.

**Example of a Relation**:



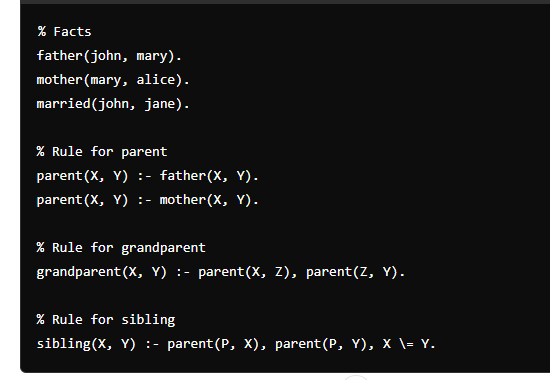
#### Family Relationship in Prolog

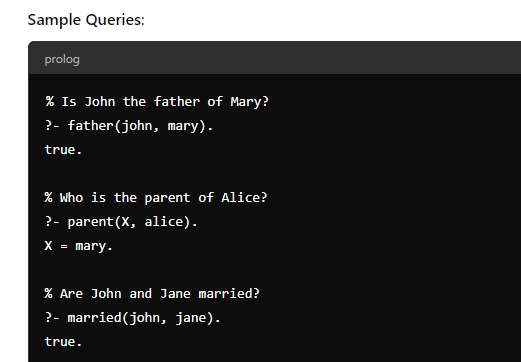
Let’s consider a simple family tree to represent relationships using facts and rules in Prolog.

**Family Tree Representation**:

* John is the father of Mary.
* Mary is the mother of Alice.
* John is married to Jane.

These relationships can be expressed using Prolog facts and rules:





1. **Implementation of Depth First Search for Water Jug Problem**

#### ****Objective:****

To develop a Prolog program that implements Depth First Search (DFS) to solve the Water Jug problem. The objective is to measure a specific quantity of water using two jugs of different capacities.

#### ****Theory:****

The Water Jug problem is a classical problem where two jugs of given capacities are used to measure a desired quantity of water. Given two jugs, with capacities of X and Y liters, and an unlimited water supply, the goal is to find a sequence of operations (fill, empty, pour) that results in one of the jugs containing exactly the required amount of water (say Z liters).

**Depth First Search (DFS)** is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root (initial state) and explores as far as possible along each branch before backtracking. It is particularly useful in solving problems like the Water Jug problem, where paths can be recursively explored.

from collections import deque

def BFS(a, b, target):

# Map is used to store the states, every

# state is hashed to binary value to

# indicate either that state is visited

# before or not

m = {}

isSolvable = False

path = []

# Queue to maintain states q = deque()

# Initialing with initial state

q.append((0, 0))

while (len(q) > 0):

# Current state

u = q.popleft()

# q.pop() #pop off used state

# If this state is already visited

if ((u[0], u[1]) in m):

continue

# Doesn't met jug constraints

if ((u[0] > a or u[1] > b or

u[0] < 0 or u[1] < 0)):

continue

# Filling the vector for constructing

# the solution path

path.append([u[0], u[1]])

# Marking current state as visited

m[(u[0], u[1])] = 1

# If we reach solution state, put ans=1

if (u[0] == target or u[1] == target):

isSolvable = True

if (u[0] == target):

if (u[1] != 0):

# Fill final state

path.append([u[0], 0])

else:

if (u[0] != 0):

# Fill final state

path.append([0, u[1]])

# Print the solution path

sz = len(path)

for i in range(sz):

print("(", path[i][0], ",",

path[i][1], ")")

break

# If we have not reached final state

# then, start developing intermediate

# states to reach solution state

q.append([u[0], b]) # Fill Jug2

q.append([a, u[1]]) # Fill Jug1

for ap in range(max(a, b) + 1):

# Pour amount ap from Jug2 to Jug1

c = u[0] + ap

d = u[1] - ap

# Check if this state is possible or not

if (c == a or (d == 0 and d >= 0)):

#if (c == a or (d >= 0)):

q.append([c, d])

# Pour amount ap from Jug 1 to Jug2

c = u[0] - ap

d = u[1] + ap

# Check if this state is possible or not

if ((c == 0 and c >= 0) or d == b):

q.append([c, d])

# Empty Jug2

q.append([a, 0])

# Empty Jug1

q.append([0, b])

# No, solution exists if ans=0

if (not isSolvable):

print("No solution")

# Driver code

if \_\_name\_\_ == '\_\_main\_\_':

Jug1, Jug2, target = 4, 3, 2

print("Path from initial state "

"to solution state ::")

BFS(Jug1, Jug2, target)

**OUTPUT**

Path from initial state to solution state ::

( 0 , 0 )

( 0 , 3 )

( 4 , 0 )

( 4 , 3 )

( 3 , 0 )

( 1 , 3 )

( 3 , 3 )

( 4 , 2 )

( 0 , 2 )

1. **Implementation of Breadth First Search for Tic-Tac-Toe problem.**

#### ****Objective:****

To develop a Prolog program that implements Breadth First Search (BFS) to solve the Tic-Tac-Toe problem. The goal is to determine the optimal moves or check if there is a winning strategy for either player using BFS.

#### ****Theory:****

Tic-Tac-Toe is a simple two-player game where players take turns placing 'X' and 'O' on a 3x3 grid. The objective is to place three of your marks in a horizontal, vertical, or diagonal row. BFS is a graph traversal technique used to explore all possible game states level by level, ensuring that the shortest solution is found.

In this implementation, the BFS algorithm explores the game tree generated by all possible moves for each player. By exploring level by level, BFS ensures that we find the quickest possible win, draw, or determine the next best move.

import queue

def print\_board(board):

for row in board:

print(" | ".join(row))

print("-" \* 9)

def is\_winner(board, player):

for row in board:

if all(cell == player for cell in row):

return True

for col in range(3):

if all(board[row][col] == player for row in range(3)):

return True

if all(board[i][i] == player for i in range(3)) or all(board[i][2 - i] == player for i in range(3)):

return True

return False

def is\_draw(board):

for row in board:

if " " in row:

return False

return True

def is\_valid\_move(board, row, col):

return 0 <= row < 3 and 0 <= col < 3 and board[row][col] == " "

def bfs\_ai\_move(board, player):

q = queue.Queue()

q.put(board)

while not q.empty():

current\_board = q.get()

for row in range(3):

for col in range(3):

if current\_board[row][col] == " ":

new\_board = [row[:] for row in current\_board]

new\_board[row][col] = player

if is\_winner(new\_board, player):

return row, col

q.put(new\_board)

return None # If no winning move is found, return None

# main game loop

def play\_tic\_tac\_toe():

board = [[" " for \_ in range(3)] for \_ in range(3)]

players = ["X", "O"]

current\_player = players[0]

while True:

print\_board(board)

if current\_player == "X":

row = int(input(f"Enter row (0-2) for player X: "))

col = int(input(f"Enter column (0-2) for player X: "))

else:

print("Player O (AI) is making a move...")

ai\_move = bfs\_ai\_move(board, current\_player)

if ai\_move:

row, col = ai\_move

else:

print("AI couldn't find a winning move, making a random move.")

# Implement a more advanced AI algorithm here.

row, col = 0, 0 # For simplicity, making a random move.

if is\_valid\_move(board, row, col):

board[row][col] = current\_player

if is\_winner(board, current\_player):

print\_board(board)

print(f"Player {current\_player} wins!")

break

if is\_draw(board):

print\_board(board)

print("It's a draw!")

break

current\_player = players[1] if current\_player == players[0] else players[0]

else:

print("Invalid move. Please try again.")

# main Function

if \_\_name\_\_ == "\_\_main\_\_":

play\_tic\_tac\_toe()

1. **Write a PROLOG program to solve N-queens problem.**

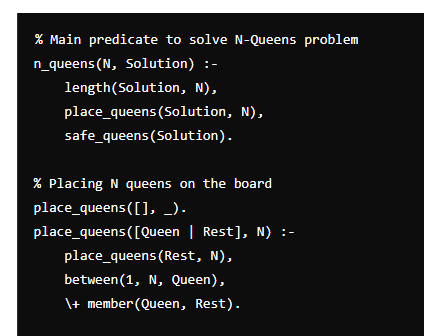
#### ****Objective:****

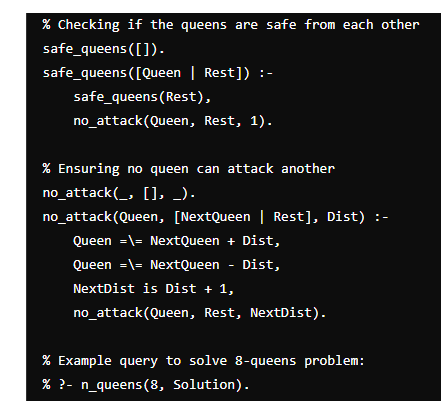
To develop a Prolog program that solves the N-Queens problem, where the goal is to place N queens on an N×N chessboard such that no two queens attack each other.

#### ****Theory:****

The N-Queens problem is a classic combinatorial problem in which we aim to place N queens on a chessboard of size N×N in such a way that no queen can attack another. A queen can attack another if it is placed in the same row, column, or diagonal.

The solution uses backtracking to place queens one by one, checking after each placement if the queen is safe from others. If it’s safe, we proceed to place the next queen; otherwise, we backtrack.





1. **Implementation of Travelling Salesman Problem.**

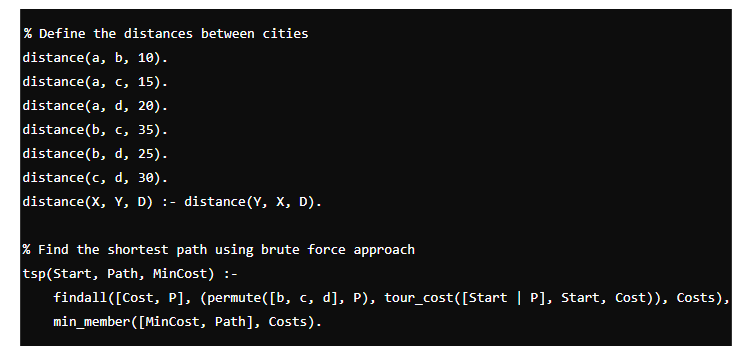
#### ****Objective:****

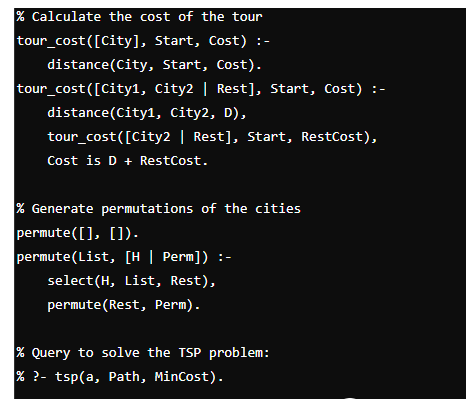
To develop a Prolog program that solves the Travelling Salesman Problem (TSP), aiming to find the shortest possible route that visits a set of cities exactly once and returns to the starting city.

#### ****Theory:****

The Travelling Salesman Problem (TSP) is a well-known combinatorial optimization problem where a salesman must visit a set of cities, visiting each city exactly once, and return to the starting city. The objective is to minimize the total distance traveled.

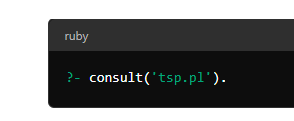
TSP is NP-hard, and solutions can be found using brute-force search or heuristic methods. In this implementation, a brute-force approach is used to generate all possible permutations of city visits and calculate the shortest path.





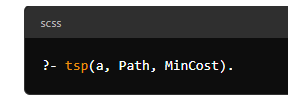
#### ****Steps to Execute the Program:****

1. **Install Prolog:**
   * Download and install SWI-Prolog from SWI-Prolog website.
   * Open the SWI-Prolog interpreter.
2. **Create the Program:**
   * Open a text editor and write the Prolog program for solving the TSP.
   * Save the file with a .pl extension (e.g., tsp.pl).
3. **Load the Program in Prolog:**
   * Open the SWI-Prolog interpreter.
   * Load the program using the consult/1 predicate or [filename]. command:



**Run the Query:**

* After loading the program, execute the following query to solve the TSP problem starting from city a:



1. **Implementation of Python Basic Libraries such as,**
2. **Math, Numpy and Scipy.**
3. **Pandas and Matplotlib**

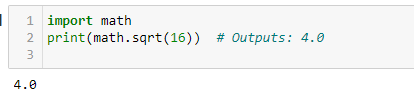
### Introduction

The Python programming language is renowned for its versatility and extensive library support. Among these libraries, Math, Numpy, and Scipy stand out for their powerful capabilities in numerical and scientific computing. This observation will delve into the theory and practical applications of these libraries, providing a solid foundation for their implementation.

**Difference between Module, library and Package**

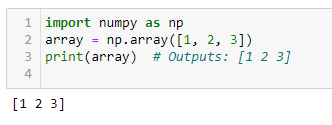
**Module**

* **Definition:** A module is a single file containing Python code. It can include functions, classes, and variables that can be used in other Python scripts.
* **Example:** The math module is a built-in module in Python that provides mathematical functions. It is implemented in a single file named math.py in the Python standard library.
* **Usage:** Modules are imported using the import statement. For instance



**Package**

* **Definition:** A package is a collection of Python modules organized in a directory hierarchy. A package directory contains a special file named \_\_init\_\_.py (which can be empty) to indicate that the directory is a package.
* **Example:** The numpy library is an example of a package. It includes several modules and sub-packages.
* **Usage:** Packages are imported using dot notation to access specific modules within the package. For instance:

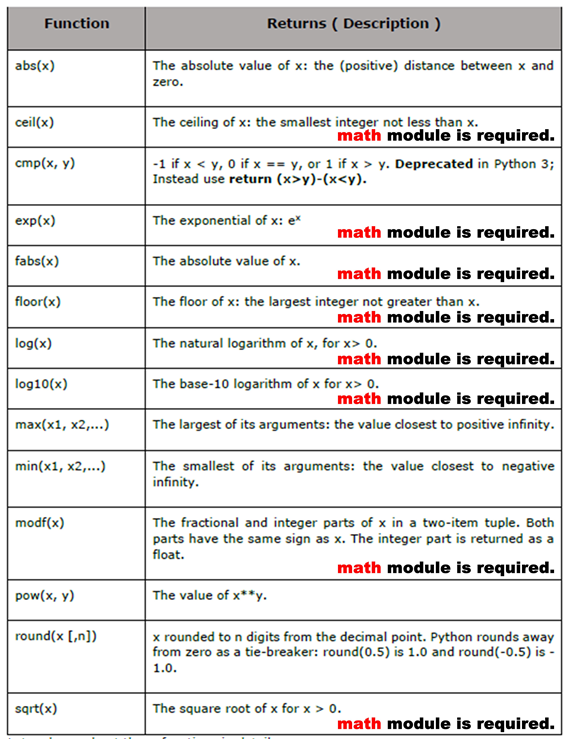


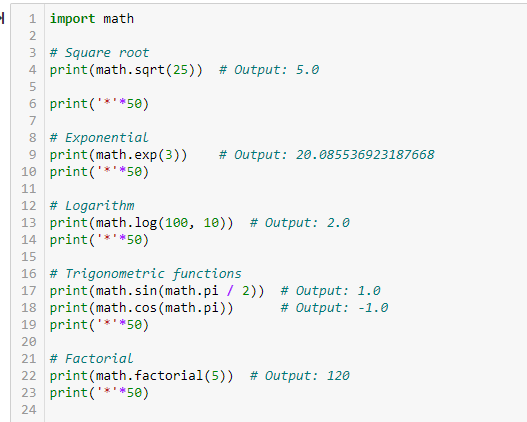
**Library**

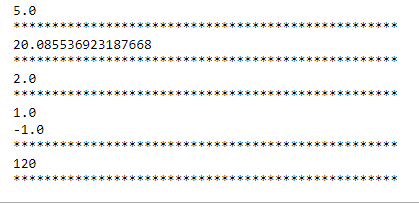
* **Definition:** The term "library" refers to a collection of related modules and packages. It is a broader concept than a module or package and encompasses all code that is available for use in a particular domain.
* **Example:** SciPy is an example of a library. It includes multiple packages and modules for scientific and technical computing.
* **Usage:** Libraries are installed and managed using package managers like pip. For instance

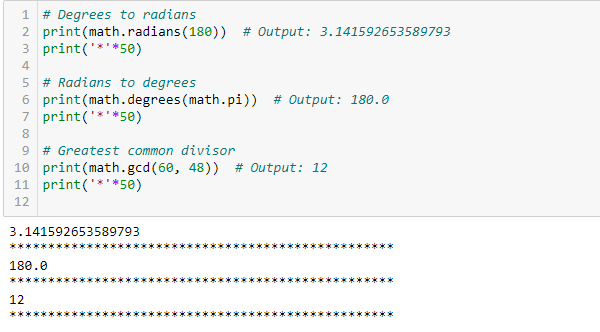


1. **Math module in python**

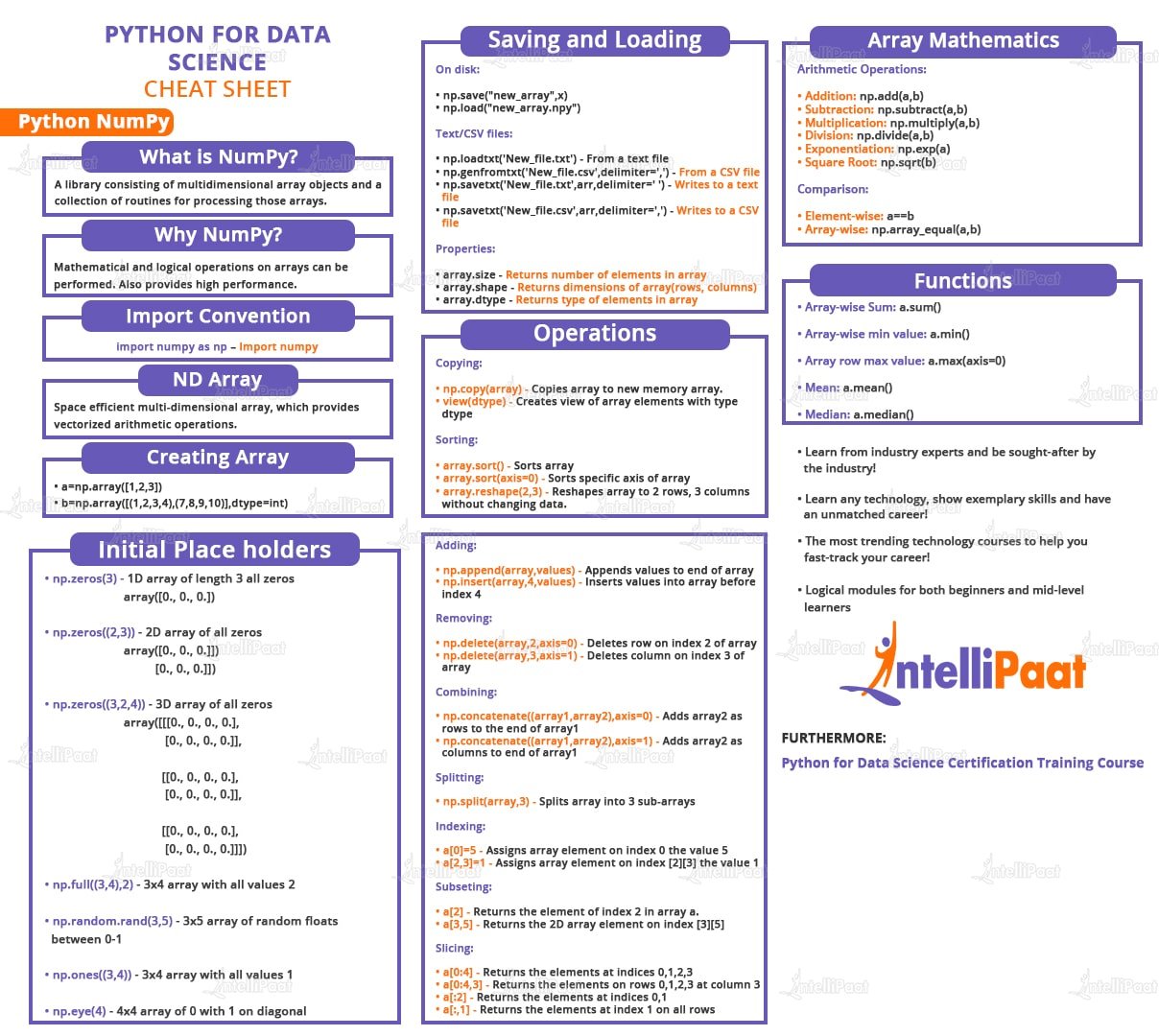








1. **Numpy library in Python**



NumPy is an open-source library available in Python that aids in mathematical, scientific, engineering, and data science programming. NumPy is an incredible library to perform mathematical and statistical operations. It works perfectly well for multi-dimensional arrays and matrices multiplication.

For any scientific project, NumPy is the tool to know. It has been built to work with the N- dimensional array, linear algebra, random number, Fourier transform, etc. It can be integrated to C/C++ and Fortran.

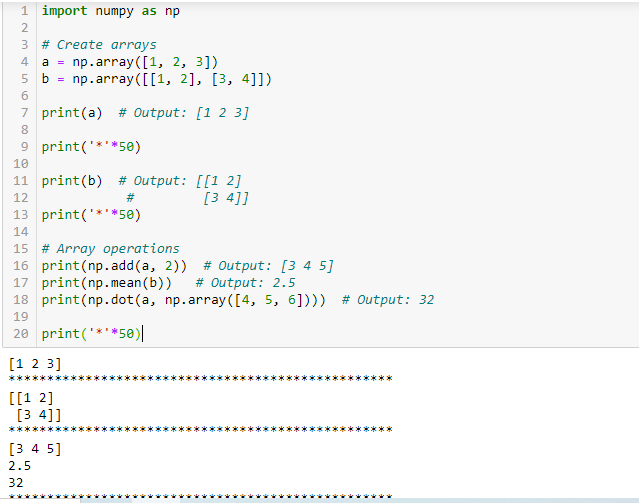
NumPy is a programming language that deals with multi-dimensional arrays and matrices. On top of the arrays and matrices, NumPy supports a large number of mathematical operations.

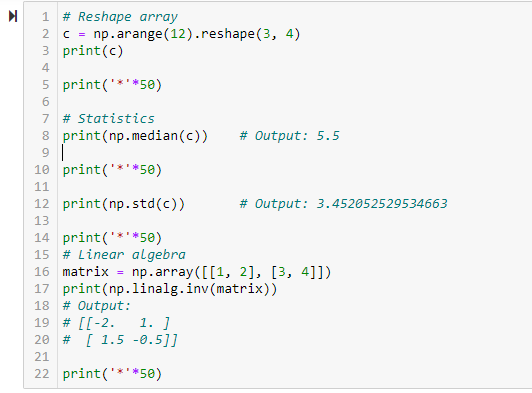
NumPy is memory efficient, meaning it can handle the vast amount of data more accessible than any other library. Besides, NumPy is very convenient to work with, especially for matrix multiplication and reshaping. On top of that, NumPy is fast. In fact, TensorFlow and Scikitlearn use NumPy array to compute the matrix multiplication in the back end.

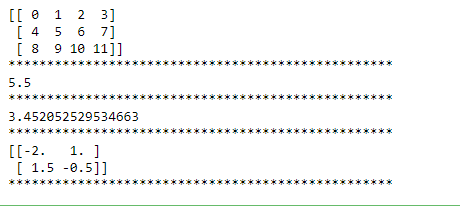
* **Arrays in NumPy:** NumPy’s main object is the homogeneous multidimensional array.
* It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers.
* In NumPy dimensions are called axes. The number of axes is rank.
* NumPy’s array class is called **ndarray**. It is also known by the alias **array**.

We use python numpy array instead of a list because of the below three reasons:

1. Less Memory
2. Fast
3. Convenient

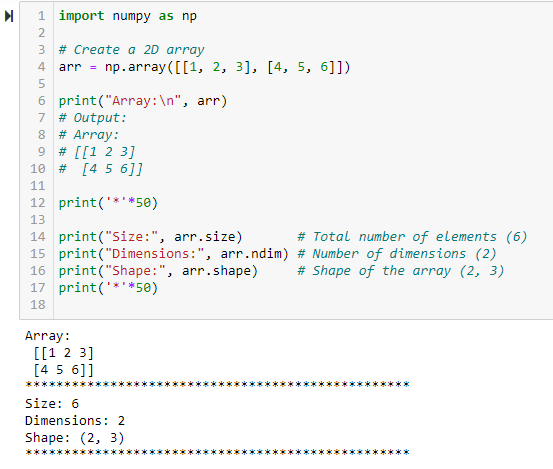






**Size, Dimension, and Shape**

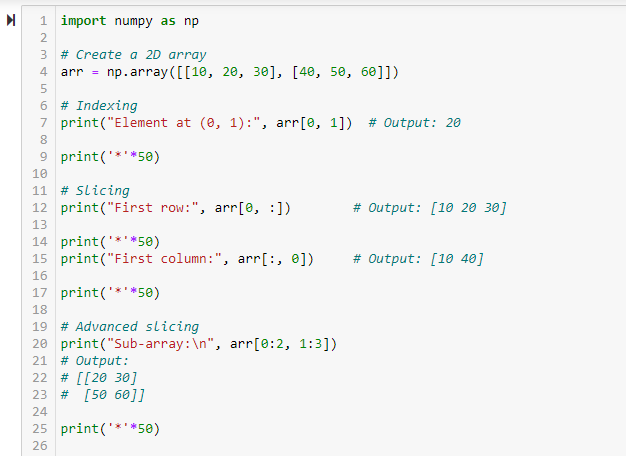
* **Size**: Total number of elements in the array.
* **Dimension**: Number of axes or dimensions.
* **Shape**: Tuple indicating the size of each dimension.

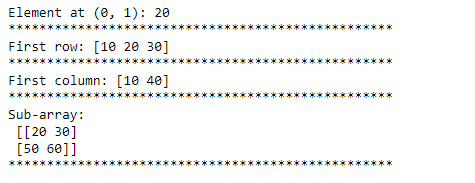


### **Indexing and Slicing**

Indexing allows you to access specific elements, while slicing helps you extract a portion of the array.

**Example: Indexing and Slicing**





**3. Python Scipy Library**

SciPy is an Open Source Python-based library, which is used in mathematics, scientific computing, Engineering, and technical computing. SciPy also pronounced as "Sigh Pi."

* SciPy contains varieties of sub packages which help to solve the most common issue related to Scientific Computation.
* SciPy is the most used Scientific library only second to GNU Scientific Library for C/C++ or Matlab's.
* Easy to use and understand as well as fast computational power.
* It can operate on an array of NumPy library.

**Numpy VS SciPy**

**Numpy:**

1. Numpy is written in C and used for mathematical or numeric calculation.
2. It is faster than other Python Libraries
3. Numpy is the most useful library for Data Science to perform basic calculations.
4. Numpy contains nothing but array data type which performs the most basic operation like sorting, shaping, indexing, etc.

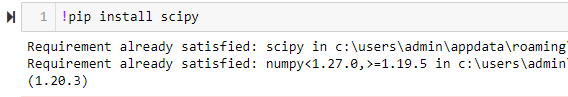
**SciPy:**

1. SciPy is built in top of the NumPy
2. SciPy is a fully-featured version of Linear Algebra while Numpy contains only a few features.
3. Most new Data Science features are available in Scipy rather than Numpy. 1. Linear Algebra of SciPy is an implementation of BLAS and ATLAS LAPACK libraries.
4. Performance of Linear Algebra is very fast compared to BLAS and LAPACK.
5. Linear algebra routine accepts two-dimensional array object and output is also a two- dimensional array.

**Linear Algebra with SciPy**

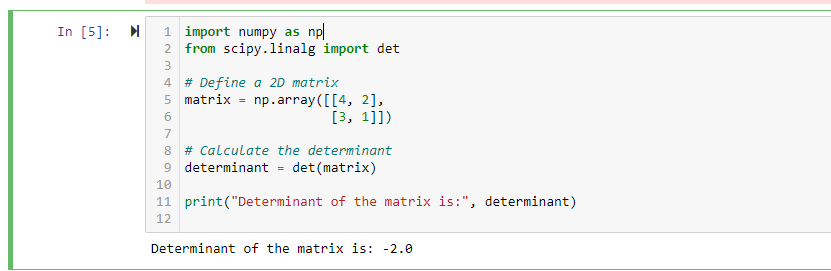
Basic linear algebra operations include solving linear systems and computing matrix properties.

Now let's do some test with **scipy.linalg,**

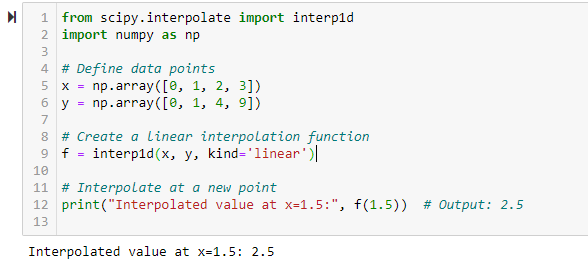


**Calculate the determinant**:

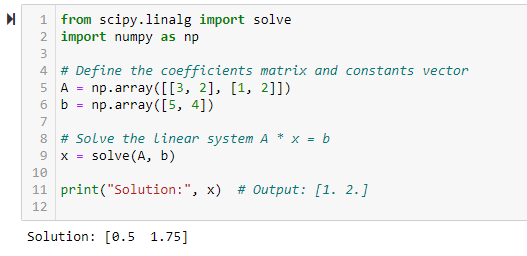
Here’s a sample code to compute the determinant of a 2D matrix:



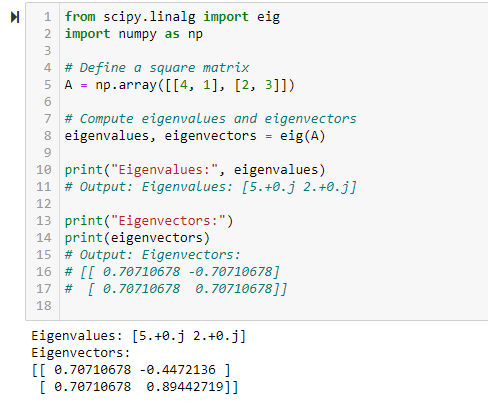
**Interpolation**



**Example: Solving a System of Linear Equations**



**Eigen Values and Eigen Vectors`**

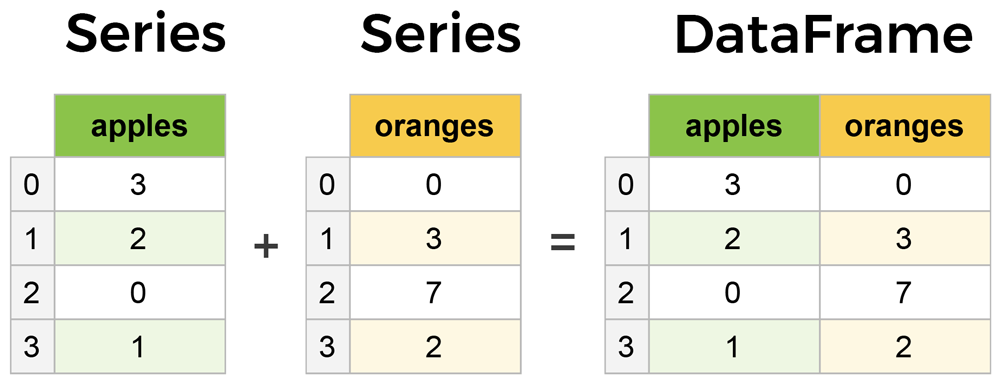


**4. Pandas in Python**

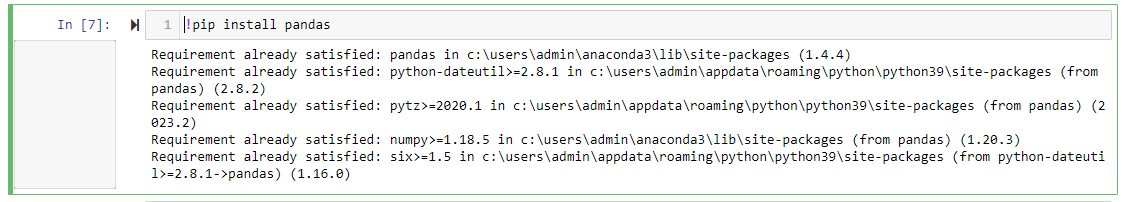
**Pandas** is a powerful data manipulation and analysis library for Python. It provides data structures like Series and DataFrame that make it easy to work with structured data.

#### **Key Components:**

* **Series:** A one-dimensional array-like object.
* **DataFrame:** A two-dimensional table (like a spreadsheet) where you can store data with rows and columns.



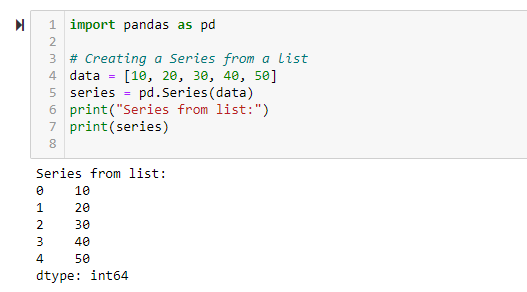
#### **Basic Operations:**

**Installation:**

**Importing Pandas:**



**Creating a Series**

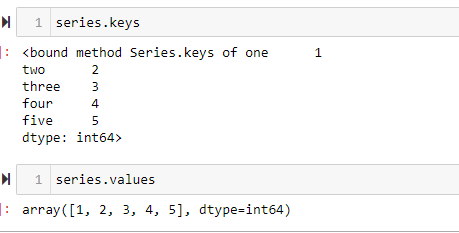


**Creating a Series with Custom Index**

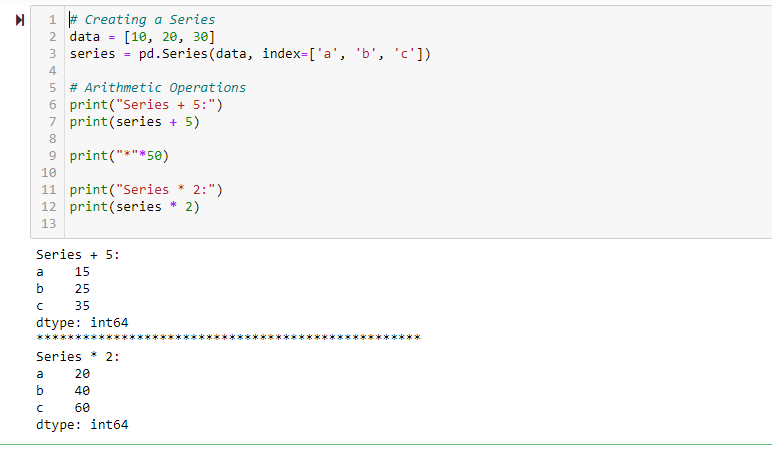


**Accessing Elements**

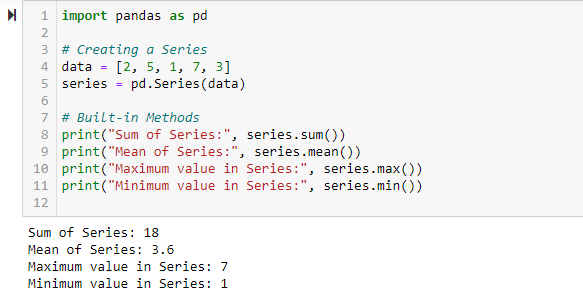




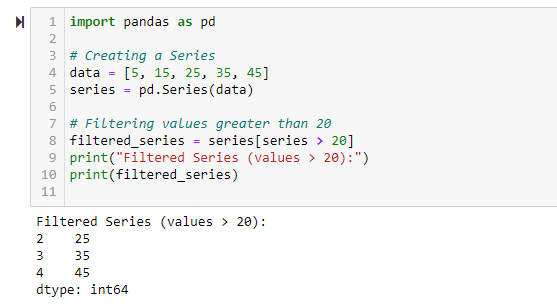
**Basic Operations**



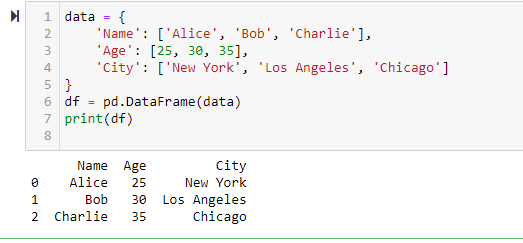
**Using Built-in Methods**



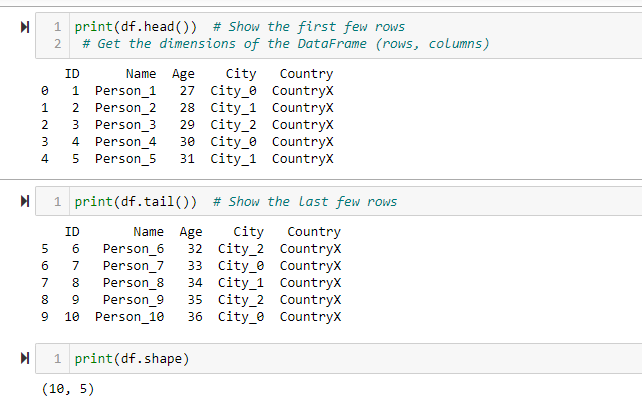
**Filtering Data**



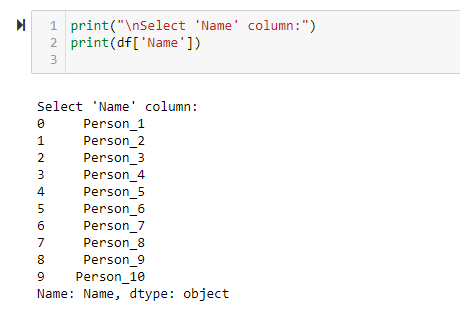
**Creating a DataFrame:**



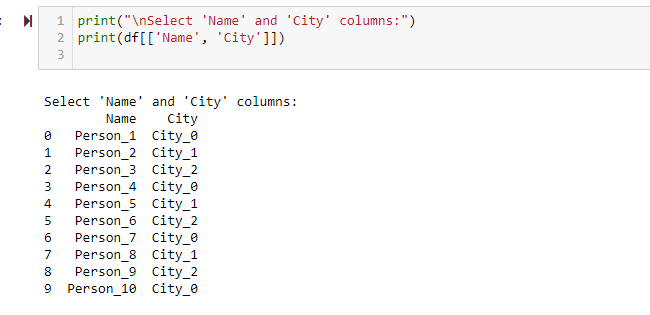




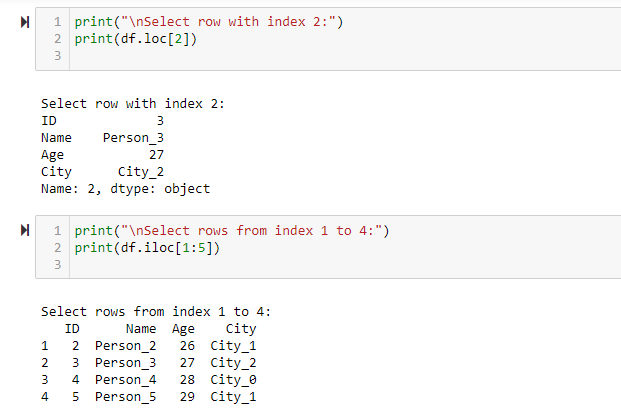
**Select a single column**



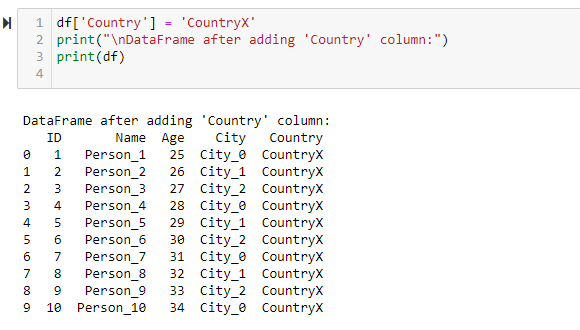
**Select multiple columns**



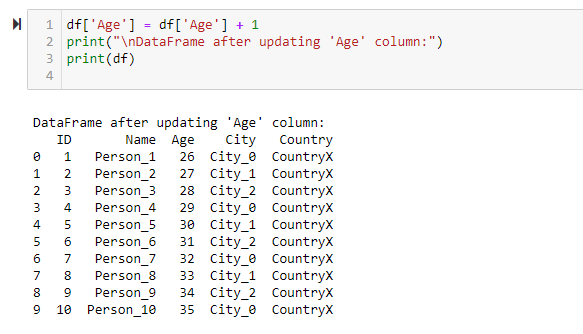
**Indexing**



**Adding a column**



**Updating a column**



**5. Matplotlib Library in Python**

Pyplot is a module of Matplotlib which provides simple functions to add plot elements like lines, images, text, etc. to the current axes in the current figure.

## **Matplotlib Methods for Plotting**

* **plot(x-axis values, y-axis values)**
  + Plots a simple line graph with x-axis values against y-axis values.
* **show()**
  + Displays the graph.
* **title("string")**
  + Sets the title of the plot as specified by the string.
* **xlabel("string")**
  + Sets the label for the x-axis as specified by the string.
* **ylabel("string")**
  + Sets the label for the y-axis as specified by the string.
* **figure()**
  + Used to control figure-level attributes.
* **subplot(nrows, ncols, index)**
  + Adds a subplot to the current figure.
* **suptitle("string")**
  + Adds a common title to the figure specified by the string.
* **subplots(nrows, ncols, figsize)**
  + A convenient way to create subplots in a single call. Returns a tuple of a figure and a number of axes.
* **set\_title("string")**
  + An axes-level method used to set the title of subplots in a figure.
* **bar(categorical variables, values, color)**
  + Creates vertical bar graphs.
* **barh(categorical variables, values, color)**
  + Creates horizontal bar graphs.
* **legend(loc)**
  + Adds a legend to the graph.
* **xticks(index, categorical variables)**
  + Gets or sets the current tick locations and labels of the x-axis.
* **pie(values, categorical variables)**
  + Creates a pie chart.
* **hist(values, number of bins)**
  + Creates a histogram.
* **xlim(start value, end value)**
  + Sets the limit of values for the x-axis.
* **ylim(start value, end value)**
  + Sets the limit of values for the y-axis.
* **scatter(x-axis values, y-axis values)**
  + Plots a scatter plot with x-axis values against y-axis values.
* **axes()**
  + Adds an axes to the current figure.
* **set\_xlabel("string")**
  + An axes-level method used to set the x-label of the plot specified as a string.
* **set\_ylabel("string")**
  + An axes-level method used to set the y-label of the plot specified as a string.
* **scatter3D(x-axis values, y-axis values, z-axis values)**
  + Plots a three-dimensional scatter plot with x-axis, y-axis, and z-axis values.
* **plot3D(x-axis values, y-axis values, z-axis values)**
  + Plots a three-dimensional line graph with x-axis, y-axis, and z-axis values.

**Installation:**

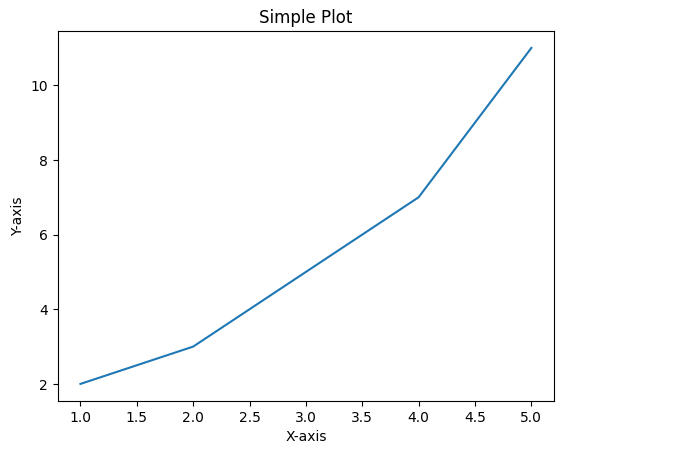


**Importing Matplotlib:**

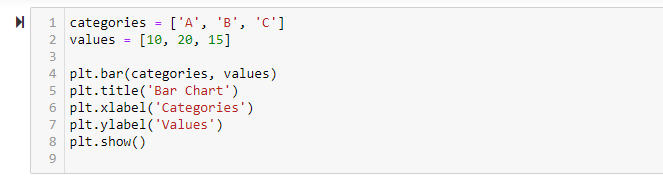


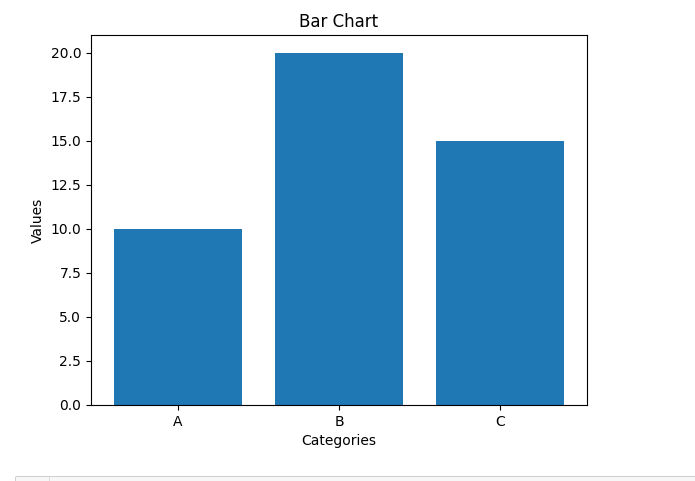
**Creating a Simple Plot:**



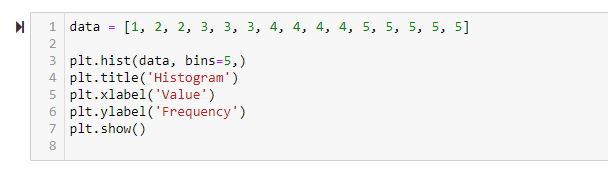


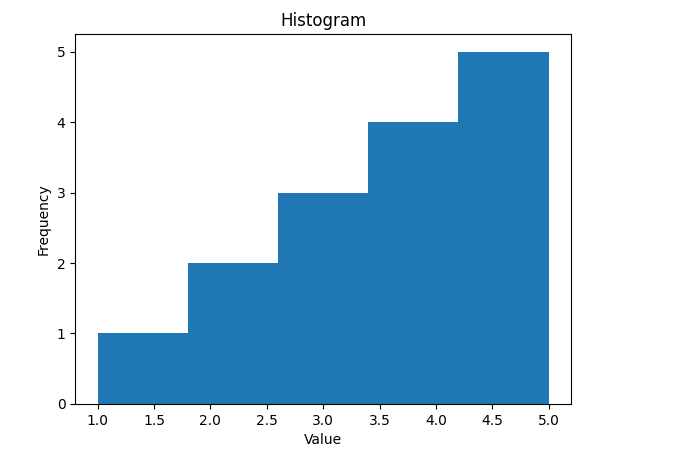
**Creating a Bar Chart:**



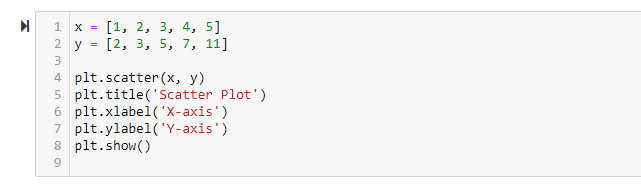


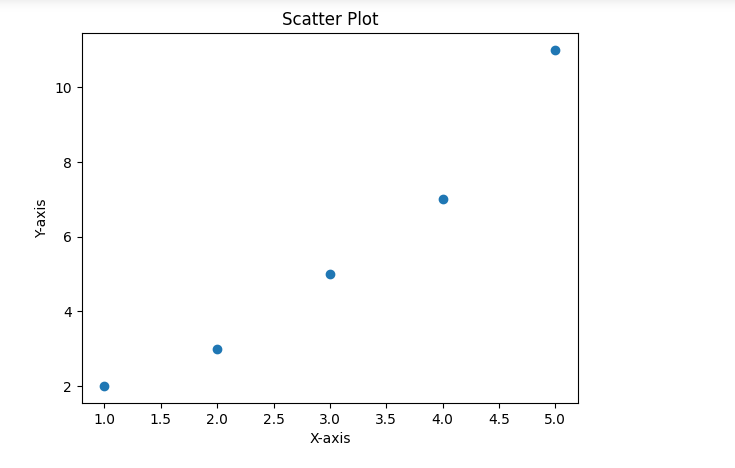
**Creating a Histogram:**





**Creating a Scatter Plot:**

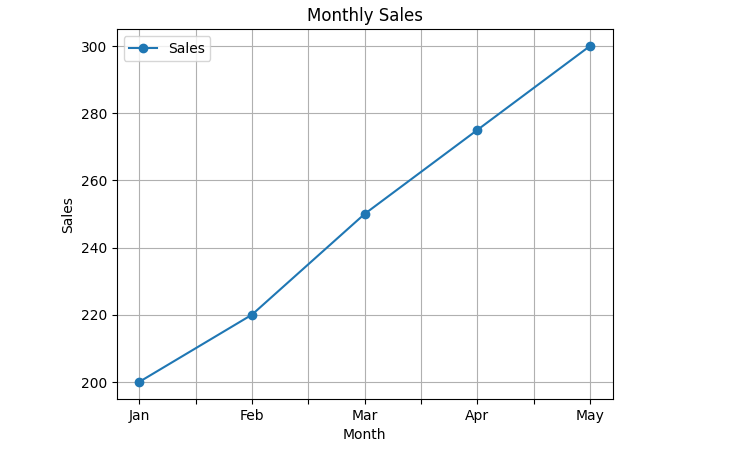




### **Combining Pandas and Matplotlib**

You can use Pandas and Matplotlib together for data visualization. Here’s an example:



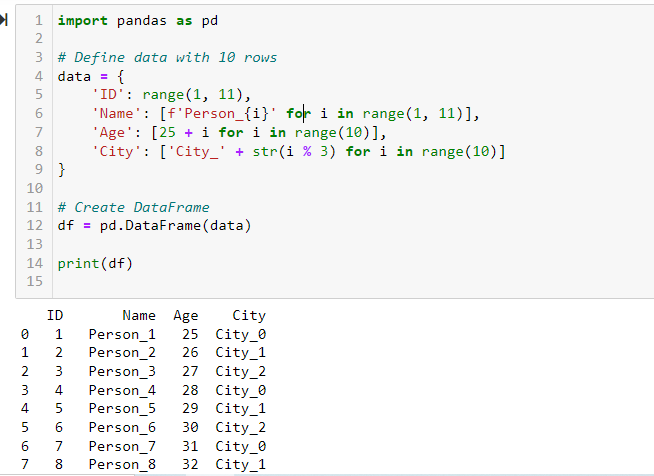


**7. a) Creation and loading of different datasets in Python.**

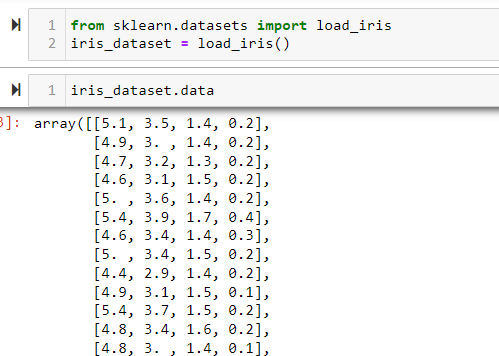
**b) Write a python program to compute mean, Median, Mode, variance, and Standard Deviation**

**a) Creation and loading of different datasets in Python.**

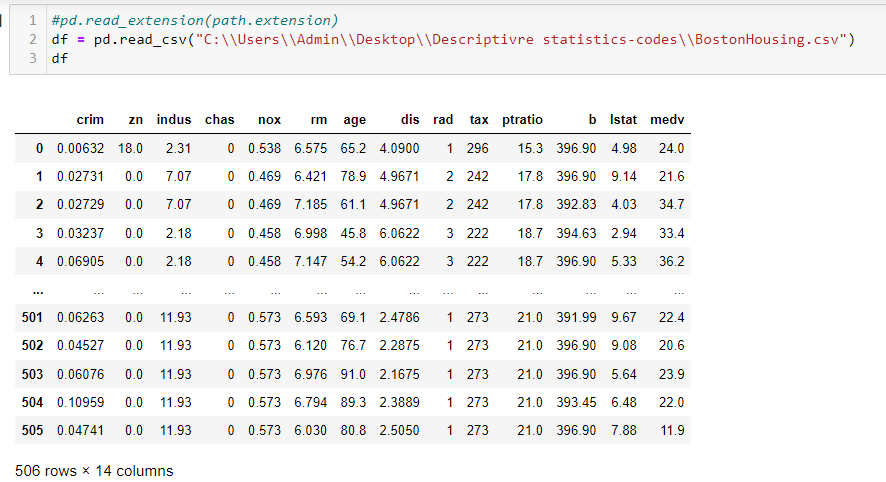
**Method 1:**



**Method 2**



**Method 3:**



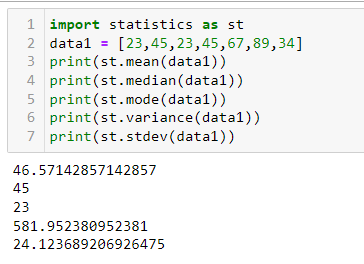
**b) Write a python program to compute mean, Median, Mode, variance, and Standard Deviation**

**Descriptive Statistics**

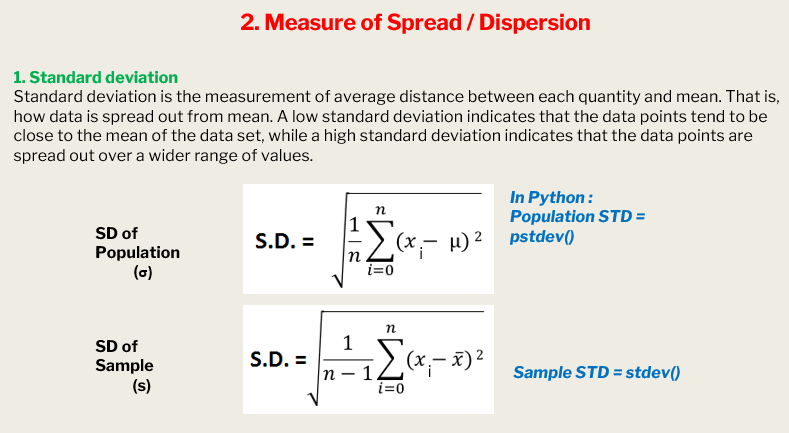
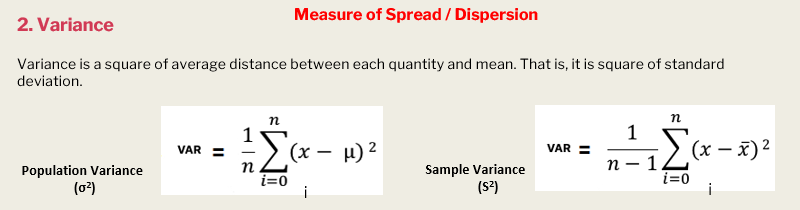
deals with the processing of data without attempting to draw any inferences from it. The characteristics of the data are described in simple terms. Events that are dealt with include everyday happenings such as accidents, prices of goods, business, incomes, epidemics, sports data, population data.

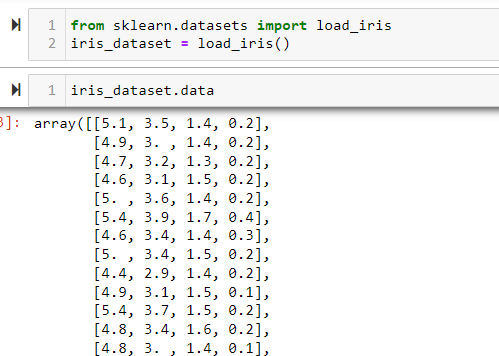
When we give description of data, there can be 3 kinds:

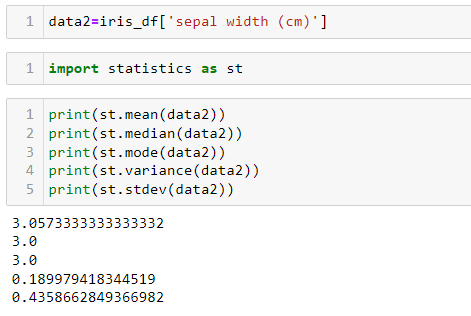
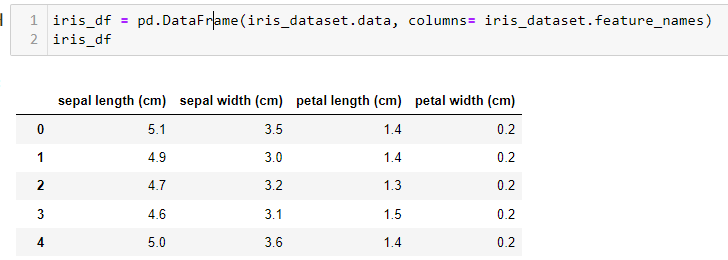
1. Measures of Central Tendency – Mean, Median and Mode
2. Measures of Dispersion – Standard Deviation, Variance, Range, IQR (Inter Quartile Range)
3. Measure of Symmetricity/Shape – Skewness and Kurtosis



**Measure of Central Tendency**







* 1. **a) Implementation of Find-S Algorithm**

**b) Implementation of Candidate elimination Algorithm**

**a) Find-S Algorithm Explanation**

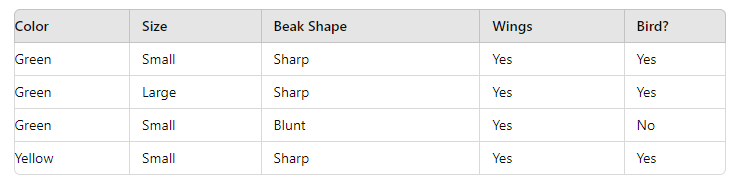
The **Find-S (Find-Specific)** algorithm is a simple algorithm used in machine learning for finding the most specific hypothesis that fits all the positive examples in a given set of data. It is specifically used in the context of learning a conjunctive concept, where the hypothesis space consists of all possible conjunctions of features.

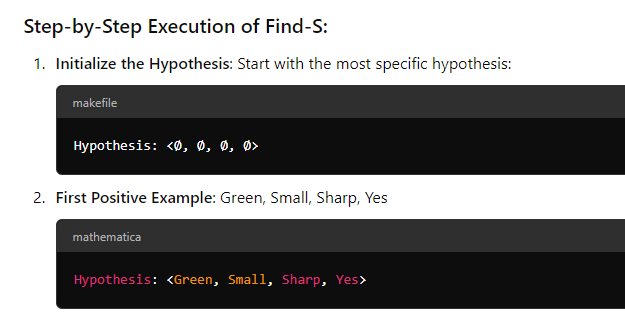
**Steps of the Find-S Algorithm**

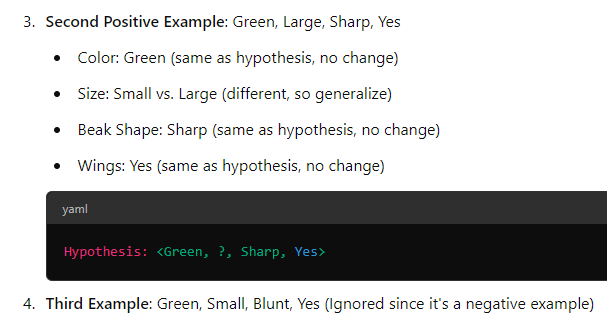
1. **Initialize the Hypothesis**: Start with the most specific hypothesis. In other words, start with a hypothesis that assigns the most specific values to each attribute. This is usually represented by the null vector (i.e., the most specific hypothesis is one where all attributes are null).
2. **For each positive training example**:
   * Compare the hypothesis with the training example.
   * For each attribute:
     + If the attribute in the hypothesis is the same as the attribute in the example, do nothing.
     + If the attribute in the hypothesis is different, generalize the hypothesis by replacing the attribute with the attribute value from the example.
3. **Ignore negative examples**: The Find-S algorithm only considers positive examples when updating the hypothesis.
4. **Output the Hypothesis**: After going through all positive examples, the final hypothesis is the most specific hypothesis that fits all the positive examples.

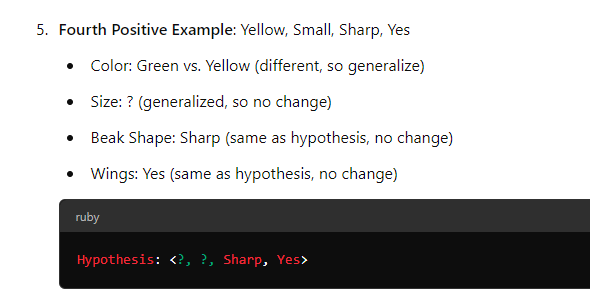
**Example**

Let's consider a simple dataset where we want to determine whether an object is a "Bird" based on four features: color, size, beak shape, and wings.



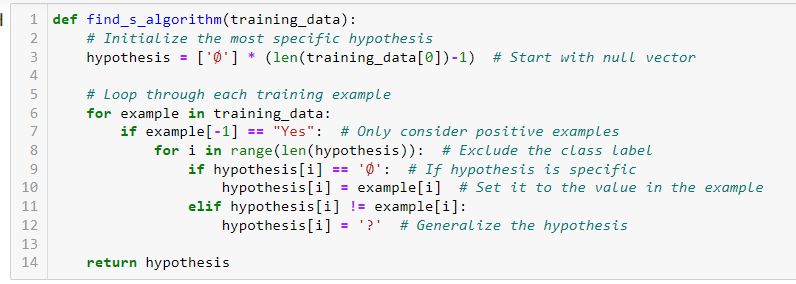


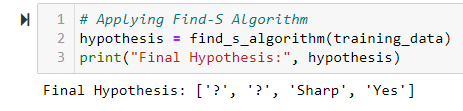






This means the algorithm concludes that any object with a Sharp beak and Wings is likely to be a Bird.

**Python Code Implementation**



**b) Candidate elimination Algorithm**

The candidate elimination algorithm incrementally builds the version space given a hypothesis space H and a set E of examples. The examples are added one by one; each example possibly shrinks the version space by removing the hypotheses that are inconsistent with the example. The candidate elimination algorithm does this by updating the general and specific boundary for each new example.

* You can consider this as an extended form of the Find-S algorithm.
* Consider both positive and negative examples.
* Actually, positive examples are used here as the Find-S algorithm (Basically they are generalizing from the specification).
* While the negative example is specified in the generalizing form.

**Terms Used:**

* **Concept learning:** Concept learning is basically the learning task of the machine (Learn by Train data)
* **General Hypothesis:**Not Specifying features to learn the machine.
* **G = {‘?’, ‘?’,’?’,’?’…}:**Number of attributes
* **Specific Hypothesis:** Specifying features to learn machine (Specific feature)
* **S= {‘**Φ**’,’** Φ**’,’** Φ**’…}: The number** of pi depends on a number of attributes.
* **Version Space:** It is an intermediate of general hypothesis and Specific hypothesis. It not only just writes one hypothesis but a set of all possible hypotheses based on training data-set.

**Step1:** Load Data set

**Step2:** Initialize General Hypothesis and Specific Hypothesis.

**Step3:** For each training example

**Step4:** If example is positive example

if attribute\_value == hypothesis\_value:

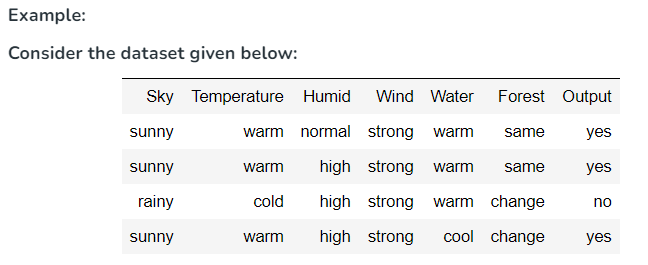
Do nothing

else:

replace attribute value with '?' (Basically generalizing it)

**Step5:** If example is Negative example

Make generalize hypothesis more specific.



**Algorithmic steps:**

**Initially:** G =[[?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?],

[?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?]]

S = [Φ, Φ, Φ, Φ, Φ, Φ]

**For instance 1:** <'sunny','warm','normal','strong','warm ','same'> and positive output.

G1 = G

S1 = ['sunny','warm','normal','strong','warm ','same']

**For instance 2:** <'sunny','warm','high','strong','warm ','same'> and positive output.

G2 = G

S2 = ['sunny','warm',?,'strong','warm ','same']

**For instance 3:** <'rainy','cold','high','strong','warm ','change'> and negative output.

G3 = [['sunny', ?, ?, ?, ?, ?], [?, 'warm', ?, ?, ?, ?], [?, ?, ?, ?, ?, ?],

[?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, ?], [?, ?, ?, ?, ?, 'same']]

S3 = S2

**For instance 4:** <'sunny','warm','high','strong','cool','change'> and positive output.

G4 = G3

S4 = ['sunny','warm',?,'strong', ?, ?]

At last, by synchronizing the G4 and S4 algorithm produce the output.

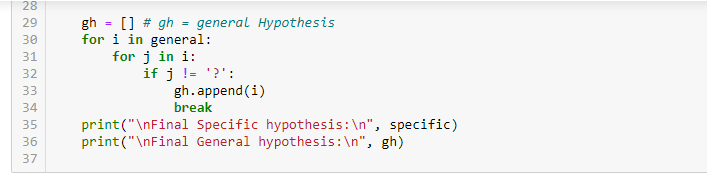
#### Output:

G = [['sunny', ?, ?, ?, ?, ?], [?, 'warm', ?, ?, ?, ?]]

S = ['sunny','warm',?,'strong', ?, ?]







* 1. **Write a Python program to implement Simple Linear Regression and plot the graph**

Linear regression is a statistical technique to describe relationships between dependent variables with a number of independent variables.

### **Simple Linear Regression**

**Simple Linear Regression** is a statistical method to model and analyse the relationship between two continuous variables. It aims to fit a linear equation to the observed data. The linear equation has the form:

consider a dataset in which we have a number of responses **y** per feature **x**:



where:

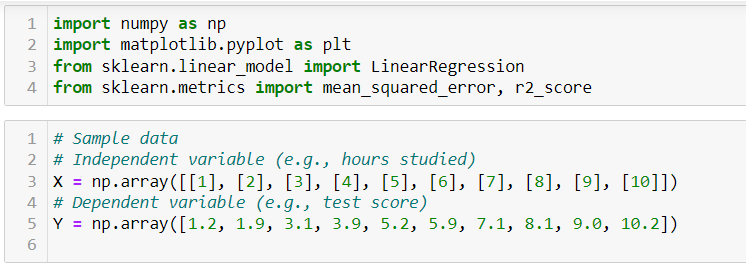
* Y is the dependent variable (response).
* X is the independent variable (predictor).
* β0 is the y-intercept of the line.
* β1​ is the slope of the line.
* ϵ is the error term, representing the difference between the observed and predicted values.

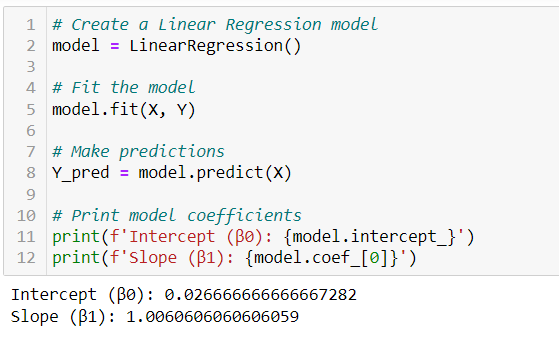
**Steps to perform Simple Linear Regression:**

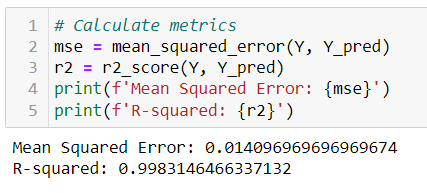
1. **Collect Data**: Gather data that includes both the independent and dependent variables.
2. **Fit the Model**: Determine the values of β0\beta\_0β0​ and β1\beta\_1β1​ using statistical methods such as least squares.
3. **Predict Values**: Use the fitted model to predict the dependent variable values from new data.
4. **Evaluate the Model**: Assess the model's performance using metrics like R-squared or mean squared error.
5. **Plot the Results**: Visualize the data points and the fitted line to understand the relationship between variables.

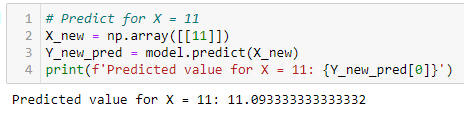
### **Python Code for Simple Linear Regression**

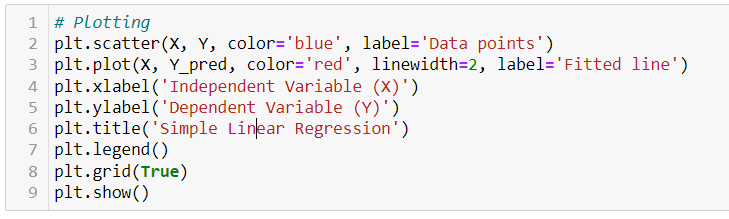
Below is a Python program to implement Simple Linear Regression using scikit-learn and plot the results using matplotlib.

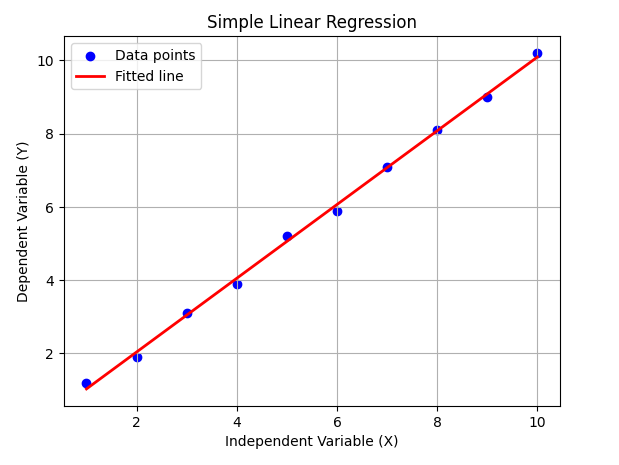












**VIVA QUESTIONS**

### **1. Study of PROLOG Programming Language and its Function**

1. What is the purpose of facts and rules in Prolog?
2. How does Prolog handle logical inferences and backtracking?
3. Explain how Prolog resolves queries using unification.
4. How does Prolog differ from other programming languages like Python or C?
5. What are some real-world applications of Prolog?

### **2. Depth First Search for Water Jug Problem**

1. Explain the state-space representation in the Water Jug problem.
2. Why is Depth First Search (DFS) suitable for solving this problem?
3. How does your DFS algorithm ensure that all possible configurations are explored?
4. Can the Water Jug problem be solved using BFS? How would it differ?
5. What are some potential drawbacks of using DFS in this problem?

### **3. Breadth First Search for Tic-Tac-Toe Problem**

1. How does Breadth First Search (BFS) ensure that the shortest path to a solution is found?
2. What data structures are typically used to implement BFS in Prolog?
3. Explain how BFS handles tie-breaking when multiple solutions exist in the Tic-Tac-Toe problem.
4. What are the differences between BFS and DFS when solving Tic-Tac-Toe?
5. How does your program handle different game outcomes like wins, losses, and draws?

### **4. N-Queens Problem (Prolog)**

1. What is the N-Queens problem, and why is it challenging?
2. How does Prolog handle backtracking when finding solutions for the N-Queens problem?
3. What strategies can be used to reduce the search space in solving the N-Queens problem?
4. How do you represent the chessboard and queens in your Prolog program?
5. Can the N-Queens problem be generalized for any N? How does the complexity change?

### **5. Travelling Salesman Problem (Prolog)**

1. What is the Travelling Salesman Problem (TSP), and why is it considered NP-hard?
2. How does your Prolog program calculate the shortest route?
3. What are the limitations of using a brute-force approach for TSP?
4. How would you improve the performance of your TSP solution?
5. What are some practical applications of the TSP?

### **6. a) Implementation of Python Basic Libraries (Math, Numpy, Scipy)**

1. What are the main uses of the math, numpy, and scipy libraries?
2. How does numpy handle large datasets more efficiently than Python lists?
3. What is the role of scipy in scientific computing?
4. Can you explain how you used numpy to perform matrix operations in your program?
5. How does Python handle floating-point precision in mathematical operations?

### **6. b) Python Libraries for ML Applications (Pandas, Matplotlib)**

1. What is the purpose of the Pandas library in machine learning?
2. How do you load and manipulate datasets using Pandas?
3. What are some common use cases of Matplotlib in data visualization?
4. How can you handle missing data using Pandas?
5. Explain how you plotted graphs using Matplotlib in your program.

### **7. a) Creation and Loading of Datasets in Python**

1. How do you load external datasets into Python using Pandas?
2. What are some common formats of datasets, and how does Python handle them?
3. How does Pandas handle large datasets with limited memory?
4. Explain how you can filter and clean data using Pandas.
5. Can you describe the process of saving a dataset to a file after manipulation?

### **7. b) Computing Mean, Median, Mode, Variance, and Standard Deviation Using Datasets**

1. How do you calculate the mean, median, and mode of a dataset in Python?
2. What is the difference between variance and standard deviation?
3. How does Pandas make statistical calculations simpler?
4. Can you explain how missing values in the dataset affect these calculations?
5. What are some real-world applications of these statistical measures?

### **8. a) Find-S Algorithm**

1. What is the Find-S algorithm, and what type of learning does it represent?
2. How does Find-S find the most specific hypothesis?
3. What are the limitations of the Find-S algorithm?
4. How does Find-S handle inconsistent training data?
5. Can you explain a scenario where the Find-S algorithm might fail to generalize?

### **8. b) Candidate Elimination Algorithm**

1. What is the Candidate Elimination algorithm, and how is it different from Find-S?
2. How does the Candidate Elimination algorithm refine both the general and specific hypotheses?
3. What are the limitations of the Candidate Elimination algorithm?
4. How does Candidate Elimination handle noisy data?
5. Can you give a real-world example where the Candidate Elimination algorithm is applicable?

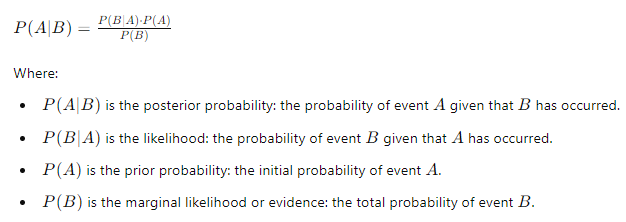
### **9. Simple Linear Regression**

1. What is the purpose of Simple Linear Regression in machine learning?
2. How do you calculate the slope and intercept in a regression model?
3. What assumptions does Simple Linear Regression make about the data?
4. How do you evaluate the accuracy of a regression model?
5. Can you explain how you plotted the regression line in your program?

### **10. Naive Bayes Classifier**

1. What is the Naive Bayes classifier, and how does it work?
2. Why is the assumption of feature independence important in Naive Bayes?
3. What are the advantages and disadvantages of the Naive Bayes algorithm?
4. How does Naive Bayes handle continuous data?
5. Can you explain a real-world application where the Naive Bayes classifier is useful?
   1. **Implementation of Naive bayes classifier**

**Bayes' Theorem:** Bayes' theorem is a fundamental concept in probability theory and statistics. It describes how to update the probability of a hypothesis based on new evidence. The theorem is expressed as:

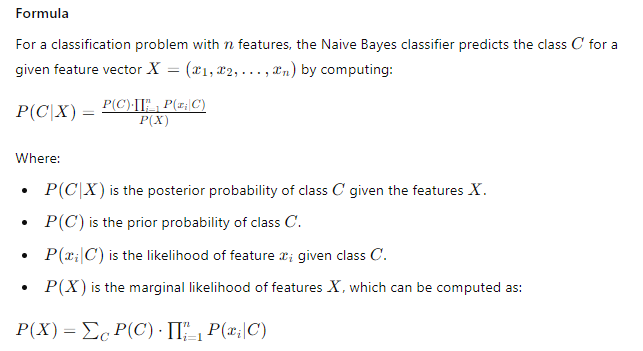


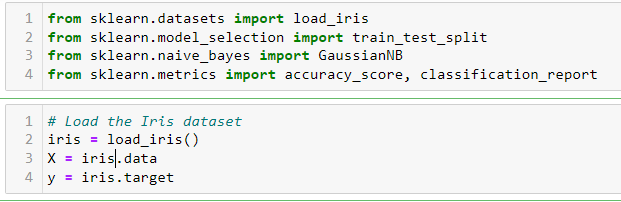
### Naive Bayes Theorem

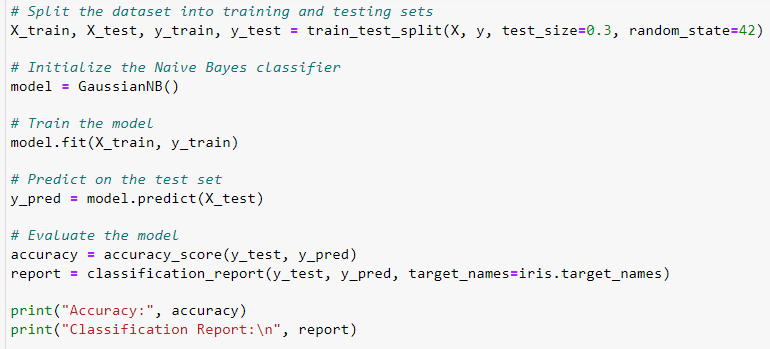
The Naive Bayes theorem is a classification technique based on Bayes' theorem. It assumes that the features (or attributes) of a dataset are conditionally independent given the class label. This assumption is called "naive" because it's often not true in real-world data, but it simplifies the computation and works well in practice.

#### Formula

For a classification problem with n features, the Naive Bayes classifier predicts the class CCC for a given feature vector X=(x1,x2,…,xn) by computing:







OUTPUT

Accuracy: 1.00

Classification Report:

precision recall f1-score support

setosa 1.00 1.00 1.00 10

versicolor 1.00 1.00 1.00 9

virginica 1.00 1.00 1.00 11

accuracy 1.00 30

macro avg 1.00 1.00 1.00 30

weighted avg 1.00 1.00 1.00 30

* here **Precision**: All classes have a precision of 1.00, indicating that every positive prediction made by the model was correct.
* **Recall**: All classes have a recall of 1.00, showing that the model successfully identified all true instances of each class.
* **F1-Score**: An F1-score of 1.00 for each class means the model has a perfect balance between precision and recall.
* **Support**: This indicates the number of true instances for each class (10 for setosa, 9 for versicolor, and 11 for virginica).

Overall, the model shows excellent performance with 100% accuracy.

 **Accuracy (1.00)**: This is the overall accuracy of the model, indicating that it correctly classified all 30 samples (100% accuracy).

 **Macro Average**:

* **Precision**: 1.00
* **Recall**: 1.00
* **F1-Score**: 1.00
* **Support**: 30
* The macro average calculates the average of the metrics for each class without considering class imbalance. In your case, it shows that the model performs perfectly across all classes.

 **Weighted Average**:

* **Precision**: 1.00
* **Recall**: 1.00
* **F1-Score**: 1.00
* **Support**: 30
* The weighted average accounts for the number of instances in each class. Since all classes performed perfectly, the weighted averages are also 1.00.