PYTHON CHEAT SHEET

Basic Commands:

- **print("Hello, World!")**: Display output
- **type(x)**: Get data type of x
- **id(x)**: Get memory address of x
- **help()**: Show documentation
- **dir(obj)**: List object attributes and methods

Conditional Statements

- **for**: Iterates over a sequence.
- **while**: Runs while a condition is true.
- **break**: Exits the loop.
- **continue**: Skips to the next iteration.
- **pass**: Does nothing (placeholder).

File Handling

- **open("file.txt", "r")**: Read file
- **open("file.txt", "w")**: Write file
- **open("file.txt", "a")**: Append file
- **file.read()**: Read file content
- **file.readline()**: Read one line
- **file.write("text")**: Write to file
- **file.close()**: Close file
- Use **with open("file.txt", "w") as f:** for safe handling

List Methods

- **append(x)**: Add element to end
- **insert(i, x)**: Insert at index i
- **remove(x)**: Remove first occurrence of x
- **pop(i)**: Remove element at index i
- **reverse()**: Reverse list order
- **sort()**: Sort list

Variables & Data Types:

- **int**: Whole numbers.
- **float**: Decimal numbers.
- **str**: Text values.
- **bool**: True or False values.
- **list**: Ordered, changeable collection.
- **tuple**: Ordered, unchangeable collection.
- **set**: Unordered, unique collection.
- **dict**: Key-value pair collection.

Functions

- **def**: Define a new function
- **return**: Return value from a function
- **lambda**: Create a anonymous function

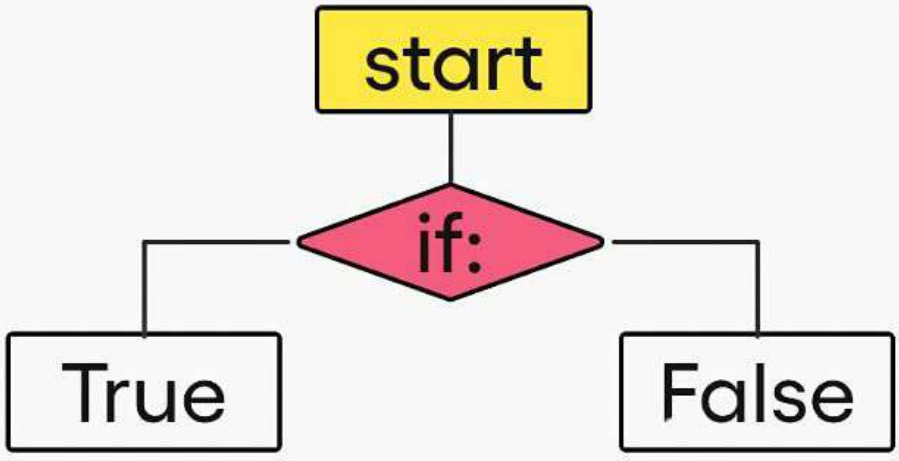
Built-in Functions

- **len(x)**: Length of x
- **max(x)**: Maximum value in x
- **min(x)**: Minimum value in x
- **sum(x)**: Sum of elements
- **sorted(x)**: Sorted list
- **range(start, stop, step)**: Generate sequence
- **map(func, iterable)**: Apply function to elements
- **filter(func, iterable)**: Filter elements
- **zip(iter1, iter2)**: Combine iterables

DATAPATASHALA
YOUR ANALYTICS GURU

PYTHON

STATEMENTS



datapatashala_official
datapatashala
learn.datapatashala.in
Info@datapatashala.in

DATAPATASHALA
YOUR ANALYTICS GURU

What is a Statement in Python

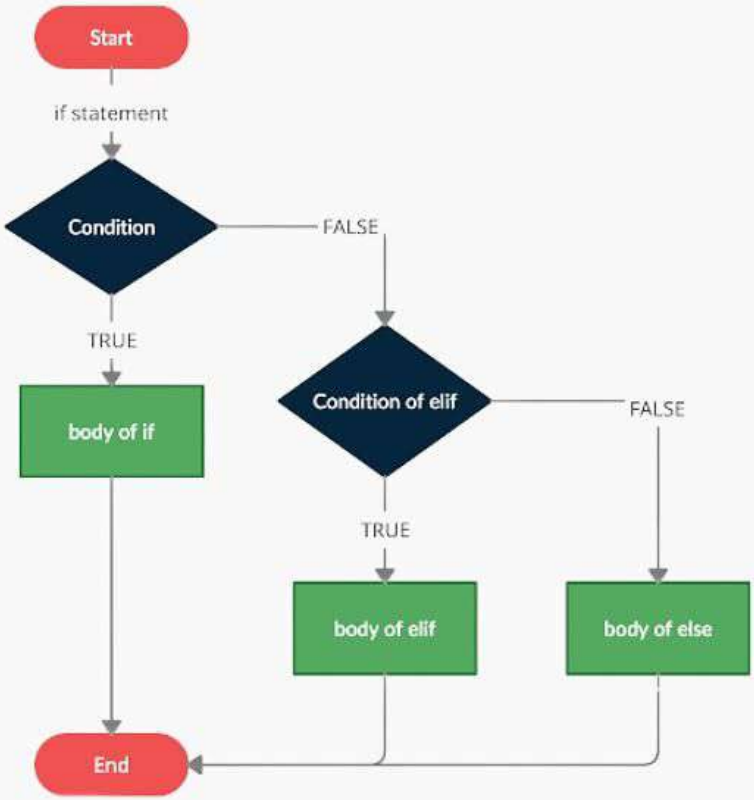
A statement is an instruction that a Python interpreter can execute. So, in simple words, we can say anything written in Python is a statement.

There are mainly four types of statements in Python, **print statements**, **Assignment statements**, **Conditional statements**, **Looping statements.**

DATAPATASHALA
YOUR ANALYTICS GURU

Selection Statements

These statements are used to select whether code is executed depending on a condition.



DATAPATASHALA
YOUR ANALYTICS GURU

if:

This construct only executes the code statements in the group if a condition has been met.

```
if condition == true:
  #execute This
```

else:

This construct only executes if the condition in the if-statement has not been met.

```
if condition == True:
  #execute this
else:
  #else execute this
```

elif:

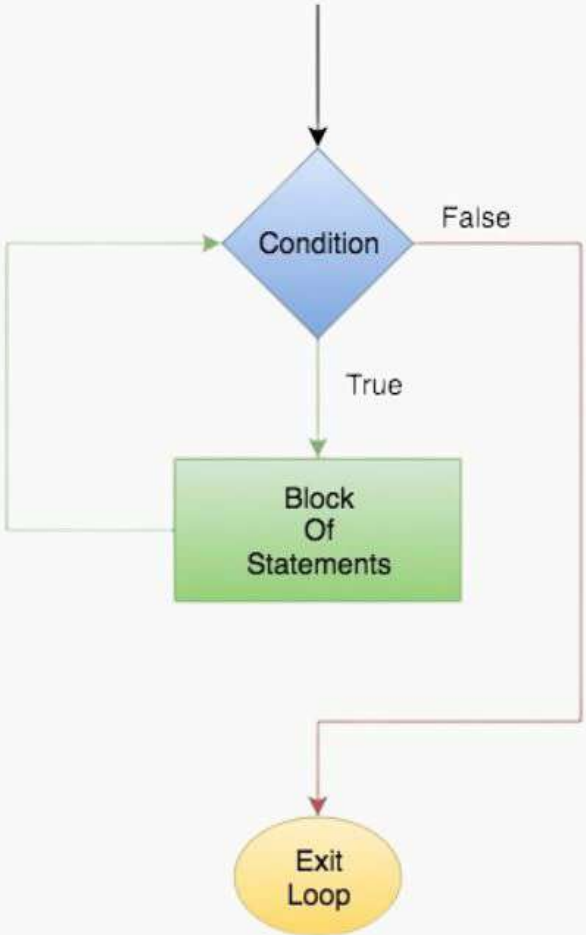
This construct allows us to provide additional conditions to an if statement to execute different codes.

```
if condition 1 == True:
  # execute this
elif condition 2 =- True:
  #execute this
```

DATAPATASHALA
YOUR ANALYTICS GURU

Iteration Statements:

These statements are used to repeat a group of instructions for a set number of times until a condition is met.



DATAPATASHALA
YOUR ANALYTICS GURU

for:

For loop is used for sequential traversal i.e. it is used for iterating over an iterable like String, Tuple, List, etc.

```
for i in range(1,11):
  print(i)
```

while:

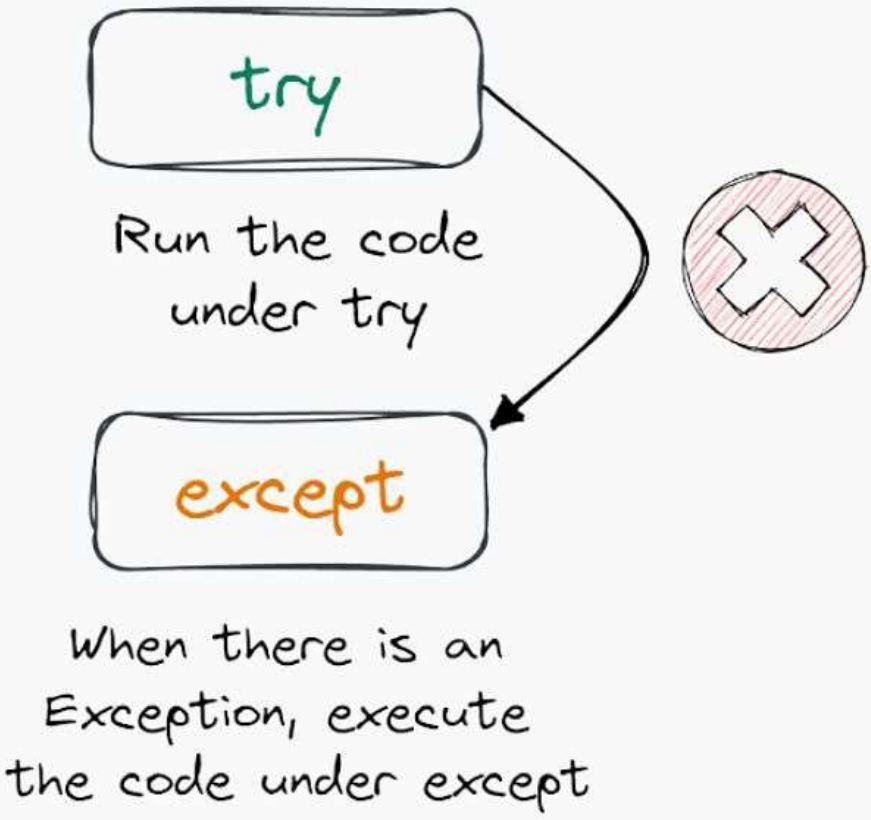
This condition-controlled loop executes its code if a condition is met. After reaching the last statement it checks the condition before starting again

```
while condition == True:
  # execute this
```

DATAPATASHALA
YOUR ANALYTICS GURU

Exception Handling Statements:

These statements allow hose code instructions that may cause errors and kill the program.



DATAPATASHALA
YOUR ANALYTICS GURU

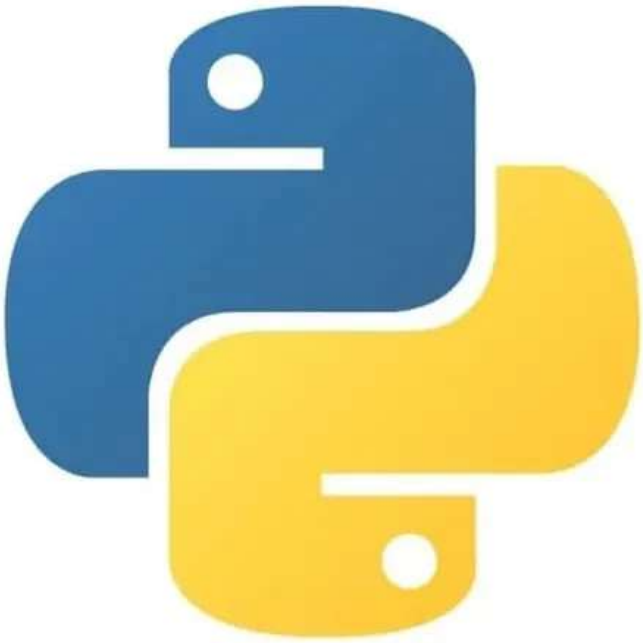
try:

The try block is used to check some code for errors ie the code inside the try block will execute when there is no error in the program.

except:

The code inside the except block will execute whenever the program encounters some error in the preceding try block.

```
try:
  100/0 #Error
except:
  print("Division with zero
error!")
```



Local vs Global

Variable in Python

Local Variables:-

A local variable is a variable that you create inside a function.

Scope: It can only be used inside that function. You can't use it outside the function.

Lifetime: It exists only while the function is running. Once the function is done, the local variable is gone.

For Example:

```
def my_function():
    local_var = 10  # This is a local variable
    print(local_var)

my_function()  # This will print 10
print(local_var)
# This will cause an error because local_var is not
accessible outside the function
```

Global Variables:-

A global variable is a variable that you create outside of all functions.

Scope: It can be used anywhere in your code, inside or outside of functions.

Lifetime: It exists as long as your program is running.

For Example:

```
global_var = 20  # This is a global variable

def my_function():
    print(global_var)  # This will print 20 because
global_var is accessible here

my_function()
print(global_var)  # This will also print 20
```

Key Differences:-

1. Where They're Defined:
   - Local: Inside a function.
   - Global: Outside of all functions.

2. Where You Can Use Them:
   - Local: Only inside the function where it's defined.
   - Global: Anywhere in the code.

3. How Long They Last:
   - Local: Only while the function is running.
   - Global: As long as the program is running.

Important Note:-

If you need to modify a global variable inside a function, you must use the global keyword.

For Example:-

```
global_var = 20

def my_function():
    global global_var
    global_var = 30  # This changes the global variable
    print(global_var)

my_function()  # This will print 30
print(global_var)  # This will also print 30 because the
global variable was changed inside the function
```

PYTHON CHEAT SHEET

Basic Commands:

- **print("Hello, World!")**: Display output
- **type(x)**: Get data type of x
- **id(x)**: Get memory address of x
- **help()**: Show documentation
- **dir(obj)**: List object attributes and methods

Conditional Statements

- **for**: Iterates over a sequence.
- **while**: Runs while a condition is true.
- **break**: Exits the loop.
- **continue**: Skips to the next iteration.
- **pass**: Does nothing (placeholder).

File Handling

- **open("file.txt", "r")**: Read file
- **open("file.txt", "w")**: Write file
- **open("file.txt", "a")**: Append file
- **file.read()**: Read file content
- **file.readline()**: Read one line
- **file.write("text")**: Write to file
- **file.close()**: Close file
- Use **with open("file.txt", "w") as f:** for safe handling

List Methods

- **append(x)**: Add element to end
- **insert(i, x)**: Insert at index i
- **remove(x)**: Remove first occurrence of x
- **pop(i)**: Remove element at index i
- **reverse()**: Reverse list order
- **sort()**: Sort list

Variables & Data Types:

- **int**: Whole numbers.
- **float**: Decimal numbers.
- **str**: Text values.
- **bool**: True or False values.
- **list**: Ordered, changeable collection.
- **tuple**: Ordered, unchangeable collection.
- **set**: Unordered, unique collection.
- **dict**: Key-value pair collection.

Functions

- **def**: Define a new function
- **return**: Return value from a function
- **lambda**: Create a anonymous function

Built-in Functions

- **len(x)**: Length of x
- **max(x)**: Maximum value in x
- **min(x)**: Minimum value in x
- **sum(x)**: Sum of elements
- **sorted(x)**: Sorted list
- **range(start, stop, step)**: Generate sequence
- **map(func, iterable)**: Apply function to elements
- **filter(func, iterable)**: Filter elements
- **zip(iter1, iter2)**: Combine iterables

String Methods

- str.lower() - Converts to lowercase
- str.upper() - Converts to uppercase
- str.capitalize() - Capitalizes the first letter
- str.title() - Capitalizes the first letter of each word
- str.strip() - Removes leading/trailing spaces
- str.split() - Splits a string into a list

String Methods

- str.join(iterable) - Joins elements of an iterable into a single string
- str.replace(old, new) - Replaces occurrences of a substring
- str.find(substring) - Returns the index of the first occurrence of a substring, or -1 if not found
- str.index(substring) - Like find(), but raises an error if the substring is not found

List Methods

- list.append(item) - Adds an item to the end
- list.extend(iterable) - Extends the list
- list.insert(index, item) - Inserts at a position
- list.remove(item) - Removes first occurrence
- list.pop(index) - Removes & returns an item
- list.sort() - Sorts the list
- list.reverse() - Reverses order

Lists are powerful for handling collections of data

Dictionary Methods

- dict.get(key) - Retrieves value safely
- dict.keys() - Returns all keys
- dict.values() - Returns all values
- dict.items() - Returns key-value pairs
- dict.pop(key) - Removes a key
- dict.update(other_dict) - Merges dictionaries

Set Methods

- set.add(item) - Adds an item
- set.remove(item) - Removes an item (error if missing)
- set.discard(item) - Removes an item (no error if missing)
- set.union(other_set) - Combines two sets
- set.intersection(other_set) - Finds common elements
- set.difference(other_set) - Finds unique elements

Tuple Methods

- `tuple.count(item)` - Counts occurrences
- `tuple.index(item)` - Finds first occurrence

Tuples are immutable - great for fixed collections

File Handling

- file.read() - Reads file content
- file.readline() - Reads one line
- file.readlines() - Reads all lines as a list
- file.write(string) - Writes a string to a file
- file.seek(offset) - Moves file pointer
- file.tell() - Returns current position

Essential Built-in Functions

- len(object) - Returns length
- max(iterable) - Finds the max value
- min(iterable) - Finds the min value
- sum(iterable) - Returns the sum
- sorted(iterable) - Sorts the elements
- enumerate(iterable) - Returns index-value pairs
- zip(iter1, iter2) - Combines iterables

Programming

- map(func, iterable) - Applies function to iterable
- filter(func, iterable) - Filters elements
- all(iterable) - Checks if all elements are True
- any(iterable) - Checks if any element is True
- range(start, stop, step) - Generates a number sequence

Functional programming saves time & effort

Lambda Functions

- `lambda x: x + 2` - Simple anonymous function
- `lambda x, y: x * y` - Multiply two numbers

Lambdas = Quick & powerful one-liners in Python!

Introduction

Matplotlib is a Python library for data visualization. It helps you create graphs, charts, and plots to understand data better.

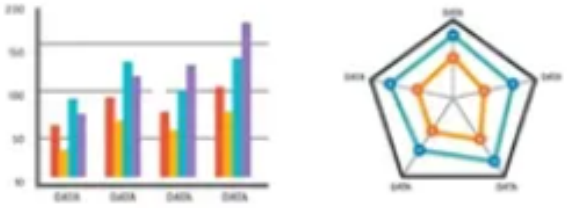
Think of it as a graphing tool for Python—just like how Excel creates charts, but with more flexibility and control.

With Matplotlib, you can make:

- ✅ Line Plots
- ✅ Bar Charts
- ✅ Scatter Plots
- ✅ Histograms & more!

It's widely used in ***Data Science*** & ***Machine Learning*** to visualize trends and patterns in data.

matplotlib



Akash • Python & Tech Enthusiast 🚀
@pycode.hubb

Installing Matplotlib

Before we start plotting, let's get Matplotlib installed! It's super easy:

👉***Open your terminal or CMD & type:***

```
pip install matplotlib
```

✅ ***That's it!*** *Now you're ready to create awesome visualizations in Python.*

💡 ***Tip:*** *If you're using Jupyter Notebook, also run:*

```
%matplotlib inline
```

This makes your plots show up directly in the notebook!

Basic Plot with pyplot

Let's create our first simple plot using Matplotlib!

👉 ***First, import Matplotlib's pyplot module:***

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

👉 ***Now, plot some data:***

```
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 40]

plt.plot(x, y)  # Create the line plot
plt.show()  # Display the plot
```

✅ ***What this does:***

- *plt.plot(x, y)*: Plots the values.
- *plt.show()*: Displays the graph.

Try changing the ***y values*** and see how the plot changes!

Customizing Plots

Let's make our plots look better by adding labels, titles, and colors!

👉 ***Adding Labels & Title:***

```python
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 40]

plt.plot(x, y)

# Customization
plt.xlabel("X Axis")  # Label for X-axis
plt.ylabel("Y Axis")  # Label for Y-axis
plt.title("My First Custom Plot")  # Title

plt.show()
```

👉 ***Changing Line Style & Color:***

```python
plt.plot(x, y, color="red", linestyle="--", marker="o")
```

👉 color="red" → Changes the line color.

👉 linestyle="--" → Makes the line dashed.

👉 marker="o" → Adds circular markers.

Different Types of Plots

Matplotlib isn't just for line plots! Let's explore some common plot types.

1 ***Line Plot (Default Plot):*** *to show trends over time.*

```
plt.plot(x, y, color="blue", marker="o")
plt.title("Line Plot")
plt.show()
```

2 ***Scatter Plot:*** *for visualizing relationships between two variables.*

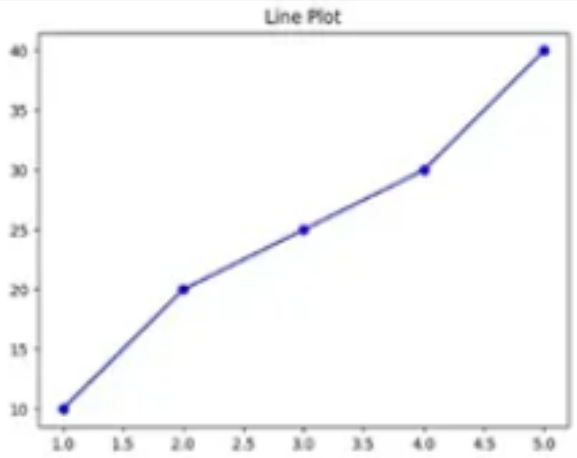
```
plt.scatter(x, y, color="red")
plt.title("Scatter Plot")
plt.show()
```

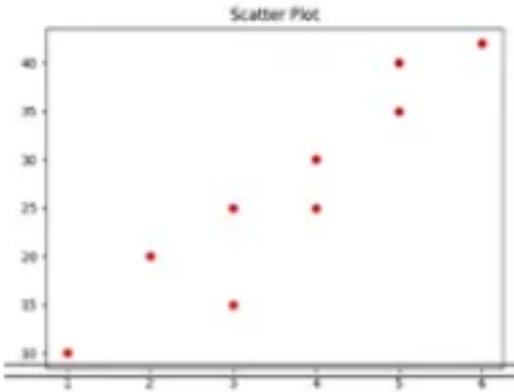
3 ***Bar Chart:*** *Great for comparing categories.*

```
plt.bar(x, y, color="green")
plt.title("Bar Chart")
plt.show()
```

Different Types of Plots

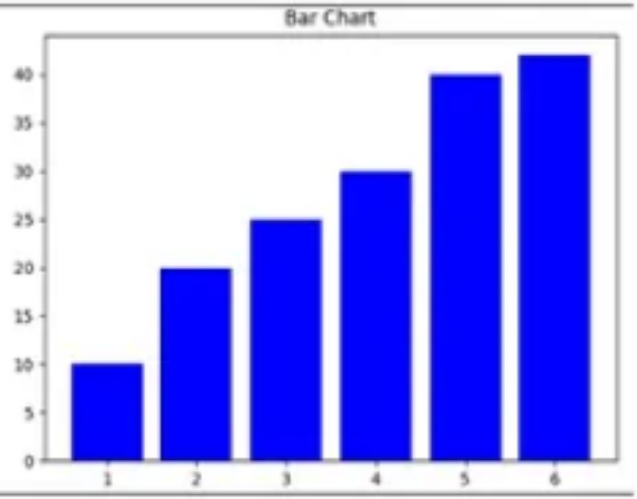
1 Line Plot





2 Scatter Plot

3 Bar Chart



Libraries For Data Science In Python

1. **NumPy:** Supports large arrays, matrices; provides mathematical functions for array operations in Python.
2. **Pandas:** Manipulate, clean, analyze structured data effortlessly with DataFrames and Series in Python.
3. **Matplotlib:** 2D plotting library for static, animated, interactive visualizations; widely used in Python.
4. **Seaborn:** Enhances Matplotlib; simplifies data visualization with high-level interface and statistical graphics.
5. **Scipy:** Scientific and technical computing; built on NumPy, offers optimization, integration, interpolation functions.
6. **Scikit-learn:** Efficient machine learning tools; includes algorithms for classification, regression, clustering, dimensionality reduction.
7. **Statsmodels:** Focuses on estimating/testing statistical models like linear regression, generalized linear models.
8. **TensorFlow:** Leading deep learning frameworks for building, training neural networks in Python.
9. **Spacy:** Libraries for natural language processing; tasks like tokenization, stemming, tagging, parsing.
10. **Jupyter:** Interactive computing environments for creating and sharing documents with live