Variable

In [2]:
```
name ="VaibhaV"
```

In [3]:
```
print(name)
```

```
VaibhaV
```

Object Types / Data Types

- Number : 1234, 3.1415, 3+4j, 0b111, Decimal(), Fraction()
- String : 'spam', "Bob's", b'a\x01c', u'sp\xc4m'
- List : [1, [2, 'three'], 4.5], list(range(10))
- Tuple : (1, 'spam', 4, 'U'), tuple('spam'), namedtuple
- Dictionary : {'food': 'spam', 'taste': 'yum'}, dict(hours=10)
- Set : set('abc'), {'a', 'b', 'c'}
- File : open('eggs.txt'), open(r'C:\ham.bin', 'wb')
- Boolean : True, False
- None : None
- Funtions, modules, classes
- Advance: Decorators, Generators, Iterators, MetaProgramming

Data type

- int
- float
- Boolean
- String

In [4]:
```
num1= 1
print(type(num1))
num2= 2.303
print(type(num2))
num3= True
print(type(num3))
num4= "VaibhaV"
print(type(num4))
```

```
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'str'>
```

Operators

Arithmatic Operator +, -, *, /

In [5]:
```
num1, num2 = 36, 7
```

In [6]:
```
num1 + num2
```

Out[6]: 43

In [7]:
```
num1 - num2
```

Out[7]: 29

In [8]:
```
num1 * num2
```

Out[8]: 252

In [9]:
```
num1 / num2
```

Out[9]: 5.142857142857143

In [10]:
```
num1 // num2
```

Out[10]: 5

Relational Operator <, >, ==, !=

In [11]:
```
num1 > num2
```

Out[11]: True

In [12]:
```
num1 < num2
```

Out[12]: False

In [13]:
```
num1 == num2
```

Out[13]: False

In [14]:
```
num1 != num2
```

Out[14]: True

Logical Operator & |

In [15]:
```
log1 = True
log2 = False
```

In [16]:
```
log1 & log2
```

Out[16]: False

In [17]:
```
log1 & log1
```

Out[17]: True

In [18]:
```
log1 | log1
```

Out[18]:
```
True
```

Python Tokens

Smallest meaningfull componet in program

- Keywords : Special reserved words like For If Yield
- Identifiers: Names used for variables, functions or objects
- Literals : Costants in python
  - num = "Test" : here "num" is Identifier and "Test" is literal
- Operators

String:

In [19]:
```
str1 = "I love Pizza"
str1.find("Piz")
```

Out[19]:
```
7
```

In [20]:
```
fruits ="Mango, Banana, Apple"
fruits.split(", ")
```

Out[20]:
```
['Mango', 'Banana', 'Apple']
```

List

In Python, a list is a built-in data structure that allows you to store an ordered collection of items. These items can be of any data type, including integers, strings, floats, or even other lists. Lists are mutable, meaning you can change their content without changing their identity.

Here is a basic definition and example of a list in Python:

Definition:A list is defined by placing a comma-separated sequence of items within square brackets [].

Key Characteristics:

- Ordered: The items have a defined order, and that order will not change unless you explicitly modify the list.
- Mutable: You can change, add, or remove items after the list has been created.
- Heterogeneous: A list can contain items of different data types.

Common Operations:

- Accessing Elements: Use indexing to access individual elements.
- Slicing: Use slicing to access a range of elements.

- Appending: Use append() to add an element to the end of the list.
- Inserting: Use insert() to add an element at a specific position.
- Removing: Use remove() to remove a specific element.
- Popping: Use pop() to remove an element at a specific position and return it.
- Length: Use len() to get the number of elements in the list.

In [21]:
```
l1 = [1, "a", 2, True]
```

In [22]:
```
l1[0]=100
l1
```

Out[22]:
```
[100, 'a', 2, True]
```

In [23]:
```
l1.append("VaibhaV")
l1
```

Out[23]:
```
[100, 'a', 2, True, 'VaibhaV']
```

In [24]:
```
l1.pop()
```

Out[24]:
```
'VaibhaV'
```

In [25]:
```
l1
```

Out[25]:
```
[100, 'a', 2, True]
```

In [26]:
```
l1.reverse()
l1
```

Out[26]:
```
[True, 2, 'a', 100]
```

In [27]:
```
l1.insert(1, "Test")
l1
```

Out[27]:
```
[True, 'Test', 2, 'a', 100]
```

In [28]:
```
l1.sort()
l1
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[28], line 1
----> 1 l1.sort()
      2 l1

TypeError: '<' not supported between instances of 'str' and 'bool'
```

In [1]:
```
l2 = ['Mango', 'Banana', 'Apple']
l2.sort()
l2
```

Out[1]:
```
['Apple', 'Banana', 'Mango']
```

In [29]:
```
l1 = ["a", "b", "c"]
l2 = [1, 2, 3]
l1+ l2
```

Out[29]:
```
['a', 'b', 'c', 1, 2, 3]
```

In [30]:
```
l1 = [1, "b", True]
l1 * 3
```

Out[30]:
```
[1, 'b', True, 1, 'b', True, 1, 'b', True]
```

Tuple

- Tuple are ordered collection of elements enclosed within ()
- Tuples are immutable, once created we can not update or change values of tuple

In [31]:
```
tup1 = (1, "b", True)
tup1
```

Out[31]:
```
(1, 'b', True)
```

In [32]:
```
type(tup1)
```

Out[32]:
```
tuple
```

In [33]:
```
tup2 = tup1 *3
tup2
```

Out[33]:
```
(1, 'b', True, 1, 'b', True, 1, 'b', True)
```

In [34]:
```
tup2[1:6]
```

Out[34]:
```
('b', True, 1, 'b', True)
```

In [35]:
```
tup2[::2]
```

Out[35]:
```
(1, True, 'b', 1, True)
```

In [36]:
```
tup2[2]="Test"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[36], line 1
----> 1 tup2[2]="Test"

TypeError: 'tuple' object does not support item assignment
```

In [37]:
```
tup1 = ("a", "b", "c")
tup2 = (1, 2, 3)
tup1 + tup2
```

Out[37]:
```
('a', 'b', 'c', 1, 2, 3)
```

Dictionary

- Dictionary is an unordered collection of key-value pairs enclosed with {}
- Dictionary is mutable

In [38]:
```
fruit = {"Mango": 10, 'Banana': 20, 'Apple':30}
```

In [39]:
```
fruit.keys()
```

Out[39]:
```
dict_keys(['Mango', 'Banana', 'Apple'])
```

In [40]:
```
fruit.values()
```

Out[40]:
```
dict_values([10, 20, 30])
```

In [41]:
```
fruit["Guava"]=40
fruit
```

Out[41]:
```
{'Mango': 10, 'Banana': 20, 'Apple': 30, 'Guava': 40}
```

In [42]:
```
fruit
```

Out[42]:
```
{'Mango': 10, 'Banana': 20, 'Apple': 30, 'Guava': 40}
```

In [43]:
```
f1 = {'Mango': 10, 'Banana': 20}
f2 = {'Apple': 30, 'Guava': 40}
```

In [44]:
```
f1.update(f2)
f1
```

Out[44]:
```
{'Mango': 10, 'Banana': 20, 'Apple': 30, 'Guava': 40}
```

In [45]:
```
f1.pop("Guava")
```

Out[45]:
```
40
```

In [46]:
```
f1
```

Out[46]:
```
{'Mango': 10, 'Banana': 20, 'Apple': 30}
```

Set

- Set ia a unordered and unindexed collection of elements enclosed with {}
- Duplicates are not allowed in Set

In [52]:
```
# Try duplicates with set
s1 = {"c", "a", "b", 1, 2, 3, "a", "b", 1, 2}
s1
```

Out[52]:
```
{1, 2, 3, 'a', 'b', 'c'}
```

In [53]:
```
s1.add("Test")
s1
```

Out[53]: {1, 2, 3, 'Test', 'a', 'b', 'c'}

In [54]:
```
s1.remove("b")
s1
```

Out[54]: {1, 2, 3, 'Test', 'a', 'c'}

In [55]:
```
s1 = {'Mango', 'Banana', 'Apple', 1, 2, 'a'}
s2 = {'a', 'b', 'c', 1, 2}
s1.union(s2)
```

Out[55]: {1, 2, 'Apple', 'Banana', 'Mango', 'a', 'b', 'c'}

In [56]:
```
s1
```

Out[56]: {1, 2, 'Apple', 'Banana', 'Mango', 'a'}

In [57]:
```
s2
```

Out[57]: {1, 2, 'a', 'b', 'c'}

In [58]:
```
s1.intersection(s2)
```

Out[58]: {1, 2, 'a'}

In [59]:
```
s1.update(s2)
s1
```

Out[59]: {1, 2, 'Apple', 'Banana', 'Mango', 'a', 'b', 'c'}

Decision making / Conditions

In [60]:
```
a, b= 10, 20
```

In [61]:
```
if a>b:
    print("A is greater than B")
else:
    print("B is greater than A")
```

B is greater than A

In [62]:
```
result= "A is greater than B" if a>b else "B is greater than A"
result
```

Out[62]: 'B is greater than A'

In [63]:
```
tup= ("a", "b", "c")
```

In [63]:
```
if "z" in tup:
    print("z is in tup")
else:
    print("z is not in tup")
```

```
z is not in tup
```

In [64]:
```
"z is in tup" if "z" in tup else "z is not in tup"
```

Out[64]:
```
'z is not in tup'
```

Looping Statement

To repeat task multiple times

In [65]:
```
fruit = ['Mango', 'Banana', 'Apple']
for frt in fruit:
    print(frt)
```

```
Mango
Banana
Apple
```

In [66]:
```
i= 1
while i<=10:
    print(i)
    i+=1
```

```
1
2
3
4
5
6
7
8
9
10
```

Object Oriented Programming

- Class: user defined data type
  - Attributes
  - Methods
- Objects : Specific instance of a class

In [67]:
```
class Phone:
    def make_call(self):
        print("Making phone call")
    def play_game(self):
        print("Playing game")
    def set_color(self, color):
        self.color = color
    def set_cost(self, cost):
        self.cost = cost
    def show_color(self):
        return self.color
    def show_cost(self):
        return self.cost
```

In [68]:
```
p1 = Phone()
```

In [69]:
```
p1.make_call()
```

```
Making phone call
```

In [70]:
```
p1.play_game()
```

```
Playing game
```

In [71]:
```
p1.set_color("Red")
```

In [72]:
```
p1.set_cost(1000)
```

In [73]:
```
p1.show_color()
```

Out[73]:
```
'Red'
```

In [74]:
```
p1.show_cost()
```

Out[74]:
```
1000
```

Constructor

In [75]:
```
class Phone:
    def __init__(self, color, cost):
        self.color = color
        self.cost = cost
    def make_call(self):
        print("Making phone call")
    def play_game(self):
        print("Playing game")
```

In [76]:
```
p1 = Phone("Red", 2000)
```

In [77]:
```
p1.color
```

Out[77]:
```
'Red'
```

Inheritance

With inheritance one class can derive the properties of another class

In [78]:
```
class Vehicle:
    def __init__(self, mileage, cost):
        self.mileage = mileage
        self.cost = cost

    def vehicle_details(self):
        print(f"Vehicle has {self.mileage} mileage and it cost ${self.cos
```

In [79]:
```
v1 = Vehicle(27, 15000)
v1.vehicle_details()
```

```
Vehicle has 27 mileage and it cost $15000
```

In [80]:
```
class Car(Vehicle):
    def show_car(self):
        print("I am Car")
```

In [81]:
```
c1 = Car(200, 15000)
```

In [82]:
```
c1.show_car()
```

```
I am Car
```

In [83]:
```
c1.vehicle_details()
```

```
Vehicle has 200 mileage and it cost $15000
```

In [84]:
```
class Car(Vehicle):
    def __init__(self, mileage, cost, hp ):
        super().__init__(mileage, cost)
        self.hp = hp
    def show_car(self):
        print(f"I am car with mileage: {self.mileage}, cost: {self.cost},
```

In [85]:
```
c1= Car(100, 15000, 2000)
```

In [86]:
```
c1.show_car()
```

```
I am car with mileage: 100, cost: 15000, power: 2000
```

In [87]:
```
c1.vehicle_details()
```

```
Vehicle has 100 mileage and it cost $15000
```

Multiple Inheretance

Child inherite more than one parent class

In [88]:
```
class Parent1:
    def assign_string_1(self, name):
        self.name1 = name
    def show_string_1(self):
        return self.name1

class Parent2:
    def assign_string_2(self, name):
        self.name2 = name
    def show_string_2(self):
        return self.name2

class Child(Parent1, Parent2):
    def assign_string_3(self, name):
        self.name3 = name
    def show_string_3(self):
        return self.name3
```

In [89]:
```
c1 = Child()
c1.assign_string_1("One")
```

```
c1.assign_string_2("Two")
c1.assign_string_3("Three")
```

In [90]:
```
print(c1.show_string_1())
print(c1.show_string_2())
print(c1.show_string_3())
```

```
One
Two
Three
```

Multi Level Inheritance

-We have Parent, Child, grand-child

In [91]:
```
class Parent1:
    def assign_name(self, name):
        self.name = name
    def show_name(self):
        return self.name

class Child(Parent1):
    def assign_age(self, age):
        self.age = age
    def show_age(self):
        return self.age

class GrandChild(Child):
    def assign_gender(self, gender):
        self.gender = gender
    def show_gender(self):
        return self.gender
```

In [92]:
```
gc = GrandChild()
```

In [93]:
```
gc.assign_name("Test")
```

In [94]:
```
gc.assign_age(21)
```

In [95]:
```
gc.assign_gender("Male")
```

In [96]:
```
print(f"{gc.show_name()} is a {gc.show_gender()} aged {gc.show_age()}")
```

```
Test is a Male aged 21
```

File Handling

- Open mode : Open text file for reading, writing and doing some other stuff
- Read mode : To read the text which is already stored in your text file
- Write mode : To write your text in .txt file

In [97]:
```
file = open('processed_data/test.txt', 'w')

try:
```

```
    file.write('Chai aur code line 1')
finally:
    file.close()
```

In [98]:
```
file = open('processed_data/test.txt', 'a')

try:
    file.write('\nChai aur code line 2')
finally:
    file.close()
```

In [100…
```
file = open('processed_data/test.txt', 'r')
data_read = file.read()
print(data_read)
total_count = len(data_read)
print(total_count)
```

```
Chai aur code line 1
Chai aur code line 2
41
```

Readline functions

In [108…
```
f = open('processed_data/test.txt', 'w')
f.write("I am learning file handling\n")
f.write("Topic is file handling, read and write")
```

Out[108… 38

In [109…
```
f = open('processed_data/test.txt', 'r')
```

In [110…
```
print(f.readline())
```

```
I am learning file handling
```

In [120…
```
# Using 'with' to handle file operations
file_path = 'processed_data/example.txt'

# Writing to a file
with open(file_path, 'w') as file:
    file.write("Hello, this is a file handling example using 'with'.\n")
    file.write("This automatically manages the file resource.\n")

# Reading from a file
with open(file_path, 'r') as file:
    content = file.read()

# Output the content
print("File Content:\n")
print(content)
```

```
File Content:

Hello, this is a file handling example using 'with'.
This automatically manages the file resource.
```

In [111…
```
print(f.readline())
```

```
Topic is file handling, read and write
```

In [112…
```
print(f.readline())
```

Try Except

In [113…
```
a =input("Enter the number 1 : ")
b =input("Enter the number 2 : ")

try:
    c= int(a) + b
    print(c)
except Exception as e:
    print(e)
```

```
unsupported operand type(s) for +: 'int' and 'str'
```

In [121…
```
try:
    c= int(a) + b
    print(c)
except:
    print("Error in try block")
```

```
Error in try block
```

Try with else clause

Else clause is used with the try clause when you want to execute the set of instructions in the absence of exceptions in your code

In [115…
```
a =input("Enter the number 1 : ")
b =input("Enter the number 2 : ")

try:
    c= int(a) + int(b)
    print(c)
except Exception as e:
    print(e)
else:
    print("All inputs are good, Else clause got executed!!")
```

```
3
All inputs are good, Else clause got executed!!
```

Try Finally keyword

Finally is a keyword which surely executes after the execution of the try except block of statement

In [116…
```
a =input("Enter the number 1 : ")
b =input("Enter the number 2 : ")
```

```
try:
    c= int(a) + b
    print(c)
except:
    print("Error in try block")
finally:
    print("Finally runs post try except")
```

```
Error in try block
Finally runs post try except
```

In [119…

```
a = 1
b = 'w'

try:
    c= int(a) + int(b)
    print(c)
except Exception as e:
    print(e)
else:
    print("All inputs are good, Else clause got executed!!")
finally:
    print("Finally runs post try except")
```

```
invalid literal for int() with base 10: 'w'
Finally runs post try except
```

Data Structures And Algorithms

- Arrays
- Stack
- Queue
- Linked List
- Linear Search
- Binary Search
- Insersion Sort
- Quick Sort
- Merge Sort

Arrays:

- Linear Data structure
- Continous Memory Locations
- Access elements randomly
- Homogeneous elements i.e. similar elements

Applications

- Storing Information - linear fashion
- Suitable for applications that require frequent searching

1-Dimentional Array

- 1D can be related to a row
- Elements are stored one after another
- Only one subscript or index is used

Declaration and Initialization

- Array declaration:
  - Datatype varname [size]
- Can also do declaration and initialization at once:
  - Datatype varname [] = {ele1, ele2, ele3, ele4};

2-Dimentional Array

- 2D can be related to a table or matrix
- Elements are stored one after another i.e. one 1D array inside another.
- Two subscripts or indices are used, one row and one column.
- Dimensions depends upon the number of subscripts used.

In [126…
```
# 1D array
print("How many elements to store inside the array", end="")
num = input()
arr = []
print("\nEnter", num, "Element:", end="")
num = int(num)
for i in range(num):
    element = input()
    arr.append(element)
print("\nThe array elements are")
for i in range(num):
    print(arr[i], end=" ")
```

```
How many elements to store inside the array
Enter 2 Element:
The array elements are
3 4
```

In [127…
```
r_num = int(input("Input number of rows: "))
c_num = int(input("Input number of columns: "))
twoD_arr = [[0 for col in range(c_num)] for row in range(r_num)]
print(twoD_arr)
```

```
[[0, 0, 0], [0, 0, 0]]
```

In [128…
```
# 2D Array
r_num = int(input("Input number of rows: "))
c_num = int(input("Input number of columns: "))
twoD_arr = [[0 for col in range(c_num)] for row in range(r_num)]
# print("Enter the elements of the matrix: ")
# for i in range(r_num):
#     for j in range(c_num):
#         twoD_arr[i][j] = int(input())
print("The matrix is: ")
for i in range(r_num):
    for j in range(c_num):
        print(twoD_arr[i][j], i*j)
```

```
        twoD_arr[i][j]= i*j
print(twoD_arr)
```

```
The matrix is:
0 0
0 0
0 0
0 0
0 1
0 2
[[0, 0, 0], [0, 1, 2]]
```

In [131…

```
# Array with user defined value
print(end="Enter the size of Array: ")
tot = int(input())
arr = []
print(end="Enter " +str(tot)+ " Elements: ")
for i in range(tot):
    arr.append(input())
print("\nThe new array is: ")
for i in range(tot):
    print(end=arr[i]+" ")
print(end="\nEnter the value to Delete: ")
val = input()
if val in arr:
    arr.remove(val)
    print("\nThe new array is: ")
    for i in range(tot-1):
        print(end=arr[i]+" ")
else:
    print("\nElement doent exist in the List!")
```

```
Enter the size of Array: Enter 3 Elements:
The new array is:
2 4 6
Enter the value to Delete:
The new array is:
2 6
```

In [132…

```
# Sort elements in an array
arr = [10, 22, 38, 27, 11]
temp = 0;

#Displaying elements of original array
print("Elements of original array: ");
for i in range(0, len(arr)):
    print(arr[i], end=" ");

#Sort the array in ascending order
for i in range(0, len(arr)):
    for j in range(i + 1, len(arr)):
        if(arr[i] > arr[j]):
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
print();
#Displaying elements of the array after sorting
print("\nElements of array sorted in ascending order: ")
for i in range(0, len(arr)):
    print(arr[i], end=" ");
```

```
Elements of original array:
10 22 38 27 11

Elements of array sorted in ascending order:
10 11 22 27 38
```

In [133…
```python
# Search element in an array
import array
arr = array.array('i', [1, 2, 3, 1, 2, 5])
print("Elements of original array: ")
for i in range(0, 6):
    print(arr[i], end=" ");
print("\r");
print("The index of first occurance of 2 is: ", end="")
print(arr.index(2))
print("The index of first occurance of 1 is: ", end="")
print(arr.index(1))
```

```
Elements of original array:
1 2 3 1 2 5
The index of first occurance of 2 is: 1
The index of first occurance of 1 is: 0
```

Advantages of an array

- Random access elements
- Easy sorting and iteration
- Replacement of multiple variables

Disadvantage of an array

- Size is fixed
- Difficult to insert and delete
- If capacity is more and occupancy less most of the array gets wasted
- Needs continous memory

Stack

- Linear data structure
- It follows Last In First Out(LIFO) order
- Insertion and removal of the element has done at one end
- Push is used for inserting an element in a stack
- Pop is used to removal an element in a stack

Functions

- push(x) - it is used to insert the element 'x' at the end of a stack.
- pop() - it is used to remove the topmost/last element of a stack.
- size() - gives the size/length of a stack.
- top() - give reference of last element present in stack
- empty() - return true for an empty stack

Implementation of Stack

Several ways to implement stack in python

- list
- collections.deque
- queue.LifoQueue

Implementation of stack using list

List in python can be used as stack

- append() - it is used to insert the element
- pop() - it is used to remove the last element

Logic-

stack = []

stack.append("abc")

print(stack.pop())

In [134…
```
# Implementation of stack using list
stack= []
stack.append("Welcome")
stack.append("to")
stack.append("great learning")
print(stack)
print(stack.pop())
print(stack)
```

```
['Welcome', 'to', 'great learning']
great learning
['Welcome', 'to']
```

Implementation of stack using deque

- STACK in python are created by the collection module which provides deque class.
- Append and pop operations are faster in deque as compared to list

Logic:

from collectios import deque

stack = deque()

stack.append("abc")

print(stack.pop())

In [135…
```
### Implementation of stack using deque
from collections import deque
```

```
stack = deque()
stack.append("Welcome")
stack.append("to")
stack.append("great learning")
print(stack)
print(stack.pop())
print(stack)
```

```
deque(['Welcome', 'to', 'great learning'])
great learning
deque(['Welcome', 'to'])
```

Implementation of stack using queue

- Queue module contains the LIFO queue
- It is having some additional functions and works same as stack
- Put function is used to insert the data in queue
- Get function is used to remove the element

Functions available in queue model

- get() - it is used to remove the element from queue
- maxsize() - Used to put the maximum number of items allowed in queue
- empty() - It return true when queue is empty else false
- full() - When queue is full returns True
- put(x) - It is used to insert x in queue
- qsize() - Gives size of a queue

Logic:

from queue import LifoQueue

stack = Lifoqueue()

In [136…
```
# Implementation of stack using queue
from queue import LifoQueue
stack = LifoQueue(maxsize=3)
stack.put("Welcome")
stack.put("to")
stack.put("great learning")
print(stack)
print(stack.qsize())
print(stack.full())
print(stack.get())
print(stack.qsize())
print(stack)
```

```
<queue.LifoQueue object at 0x119c51090>
3
True
great learning
2
<queue.LifoQueue object at 0x119c51090>
```

What is Queue

- Linear data structure
- Follows FIFO: First in first out
- Insertion can take place from the rear end
- Deletion can take place from front end
- Queue at ticket counter, bus station
- 4 major operations
  - enqueue(ele) - used to insert element at top
  - dequeue() - removes the top element from queue
  - peekfirst() - to get first element of queue
  - peeklast() - to get last element of queue
- All operation works in constant time i.e. 0(1)

Applications of Queue

- Scheduling
- Maintaining playlist
- Interrupt handling

Queue Implementation

- Enqueue
- Dequeue
- Display

In [137…
```
# Implementation of stack using queue
class Queue:
    def __init__(self):
        self.queue = []
    def enqueue(self, data):
        self.queue.append(data)
    def dequeue(self):
        if len(self.queue)<1:
            return None
        return self.queue.pop(0)
    def display(self):
        print(self.queue)
    def size(self):
        return len(self.queue)
```

In [138…
```
q = Queue()
q.enqueue(1)
q.enqueue(2)
q.enqueue(3)
q.enqueue(4)
q.display()
q.dequeue()
q.display()
```

```
[1, 2, 3, 4]
[2, 3, 4]
```

Queue Implementation

A program implementation for circular queue:

- Enqueue
- Dequeue

In [139…

```
# circular queue
class MyCircularQueue():
    def __init__(self, k):
        self.k = k
        self.queue = [None] * k
        self.head = self.tail = -1

    def enqueue (self, data):
        if((self.tail + 1) % self.k == self.head):
            print("Queue is full")
        elif (self.head == -1):
            self.head = 0
            self.tail = 0
            self.queue[self.tail] = data
        else:
            self.tail = (self.tail + 1) % self.k
            self.queue[self.tail] = data

    def dequeue(self):
        if(self.head == -1):
            print("Queue is empty")
        elif(self.head == self.tail):
            temp = self.queue[self.head]
            self.head = -1
            self.tail = -1
            return temp
        else:
            temp = self.queue[self.head]
            self.head = (self.head + 1) % self.k
            return temp

    def printCQueue(self):
        if(self.head == -1):
            print("No element in circular queue is found")
        elif(self.tail >= self.head):
            for i in range(self.head, self.tail + 1):
                print(self.queue[i], end = " ")
            print()
        else:
            for i in range(self.head, self.k):
                print(self.queue[i], end = " ")
            for i in range(0, self.tail + 1):
                print(self.queue[i], end = " ")
            print()
```

In [140…

```
obj = MyCircularQueue(5)
obj.enqueue(12)
obj.enqueue(22)
obj.enqueue(31)
obj.enqueue(44)
```

```
obj.enqueue(57)
print("Initial queue values")
obj.printCQueue()

obj.dequeue()
print("After removing an element from the queue")
obj.printCQueue()
```

```
Initial queue values
12 22 31 44 57
After removing an element from the queue
22 31 44 57
```

Advantages of queue

- Maintains data in FIFO manner
- Insersion from beginning and deletion from end takes O(1) time

Disadvantages of queue

- Manipulation is restricted front and rear
- Not much flexible

Linked List

It is collection or group of nodes

Each node contains data and reference (pointer) which contains the address of next node.

It is linear data structure

Elements are stored randomly in memory

Why Linked List

- Linked list is having more efficiency for performing the operations as compared to list
- Elements are stored randomly whereas in list at continuous memory
- Accessing the elements in linked list will be slower as compared to list
- Utilization of memory is higher than the list

Singly Linked List

- It is traversed only in one direction

Operations of Singly Linked List

- Insertion
- Deletion
- Traversal

Pseudo Code

Creating a node in Singly Linked List

Class Node:

```
    def __init__(self, data):

        self.data=data;

        self.reference=None;
```

node1= Node(7)

print(node1.data)

print(node1.reference)

Creating a class of singly Linked list

class LinkedList

```
    def __init__(self):

        self.head=None;
```

In [141…
```
# Creating a node class
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

n1 = Node(7)
print(n1.data)
print(n1.next)
```

```
7
None
```

In [142…
```
# creation of Singly Linked List
class SinglyLinkedList:
    def __init__(self):
        self.head = None

sll = SinglyLinkedList()
print(sll.head)
```

```
None
```

Searching Algorithms

Linear Search Algorithm

What is Linear Search

- It helps you to search for an element in a linear data structure.
- It checks each and every element for the element to be searched.
- Since this is done in linear fashion, it is termed as linear search.

In [143…

```
# linearSearch(arr, item)
#     for each element in array
#         if element == item
#             return index
#     return -1

def linearSearch(arr, n, x):
    for i in range(0, n):
        if (array[i] == x):
            return i
    return -1

array =[2, 4, 0, 1, 9]
n = len(array)
x = 1
result = linearSearch(array, n, x)
if(result == -1):
    print("Element not found")
else:
    print("Element found at index: ", result)
```

```
Element found at index:  3
```

Linear search - Time complexity

- Best Time complexity: O(1)
- Average Time complexity: O(n)
- Worst Time complexity: O(n)
- Here all the elements need to be compared in worst case to search for a given element.
- Best case could be the case where the first element matched to the element to be searched

Linear search - Space complexity

- No auxiliary space is required in linear search implementation.
- Hence space complexity is : O(1)

What is Binary Search

- Binary search is one of the searching technique
- It is used to search an element in a sorted array
- This searching technique is based on divide and conquer strategy and search space always reduces to half in every iteration
- This is very efficient technique for searching but it needs some order on which partition of the array will occur

In [ ]:
```
## Binary Search - Iterative Algorithm
# binarySearch(arr, size)
#     loop until beg is not equal to end
#         midindex = (beg + end) / 2
#         if arr[midindex] == item
#             return midindex
#         else if item > arr[midindex]
#             beg = midindex + 1
#         else
#             end = midindex - 1
```

In [ ]:
```
## Binary Search - Recursive Algorithm
# binarysearch(arr, beg, end, item)
#     if beg <= end
#         mid = (beg + end) / 2
#         if arr[mid] == item
#             return mid
#         else if item > arr[mid]
#             return binarysearch(arr, mid + 1, end, item)
#         else
#             return binarysearch(arr, beg, mid - 1, item)
#     else
#         return -1
```

In [144…
```
# Binary search in Python
def binarySearch(arr, l, r, x):
    while l <= r:
        mid = l + (r - l) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            l = mid + 1
        else:
            r = mid - 1
    return -1
```

In [145…
```
array = [3,4,5,6,7,8,9]
x = 4
result = binarySearch(array, 0, len(array)-1, x)
if(result == -1):
    print("Element not found")
else:
    print("Element found at index: ", result)
```

```
Element found at index:  1
```

Binary Search - Time Complexity O(log n)

- Best time complexity: O(1)
- Average time complexity: O(log n)
- Worst time complexity: O(log n)
- In each iteration, the search space is getting divided by 2. That means that in the current iteration you have to deal with half of the previous iteration array. And above steps continue till beg<end.

- Best case could be the case where the first mid-value get matched to the element to be searched.
- Worst case could be the case where the last mid-value get matched to the element to be searched.

Binary Search - Space Complexity O(1)

- No auxiliary space is required in binary search implementation.
- Hence space complexity is O(1).

What is insertion sort

- It is one of the easiest and brute force sorting algorithms
- Insertion sort is used to sort elements in either ascending or descending order
- In insersion sort, we maintain a sorted part and unsorted part
- It works just like playing cards i.e. picking one card and sorting it with the cards that we have in our hand already which in turn are sorted
- With every iteration, one item from unsorted is moved to the sorted part
- First element is picked and considered as sorted
- Then we start picking from 2nd elements onwards and start comparing it with elements in sorted part.
- We shift the elements from sorted by one element until an appropriate location is not found for the picked element
- This continues till all the elements get exhausted.

In [138…
```python
# Insertion sort using Python
def insersion_sort(array):
    for step in range(1, len(array)):
        key = array[step]
        j = step - 1
        while j >= 0 and key < array[j]:
            array[j + 1] = array[j]
            j = j - 1
        array[j + 1] = key
    return array
```

In [142…
```python
data = [5,2,1,7,8]
insersion_sort(data)
print(f"Sorted data: {data}")
```

```
Sorted data: [1, 2, 5, 7, 8]
```

Insertion sort - Time Complexity

- In the worst case, it will take n to pick all elements and then at max n shifts to set it to the right position
- In best case that is sorted array we will just poick the elements but no shifting will take place leading it to n time complexity that is every element is traversed at least once

- Best Time complexity: O(n)
- Average Time complexity: O(n^2)
- Worst Time complexity: O(n^2)

Insertion sort - Space Complexity

- No auxilary space is required in insertion sort implementation that is we are not using any arrays, linked list, stack, queue etc to store our elements
- Space Complexity: O(1)

Insertion sort - Analysis

- Number of comparisons: O(n^2)
- Number of Swaps
- Stable or Unstable
- Inplace or Outplace

Quick sort

- It is one of the most widely used sorting algorithm
- It follows divide and conquer algorithm
- Recursion is used in quicksort implementation
- In each recursive call, a pivot is chosen then the array is partitioned in such a way that all the elements less than pivot lie to the left and all the elements greater than pivot lie to the right
- After every call the chosen pivot occupies its correct position in the array which is supposed to be in sorted order
- So with each step, our problem gets reduced by 2 which leads to quick sorting
- Pivot can be last element of current array, first element of current array or any random element

In [139…

```python
## Quick sort using python
def partition(array, low, high):
    pivot = array[high]
    i = low - 1
    for j in range(low, high):
        if array[j] <= pivot:
            i = i + 1
            (array[i], array[j]) = (array[j], array[i])
    (array[i + 1], array[high]) = (array[high], array[i + 1])
    return i + 1

## Quick sort function
def quick_sort(array, low, high):
    if low < high:
        pi = partition(array, low, high)
        quick_sort(array, low, pi - 1)
        quick_sort(array, pi + 1, high)
    return array
```

In [140…
```
d = [9,8,7,2,10,20,1]
print("Unsorted data: ", d)
size = len(d)
d = quick_sort(d, 0, size - 1)
print("Sorted data: ", d)
```

```
Unsorted data:  [9, 8, 7, 2, 10, 20, 1]
Sorted data:  [1, 2, 7, 8, 9, 10, 20]
```

Quick Sort - Time Complexity

- Best Case: O(nlogn)
- Average Case: O(nlogn)
- Worst Case: O(n^2)
- Partition of elements take n time and in quicksort problem is divide by the factor 2

Quick Sort - Space Complexity

- O(n): basic approach
- O(logn): modified approach

Quick Sort - Stability

- It is unstable algorithm

Quick Sort - In Place

- Yes

What is Merge Sort

- In merge sort problem is divided into two sub problmes in every iteration.
- Hence efficiency is increased drastically
- I follows divide and conquer approach
- Divide break the problem in two sub problems which continues until problem set is left with one element only
- Conquer basically merges the 2 sorted array into the original array

In [141…
```
### Merge sort -Algorithm
# mergseSort(arr, left, right)
#     if left>= right
#         return
#     mid = (left + right) / 2
#     mergeSort(arr, left, mid)
#     mergeSort(arr, mid + 1, right)
#     merge(arr, left, mid, right)
# end
```

Merge - Algorithm

- Create 2 subarrays left and right
- Create 3 iterators i, j, k
- Insert elements in lft and right (i& j)
- k - Replace the values in the original array
- Pick the larger elements from left and right and place them in the correct position
- If there are no elements in either left or right, pick up the remaining elements either from left or right and insert in original array

In [146…

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_arr = arr[:mid]
        right_arr = arr[mid:]

        # Recursive call on each half
        merge_sort(left_arr)
        merge_sort(right_arr)

        # Two iterators for traversing the two halves
        i = j = k = 0

        # Compare elements of left and right halves
        while i < len(left_arr) and j < len(right_arr):
            if left_arr[i] < right_arr[j]:
                arr[k] = left_arr[i]
                i += 1
            else:
                arr[k] = right_arr[j]
                j += 1
            k += 1

        # Checking if any element was left
        while i < len(left_arr):
            arr[k] = left_arr[i]
            i += 1
            k += 1

        while j < len(right_arr):
            arr[k] = right_arr[j]
            j += 1
            k += 1

def printList(arr):
    for i in range(len(arr)):
        print(arr[i], end=" ")
    print()
```

In [147…

```
if __name__ == '__main__':
    arr = [11,34,2,18,33,22,88,9]
    merge_sort(arr)
    print("Sorted array is: ", arr)
```

```
Sorted array is:  [2, 9, 11, 18, 22, 33, 34, 88]
```

Python For Machine Learning

- Numpy
- Pandas
- Matplot Lib
- Seaborn

Numpy

- Numpy stands for Numerical python and is the core library for numeric and scientific computing
- It consists of multi-dimensional array objects and a collection of routines for processing those arrays

In [148…
```
import numpy as np

arr = np.array([10, 22, 38, 27, 11])
arr
```

Out[148…
```
array([10, 22, 38, 27, 11])
```

In [149…
```
type(arr)
```

Out[149…
```
numpy.ndarray
```

In [150…
```
# Multi-dimensional
n1 = np.array([[1, 2, 3], [4, 5, 6]])
n1
```

Out[150…
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [151…
```
# Numpy array with zeros
n2 = np.zeros((5,5))
n2
```

Out[151…
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [152…
```
n3 = np.zeros((1,2))
n3
```

Out[152…
```
array([[0., 0.]])
```

In [153…
```
# Numpy with onces
n3= np.ones((3,2))
n3
```

Out[153…
```
array([[1., 1.],
       [1., 1.],
       [1., 1.]])
```

In [154…
```
# Numpy array with same number
n4 = np.full((2,2),10)
n4
```

Out[154…
```
array([[10, 10],
       [10, 10]])
```

In [155…
```
# Numpy array within a range
n5 = np.arange(11, 21)
n5
```

Out[155…
```
array([11, 12, 13, 14, 15, 16, 17, 18, 19, 20])
```

In [156…
```
n6 = np.arange(10,50,5)
n6
```

Out[156…
```
array([10, 15, 20, 25, 30, 35, 40, 45])
```

In [157…
```
n7 = np.arange(50,10,-5)
n7
```

Out[157…
```
array([50, 45, 40, 35, 30, 25, 20, 15])
```

In [158…
```
# Numpy with random numbers
n8 = np.random.randint(1, 100, 5)
n8
```

Out[158…
```
array([16, 47, 85, 30, 70])
```

In [159…
```
# Numpy with random numbers
n9 = np.random.randint(-100, -1, 5)
n9
```

Out[159…
```
array([ -67,  -71,  -27,  -44, -100])
```

In [160…
```
# Numpy shape
n10 = np.array([[1, 2, 3], [4, 5, 6]])
n10.shape
```

Out[160…
```
(2, 3)
```

In [161…
```
# Numpy shape change
n10.shape = (3,2)
n10
```

Out[161…
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

In [162…
```
# vstack()
n1 = np.array([1, 2, 3])
n2 = np.array([4, 5, 6])
np.vstack((n1, n2))
```

Out[162…
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [163…
```
# vstack()
n1 = np.array([1, 2, 3])
n2 = np.array([4, 5, 6])
n3 = np.array([1, 2, 3])
np.vstack((n1, n2, n3))
```

Out[163…
```
array([[1, 2, 3],
       [4, 5, 6],
       [1, 2, 3]])
```

In [164…
```
# hstack()
n1 = np.array([1, 2, 3])
n2 = np.array([4, 5, 6])
np.hstack((n1, n2))
```

Out[164…
```
array([1, 2, 3, 4, 5, 6])
```

In [165…
```
# hstack()
n1 = np.array([1, 2, 3])
n2 = np.array([4, 5, 6])
n3 = np.array([4])
np.hstack((n1, n2, n3))
```

Out[165…
```
array([1, 2, 3, 4, 5, 6, 4])
```

In [166…
```
# column_stack()
n1 = np.array([1, 2, 3])
n2 = np.array([4, 5, 6])
np.column_stack((n1, n2))
```

Out[166…
```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

In [167…
```
# Numpy intersection
n1 = np.arange(10, 61, 10)
n2 = np.arange(50, 91, 10)
np.intersect1d(n1, n2)
```

Out[167…
```
array([50, 60])
```

In [168…
```
# Numpy unique in one array with respect to other array
n1 = np.arange(10, 61, 10)
n2 = np.arange(50, 91, 10)
np.setdiff1d(n1, n2)
```

Out[168…
```
array([10, 20, 30, 40])
```

In [169…
```
# Numpy unique in one array with respect to other array
n1 = np.arange(10, 61, 10)
n2 = np.arange(50, 91, 10)
np.setdiff1d(n2, n1)
```

Out[169…
```
array([70, 80, 90])
```

In [170…
```
# Numpy Array Mathematics: total addition of all numbers
n1 = np.array([1, 2, 3])
n2 = np.array([4, 5, 6])
n3 = np.sum([n1, n2])
n3
```

Out[170… 21

In [171…
```
# Numpy Array Mathematics: addition of all columns
n1 = np.array([1, 2, 3])
n2 = np.array([4, 5, 6])
np.sum([n1, n2], axis =0)
```

Out[171… array([5, 7, 9])

In [172…
```
# Numpy Array Mathematics: addition of all rows
n1 = np.array([1, 2, 3])
n2 = np.array([4, 5, 6])
np.sum([n1, n2], axis =1)
```

Out[172… array([6, 15])

In [173…
```
# Numpy scaler Operations
# Addition
n1 = np.array([1, 2, 3])
n1+1
```

Out[173… array([2, 3, 4])

In [174…
```
# Subtraction
n1 = np.array([1, 2, 3])
n1-1
```

Out[174… array([0, 1, 2])

In [175…
```
# Multiplication
n1 = np.array([1, 2, 3])
print(n1 * 2)
print(n1)
```

```
[2 4 6]
[1 2 3]
```

In [176…
```
# Division
n1 = np.array([1, 2, 3])
n1/2
```

Out[176… array([0.5, 1. , 1.5])

In [177…
```
# Mathematical functions
n1 = np.array([10,22,37,41,59,63,74,85])
print(f"Mean is : {np.mean(n1)}")
print(f"median is : {np.median(n1)}")
print(f"Standard deviation is : {np.std(n1)}")
```

```
Mean is : 48.875
median is : 50.0
Standard deviation is : 24.17352632530058
```

In [178…
```
# Numpy Matrix
n1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
n1
```

Out[178…
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [179…
```
# rows
print(n1[0])
print(n1[1])
print(n1[2])
```

```
[1 2 3]
[4 5 6]
[7 8 9]
```

In [180…
```
# columns
print(n1[:, 0])
print(n1[:, 1])
print(n1[:, 2])
```

```
[1 4 7]
[2 5 8]
[3 6 9]
```

In [181…
```
# Matrix Transpose: Interchanging rows and columns
print(f"Original matrix is {n1}")
print(f"Transposed matrix is {n1.transpose()}")
```

```
Original matrix is [[1 2 3]
 [4 5 6]
 [7 8 9]]
Transposed matrix is [[1 4 7]
 [2 5 8]
 [3 6 9]]
```

In [182…
```
# Matrix Multiplication:
n1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
n2 = np.array([[9,7,8], [6,5,4], [3,2,1]])
n3 = np.dot(n1, n2)
n4 = np.dot(n2, n1)
print(f"np.dot(n1, n2) : {n3}")
print(f"np.dot(n2, n1) : {n3}")
n5 = n1.dot(n2)
n6 = n2.dot(n1)
print(f"n1.dot(n2) : {n5}")
print(f"n2.dot(n1) : {n6}")
```

```
np.dot(n1, n2) : [[ 30  23  19]
 [ 84  65  58]
 [138 107  97]]
np.dot(n2, n1) : [[ 30  23  19]
 [ 84  65  58]
 [138 107  97]]
n1.dot(n2) : [[ 30  23  19]
 [ 84  65  58]
 [138 107  97]]
n2.dot(n1) : [[ 93 117 141]
 [ 54  69  84]
 [ 18  24  30]]
```

In [183…
```
# Saving and loading numpy array
n1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
np.save('processed_data/test.npy', n1)
n2 = np.load('processed_data/test.npy')
n2
```

Out[183…
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Pandas

- Pandas stand for panel data and is the core library for data manipulation and data analysis
- It consist of single and multi dimentional data structures for data manipulation
- Series Object: One-dimensional labeled array
- Data-frame: Multi-dimensional labeled array

In [184…
```
import pandas as pd
```

In [187…
```
# Series Object: One-dimensional labeled array
s1 = pd.Series([1, 2, 3, 4, 5])
s1
```

Out[187…
```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

In [188…
```
type(s1)
```

Out[188…
```
pandas.core.series.Series
```

In [189…
```
# Changing index
s1= pd.Series([1, 2, 3, 4, 5], index = ["a", "b", "c", "d", "e"])
s1
```

Out[189…
```
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

In [190…
```
type(s1)
```

Out[190…
```
pandas.core.series.Series
```

In [191…
```
# Series object from dictionary
d1= {"a": 1, "b": 2, "c": 3, "d": 4, "e": 5}
s2 = pd.Series(d1)
s2
```

Out[191…
```
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

In [192…
```
# Change index positions
d1= {"a": 1, "b": 2, "c": 3, "d": 4, "e": 5}
s2 = pd.Series(d1, index=["b", "e",  "c", "a", "d"])
s2
```

Out[192…
```
b    2
e    5
c    3
a    1
d    4
dtype: int64
```

In [193…
```
# Pandas extracting induvidual elements
s1 = pd.Series([1, 2, 3, 4, 5, 6, 7])
# Exctracting element at index 3
s1[3]
```

Out[193…
```
4
```

In [194…
```
# Extracting elemenst till index 4
s1[:4]
```

Out[194…
```
0    1
1    2
2    3
3    4
dtype: int64
```

In [195…
```
# Extracting elemenst from index 3 till last
s1[3:]
```

Out[195…
```
3    4
4    5
5    6
6    7
dtype: int64
```

In [196…
```
# Extracting last 3 elements
s1[-3:]
```

Out[196…
```
4    5
5    6
6    7
dtype: int64
```

In [197…
```
# Basic math operation on Series
s1 = pd.Series([1, 2, 3, 4, 5, 6, 7])
s1 + 5
```

Out[197…
```
0     6
1     7
2     8
3     9
4    10
5    11
6    12
dtype: int64
```

In [198…
```
s1 = pd.Series([1, 2, 3, 4, 5, 6, 7])
s2 = pd.Series([0, 1, 2, 3, 4, 5, 6])
s1 + s2
```

Out[198…
```
0     1
1     3
2     5
3     7
4     9
5    11
6    13
dtype: int64
```

Pandas Dataframe

- Dataframe is a 2-dimentional labelled data-structure
- A dataframe is a collection of series (A data-frame comprises of rows and columns)

In [199…
```
df = pd.DataFrame({"Name": ["Meera", "Mirukali", "Ganu"], "Marks": [7, 5,
df
```

Out[199…

	Name	Marks	Age
0	Meera	7	7
1	Mirukali	5	8
2	Ganu	6	7

In [200…
```
type(df)
```

Out[200…
```
pandas.core.frame.DataFrame
```

In [201…
```
iris = pd.read_csv('data/iris.csv')
iris.head()
```

Out[201…

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [202…
```
iris.tail()
```

Out[202…

	sepal_length	sepal_width	petal_length	petal_width	species
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

In [203…
```
iris.count()
```

Out[203…
```
sepal_length    150
sepal_width     150
petal_length    150
petal_width     150
species         150
dtype: int64
```

In [204…
```
iris.shape
```

Out[204…
```
(150, 5)
```

In [205…
```
iris.describe()
```

Out[205…

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [206…
```
# get individual data from dataframe using .iloc[]
iris.iloc[0]
```

Out[206…
```
sepal_length       5.1
sepal_width        3.5
petal_length       1.4
petal_width        0.2
species         setosa
Name: 0, dtype: object
```

In [207…
```
# Get first three rows and three column
iris.iloc[:3, :3]
```

Out[207…

	sepal_length	sepal_width	petal_length
0	5.1	3.5	1.4
1	4.9	3.0	1.4
2	4.7	3.2	1.3

In [208…
```
iris.iloc[30:40, 3:]
```

Out[208…

	petal_width	species
30	0.2	setosa
31	0.4	setosa
32	0.1	setosa
33	0.2	setosa
34	0.1	setosa
35	0.2	setosa
36	0.2	setosa
37	0.1	setosa
38	0.2	setosa
39	0.2	setosa

In [209…
```
iris.loc[30:40, ("sepal_length", "petal_length")]
```

Out[209…

	sepal_length	petal_length
30	4.8	1.6
31	5.4	1.5
32	5.2	1.5
33	5.5	1.4
34	4.9	1.5
35	5.0	1.2
36	5.5	1.3
37	4.9	1.5
38	4.4	1.3
39	5.1	1.5
40	5.0	1.3

In [210…
```
iris.head()
```

Out[210…

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [211…

```
# Dropping columns
iris.drop('sepal_length', axis=1)
```

Out[211…

	sepal_width	petal_length	petal_width	species
0	3.5	1.4	0.2	setosa
1	3.0	1.4	0.2	setosa
2	3.2	1.3	0.2	setosa
3	3.1	1.5	0.2	setosa
4	3.6	1.4	0.2	setosa
…	…	…	…	…
145	3.0	5.2	2.3	virginica
146	2.5	5.0	1.9	virginica
147	3.0	5.2	2.0	virginica
148	3.4	5.4	2.3	virginica
149	3.0	5.1	1.8	virginica

150 rows × 4 columns

In [212…

```
iris.head()
```

Out[212…

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [213…

```
iris.drop([1,2,3], axis=0)
```

Out [213…

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
…	…	…	…	…	…
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

147 rows × 5 columns

In [214…

```
# print(f"Mean of Iris data is : {iris.mean()}")
iris.mean()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[214], line 2
      1 # print(f"Mean of Iris data is : {iris.mean()}")
----> 2 iris.mean()

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/core/frame.py:11693, in DataFrame.mean(self, axis, skipna, numeric_only, **kwargs)
  11685 @doc(make_doc("mean", ndim=2))
  11686 def mean(
  11687     self,
   (...)
  11691     **kwargs,
  11692 ):
> 11693     result = super().mean(axis, skipna, numeric_only, **kwargs)
  11694     if isinstance(result, Series):
  11695         result = result.__finalize__(self, method="mean")

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/core/generic.py:12420, in NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)
  12413 def mean(
  12414     self,
  12415     axis: Axis | None = 0,
   (...)
  12418     **kwargs,
  12419 ) -> Series | float:
> 12420     return self._stat_function(
  12421         "mean", nanops.nanmean, axis, skipna, numeric_only, **kwargs
  12422     )

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/core/generic.py:12377, in NDFrame._stat_function(self, name, func, axis, skipna, numeric_only, **kwargs)
  12373 nv.validate_func(name, (), kwargs)
  12375 validate_bool_kwarg(skipna, "skipna", none_allowed=False)
> 12377 return self._reduce(
  12378     func, name=name, axis=axis, skipna=skipna, numeric_only=numeric_only
  12379 )

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/core/frame.py:11562, in DataFrame._reduce(self, op, name, axis, skipna, numeric_only, filter_type, **kwds)
  11558     df = df.T
  11560 # After possibly _get_data and transposing, we are now in the
  11561 #  simple case where we can use BlockManager.reduce
> 11562 res = df._mgr.reduce(blk_func)
  11563 out = df._constructor_from_mgr(res, axes=res.axes).iloc[0]
  11564 if out_dtype is not None and out.dtype != "boolean":

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/core/internals/managers.py:1500, in BlockManager.reduce(self, func)
   1498 res_blocks: list[Block] = []
   1499 for blk in self.blocks:
-> 1500     nbs = blk.reduce(func)
```

```
   1501     res_blocks.extend(nbs)
   1503 index = Index([None])  # placeholder

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/cor
e/internals/blocks.py:404, in Block.reduce(self, func)
    398 @final
    399 def reduce(self, func) -> list[Block]:
    400     # We will apply the function and reshape the result into a sin
gle-row
    401     #  Block with the same mgr_locs; squeezing will be done at a h
igher level
    402     assert self.ndim == 2
--> 404     result = func(self.values)
    406     if self.values.ndim == 1:
    407         res_values = result

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/cor
e/frame.py:11481, in DataFrame._reduce.<locals>.blk_func(values, axis)
  11479         return np.array([result])
  11480 else:
> 11481     return op(values, axis=axis, skipna=skipna, **kwds)

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/cor
e/nanops.py:147, in bottleneck_switch.__call__.<locals>.f(values, axis, sk
ipna, **kwds)
    145         result = alt(values, axis=axis, skipna=skipna, **kwds)
    146 else:
--> 147     result = alt(values, axis=axis, skipna=skipna, **kwds)
    149 return result

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/cor
e/nanops.py:404, in _datetimelike_compat.<locals>.new_func(values, axis, s
kipna, mask, **kwargs)
    401 if datetimelike and mask is None:
    402     mask = isna(values)
--> 404 result = func(values, axis=axis, skipna=skipna, mask=mask, **kwarg
s)
    406 if datetimelike:
    407     result = _wrap_results(result, orig_values.dtype, fill_value=i
NaT)

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/cor
e/nanops.py:720, in nanmean(values, axis, skipna, mask)
    718 count = _get_counts(values.shape, mask, axis, dtype=dtype_count)
    719 the_sum = values.sum(axis, dtype=dtype_sum)
--> 720 the_sum = _ensure_numeric(the_sum)
    722 if axis is not None and getattr(the_sum, "ndim", False):
    723     count = cast(np.ndarray, count)

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/cor
e/nanops.py:1686, in _ensure_numeric(x)
   1683 inferred = lib.infer_dtype(x)
   1684 if inferred in ["string", "mixed"]:
   1685     # GH#44008, GH#36703 avoid casting e.g. strings to numeric
-> 1686     raise TypeError(f"Could not convert {x} to numeric")
   1687 try:
   1688     x = x.astype(np.complex128)

TypeError: Could not convert ['setosasetosasetosasetosasetosasetosasetosas
etosasetosasetosasetosasetosasetosasetosasetosasetosasetosasetosasetosaset
```

```
osasetosasetosasetosasetosasetosasetosasetosasetosasetosasetosasetosasetos
asetosasetosasetosasetosasetosasetosasetosasetosasetosasetosasetosasetosas
etosasetosasetosasetosasetosaversicolorversicolorversicolorversicolo
rversicolorversicolorversicolorversicolorversicolorversicolorversicolorver
sicolorversicolorversicolorversicolorversicolorversicolorversicolorversico
lorversicolorversicolorversicolorversicolorversicolorversicolorversicolorv
ersicolorversicolorversicolorversicolorversicolorversicolorversicolorversi
colorversicolorversicolorversicolorversicolorversicolorversicolorversicolo
rversicolorversicolorversicolorversicolorversicolorversicolorversicolorver
sicolorversicolorvirginicavirginicavirginicavirginicavirginicavirginicavir
ginicavirginicavirginicavirginicavirginicavirginicavirginicavirginicavirgi
nicavirginicavirginicavirginicavirginicavirginicavirginicavirginicavirgini
cavirginicavirginicavirginicavirginicavirginicavirginicavirginicavirginica
virginicavirginicavirginicavirginicavirginicavirginicavirginicavirginicavi
rginicavirginicavirginicavirginicavirginicavirginicavirginicavirginicavirg
inicavirginicavirginica'] to numeric
```

In [215…

```
iris.median()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[215], line 1
----> 1 iris.median()

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/core/frame.py:11706, in DataFrame.median(self, axis, skipna, numeric_only, **kwargs)
  11698 @doc(make_doc("median", ndim=2))
  11699 def median(
  11700     self,
   (...)
  11704     **kwargs,
  11705 ):
> 11706     result = super().median(axis, skipna, numeric_only, **kwargs)
  11707     if isinstance(result, Series):
  11708         result = result.__finalize__(self, method="median")

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/core/generic.py:12431, in NDFrame.median(self, axis, skipna, numeric_only, **kwargs)
  12424 def median(
  12425     self,
  12426     axis: Axis | None = 0,
   (...)
  12429     **kwargs,
  12430 ) -> Series | float:
> 12431     return self._stat_function(
  12432         "median", nanops.nanmedian, axis, skipna, numeric_only, **kwargs
  12433     )

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/core/generic.py:12377, in NDFrame._stat_function(self, name, func, axis, skipna, numeric_only, **kwargs)
  12373 nv.validate_func(name, (), kwargs)
  12375 validate_bool_kwarg(skipna, "skipna", none_allowed=False)
> 12377 return self._reduce(
  12378     func, name=name, axis=axis, skipna=skipna, numeric_only=numeric_only
  12379 )

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/core/frame.py:11562, in DataFrame._reduce(self, op, name, axis, skipna, numeric_only, filter_type, **kwds)
  11558     df = df.T
  11560 # After possibly _get_data and transposing, we are now in the
  11561 #  simple case where we can use BlockManager.reduce
> 11562 res = df._mgr.reduce(blk_func)
  11563 out = df._constructor_from_mgr(res, axes=res.axes).iloc[0]
  11564 if out_dtype is not None and out.dtype != "boolean":

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/core/internals/managers.py:1500, in BlockManager.reduce(self, func)
   1498 res_blocks: list[Block] = []
   1499 for blk in self.blocks:
-> 1500     nbs = blk.reduce(func)
   1501     res_blocks.extend(nbs)
```

```
   1503 index = Index([None])  # placeholder

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/cor
e/internals/blocks.py:404, in Block.reduce(self, func)
    398 @final
    399 def reduce(self, func) -> list[Block]:
    400     # We will apply the function and reshape the result into a sin
gle-row
    401     #  Block with the same mgr_locs; squeezing will be done at a h
igher level
    402     assert self.ndim == 2
--> 404     result = func(self.values)
    406     if self.values.ndim == 1:
    407         res_values = result

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/cor
e/frame.py:11481, in DataFrame._reduce.<locals>.blk_func(values, axis)
  11479         return np.array([result])
  11480 else:
> 11481     return op(values, axis=axis, skipna=skipna, **kwds)

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/cor
e/nanops.py:147, in bottleneck_switch.__call__.<locals>.f(values, axis, sk
ipna, **kwds)
    145         result = alt(values, axis=axis, skipna=skipna, **kwds)
    146 else:
--> 147     result = alt(values, axis=axis, skipna=skipna, **kwds)
    149 return result

File /opt/anaconda3/envs/genAI_env/lib/python3.11/site-packages/pandas/cor
e/nanops.py:787, in nanmedian(values, axis, skipna, mask)
    785     inferred = lib.infer_dtype(values)
    786     if inferred in ["string", "mixed"]:
--> 787         raise TypeError(f"Cannot convert {values} to numeric")
    788 try:
    789     values = values.astype("f8")

TypeError: Cannot convert [['setosa' 'setosa' 'setosa' 'setosa' 'setosa'
'setosa' 'setosa' 'setosa'
  'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
  'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
  'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
  'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
  'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa' 'setosa'
  'setosa' 'setosa' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
  'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
  'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
  'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
  'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
  'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
  'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
  'versicolor' 'versicolor' 'versicolor' 'versicolor' 'versicolor'
  'versicolor' 'virginica' 'virginica' 'virginica' 'virginica'
  'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
  'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
  'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
  'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
  'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
```

```
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'virginica' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica']] to numeric
```

In [216…
```
iris.min()
```

Out[216…
```
sepal_length       4.3
sepal_width        2.0
petal_length       1.0
petal_width        0.1
species         setosa
dtype: object
```

In [217…
```
iris.max()
```

Out[217…
```
sepal_length          7.9
sepal_width           4.4
petal_length          6.9
petal_width           2.5
species         virginica
dtype: object
```

Matplotlib

- Matplotlib is a python library used for data visualisation

In [218…
```
import numpy as np
from matplotlib import pyplot as plt
```

In [219…
```
x = np.arange(1,11)
x
```

Out[219…
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [220…
```
y= 2 * x
y
```

Out[220…
```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

Line Graph:

A line graph is a visual representation of data that shows how values change over time or in relation to another variable. It consists of points connected by straight lines.

Key components of a line graph:

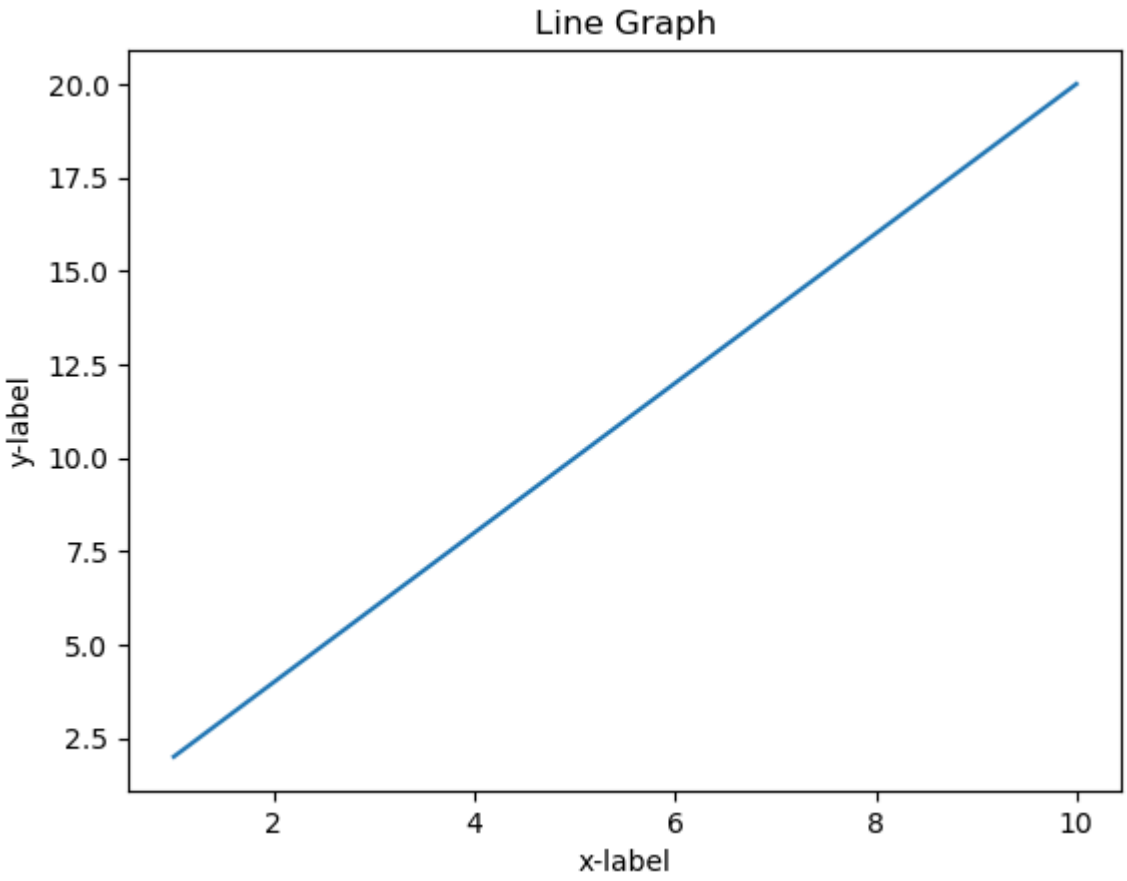
- X-axis: Represents the independent variable (often time).
- Y-axis: Represents the dependent variable (the value being measured).
- Data points: Represent specific values at different points on the x-axis.
- Line segments: Connect the data points to show trends and patterns.

When to use a line graph:

- To show trends over time: For example, temperature changes, stock prices, or population growth.
- To compare multiple data sets: For example, sales of different products over time.
- To identify patterns and correlations: For example, the relationship between hours studied and exam scores.

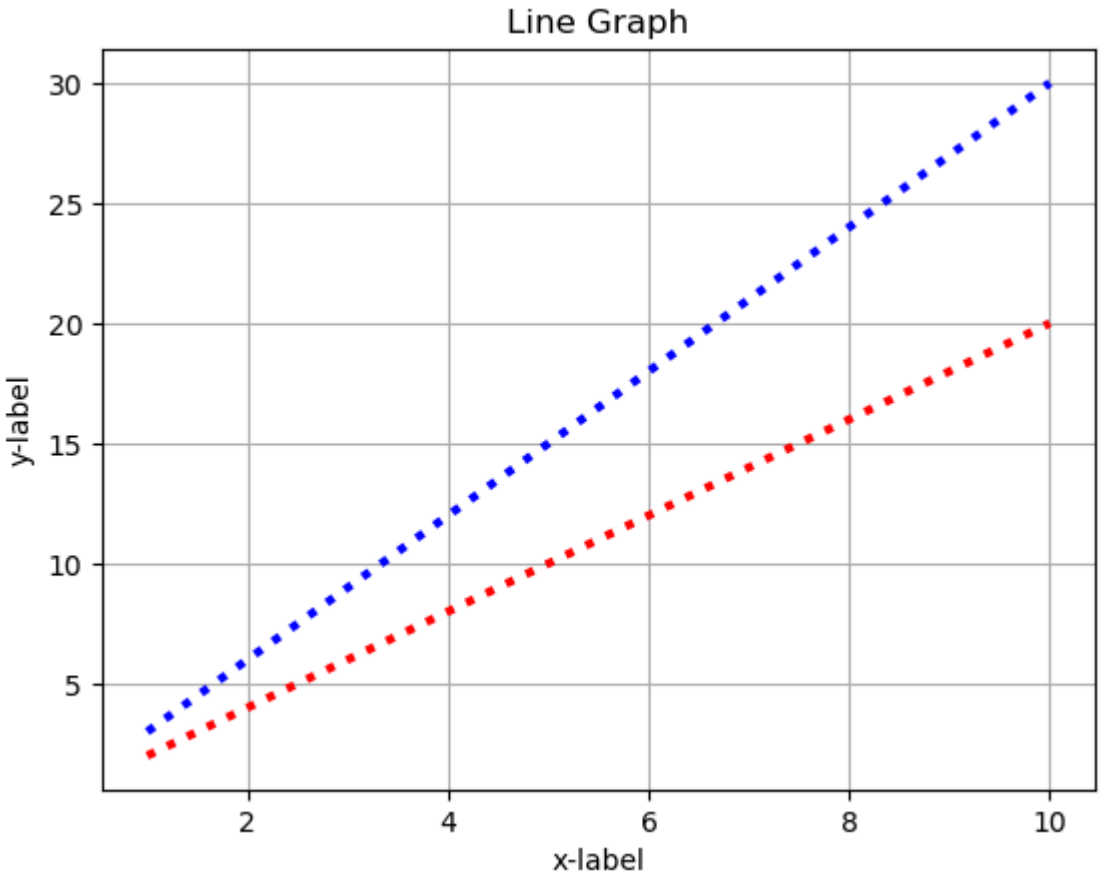
In [221…

```
plt.plot(x,y)
plt.title("Line Graph")
plt.xlabel("x-label")
plt.ylabel("y-label")
plt.show()
```



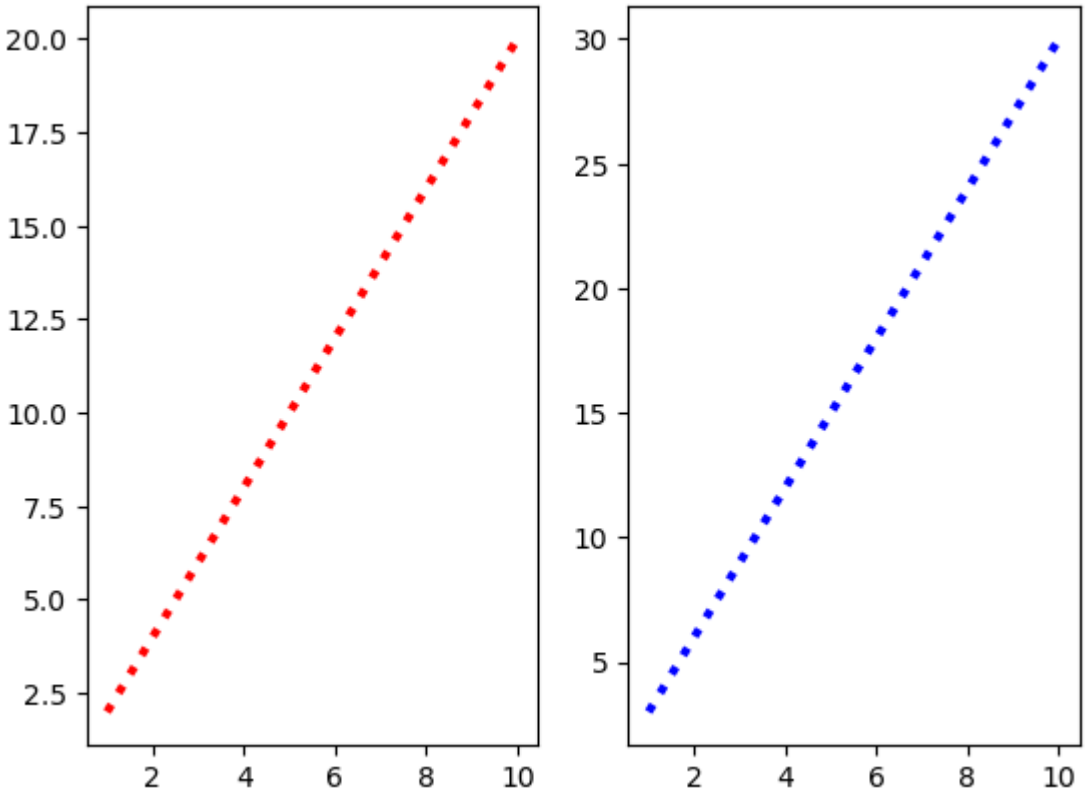
In [222…

```
x = np.arange(1,11)
y = 2 * x
y2 = 3 * x
plt.plot(x,y, color='r', linestyle=':', linewidth=3)
plt.plot(x,y2, color='b', linestyle=':', linewidth=3)
plt.title("Line Graph")
plt.xlabel("x-label")
plt.ylabel("y-label")
plt.grid(True)
plt.show()
```



In [223…

```
x = np.arange(1,11)
y = 2 * x
y2 = 3 * x

# plt.title("Line Graph")
# plt.xlabel("x-label")
# plt.ylabel("y-label")
# plt.grid(True)

plt.subplot(1,2,1)
plt.plot(x,y, color='r', linestyle=':', linewidth=3)
plt.subplot(1,2,2)
plt.plot(x,y2, color='b', linestyle=':', linewidth=3)

plt.show()
```



Bar Plot

A bar plot is a type of chart or graph that represents categorical data with rectangular bars. The length of each bar is proportional to the value it represents.

Key components of a bar plot:

- X-axis: Represents the categories or groups of data.
- Y-axis: Represents the values or frequency of each category.
- Bars: Rectangular shapes whose height or length corresponds to the data value.

Types of bar plots:

- Vertical bar plot: Bars are oriented vertically.
- Horizontal bar plot: Bars are oriented horizontally.
- Stacked bar plot: Multiple bars are stacked on top of each other to show the composition of a category.
- Grouped bar plot: Bars are grouped together to compare multiple categories.

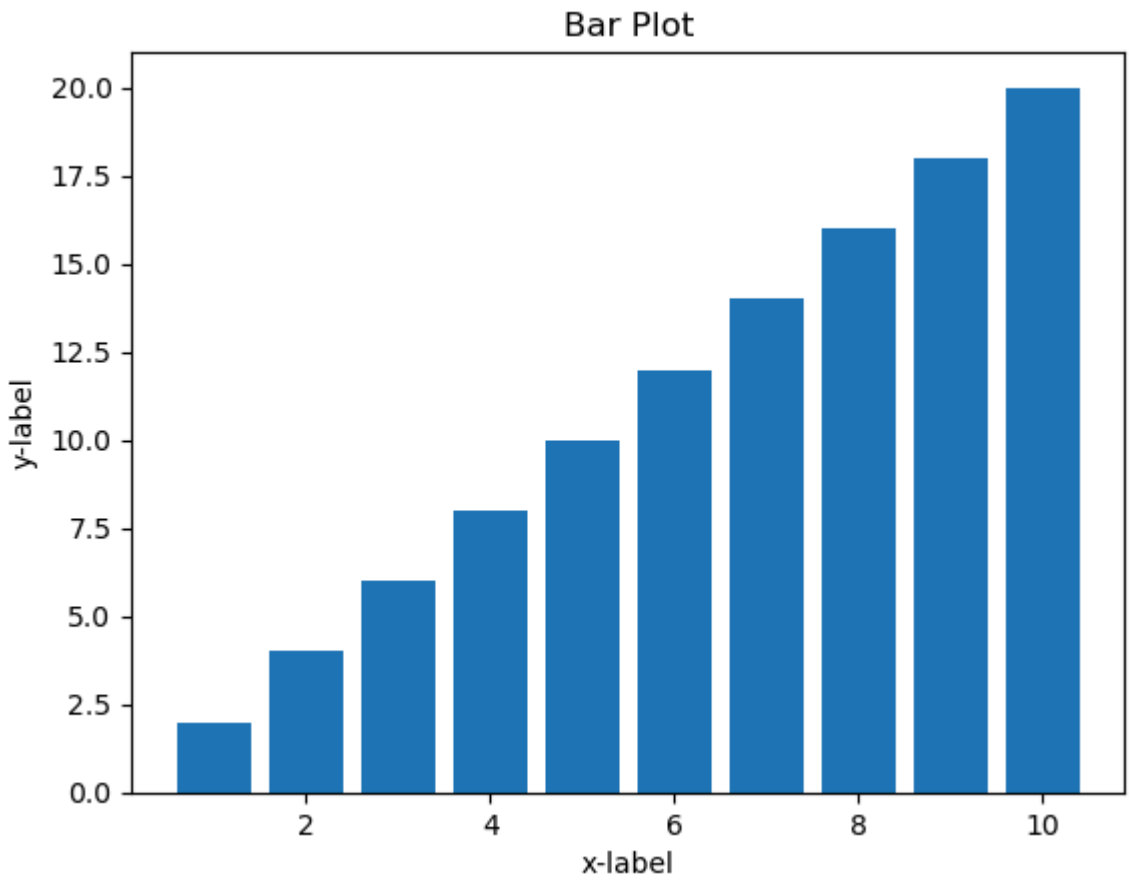
When to use a bar plot:

- To compare values across different categories.
- To show the distribution of categorical data.
- To visualize changes in values over time (using grouped or stacked bar plots).

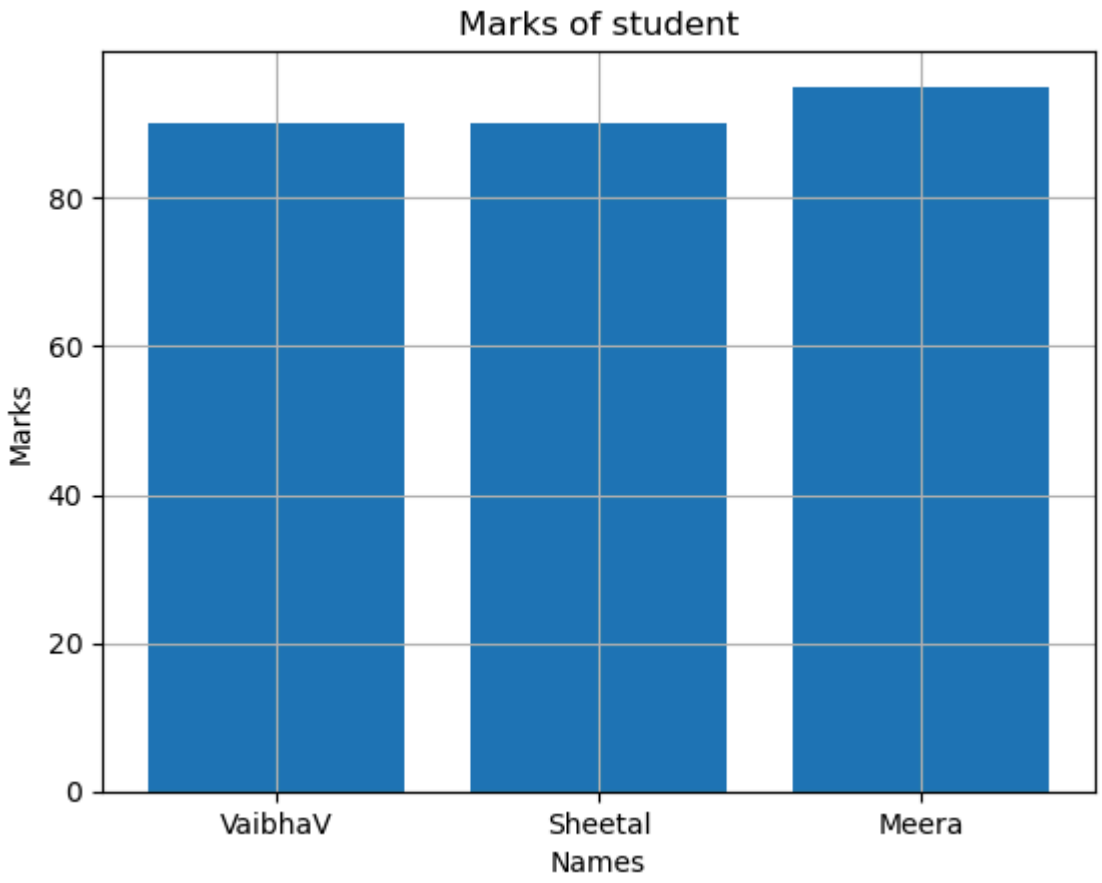
In [224…

```
x = np.arange(1,11)
y = 2 * x
```

```
plt.bar(x,y)
plt.title("Bar Plot")
plt.xlabel("x-label")
plt.ylabel("y-label")
plt.show()
```

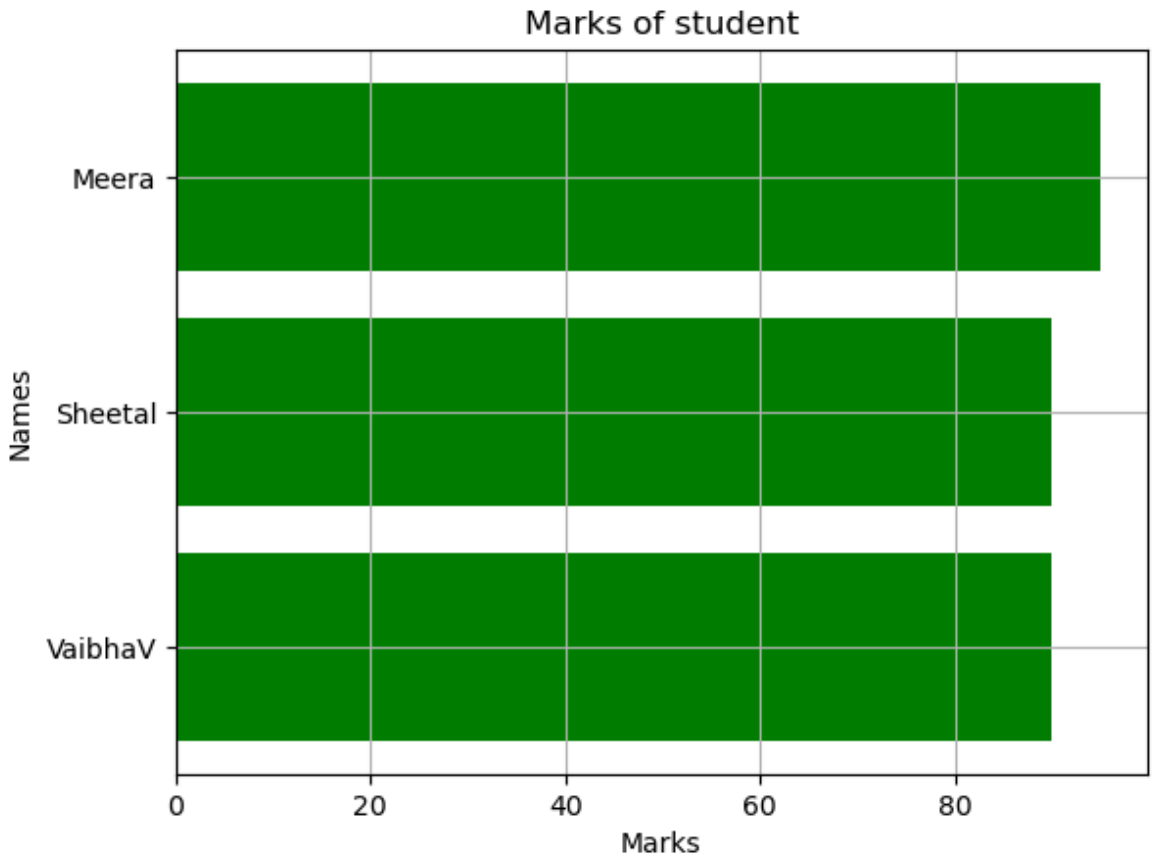


In [225…
```
studets= {"VaibhaV": 90, "Sheetal": 90, "Meera": 95}
names = list(studets.keys())
marks = list(studets.values())
plt.bar(names,marks)
plt.title("Marks of student")
plt.xlabel("Names")
plt.ylabel("Marks")
plt.grid(True)
plt.show()
```



In [226…

```
# Horizontal Bar Plot
plt.barh(names,marks, color='g')
plt.title("Marks of student")
plt.xlabel("Marks")
plt.ylabel("Names")
plt.grid(True)
plt.show()
```



Scatter Plot

A scatter plot is a type of graph that displays the relationship between two numerical variables. Each data point is represented by a dot on the graph, with the position of the dot determined by its values for the two variables.

Key components of a scatter plot:

- X-axis: Represents one numerical variable.
- Y-axis: Represents the other numerical variable.
- Data points: Represent individual observations, with their position determined by their values on the x and y axes.

When to use a scatter plot:

- To explore the relationship between two numerical variables.
- To identify patterns or trends in the data.
- To detect outliers or unusual data points.

Examples of scatter plots:

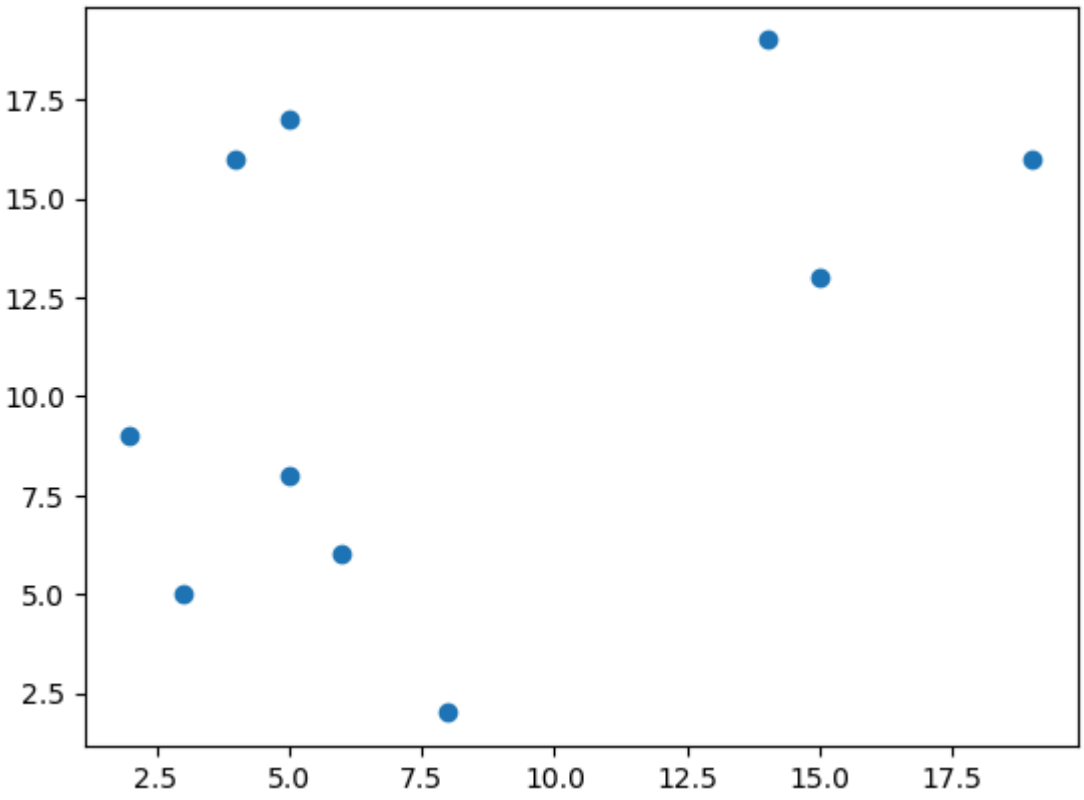
- Relationship between height and weight.
- Correlation between study hours and exam scores.
- Distribution of house prices based on square footage.

Advantages of scatter plots:

- Easy to visualize the relationship between two variables.
- Can reveal patterns, trends, and outliers.
- Useful for exploratory data analysis.

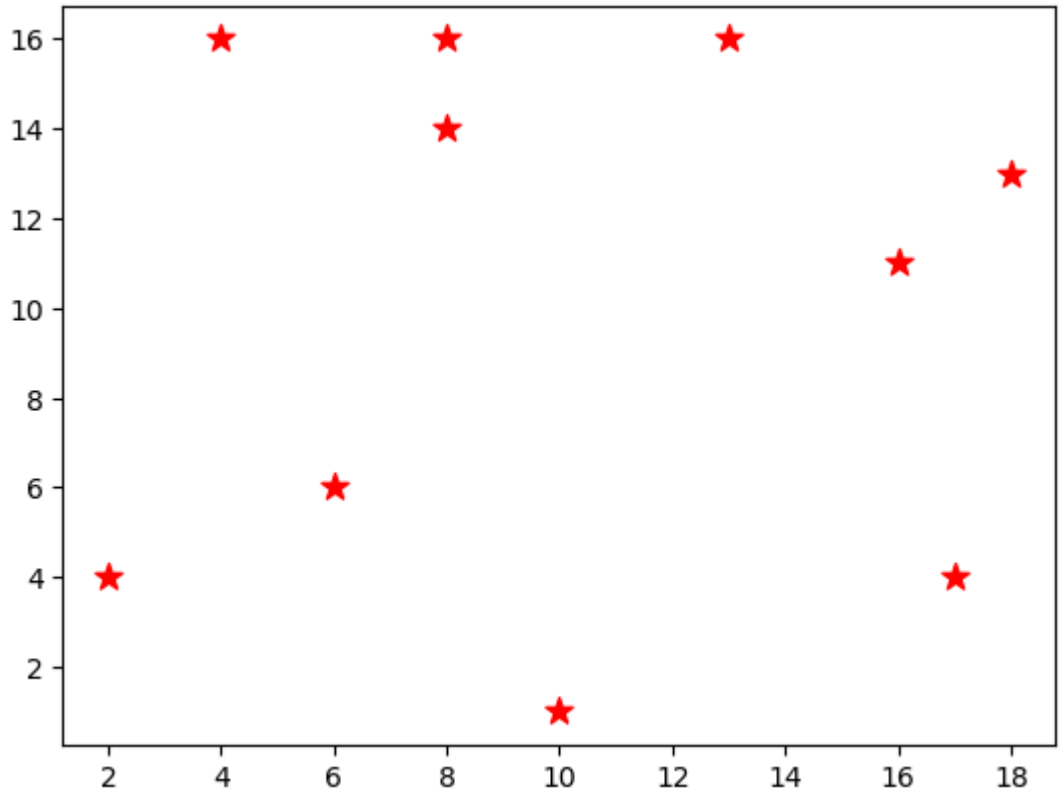
In [227…
```
# Scatter plot
x = np.random.randint(1,20,10)
print(f"x: {x}")
y = np.random.randint(1,20,10)
print(f"y: {y}")
plt.scatter(x,y)
plt.show()
```

```
x: [ 5  4  5  8  2  3 15 19  6 14]
y: [17 16  8  2  9  5 13 16  6 19]
```



In [228…
```
x = np.random.randint(1,20,10)
print(f"x: {x}")
y = np.random.randint(1,20,10)
print(f"y: {y}")
plt.scatter(x,y, color='r', marker='*', s=100)
plt.show()
```

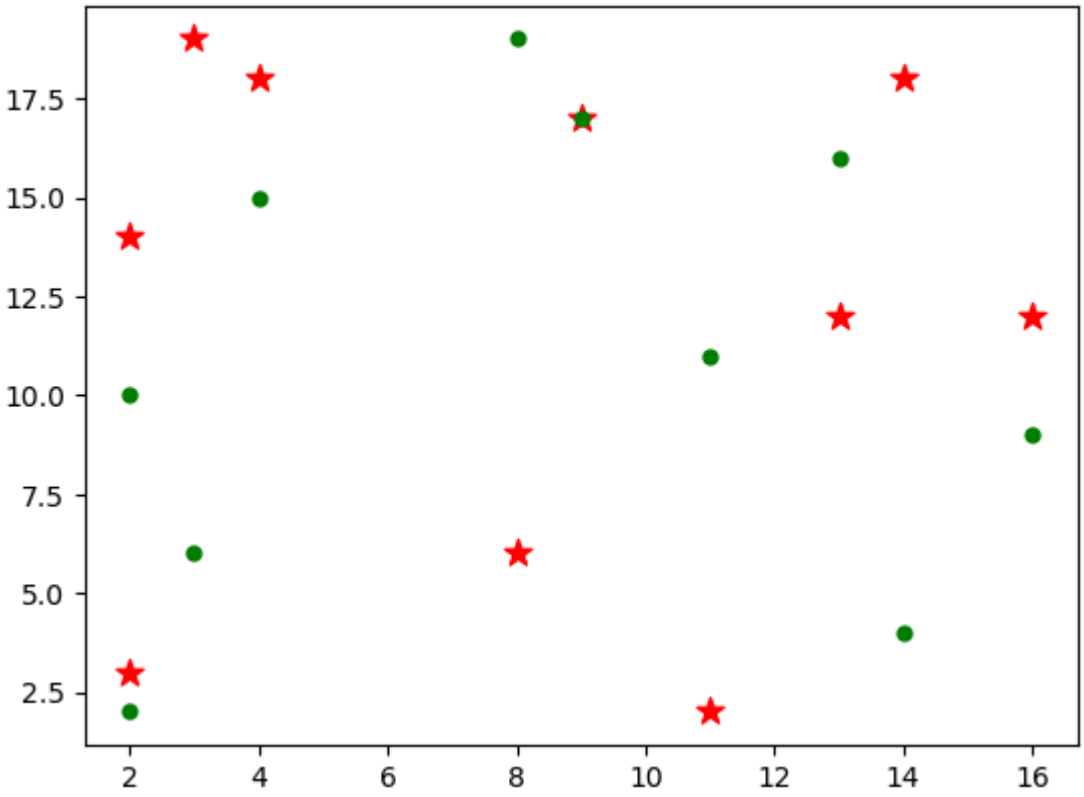
```
x: [ 4 16 13  6  2 18  8 17  8 10]
y: [16 11 16  6  4 13 16  4 14  1]
```



In [229…

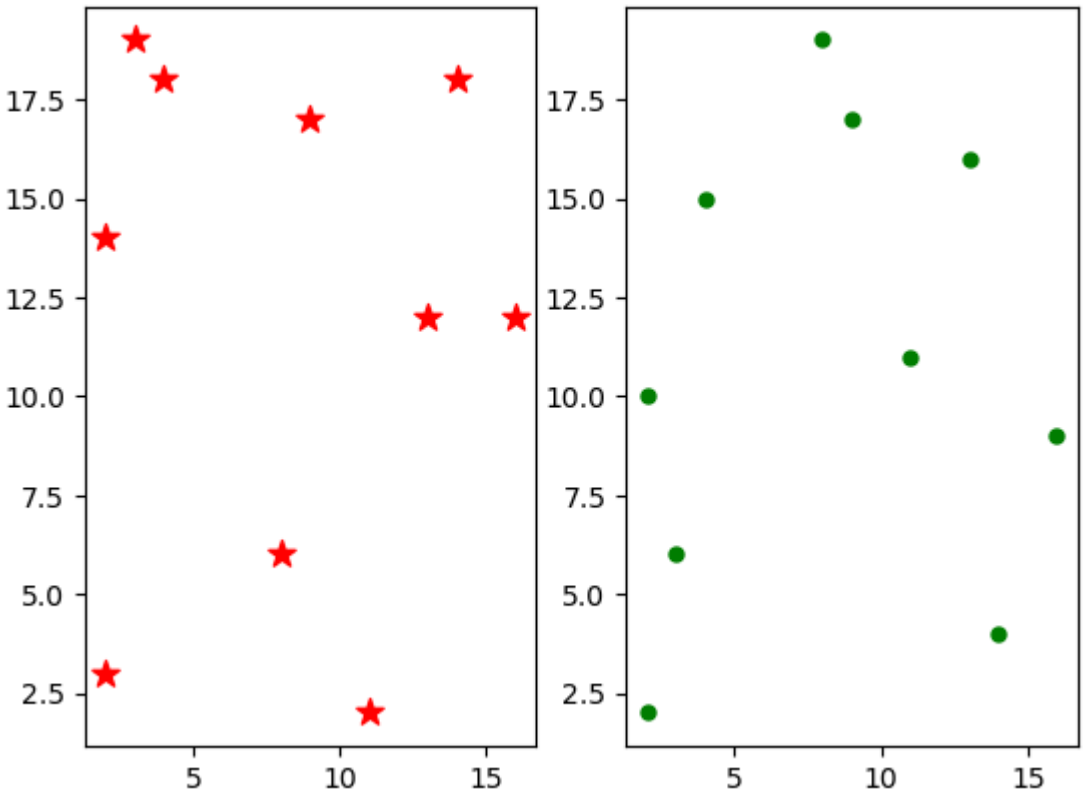
```
x = np.random.randint(1,20,10)
print(f"x: {x}")
y = np.random.randint(1,20,10)
print(f"y: {y}")
y2 = np.random.randint(1,20,10)
print(f"y2: {y2}")
plt.scatter(x,y, color='r', marker='*', s=100)
plt.scatter(x,y2, color='g', marker='.', s=100)
plt.show()
```

```
x: [ 8  2  4 14  3 16  9 13  2 11]
y: [ 6 14 18 18 19 12 17 12  3  2]
y2: [19 10 15  4  6  9 17 16  2 11]
```



In [230…

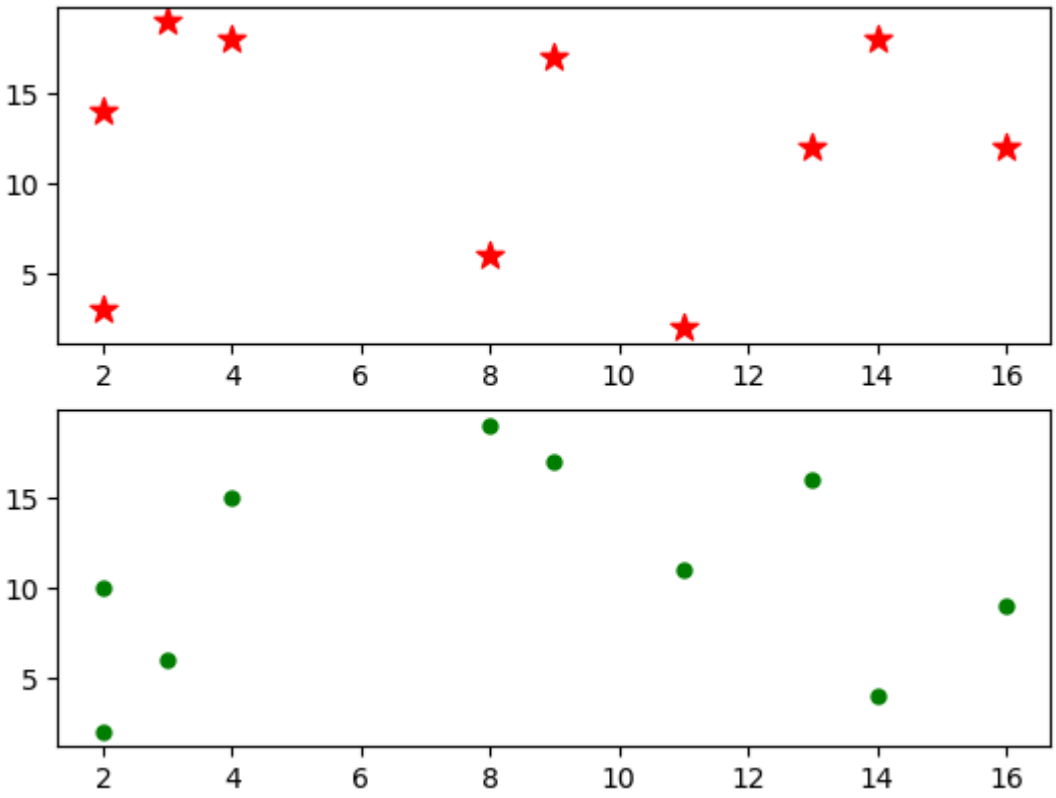
```
plt.subplot(1,2,1)
plt.scatter(x,y, color='r', marker='*', s=100)
plt.subplot(1,2,2)
plt.scatter(x,y2, color='g', marker='.', s=100)
plt.show()
```



In [231…

```
plt.subplot(2,1,1)
plt.scatter(x,y, color='r', marker='*', s=100)
plt.subplot(2,1,2)
```

```
plt.scatter(x,y2, color='g', marker='.', s=100)
plt.show()
```



Histogram

A histogram is a graphical representation of the distribution of numerical data. It's similar to a bar chart, but there are key differences:

- Bars are adjacent: Unlike bar charts, the bars in a histogram touch each other to indicate continuous data.
- Horizontal axis represents intervals: The x-axis shows ranges of values, called bins or class intervals.
- Vertical axis represents frequency: The y-axis shows the number of data points that fall within each bin.

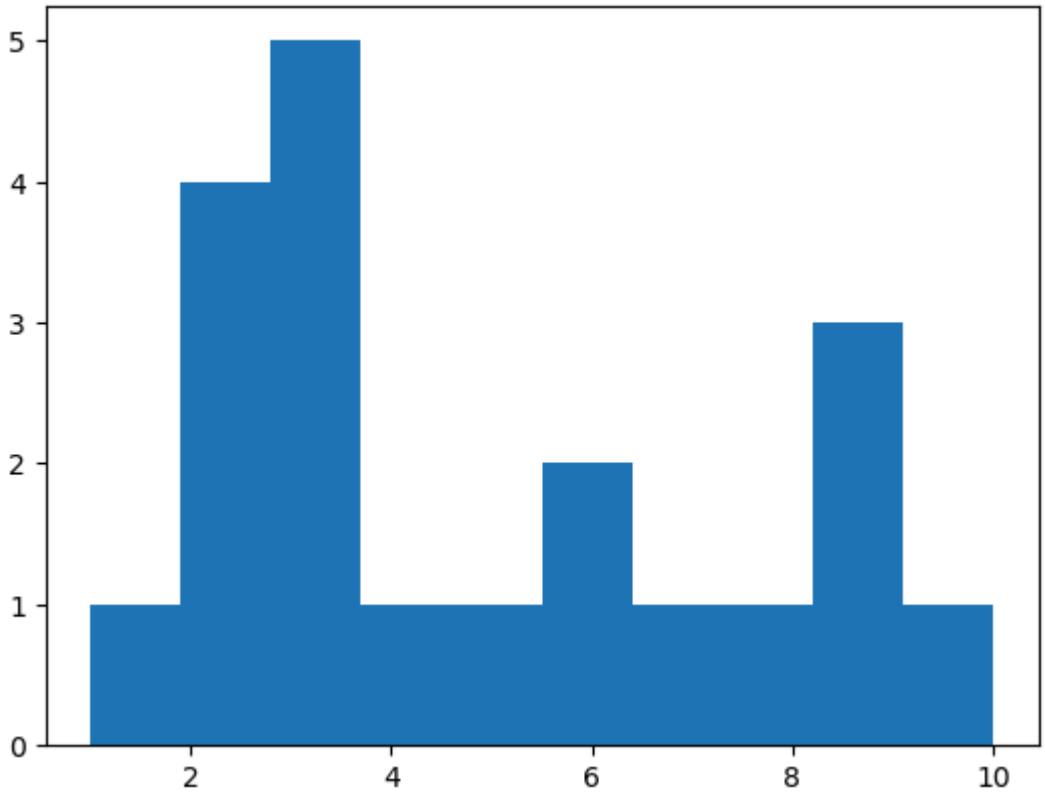
Key uses of a histogram:

- Understanding data distribution: It helps visualize the shape, center, and spread of data.
- Identifying outliers: Unusual data points can be easily spotted.
- Comparing distributions: Histograms can be used to compare different datasets.

In [232…

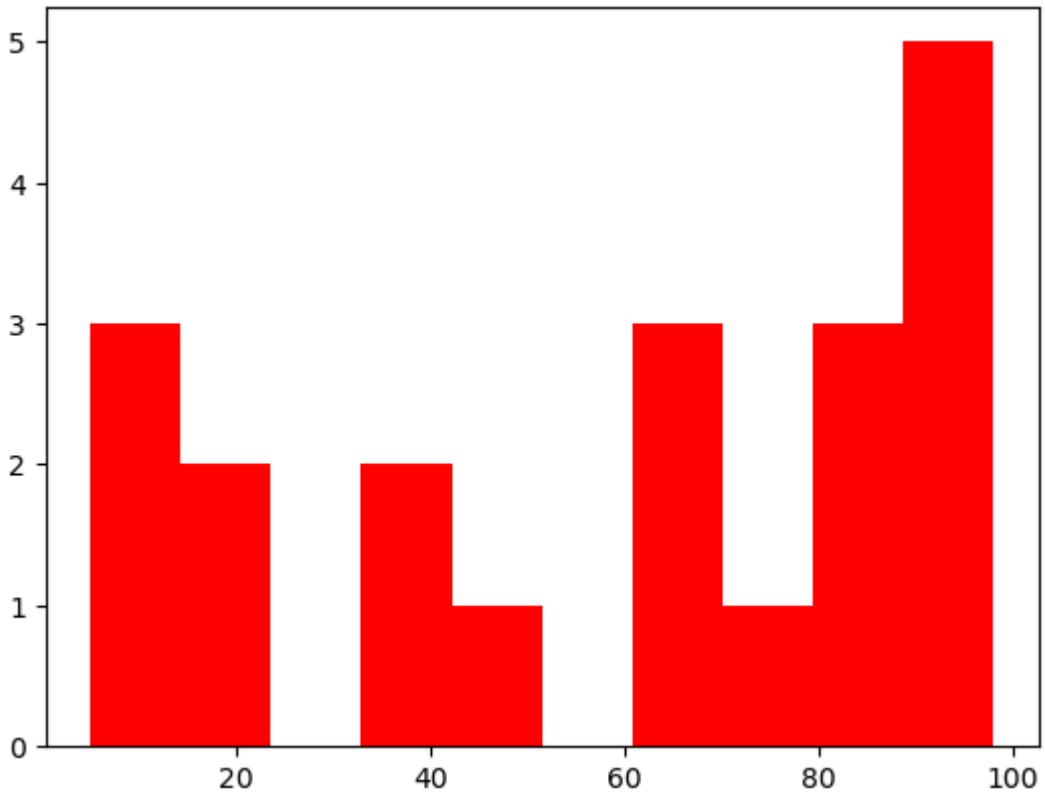
```
# Histogram

x = np.random.randint(1,11,20)
print(f"x: {x}")
plt.hist(x)
plt.show()
```

```
x: [ 7  6  8  2  9  3  2  3 10  4  9  2  9  1  3  3  5  2  3  6]
```



In [233…
```
x = np.random.randint(1,100,20)
print(f"x: {x}")
plt.hist(x, color='r', bins=10)
plt.show()
```

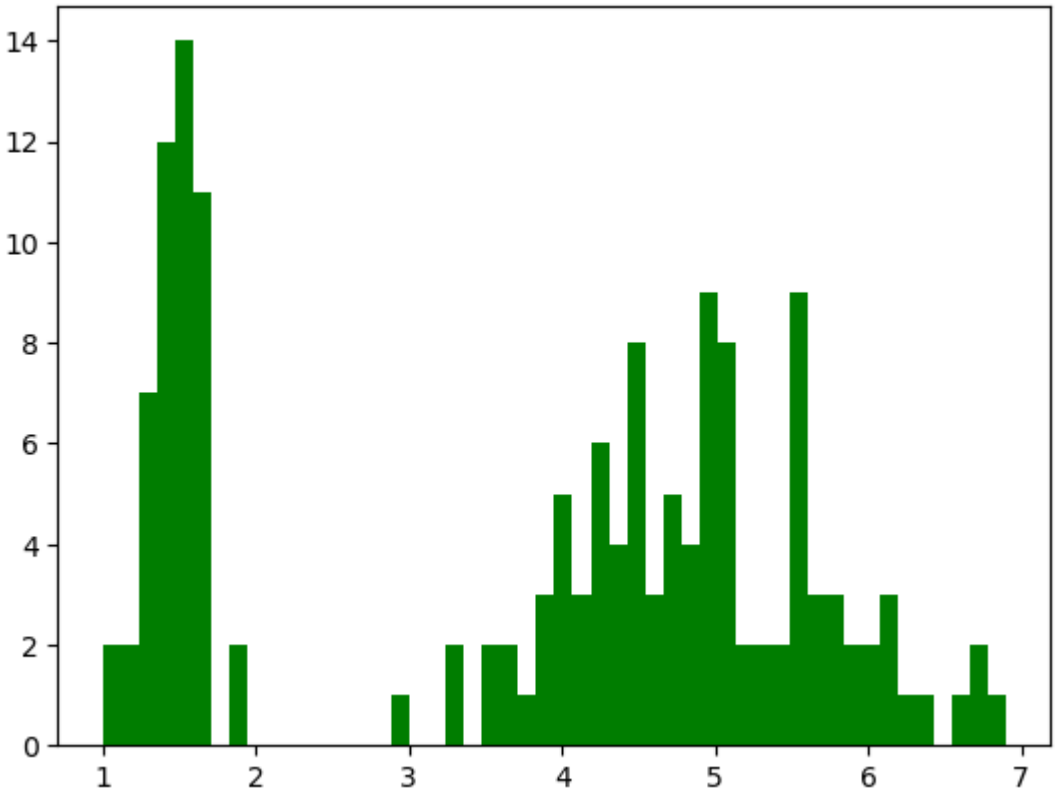
```
x: [49 39  5  5 74 35 84 87 63 66 98 96 93 20 23  5 85 96 70 97]
```



In [235…
```
iris= pd.read_csv('data/iris.csv')
print(iris.tail())
x= iris['petal_length']
```

```
plt.hist(x, color='g', bins=50)
plt.show()
```

	sepal_length	sepal_width	petal_length	petal_width	species
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica



Box Plot

A box plot (also known as a box-and-whisker plot) is a graphical representation of the distribution of a dataset. It provides a visual summary of five key statistical measures:

- Minimum: The smallest value in the dataset (excluding outliers).
- First quartile (Q1): The value below which 25% of the data lies.
- Median (Q2): The middle value of the dataset.
- Third quartile (Q3): The value below which 75% of the data lies.
- Maximum: The largest value in the dataset (excluding outliers).

How to interpret a box plot:

- The box: Represents the interquartile range (IQR), which contains the middle 50% of the data.
- The line within the box: Indicates the median.
- The whiskers: Extend from the box to the minimum and maximum values (excluding outliers).

- Outliers: Data points that fall outside the whiskers are often represented as individual points.

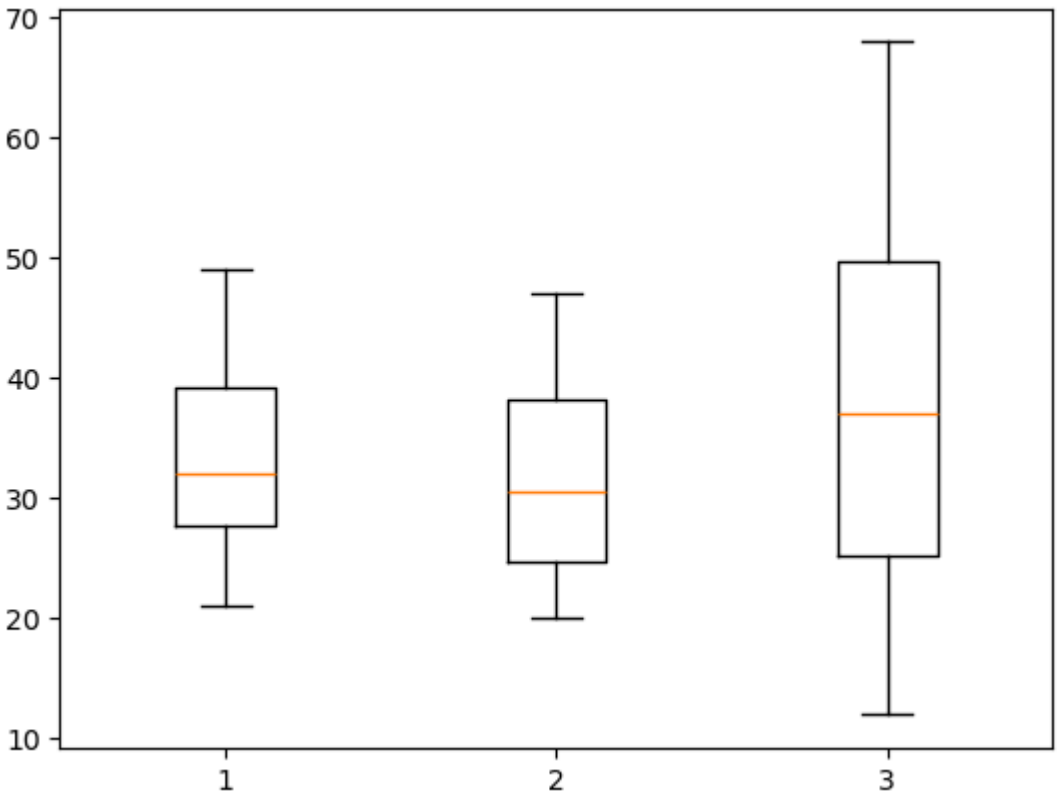
Why use a box plot?

- Quick overview of data: Provides a summary of the distribution at a glance.
- Comparison of groups: Multiple box plots can be used to compare different datasets.
- Identification of outliers: Unusual data points can be easily spotted.

In [236…

```
# Box Plot

x= np.random.randint(20,50,20)
y= np.random.randint(20,50,20)
z= np.random.randint(10,70,30)
print(f"x: {x}")
print(f"y: {y}")
print(f"z: {z}")
data= list([x,y,z])
plt.boxplot(data)
plt.show()
```

```
x: [21 24 33 49 29 39 35 23 36 48 32 28 43 40 27 29 32 23 46 32]
y: [24 24 27 29 20 25 41 31 30 34 45 39 34 47 24 38 39 34 23 29]
z: [25 27 44 34 58 54 32 12 31 48 23 32 53 68 25 21 49 20 68 40 38 39 55 4
1
 12 62 50 15 36 26]
```



Violin Plot

A violin plot is a statistical graphic for comparing probability distributions. It combines elements of a box plot and a kernel density plot.

Key components:

- Density plot: The shape of the violin represents the probability density of the data.
- Box plot: Typically overlaid on the violin, showing the median, quartiles, and sometimes outliers.

Advantages of violin plots:

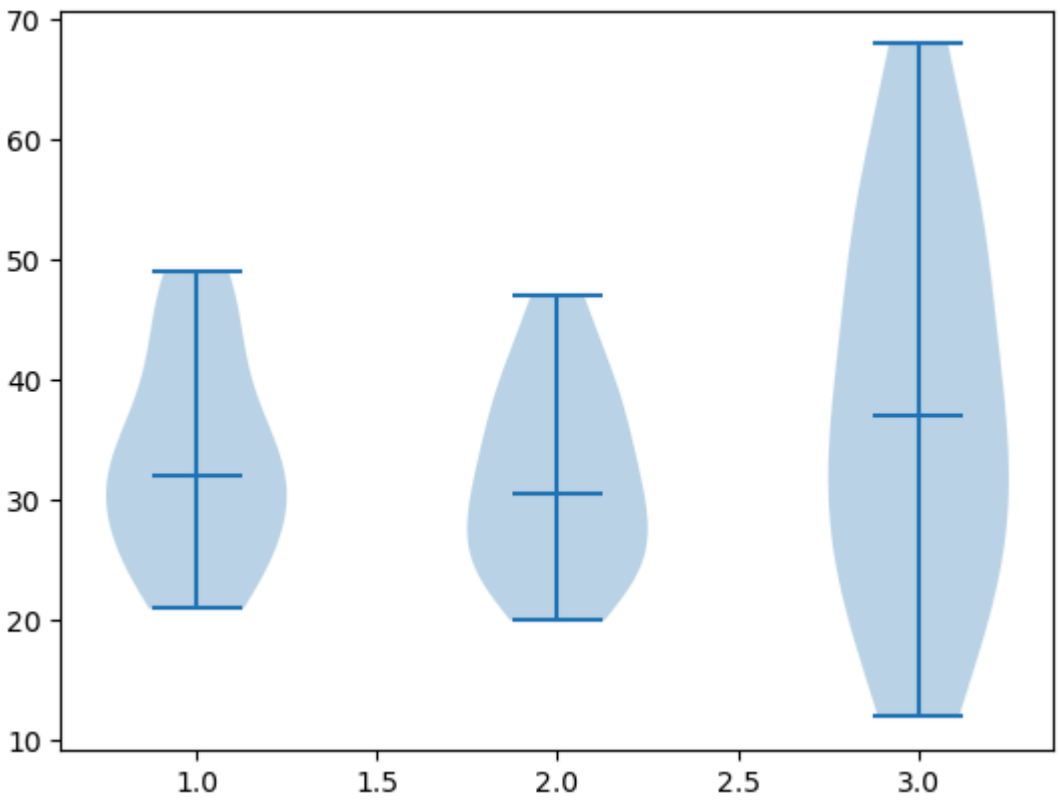
- Shows distribution: Provides a detailed view of the data distribution, including peaks, valleys, and skewness.
- Comparison: Effective for comparing distributions across different groups.
- Outlier detection: Can help identify potential outliers.

When to use a violin plot:

- When you want to visualize the distribution of numerical data.
- When comparing distributions between multiple groups.
- To complement box plots by providing more detailed information.

In [237…

```
plt.violinplot(data, showmedians=True)
plt.show()
```



Pie Chart

A pie chart is a circular statistical graphic which is divided into slices to illustrate numerical proportions. In a pie chart, the arc length of each slice is proportional to the quantity it represents. All sectors will sum up to 100%.

Best used for:

- Showing the composition of a whole.
- Comparing the relative sizes of different categories.

Donut Chart

A donut chart is similar to a pie chart, but with a hole in the center. This allows for additional information to be displayed in the center, such as totals or averages.

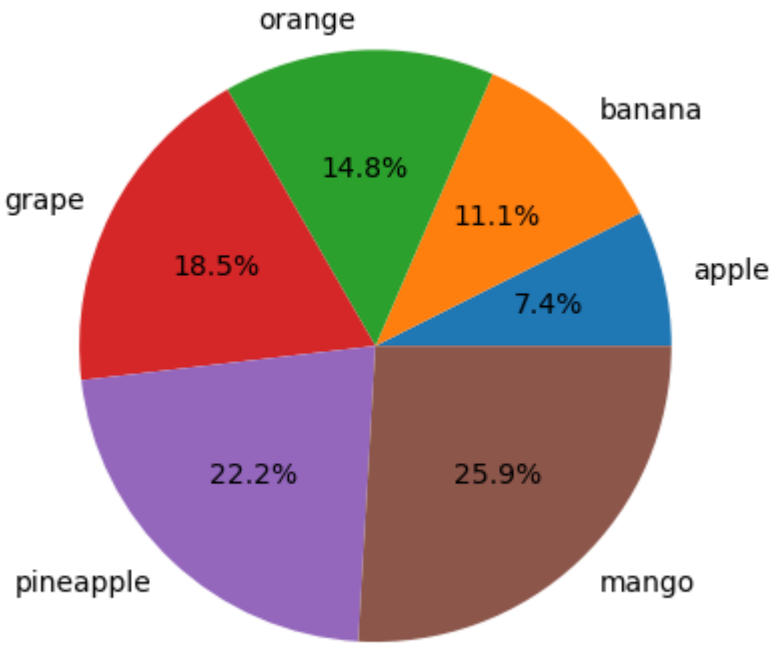
Best used for:

- The same purposes as a pie chart, with the added benefit of displaying additional information in the center.

Important note: While pie and donut charts are visually appealing, they can be difficult to interpret when there are too many categories. In such cases, other chart types like bar charts or column charts might be more effective.

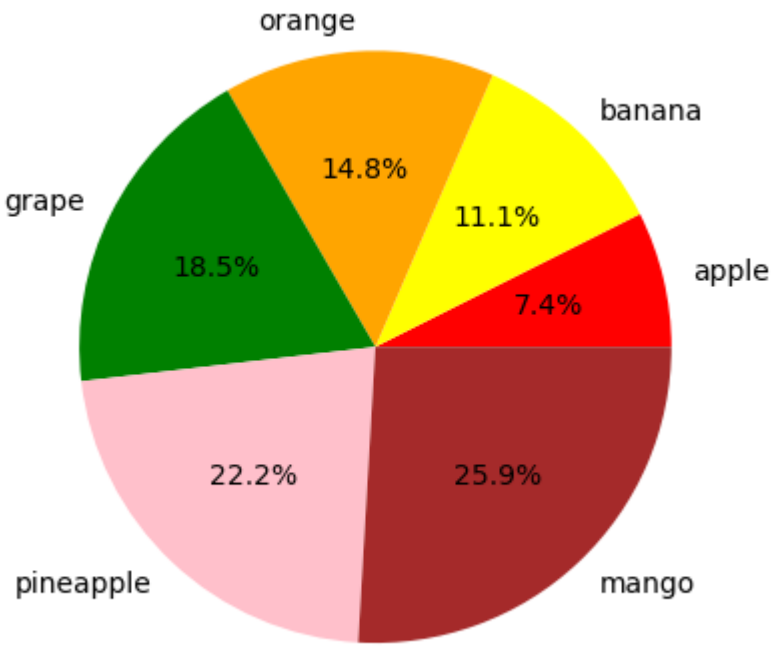
In [238…

```
# Pie-Chart
#Dictionary of fruits and their quantity
fruits = {'apple': 10, 'banana': 15, 'orange': 20, 'grape': 25, 'pineappl

# Create a pie chart
plt.pie(fruits.values(), labels=fruits.keys(), autopct='%1.1f%%')
plt.show()
```



In [239…

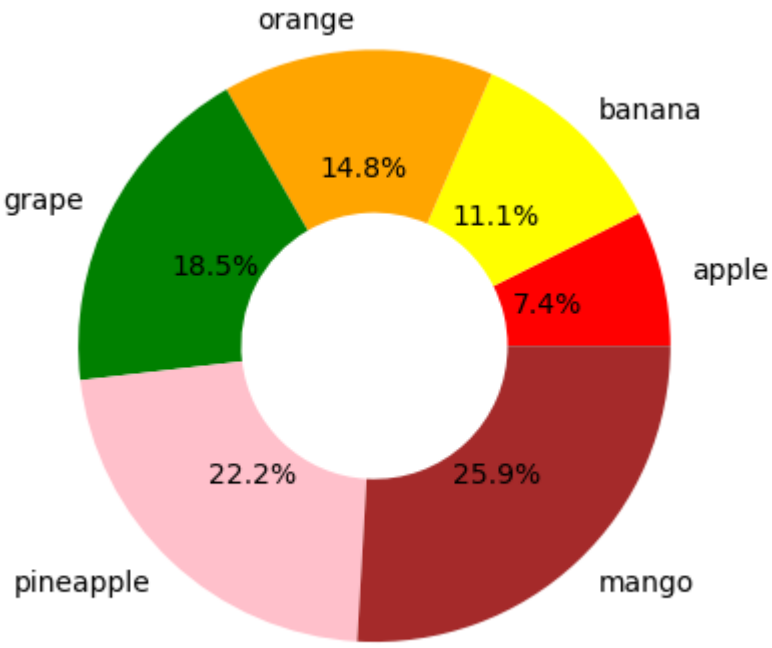
```
# Pie-Chart
#Dictionary of fruits and their quantity
fruits = {'apple': 10, 'banana': 15, 'orange': 20, 'grape': 25, 'pineappl
# Dictinary of colors for above furits
colors = {'apple': 'red', 'banana': 'yellow', 'orange': 'orange', 'grape'

# Create a pie chart
plt.pie(fruits.values(), labels=fruits.keys(), autopct='%1.1f%%', colors=
plt.show()
```



In [240…

```
# Donughnut-Chart
#Dictionary of fruits and their quantity
```

```
fruits = {'apple': 10, 'banana': 15, 'orange': 20, 'grape': 25, 'pineappl
# Dictinary of colors for above furits
colors = {'apple': 'red', 'banana': 'yellow', 'orange': 'orange', 'grape'

# Create a donughnut chart
plt.pie(fruits.values(), labels=fruits.keys(), autopct='%1.1f%%', colors=
plt.show()
```



Seaborn

Seaborn is built on top of matplotlib hence , we need to import both seaborn and matplotlib to use seaborn

Seaborn: A Python Visualization Library

Seaborn is a Python data visualization library built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics.

Key Features:

- High-level interface: Simplifies the process of creating complex visualizations.
- Statistical graphics: Specializes in creating plots that effectively communicate statistical insights.
- Attractive defaults: Offers visually appealing styles and color palettes.
- Integration with Pandas: Works seamlessly with Pandas DataFrames.

Why use Seaborn?

- Explore data: Quickly visualize relationships and patterns within your dataset.

- Communicate findings: Create visually compelling plots to share insights with others.
- Save time: Benefit from pre-built functions and customization options.

Common Plot Types:

- Categorical plots: Countplot, barplot, boxplot, violinplot, swarm plot, catplot
- Distribution plots: Distplot, histplot, kdeplot, rugplot
- Relational plots: Scatterplot, lineplot, relplot
- Matrix plots: Heatmap, clustermap, pairplot
- By leveraging Seaborn's capabilities, you can effectively explore and understand your data through visually appealing and informative plots.

In [241…
```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
```

Seaborn Lineplot

Seaborn's lineplot function is used to visualize relationships between variables where one of the variables is continuous. It's particularly useful for visualizing trends over time or other continuous variables.

Key Features and Parameters:

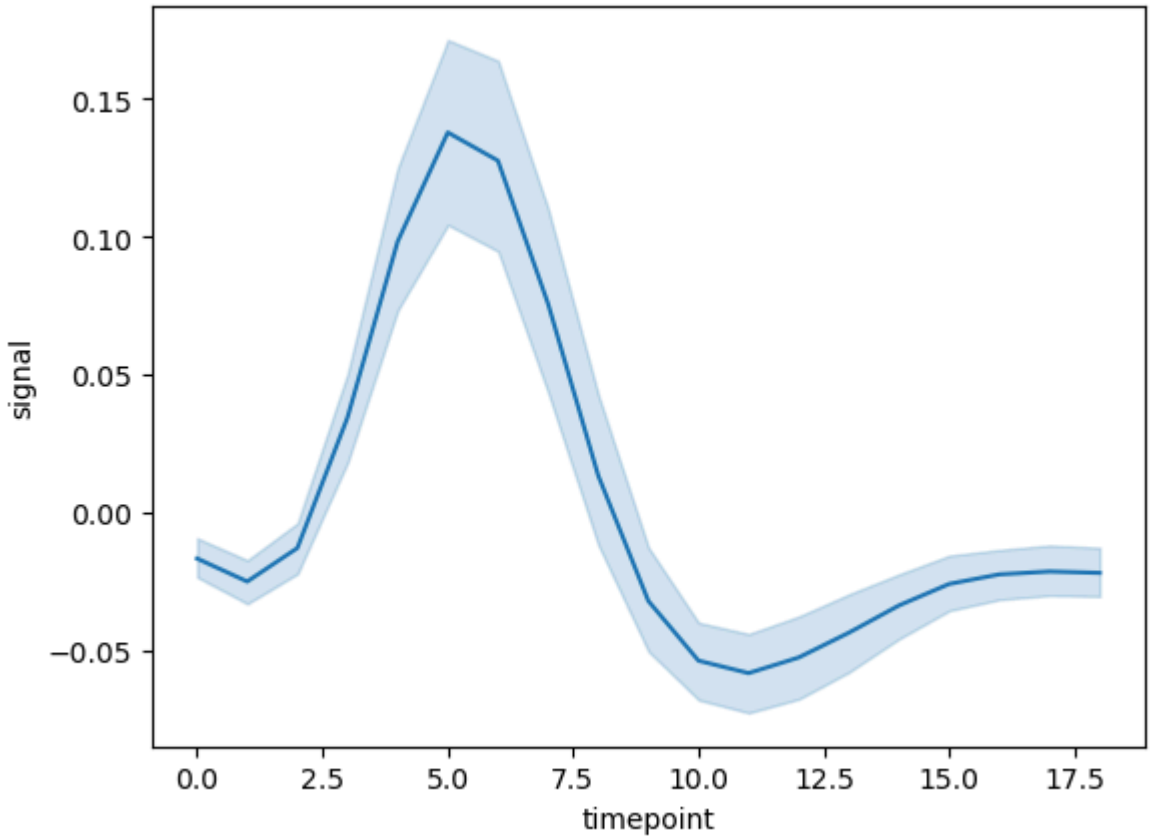
- x and y: Specify the variables for the x and y axes.
- data: The dataset to be used.
- hue: Separates observations into multiple lines based on a categorical variable.
- style: Uses different line styles to distinguish between categories.
- size: Adjusts the thickness of lines based on a numerical variable.
- markers: Adds markers to the line plot.
- ci: Controls the confidence interval around the line.

Additional Customization:

- Color palettes: Use palette parameter to set custom colors.
- Line styles: Control line styles using style parameter.
- Marker styles: Customize marker appearance with markers parameter.
- Axes labels and title: Add labels and titles using plt.xlabel, plt.ylabel, and plt.title.
- By effectively utilizing Seaborn's lineplot function and its parameters, you can create informative and visually appealing line plots to explore your data.

In [242…
```
# Line Plot
fmri = sns.load_dataset("fmri")
print(fmri.head())
sns.lineplot(x="timepoint", y="signal", data=fmri)
plt.show()
```

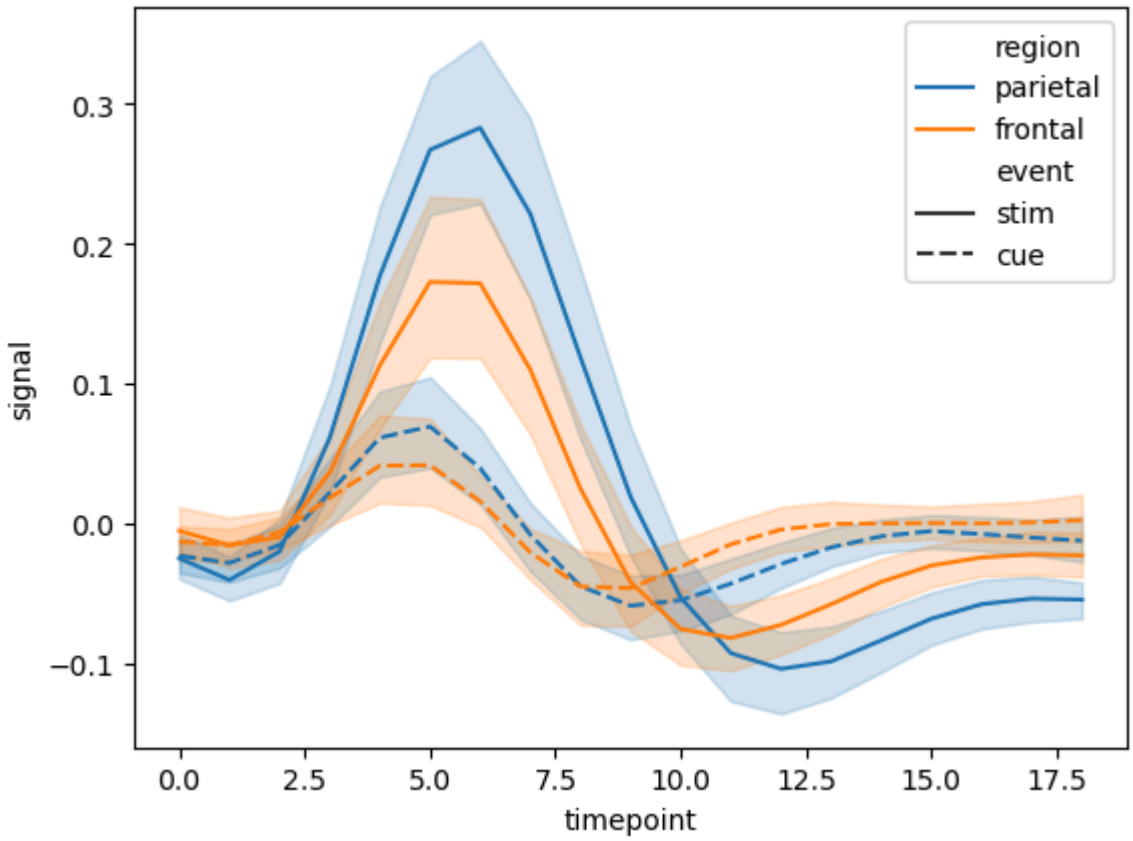
```
  subject  timepoint event    region    signal
0     s13         18  stim  parietal -0.017552
1      s5         14  stim  parietal -0.080883
2     s12         18  stim  parietal -0.081033
3     s11         18  stim  parietal -0.046134
4     s10         18  stim  parietal -0.037970
```



In [243…

```
# Line Plot
fmri = sns.load_dataset("fmri")
print(fmri.head())
sns.lineplot(x="timepoint", y="signal", data=fmri , hue="region", style="
plt.show()
```

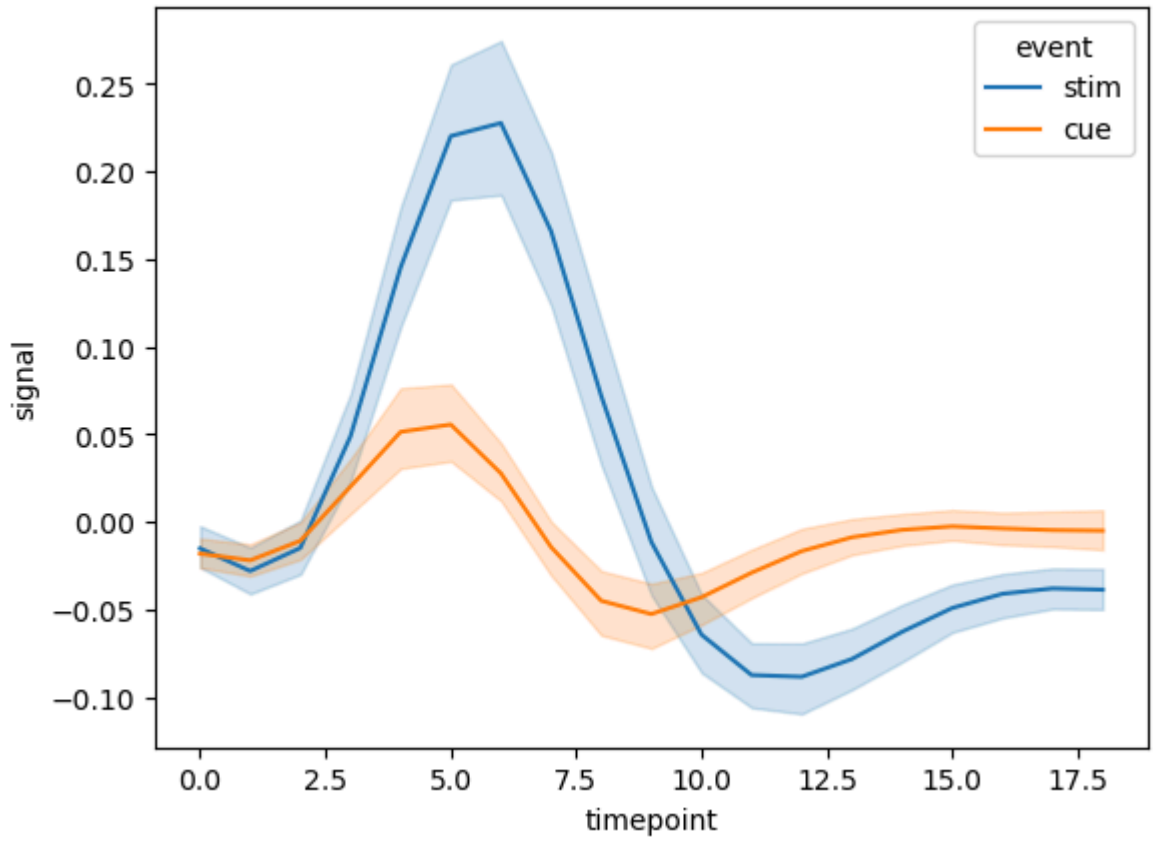
```
  subject  timepoint event    region    signal
0     s13         18  stim  parietal -0.017552
1      s5         14  stim  parietal -0.080883
2     s12         18  stim  parietal -0.081033
3     s11         18  stim  parietal -0.046134
4     s10         18  stim  parietal -0.037970
```



In [244…

```
# Line Plot
fmri = sns.load_dataset("fmri")
print(fmri.head())
sns.lineplot(x="timepoint", y="signal", data=fmri , hue="event")
plt.show()
```

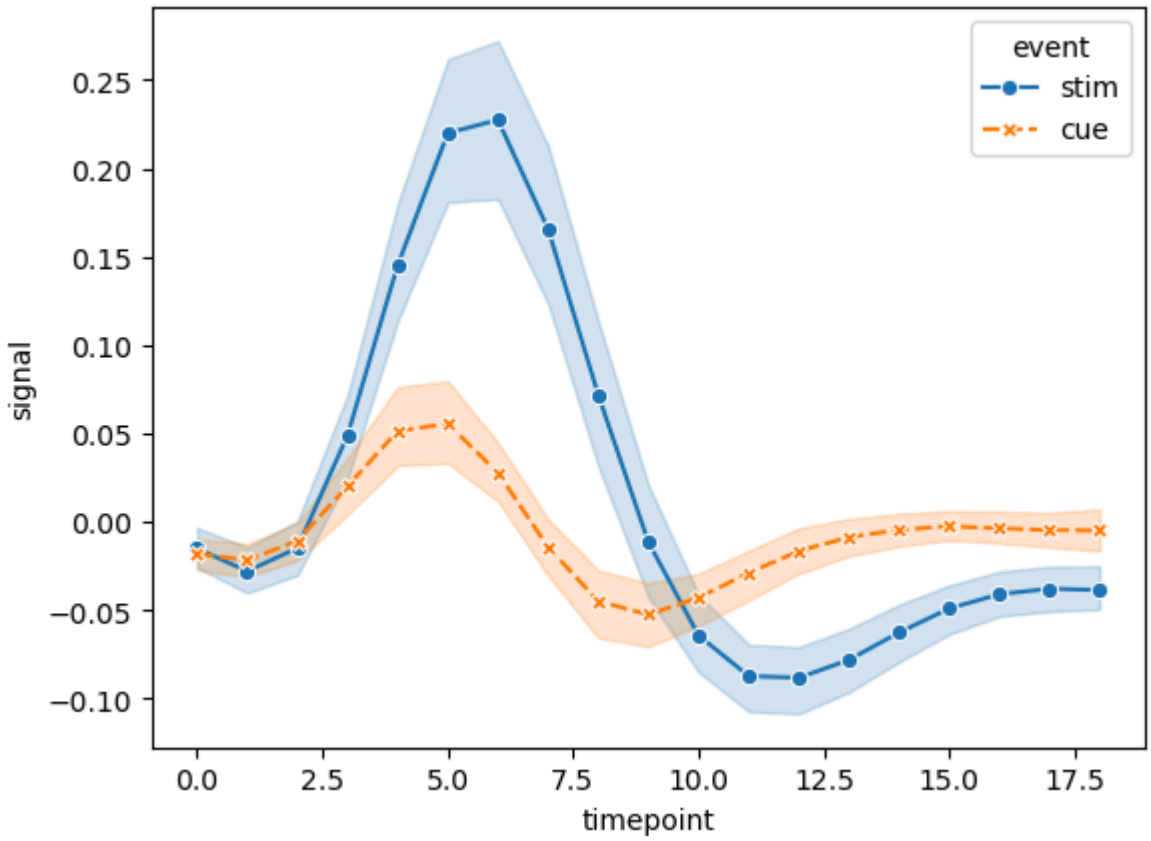
```
  subject  timepoint event    region    signal
0     s13         18  stim  parietal -0.017552
1      s5         14  stim  parietal -0.080883
2     s12         18  stim  parietal -0.081033
3     s11         18  stim  parietal -0.046134
4     s10         18  stim  parietal -0.037970
```



In [245…

```
# Line Plot
fmri = sns.load_dataset("fmri")
print(fmri.head())
sns.lineplot(x="timepoint", y="signal", data=fmri , hue="event", style="e
plt.show()
```

```
  subject  timepoint event    region    signal
0     s13         18  stim  parietal -0.017552
1      s5         14  stim  parietal -0.080883
2     s12         18  stim  parietal -0.081033
3     s11         18  stim  parietal -0.046134
4     s10         18  stim  parietal -0.037970
```



Seaborn Bar Plot

Seaborn's barplot function is used to visualize the relationship between a categorical variable and one or more continuous numerical variables. It represents an estimate of central tendency for a numeric variable with the height of each rectangle and indicates the uncertainty around that estimate using error bars.

Key Parameters:

- x: The name of the categorical variable to be used on the x-axis.
- y: The name of the numerical variable to be aggregated and visualized.
- data: The dataset to be used.
- hue: Optional categorical variable for creating multiple bars within each category.
- estimator: Function to aggregate the y-value (default is np.mean).
- ci: Confidence interval to display (default is 95%).
- palette: Color palette for the bars.
- errorbar: Type of error bar to display (default is 'ci').

Additional Customization:

- orient: To create horizontal bar plots.
- saturation: Adjust the color saturation.
- ax: To plot on a specific matplotlib axes.

- Other aesthetic parameters from Matplotlib can be used for further customization.

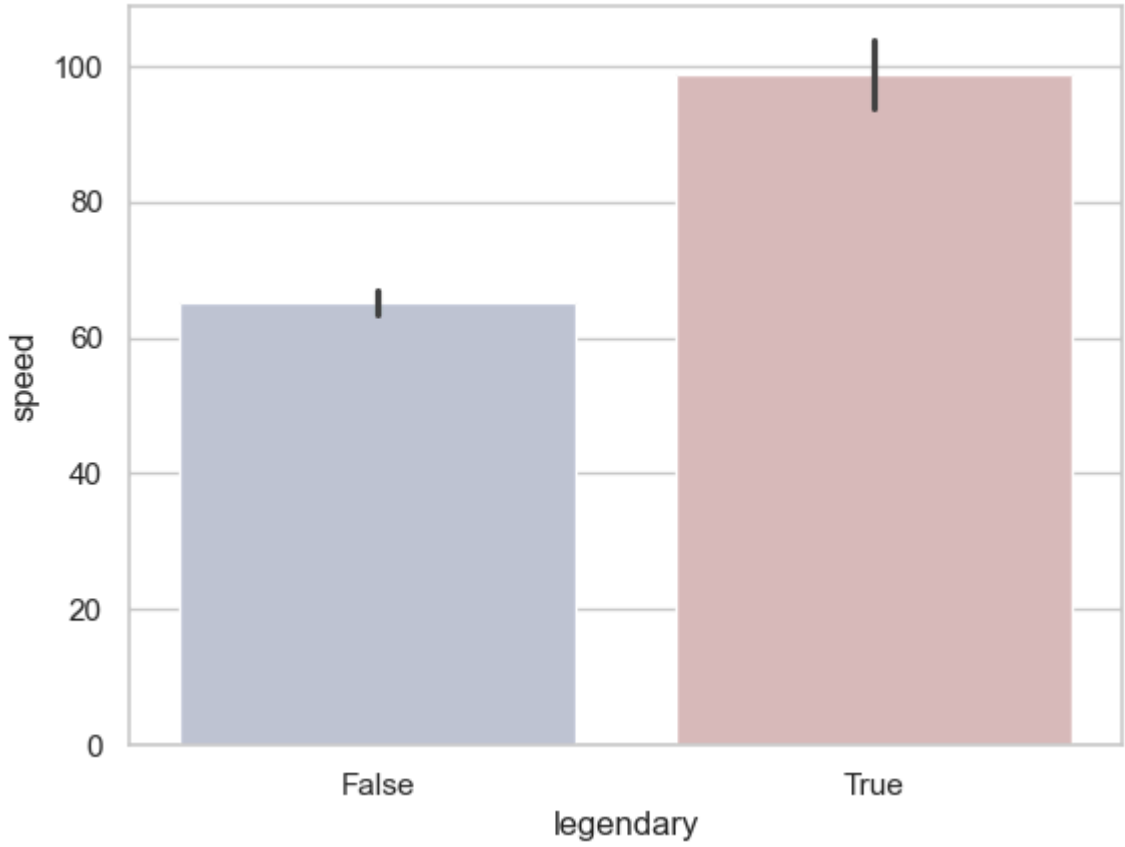
Seaborn's bar plot is a versatile tool for visualizing categorical data and understanding relationships between categorical and numerical variables. By exploring the different parameters and customization options, you can create informative and visually appealing bar plots.

In [246…

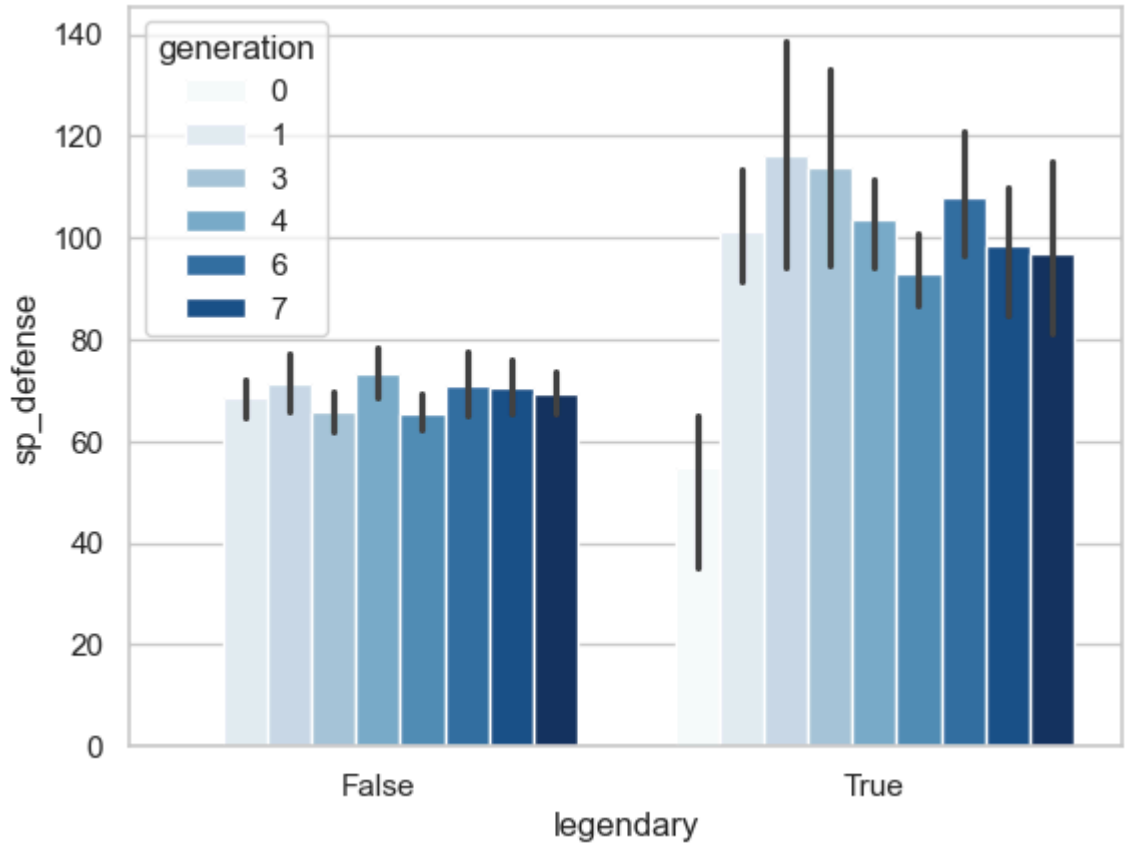
```
sns.set_theme(style="whitegrid")
pokemon = pd.read_csv('data/pokemon.csv')
print(pokemon.head())
sns.barplot(x="legendary", y="speed", data=pokemon, palette="vlag")
plt.show()
```

```
   number                 name  type1   type2  total  hp  attack  defense
\
0       1            Bulbasaur  Grass  Poison    318  45      49       49
1       2              Ivysaur  Grass  Poison    405  60      62       63
2       3             Venusaur  Grass  Poison    525  80      82       83
3       3        Mega Venusaur  Grass  Poison    625  80     100      123
4       3  Gigantamax Venusaur  Grass  Poison    525  80      82       83

   sp_attack  sp_defense  speed  generation  legendary
0         65          65     45           1      False
1         80          80     60           1      False
2        100         100     80           1      False
3        122         120     80           1      False
4        100         100     80           1      False
```

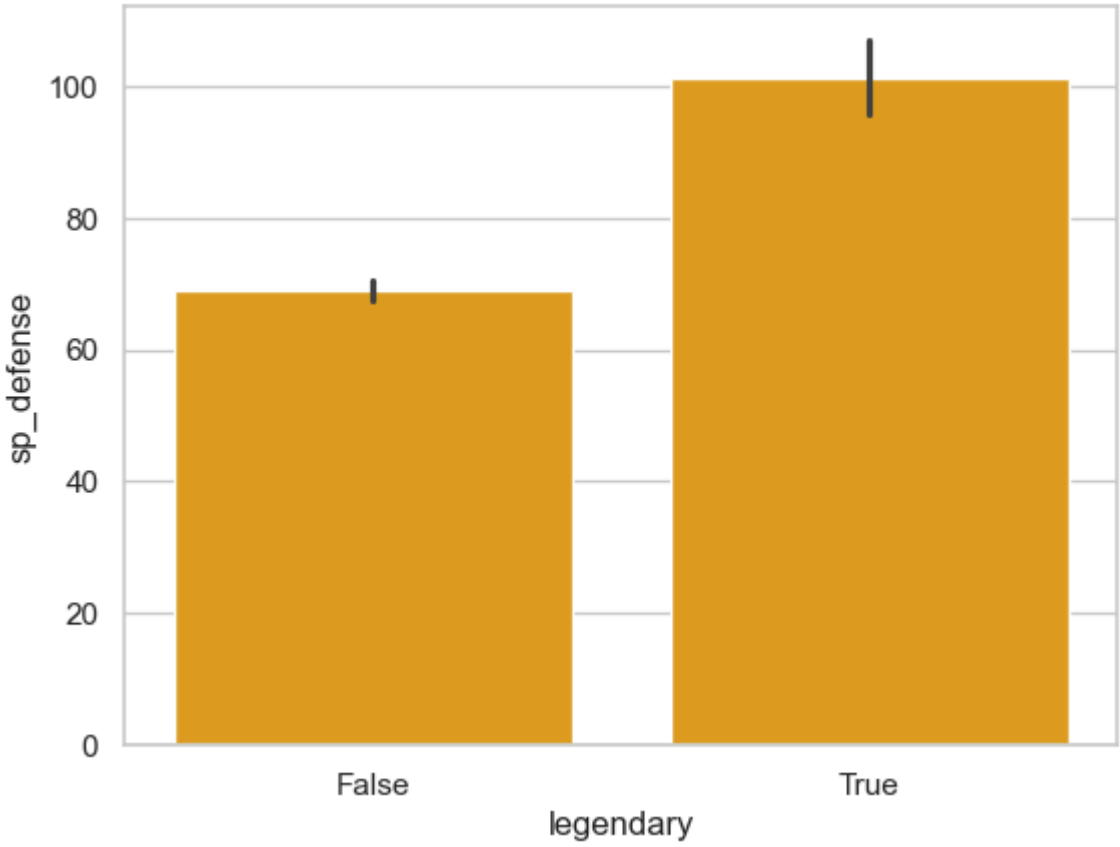
```
/var/folders/8h/zprf7hjs319_78816p34b90c0000gn/T/ipykernel_55511/296811553
2.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.barplot(x="legendary", y="speed", data=pokemon, palette="vlag")
```



In [247…
```
sns.barplot(x="legendary", y="sp_defense",hue="generation", data=pokemon,
plt.show()
```



In [248…
```
sns.barplot(x="legendary", y="sp_defense", data=pokemon, color="orange")
plt.show()
```



In [249…
```
iris =pd.read_csv('data/iris.csv')
iris.head()
```

Out[249…

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Seaborn Scatterplot

Seaborn's scatterplot function is used to visualize the relationship between two numerical variables. Each data point is represented by a marker, and the position of the marker on the x and y axes corresponds to the values of the two variables.

Key Parameters:

- x: The name of the numerical variable to be used on the x-axis.
- y: The name of the numerical variable to be used on the y-axis.
- data: The dataset to be used.
- hue: Optional categorical variable for grouping data points by color.
- style: Optional categorical variable for grouping data points by marker style.
- size: Optional numerical variable for controlling the size of the markers.

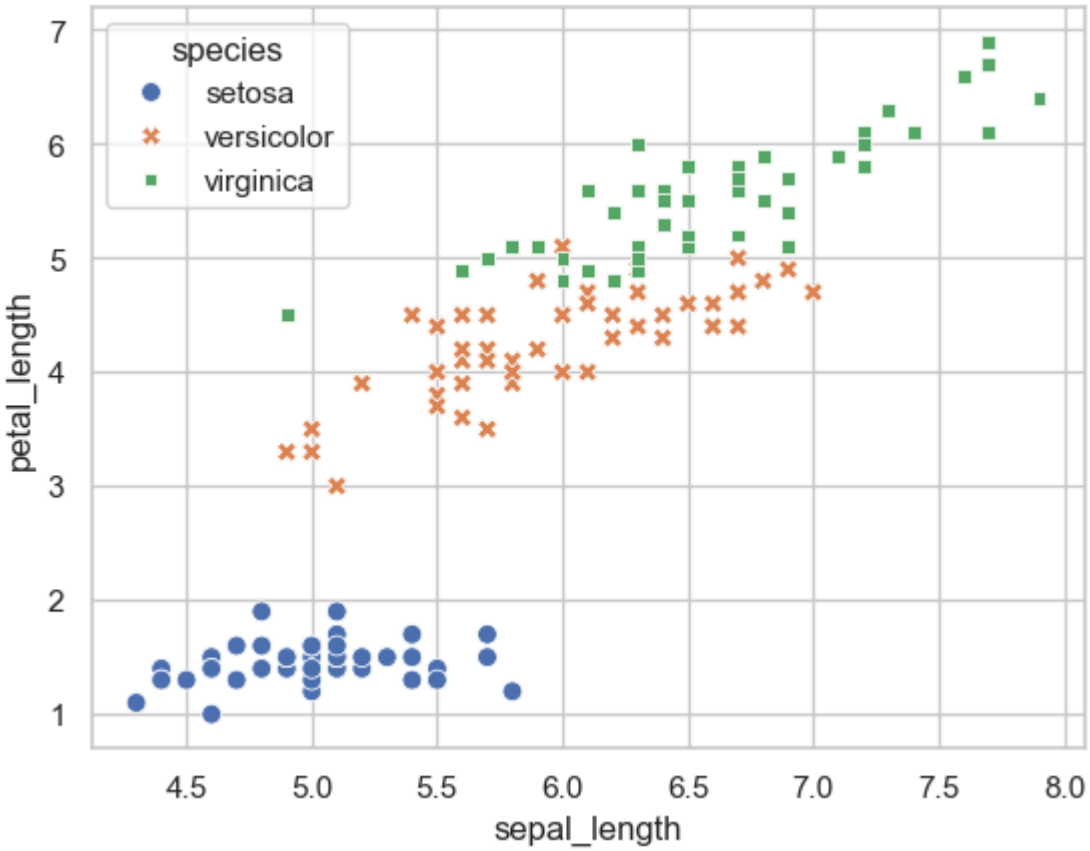
- palette: Color palette for the scatter plot.
- alpha: Transparency of the markers.

Additional Customization:

- markers: Customize the marker style.
- edgecolor: Set the color of the marker edges.
- linewidth: Adjust the width of the marker edges.
- s: Manually set the marker size.

Seaborn's scatterplot is a versatile tool for exploring relationships between numerical variables. By using the various parameters, you can create informative and visually appealing scatter plots to understand your data.

In [250…
```
sns.scatterplot(x="sepal_length", y="petal_length",  style="species", dat
plt.show()
```



Seaborn Histogram

Seaborn's histplot function is used to visualize the distribution of a numerical variable. It counts the number of observations that fall within discrete bins and represents this count with rectangular bars.

Key Parameters:

- data: The dataset to be used.

- x: The name of the numerical variable to be plotted.
- y: Optional variable for bivariate histograms.
- hue: Optional categorical variable for creating multiple histograms.
- bins: Number of bins for the histogram.
- kde: Whether to plot a kernel density estimate (KDE) curve.
- stat: The statistic to compute in each bin (e.g., 'count', 'density', 'probability').
- common_norm: Whether to normalize histograms when using hue.
- element: The visual representation of the data (e.g., 'bars', 'step', 'poly')

Additional Customization:

- Color palettes: Use the palette parameter to set custom colors.
- Binning: Control the number of bins with the bins parameter.
- Density estimation: Adjust the KDE bandwidth with kde_kws.
- Axes labels and title: Add labels and titles using plt.xlabel, plt.ylabel, and plt.title.

Seaborn's histplot is a versatile tool for understanding the distribution of your data. By combining it with other Seaborn functions and Matplotlib customizations, you can create informative and visually appealing histograms.

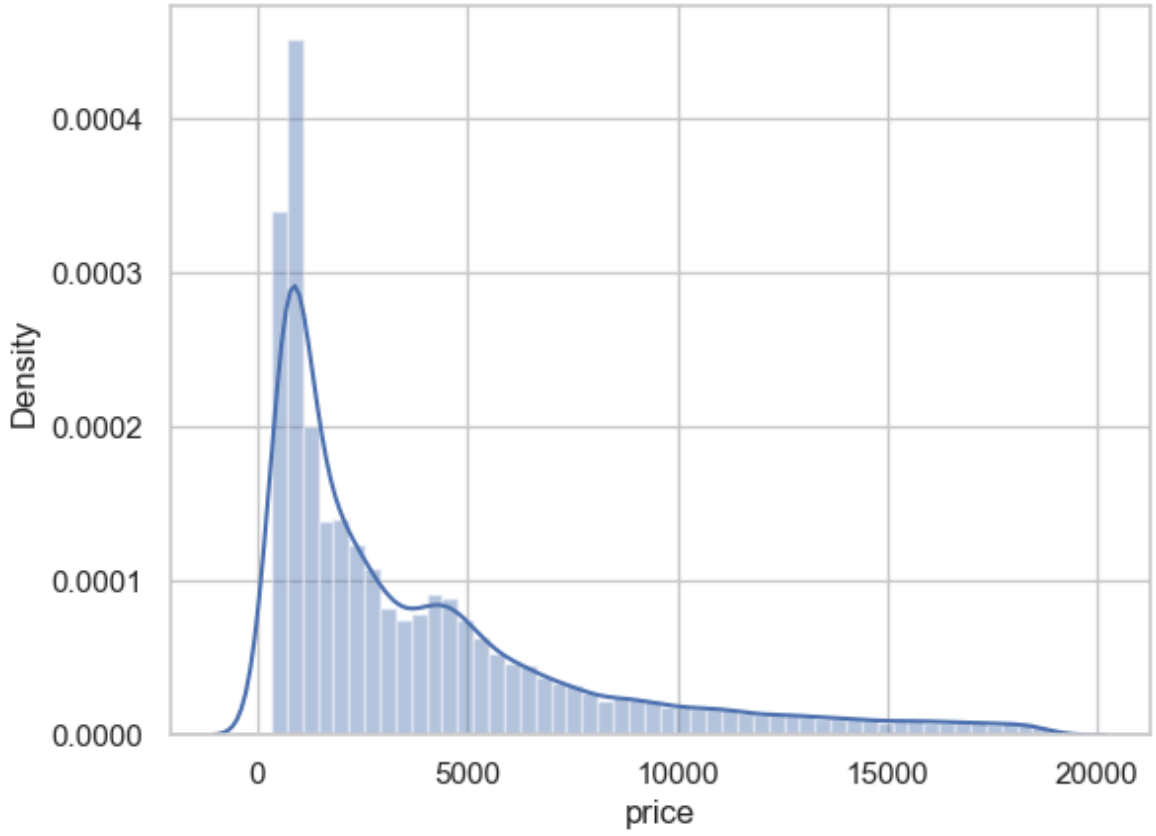
In [251…
```
# SNS histogram or Distribution plot
diamond = sns.load_dataset("diamonds")
diamond.head()
```

Out[251…

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

In [252…
```
sns.distplot(diamond['price'])
plt.show()
```

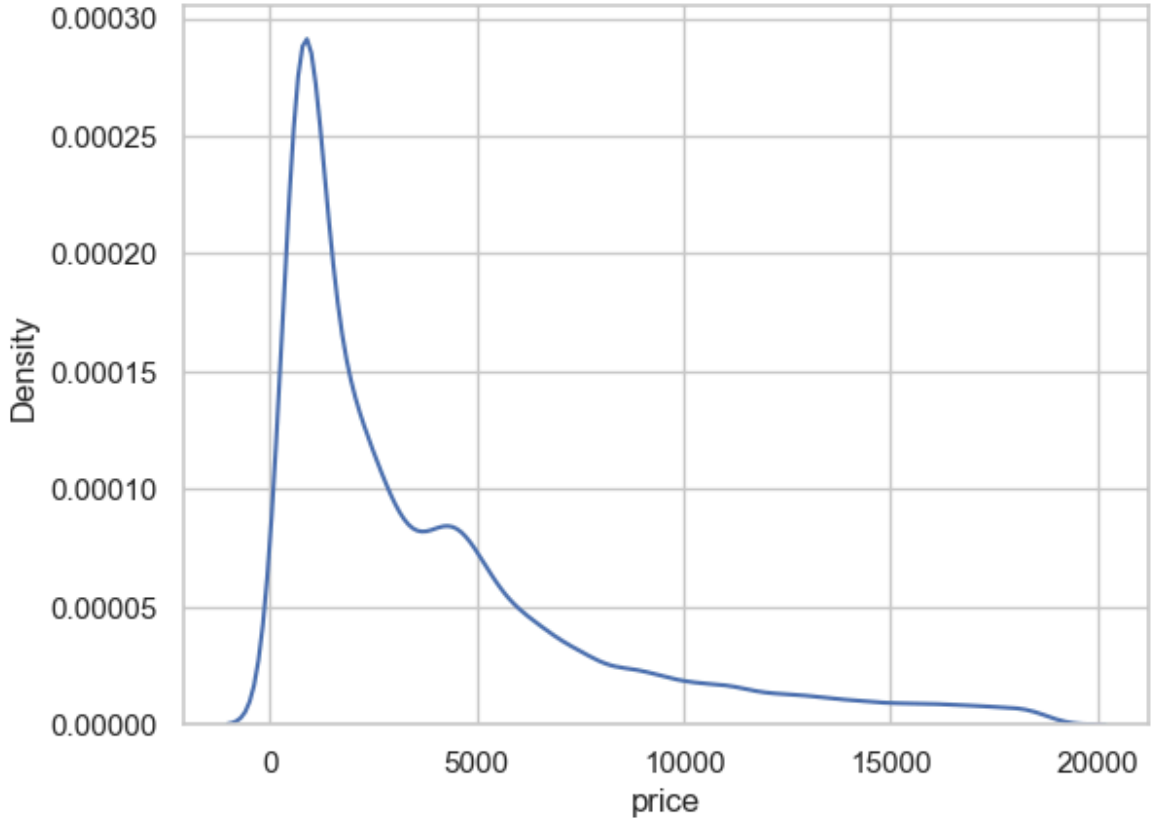
```
/var/folders/8h/zprf7hjs319_78816p34b90c0000gn/T/ipykernel_55511/425555065
2.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `histplot` (an axes-level function for histogram
s).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(diamond['price'])
```



In [253…

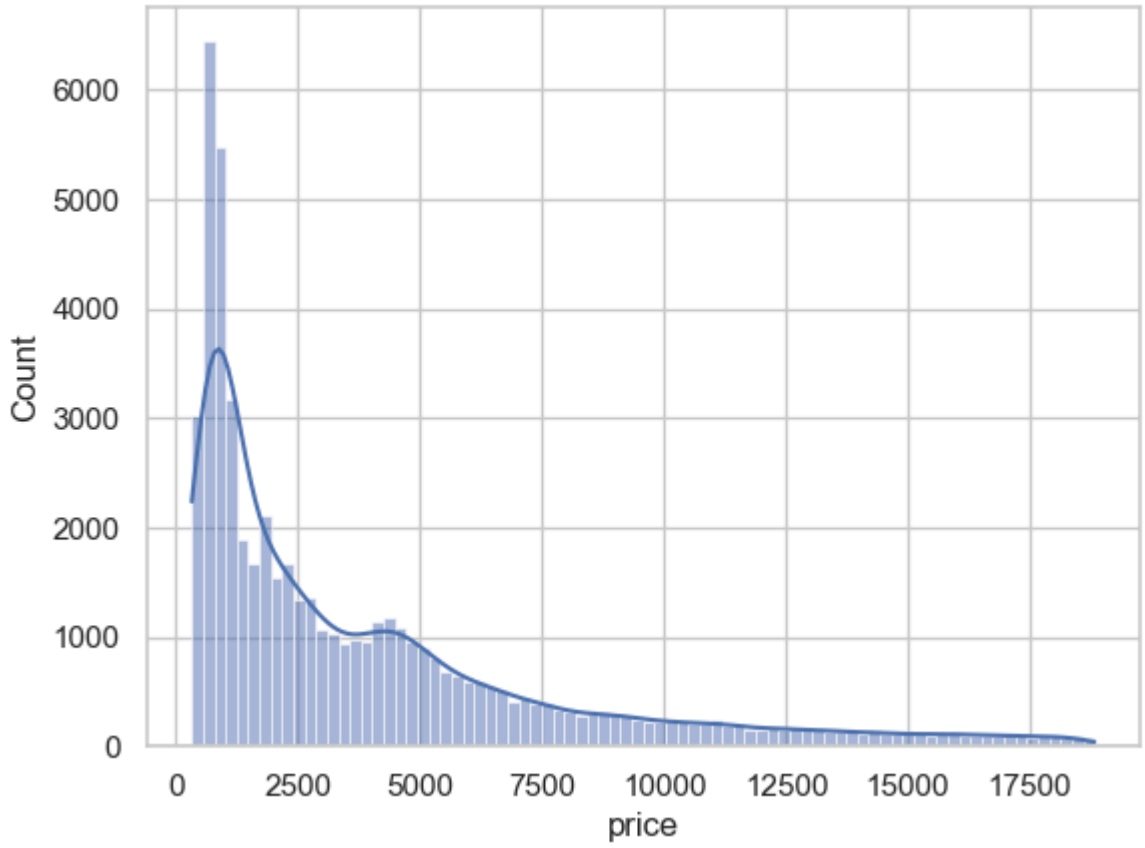
```
sns.distplot(diamond['price'], hist=False)
plt.show()
```

```
/var/folders/8h/zprf7hjs319_78816p34b90c0000gn/T/ipykernel_55511/41812979
0.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `kdeplot` (an axes-level function for kernel densi
ty plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(diamond['price'], hist=False)
```



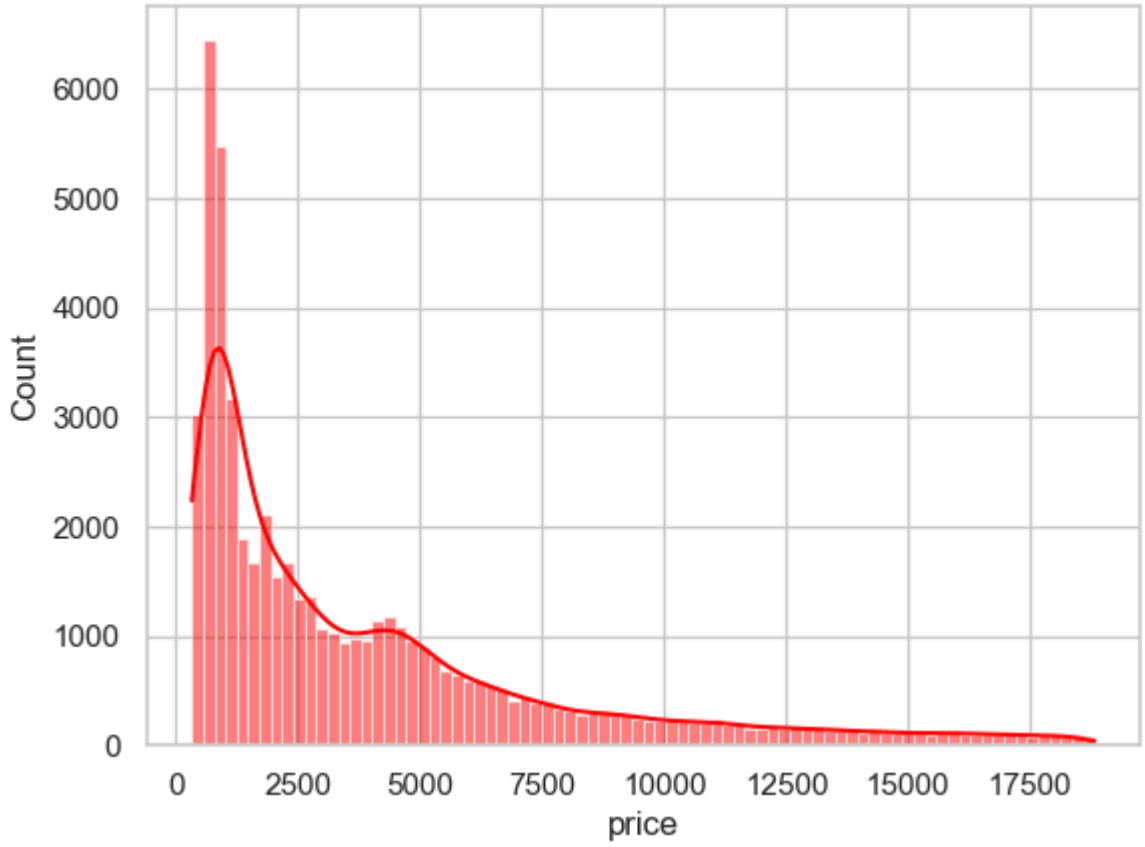
In [254…
```
sns.histplot(x="price", data=diamond, kde=True)
```

Out[254… <Axes: xlabel='price', ylabel='Count'>



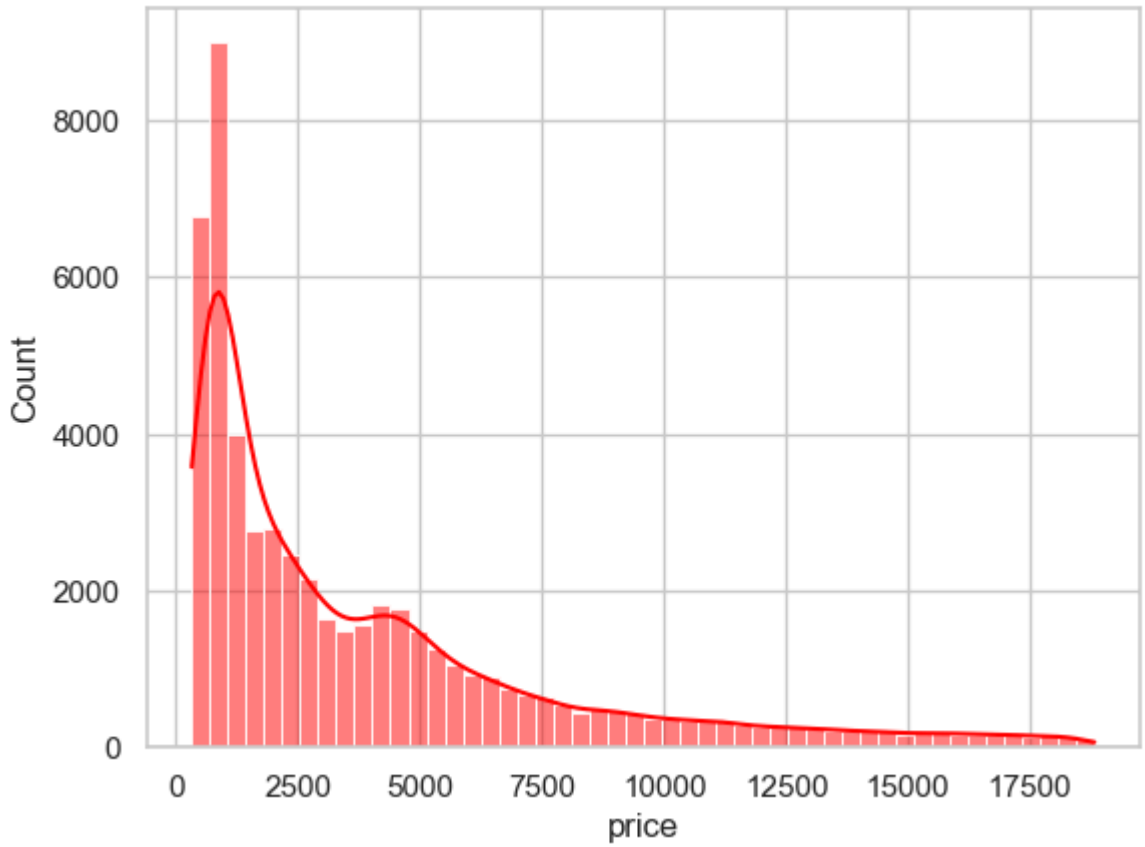
In [255…
```
sns.histplot(diamond['price'], kde=True, color="red")
```

Out[255… <Axes: xlabel='price', ylabel='Count'>



In [256…
```
sns.histplot(diamond['price'], kde=True, color="red", bins=50)
```

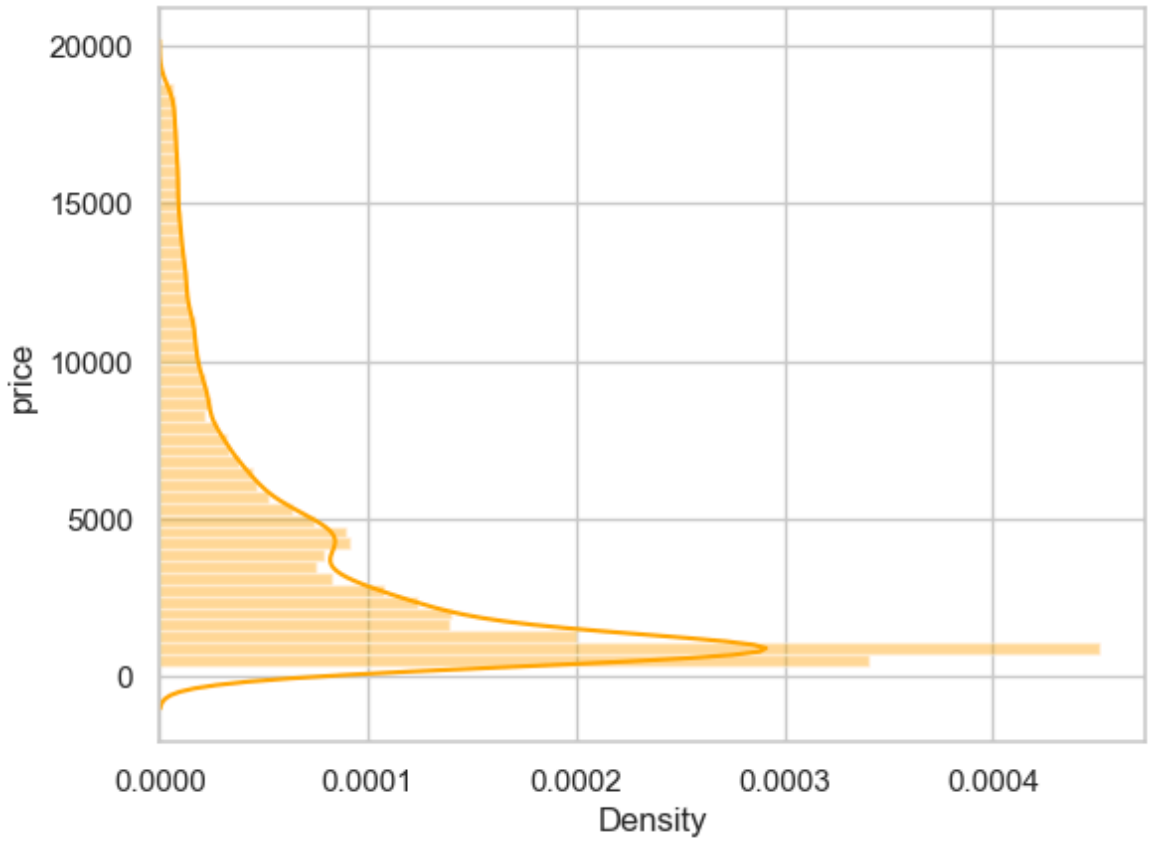
Out[256…
```
<Axes: xlabel='price', ylabel='Count'>
```



In [257…
```
sns.distplot(diamond['price'], kde=True, color="orange", bins=50, vertica
```

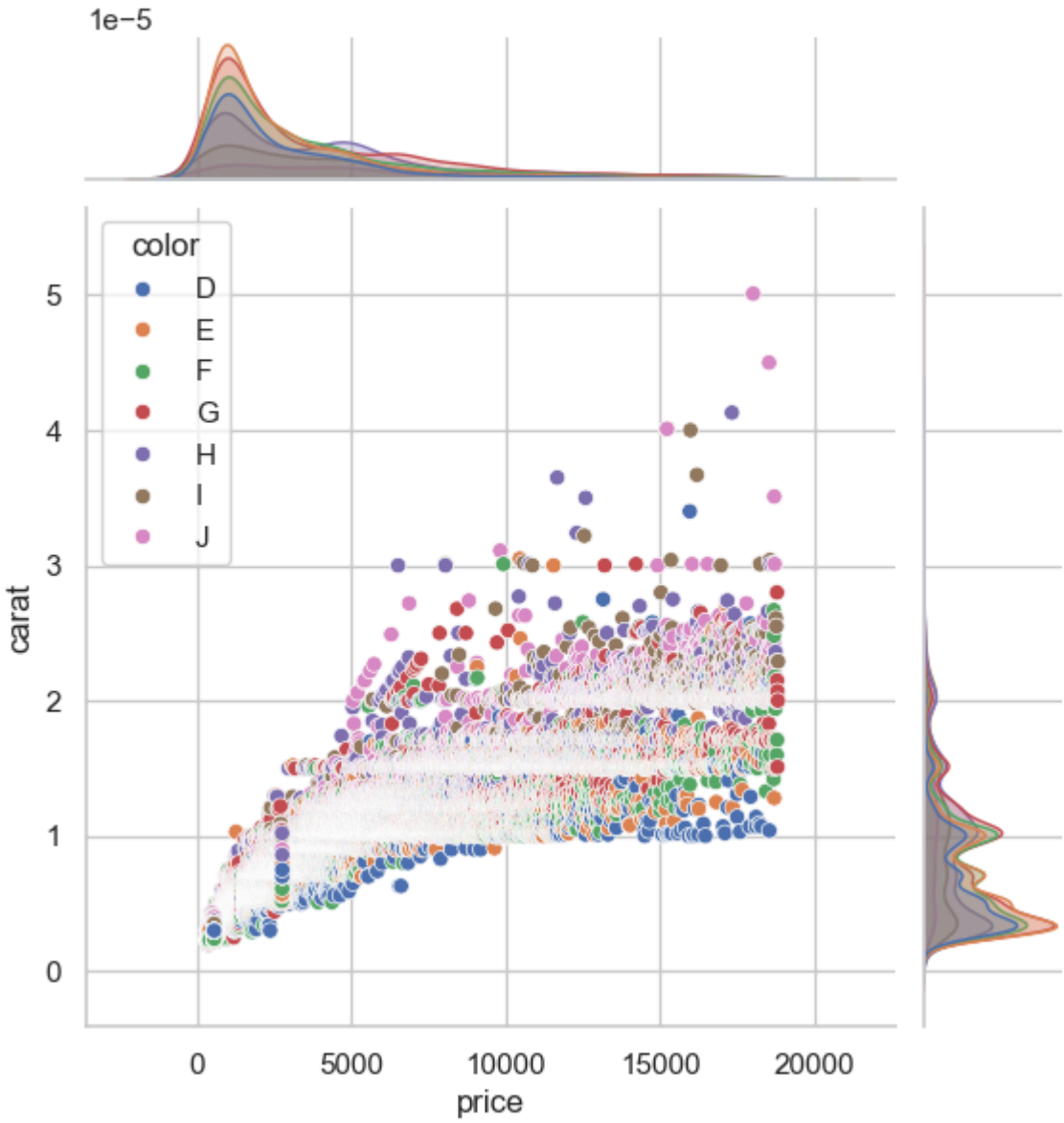
```
/var/folders/8h/zprf7hjs319_78816p34b90c0000gn/T/ipykernel_55511/97578017
6.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `histplot` (an axes-level function for histogram
s).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(diamond['price'], kde=True, color="orange", bins=50, vertic
al=True)
```

Out[257… <Axes: xlabel='Density', ylabel='price'>



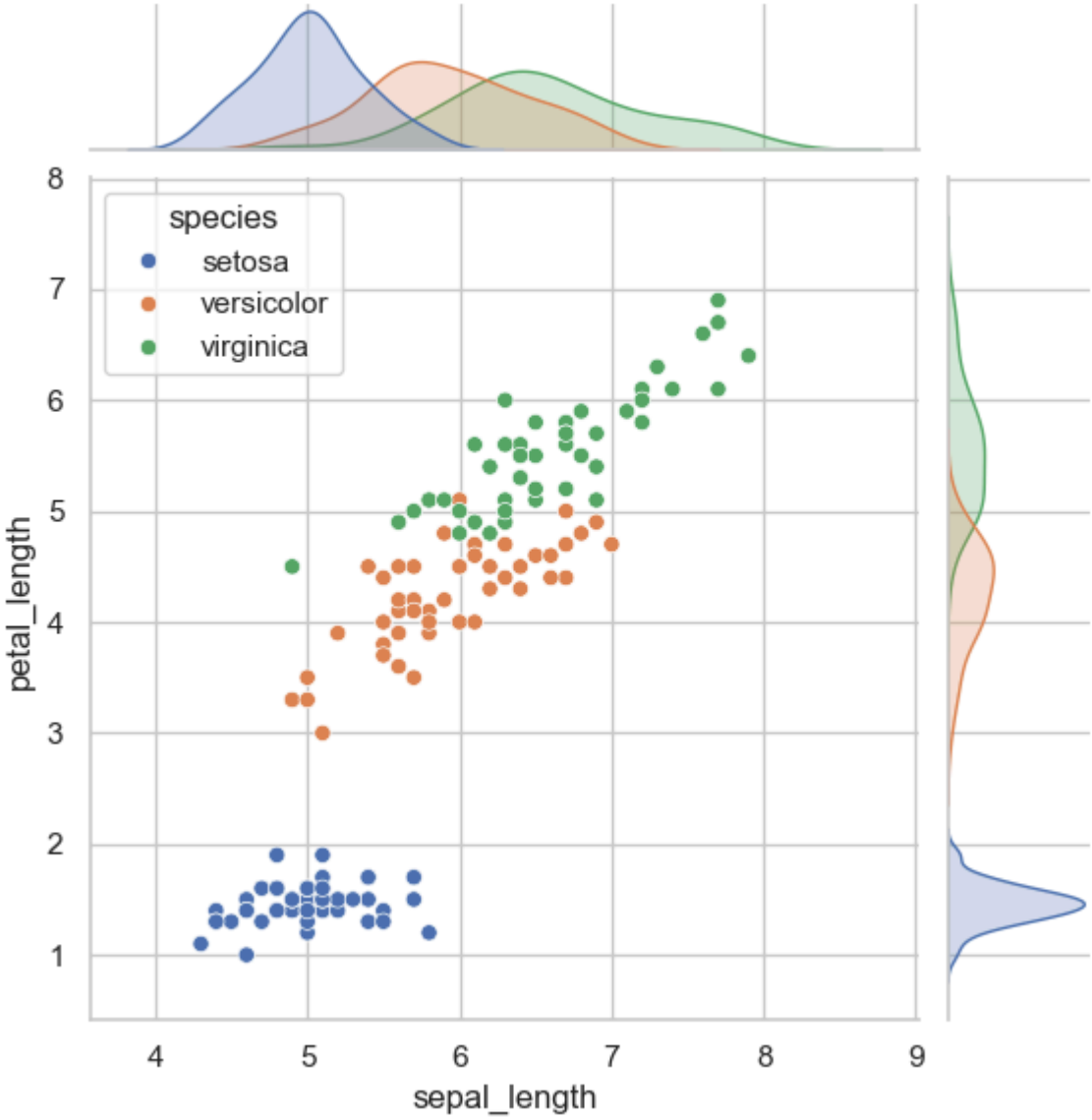
In [258…

```
##JointPlot

sns.jointplot(x="price", y="carat", data=diamond, hue="color")
plt.show()
```



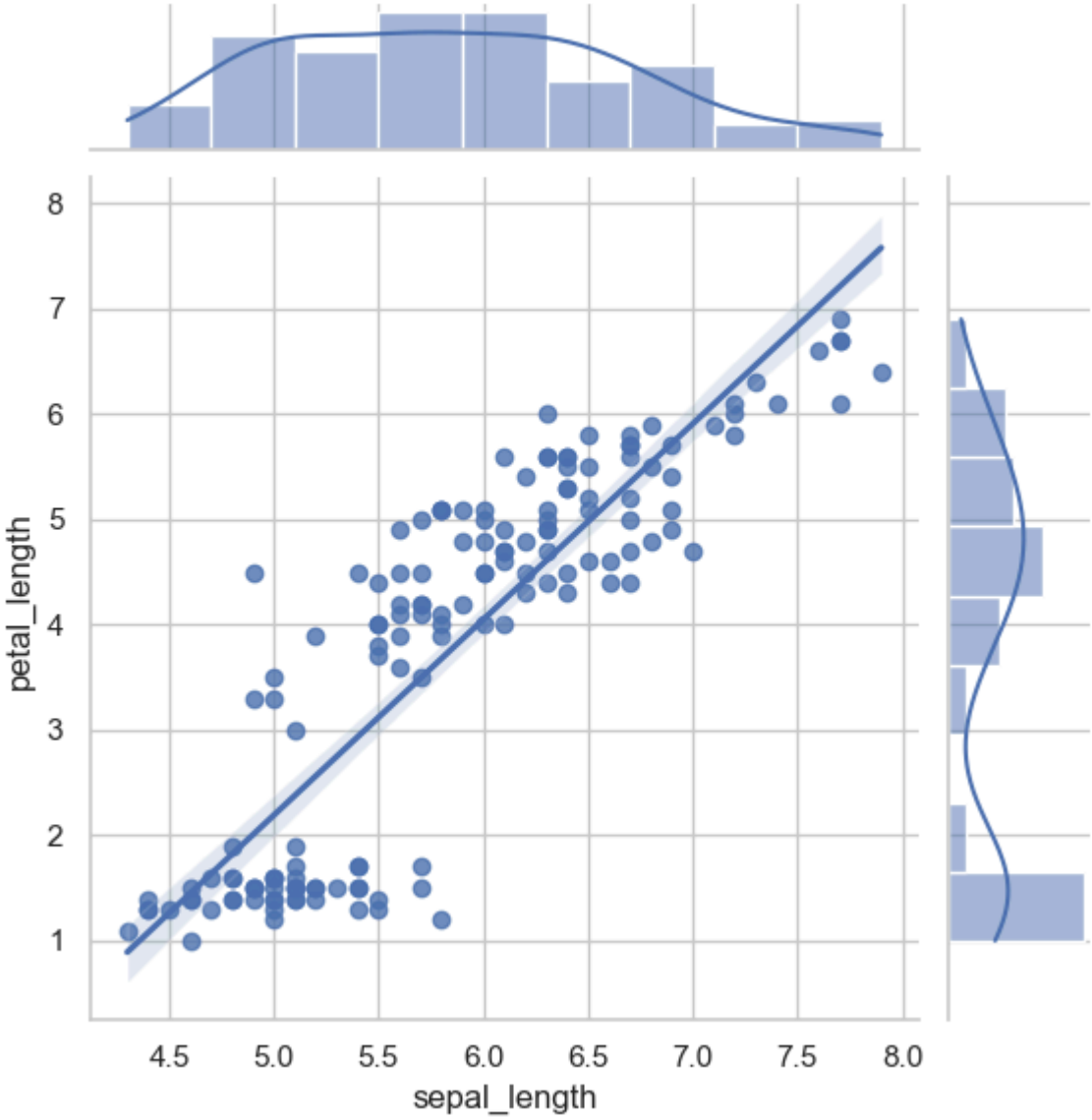
In [259…

```
iris = pd.read_csv('data/iris.csv')

sns.jointplot(x="sepal_length", y="petal_length", data=iris, hue="species
plt.show()
```



In [260…

```
sns.jointplot(x="sepal_length", y="petal_length", data=iris, kind="reg")
plt.show()
```



Seaborn Boxplot

Seaborn's boxplot function is used to visualize the distribution of numerical data across different categories. It provides a compact representation of the data, showing the median, quartiles, and potential outliers.

Key Parameters:

- x: The name of the categorical variable to be used on the x-axis.
- y: The name of the numerical variable to be visualized.
- data: The dataset to be used.
- hue: Optional categorical variable for creating multiple boxplots within each category.
- width: Width of the boxes.
- palette: Color palette for the boxes.
- whis: Determines the length of the whiskers.
- showfliers: Whether to show outliers.

Additional Customization:

- orient: To create horizontal boxplots.
- saturation: Adjust the color saturation.
- ax: To plot on a specific matplotlib axes.
- Other aesthetic parameters from Matplotlib can be used for further customization.

Seaborn's boxplot is a powerful tool for comparing distributions across different categories. By exploring the different parameters and customization options, you can create informative and visually appealing boxplots.

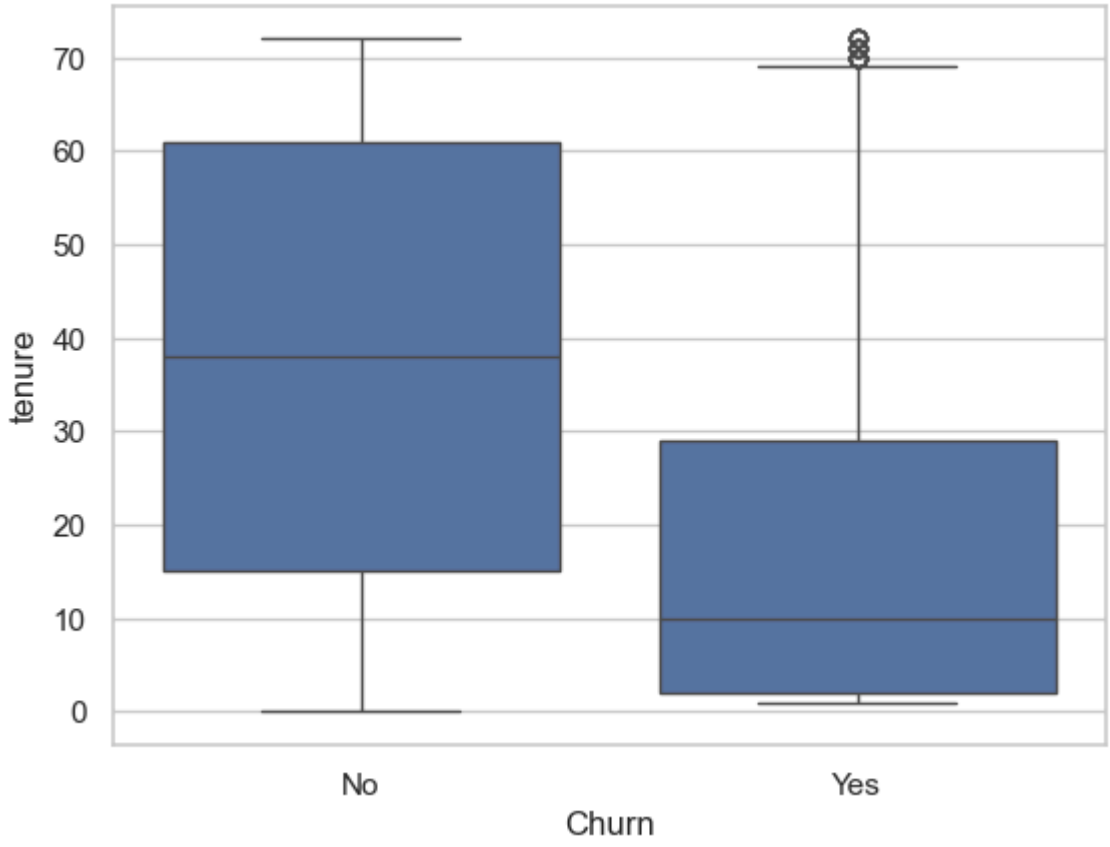
In [261…
```
# BOXPLOT
churn = pd.read_csv('data/Telco-Customer-Churn.csv')
churn.head()
```

Out[261…

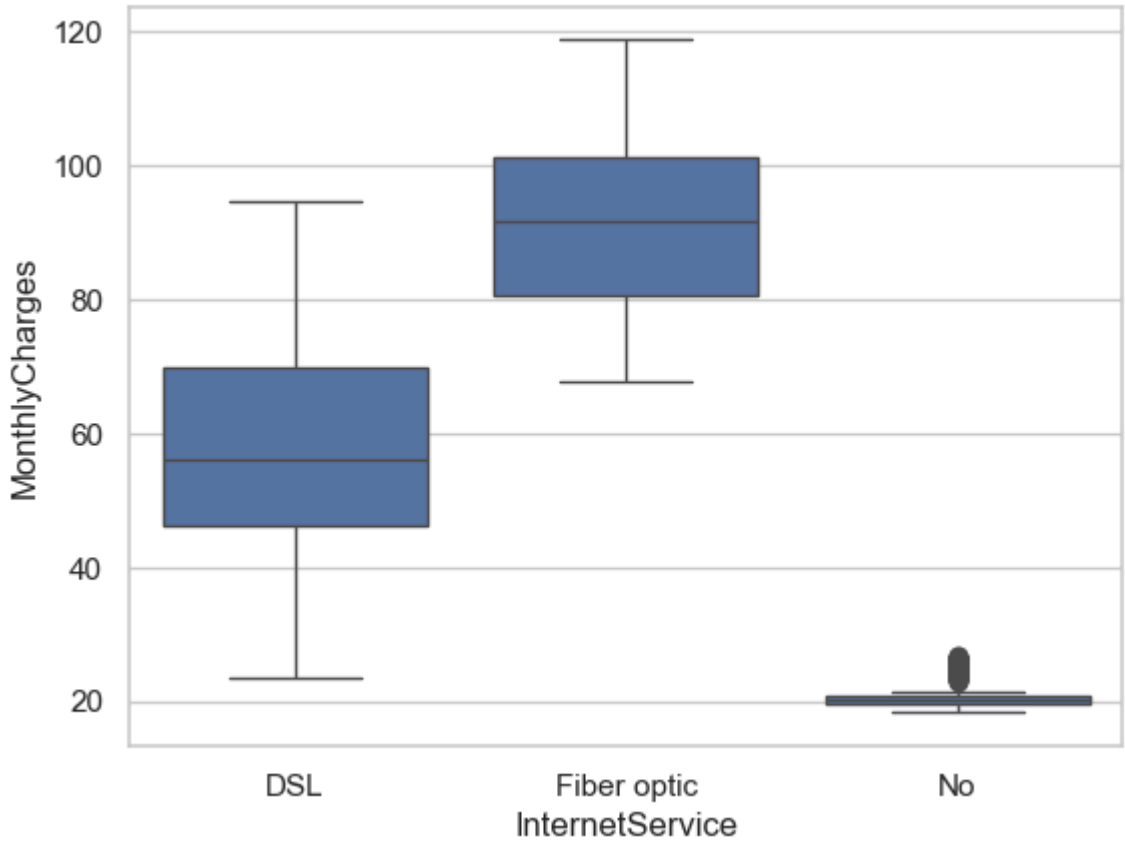
	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService
0	7590-VHVEG	Female	0	Yes	No	1	No
1	5575-GNVDE	Male	0	No	No	34	Yes
2	3668-QPYBK	Male	0	No	No	2	Yes
3	7795-CFOCW	Male	0	No	No	45	No
4	9237-HQITU	Female	0	No	No	2	Yes

5 rows × 21 columns

In [262…
```
sns.boxplot(x="Churn", y="tenure", data=churn)
plt.show()
```

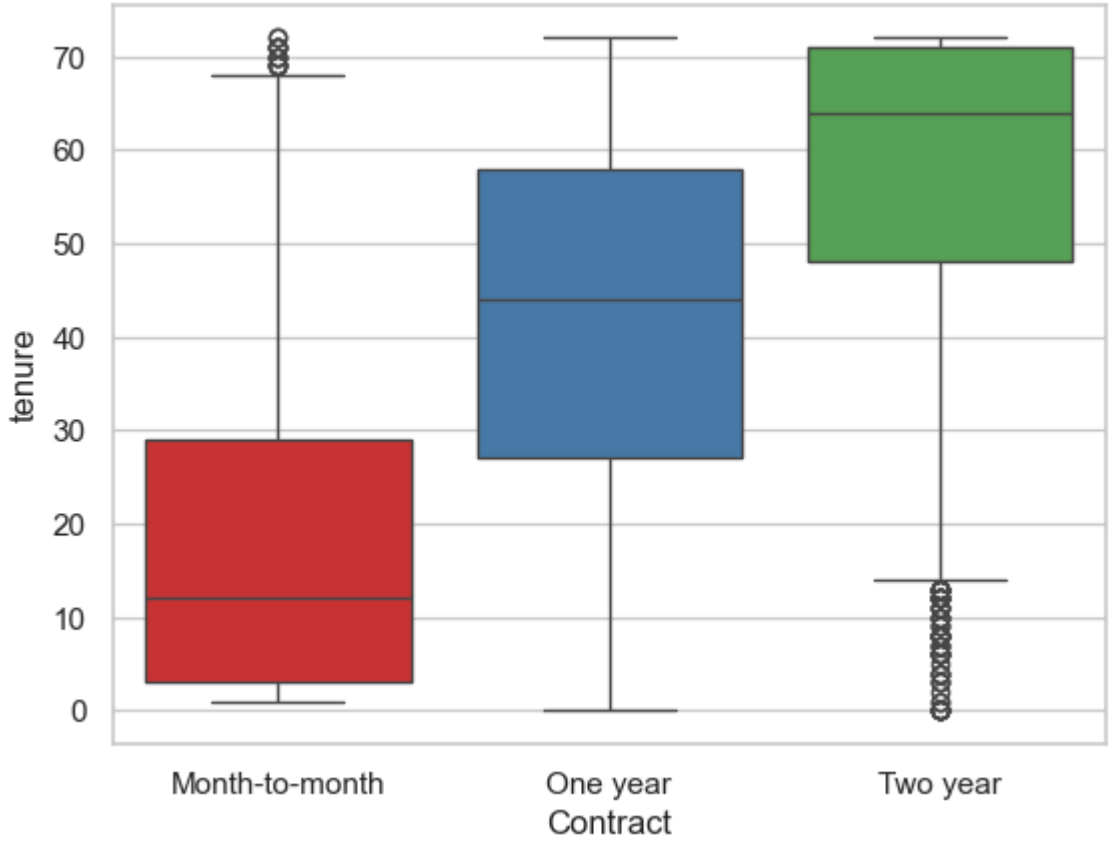


In [263…
```
sns.boxplot(x='InternetService', y='MonthlyCharges', data=churn)
plt.show()
```



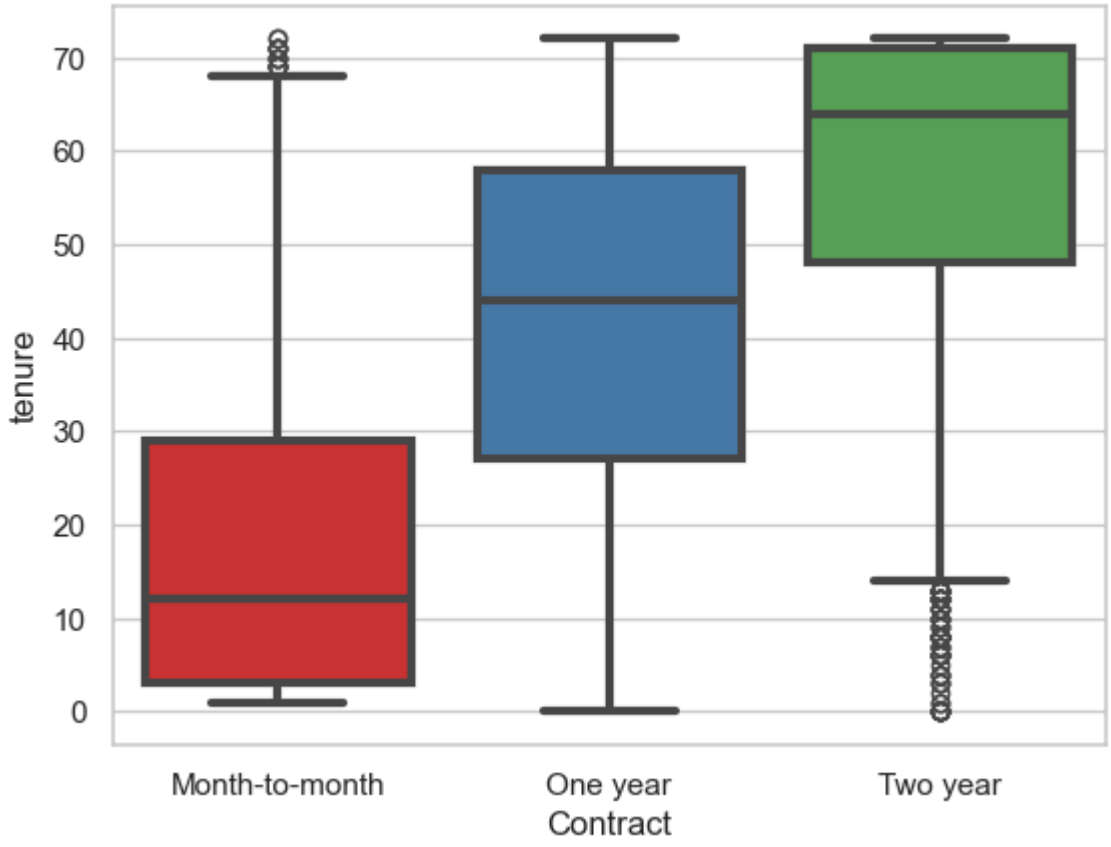
In [264…
```
sns.boxplot(x='Contract', y='tenure', data=churn, palette="Set1")
plt.show()
```

```
/var/folders/8h/zprf7hjs319_78816p34b90c0000gn/T/ipykernel_55511/2094913177.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='Contract', y='tenure', data=churn, palette="Set1")
```



In [265…
```
sns.boxplot(x='Contract', y='tenure', data=churn, palette="Set1", linewid
plt.show()
```

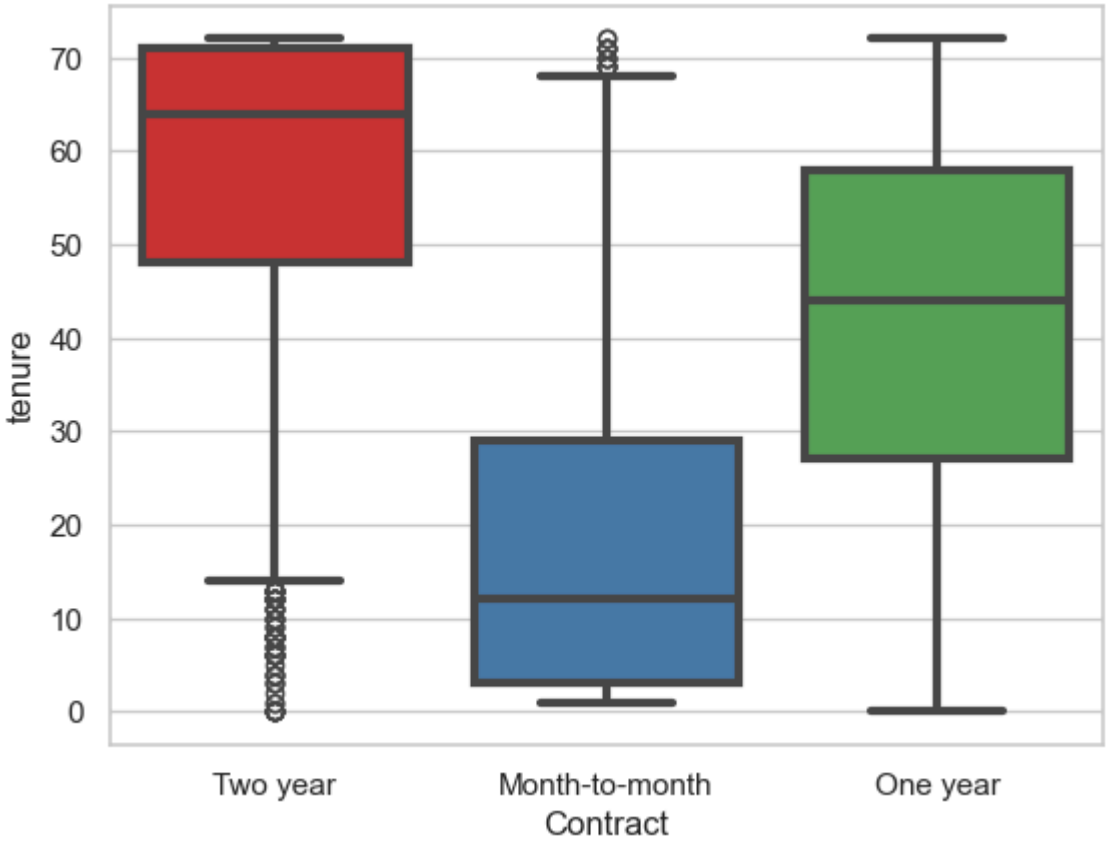
```
/var/folders/8h/zprf7hjs319_78816p34b90c0000gn/T/ipykernel_55511/3845598001.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='Contract', y='tenure', data=churn, palette="Set1", linewidth=3)
```



In [266…

```
sns.boxplot(x='Contract', y='tenure', data=churn, palette="Set1", linewid
plt.show()
```

```
/var/folders/8h/zprf7hjs319_78816p34b90c0000gn/T/ipykernel_55511/18467638
7.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for
the same effect.

  sns.boxplot(x='Contract', y='tenure', data=churn, palette="Set1", linewi
dth=3, order=['Two year','Month-to-month', 'One year'])
```



GenAI

Generative AI, or Gen AI, functions by employing a neural network to analyse data patterns and generate new content based on those patterns.

- Discriminative Classifies data in a way similar to judge
- Generative AI Create, transforms or generates its own content like an artist

Introduction to Generative AI Applications

Overview:

- Generative AI refers to algorithms that enable machines to produce content that is not only new and original but also reflective of the data they were trained on.
- Includes models like GANs (Generative Adversarial Networks), VAEs (Variational Autoencoders), and Transformer-based models (e.g., GPT).

Significance:

- Creativity Boost: Enhances creative processes by providing novel content ideas.
- Efficiency: Automates content creation, saving time and resources.
- Personalization: Generates personalized content for users in various applications.

Development Environment Setup

Setting up the development environment:

- Step 1: Open Command Line Interface (CLI)
  - For Windows: Use Command Prompt or PowerShell.
  - For macOS/Linux: Use Terminal.
- Step 2: Install Libraries using pip
  - pip install numpy
  - pip install flask
  - pip install streamlit
  - pip install torch torchvision torchaudio
  - pip install transformers
  - pip install python-dotenv
- Step 3: Create a .env file Create a file named .env at the root of your project. Inside, add your environment variables in the format KEY=VALUE, one per line:

  API_KEY=your_api_key

  DATABASE_URL=your_database_url

  SECRET_KEY=your_secret_key

  Important: Make sure to add .env to your .gitignore file to prevent committing sensitive information to your version control system.

- Step 4: Load the environment variables In your Python script, import the dotenv module and load the .env file:

  from dotenv import load_dotenv

  import os

  load_dotenv()

  ### Access environment variables

  api_key = os.environ.get('API_KEY')

  database_url = os.environ.get('DATABASE_URL')

  secret_key = os.environ.get('SECRET_KEY')

Verify the Installation

Step 1: Check Python Installation

Open CLI and type: python --version

Ensure it returns the installed version of Python.

Step 2: Verify Library Installations

Open Python interactive shell: python

Import each library to ensure they are installed correctly:

- import numpy
- import flask
- import streamlit
- import torch
- import transformers

Introduction to OpenAI GPT API

API Features

- Text generation, completion, and conversation capabilities.
- Fine-tuning and customization for specific tasks.

Getting Started

- Sign up for an API key on the OpenAI website.
- Refer to the API documentation for detailed instructions.

Flask ChatGPT App

Basic Setup:

Objective: Build a basic chat application using Flask and OpenAI GPT API.

Components:

- Flask for web framework.
- OpenAI GPT API for generating responses.

Let's explore more with a demo !!

Step 1: Setting Up the Environment

First, make sure you have Python and the required libraries installed. You can create a virtual environment and install the necessary packages:

```
python –m venv venv
source venv/bin/activate # On Windows use "venv\Scripts\activate"
```

Explanation:

- python: This invokes the Python interpreter. Depending on your system setup, you might need to specify python3 instead of python to use Python 3.

- -m venv: This option tells Python to run the venv module as a script. The venv module is used to create virtual environments.
- venv: This is the name of the directory where the virtual environment will be created. You can name this directory anything you like, but venv is a common convention.

Step 2: Create the Flask Application

Create a new directory for your project

```
mkdir gtp_chat_app
cd gpt_chat_app
```

Step 3. Create the main flask application file (app.py)

In [ ]:

```
from flask import Flask, request, jsonify, render_template
import requests
import time

from dotenv import load_dotenv
import os

load_dotenv()

# Access environment variables
# OpenAI API key (Note: It's best to keep API keys secure and not hardcod
OPENAI_API_KEY = os.environ.get('OPENAI_API_KEY')

# Initialize the Flask application
app = Flask(__name__)

# Define the route for the home page
@app.route('/')
def index():
    return render_template('index.html')  # Render the 'index.html' templ

# Define the route for the chat endpoint, which accepts POST requests
@app.route('/chat', methods=["POST"])
def chat():
    # Get the user's message from the JSON payload of the request
    user_input = request.json.get('message')
    print(f"Received message: {user_input}")  # Debugging statement to pr

    # Get the response from GPT-3
    response = get_gpt_response(user_input)
    print(f"GPT-3 response: {response}")  # Debugging statement to print

    # Return the GPT-3 response as a JSON object
    return jsonify({'response': response})

def get_gpt_response(user_input, retry_count=3):
    # Set up the headers for the API request
    headers = {
        'Authorization': f'Bearer {OPENAI_API_KEY}',
        'Content-Type': 'application/json'
```

```
    }

    # Set up the data for the API request
    data = {
        'model': 'gpt-3.5-turbo-16k',  # Use a current model
        'messages': [{'role': 'user', 'content': user_input}],  # User's
        'max_tokens': 300  # Limit the response length
    }
    for attempt in range(retry_count):
        response = requests.post('https://api.openai.com/v1/chat/completi

        if response.status_code == 200:
            try:
                # Return the content of the first choice in the response
                return response.json()['choices'][0]['message']['content'
            except (KeyError, IndexError) as e:
                print(f"Error parsing response JSON: {e}")  # Print the e
                print(f"Response JSON: {response.json()}")  # Print the f
                return "An error occurred while processing the response f
        elif response.status_code == 429:
            # Handle rate limiting errors
            print(f"Request to OpenAI failed with status code: {response.
            print(f"Response: {response.text}")
            if attempt < retry_count - 1:
                print("Retrying...")
                time.sleep(2 ** attempt)  # Exponential backoff before re
            else:
                return "You have exceeded your current quota. Please chec
        else:
            # Handle other errors
            print(f"Request to OpenAI failed with status code: {response.
            print(f"Response: {response.text}")
            return "An error occurred while communicating with OpenAI."

# Run the Flask application
if __name__ == '__main__':
    app.run(port=5000, debug=True) # Run the app in debug mode
```

Step 4: Use the index.html

Always keep all html file in `templates` folder

Step 5: Run the Flask Application

> python app.py

For Output: Open your web browser and go to http://127.0.0.1:5000 to see the GPT chat application in action.

Flask Text-to-Image App

Introduction to Text-to-Image Generation:

Text-to-image generation involves creating images from textual descriptions using AI models.

Significance:

- Enhances creativity and design processes.
- Useful in various fields like advertising, entertainment, and virtual environments.

Implementation in Flask

Objective: Build a web application that converts text descriptions into images.

Components:

- Flask for the web framework.
- OpenAI
- HTML/CSS for the front-end interface.

Prerequisites:

- Python installed on your system.
- Required libraries: flask, openai.
- API Key from OpenAI

Let's explore more with a demo !!

Step 1: Setting Up the Environment

Install Python (if not already installed).

Step 2: Create a virtual environment

python -m venv venv

- python: This invokes the Python interpreter. Depending on your system setup, you might need to specify python3 instead of python to use Python 3.
- -m venv: This option tells Python to run the venv module as a script. The venv module is used to create virtual environments.
- venv: This is the name of the directory where the virtual environment will be created. You can name this directory anything you like, but venv is a common convention.

Step 3: Activate the virtual environment

> venv\Scripts\activate

Step 4: Install Flask and OpenAI

> pip install flask openai

Step 5: Create a project directory

> mkdir flask_text_to_image cd flask_text_to_image

Step 6: Create the main Flask application file (app.py)

In [ ]:
```python
# Import necessary libraries from Flask and OpenAI
from flask import Flask, request, jsonify, render_template
import openai

# Initialize the Flask application
app = Flask(__name__)

# Set your OpenAI API key here
openai.api_key = 'your_openai_api_key'  # Replace 'your_openai_api_key'

@app.route("/")
def home():
    # Render the 'index.html' template when the home page is accessed
    return render_template('text_to_image.html')

# Define the route for the generate_image endpoint, which accepts POST re
@app.route('/generate_image', methods=['POST'])
def generate_image():

    # Get the 'prompt' from the JSON payload of the POST request
    prompt = request.json.get('prompt')

    # Use the OpenAI API to generate an image based on the prompt
    response = openai.Image.create(
        prompt=prompt,  # The text prompt for generating the image
        n=1,            # Number of images to generate
        size="512x512"  # Size of the generated image
    )
    image_url = response['data'][0]['url']

    # Return the image URL as a JSON response
    return jsonify({'image_url': image_url})

# Run the Flask application
if __name__ == "__main__":
    # Start the Flask app in debug mode, which provides detailed error me
    app.run(debug=True)
```

Imports:

Flask, request, jsonify, render_template from the Flask framework for handling web requests and rendering HTML templates

Openai for inretacting with the Open AI API.

App Initialization:

app = Flask(**name**) # initializes the Flask application.

openai.api_key =

'your_openai_api_key' # sets the OpenAI API key.

Routes

@app.route('/') # The home route renders the index.html template when accessed. This is the main page of the application. def home(): return render_template('index.html')

@app.route('/generate_image', methods=['POST']) # This route handles POST requests to generate an image based on a provided prompt. The prompt is extracted from the request's JSON payload. def generate_image(): prompt = request.json.get('prompt') # Extracts the prompt from the JSON request body.

```
response = openai.image.create(prompt=prompt, n=1,
size="512x512")  # Uses the OpenAI API to generate an
image. The parameters specify that one image (n=1) of size
512x512 pixels should be generated.

image_url = response['data'][0]['url']  # Extracts the URL
of the generated image from the API response.

return image_url
```

Returning the Image URL:

return jsonify({'image_url': image_url}) # Returns the image URL as a JSON response

Running the App:

if **name** == 'main':

```
app.run(debug=True)  # Runs the Flask application in debug
mode, which is useful for development as it provides
detailed
                     # error messages and automatically
reloads the server when code changes are detected.
```

Step 7 : Create a simple client side HTML interface:

In [ ]:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
    <title>Text-to-Image Generator</title>
</head>
<body>
    <h1>Text-to-Image Generator</h1>
    <div>
        <label for="prompt">Enter your description:</label>
        <input type="text" id="prompt" name="prompt">
        <button onclick="generateImage()">Generate Image</button>
    </div>
    <div id="image-container"></div>
    <script>
        async function generateImage() {
            const prompt = document.getElementById('prompt').value;
            const responseDiv = document.getElementById('image-container'
            const response = await fetch('/generate_image', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ prompt: prompt })
            });
            const data = await response.json();
            const img = document.createElement('img');
            img.src = data.image_url;
            responseDiv.innerHTML = '';
            responseDiv.appendChild(img);
        }
    </script>
</body>
</html>
```

Step 8: Run the application:

$ python app.py

- Quick Recap:
- Environment Setup: Install Python, Flask, and OpenAI.
- Basic Flask Setup: Create a simple Flask application.
- Text-to-Image Functionality:
  - Integrate OpenAI's text-to-image generation API.
  - Create a route to handle image generation.
  - Develop a simple HTML interface for user interaction.
- Run and Access the Application: Start the Flask server and interact with the text-to-image application through the web.

LangChain Apps

Overview:

- LangChain streamlines the development process for applications that utilize LLMs by offering a modular and extensible architecture.
- It supports a wide range of use cases, from chatbots and personal assistants to complex NLP tasks and data analysis.
- The framework is built to be highly customizable, allowing developers to tailor it to their specific needs and integrate it with various external data sources and APIs.

Let's understand more with the help of LangChain Case Study.

Personalized Story Generator:

This project will take inputs like character names, settings, and themes from the user and generate a unique story using a text generation model like GPT-3.5.

Steps to Create the Project:

1. Set Up Environment
2. Collect User Inputs
3. Generate Story Using AI Model
4. Display the Generated Story

Step 1. Set Up Environment

- Install required libraries

pip install openai langchain

Step 2. Collect User Inputs

Create a script to collect inputs from the user.

```
def get_user_inputs():
    print("Welcome to the Personalized Story Generator!")
    character_name = input("Enter the main character's
name: ")
    setting = input("Enter the setting of the story: ")
    theme = input("Enter the theme of the story (e.g.,
```

```
adventure, mystery): ")
    return character_name, setting, theme
```

Step 3. Generate Story Using AI Model

```
story_generator.py
# Import necessary classes and functions from langchain
and user_input modules
from langchain import Chain, Prompt, TextModel
from user_input import get_user_inputs

# Function to generate a story based on character name,
setting, and theme
def generate_story(character_name, setting, theme):
    # Create a template for the story prompt with
placeholders for character name, setting, and theme
    prompt_template = f"Create a story with the following
details:\n\n" \
                      f"Main Character:
{character_name}\n" \
                      f"Setting: {setting}\n" \
                      f"Theme: {theme}\n\n" \
                      f"Story:\n"
    # Create a Prompt object using the template
    prompt = Prompt(template=prompt_template)

    # Initialize a TextModel object with a specific model
name and API key
    # Replace "your_openai_api_key" with your actual
OpenAI API key
    model = TextModel(model_name="gpt-3.5-turbo",
api_key="your_openai_api_key")

    # Create a Chain object that uses the prompt and model
in a pipeline
    chain = Chain(pipeline=[prompt, model])

    # Run the chain with the initial input (which is empty
in this case)
    output = chain.run(input_data={})

    # Return the generated story output
    return output

# Main block to execute the script
if __name__ == "__main__":
    # Get user inputs for character name, setting, and
theme
    character_name, setting, theme = get_user_inputs()

    # Generate a story based on the user inputs
    story = generate_story(character_name, setting, theme)

    # Print the generated story
    print("\nGenerated Story:\n")
```

```
print(story)
```

Step 4. Display the Generated Story

python story_generator.py

Quick Recap:

Environment Setup:

- Install Python and required libraries (langchain, openai).
- Ensure you have your OpenAI API key.

Basic Script Setup:

- Create the main script (story_generator.py).

Story Generation Functionality:

- Integrate OpenAI's GPT-3 for text generation.
- Create a function to generate a story based on user inputs (character name, setting, theme).
- Develop a simple user input module to get inputs.

Run and Generate the Story:

- Execute the script.
- Provide the required inputs when prompted.
- View the generated story printed in the console.

LangChain Apps

Overview:

- LangChain streamlines the development process for applications that utilize LLMs by offering a modular and extensible architecture.
- It supports a wide range of use cases, from chatbots and personal assistants to complex NLP tasks and data analysis.
- The framework is built to be highly customizable, allowing developers to tailor it to their specific needs and integrate it with various external data sources and APIs.

Selenium : Automation Testing

Once we develop a software component/product, we have to analyze and inspect its features and also evaluate the component for potential errors and bugs so that when it gets delivered in the market, it is free of any bugs and errors. It is the point where we need extensive testing of the software.

Testing is done when the application is built, is ready to test and deployed in the test servers/environments.

(Manual Testing)

Manual testing means the (web) application is tested manually by QA testers. Tests need to be performed manually in every environment, using a different data set and the success/failure rate of every transaction should be recorded.

Manual testing is mandatory for every newly developed software before automated testing. This testing requires great efforts and time, but it gives the surety of bug-free software.

Challenges in Manual Testing

Manual Testing requires more time and more resources.

GUI objects size differences and color combinations, etc., are not easy to find in Manual Testing.

Executing the same tests again and again is a time-consuming and tedious process.

Automation Testing:

As the name suggests, automation testing takes software testing activities and executes them via an automation toolset or framework. In simple words, it is a type of testing in which a tool executes a set of tasks in a defined pattern automatically.

This automation testing method uses scripted sequences that are executed by testing tools. Automated testing tools execute examinations of the software, report outcomes, and compare results with earlier test runs.

What is Selenium?

Selenium was introduced by Jason Huggins in 2004. Jason Huggins, an Engineer at Thoughtworks, was doing his work on some web application and he suddenly required testing.

Testing done using Selenium is often referred to as Selenium Testing.

Selenium is an open-source tool and portable framework that is used for automating the tests administered on web browsers. It is only used for testing web applications

such as Shopping Carts, Email Programs like Gmail, Yahoo.

Why Selenium with Python?

Python runs very faster and makes use of indentation to initiate and end blocks.

It is very simple as well as compact as compared to other programming languages.

The most important tool for easy user interfaces, WebDriver, has strong bindings for Python.

Runs rapidly when compared to other programming languages.

The programming language is also free and available as open source.

Whoever needs it can easily download and use it freely in any environment.

Easy to code and easy to read

Advantages of Selenium Testing

Various programming languages are supported to write the test scripts

Selenium supports various browsers like Mozilla Firefox, Google Chrome etc.

Selenium supports parallel test execution.

Selenium is an open source software.

Selenium supports different operating systems like Windows, Linux, Mac etc.

Limitations of Selenium Testing

Selenium supports web-based applications only.

The instability of new features. They may work or may not.

Selenium can't perform testing on images.

Captchas are not automated using Selenium.

Barcodes can't be automated using Selenium.

Summary : Let's quickly recap:

- In Python Fundamentals, we discussed the introduction and installation of Python, variables, data types, operators, Python tokens, flow control statements, and basic data structures like Tuple, List, Dictionary, and Set.

- Advanced Python concepts focused on object-oriented programming, inheritance, exception handling, and file handling in Python.
- Data Structures and Algorithms covered arrays, stacks, queues, linked lists, linear and binary search, insertion sort, quick sort, and merge sort.
- Python for ML introduced libraries such as Numpy, Pandas, Matplotlib, and Seaborn.
- Python for Generative AI provided an overview of generative AI concepts and Python's application in this field.