

## Advanced Python Programming & Applications

---

By : [Dr Atif Khan](#)

Associate Professor at Islamia College Peshawar

Prepared By : [Irfan Ullah Khan](#)

---

### ▼ Notebook: 1) Python Lists.ipynb

Double-click (or enter) to edit

- list is an important datatype or data structure
- what is a list

- list vs Array
- how to create a list
- Access the elements of list
- Edit the list
- Add an element to the list
- Delete operation on list
- operations on list
- Functions available for list

List is a datatype where you can store multiple items of different types. List and array are fundamentally same with slight differences.

1. Array is homogenous(all elements in the array will be of same type, means int array will contain elements of integer type while character array will have elements of character type. However, list can be heterogenous, means elements in the list can be of different types)
2. Array stores elements on continuous memory locations while list does not store element on continuous memory location
3. becos of continuous memory allocation, Arrays are much faster than lists, as it takes less time to fetch elements from the array.
4. lists are more programmer friendly, programmer can build logic effectively on lists as compared to array

```
1 # how to create a list
2 # create an empty list
3 l=[]
4 l
```

→ []

```
1 le= list()
2 le
```

→ []

```
1 # create a homogenous list where all elements are of same type
2 l2= [1,2,3,4,5]
3 l2
```

→ [1, 2, 3, 4, 5]

```
1 # create a heterogeneous list where elements are of different
2 # this is not possible in array
3 l2= ["Python",22,3.5,4+3j,True]
4 l2
```

```
→ ['Python', 22, 3.5, (4+3j), True]
```

```
1 # Multi-dimensional list-2D list- list within a list, it is
2 l3= [2,3,4,[5,6]]      #here 2 on 'index=0" and 3 on index=1 a
3 l3
4 #len(l3)
```

```
→ [2, 3, 4, [5, 6]]
```

```
1 l3[-1] # to print last index
```

```
→ [5, 6]
```

```
1 l3[-3] # as we know that l3 has 4 item , when we in reverse
2 # on 1 index we have [5,6] and on 2 index we have 4 and on 3
```

```
→ 3
```

```
1 # 2D array
2 l4=[
3     [1,2,3],
4     [4,5,6]
5 ]
6 l4
```

```
→ [[1, 2, 3], [4, 5, 6]]
```

```
1 # total number of list or array
2 len(l4)
```

```
→ 2
```

```
1 # to print first list or ID list
2 l4[0]
```

```
→ [1, 2, 3]
```

```
1 # # to print 2nd list or 2ndD list  
2  
3 14[1]
```

→ [4, 5, 6]

```
1 # to print from first list which is on '0' index , so in fir  
2 14[0][-1]
```

→ 3

```
1 # to print from 2nd list which is on '1' index , so in 2nd li  
2 14[1][-1]
```

→ 6

The first layer contains two lists. Each list in the first layer contains two lists. Each list in the second layer contains three values. So, the structure is like a set of nested containers: first, you choose which big container (first layer), then which smaller container inside it (second layer), and finally, you find the value in the smallest container.

Acessing items from a list The first layer contains two lists. Each list in the first layer contains two lists. Each list in the second layer contains three values. So, the structure is like a set of nested containers: first, you choose which big container (first layer), then which smaller container inside it (second layer), and finally, you find the value in the smallest container.

```
1 # create a 3d list, list within a list within a list  
2 # Creating a 3D list  
3 three_d_list = [  
4     # First layer  
5     [  
6         [1, 2, 3],  # Second layer  
7         [4, 5, 6]  
8     ],  
9     [  # Another set of lists in the first layer  
10        [7, 8, 9],  
11        [10, 11, 12]  
12    ]  
13 ]
```

```
14 three_d_list
```

```
15
```

```
16
```

```
→ [[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]]
```

```
1 len(three_d_list)
```

```
→ 2
```

```
1
```

```
2 three_d_list[0][-1][1]
```

```
3
```

```
→ 5
```

```
1
```

```
2 three_d_list[0][1][1]
```

```
3
```

```
→ 5
```

```
1 three_d_list[1][1]
```

```
→ [10, 11, 12]
```

```
1 three_d_list[1][0]
```

```
→ [7, 8, 9]
```

```
1 # Accessing elements
```

```
2 print(three_d_list[0][1][2]) # Accessing the value 6
```

```
3 print(three_d_list[1][0][1]) # Accessing the value 8
```

```
→ 6
```

```
8
```

```
1 # creating list with type conversion
```

```
2 l= list("python") # this will convert strings into a list,
```

```
3 l
```

```
→ ['p', 'y', 't', 'h', 'o', 'n']
```

```
1 l= ''.join(l) # this will convert strings into a list, rare  
2 l
```

```
→ 'python'
```

```
1 newlist=[ ]  
2 newlist.append(l)  
3 newlist
```

```
→ ['python']
```

```
1 l6=list() # create an empty list  
2 l6
```

```
→ []
```

```
1 Start coding or generate with AI.
```

Accessing items from a list python code is consistent, when you learn one data type, the same operations are also applicable on other datatype.

The general syntax for slicing a list in Python is: list[start:stop:step] Where:

start → The index where the slice begins (inclusive). stop → The index where the slice ends (exclusive). step → The interval (or step size) between elements

```
1 #Basic Slicing (Extracting a Portion of the List)  
2 numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
3  
4 # Extract elements from index 2 to 6 (excluding index 6)  
5 print(numbers[3:8])  
6
```

```
→ [3, 4, 5, 6, 7]
```

```
1 #Print first 5 numbers  
2 print(numbers[5:])
```

```
→ [0, 1, 2, 3, 4]
```

```
1 # Omitting start (default is 0, i.e., start from the beginning)
2 print(numbers[:5]) # [0, 1, 2, 3, 4]
3
4 # Omitting stop (default is the end of the list)
5 print(numbers[5:]) # [5, 6, 7, 8, 9]
6
7 # Omitting both (copies the entire list)
8 print(numbers[:]) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
9
```

```
→ [0, 1, 2, 3, 4]
[5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
1 pwd
```

```
→ 'C:\\\\Users\\\\Dr. Atif Khan\\\\NAVTAC'
```

```
1 # To check where or in which your notebook is save
2 pwd
```

```
→ 'E:\\\\Python For Advance Application'
```

```
1 print(numbers[::-1])
```

```
→ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
1 #Using a step Value
2 # Extract every second element
3 print(numbers[::-2])
4
5 #Reversing a List with a Negative Step
6 # Reverse the list
7 #A step of -1 means moving backward from the last element to
8 print(numbers[::-1])
9
```

```
→ [0, 2, 4, 6, 8]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

```
1 # Print all list item  
2 print(numbers[::-1])
```

→ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```
1 l2= [1,2,3,4,5]  
2 print("first element of list ", l2[0])    # access the first item  
3  
4 print("last element ", l2[-1])  # access the last item  
5  
6 print(l2[1:3])  # access the second and third items  
7  
8 print(l2[::-1]) # reverse the list  
9  
10 #Normally, l2[:] creates a full copy of the list.  
11 #By adding -1 as the step, it starts from the last element and goes back.  
12 #As a result, it returns a reversed copy of l2 without modifying the original list.
```

→ first element of list 1  
last element 5  
[2, 3]  
[5, 4, 3, 2, 1]

```
1 l2[2:4]
```

→ [3, 4]

```
1 l3  # print contents of 2d list
```

→ [2, 3, 4, [5, 6]]

```
1 lst[2:4]
```

→ -----  
NameError Traceback (most recent call last)  
Cell In[78], line 1  
----> 1 lst[2:4]  
  
NameError: name 'lst' is not defined



```
1 l3= [2,3,4,[5,6]]
```

```
2 l3
```

```
3
```

```
→ [2, 3, 4, [5, 6]]
```

```
1 x=l3[-1]
```

```
2 x
```

```
→ [5, 6]
```

```
1 x[1]
```

```
→ 6
```

```
1 x[0]
```

```
→ 5
```

```
1 # Access the last item, in 2d list, which is itself a list,u
```

```
2 print(l3[-1][0])
```

```
3 print(l3[-1][1])
```

```
4
```

```
5 print(l3[-1][-1])
```

```
6 print(l3[-1][-2])
```

```
→ 5
```

```
6
```

```
6
```

```
5
```

```
1 print(three_d_list)
```

```
→ [[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]]
```

```
1 # fetch 10 out of the three_d_list, use square brackets equa
```

```
2 # outer list has two lists,10 lie in second list,now the sec
```

```
3 #and the desired item lie in second list(put 1),and then 10
```

```
4 # u can use both +ve and -ve indexing
```

```
5
```

```
6 print(three_d_list[-1][-1][0])
```

```
7 print(three_d_list[1][1][0])
8
```

```
→ 10
  10
```

```
1 # Acess element 3
2 print(three_d_list[0][0][2])
```

```
→ 3
```

```
1 l2= [1,
2      5]
3 l2
4 print(id(l2))
```

```
→ 2948162370752
```

```
1 #Edit a list, # lists in pythons are mutable means you can c
2 print(l2)
3 # replace 1 with 100
4 l2[-1]=500
5 print(l2)
6 print(id(l2))
```

```
→ [1, 5]
  [1, 500]
  2948162370752
```

```
1 #Edit list using -ve indexing
2 l2[-1]=500
3 l2
```

```
→ [1, 500]
```

```
1 # Edit list using slicing
2 l2[2:]=[300,400,500]
3 l2
```

```
→ [1, 500, 300, 400, 500]
```

```
1 # how to add new items in a list
2 # three methods for adding items in list: 1.Append() 2. Ext
3 # Append and extend both add items at the end of list but ap
```

```
1 12
```

```
→ [1, 500, 300, 400, 500]
```

```
1 lst=[]
2 for i in range(3):
3     value=eval(input('enter values'))
4     lst.append(value)
5 print(lst)
```

```
→ enter values 4-2
    enter values 5
    enter values 4
[2, 5, 4]
```

```
1 lst=[]
2 for i in range(3):
3     value=eval(input('enter values'))
4     lst.append(value)
5 print(lst)
```

```
→ enter values 2+3
    enter values 3
    enter values 7
[5, 3, 7]
```

```
1 s=eval('2+3')
2 print(s)
```

```
→ 5
```

```
1 t=eval('4-3')
2 print(t)
```

```
→ 1
```

```
1 # Append() adds new item (integer, string) at the end of lis
2 12.append(6000)
```

```
3 12
```

```
→ [1, 500, 300, 400, 500, 6000]
```

```
1 l2.append([7000,8000]) # here list will be added as a singl
2 l2
```

```
→ [1, 500, 300, 400, 500, 6000, [7000, 8000]]
```

```
1 l2.extend(['Ziaullah',21, 'hangu',51.5]) #here will added a
2 l2
```

```
→ [1,
  500,
  300,
  400,
  500,
  6000,
  [7000, 8000],
  'Ziaullah',
  21,
  'hangu',
  51.5,
  'Ziaullah',
  21,
  'hangu',
  51.5,
  'Ziaullah',
  21,
  'hangu',
  51.5]
```

```
1 # Extend() adds multiple items at the end of list simultaneo
2 l2.extend([7000,8000, 'Python'])
3 l2
```

```
→ [1,
  500,
  300,
  400,
  500,
  6000,
  [7000, 8000],
  'Ziaullah',
  21,
  'hangu',
  51.5,
  'Ziaullah',
  21,
  'hangu',
```

```
51.5,  
'Ziaullah',  
21,  
'hangu',  
51.5,  
7000,  
8000,  
'Python']
```

```
1 for i in 'Islamia Collge':  
2     print(i)  
3
```

```
→ I  
s  
l  
a  
m  
i  
a  
  
C  
o  
l  
l  
g  
e
```

```
1 for i in "hello":  
2     print(i)
```

```
→ h  
e  
l  
l  
o
```

```
1 l=[3,4,5,6]  
2 l.extend([3,4.6,'python'])
```

```
1 l
```

```
→ [3, 4, 5, 6, 3, 4.6, 'python']
```

```
1 l.append('Java')
```

```
1 l
```

```
→ [3, 4, 5, 6, 3, 4.6, 'python', 'Java']
```

```
1 # first string will be converted into list,then each element
2 # extend will also try to add multiple items
3 l2.extend("hello")
4 l2
```

```
→ [1,
  500,
  300,
  400,
  500,
  6000,
  [7000, 8000],
  'Ziaullah',
  21,
  'hangu',
  51.5,
  'Ziaullah',
  21,
  'hangu',
  51.5,
  'Ziaullah',
  21,
  'hangu',
  51.5,
  7000,
  8000,
  'Python',
  'h',
  'e',
  'l',
  'l',
  'o',
  'h',
  'e']
```

```
'1',  
'1',  
'o']
```

```
1 # first string will be converted into list, then each element  
2 # extend will also try to add multiple items  
3 l2.extend("hello")  
4 print(l2)
```

```
→ [1, 500, 300, 400, 500, 6000, [7000, 8000], 'Ziaullah', 21, 'hangu', 51.5, 'Ziaullah', 2]
```



```
1 a=[1,2,3,4,5]  
2 a.extend('khan')  
3 print(a)
```

```
→ [1, 2, 3, 4, 5, 'k', 'h', 'a', 'n']
```

```
1 a=[1,2,3,4,5]  
2 a.extend('khan')  
3 print(*a)
```

```
→ 1 2 3 4 5 k h a n
```

```
1 Start coding or generate with AI.
```

```
1 # Insert method adds an item at desired index position
```

```
1 l2.insert()
```

```
→ -----  
      TypeError  
      Cell In[122], line 1  
      ----> 1 l2.insert()  
  
      Traceback (most recent call last)
```

```
TypeError: insert expected 2 arguments, got 0
```

```
1 l2.insert(1,'Esa')  
2 l2
```

```
→ [1,  
  'Esa',
```

```
500,  
300,  
400,  
500,  
6000,  
[7000, 8000],  
'Ziaullah',  
21,  
'hangu',  
51.5,  
'Ziaullah',  
21,  
'hangu',  
51.5,  
'Ziaullah',  
21,  
'hangu',  
51.5,  
7000,  
8000,  
'Python',  
'h',  
'e',  
'l',  
'l',  
'o',  
'h',  
'e',  
'l',  
'l',  
'o']
```

```
1 l2.insert(-1, 'Atiaq')
```



```
1 lst=[4,5,6,7]
```

```
2 lst
```

```
→ [4, 5, 6, 7]
```

```
1 lst.insert(-1,'Atiq')
```

```
1 lst.insert(0,'aslam')
```

```
2 lst
```

```
→ ['aslam', 4, 5, 6, 'Atiq', 7]
```

```
1 lst.insert(1,'Uzair')
```

```
2 lst
```

```
→ ['aslam', 'Uzair', 4, 5, 6, 'Atiq', 7]
```

```
1 lst
```

```
→ ['aslam', 'Uzair', 4, 5, 6, 'Atiq', 7]
```

## ▼ 4 ways to delete elements from list

1. del keyword
2. remove()
3. pop()
4. clear()

```
1 Start coding or generate with AI.
```

```
1 l3
```

```
→ [2, 3, 4, [5, 6]]
```

```
1 # you can delete the entire list using del keyword
```

```
2 l3=l2
```

```
3 del l2
```

```
1 l2
```



NameError

Cell In[146], line 1  
----> 1 12

Traceback (most recent call last)

NameError: name '12' is not defined

```
1 lst=[4,5,6,7,8]
2 del lst[0]
3 lst
```

→ [5, 6, 7, 8]

```
1 del lst[-1]
2 lst
```

→ [5, 6, 7]

```
1 # delete an item from list using +ve and -ve index
2 del l3[0]
3 l3
```

→ ['Esa',  
 500,  
 300,  
 400,  
 500,  
 6000,  
 [7000, 8000],  
 'Ziaullah',  
 21,  
 'hangu',  
 51.5,  
 'Ziaullah',  
 21,  
 'hangu',  
 51.5,  
 'Ziaullah',  
 21,  
 'hangu',  
 51.5,  
 7000,  
 8000,  
 'Python',  
 'h',  
 'e',  
 'l',  
 'l',

```
'o',
'h',
'e',
'l',
'l',
'Atiaq',
'o']
```

```
1 del l3[-1] # delete the last item
2 l3
```

```
→ ['Esa',
 500,
 300,
 400,
 500,
 6000,
 [7000, 8000],
 'Ziaullah',
 21,
 'hangu',
 51.5,
 'Ziaullah',
 21,
 'hangu',
 51.5,
 'Ziaullah',
 21,
 'hangu',
 51.5,
 7000,
 8000,
 'Python',
 'h',
```

113

```
[→] [ 'Esa',
      500,
      300,
      400,
      500,
      6000,
      [7000, 8000],
      'Ziaullah',
      21,
      'hangu',
      51.5,
      'Ziaullah',
      21,
      'hangu',
      51.5,
      'Ziaullah',
      21,
      'hangu',
      51.5,
      7000,
      8000,
      'Python',
      'h',
```

'e',  
'l',  
'l',  
'o',  
'h',  
'e',  
'l',  
'l',  
'l',  
'o',  
'h',  
'e',  
'l',  
'l',  
'Atiaq']

```
1 lst8=['Esa', 200, 300, 400, 500, 6000, [7000, 8000], 7000, 8  
2 lst8
```

3

```
→ ['Esa', 200, 300, 400, 500, 6000, [7000, 8000], 7000, 8000, 'Python']
```

```
1 del lst8[6:]  
2 lst8
```

```
→ ['Esa', 200, 300, 400, 500, 6000]
```

```
1 # delete the substring 'hel'  
2 del 13[-4:]  
3 13
```

```
→ [ 'Esa' ,  
    500 ,  
    300 ,  
    400 ,  
    500 ,
```

```
6000,  
[7000, 8000],  
'Ziaullah',  
21,  
'hangu',  
51.5,  
'Ziaullah',  
21,  
'hangu',  
51.5,  
'Ziaullah',  
21,  
'hangu',  
51.5,  
7000,  
8000,  
'Python',  
'h',  
'e',  
'l',  
'l',  
'o',  
'h']
```

```
1 del 13[-2]
```

```
1 13
```

```
→ ['Esa',  
 500,  
 300,  
 400,  
 500,
```

```
6000,  
[7000, 8000],  
'Ziaullah',  
21,  
'hangu',  
51.5,  
'Ziaullah',  
21,  
'hangu',  
51.5,  
'Ziaullah',  
21,  
'hangu',  
51.5,  
7000,  
8000,  
'Python',  
'h',  
'e',  
'l',  
'l',  
'o',  
'h',  
'e',  
'l',  
'l',  
'h']
```

```
1 lst6=['Esa', 2, 300, 400, 500, 6000, 'Python']  
2 lst6  
3
```

```
→ ['Esa', 2, 300, 400, 500, 6000, 'Python']
```

```
1 lst6.remove('Python')  
2 lst6
```

```
→ ['Esa', 2, 300, 400, 500, 6000]
```

```
1 l3.remove(400)
```

```
2 l3
```

```
→ ['Esa',
  500,
  300,
  500,
  6000,
  [7000, 8000],
  'Ziaullah',
  21,
  'hangu',
  51.5,
  'Ziaullah',
  21,
  'hangu',
  51.5,
  'Ziaullah',
  21,
  'hangu',
  51.5,
  7000,
  8000,
  'Python',
  'h',
  'e',
  'l',
  'l',
  'o',
  'h',
  'e',
  'l',
  'l',
  'h']
```

```
1 lst3.remove('Python')
```

```
2 lst3
```

```
→ ['Esa',
 500,
300,
500,
6000,
[7000, 8000],
'Ziaullah',
21,
'hangu',
51.5,
'Ziaullah',
21,
'hangu',
51.5,
'Ziaullah',
21,
'hangu',
51.5,
7000,
8000,
'h',
'e',
'l',
'l',
'o',
'h',
'e',
'l',
'l',
'h']
```

```
1 if 'Esa' in lst6:
2     lst6.remove('Esa')
3     print(lst6)
```

```
4 else:  
5     print('NO')
```

→ [2, 300, 400, 500, 6000]

```
1 z=['gul wali','Said wali','asif wali','numan wali']  
2 z  
3
```

→ ['gul wali', 'Said wali', 'asif wali', 'numan wali']

```
1 if 'asif wali' in z:  
2     z.remove('asif wali')  
3     print(z)  
4 else:  
5     print('No')
```

→ ['gul wali', 'Said wali', 'numan wali']

```
1 if 'asif wali' in z:  
2     z.remove('asif wali')  
3     print(z)  
4 else:  
5     print('No')
```

→ No

```
1 type("2+2")
```

→ str

```
1 # knowing that item exists in the list, but i don't know its  
2 if 'Python' in l3:  
3     l3.remove('Python')  
4 print(l3)
```

→ ['Esa', 500, 300, 500, 6000, [7000, 8000], 'ziaullah', 21, 'hangu', 51.5, 'Ziaullah', 21]



```
1 l2=[4,5,6,7,7,8,9,0]
```

```
2 l2
```

```
→ [4, 5, 6, 7, 7, 8, 9, 0]
```

```
1 l2.pop(3)
```

```
2 l2
```

```
→ [4, 5, 6, 7, 8, 9, 0]
```

```
1 #pop() removes the element at a given position and return th
```

```
2 #pop() deletes the last item in the list and return it, if t
```

```
3 l2.pop(3)
```

```
→ 7
```

```
1 l2
```

```
→ [4, 5, 6, 8, 9, 0]
```

```
1 l2
```

```
→ [4, 5, 6, 8, 9, 0]
```

```
1 lst10=[3,4,5,6,7]
```

```
1 lst10.pop()
```

```
→ 7
```

```
1 y=[1,2,3,4,5]
```

```
2 y
```

```
→ [1, 2, 3, 4, 5]
```

```
1 y.pop(2)
```

```
2 y
```

```
→ [1, 2, 4, 5]
```

```
1 y.pop() # it means that if you don't give the index number ,
```

→ 5

1 y

→ [1, 2, 4]

1 13

→ ['Esa',  
500,  
300,  
500,  
6000,  
[7000, 8000],  
'Ziaullah',  
21,  
'hangu',  
51.5,  
'Ziaullah',  
21,  
'hangu',  
51.5,  
'Ziaullah',  
21,  
'hangu',  
51.5,  
7000,  
8000,  
'h',  
'e',  
'l',  
'l',  
'o',  
'h',  
'e',  
'l',  
'l',  
'h']

```
1 l3.pop()
```

```
→ 'h'
```

```
1 #clear does not delete the list but it will emptied the list
2 l3.clear()
3 l3
```

```
→ []
```

```
1 empty=[2,3,4,5]
2 empty
```

```
→ [2, 3, 4, 5]
```

```
1 empty.clear()
2 empty
```

```
→ []
```

```
1 #operations on list: 4 major operations: 1. concatenation 2.
2 l1=[1,2,3,4,3]
3 l2=[4,5,6,7,7,5]
4 l3=l1+l2 # new list will be formed after concatenation
5 l3
```

```
→ [1, 2, 3, 4, 3, 4, 5, 6, 7, 7, 5]
```

```
1 l3.count(4)
```

```
→ 2
```

```
1 x=[1,3,4,5]
2 w=[5,6,7,8,3]
3 v=x+w
4 v
```

```
→ [1, 3, 4, 5, 5, 6, 7, 8, 3]
```

```
1 v.count(5) # it will count the how many time any value comes
```

→ 2

```
1 yy=['pythonpython','p','y','p']
```

```
2 yy
```

→ ['pythonpython', 'p', 'y', 'p']

```
1 yy.count('p')
```

→ 2

```
1 for i in 'python':
```

```
2     print(i)
```

→ p  
y  
t  
h  
o  
n

```
1 Start coding or generate with AI.
```

→ 0

```
1 # count() returns the number of times an element x appears in
```

```
2 #l3.count(4)
```

```
3 result=[]
```

```
4 for i in l3:
```

```
5 if i not in result:
```

```
6     result.append(i)
```

```
7     print(f"{i} appears {l3.count(i)} times")
```

→ 1 appears 1 times  
2 appears 1 times  
3 appears 2 times  
4 appears 2 times  
5 appears 2 times  
6 appears 1 times  
7 appears 2 times

```
1 result=[]
2 for i in yy:
3     if i not in result:
4         result.append(i)
5     print(f'{i} appears {yy.count(i)} times')
6
```

→ pythonpython appears 1 times  
p appears 2 times  
y appears 1 times

```
1 q=[1,2,3,4,4,5,5,5]
2 result=[]
3 for i in q:
4     if i not in result:
5         result.append(i)
6     print(f'{i} appears {q.count(i)} times')
```

→ 1 appears 1 times  
2 appears 1 times  
3 appears 1 times  
4 appears 2 times  
5 appears 3 times

```
1 books=['urdu','english','math','urdu','math','english','phys'
2 result=[]
3 for i in books:
4     if i not in result:
5         result.append(i)
6     print(f'{i} appears {books.count(i)} times')
```

→ urdu appears 2 times  
english appears 2 times  
math appears 2 times  
physics appears 1 times

```
1 #Multiplying a Python list by an integer extends the list by
2 #a certain number of times.
3 # L * 3, it will concatenate three copies of the list together
4 1*3
```

→ [1, 2, 3, 4, 3, 1, 2, 3, 4, 3, 1, 2, 3, 4, 3]

```
1 l4=[4,5,6]
2 l4=l4*2 # it print the list 2 times but in one list , [4,5,6
3 l4
```

```
→ [4, 5, 6, 4, 5, 6]
```

```
1 aa=['pak','kpk','pesh']
2 aa=aa*2
3 aa
```

```
→ ['pak', 'kpk', 'pesh', 'pak', 'kpk', 'pesh']
```

```
1 # loop on list
2 for i in l4:
3     print(i)
```

```
→ 4
5
6
4
5
6
```

```
1 aa=['pak','kpk','pesh']
2 for i in aa: # it print the item according list have , for e:
3     print(aa)
```

```
→ ['pak', 'kpk', 'pesh', 'pak', 'kpk', 'pesh']
```

```
1 for i in aa:
2     print(i)
```

```
→ pak
kpk
pesh
pak
kpk
pesh
```

```
1 count=[1,2,3,4]
2 for i in count:
3     print(i)
```

→ 1  
2  
3  
4

```
1 for j in [4,5,6,7]:
2     print(j)
```

→ 4  
5  
6  
7

```
1 l2=[1,2,3,[4,5]]
2 l2
```

→ [1, 2, 3, [4, 5]]

```
1 for i in l2:
2     print(i)
```

→ 1  
2  
3  
[4, 5]

```
1 # membership operator 'in'
2 4 in l2
```

→ False

```
1 count=[1,2,3,4]
2 3 in count
3
```

→ True

```
1 ls=[4,6,7,8,9]
2 if 7 in ls:
```

```
3     print(True)
4 else:
5     print(False)
```

→ True

```
1 count=[1,2,3,4]
2 if 4 in count:
3     print('True')
4 else:
5     print("False")
```

→ True

```
1 ls.append([4,5])
2 ls
```

→ [4, 6, 7, 8, 9, [4, 5], [4, 5]]

```
1 count.append([4,5])
2 count
```

→ [1, 2, 3, 4, [4, 5], [4, 5]]

```
1 [4,6] in ls
```

→ False

```
1 [4,5] in count
```

→ True

```
1 2 in l2
```

→ True

```
1 [4, 5] in l2
```

→ True

```
1 l1=[4,5,6,7, [7,9]]  
2 max(l1)
```

→ -----

TypeError Traceback (most recent call last)  
Cell In[372], line 2  
 1 l1=[4,5,6,7, [7,9]]  
----> 2 max(l1)

TypeError: '>' not supported between instances of 'list' and 'int'

```
1 # Functions available for list  
2 l1=[4,5,6,7,2,3]  
3
```

```
1 len(l1) # number of items in list
```

→ 6

```
1 min(l1) # needs numeric list
```

→ 2

```
1 max(l1)
```

→ 7

```
1 sum(l1)
```

→ 27

```
1 sorted(l1)
```

→ [2, 3, 4, 5, 6, 7]

```
1 zz=[4,5,6,7,8]  
2 zz  
3
```

→ [4, 5, 6, 7, 8]

1 len(zz)

5

1 max(zz)

8

`1 min(zz)`

4

1 sum(zz)

30

```
1 sorted(zz)
```

→ [4, 5, 6, 7, 8]

```
1 sorted(zz,reverse=True) # Sorted just change temporary or it
```

$\rightarrow [8, 7, 6, 5, 4]$

```
1 zz.sort() # it change permanent , it change the original list
```

2 zz

$\rightarrow$  [4, 5, 6, 7, 8]

1 Start coding or generate with AI.

```
1 sorted(l1, reverse=True) # sorted will not make changes in  
2 #new list in memory. It will print 1
```

→ [7, 6, 5, 4, 3, 2]

```
1 l1.sort() # this operation is permanent, sort() will make ch  
2 l1
```

→ [2, 3, 4, 5, 6, 7]

```
1 l1
```

```
→ [2, 3, 4, 5, 6, 7]
```

```
1 l1.sort(reverse=True) # settin reverse=True sorts the list i  
2 l1
```

```
→ [7, 6, 5, 4, 3, 2]
```

```
1 zz.sort(reverse=True) # mean that we can use reverse with so  
2 zz
```

```
→ [8, 7, 6, 5, 4]
```

```
1 l1.sort()  
2 l1
```

```
→ [2, 3, 4, 5, 6, 7]
```

```
1 l1.index(4) # it will return the index position of item 4
```

```
→ 3
```

```
1 zz
```

```
→ [8, 7, 6, 5, 4]
```

```
1 zz.index(6) # it will give the index number of '6' , note: a  
2 # while in array list itme in contiues order like 101,102,10  
3 # so here '6' stored index or reference is 2
```

```
→ 2
```

```
1 l1=[543,5,6,7,8,9,10]  
2 l1
```

```
→ [543, 5, 6, 7, 8, 9, 10]
```

```
1 l1[l1.index(543)]
```

```
→ 543
```

```
1 "waqas khan software".title() # it will capatalize the first
```

```
→ 'Waqas Khan Software'
```

```
1 'Pakistan is our home country'.title()
```

```
→ 'Pakistan Is Our Home Country'
```

## 1 Task1

Build custom logic and capatalize the first letter of each word The split() method splits a string into a list of substrings based on a delimiter. By default, it splits the string using whitespace characters (spaces, tabs, newlines) as delimiters.

```
1 sample="hello how are you"
2 lst=sample.split()
3 lenght=len(sample.split())
4 print(f'lenght is {lenght}')
5 # will convert string into list of sub-strings
```

```
→ lenght is 4
```

```
1 # Sentence to words
2 sample1=' We here to learn python '
3 spt=sample1.split() # it will convert sentence to words
4 spt
```

```
→ ['We', 'here', 'to', 'learn', 'python']
```

```
1 # Count the Words of sentence
2 sample1=' We here to learn python '
3 spt=sample1.split()
4 lent=len(sample1.split()) # it will find the lenght of sentence
5 lent
6 print(f'lenght is {lent}')
```

```
→ lenght is 5
```

```
1 nn='we are all in python class'
2 print(nn.split())
```

```
3
```

```
→ ['we', 'are', 'all', 'in', 'python', 'class']
```

```
1 for i in nn.split():
2     print(i.capitalize())
```

```
→ <built-in method capitalize of str object at 0x000002AE6C2EDBB0>
<built-in method capitalize of str object at 0x000002AE6C3298F0>
<built-in method capitalize of str object at 0x000002AE6C329A70>
<built-in method capitalize of str object at 0x000002AE6C2D5A30>
<built-in method capitalize of str object at 0x000002AE6C317CF0>
<built-in method capitalize of str object at 0x000002AE6C314A30>
```

```
1 sample="hello how are you"
2 print(sample.split())
```

```
→ ['hello', 'how', 'are', 'you']
```

```
1 for i in sample.split():          # i is every word
2     print(i.capitalize())
```

```
→ <built-in method capitalize of str object at 0x000002AE6C2572F0>
<built-in method capitalize of str object at 0x000002AE6C2EDBB0>
<built-in method capitalize of str object at 0x000002AE6C28FCF0>
<built-in method capitalize of str object at 0x000002AE6C35F730>
```

```
1 ls=[]
2 for i in sample.split():          # i is every word
3     print(i.capitalize())
4     ls.append(i.capitalize())
5 ls
6
```

```
→ Hello
How
Are
You
['Hello', 'How', 'Are', 'You']
```

```
1 cap=[]
2 for i in nn.split():
3     print(i.capitalize())
```

```
4     cap.append(i.capitalize())
5 cap
```

→ We  
Are  
All  
In  
Python  
Class  
['We', 'Are', 'All', 'In', 'Python', 'Class']

```
1 #If you call " ".join(ls), it will join each word in the lis
2 #effectively creating a single string where words are separa
3 print(" ".join(ls))
```

→ Hello How Are You

```
1
2 print(" ".join(cap))
```

→ we are all in python class  
We Are All In Python Class

```
1 # Task2; extract atif from atif@gmail.com
2 str1= "atif@yahoo.com"
3 username=str1[:str1.index('@')]
4
5 password= str1[str1.index('@')+1:]
6 print(f'username is {username} and password is {password}')
```

→ username is atif and password is yahoo.com

```
1 email='programmarself@gmail.com'
2 u_name=email[:email.index('@')] # it will print the user name
3 p_word=email[email.index('@')+1:] # it will print the password
4 print(f'User Name is : {u_name} and Password is : {p_word}')
```

→ User Name is : programmarself and Password is : gmail.com

```
1 lst=[4,5,6,7,8,9,0,99,10]
2 lst[lst.index(99)] # it means that when we want to get the s
```

→ 99

```
1 str1= "atif@gmail.com"
2 str1[:str1.find('@')]
```

→ 'atif'

```
1 email='programamrself@gmail.com'
2 email[:email.find('@')]
```

→ 'programamrself'

```
1 u='we love python'
2 u[:u.find('love')] # if colon ': ' in the start of the list
3 # or it will print the word before the 'love'
```

→ 'we '

```
1 u='we love python'
2 u[u.find('love'): ] # if colon ': ' in the end of the prenthesis
3 # or it will print the word aftr the 'love'
```

→ 'love python'

```
1 u='we love python'
2 u[:u.index('love')]
```

→ 'we '

```
1 # how to remove the duplicate value
2 l=[1,1,2,2,3,3,4,4]
3 tmp=[]
4 for i in l:
5     if i not in tmp:
6         tmp.append(i)
7 tmp
```

→ [1, 2, 3, 4]

```
1 dup=[1,2,1,1,2,3,2,3,4,3,4]
2 tmp=[]
3 for i in dup:
```

```
4     if i not in tmp:  
5         tmp.append(i)  
6 tmp
```

```
→ [1, 2, 3, 4]
```

```
1 sol=['a','a','b','a','b','c','c']  
2 tmp=[]  
3 for i in sol:  
4     if i not in tmp:  
5         tmp.append(i)  
6 tmp
```

```
→ ['a', 'b', 'c']
```

```
1 sol=['a','a','b','a','b','c','cdddd']  
2 tmp=[]  
3 for i in sol:  
4     if i not in tmp:  
5         tmp.append(i) # can not remove under ''  
6 tmp
```

```
→ ['a', 'b', 'c', 'cdddd']
```

## ▼ Notebook: 10) Python Files.ipynb

### Python Files

Python provides various functions to perform different file operations, a process known as File Handling.

```
1 #Here are a few examples of opening a file in different mode
2
3 # open a file in default mode (read and text)
4 #file1 = open("test.txt")           # equivalent to open("test.tx
5
6 # open a file in write and text mode
7 f = open("test.txt",'w')
8
9 # open a file in read, write and binary mode
10 #file1 = open("img.bmp",'+b')
```

```
1 f.write('my name is Ihsan1\n')
2 f.write('my name is Ihsan2\n')
3 f.close()
```

```
1 file1 = open("test.txt")
2 print(file1.read())
```

→ my name is Ihsan1  
my name is Ihsan2

```
1 #Opening a File Using its Full Path
2 #We can also open a file using its full path.
3
4 file_path = 'D:/BS_work/Spring 2023_Database/All Oracle SQL
5 f = open(file_path)
6 print(f.read())
```

→ SQL> select \* from emp where job='MANAGER' and deptno=20 and sal>=2000;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7566	JONES	MANAGER	7839	02-APR-81	2975		20

SQL> select \* from emp;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30

7698	BLAKE	MANAGER	7839	01-MAY-81	2850	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	10
7788	SCOTT	ANALYST	7566	19-APR-87	3000	20
7839	KING	PRESIDENT		17-NOV-81	5000	10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0
7876	ADAMS	CLERK	7788	23-MAY-87	1100	20
7900	JAMES	CLERK	7698	03-DEC-81	950	30
7902	FORD	ANALYST	7566	03-DEC-81	3000	20
7934	MILLER	CLERK	7782	23-JAN-82	1300	10

14 rows selected.

SQL> select ename, job, sal,sal\*20/100, deptno from emp;

ENAME	JOB	SAL	SAL*20/100	DEPTNO
SMITH	CLERK	800	160	20
ALLEN	SALESMAN	1600	320	30
WARD	SALESMAN	1250	250	30
JONES	MANAGER	2975	595	20
MARTIN	SALESMAN	1250	250	30
BLAKE	MANAGER	2850	570	30
CLARK	MANAGER	2450	490	10
SCOTT	ANALYST	3000	600	20
KING	PRESIDENT	5000	1000	10
TURNER	SALESMAN	1500	300	30
ADAMS	CLERK	1100	220	20
JAMES	CLERK	950	190	30
FORD	ANALYST	3000	600	20
MILLER	CLERK	1300	260	10

14 rows selected.

SQL> select ename, job, sal,sal\*20/100 "Tax", deptno from emp;

ENAME	JOB	SAL	Tax	DEPTNO
SMITH	CLERK	800	160	20
ALLEN	SALESMAN	1600	320	30
WARD	SALESMAN	1250	250	30
JONES	MANAGER	2975	595	20

```
1 file_path = 'D:\\BS_work\\Spring 2023_Database\\All Oracle S
2 f = open(file_path)
3 print(f.read())
```

→ SQL> select \* from emp where job='MANAGER' and deptno=20 and sal>=2000;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7566	JONES	MANAGER	7839	02-APR-81	2975		20

SQL> select \* from emp;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

14 rows selected.

SQL> select ename, job, sal,sal\*20/100, deptno from emp;

ENAME	JOB	SAL	SAL*20/100	DEPTNO
SMITH	CLERK	800	160	20
ALLEN	SALESMAN	1600	320	30
WARD	SALESMAN	1250	250	30
JONES	MANAGER	2975	595	20
MARTIN	SALESMAN	1250	250	30
BLAKE	MANAGER	2850	570	30
CLARK	MANAGER	2450	490	10
SCOTT	ANALYST	3000	600	20
KING	PRESIDENT	5000	1000	10
TURNER	SALESMAN	1500	300	30
ADAMS	CLERK	1100	220	20
JAMES	CLERK	950	190	30
FORD	ANALYST	3000	600	20
MILLER	CLERK	1300	260	10

14 rows selected.

SQL> select ename, job, sal,sal\*20/100 "Tax", deptno from emp;

ENAME	JOB	SAL	Tax	DEPTNO
SMITH	CLERK	800	160	20
ALLEN	SALESMAN	1600	320	30
WARD	SALESMAN	1250	250	30
JONES	MANAGER	2975	595	20
MARTIN	SALESMAN	1250	250	30

```
1 f=open('waqas.txt','w')
2 f.write('I am Sick\n')
3 f.write('My mood off\n,')
```

```
4 f.write('I am tired\n')
5 f.write('I am hungry')
6 f.close()
7
```

```
1 f=open('waqas.txt','r')
2 print(f.read())
3 f.close()
```

→ I am Sick  
My mood off,I am tired,I am hungry

```
1 #Writing to Files in Python
2 #To write to a Python file, we need to open it in write mode
3 #Be Careful While Writing to a File
4 #If we try to perform the write operation to a file that alr
5
6 f= open('atif.txt','w') #his function returns a file object,
7 f.write('Today we will start python file handling\n') # writ
8 f.write('try except clause in exception\n')
9 f.write('else and finally block in exception\n')
10 f.close()
11 #Note: Closing a file will free up the resources that are ti
12
```

```
1 f= open('atif.txt','r')
2 print(f.readline())
3 print(f.readline())
4 print(f.readline())
5 f.close()
```

→ Today we will start python file handling  
try except clause in exception  
else and finally block in exception

```
1 f= open('waqas.txt','r')
2 print(f.readline())
3 print(f.readline())
```

```
4 print(f.readline())
5 f.close()
```

→ I am Sick

```
My mood off
,I am tired
```

```
1 # To append the data, you need to open the file in 'a' mode
2
3 f= open('atif.txt','a')
4 f.write('Some students have done their assignments\n') # wri
5 f.write('While others are still strugling to complete it\n')
6 f.write('Classses in exception\n')
7 f.close()
```

```
1 # Reading all data at once
2 f= open('atif.txt','r')
3 contents=f.read() # read entire content of file at once
4 print(contents)
5 f.close()
```

→ Today we will start python file handling  
try except clause in exception  
else and finally block in exception  
Some students have done their assignments  
While others are still strugling to complete it  
Classses in exception  
Some students have done their assignments  
While others are still strugling to complete it  
Classses in exception  
Some students have done their assignments  
While others are still strugling to complete it  
Classses in exception

```
1 # Reading all lines as an array
2 f= open('atif.txt','r')
3 contents=f.readlines()
4 print(contents)
5 f.close()
```

→ ['Today we will start python file handling\n', 'try except clause in exception\n', 'else

```
1 for line in contents:  
2     print(line)
```

→ Today we will start python file handling

```
try except clause in exception  
  
else and finally block in exception  
  
Some students have done their assignments  
  
While others are still strugling to complete it  
  
Classses in exception  
  
Some students have done their assignments  
  
While others are still strugling to complete it  
  
Classses in exception  
  
Some students have done their assignments  
  
While others are still strugling to complete it  
  
Classses in exception
```

```
1 # Reading the first line or raading one line at a time  
2 f= open('atif.txt','r')  
3 print(f.readline() ) # The readline() method is called on thi  
4 print(f.readline()) #The readline() method is called again o  
5 f.close()
```

→ Today we will start python exception

```
try except clause in exception
```

```
1 # looping through the data using for loop  
2 # we can iterate through the file using file pointer  
3 f= open('atif.txt','r')  
4 #print(type(f))  
5  
6 for text in f:
```

```
7     print(text)
8 f.close()
```

→ Today we will start python file handling

try except clause in exception

else and finally block in exception

Some students have done their assignments

While others are still strugling to complete it

Classses in exception

Some students have done their assignments

While others are still strugling to complete it

Classses in exception

Some students have done their assignments

While others are still strugling to complete it

Classses in exception

## Opening a Python File Using with...open

In Python, there is a better way to open a file using with...open. For example,

The code uses the with statement to open the file atif.txt in read mode ("r") and creates a file object (file1).

The with statement ensures that the file is automatically closed after the block of code inside the with statement is executed.

This is a good practice because it ensures that resources are properly managed.

```
1 #Here, with...open automatically closes the file, so we don't
2 with open("atif.txt", "r") as file1:
3     read_content = file1.read()
4     print(read_content)
5
```

→ Today we will start python file handling

try except clause in exception

else and finally block in exception

Some students have done their assignments

While others are still strugling to complete it

Classses in exception

```
Some students have done their assignments  
While others are still struggling to complete it  
Classes in exception
```

## Exception Handling in Files

If an exception occurs while working with a file, the code exits without closing the file. Hence, it's a good practice to use the try...finally block.

```
1 file1 = open("mubeen.txt", "r")  
2 read_content = file1.read()  
3 print(read_content)  
4
```

```
FileNotFoundError                                     Traceback (most recent call last)  
Cell In[25], line 1  
----> 1 file1 = open("mubeen.txt", "r")  
      2 read_content = file1.read()  
      3 print(read_content)  
  
File ~\AppData\Local\Programs\Python\Python312\Lib\site-  
packages\IPython\core\interactiveshell.py:324, in _modified_open(file, *args, **kwargs)  
  317 if file in {0, 1, 2}:  
  318     raise ValueError(  
  319         f"IPython won't let you open fd={file} by default "  
  320         "as it is likely to crash IPython. If you know what you are doing, "  
  321         "you can use builtins' open."  
  322     )  
--> 324 return io_open(file, *args, **kwargs)  
  
FileNotFoundError: [Errno 2] No such file or directory: 'mubeen.txt'
```

```
1 try:  
2     file1 = open("mubeen.txt", "r")  
3     read_content = file1.read()  
4     print(read_content)  
5  
6 except FileNotFoundError as e: #This exception occurs when t  
7     print(f"Error: The file was not found. Details: {e}")  
8  
9 finally:  
10    # close the file
```

```
11     print("hello")
12     file1.close() #Here, the finally block is always execute
```

→ Error: The file was not found. Details: [Errno 2] No such file or directory: 'mubeen.txt'  
hello



The current\_position obtained from file.tell() in Python indicates the current byte offset from the beginning of the file. In other words, it represents the current position of the file object's cursor or pointer within the file.

When you open a file in a particular mode (e.g., read or write), a file object is created, and the file's internal cursor is positioned at the start of the file. As you read from or write to the file, the cursor moves forward within the file. When you call file.tell(), it returns the current byte offset of the cursor from the start of the file. This tells you the position in the file where the next read or write operation will occur. For example:

If you read a certain number of bytes or lines from a file, file.tell() will give you the current position in bytes of where the next read operation will start. Similarly, if you write data to a file, file.tell() will provide the current position in bytes of where the next write operation will start. This information is useful for tracking the file object's position and for seeking back to a specific position in the file using file.seek() if necessary.

File's beginning: The beginning of the file is the very start of the file, where the first byte of data is stored. Byte offset: An offset is a measure of distance, and in this context, it refers to the number of bytes from the start of the file to the current position. For instance, if you are at the beginning of the file, the byte offset is 0. As you read through the file, the byte offset increases. Current byte offset: This is the position of the file object's cursor at the moment you call tell(). It tells you where in the file the cursor is, in terms of bytes from the start of the file. So, when you use the tell() method and get a value, this value is the current byte offset from the beginning of the file. It indicates how many bytes you have read from the start of the file to reach the current position of the cursor. For example, if tell() returns 20, it means you are currently at the 20th byte from the start of the file.

```
1 # Open the file in read mode
2 with open("atif.txt", "r") as file:
3     # Read the first line
4     first_line = file.readline()
5     print("First line:", first_line)
```

```
6  
7     # Get the current position of the file object  
8     current_position = file.tell()  
9     print("Current position in the file:", current_position)  
10  
11    # Read the rest of the file  
12    rest_of_file = file.read()  
13    print("Rest of the file:", rest_of_file)  
14
```

→ First line: Today we will start python file handling

```
Current position in the file: 42  
Rest of the file: try except clause in exception  
else and finally block in exception  
Some students have done their assignments  
While others are still strugling to complete it  
Classses in exception
```

```
1 # Open the file in write mode  
2 with open("output_file.txt", "w") as file:  
3     # Define a string to write to the file  
4     string_to_write = "This is a test string.\nthis is another test string.  
5  
6     # Write the string to the file  
7     file.write(string_to_write)  
8  
9     print(f"'{string_to_write.strip()}' was written to the file")  
10
```

→ 'This is a test string.  
this is another test string.' was written to the file.

```
1 # Open the file in write mode  
2 with open("output_file_lines.txt", "w") as file:  
3     # Define a list of lines to write to the file  
4     lines_to_write = [  
5         "Line 1: Hello, world!\n",  
6         "Line 2: This is a test.\n",  
7         "Line 3: Python is great.\n"  
8     ]
```

```
9  
10     # Write the list of lines to the file  
11     file.writelines(lines_to_write)  
12  
13     print(f"{len(lines_to_write)} lines were written to the file")  
14
```

→ 3 lines were written to the file.

image.png image.png

```
import csv
```

Then, the `csv.reader()` is used to read the file, which returns an iterable reader object.

The reader object is then iterated using a for loop to print the contents of each row.

```
with open('people.csv', 'r') as file:  
    reader = csv.reader(file)
```

```
    for row in reader:  
        print(row)
```

## Write to CSV Files with Python

The `csv` module provides the `csv.writer()` function to write to a CSV file.

Let's look at an example.

```
1 !pip install csv
```

```
1 import csv
```

```
1 #!pip install csv
2 import csv
3 with open('students.csv','r') as fg: # fg is refering to csv
4     rd= csv.reader(fg) # reader returns reader object--itera
5     for row in rd:
6         print(row)
```

```
→ ['Student_name', 'CGPA']
  ['Farhan', '3.8']
  ['Iqrar', '3.8']
  ['Usman', '3.8']
```

```
1 import csv
2 with open('Navtac.csv', 'w', newline='') as file:
3     writer = csv.writer(file)
4     #rites the header row with column names to the CSV file.
5     writer.writerow(["SN", "Name", "Address"])
6     #writes the first data row to the CSV file.
7     writer.writerow([1, "ESSa", "Peshawar"])
8     writer.writerow([2, "Iqrar", "Hangu"])
9     writer.writerow([3, "fazal", "Swabi"])
```

```
1 import csv
2 with open('Navtac.csv','r') as fg: # fg is refering to csv o
3     rd= csv.reader(fg) # reader returns reader object--itera
4     for row in rd:
5         print(row)
```

```
→ ['SN', 'Name', 'Address']
  ['1', 'ESSa', 'Peshawar']
  ['2', 'Iqrar', 'Hangu']
  ['3', 'fazal', 'Swabi']
```



```
1 import csv
2 #we have opened the innovators.csv file in writing mode usin
3 #newline='':
4 #The newline parameter is set to an empty string ('').
5 #This parameter is important when writing to a CSV file in P
6 #you may encounter issues such as extra blank lines being ad
```

```
7
8 with open('innovators.csv', 'w', newline='') as file:
9     #Next, the csv.writer() function is used to create a writer object
10    writer = csv.writer(file)
11
12    #The writer.writerow() function is then used to write single rows
13    writer.writerow(["SN", "Name", "Contribution"])
14
15    writer.writerow([1, "Linus Torvalds", "Linux Kernel"])
16    writer.writerow([2, "Tim Berners-Lee", "World Wide Web"])
17    writer.writerow([3, "Guido van Rossum", "Python Programming"])
```

## Writing Multiple Rows with writerows()

If we need to write the contents of the 2-dimensional list to a CSV file, here's how we can do it.

```
1 #If we need to write the contents of the 2-dimensional list
2 import csv
3 row_list = [[{"SN": 1, "Name": "Linus Torvalds", "Contribution": "Linux Kernel"}, {"SN": 2, "Name": "Tim Berners-Lee", "Contribution": "World Wide Web"}, {"SN": 3, "Name": "Guido van Rossum", "Contribution": "Python Programming"}]
4
5 with open('employees.csv', 'w', newline='') as file:
6     w=csv.writer(file)
7     w.writerows(row_list)
8
9
10
```

```
1 with open('employees.csv', 'r') as file:
2     reader=csv.reader(file)
3     for row in reader:
4         print(';'.join(row))
5
```

→ SN;Name;Contribution  
1;Linus Torvalds;Linux Kernel  
2;Tim Berners-Lee;World Wide Web  
3;Guido van Rossum;Python Programming



```
1 #Write CSV File Having Pipe Delimiter
2 import csv
3 data_list = [[{"SN": "Name", "Contribution": ""}],
4              [1, "Linus Torvalds", "Linux Kernel"],
5              [2, "Tim Berners-Lee", "World Wide Web"],
6              [3, "Guido van Rossum", "Python Programming"]]
7
8 with open('innovators_pipe.csv', 'w', newline='') as file:
9
10    writer = csv.writer(file, delimiter='|')
11    writer.writerows(data_list)
```

```
1 with open('innovators_pipe.csv', 'r') as file:
2     reader = csv.reader(file)
3
4     for row in reader:
5         print(row)
```

```
→ ['SN|Name|Contribution']
['1|Linus Torvalds|Linux Kernel']
['2|Tim Berners-Lee|World Wide Web']
['3|Guido van Rossum|Python Programming']
```

```
1 #Write CSV files with quotes
2 import csv
3 row_list = [
4     [{"SN": "Name", "Profession": ""}],
5     [1, "Iqrar", "Manager"],
6     [2, "Uzair Khan", "CEO Google"],
7     [3, "Asnad", "CEO Tesla"]
8 ]
9 with open('quotes.csv', 'w', newline='') as file:
10    writer = csv.writer(file, quoting=csv.QUOTE_NONNUMERIC,
11    writer.writerows(row_list)
```

```
1 with open('quotes.csv', 'r') as file:
2     reader = csv.reader(file, quoting=csv.QUOTE_NONNUMERIC,
```

```
3  
4     for row in reader:  
5         print(row)
```

```
→ ['SN', 'Name', 'Profession']  
[1.0, 'Iqrar', 'Manager']  
[2.0, 'Uzair Khan', 'CEO Google']  
[3.0, 'Asnad', 'CEO Tesla']
```

```
1 #Read CSV files with quotes  
2 import csv  
3 with open('quotesdf.csv', 'r') as file:  
4     reader = csv.reader(file)  
5     for row in reader:  
6         print(row)
```

```
→ ['SN', ' "Name"', ' "Quotes"']  
['1', ' Buddha', ' "What we think we become"']  
['2', ' Mark Twain', ' "Never regret anything that made you smile"']  
['3', ' Oscar Wilde', ' "Be yourself everyone else is already taken"']
```

```
1 #Read CSV files with quotes  
2 import csv  
3 with open('quotesdf.csv', 'r') as file:  
4     reader = csv.reader(file, skipinitialspace=True)  
5     for row in reader:  
6         print(row)
```

```
→ ['SN', 'Name', 'Quotes']  
['1', 'Buddha', 'What we think we become']  
['2', 'Mark Twain', 'Never regret anything that made you smile']  
['3', 'Oscar Wilde', 'Be yourself everyone else is already taken']
```

```
1 #Read CSV files with quotes  
2 import csv  
3 with open('quotesdf.csv', 'r') as file:  
4 #we have passed csv.QUOTE_ALL to the quoting parameter. It is  
5 #csv.QUOTE_ALL specifies the reader object that all the values  
6  
7 csv.QUOTE_ALL specifies the reader object that all the values  
8     reader = csv.reader(file, quoting=csv.QUOTE_ALL, skipinitialspace=True)
```

```
9     for row in reader:  
10         print(row)
```

```
→ ['SN', 'Name', 'Quotes']  
['1', 'Buddha', 'What we think we become']  
['2', 'Mark Twain', 'Never regret anything that made you smile']  
['3', 'Oscar Wilde', 'Be yourself everyone else is already taken']
```

```
1  
2 with open('quotes.csv', 'r', newline='') as file:  
3     reader = csv.reader(file, quoting=csv.QUOTE_NONNUMERIC,  
4     for row in reader:  
5         # Convert each row list to a string and print it  
6         print(";" .join([str(field) for field in row]))
```

```
→ SN;Name;Quotes  
1.0;Buddha;What we think we become  
2.0;Mark Twain;Never regret anything that made you smile  
3.0;Oscar Wilde;Be yourself everyone else is already taken
```

```
1 #Write CSV files without quotes  
2 import csv  
3 row_list = [  
4     ["SN", "Name", "Quotes"],  
5     [1, "Buddha", "What we think we become"],  
6     [2, "Mark Twain", "Never regret anything that made you si  
7     [3, "Oscar Wilde", "Be yourself everyone else is already  
8 ]  
9 with open('withoutquotes.csv', 'w', newline='') as file:  
10    writer = csv.writer(file, quoting=csv.QUOTE_NONE, delimi  
11    writer.writerows(row_list)
```

### Reader Output:

1. The `csv.reader` object returns each row as a list of strings. This is consistent with how the reader works:
2. each row in the file is split into fields based on the specified delimiter (;), and the fields are returned as a list.

### Desired Output:

3. To achieve the output format you specified (SN;Name;Quotes, etc.), you need to convert each list of fields into a single string with semicolons (;) separating the fields.

4. You can achieve this by using the join() method to join the fields in each list with a semicolon (;) and then print the resulting string.

```
1 with open('withoutquotes.csv', 'r', newline='') as file:  
2     reader = csv.reader(file, delimiter=';')  
3     for row in reader:  
4  
5         print(row)
```

```
→ ['SN', 'Name', 'Quotes']  
['1', 'Buddha', 'What we think we become']  
['2', 'Mark Twain', 'Never regret anything that made you smile']  
['3', 'Oscar Wilde', 'Be yourself everyone else is already taken']
```

```
1 with open('withoutquotes.csv', 'r', newline='') as file:  
2     reader = csv.reader(file, quoting=csv.QUOTE_NONE, delimiter=';')  
3     for row in reader:  
4         # Convert each row list to a string with semicolons  
5         print(';'.join(row))  
6 #The join() method is called on the semicolon string (';'),  
7 #The join() method takes the list of strings and concatenate  
8 #The resulting string is '1;Buddha;What we think we become'.
```

```
→ SN;Name;Quotes  
1;Buddha;What we think we become  
2;Mark Twain;Never regret anything that made you smile  
3;Oscar Wilde;Be yourself everyone else is already taken
```



```
1 #Writing CSV files with custom quoting character  
2 import csv  
3 row_list = [  
4     ["SN", "Name", "Quotes"],  
5     [1, "Buddha", "What we think we become"],  
6     [2, "Mark Twain", "Never regret anything that made you smile"],  
7     [3, "Oscar Wilde", "Be yourself everyone else is already taken"]  
8 ]  
9 #Here, we can see that quotechar='*' parameter instructs the  
10 with open('quotes_quotechar.csv', 'w', newline='') as file:  
11     writer = csv.writer(file, quoting=csv.QUOTE_NONNUMERIC,
```

```
12                         delimiter=';', quotechar='*')
13     writer.writerows(row_list)
```

```
1 with open('quotes_quotech.csv', 'r', newline='') as file:
2     reader = csv.reader(file, delimiter=';')
3     for row in reader:
4         #print(row)
5         print(';'.join(row))
```

→ \*SN\*;\*Name\*;\*Quotes\*
1;\*Buddha\*;\*What we think we become\*
2;\*Mark Twain\*;\*Never regret anything that made you smile\*
3;\*Oscar Wilde\*;\*Be yourself everyone else is already taken\*



1. While creating the writer object, we pass dialect='myDialect' to specify that the writer instance must use that particular dialect.
2. The advantage of using dialect is that it makes the program more modular. Notice that we can reuse myDialect to write other CSV files without having to re-specify the CSV format.

```
1 import csv
2 row_list = [
3     ["ID", "Name", "Email"],
4     [878, "Alfonso K. Hamby", "alfonsokhamby@rhyta.com"],
5     [854, "Susanne Briard", "susannebriard@armyspy.com"],
6     [833, "Katja Mauer", "kmauer@jadoop.com"]
7 ]
8 #we can see that the csv.register_dialect() function is used
9 csv.register_dialect('farhanDialect',
10                     delimiter='|',
11                     quoting=csv.QUOTE_ALL)
12
13 with open('office.csv', 'w', newline='') as file:
14     writer = csv.writer(file, dialect='farhanDialect')
15     writer.writerows(row_list)
```

```
1 with open('office.csv', 'r', newline='') as file:  
2     reader = csv.reader(file, dialect='farhanDialect')  
3     for row in reader:  
4         print(row)  
5     # print('|'.join(row))
```

```
→ ['ID', 'Name', 'Email']  
['878', 'Alfonso K. Hamby', 'alfonsokhamby@rhyta.com']  
['854', 'Susanne Briard', 'susannebriard@armyspy.com']  
['833', 'Katja Mauer', 'kmauer@jadoop.com']
```

```
1 #The CSV file has initial spaces, quotes around each entry,  
2 #Instead of passing three individual formatting patterns, le  
3  
4 with open('office.csv', 'r', newline='') as file:  
5     reader = csv.reader(file, dialect='myDialect')  
6     for row in reader:  
7         print(row)  
8     #print('|'.join(row))
```

```
→ ['ID', 'Name', 'Email']  
['A878', 'Alfonso K. Hamby', 'alfonsokhamby@rhyta.com']  
['F854', 'Susanne Briard', 'susannebriard@armyspy.com']  
['E833', 'Katja Mauer', 'kmauer@jadoop.com']
```



Then, the `csv.reader()` is used to read the file, which returns an iterable reader object.

The reader object is then iterated using a for loop to print the contents of each row.

```
1 import csv  
2 with open('innovators.csv', 'r') as file:  
3     reader = csv.reader(file)  
4     for row in reader:  
5         print(row)
```

```
→ ['SN', 'Name', 'Contribution']  
['1', 'Linus Torvalds', 'Linux Kernel']  
['2', 'Tim Berners-Lee', 'World Wide Web']  
['3', 'Guido van Rossum', 'Python Programming']
```



```
1 #Example 2: Read CSV file Having Tab Delimiter
2 #As we can see, the optional parameter delimiter = '\t' help
3 import csv
4 with open('innovators-tab.csv', 'r') as file:
5     reader = csv.reader(file, delimiter = '\t')
6     for row in reader:
7         print(row)
```

```
→ ['SN', 'Name', 'Contribution']
['1', 'Linus Torvalds', 'Linux Kernel']
['2', 'Tim Berners-Lee', 'World Wide Web']
['3', 'Guido van Rossum', 'Python Programming']
```



```
1 #Read CSV files with initial spaces
2 import csv
3 with open('people.csv', 'r') as csvfile:
4     reader = csv.reader(csvfile, skipinitialspace=True) #This
5     for row in reader:
6         print(row)
```

```
→ ['SN', 'Name', 'City']
['1', 'John', 'Washington']
['2', 'Eric', 'Los Angeles']
['3', 'Brad', 'Texas']
```

### CSV files with Custom Delimiters

By default, a comma is used as a delimiter in a CSV file. However, some CSV files can use delimiters other than a comma. Few popular ones are | and \t.

Suppose the innovators.csv file in Example 1 was using tab as a delimiter.

To read the file, we can pass an additional delimiter parameter to the csv.reader() function. the optional parameter delimiter = '\t' helps specify the reader object that the CSV file we are reading from, has tabs as a delimiter.

```
1 import csv
2 with open('innovators.csv', 'r') as file:
3     reader = csv.reader(file, delimiter = '\t')
4     for row in reader:
5         print(row)
```

```
→ ['SN,Name,Contribution']
['1,Linus Torvalds,Linux Kernel']
['2,Tim Berners-Lee,World Wide Web']
['3,Guido van Rossum,Python Programming']
```



The writeheader() method is called on the writer object to write a header row to the CSV file.

1. The header row contains the field names specified in fieldnames (player\_name and fide\_rating).
2. This row is written as the first row in the CSV file.

```
1 #Python csv.DictWriter()
2 import csv
3
4 with open('students.csv', 'w', newline='') as file:
5     heading_names = ['Student_name', 'CGPA']
6     #A csv.DictWriter object is created using the file object
7     writer = csv.DictWriter(file, fieldnames=heading_names)
8
9     writer.writeheader()
10    writer.writerow({'Student_name': 'Farhan', 'CGPA': 3.8})
11    writer.writerow({'Student_name': 'Iqrar', 'CGPA': 3.8})
12    writer.writerow({'Student_name': 'Usman', 'CGPA': 3.8})
```

```
1 #Read CSV files with csv.DictReader()
2 #The objects of a csv.DictReader() class can be used to read
3 import csv
4 with open("students.csv", 'r') as file:
5     csv_file = csv.DictReader(file)
6     for row in csv_file:
7         print(row)
8         #print(dict(row))
```

```
→ { 'Student_name': 'Farhan', 'CGPA': '3.8' }
  { 'Student_name': 'Iqrar', 'CGPA': '3.8' }
  { 'Student_name': 'Usman', 'CGPA': '3.8' }
```

```
1 # Specify the path to the output CSV file
2 #script that demonstrates how to create a CSV file that uses
3 #the columns S.No, name, and city:
4 csv_file_path = "people_delimeter_space.csv"
5
6 # Define the data to write to the CSV file
7 data = [
8     [1, "John Doe", "New York"],
9     [2, "Jane Smith", "Los Angeles"],
10    [3, "Alice Brown", "Chicago"],
11 ]
12
13 # Define the delimiter to be a comma followed by a space
14 delimiter = ", "
15
16 # Open the file in write mode
17 with open(csv_file_path, 'w') as file:
18     header = ["S.No", "name", "city"]
19 # Join the header list using the delimiter and add a newline
20     file.write(delimiter.join(header) + '\n')
21
22     # Write the data rows
23     for row in data:
24         # Convert each element in the row to a string, join
25         file.write(delimiter.join(map(str, row)) + '\n')
26
27 print(f"CSV file '{csv_file_path}' has been created with a d
28
```

```
→ CSV file 'people_delimeter_space.csv' has been created with a delimiter ', '.
```

```
1 import csv
2 with open('people_delimeter_space.csv', 'r') as file:
3     reader = csv.reader(file, quoting=csv.QUOTE_NONE, skipinitialspace=True)
4     # Iterate over each row in the CSV file
```

```
5     for row in reader:
6         print(row)
7             # Remove double quotes from each item in the row
8             # Use list comprehension to strip double quotes from
9             #cleaned_row = [item.strip('"'') for item in row]
10
11         # Print the cleaned row
12         #print(cleaned_row)
13
```

```
→ ['S.No', 'name', 'city']
['1', 'John Doe', 'New York']
['2', 'Jane Smith', 'Los Angeles']
['3', 'Alice Brown', 'Chicago']
```

```
1 def check_csv_delimiter(file_path):
2     # Define possible delimiters to check
3     possible_delimiters = [',', ';', '\t', '|']
4
5     # Read the first line of the CSV file
6     with open(file_path, 'r') as file:
7         first_line = file.readline()
8
9     # Initialize a dictionary to count occurrences of each d
10    delimiter_count = {delimiter: first_line.count(delimiter)}
11
12    # Find the delimiter with the maximum count
13    detectedDelimiter = max(delimiter_count, key=delimiter_
14
15    # Print the detected delimiter and its count in the firs
16    print(f"Detected delimiter: '{detectedDelimiter}' with"
17
18    return detectedDelimiter
19
20 # Specify the path to the input CSV file
21 csv_file_path = "innovators.csv"
22
23 # Check the delimiter used by the input CSV file
24 detectedDelimiter = check_csv_delimiter(csv_file_path)
25
```

→ Detected delimiter: ',' with count: 2

1 Start coding or generate with AI.

▼ Notebook: 11) getcwd().ipynb

```
1 import os  
2 os.getcwd()
```

→ 'E:\\Python For Advance Application'

```
1 os.chdir("E:\\Python For Advance Application\\notebooks")
```

```
1 os.getcwd()
```

→ 'E:\\Python For Advance Application\\notebooks'

1 Start coding or generate with AI.

▼ Notebook: 12) OOP1\_Lecture1.ipynb

1 Start coding or generate with AI.



1. Everything in Python is an object, and every object belongs to a class. For instance, there may be many LCD objects, so a given LCD object might come from the LG class (company). Integers, lists, and tuples are objects, and these objects belong to specific classes.
2. From now on, variables and objects can be considered the same thing.
3. A class is a blueprint that defines a data type. In Python, defining a variable of a particular data type creates an object of that class.
4. There are many different data types in Python, each developed by someone. Developing a data type involves defining its class or blueprint, which describes how an object will behave or function.

5. The behavior of an object is determined by the data members and methods included in its class.
6. Creating a class requires more time, effort, and code than creating an object.
7. For example, when you create a list object, you have access to a lot of methods because these methods are already defined by the developer when creating the list class.
8. Class consist of 2 things data: Description about an object (nouns) Function: behavior of an object (verb) Suppose you are building a class car; Data: No. of wheels, color, engine type, brand Functions: calculate mileage, calculate average speed, open air bag, show gps navigation

```
1 a=2      # a is the integers object, which belongs to class 'int'
2 print(type(a))
```

→ <class 'int'>

```
1 #
2 l=[1,2,3,4] # list object of class 'list'
3 print(type(l))
```

→ <class 'list'>



Here are three commonly used naming conventions for identifiers:

1. Pascal Case (a.k.a. Upper Camel Case): In Pascal case, each word in the identifier starts with an uppercase letter, and there are no underscores between words. Example: FirstName, TotalAmount, CalculateArea
2. Camel Case (a.k.a. Lower Camel Case): In camel case, the first word starts with a lowercase letter, and each subsequent word starts with an uppercase letter. Example: firstName, totalAmount, calculateArea
3. Snake Case: In snake case, all words are lowercase, and words are separated by underscores (\_). Example: first\_name, total\_amount, calculate\_area

```
1 Class Basic Structure
2 Class Car: # class name should be in Pascal case
3     color='Blue' # data...data members should be in snake
4     model='Sports' # data
```

```
5 def calculate_avg_speed(km, miles): # method names should be  
6     #some code
```

**Diagrammatic Representation of Class** private members of class are denoted by - sign while public members are denoted by + sign. Data members are usually private and are not accessible outside the class and methods are public and therefore accessible outside the class in the program



Unsupported Cell Type. Double-Click to inspect/edit the content.

1. If list is a class, then why we create list object like this
2. When you create a list using a literal expression like `lst = [2, 3, 45, 6, 5]`, Python internally calls `list()` to create an object of the list class.
3. Python provides a feature, object literal, which provides an easy way to create objects of built-in classes.
4. In Python, you often create objects of built-in classes (such as integers, lists, and dictionaries) using object literal or literal expressions, such as `42` for integers, `[]` for lists, and `{}` for dictionaries.
5. These object literals provide a straightforward and concise way to create instances of these built-in classes.
6. You can create list object like this as well, `lst = list()`, syntax similar to the one discussed in above cell
7. String object; `s = str()`

```
1 s=str("hello")  
2 print(s)
```

→ hello

```
1 l= list([2,3,4,5])  
2 print(l)
```

→ [2, 3, 4, 5]

We are developing Application/software of ATM using OOP. So we will need a representation/class of how customer behave when using ATM machine. For class, we would require data members, 1. account balance (when we deposit money, balance will be updated), when we withdraw, account balance will be deducted, similarly, balance check is also related to account balance

2. ATM Pin, without this, we can do nothing on ATM Behaviors/Methods: `create_pin()`, `deposit()`, `withdrawal()`, `check_bal()`
3. Difference between Function vs Method
4. A method is a special function defined within a class that can be accessed and called by an object of that class.
5. A function, on the other hand, refers to a general function that exists outside of a class and can be used independently of class instances.
6. Functions can operate on different data types, such as lists, strings, and integers.
7. For example, `len(l)` is a general function that works with various iterable data types, while `l.append(2)` calls the append method, which is a special method defined within the list class and can be accessed only by list objects."

extra Method is a special function defined inside class and can be accessed by object of that class  
 Function refers to normal/general function which does not exist in any class, can be used any user, and can be applied on list, string and integers. for example `len(l)` is a general function, while `l.append(2)`, `append` is special method defined inside list class and can be accessed only by list object.

```

1 #In Python, the len() function is used to determine the length of an iterable
2 #It is not used on integers directly because integers are not collections
3 #Therefore, calling len() on an integer will result in a TypeError
4 num = 1234
5 length = len(num) # This will raise a TypeError
6

```

→

TypeError	Traceback (most recent call last)
Cell In[4], line 4	
1 #In Python, the len() function is used to determine the length of an iterable or a collection, such as a list, string, or dictionary. It is not used on integers directly because integers are not collections and do not have a length.	
2 #Therefore, calling len() on an integer will result in a TypeError:	
3 num = 1234	
----> 4 length = len(num) # This will raise a TypeError	

TypeError: object of type 'int' has no len()

```

1 #If you want to determine the length of a string representation of an integer
2 #and then apply the len() function:
3 num = 1234
4 length = len(str(num)) # Convert the integer to a string and then determine its length

```

```
5 print(f"The length of {num} as a string is: {length}")
6
```

→ The length of 1234 as a string is: 4

1. variables in python are declared and initialized in constructor
2. **init** is a constructor, which is a special method defined in the class whose code is automatically executed when you create an object of class. Constructor name is same as class name in Java and C++, however, in python, it will be **init** we can prove this see below

```
1 class Atm:
2     def __init__(self): # constructor
3
4     def menu(self):
5         pass
6 obj= Atm()
```

1. In Python, instance variables are variables that are defined within a class but outside of any method, typically in the class's constructor (**init** method). Each instance of the class (each object) will have its own copy of these variables.
2. In the context of your Atm class, variables such as pin and balance are defined in the constructor (**init** method). This means that when you create a new instance of the Atm class, each instance will have its own copy of pin and balance:

```
1 class Atm:
2     def __init__(self):
3         self.pin=""
4         self.balance=0
5         self.menu()
6     def menu(self):
7         user_input= int(input(""""
8                         Hello, How would you like to proceed?
9                         1. Enter 1 to create pin
10                        2. Enter 2 to deposit balance
11                        3. Enter 3 to withdraw balance
12                        4. Enter 4 to check balance
13                        5. Enter 5 to exit
14                    """))
```

```
15     if user_input==1:
16         self.create_pin()
17         #print("pin created")
18     elif user_input==2:
19         self.deposit_balance()
20         #print("deposit balance")
21     elif user_input==3:
22         self.withdraw()
23         #print("withdraw balance")
24     elif user_input==4:
25         self.checkbalance()
26         print("cehk balance")
27     else:
28         print("bye")
29
30     def create_pin(self):
31         self.pin=input("Enter your pin")
32         print("pin created successfully")
33         self.menu()
34
35     def withdraw(self):
36         tmp=input("Please Enter your pin")
37         if (tmp==self.pin):
38             amt= int(input("Please Enter the amount u want t")
39             if amt<self.balance:
40                 self.balance=self.balance - amt
41                 print("Withdrawl successfully")
42             else:
43                 print("Invalid pin")
44             self.menu()
45
46     def deposit_balance(self):
47         tmp=input("Please Enter your pin")
48         if (tmp==self.pin):
49             amt= int(input("Please Enter the amount u want t")
50             self.balance=self.balance + amt
51             print("Amount deposited successfully")
52         else:
53             print("Invalid pin")
```

```
54         self.menu()
55     def checkbalance(self):
56         tmp=input("Please Enter your pin")
57         if (tmp==self.pin):
58             print(self.balance)
59
60         else:
61             print("Invalid pin")
62         self.menu()
```

```
1 obj=Atm()  
2 #obj.deposit_balance()  
3 #obj.withdraw()  
4 #obj.checkbalance()
```

3

Hello, How would you like to proceed?

1. Enter 1 to create pin
2. Enter 2 to deposit balance
3. Enter 3 to withdraw balance
4. Enter 4 to check balance
5. Enter 5 to exit

1

Enter your pin 5  
pin created successfully

Hello, How would you like to proceed?

1. Enter 1 to create pin
2. Enter 2 to deposit balance
3. Enter 3 to withdraw balance
4. Enter 4 to check balance
5. Enter 5 to exit

bye

```
10                      1. Enter 1 to create pin
11                      2. Enter 2 to deposit balance
12                      3. Enter 3 to withdraw balance
13                      4. Enter 4 to check balance
14                      5. Enter 5 to exit
15                      """)
16      if user_input=="1":
17          self.create_pin()
18      elif user_input=="2":
19          self.deposit_balance()
20      elif user_input=="3":
21          self.withdraw_balance()
22      elif user_input=="4":
23          self.check_balance()
24      else:
25          print("good bye")
26  def create_pin(self):
27      self.pin= input("Enter your pin")
28      print("pin set successfully")
29
30  def deposit_balance(self):
31      tmp= input("Enter your pin")
32      if (tmp==self.pin):
33          amt=int(input("Enter the amount you want to depo
34          self.balance=self.balance + amt
35          print("Amount deposited Successfully")
36      else:
37          print("Invalid Pin")
38
39  def withdraw_balance(self):
40      tmp= input("Enter your pin")
41      if (tmp==self.pin):
42          amt=int(input("Enter the amount you want to with
43          if amt < self.balance:
44              self.balance= self.balance - amt
45              print("Amount withdrawl Successfully")
46          else:
47              print("Insufficient funds")
48      else:
```

```
49         print("Invalid Pin")
50
51     def check_balance(self):
52         tmp= input("Enter your pin")
53         if (tmp==self.pin):
54             print(self.balance)
55         else:
56             print("Invalid Pin")
```

```
1 obj= Atm()
2 obj.deposit_balance()
```

```
→
Hello, How would you like to proceed?
1. Enter 1 to create pin
2. Enter 2 to deposit balance
3. Enter 3 to withdraw balance
4. Enter 4 to check balance
5. Enter 5 to exit
1
etnering
Enter your pin 1234
pin set successfully
Enter your pin 1234
Enter the amount you want to deposit 1000
Amount deposited Successfully
```

```
1 obj.check_balance()
```

```
→ -----
NameError                                                 Traceback (most recent call last)
Cell In[3], line 1
----> 1 obj.check_balance()

NameError: name 'obj' is not defined
```

```
1 obj.withdraw_balance()
2 obj.check_balance()
```

```
→ Enter your pin 1234
Enter the amount you want to withdraw 500
Amount withdrawl Successfully
Enter your pin 1234
500
```

```
1 obj.balance
```

```
→ -----
NameError                                 Traceback (most recent call last)
Cell In[1], line 1
----> 1 obj.balance

NameError: name 'obj' is not defined
```

```
1 obj2= Atm()
2 obj2.deposit_balance()
```

```
→
Hello, How would you like to proceed?
1. Enter 1 to create pin
2. Enter 2 to deposit balance
3. Enter 3 to withdraw balance
4. Enter 4 to check balance
5. Enter 5 to exit
1
etnering
Enter your pin 3456
pin set successfully
Enter your pin 3456
Enter the amount you want to deposit 90000
Amount deposited Successfully
```

```
1 obj2.check_balance()
```

```
→ Enter your pin 3456
90000
```

1. now our code is well managed, all the backened logic is written in class, now create objects and peform differnt jobs through them
2. Magic/dunder in Python are the special methods that start and end with the double underscores.
3. They are also called dunder methods. They are trigerred automatically
4. Objects can not call magic methods. Magic methods are pre-defined in Python, we can not create magic methods.
5. Magic methods are not meant to be invoked directly by you, but the invocation happens internally from the class on a certain action. For example, when you add two numbers using the + operator, internally, the **add()** method will be called.
6. Built-in classes in Python define many magic methods. **init** constructor is called automatically when object is created. Use the **dir()** function to see the number of magic

methods inherited by a class. For example, the following lists all the attributes and methods defined in the int class.

```
1 print(dir(int)) #For example, the following lists all the at:
```

```
→ ['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__', '__
```

◀ ─────────── ▶

1. when you do num+10, the + operator calls the **add(10)** method. You can also call num.**add(5)** directly which will give the same result. However, as mentioned before, magic methods are not meant to be called directly,
2. but internally, through some other methods or actions.

```
1 n=5+5
2 print(n)
```

```
→ 10
```

```
1 #the int class includes various magic methods surrounded by .
2 #For example, the __add__ method is a magic method which get
3 #Consider the following example.
4 #Example: Dunder Method Copy
5 num=10
6 res = num.__add__(5)
7 print(res)
```

```
→ 15
```

1. "A constructor is a special magic method in Python that is automatically executed when an object is created. Its control is not in the hands of the user. When the application starts, the code inside the constructor runs. Typically, we use the constructor for configuration-related tasks that should not rely on user input, such as establishing an internet connection. For example, if an application requires an internet connection and you ask the user to click a 'Connect to the Internet' button, the app may not work if the user does not click the button. Therefore, the app should not rely on user action for critical tasks."
2. "Code for connecting to a database or establishing GPS connectivity should be placed in the constructor if these are essential tasks that need to be executed as soon as the application starts." "We include configuration-related tasks or code in the application's constructor so that when the application opens, these tasks are executed automatically. This can include

connecting to a database, the internet, hardware, or performing other configuration-related tasks. This approach ensures the application runs smoothly without requiring user interaction for essential tasks."

Extra no need

4. constructor is one of the magic methods whose control is not with user and is executed automatically. As the application open up, the code inside the constructor is executed. Generally, we do configuration related tasks in constructor, which u dont rely on user, like connecting with internet, for instance, when application open, and u ask user to click on the button connect to the Internet, and he does not click the button then app will not work, so app should not rely on user.
5. code connecting to database, connecting to GPS should be placed in constructor
6. We put that configuration related task/code in the constructor of the applicaiton, so that when the application opens, the code facililate the application to run without asking from user,like connectivity with database, internet or hardware or any configuration related task.

## ▼ Notebook: 13) OOP\_All\_in\_One.ipynb

1 Start coding or generate with AI.



1. Everything in Python is an object, and every object belongs to a class. For instance, there may be many LCD objects, so a given LCD object might come from the LG class (company). Integers, lists, and tuples are objects, and these objects belong to specific classes.
2. From now on, variables and objects can be considered the same thing.
3. A class is a blueprint that defines a data type. In Python, defining a variable of a particular data type creates an object of that class.
4. There are many different data types in Python, each developed by someone. Developing a data type involves defining its class or blueprint, which describes how an object will behave or function.
5. The behavior of an object is determined by the data members and methods included in its class.
6. Creating a class requires more time, effort, and code than creating an object.
7. For example, when you create a list object, you have access to a lot of methods because these methods are already defined by the developer when creating the list class.

8. Class consist of 2 things data: Description about an object (nouns) Function: behavior of an object (verb) Suppose you are building a class car; Data: No. of wheels, color, engine type, brand Functions: calculate mileage, calculate average speed, open air bag, show gps navigation

```
1 lst=[1,2,3,4]
2 a=2
3 s='hello'
4 v= 3+5j
```

```
1 a=2      # a is the intgers object, which belongs to class 'int'
2 print(type(a))
```

```
→ <class 'int'>
```

```
1 #
2 l=[1,2,3,4] # list object of class 'list'
3 print(type(l))
```

```
→ <class 'list'>
```



Here are three commonly used naming conventions for identifiers:

1. Pascal Case (a.k.a. Upper Camel Case): In Pascal case, each word in the identifier starts with an uppercase letter, and there are no underscores between words. Example: FirstName, TotalAmount, CalculateArea
2. Camel Case (a.k.a. Lower Camel Case): In camel case, the first word starts with a lowercase letter, and each subsequent word starts with an uppercase letter. Example: firstName, totalAmount, calculateArea
3. Snake Case: In snake case, all words are lowercase, and words are separated by underscores (\_). Example: first\_name, total\_amount, calculate\_area

```
1 #Class Basic Structure
2 class Car: # class name should be in Pascal case
3     color='Blue' # data...data members should be in snake
4     model='Sports' # data
```

```
5 def calculate_avg_speed(km, miles): # method names should be  
6         pass
```

**Diagrammatic Representation of Class** private members of class are denoted by - sign while public members are denoted by + sign. Data members are usually private and are not accessible outside the class and methods are public and therefore accessible outside the class in the program



- Class is the datatype and object is the instance of class

## Object Examples

car----->Vitz Vitz= Car() object= classname() Sports---->Coupe Coupe= Sports() Animals---->Camel Camel=Animals()

- if list is a class, then why we create list object like this
- When you create a list using a literal expression like lst = [2, 3, 45, 6, 5], Python internally calls list() to create an object of the list class.
- Python provides a feature, object literal, which provides an easy way to create objects of built-in classes.
- In Python, you often create objects of built-in classes (such as integers, lists, and dictionaries) using object literal or literal expressions, such as 42 for integers, [] for lists, and {} for dictionaries.
- These object literals provide a straightforward and concise way to create instances of these built-in classes.
- you can create list object like this as well, lst= list(), syntax similar to the one discussed in above cell
- string object; s= str()

```
1 lst= list([1,2])  
2 lst
```

→ [1, 2]

```
1 lst1=[2,3,4,5]  
2 lst1
```

```
→ [2, 3, 4, 5]
```

```
1 a= str("hello")
2 s
```

```
→ 'hello'
```

```
1 print(type('hello'))
```

```
→ <class 'str'>
```

```
1 name= str('Atif')
2 print(name)
```

```
→ Atif
```

```
1 l=list([2,3,4])
2 print(l)
```

```
→ [2, 3, 4]
```

```
1 s=str("hello")
2 print(s)
```

```
→ hello
```

```
1 l= list([2,3,4,5])
2 print(l)
```

```
→ [2, 3, 4, 5]
```

We are developing Application/software of ATM using OOP. So we will need a representation/class of how customer behave when using ATM machine. For class, we would require data members, 1. account balance ( when we deposit money, balance will be updated), when we withdrawl, account balance will be deducted, similary, balance check is also related to account balance

2. ATM Pin, without this, we can do nothing on ATM Behaviors/Methods: create\_pin(), deposit(), withdrawal(), check\_bal()
3. Difference between Function vs Method
4. A method is a special function defined within a class that can be accessed and called by an object of that class.

5. A function, on the other hand, refers to a general function that exists outside of a class and can be used independently of class instances.
6. Functions can operate on different data types, such as lists, strings, and integers.
7. For example, `len(l)` is a general function that works with various iterable data types, while `l.append(2)` calls the `append` method, which is a special method defined within the list class and can be accessed only by list objects."

extra Method is a special function defined inside class and can be accessed by object of that class  
 Function refers to normal/general function which does not exist in any class, can be used any user, and can be applied on list, string and integers. for exmaple `len(l)` is a general function, while `l.append(2)`, `append` is special method defined inside list class and can be accesed only by list object.

```

1 #In Python, the len() function is used to determine the length of an iterable
2 #It is not used on integers directly because integers are not collections
3 #Therefore, calling len() on an integer will result in a TypeError
4 num = 1234
5 length = len(num) # This will raise a TypeError
6

```

→

---

**TypeError** Traceback (most recent call last)  
 Cell In[12], line 5  
     1 #In Python, the Len() function is used to determine the Length of an iterable  
     or a collection, such as a List, string, or dictionary.  
     2 #It is not used on integers directly because integers are not collections and  
     do not have a Length.  
     3 #Therefore, calling Len() on an integer will result in a TypeError:  
     4 num = 1234  
     ----> 5 length = len(num) # This will raise a TypeError  
  
**TypeError:** object of type 'int' has no len()

```

1 #If you want to determine the length of a string representation of an integer,  

2 #and then apply the len() function:
3 num = 1234
4 length = len(str(num)) # Convert the integer to a string and then apply len()
5 print(f"The length of {num} as a string is: {length}")
6

```

→ The length of 1234 as a string is: 4

1. variables in python are declared and initialized in constructor

2. **init** is a constructor, which is a special method defined in the class whose code is automatically executed when you create an object of class. Constructor name is same as class name in Java and C++, however, in python, it will be **init** we can prove this see below

```
1 class Atm:  
2     def __init__(self): # constructor  
3         pass  
4     def menu(self):  
5         pass  
6  
7 obj= Atm()
```

1. In Python, instance variables are variables that are defined within a class but outside of any method, typically in the class's constructor (**init** method). Each instance of the class (each object) will have its own copy of these variables.
2. In the context of your Atm class, variables such as pin and balance are defined in the constructor (**init** method). This means that when you create a new instance of the Atm class, each instance will have its own copy of pin and balance:

```
1 class Atm:  
2     def __init__(self):  
3         self.pin=""  
4         self.balance=0  
5         self.menu()  
6     def menu(self):  
7         user_input= int(input(""  
8                         Hello, How would you like to proceed?  
9                         1. Enter 1 to create pin  
10                        2. Enter 2 to deposit balance  
11                        3. Enter 3 to withdraw balance  
12                        4. Enter 4 to check balance  
13                        5. Enter 5 to exit  
14                         """))  
15         if user_input==1:  
16             self.create_pin()  
17             #print("pin created")  
18         elif user_input==2:  
19             self.deposit_balance()
```

```
20         #print("deposit balance")
21     elif user_input==3:
22         self.withdraw()
23             #print("withdraw balance")
24     elif user_input==4:
25         self.checkbalance()
26             print("cehk balance")
27 else:
28     print("bye")
29
30 def create_pin(self):
31     self.pin=input("Enter your pin")
32     print("pin created successfully")
33     self.menu()
34
35 def withdraw(self):
36     tmp=input("Please Enter your pin")
37     if (tmp==self.pin):
38         amt= int(input("Please Enter the amount u want to withdraw"))
39         if amt<self.balance:
40             self.balance=self.balance - amt
41             print("Withdrawl successfully")
42         else:
43             print("Invalid pin")
44         self.menu()
45
46 def deposit_balance(self):
47     tmp=input("Please Enter your pin")
48     if (tmp==self.pin):
49         amt= int(input("Please Enter the amount u want to deposit"))
50         self.balance=self.balance + amt
51         print("Amount deposited successfully")
52     else:
53         print("Invalid pin")
54     self.menu()
55 def checkbalance(self):
56     tmp=input("Please Enter your pin")
57     if (tmp==self.pin):
58         print(self.balance)
```

```
59
60     else:
61         print("Invalid pin")
62         self.menu()
```

```
1 obj=Atm()
2 #obj.deposit_balance()
3 #obj.withdraw()
4 #obj.checkbalance()
```



```
Hello, How would you like to proceed?
1. Enter 1 to create pin
2. Enter 2 to deposit balance
3. Enter 3 to withdraw balance
4. Enter 4 to check balance
5. Enter 5 to exit
```

```
1
```

```
Enter your pin 1234
pin created successfully
```

```
Hello, How would you like to proceed?
1. Enter 1 to create pin
2. Enter 2 to deposit balance
3. Enter 3 to withdraw balance
4. Enter 4 to check balance
5. Enter 5 to exit
```

```
2
```

```
Please Enter your pin 1234
```

```
Please Enter the amount u want to deposit 5000
Amount deposited successfully
```

```
Hello, How would you like to proceed?
1. Enter 1 to create pin
2. Enter 2 to deposit balance
3. Enter 3 to withdraw balance
4. Enter 4 to check balance
5. Enter 5 to exit
```

```
4
```

```
Please Enter your pin 1234
```

```
5000
```

```
Hello, How would you like to proceed?
1. Enter 1 to create pin
2. Enter 2 to deposit balance
3. Enter 3 to withdraw balance
4. Enter 4 to check balance
5. Enter 5 to exit
```

```
3
```

```
Please Enter your pin 1234
```

```
Please Enter the amount u want to withdraw 3000
```

```
Withdrawl successfully
```

```
Hello, How would you like to proceed?  
1. Enter 1 to create pin  
2. Enter 2 to deposit balance  
3. Enter 3 to withdraw balance  
4. Enter 4 to check balance  
5. Enter 5 to exit  
4  
Please Enter your pin 1234  
2000
```

```
Hello, How would you like to proceed?  
1. Enter 1 to create pin  
2. Enter 2 to deposit balance  
3. Enter 3 to withdraw balance  
4. Enter 4 to check balance
```

```
1 class Atm:  
2     def __init__(self): # constructor  
3         self.pin=""  
4         self.balance=0  
5         self.menu()  
6  
7     def menu(self):  
8         user_input= input("""  
9                         Hello, How would you like to proceed?  
10                        1. Enter 1 to create pin  
11                        2. Enter 2 to deposit balance  
12                        3. Enter 3 to withdraw balance  
13                        4. Enter 4 to check balance  
14                        5. Enter 5 to exit  
15                         """)  
16         if user_input=="1":  
17             self.create_pin()  
18         elif user_input=="2":  
19             self.deposit_balance()  
20         elif user_input=="3":  
21             self.withdraw_balance()  
22         elif user_input=="4":  
23             self.check_balance()  
24         else:  
25             print("good bye")  
26     def create_pin(self):  
27         self.pin= input("Enter your pin")
```

```
28         print("pin set successfully")
29
30     def deposit_balance(self):
31         tmp= input("Enter your pin")
32         if (tmp==self.pin):
33             amt=int(input("Enter the amount you want to depo")
34             self.balance=self.balance + amt
35             print("Amount deposited Successfully")
36         else:
37             print("Invalid Pin")
38
39     def withdraw_balance(self):
40         tmp= input("Enter your pin")
41         if (tmp==self.pin):
42             amt=int(input("Enter the amount you want to with")
43             if amt < self.balance:
44                 self.balance= self.balance - amt
45                 print("Amount withdrawl Successfully")
46             else:
47                 print("Insufficient funds")
48         else:
49             print("Invalid Pin")
50
51     def check_balance(self):
52         tmp= input("Enter your pin")
53         if (tmp==self.pin):
54             print(self.balance)
55         else:
56             print("Invalid Pin")
```

```
1 obj= Atm()
2 obj.deposit_balance()
3
```



```
Hello, How would you like to proceed?
1. Enter 1 to create pin
2. Enter 2 to deposit balance
3. Enter 3 to withdraw balance
4. Enter 4 to check balance
5. Enter 5 to exit
```

1

```
Enter your pin 1234
pin set successfully
Enter your pin 1234
Enter the amount you want to deposit 3457
Amount deposited Successfully
```

1 obj.check\_balance()

→ Enter your pin 1234  
3457

1 obj.withdraw\_balance()
2 obj.check\_balance()

→ Enter your pin 1234
Enter the amount you want to withdraw 3000
Amount withdrawl Successfully
Enter your pin 1234
457

1 obj.balance

→ 457

1 obj.balance=0

1 obj.balance

→ 0

1 obj2= Atm()
2 obj2.deposit\_balance()

→ Hello, How would you like to proceed?
1. Enter 1 to create pin
2. Enter 2 to deposit balance
3. Enter 3 to withdraw balance
4. Enter 4 to check balance
5. Enter 5 to exit
1
etnering
Enter your pin 3456
pin set successfully
Enter your pin 3456
Enter the amount you want to deposit 90000

```
Amount deposited Successfully
```

## 1 obj2.check\_balance()

```
→ Enter your pin 3456  
90000
```

1. now our code is well managed, all the backened logic is written in class, now create objects and peform differnt jobs through them
2. Magic/dunder in Python are the special methods that start and end with the double underscores.
3. They are also called dunder methods. They are trigerred automatically
4. Objects can not call magic methods. Magic methods are pre-defined in Python, we can not create magic methods.
5. Magic methods are not meant to be invoked directly by you, but the invocation happens internally from the class on a certain action. For example, when you add two numbers using the + operator, internally, the **add()** method will be called.
6. Built-in classes in Python define many magic methods. **init** constructor is called automatically when object is created. Use the **dir()** function to see the number of magic methods inherited by a class. For example, the following lists all the attributes and methods defined in the int class.

## 1 print(dir(int)) #For example, the following lists all the at:

```
→ ['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__', '__
```



1. when you do num+10, the + operator calls the **add(10)** method. You can also call num.**add(5)** directly which will give the same result. However, as mentioned before, magic methods are not meant to be called directly,
2. but internally, through some other methods or actions.

```
1 n=5+5  
2 print(n)
```

```
→ 10
```

```
1 #the int class includes various magic methods surrounded by  
2 #For example, the __add__ method is a magic method which get
```

```
3 #Consider the following example.  
4 #Example: Dunder Method Copy  
5 num1=10  
6 num2=10  
7 res = num1.__add__(num2)  
8 print(res)
```

→ 20

1. "A constructor is a special magic method in Python that is automatically executed when an object is created. Its control is not in the hands of the user. When the application starts, the code inside the constructor runs. Typically, we use the constructor for configuration-related tasks that should not rely on user input, such as establishing an internet connection. For example, if an application requires an internet connection and you ask the user to click a 'Connect to the Internet' button, the app may not work if the user does not click the button. Therefore, the app should not rely on user action for critical tasks."
2. "Code for connecting to a database or establishing GPS connectivity should be placed in the constructor if these are essential tasks that need to be executed as soon as the application starts." "We include configuration-related tasks or code in the application's constructor so that when the application opens, these tasks are executed automatically. This can include connecting to a database, the internet, hardware, or performing other configuration-related tasks. This approach ensures the application runs smoothly without requiring user interaction for essential tasks."

Extra no need

4. constructor is one of the magic methods whose control is not with user and is executed automatically. As the application open up, the code inside the constructor is executed. Generally, we do configuration related tasks in constructor, which u dont rely on user, like connecting with internet, for instance, when application open, and u ask user to click on the button connect to the Internet, and he does not click the button then app will not work, so app should not rely on user.
5. code connecting to database, connecting to GPS should be placed in constructor
6. We put that configuration related task/code in the constructor of the applicaiton, so that when the application opens, the code facililate the application to run without asking from user,like connectivity with database, internet or hardware or any configuration related task.

```
1 #self is the object with which u are currently working with  
2 #or self is a reference to the current instance(object) of t  
3 # define a class
```

```
4 class Student:  
5     def __init__(self):  
6         self.name=""  
7         self.age= 0  
8         print(f"The address of the current object is {id(sel  
9  
10 # create object of class  
11 std1 = Student()  
12 #std2=Student()  
13 ## Print the memory address of the std1 object  
14 print(f"Address of std1 object is {id(std1)}")  
15 # std1 itself is self  
16 std1.age = 21  
17 std1.name = "Hassan"  
18 print(f"Name: {std1.name}, id: {std1.age} ")  
19  
20  
21
```

→ The address of the current object is 3005734103968  
Address of std1 object is 3005734103968  
Name: Hassan, id: 21

```
1 std2=Student()  
2 ## Print the memory address of the std1 object  
3 print(f"Address of std2 object is {id(std2)}")  
4
```

→ The address of the current object is 3005734097056  
Address of std2 object is 3005734097056

Why self is required in each method of clas fundamental concept in OOP: Class contains data and methods and both of them can be accessed by object of that class. Concept2: A method of a class not access data or method of the same class, but many situations will encounter where one method will require access to another method, just like init requires access to menu method, but how, because we need object of class, and object will come from self and self refers to current object that just called the method

or Class Structure: In OOP, a class is a blueprint that defines data attributes (fields) and methods (functions) that operate on that data. Objects are instances of a class, and each object contains its own set of the class's data and methods. Self Reference: In Python, the first parameter of each

method within a class is typically self, which is a reference to the instance (object) that is calling the method. By convention, it is always called self to make the code clear and readable.

Accessing Data and Methods: The self reference allows methods within a class to access both the data attributes and other methods of the class. This is essential for object-oriented design, as it allows for encapsulation and modularization. Method Interactions: As you mentioned, a method within a class may need to interact with another method of the same class. Since self refers to the current instance (object), methods can use self to call other methods of the same class. For example, in your statement, `init` might need to call menu to initialize some part of the object; it can do so using `self.menu()`.

Fractions can not be handled in python nor in any language, if u write  $4/5$ , python will return 0.8  
Now we want to create a fraction datatype. then will create fraction object in python. data related to fraction object: numerator, denominator

```
1 4/5
```

```
→ 0.8
```

```
1 class Fraction:  
2     def __init__(self,n, d):  
3         self.nun= n  
4         self.den=d  
5
```

```
1 fobj1= Fraction(4,5)  
2 print(fobj1) # object of type fraction, object is created,  
3 print(type(fobj1))  
4
```

```
→ <__main__.Fraction object at 0x000002BBD3B69130>  
<class '__main__.Fraction'>
```

```
1 #fobjec1 is just integers and string objects, you can perform  
2 # with integer  
3 l=[1,2,3,fobj1] # fobj1 is not seen because we did not speci·  
4 l                      # and object specifications will be provided
```

```
→ [1, 2, 3, <__main__.Fraction at 0x2bbd3b69130>]
```

```
1 fobj2= Fraction(4,5)
2 print(fobj1) # object of type fraction, object is created,
3 # i want to show fraction object like 4/5, here magic method
4 # which are not called by objects and are called automatical
5 # is the __str__(), which is called automatically, when u pr
6
```

→ <\_\_main\_\_.Fraction object at 0x000002BBD3B69130>

```
1 class Fraction:
2     def __init__(self,n, d):
3         self.nun= n
4         self.den=d
5
6     def __str__(self):
7         return "hello"
```

```
1 fobj3= Fraction(4,5)
2 fobj4=Fraction(4,3)
3
4 print(fobj3)
5 print(fobj4)
```

→ hello  
hello

```
1 # but now i pass string formatting. we can now show how our
2 class Fraction:
3     def __init__(self,n, d):
4         self.num= n
5         self.den=d
6     def __str__(self):
7         #return "{}/{}".format(self.num,self.den)
8         return f'{self.num}/{self.den}'
```

```
1 fobj3= Fraction(4,5)
2 fobj4=Fraction(2,3)
3
4
```

```
5 print(fobj3)
6 print(fobj4)
```

→ 4/5  
2/3

```
1 #If i try to add two fraction objects, fobj3 + fobj4, will it be done? No
2 #because you did not specify how to add two fraction objects
3 print(fobj3+fobj4)
```

→ -----  
**TypeError** Traceback (most recent call last)  
Cell In[14], line 3  
 1 #If i try to add two fraction objects, fobj3 + fobj4, will it be done? No  
 2 #because you did not specify how to add two fraction objects  
----> 3 print(fobj3+fobj4)  
  
**TypeError**: unsupported operand type(s) for +: 'Fraction' and 'Fraction'

```
1 s1={3,4,5}
2 s2={4,5,6}
3 s1+s2
4 # whoever created the class, did not specify how to add two
```

→ -----  
**TypeError** Traceback (most recent call last)  
Cell In[15], line 3  
 1 s1={3,4,5}
 2 s2={4,5,6}
----> 3 s1+s2
 4 # whoever created the class, did not specify how to add two set objects  
  
**TypeError**: unsupported operand type(s) for +: 'set' and 'set'

```
1 # what if i want to add two fraction objects, here third magic
2 # triggered/called automatically by python, when u try to add
3 # and the code inside the __add__() is executed.
```



1. In fraction addition, numerator will be sum of cross multiplication while denominator will be multiplication of two denominators

```
1 class Fraction:
2     def __init__(self, n, d):
3         self.num = n
4         self.den = d
5
6     def __add__(self, other):
7
8         temp_num = self.num * other.den + other.num * self.de
9         temp_den = self.den * other.den
10        return "{} / {}".format(temp_num, temp_den)
11
12    def __str__(self):
13        return "{} / {}".format(self.num, self.den)
```

```
1 x = Fraction(4, 5)
2 y = Fraction(4, 3)
3 print(x)
4 print(y)
5 print(x + y)
6 #print(fobj3)
```

→ 4/5  
4/3  
32/15

```
1 # still the above code will not handle a-b, becos u did not
2 # 3rd magic method __sub__ comes, which is automatically tri
3
4 class Fraction:
5     def __init__(self, n, d):
6         self.num = n
7         self.den = d
8
9     def __add__(self, other):
10        temp_num = self.num * other.den + other.num * self.de
11        temp_den = self.den * other.den
12        return "{} / {}".format(temp_num, temp_den)
13
14    def __sub__(self, other):
```

```

15     temp_num= self.num * other.den - other.num * self.de
16     temp_den= self.den * other.den
17     return "{}{}".format(temp_num,temp_den)
18
19     def __str__(self):
20         return "{}{}".format(self.num,self.den)

```

```

1 x= Fraction(4,5)
2 y=Fraction(4,3)
3 print(x)
4 print(y)
5 #print(x + y)
6 print(x-y)

```

→ 4/5  
4/3  
-8/15

```

1 #3rd magic method __mul__ comes, which is automatically trig
2 class Fraction:
3     def __init__(self,n, d):
4         self.num= n
5         self.den=d
6
7     def __add__(self, other):
8         temp_num= self.num * other.den + other.num * self.de
9         temp_den= self.den * other.den
10        return "{}{}".format(temp_num,temp_den)
11
12    def __sub__(self, other):
13        temp_num= self.num * other.den - other.num * self.de
14        temp_den= self.den * other.den
15        return "{}{}".format(temp_num,temp_den)
16
17    def __mul__(self, other):
18        temp_num= self.num * other.num
19        temp_den= self.den * other.den
20        return "{}{}".format(temp_num,temp_den)
21

```

```
22     def __str__(self):  
23         return "{} / {}".format(self.num, self.den)
```

```
1 x= Fraction(4,5)  
2 y=Fraction(4,3)  
3 print(x)  
4 print(y)  
5 #print(x + y)  
6 print(x*y)
```

→ 4/5  
4/3  
16/15

```
1 # 3rd magic method __truediv__ comes, which is automatically  
2 class Fraction:  
3     def __init__(self, n, d):  
4         self.num = n  
5         self.den = d  
6  
7     def __add__(self, other):  
8         temp_num = self.num * other.den + other.num * self.de  
9         temp_den = self.den * other.den  
10        return "{} / {}".format(temp_num, temp_den)  
11  
12    def __sub__(self, other):  
13        temp_num = self.num * other.den - other.num * self.de  
14        temp_den = self.den * other.den  
15        return "{} / {}".format(temp_num, temp_den)  
16  
17    def __mul__(self, other):  
18        temp_num = self.num * other.den  
19        temp_den = self.den * other.den  
20        return "{} / {}".format(temp_num, temp_den)  
21  
22    def __truediv__(self, other):  
23        temp_num = self.num * other.den  
24        temp_den = self.den * other.num  
25        return "{} / {}".format(temp_num, temp_den)
```

```
26
27     def __str__(self):
28         return "{} / {}".format(self.num, self.den)
```

```
1 x = Fraction(4, 5)
2 y = Fraction(4, 3)
3 print(x)
4 print(y)
5 print(x + y)
6 print(x - y)
7 print(x * y)
8 print(x/y)
```

```
→ 4/5
  4/3
  32/15
  -8/15
  12/15
  12/20
```

## ▼ Need for encapsulation

### Private attributes

### Getter and Setter methods

### Class Diagrams

Any variable that is created in constructor is called instance variable because instance variable possesses a different value for each object for instance each customer should have a different pin and balance value.

or instance variable is a kind of variable which has a different value for different objects

when you create a class, it's a good practice to hide the data members of the class by putting double underscore in front of variable, if you want, you can also hide methods from other users by putting double underscore. The private data/method members will be accessible within class but not accessible outside class

```
1 class Atm:
2     def __init__(self): # constructor
3         self.pin=""
4         self.balance=0
5         self.menu()
6
7     def menu(self):
8         user_input= input("""
9                         Hello, How would you like to proceed?
10                        1. Enter 1 to create pin
11                        2. Enter 2 to deposit balance
12                        3. Enter 3 to withdraw balance
13                        4. Enter 4 to check balance
14                        5. Enter 5 to exit
15                    """)
16     if user_input=="1":
17         self.create_pin()
18     elif user_input=="2":
19         self.deposit_balance()
20     elif user_input=="3":
21         self.withdraw_balance()
22     elif user_input=="4":
23         self.check_balance()
24     else:
25         print("good bye")
26     def create_pin(self):
27         self.pin= input("Enter your pin")
28         print("pin set successfully")
29
30     def deposit_balance(self):
31         tmp= input("Enter your pin")
32         if (tmp==self.pin):
33             amt=int(input("Enter the amount you want to depo
34             self.balance=self.balance + amt
35             print("Amount deposited Successfully")
36         else:
37             print("Invalid Pin")
38
```

```
39     def withdraw_balance(self):
40         tmp= input("Enter your pin")
41         if (tmp==self.pin):
42             amt=int(input("Enter the amount you want to withdraw"))
43             if amt < self.balance:
44                 self.balance= self.balance - amt
45                 print("Amount withdrawl Successfully")
46             else:
47                 print("Insufficient funds")
48         else:
49             print("Invalid Pin")
50
51     def check_balance(self):
52         tmp= input("Enter your pin")
53         if (tmp==self.pin):
54             print(self.balance)
55         else:
56             print("Invalid Pin")
```

```
1 hbl= Atm()
2 hbl.balance='xyz'
3 hbl.deposit_balance()
```



```
Hello, How would you like to proceed?  
1. Enter 1 to create pin  
2. Enter 2 to deposit balance  
3. Enter 3 to withdraw balance  
4. Enter 4 to check balance  
5. Enter 5 to exit  
1  
Enter your pin 1234  
pin set successfully  
Enter your pin 1234  
Enter the amount you want to deposit 444  
-----  
TypeError Traceback (most recent call last)  
Cell In[27], line 3  
  1 hbl= Atm()  
  2 hbl.balance='xyz'  
--> 3 hbl.deposit_balance()  
  
Cell In[26], line 34, in Atm.deposit_balance(self)  
 32 if (tmp==self.pin):  
 33     amt=int(input("Enter the amount you want to deposit"))  
--> 34     self.balance=self.balance + amt  
 35     print("Amount deposited Successfully")  
 36 else:  
  
TypeError: can only concatenate str (not "int") to str
```

in the above code, issue is that object hbl can access all

the members data members or methods since #they are

- ✓ public, junior pogrammar or any other may change the data members, change the pin, or assign string to balance variable

and lead the code to crash like the one above, however, its responsibility of senior programmer to avoid changing the data members, solution is to rotect/hide the data members to avoid access from outside the class.

in order to hide the data members from outside class, put double underscore in front of data members

```
1 class Atm:
2     def __init__(self): # constructor
3         self.__pin=""
4         self.__balance=0
5         self.__menu()
6
7     def __menu(self):
8         user_input= input("""
9                         Hello, How would you like to proceed?
10                        1. Enter 1 to create pin
11                        2. Enter 2 to deposit balance
12                        3. Enter 3 to withdraw balance
13                        4. Enter 4 to check balance
14                        5. Enter 5 to exit
15                    """)
16         if user_input=="1":
17             self.create_pin()
18         elif user_input=="2":
19             self.deposit_balance()
20         elif user_input=="3":
21             self.withdraw_balance()
22         elif user_input=="4":
23             self.check_balance()
24         else:
25             print("good bye")
26     def create_pin(self):
27         self.__pin= input("Enter your pin")
28         print("pin set successfully")
29
30     def deposit_balance(self):
31         tmp= input("Enter your pin")
32         if (tmp==self.__pin):
33             amt=int(input("Enter the amount you want to depo
34             self.__balance=self.__balance + amt
35             print("Amount deposited Successfully")
```

```

36     else:
37         print("Invalid Pin")
38
39     def withdraw_balance(self):
40         tmp= input("Enter your pin")
41         if (tmp==self.__pin):
42             amt=int(input("Enter the amount you want to withdraw"))
43             if amt < self.__balance:
44                 self.__balance= self.__balance - amt
45                 print("Amount withdrawl Successfully")
46             else:
47                 print("Insufficient funds")
48         else:
49             print("Invalid Pin")
50
51     def check_balance(self):
52         tmp= input("Enter your pin")
53         if (tmp==self.__pin):
54             print(self.__balance)
55         else:
56             print("Invalid Pin")

```

Suppose you have Atm class, When u put double underscore before variable, **pin, python interpreter internally convert it to \_Atmpin** at the time of execution, and hence replaces each instance of **pin** to **\_Atmpin** in the whole code. similarly, **balance-----> \_Atmbalance**, throughout the program, All the methods will now be using **Atmpin, \_Atm\_\_balance** data members internally  
**hbl2.balance='3333' # here the python interpreter creates a new variable, but this variable will not effect the existing logic and this variable has no use, since all the methods are using \_Atmbalance** and hence still our class data members are protected,

```

1 hbl2= Atm()
2 print(hbl2._Atm__balance)

```



Hello, How would you like to proceed?

1. Enter 1 to create pin
2. Enter 2 to deposit balance
3. Enter 3 to withdraw balance
4. Enter 4 to check balance
5. Enter 5 to exit

```
good bye  
0
```

```
1  
2 hbl2= Atm()  
3 hbl2.__balance='3333' #  
4 hbl2.deposit_balance()  
5 hbl2.check_balance()
```



Hello, How would you like to proceed?

1. Enter 1 to create pin
2. Enter 2 to deposit balance
3. Enter 3 to withdraw balance
4. Enter 4 to check balance
5. Enter 5 to exit

1

Enter your pin 1234

pin set successfully

Enter your pin 1234

Enter the amount you want to deposit 500

Amount deposited Successfully

Enter your pin 1234

500

hbl2= Atm() hbl2.\_Atm\_\_balance='axx' # can access the hidden data member of class if required to access the private hbl2.deposit\_balance() hbl2.check\_balance()

1. Nothing in python is truly private, python leaves the choice/decision upto developer to access the private data members outside the class or not.

2. The reasons that Python allows access to private attributes outside of the class:

3. Flexibility: Python is designed to give developers freedom and trust them to make decisions about how to write their code. This includes the ability to access or modify private attributes if necessary. Python does not strictly enforce access control (as other languages like Java or C++ do) because it trusts developers to understand the potential consequences of accessing private attributes directly. This means that Python places the responsibility on the developer to use best practices and avoid directly accessing private attributes unless they have a good reason to do so.

4. Accessibility: Allowing access to private attributes can sometimes be helpful, particularly in complex or advanced situations such as debugging or working around limitations in libraries or frameworks. For instance, a developer might need to access private data or methods in a library to fix a bug or extend functionality that is not exposed through the public interface.

5. Name Mangling: Python uses a mechanism called name mangling for private attributes and methods that begin with double underscores (`__`). This mechanism prefixes the attribute or method name with the class name, effectively "mangling" the name and making it harder to access from outside the class. However, it does not prevent access entirely; it serves as a warning to developers that these are intended to be private and should not be accessed directly.
6. Code Readability: By not strictly enforcing privacy through access modifiers (public, private, protected), Python keeps its syntax simpler and the code more readable. This decision is in line with Python's philosophy of emphasizing simplicity and readability.
7. Encapsulation: Although Python encourages the use of encapsulation by providing private attributes and methods, it leaves the choice to developers to use them appropriately. Developers are expected to respect the encapsulation principle by using public interfaces and avoiding direct access to private attributes. This helps maintain clean and maintainable code.

## ▼ Access Modifiers in Python : Public, Private and Protected

1. Various object-oriented languages like C++, Java, Python control access modifications which are used to restrict access to the variables and methods of the class.
2. Most programming languages have three forms of access modifiers, which are Public, Protected and Private in a class.
3. Python uses '`_`' symbol to determine the access control for a specific data member or a member function of a class.
4. Access specifiers in Python have an important role to play in securing data from unauthorized access and in preventing it from being exploited.
5. A Class in Python has three types of access modifiers:

Public Access Modifier      Protected Access Modifier      Private Access Modifier

### **Public Access Modifier:**

1. The members of a class that are declared public are easily accessible from any part of the program. All data members and member functions of a class are public by default.

### **Protected Access Modifier:**

2. The members of a class that are declared protected are only accessible to a class derived from it. Data members of a class are declared protected by adding a single underscore '`_`' symbol before the data member of that class.

### **Private Access Modifier:**

3. The members of a class that are declared private are accessible within the class only, private access modifier is the most secure access modifier. Data members of a class are declared

private by adding a double underscore '\_\_' symbol before the data member of that class

```
1 # program to illustrate public access modifier in a class
2
3 class Student:
4
5     # constructor
6     def __init__(self, name, age):
7
8         # public data members
9         self.Name = name
10        self.Age = age
11
12    # public member function
13    def displayAge(self):
14
15        # accessing public data member
16        print("Age: ", self.Age)
17
18 # creating object of the class
19 obj = Student("Hassan", 20)
20
21 # accessing public data member
22 print("Name: ", obj.Name)
23
24 # calling public member function of the class
25 obj.displayAge()
26
27 #In the above program, Name and Age are public data members
28 #function of the class Student.
29 #These data members of the class Student can be accessed from
30
```

→ Name: Hassan  
Age: 20

```
1 # super class
2 class Student:
3
```

```

4     # protected data members
5     _roll = None
6     _branch = None
7
8     # constructor
9     def __init__(self, roll, branch):
10        self._roll = roll
11        self._branch = branch
12
13    # protected member function
14    def _displayRollAndBranch(self):
15        # accessing protected data members
16        print("Roll: ", self._roll)
17        print("Branch: ", self._branch)
18
19 # derived class
20 class Ugradstudent(Student):
21
22    # constructor
23    def __init__(self, name, roll, branch):
24        Student.__init__(self, roll, branch)
25        self._name = name
26
27    # public member function
28    def displayDetails(self):
29        # accessing protected data members of super class
30        print("Name: ", self._name)
31
32        # accessing protected member functions of super clas
33        self._displayRollAndBranch()
34
35 # creating objects of the derived class
36 obj = Ugradstudent("Majid", 12346, "Information Technology")
37
38 # calling public member functions of the class
39 obj.displayDetails()
40

```

→ Name: Majid  
 Roll: 12346

Branch: Information Technology

```
1 # program to illustrate private access modifier in a class
2
3 class Student:
4
5     # private members
6     __name = None
7     __roll = None
8     __branch = None
9
10    # constructor
11    def __init__(self, name, roll, branch):
12        self.__name = name
13        self.__roll = roll
14        self.__branch = branch
15
16    # private member function
17    def __displayDetails(self):
18
19        # accessing private data members
20        print("Name: ", self.__name)
21        print("Roll: ", self.__roll)
22        print("Branch: ", self.__branch)
23
24    # public member function
25    def accessPrivateFunction(self):
26
27        # accessing private member function
28        self.__displayDetails()
29
30 # creating object
31 obj = Student("Farhan", 1706256, "Computer Science")
32
33 # calling public member function of the class
34 obj.accessPrivateFunction()
35
```

→ Name: Farhan  
Roll: 1706256

1. encapsulation in Python is achieved by keeping attributes and methods private, using name mangling to hide them from external access, and providing controlled access to data through public methods such as getters and setters. This helps protect the data and ensures it is used and modified according to the rules established by the class.

**idea of Encapsulation:**

1. Do not allow your data members to be public or directly accessible outside the class because anyone can change the value of data members such `hbl.balance='xyz'`, which is not desirable.

2. alternate solution is to first hide the data members with double underscore. then, if anyone needs the private data members, then access can be allowed through 2 methods, `get()` and `set()` methods. `get()` method will fetch the value while set method will set/update the values of private data members according to certain rules/logic established by programmar. means one can not use set method to change values of data members at his own choice. so we are protecting the data. so for every data member, we are creating two methods, `get()` and `set()`, and using these three, we will use the data members and keeping them protected. this is the whole idea behind encapsulation.

or "Idea of Encapsulation:

3. Encapsulation is a core principle in object-oriented programming that involves bundling data and methods that operate on the data within a class, while restricting direct access to the data from outside the class.

4. Private Data Members: It is important to keep data members private (using double underscores in Python) to prevent direct access from outside the class. This helps maintain control over how data is accessed and modified, protecting the integrity of the data. Getter and Setter Methods: To access and modify private data members, use getter (`get()`) and setter (`set()`) methods. Getter methods fetch the value of the private data members, while setter methods set or update their values according to the rules and logic established by the programmer. This way, data cannot be changed arbitrarily from outside the class. For instance, you might want to validate data before setting it, or format it in a specific way when retrieving it.

5. Protection and Flexibility: By using getter and setter methods, you can protect data integrity and enforce encapsulation. This approach also provides flexibility for future changes, as the internal implementation of the data can be modified without affecting code that uses the getter and setter methods. Encapsulation helps maintain the reliability and security of the data within a class, ensuring that data is used and modified in a controlled and predictable manner."

```
1 class Atm:
2     def __init__(self): # constructor
3         self.__pin=0
4         self.__balance=0
5         self.__menu()
6
7     def get_pin(self):
8         return self.__pin
9
10    def set_pin(self, new_pin):
11        if type(new_pin)==int:
12            self.__pin=new_pin
13            print("pin changed sucessfully")
14        else:
15            print("Invalid pin")
16
17    def __menu(self):
18        user_input= input("""
19                                Hello, How would you like to proceed?
20                                1. Enter 1 to create pin
21                                2. Enter 2 to deposit balance
22                                3. Enter 3 to withdraw balance
23                                4. Enter 4 to check balance
24                                5. Enter 5 to exit
25                                """)
26        if user_input=="1":
27            self.create_pin()
28        elif user_input=="2":
29            self.deposit_balance()
30        elif user_input=="3":
31            self.withdraw_balance()
32        elif user_input=="4":
33            self.check_balance()
34        else:
35            print("good bye")
36    def create_pin(self):
37        self.__pin= int(input("Enter your pin"))
38        print("pin set successfully")
```

```

39
40     def deposit_balance(self):
41         tmp= int(input("Enter your pin"))
42         if (tmp==self.__pin):
43             amt=int(input("Enter the amount you want to depo
44             self.__balance=self.__balance + amt
45             print("Amount deposited Successfully")
46         else:
47             print("Invalid Pin")
48
49     def withdraw_balance(self):
50         tmp= int(input("Enter your pin"))
51         if (tmp==self.__pin):
52             amt=int(input("Enter the amount you want to with
53             if amt < self.__balance:
54                 self.__balance= self.__balance - amt
55                 print("Amount withdrawl Successfully")
56             else:
57                 print("Insufficient funds")
58         else:
59             print("Invalid Pin")
60
61     def check_balance(self):
62         tmp= int(input("Enter your pin"))
63         if (tmp==self.__pin):
64             print(self.__balance)
65         else:
66             print("Invalid Pin")

```

```

1 obj=Atm() # obj is not object, it is a reference variable an
2 obj.get_pin()
3 obj.set_pin(5.6) # if you pass 5.6, self.__pin will be set/c
4 obj.set_pin("xyzz")
5 obj.set_pin("3333")
6 obj.get_pin()
7 #initially, we hide the data members and then we created two
8 # value and change the value, the above code change the valu
9 #use to be public, and setting any value for pin may crash t

```

```
10 #since we are updating the value of data member through func  
11 #data should be modified and we can validate the data before
```



```
Hello, How would you like to proceed?
```

1. Enter 1 to create pin
2. Enter 2 to deposit balance
3. Enter 3 to withdraw balance
4. Enter 4 to check balance
5. Enter 5 to exit

```
1
```

```
Enter your pin 1234
```

```
pin set successfully
```

```
Invalid pin
```

```
Invalid pin
```

```
Invalid pin
```

```
1234
```

```
1 obj=Atm()  
2 obj.get_pin()  
3 #obj.set_pin(5.6) # if you pass 5.6, self.__pin will be set/  
4 obj.set_pin("1223")  
5 obj.get_pin()
```



```
Hello, How would you like to proceed?
```

1. Enter 1 to create pin
2. Enter 2 to deposit balance
3. Enter 3 to withdraw balance
4. Enter 4 to check balance
5. Enter 5 to exit

```
1
```

```
Enter your pin 1234
```

```
pin set successfully
```

```
pin changed
```

```
'1223'
```

```
1 class Customer:  
2     def __init__(self, name):  
3         self.name=name  
4
```

```
1 cust= Customer('Mehran')  
2 print(cust.name)
```

→ Mehran

```
1 class Customer:  
2     def __init__(self, name, gender):  
3         self.name=name  
4         self.gender=gender  
5  
6 def greet(customer): # now customer and cust are referecing .  
7     if customer.gender=="Male":  
8         print("hello,",customer.name, " Sir")  
9     else:  
10        print("hello,",customer.name, " Ma'm")
```

```
1 cust= Customer("Essa","Male")  
2 greet(cust) # passing object(by reference) as argument to fu  
3  
4 cust= Customer("Mrs Essa","Female")  
5 greet(cust)
```

→ hello, Essa Sir  
hello, Mrs Essa Ma'm

```
1 class Customer:  
2     def __init__(self, name, gender):  
3         self.name=name  
4         self.gender=gender  
5  
6 def greet(customer): # now customer and cust are referecing .  
7     if customer.gender=="Male":  
8         print("hello,",customer.name, " Sir")  
9     else:  
10        print("hello,",customer.name, " Ma'm")  
11    new_cust=Customer("Adil",25) # function is returning ob  
12    return new_cust
```

```
1 cust= Customer("Fazal","Male")  
2 greet(cust) # passing object(by reference) as argument to fu  
3 cust= Customer("fareda","Female")  
4 cust=greet(cust)  
5 print(cust.name)
```

→ hello, Fazal Sir  
hello, fareda Ma'm  
Adil

```
1 class Customer:  
2     def __init__(self, name):  
3         self.name=name  
4 def greet(customer): # now customer and cust are referecing  
5     print(id(customer))           #both cust and customer are  
6
```

```
1 cust= Customer("Atif")  
2 print(id(cust))  
3 greet(cust) # passing object(by recust= Customer("fareda","F  
4  
5 #object behave in a same manner just like other datatypes i
```

→ 1373838975168  
1373838975168

1. Mutability Evidence: Since the memory address printed at the beginning and end of the function is the same,
2. it indicates that the object was edited in place without changing its memory location.
3. This proves that the customer object is mutable because its attributes can be modified without creating a new object.
4. Objects of class are mutable just like lists, dictionary and sets

```
1 class Customer:  
2     def __init__(self, name):  
3         self.name=name  
4  
5 def greet(customer): # now customer and cust are referecing  
6     print(id(customer))           #both cust and customer are  
7     customer.name='Hammad' # this change in the object will  
8     print(customer.name)  
9     print(id(customer)) # if this id of object does not chan  
10    #you are editing object on the same memory address, and  
11
```

```
1 cust= Customer("Atif")
2 print(id(cust))
3 greet(cust) # passing object by ref
4 print(cust.name)
```

```
→ 2649799457200
  2649799457200
  Hammad
  2649799457200
  Hammad
```

1. You are passing the original list l1 directly to the function change\_list.
2. This means that change\_list receives the actual list object (l1) and any modifications made to the list inside the function will also be reflected in the original list l1.
3. This behavior demonstrates that the list is mutable because the function can modify the original list directly.

```
1 # prove that list is mutable datatypes mean changes in the list
2 def change_list(l1):
3     print(id(l1))
4     l1.append(5)
5     print(id(l1))
6
```

1. You are passing a copy of the list l1 to the function change\_list.
2. This copy is created using the slice notation (l1[:]), which creates a shallow copy of the list.
3. When you pass l1[:] to change\_list, the function receives a new list object that contains the same elements as l1, but is independent of the original list.
4. Any modifications made to the list inside the function change\_list will affect only the copy, not the original list l1.

```
1 def change_list(l1):
2     print("change list object before edit", id(l1))
3     l1.append(5)
4     print("change list object after edit", id(l1))
5
6 l1= [2,13,4,3]
7 print(id(l1))
8 print(l1)
```

```
9 change_list(l1[:]) # here we sent a cloned list to avoid cha
10 print(id(l1))
```

→ 2649821151680  
[2, 13, 4, 3]  
change list object before edit 2649821201280  
change list object after edit 2649821201280  
2649821151680

1 if you pass mutable object by reference, any change made to ·  
2 reflected in the original object, while if you pass immutab  
3 be reflected in the original object.

```
1 def change_tuple(t):
2     print("befor edit", id(t))
3     t=t + (6,7)
4     print("After edit",id(t))
5
6 t1= (2,13,4,3)
7 print("before change tuple",id(t1))
8 print(t1)
9 change_tuple(t1)
10 print("After change tuple",id(t1))
11 print(t1)
```

→ before change tuple 2649821378848  
(2, 13, 4, 3)  
befor edit 2649821378848  
After edit 2649820536384  
After change tuple 2649821378848  
(2, 13, 4, 3)

```
1 #Merges loops with objects
2 # sets can have immutable data types only so objects can not
3 class Customer:
4     def __init__(self, name, age):
5         self.name=name
6         self.age=age
7
8     def intro(self):
9         print("My name is ",self.name," I am ",self.age)
10
```

```

11 cust1= Customer("hassan", 20)
12 cust2= Customer("Ali", 20)
13 cust3=cust= Customer("Hanif", 21)
14
15 lst= [cust1, cust2, cust3] # collection of objects
16 for i in lst: # can iterate via list of objects
17     #print(i)
18     #print(i.name, i.age)
19     i.intro()

```

→ My name is hassan I am 20  
 My name is Ali I am 20  
 My name is Hanif I am 21

```

1 #Merges loops with objects
2
3 class Customer:
4     major='CS'
5     def __init__(self, name, age,add):
6         self.name=name
7         self.age=age
8         self.address=add
9     def intro(self):
10         print("Name: ",self.name," Age: ",self.age, ' Address: ',self.address)
11
12 cust1= Customer("hassan", 20,'Pesh')
13 cust2= Customer("Ali", 20,'Karak')
14 cust3=cust= Customer("Hanif", 21,'swabi')
15
16 lst= [cust1, cust2, cust3] # collection of objects
17 for i in lst: # can iterate via list of objects
18     #print(i)
19     #print(i.name, i.age)
20     i.intro()

```

→ Name: hassan Age: 20 Address: Pesh Major: CS  
 Name: Ali Age: 20 Address: Karak Major: CS  
 Name: Hanif Age: 21 Address: swabi Major: CS

```

1 #static variables/class variable4- defined in class outside
2 # instance variables--inside constructor

```

```
3 class Product:
4     total_products = 0 # Static variable to track total products
5
6     def __init__(self, id, name, price, quantity):
7         self.id = id
8         self.name = name
9         self.price = price
10        self.quantity = quantity
11        Product.total_products += 1 # Increment total_products
12
13
14    def display(self):
15        print(f"Product ID: {self.id}")
16        print(f"Name: {self.name}")
17        print(f"Price: ${self.price}")
18        print(f"Quantity: {self.quantity}")
19        print()
20
21    @staticmethod
22    def display_total_products():
23        print(f"Total Products: {Product.total_products}")
24
25
26 # Now, let's demonstrate the usage of the static variable:
27
28 # Creating some products
29 product1 = Product(1, "Apple", 1.5, 100)
30 product2 = Product(2, "Banana", 0.75, 150)
31 product3 = Product(3, "Orange", 2.0, 80)
32
33 # Displaying total number of products
34 Product.display(product1)
35 Product.display_total_products()
36
37 # Adding another product
38 product4 = Product(4, "Grapes", 3.0, 120)
39
40 # Displaying total number of products again
```

```
41 #Product.display_total_products()
```

```
42
```

```
→ Product ID: 1  
Name: Apple  
Price: $1.5  
Quantity: 100
```

```
Total Products: 3
```

1. Instance variables and static variables are both types of variables in object-oriented programming, but they serve different purposes and have different scopes.

#### Instance Variable:

1. An instance variable is a variable that is defined inside a class and belongs to each instance (object) of that class individually.
2. Each object has its own copy of instance variables.
3. They are initialized inside the constructor (`init` method) using the `self` keyword.
4. Instance variables hold data that are unique to each object.
5. They are accessed using the object (instance) of the class.
6. Changes to instance variables affect only the specific instance they belong to. Example: In a Person class, name and age might be instance variables because each person has a different name and age.

#### Static Variable:

1. A static variable (also known as a class variable) is a variable that is associated with the class as a whole rather than with specific instances of the class.
2. There is only one copy of the static variable that is shared among all instances of the class.
3. Static variables are defined at the class level, outside of any method, usually at the beginning of the class definition.
4. They are accessed using the class name.
5. Static variables are used to store data that is common to all objects of the class.
6. Changes to static variables affect all instances of the class. Example: In a Car class, `num_wheels` might be a static variable because all cars typically have the same number of wheels.

```
1 class Product:  
2     total_products = 0  # Static variable to track total pr  
3  
4     def __init__(self, id, name, price, quantity):  
5         self.id = id
```

```
6         self.name = name
7         self.price = price
8         self.quantity = quantity
9         Product.total_products += 1 # Increment total_products
10
11    def display(self):
12        print(f"Product ID: {self.id}")
13        print(f"Name: {self.name}")
14        print(f"Price: ${self.price}")
15        print(f"Quantity: {self.quantity}")
16        print()
17
18    @staticmethod
19    def display_total_products():
20        print(f"Total Products: {Product.total_products}")
21
22 class Inventory:
23     def __init__(self):
24         self.products = []
25
26         self.menu() # Call menu method when Inventory object is created
27
28     def menu(self):
29         print("Menu:")
30         print("1. Press 1 to add product")
31         print("2. Press 2 to search product")
32         print("3. Press 3 to display inventory")
33         choice = input("Enter your choice: ")
34
35         if choice == "1":
36             self.add_product()
37         elif choice == "2":
38             self.search_product_by_id()
39         elif choice == "3":
40             self.display_inventory()
41         else:
42             print("Invalid choice!")
43         #self.menu() # Display menu again if choice is invalid
```

```
45     def add_product(self):
46         n = eval(input("Enter number of products u want to
47         for i in range(n):
48             id = eval(input("Enter product id "))
49             name = input("Enter product name ")
50             price = eval(input("Enter product price "))
51             quantity = int(input("Enter product qunatity "))
52             product1 = Product(id, name, price, quantity)
53             self.products.append(product1)
54         self.menu()
55
56     def search_product_by_id(self):
57         search_id = int(input("Enter product id that you wa
58
59         for product in self.products:
60             if product.id == search_id:
61                 #return product
62                 print(f"Product with ID {search_id} found:")
63                 product.display()
64             else:
65                 print(f"Product with ID {search_id} not fou
66         self.menu()
67         #return None
68
69     def display_inventory(self):
70         print("Inventory:")
71
72         for product in self.products:
73             product.display()
74         print(f"Total products are {Product.display_total_p
75         self.menu()
76 # Creating some products
77 #product1 = Product(1, "Apple", 1.5, 100)
78 #product2 = Product(2, "Banana", 0.75, 150)
79 #product3 = Product(3, "Orange", 2.0, 80)
80
81 # Creating an inventory
82 inventory = Inventory()
83
```

```

84 # Adding products to the inventory
85 #inventory.add_product(product1)
86 #inventory.add_product(product2)
87 #inventory.add_product(product3)
88
89 # Displaying the inventory
90 #inventory.display_inventory()
91
92 # Searching for a product by ID
93 search_id = 2
94 found_product = inventory.search_product_by_id(search_id)
95 if found_product:
96     print(f"Product with ID {search_id} found:")
97     found_product.display()
98 else:
99     print(f"Product with ID {search_id} not found.")
100

```

→ Menu:  
 1. Press 1 to add product  
 2. Press 2 to search product  
 3. Press 3 to display inventory  
 Enter your choice: 1  
 Enter number of products u want to add 2

```

1 class Student:
2     sno=1
3     program='Computer Science'
4     def __init__(self, name, age):
5         self.name = name
6         self.age = age
7         self.id=Student.sno # use object to access the inst
8         Student.sno=Student.sno+1 # use class name to acces
9
10
11     # Getter methods
12     def get_id(self):
13         return self.id
14
15     def get_name(self):
16         return self.name

```

```
17
18     def get_age(self):
19         return self.age
20
21     def get_Program():
22         return Student.program
23
24     # Setter methods
25     def set_id(self, id):
26         self.id = id
27
28     def set_name(self, name):
29         self.name = name
30
31     def set_age(self, age):
32         self.age = age
33
34 # Example usage:
35 student1 = Student("Ali", 20)
36 student2 = Student("hassan", 21)
37 student3 = Student("kareem", 24)
38 print(student1.id)
39 print(student2.id)
40 print(student3.id)
41 print(student1.sno) # Acess static variable using object
42 print(student2.sno)
43 print(student3.sno)
44 print(Student.sno) # Access static variable using class name
45 lst=[student1, student2, student3]
46 # Using getter methods to access attributes
47 for std in lst:
48     print("ID:", std.get_id())
49     print("Name:", std.get_name())
50     print("Age:", std.get_age())
51     print("Program:", Student.get_Program())
52     print("-----")
53
54 # Using setter methods to modify attributes
55 #student1.set_age(21)
```

```
56 #print("Updated Age:", student1.get_age())
57
```

```
→ 1
 2
 3
 4
 4
 4
 4
ID: 1
Name: Ali
Age: 20
Program: Computer Science
-----
ID: 2
Name: hassan
Age: 21
Program: Computer Science
-----
ID: 3
Name: kareem
Age: 24
Program: Computer Science
-----
```

```
1 class Student:
2     sno=1
3     program='Computer Science'
4     def __init__(self, id, name, age):
5         self.name = name
6         self.age = age
7         self.id=Student.sno # use object to access the inst
8         Student.sno=Student.sno+1 # use class name to acces
9
10
11    # Getter methods
12    def get_id(self):
13        return self.id
14
15    def get_name(self):
16        return self.name
17
18    def get_age(self):
19        return self.age
```

```

20
21     def get_Program():
22         return Student.program
23
24     # Setter methods
25     def set_id(self, id):
26         self.id = id
27
28     def set_name(self, name):
29         self.name = name
30
31     def set_age(self, age):
32         self.age = age
33
34 # Example usage:
35 Student.sno="dddd" # if junior program assignn string, code
36 student1 = Student(1, "Ali", 20)
37 student2 = Student(2, "hassan", 21)
38 student3 = Student(1, "kareem", 24)
39
40
41 # Using setter methods to modify attributes
42 #student1.set_age(21)
43 #print("Updated Age:", student1.get_age())
44

```

→ -----

TypeError Traceback (most recent call last)

```

Cell In[14], line 36
  34 # Example usage:
  35 Student.sno="dddd" # if junior program assignn string, code will crash,
  solution is to make sno as private
--> 36 student1 = Student(1, "Ali", 20)
    37 student2 = Student(2, "hassan", 21)
    38 student3 = Student(1, "kareem", 24)

Cell In[14], line 8, in Student.__init__(self, id, name, age)
      6 self.age = age
      7 self.id=Student.sno # use object to access the instance variable, here self
refers to the object
--> 8 Student.sno=Student.sno+1

TypeError: can only concatenate str (not "int") to str

```

```
1 class Student:
2     __sno=1
3     __program='Computer Science'
4     def __init__(self, name, age):
5         self.name = name
6         self.age = age
7         self.id=Student.__sno # use object to access the in
8         Student.__sno=Student.__sno+1 # use class name to a
9
10
11    # Getter methods
12    def get_id(self):
13        return self.id
14
15    def get_name(self):
16        return self.name
17
18    def get_age(self):
19        return self.age
20
21    @staticmethod # this keyword signifies that the method
22    def get_SNo():
23        return Student.__sno
24
25    @staticmethod # this keyword signifies that the method
26    def get_Program():
27        return Student.__program
28
29    # Setter methods
30    def set_id(self, id):
31        self.id = id
32
33    def set_name(self, name):
34        self.name = name
35
36    def set_age(self, age):
37        self.age = age
38
39    @staticmethod # this keyword signifies that the method
```

```
40     def set_Program(prg): # static method deals with static variable
41         if type(prg)==str:
42             Student.__program=prg
43         else:
44             print("invalid value")
45     @staticmethod # this keyword signifies that the method belongs to the class
46     def set_SNo(sno1):
47         if type(sno1)==int:
48             Student.__sno=sno1
49         else:
50             print("invalid value")
51
```

```
1
2 #Student.sno="dddd" # if junior program assignn string, code will break
3 student1 = Student("Miraj", 20)
4 student2 = Student("hassan", 21)
5 student3 = Student("kareem", 24)
6 #Student.set_SNo(4)
7 Student.get_SNo()
8
```

→ 4

```
1 Student.set_Program(222)
```

→ invalid value

```
1 Student.set_Program('SE')
2 Student.get_Program()
```

→ 'SE'

1. Two Relationships between classes in Python: Aggregation and Inheritance
2. Aggregation represents a 'Has-a' relationship, while inheritance signifies an 'Is-a' relationship.
3. In the context of classes, the relationship between the customer and address classes exemplifies aggregation, where a customer 'has' an address.
4. For instance, a car 'is a' vehicle, and a smartphone 'is a' product. Inheritance allows a car class to automatically inherit all the features of a vehicle class, illustrating the 'is-a'

relationship."

Extra

1. 2 Relationships between classes: Aggregation and Inheritance
2. Aggregation is Has-a relationship, and inheritance is is-a relationship
3. The relationship between customer and address classes is aggregation, customer has address
4. Car is a vehicle, smartphone is a product, car inherits all features of vehicle automatically

```
1 class Customer:  
2     def __init__(self, name, gender, address):  
3         self.name=name  
4         self.gender=gender  
5         self.address=address  
6  
7     def edit_profile(self, name,new_city, new_postalcode, ne  
8         self.name=name  
9         self.address.change_address(new_city, new_postalcode  
10  
11 class Address:  
12     def __init__(self, city, postalcode, state):  
13         self.city=city  
14         self.postalcode=postalcode  
15         self.state=state  
16  
17     def change_address(self,new_city, new_postalcode, new_s  
18         self.city=new_city  
19         self.postalcode=new_postalcode  
20         self.state=new_state  
21
```

```
1 address= Address("Peshawar", 2500, "KPK")  
2 cust1= Customer("Asif","male",address )  
3 print(cust1.name)  
4 print(cust1.address.city)  
5 print(cust1.address.postalcode)  
6 print(cust1.address.state)  
7 #print(cust1.edit_profile("Fazal Khan", ""))
```

```
→ Asif  
Peshawar  
2500  
KPK
```

```
1 cust1.edit_profile("Fazal Khan", "Topi", "24460", "KP")  
2 print(cust1.name)  
3 print(cust1.address.city)  
4 print(cust1.address.postalcode)  
5 print(cust1.address.state)
```

```
→ Fazal Khan  
Topi  
24460  
KP
```

```
1 # Using vars()  
2 for attribute, value in vars(cust1).items():  
3     print(attribute, ":", value)
```

```
→ name : Asif  
gender : male  
address : <__main__.Address object at 0x000001E0F31810D0>
```

```
1  
2  
3 # Using __dict__  
4 for attribute, value in cust1.__dict__.items():  
5     print(attribute, ":", value)
```

```
→ name : Asif  
gender : male  
address : <__main__.Address object at 0x000001E0F31810D0>
```

1. if `hasattr(value, 'dict')`, It checks if the value object has a **dict** attribute, which suggests that it's an instance of a user-defined class
2. If it does, the code assumes that value is an object instance and proceeds to loop over its attributes.
3. If it doesn't have a **dict** attribute, it treats value as a non-object attribute and prints its value directly.

```

1 # Function to check if the attribute is an object instance a
2 def print_attributes(obj):
3     for attribute, value in vars(obj).items():
4         if hasattr(value, '__dict__'): # Check if the attri
5             print(attribute + ":")
6             for sub_attribute, sub_value in vars(value).item
7                 print("\t" + sub_attribute + ":", sub_value)
8         else:
9             print(attribute + ":", value)
10
11 # Looping over attributes of the customer object
12 print("Customer's Attributes:")
13 print_attributes(cust1)

```

→ Customer's Attributes:

```

name: Asif
gender: male
address:
    city: Peshawar
    postalcode: 2500
    state: KPK

```

1. Inheritance--society level and biological level--property of your father is inherited to children, biological: facial or other features are inherited to children. for example, developing udemy type app, two users, students and instructors. build 2 classes.
2. Student class with methods: login(), registration(), enroll courses, reviews()
3. Instructor class methods:login(), registration(), create courses, Answer-to-questions bad programming, if you repeat the same code that you already wrote, philosophy of writing good code: principle:do not repeat yourself
4. here you can apply concept of inheritance, by creating a user class with common methods, and derive student and instructor classes from user class.
5. with the help of inheritance, the children classes can inherit all data members, constructor and all methods from parent class. only private data members can not be inherited.
6. biggest advantage of inheritance especially in big applications is code reusability, which will save time saving, code concise and optimize and code will perform better
  
1. Code Reusability: Inheritance allows you to create a new class that is based on an existing class, inheriting its attributes and methods. This promotes code reuse as you can leverage existing code without having to rewrite it.

2. Time Savings: By reusing existing code through inheritance, developers can save time that would otherwise be spent writing and debugging new code for similar functionality.
3. Conciseness: Inheritance promotes concise code by allowing you to define common behavior and attributes in a parent class, reducing the need for redundant code in subclasses.
4. Optimization: Inheritance can help optimize code structure by organizing related classes hierarchically. This makes it easier to understand and maintain the codebase, leading to better overall performance.
5. Performance: In some cases, inheritance can lead to better performance because it allows for the reuse of optimized code from parent classes.

```
1 # Student is a sub-class of user class and object of student
2 #class due to inheritance.
3 # Student class is inheriting from user class.
4 #Reverse inheritance is not possible,that is user class obj
5 class user:
6     def login(self):
7         print("login done")
8
9     def register(self):
10        print("registration done")
11
12 class Student (user):
13     def enroll (self):
14         print("enrollment done")
15
16     def review(self):
17         print("review done")
18
19 class Instructor (user):
20     def create_course (self):
21         print("Courses created")
22
23     def Answer_questions(self):
24         print("Ans done")
25
```

```
1 std= Student()
2 std.enroll()
3 std.login()
4 std.register()
5 std.review()
```

→ enrollment done  
login done  
registration done  
review done

```
1 user1=user() # reverse inheritance is not possible
2 user1.login()
3 user1.register()
4 user1.enroll()
5 user1.review()
```

→ login done  
registration done

```
AttributeError
Cell In[4], line 4
    2 user1.login()
    3 user1.register()
----> 4 user1.enroll()
      5 user1.review()
```

Traceback (most recent call last)

```
AttributeError: 'user' object has no attribute 'enroll'
```

```
1 # Inheriting constructor
2 # concept, if class B is inheriting from Class A, while crea
3 #if the constructor of class B does not exist, then class A
4
5
6 class Phone:
7     def __init__(self,price, brand, camera):
8         self.price=price
9         self.brand=brand
10        self.camera=camera
11
12 class SmartPhone(Phone): # Smartphone is inheriting from pho
13     pass
14
```

```
15 phone=SmartPhone(3000, "Samsung", 14)
16 print(phone.brand)
17 print(phone.price)
18
```

→ Samsung  
3000

```
1 # Attempting to inherit private data members
2 # Object of child class can inherit only public members of p
3 class Phone:
4     def __init__(self, price, brand, camera):
5         self.price=price
6         self.__brand=brand # private data member
7         self.camera=camera
8 class SmartPhone(Phone): # Smartphone is inheriting from pho
9     pass
10
11 phone=SmartPhone(3000, "Samsung", 14)
12 print(phone.__brand) # code will crash here
13
```

→ -----  
**AttributeError** Traceback (most recent call last)  
Cell In[7], line 12  
 9 pass  
 11 phone=SmartPhone(3000, "Samsung", 14)  
---> 12 print(phone.\_\_brand) # code will crash here  
  
**AttributeError**: 'SmartPhone' object has no attribute '\_\_brand'

```
1 # 3 concepts comes under polymorphism
2 # method overriding-->polymorphism
3 # method overloading-->polymorphism
4 # operator overloading-->polymorphism
```

```
1 #polymorphism
2 class Phone:
3     def __init__(self, price, brand, camera):
4         self.price=price
5         self.__brand=brand # private data member
```

```

6         self.camera=camera
7     def buy (self):
8         print("buying a phone")
9
10 class SmartPhone(Phone): # Smartphone is inheriting from pho
11     def buy (self):
12         print("buying a smart phone")
13
14 phone=SmartPhone(3000, "Samsung", 14)
15 phone.buy()    #method overrding
16 p1=Phone(400, "Phone", 14)
17 p1.buy()

```

→ buying a smart phone  
buying a phone

1. we see the object of the subclass can access the method of the superclass.
2. However, what if the same method is present in both the superclass and subclass?
3. In this case, the method in the subclass overrides the method in the superclass. This concept is known as method overriding in Python.

or

1. Method overriding is an ability of any object-oriented programming language that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes.
2. When a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.
3. The version of a method that is executed will be determined by the object that is used to invoke it. If an object of a parent class is used to invoke the method, then the version in the parent class will be executed,
4. but if an object of the subclass is used to invoke the method, then the version in the child class will be executed.
5. In other words, it is the type of the object being referred to (not the type of the reference variable) that determines which version of an method will be executed.

1 Analogy of method overriding from real world

2 1.Imagine a company where there's a standard procedure for s  
3 2.Each department has its own guidelines for report formatti

4 3. For example, the Marketing department might override the  
5 4. while the Finance department might include financial fore  
6 4. Despite these variations, all reports still adhere to the

```
1 class parent:  
2     def __init__(self, num):  
3         self.num=num  
4  
5     def get_num(self):  
6         return self.num  
7  
8 class child(parent):  
9     def show(self):  
10        print("child class")  
11  
12 c1= child(200) # python will call child class constructor, i·  
13 c1.get_num()
```

→ 200

```
1 class parent:  
2     def __init__(self, num):  
3         self.num=num  
4     def get_num(self):  
5         return self.num  
6  
7 class child(parent):  
8     def show(self):  
9         print("child class")  
10  
11 c1= child(100) # python will call child class constructor, i·  
12 c1.get_num()
```

→ 100

```
1 #both parent and child classes have constructor  
2 class parent:  
3     def __init__(self, num):  
4         self.__num=num           #self.num is not initialized
```

```

5
6     def get_num(self):
7         print("hello")
8         return self.__num
9
10 class child(parent): # if no constructor in child class, t
11     def __init__(self, val, num): # but if child class has i
12         self.__val= val
13         self.__num=num
14
15     def get_val(self):
16         return self.__val
17
18
19     def show(self):
20         print("child class")
21
22 son= child(100,200) # object of child clas
23 print("Child val", son.get_val())
24 print("parent num",son.get_num())
25

```

```

→ Child val 100
hello
-----
AttributeError                                     Traceback (most recent call last)
Cell In[16], line 24
      22 son= child(100,200) # object of child clas
      23 print("Child val", son.get_val())
---> 24 print("parent num",son.get_num())

Cell In[16], line 8, in parent.get_num(self)
      6 def get_num(self):
      7     print("hello")
---> 8     return self.__num

AttributeError: 'child' object has no attribute '_parent__num'

```

```

1 class parent:
2     def __init__(self, num):
3         self.__num=num           #self.num is not initialized
4
5     def get_num(self):

```

```

6         return self.__num
7
8 class child(parent): # if no constructor in child class, t
9     def __init__(self, val, num): # but if child class has i
10        super().__init__(num) # Explicitly calling the pare
11        self.__val= val
12
13    def get_val(self):
14        return self.__val
15
16    def show(self):
17        print("child class")
18
19 son= child(100,200) # object of child clas
20 print("parent num",son.get_num())
21 print("Child val", son.get_val())

```

→ parent num 200  
Child val 100

```

1 class A:
2     def __init__(self):
3         self.var1=100
4
5     def display1(self, var1):
6         print("Class A", self.var1)
7
8 class B(A):
9     def display2(self, var1):
10        print("Class B", self.var1)
11
12 obj= B()
13 obj.display1(200)

```

→ Class A 100

```

1 class Phone:
2     def __init__(self,price, brand, camera):
3         print("inside parent class constructor")
4         self.price=price

```

```

5         self.__brand=brand # private data member
6         self.camera=camera
7
8     def buy (self):
9         print("buying a phone")
10
11 class SmartPhone(Phone): # Smartphone is inheriting from pho
12     def buy (self):
13         print("buying a smart phone")
14         super().buy()
15 # super keyword is only used to call parent class methods/co
16 #it can not be used to access Parent class data member
17 #super keyword can not be used outside class
18
19 phone=SmartPhone(3000, "Samsung", 14)
20 phone.buy()    #method overrding

```

→ inside parent class constructor  
 buying a smart phone  
 buying a phone

```

1 #both parent and child have constructurs, and we used super
2
3 #constructor
4 class Phone:
5     def __init__(self,price, brand, camera):
6         print("inside parent class constructor")
7         self.__price=price
8         self.brand=brand # private data member
9         self.camera=camera
10
11     def buy (self):
12         print("buying a phone")
13
14 # smart coding, half initialization in my constrcutor and th
15 #(or half initailization) parent class constructor
16 # resuing the coding of parent class constructor rather than
17 #child class constructor
18
19 class SmartPhone(Phone): # Smartphone is inheriting from pho

```

```

20     def __init__(self, price, brand, camera, os, ram): # init
21         print("inside child class constructor")
22         # call the parent call constructor to initialized th
23         super().__init__(price, brand, camera)
24         self.os=os
25         self.ram=ram
26
27     def buy (self):
28         print("buying a smart phone")
29         super().buy()          # super keyword is only used to
30                           #super keyword can not be used
31
32 phone=SmartPhone(3000, "Samsung", 14, "Adroid",2)
33 print(phone.os)
34 print(phone.brand)
35 print(phone.camera)
36

```

→ inside child class constructor  
 inside parent class constructor  
 Adroid  
 Samsung  
 14

```

1 class parent:
2     def __init__(self, num):
3         self.__num=num           #self.num is initialized in t
4     def get_num(self):
5         return self.__num
6
7 class child(parent):
8     def __init__(self, val, num):
9
10        super().__init__(num) # the statement having super k
11        self.__val= val
12
13    def get_val(self):
14        return self.__val
15
16    def show(self):

```

```
17         print("child class")
18
19 son= child(100,200) # object of child clas
20 print("Son num",son.get_num())
21 print("SOn val", son.get_val())
```

→ Son num 200  
SOn val 100

```
1 class parent:
2     def __init__(self):
3         self.num=100
4
5     def get_num(self):
6         return self.num
7
8 class child(parent):
9     def __init__(self):
10
11         super().__init__() # the statement having super keyword
12         self.val= 200
13
14     # can we access the parent class attribute in the child
15     def show(self): # self contains reference to child obje
16         print(self.num)
17         print(self.val)
18
19 son= child() # object of child clas
20 son.show()      # son and self above are same, since son can
21
```

→ 100  
200

```
1 class parent:
2     def __init__(self):
3         self.__num=100
4
5     def show(self):
6         print("Parent:", self.__num)
```

```

7
8 class child(parent):
9     def __init__(self):
10         super().__init__() # the statement having super keyword
11         self.__val= 200
12
13
14     # can we access the parent class attribute in the child
15     def show(self): # self contains reference to child object
16
17         print("Child:", self.__val)
18
19 P=parent() #parent object
20 P.show()
21 son= child() # object of child class
22 son.show()    # Due to method overriding, child class show m

```

→ Parent: 100  
Child: 200



There are five types of inheritance in python

1. Single Level: single child, single parent, a child inherit from the parent class
2. Multi-level: child, papa, grandpapa, child inherits the properties of papa as well as grandpapa. and papa inherits the properties of grandpapa.
3. heirarchical: Two child, single parent, two child inherit the properites of single parent
4. Multiple: one child, two parents(mother, father): a child inherit properties from two parent classes
5. hybrid: single level-->multiple inheritance-->heirarchical inheritance

```

1 # Example of single level inheritance
2
3 class Phone:
4     def __init__(self,price, brand, camera):
5         print("inside parent class constructor")
6         self.__price=price
7         self.brand=brand # private data member
8         self.camera=camera

```

```
9
10     def buy (self):
11         print("buying a phone")
12
13 # smart coding, half initialization in my constructor and th
14 # resuing the coding of parent class constructor rather than
15
16 class SmartPhone(Phone): # Smartphone is inheriting from pho
17     pass
18
19 SmartPhone(3000, "Samsung", "14px").buy()
20
```

→ inside parent class constructor  
buying a phone

In Python, if a child class (like SmartPhone) does not define its own `init` constructor, then it inherits the constructor of its nearest parent.

```
1 # Example of Mutli level inheritance
2 class product:
3     def review(self):
4         print("product customer review")
5
6 class Phone(product):
7     def __init__(self,price, brand, camera):
8         print("inside parent class constructor")
9         self.__price=price
10        self.brand=brand # private data member
11        self.camera=camera
12    def buy (self):
13        print("buying a phone")
14
15 # smart coding, half initialization in my constructor and th
16 # inparent class constructor, resuing the coding of parent c
17 #same in child class constructor
18
19 class SmartPhone(Phone): # Smartphone is inheriting features
20     pass
```

```
21
22 s=SmartPhone(3000, "Samsung", "14px")
23 s.buy()      # child class accessing method of parent class
24 s.review()   # child class accessing method of product class
25 p= Phone(4000, "Apple", "13px")
26 p.buy()          #object of parent class accesing its own i
27 p.review()       #object of parent class accesing method o
```

→ inside parent class constructor  
buying a phone  
product customer review  
inside parent class constructor  
buying a phone  
product customer review

```
1 # Example of hierarchical inheritance
2
3 class Phone:
4     def __init__(self,price, brand, camera):
5         print("inside parent class constructor")
6         self.__price=price
7         self.brand=brand # private data member
8         self.camera=camera
9
10    def buy (self):
11        print("buying a phone")
12
13 class SmartPhone(Phone): # Smartphone is inheriting features.
14     pass
15
16 class Feature(Phone): # Feature is inheriting features from
17     pass
18
19 s=SmartPhone(3000, "Samsung", "14px")
20 s.buy()
21
22
```

→ inside parent class constructor  
buying a phone

```

1 class Phone:
2     def __init__(self, price, brand, camera):
3         print("inside parent class constructor")
4         self.__price = price
5         self.brand = brand # private data member
6         self.camera = camera
7
8     def buy(self):
9         print("buying a phone")
10
11
12 class SmartPhone(Phone): # Smartphone is inheriting feature
13     def __init__(self, price, brand, camera, os, ram):
14         super().__init__(price, brand, camera) # Call the p
15         self.os = os
16         self.ram = ram
17
18
19 class Feature(Phone): # Feature is inheriting features from
20     def __init__(self, f1, f2):
21         self.f1 = f1
22         self.f2 = f2
23 #he Feature class does not have its own __init__ method. Ins
24 #from its parent class, Phone.
25 #Therefore, when you create an object of the Feature class a
26 #and "10px",they are passed to the constructor of the parent
27
28 s=SmartPhone(3000, "Samsung", "14px","Android","4GB")
29 s.buy()
30 F=Feature("finger print sensor", "16GB") # Creating an obje
31
32 F.buy() # Calling the buy method
33

```

→ inside parent class constructor  
 buying a phone  
 buying a phone

```

1 # Example of Mutliple inheritance
2

```

```

3 class Product:
4     def review(self):
5         print("product customer review")
6
7 class Phone:
8     def __init__(self, price, brand, camera):
9         print("inside parent class constructor")
10        self.__price=price
11        self.brand=brand # private data member
12        self.camera=camera
13    def buy (self):
14        print("buying a phone")
15
16 #you can inherit from any number of classes
17 class SmartPhone(Phone, Product): # Smartphone is inheriting
18     pass
19
20 s=SmartPhone(3000, "Samsung", "14px")
21 s.buy()      # child class accessing method of parent class
22 s.review()   # child class accessing method of product class
23

```

→ inside parent class constructor  
 buying a phone  
 product customer review

```

1 # Example of Multiple inheritance
2 # Classes product and Phone both have constructors
3 class Product:
4     def __init__(self):
5         print("Inside the Product class constructor")
6     def review(self):
7         print("product customer review")
8
9 class Phone:
10    def __init__(self, price, brand, camera):
11        print("inside Phone class constructor")
12        self.__price=price
13        self.brand=brand # private data member
14        self.camera=camera

```

```

15     def buy (self):
16         print("buying a phone")
17
18 #you can inherit from any number of classes
19 # Smartphone inherits constructors from both Phone and Product
20 #now which constructor will be invoked,
21 # since phone class appears first, so its constructor will be invoked
22 #then product class constructor will be invoked
23
24 class SmartPhone(Phone, Product): # Smartphone is inheriting
25     pass
26
27 s=SmartPhone(3000, "Samsung", "14px")
28 s.buy()      # child class accessing method of parent class
29 s.review()   # child class accessing method of product class

```

→ inside Phone class constructor  
 buying a phone  
 product customer review

```

1 # Example of Multiple inheritance
2 # only Class product has constructor
3
4 class Product:
5     def __init__(self):
6         print("Inside the Product class constructor")
7     def review(self):
8         print("product customer review")
9
10 class Phone:
11     #def __init__(self,price, brand, camera):
12     #    print("inside Phone class constructor")
13     #    self.__price=price
14     #    self.brand=brand # private data member
15     #    self.camera=camera
16     def buy (self):
17         print("buying a phone")
18
19 #you can inherit from any number of classes
20 # Smartphone inherits constructors from both Phone and Product

```

```
21 # since phone class appears first, so its constructor will b
22 class SmartPhone(Phone, Product): # Smartphone is inheriting
23     pass
24
25 #s=SmartPhone(3000, "Samsung", "14px")
26 s=SmartPhone()
27 s.buy()      # child class accessing method of parent class
28 s.review()   # child class accessing method of product class
```

→ Inside the Product class constructor  
buying a phone  
product customer review

```
1 # Example of Multiple inheritance
2 # Both Classes product and Phone have no constructors
3 class Product:
4
5     def review(self):
6         print("product customer review")
7
8 class Phone:
9
10    def buy (self):
11        print("buying a phone")
12
13 class SmartPhone(Phone, Product): # Smartphone is inheriting
14     pass
15
16 #s=SmartPhone(3000, "Samsung", "14px")
17 s=SmartPhone()
18 s.buy()      # child class accessing method of parent class
19 s.review()   # child class accessing method of product class
```

→ buying a phone  
product customer review

```
1 # Method Resolution order, if both product and phone have bu
2 #inherit from both classes
3 # then which buy method will be called, in this conflicting
4 #based on the order in which classes are written
```

```

5 # if Phone class is written first, then its buy() will be cal
6
7 class Product:
8     def review(self):
9         print("product customer review")
10
11    def buy (self):
12        print("Product buy method")
13
14 class Phone:
15     def __init__(self,price, brand, camera):
16         print("inside parent class constructor")
17         self.__price=price
18         self.brand=brand # private data member
19         self.camera=camera
20
21     def buy (self):
22         print("buying a phone")
23
24
25 #you can inherit from any number of classes
26 class SmartPhone(Phone, Product): # Smartphone is inheriting
27     pass
28
29 s=SmartPhone(3000, "Samsung", "14px")
30 s.buy()

```

→ inside parent class constructor  
buying a phone

```

1 class Product:
2     def review(self):
3         print("product customer review")
4
5     def buy (self):
6         print("Product buy method")
7
8 class Phone:
9     def __init__(self,price, brand, camera):

```

```

10     print("inside parent class constructor")
11     self.__price=price
12     self.brand=brand # private data member
13     self.camera=camera
14
15     def buy (self):
16         print("buying a phone")
17
18
19 #you can inherit from any number of classes
20 class SmartPhone(Product, Phone): # Smartphone is inheriting
21     pass
22
23 s=SmartPhone(3000, "Samsung", "14px")
24 s.buy()

```

→ inside parent class constructor  
 Product buy method

```

1 # Multi-level Inheritance
2 class A:
3     def m1(self):
4         return 20
5
6 class B(A):
7     def m1(self):
8         return 30
9
10    def m2(self):
11        return 40
12
13 class C(B):
14     def m2(self):
15         return 20
16
17 obj1= A()
18 obj2= B()
19 obj3= C()
20 print(obj1.m1() + obj3.m1() + obj3.m2()) # 20+30+20

```

```
1 # Multi-level Inheritance
2 class A:
3     def m1(self):
4         return 20
5
6 class B(A):
7     def m1(self):
8         val= super().m1() + 30
9         return val
10
11 class C(B):
12     def m1(self):
13         val = self.m1() + 20 # infinite recursion , obj3.m1()
14         return val
15
16
17 obj3= C()
18 obj3.m1()
```

```
→ -----  
RecursionError Traceback (most recent call last)  
Cell In[27], line 18  
 14     return val  
 17 obj3= C()  
---> 18 obj3.m1()  
  
Cell In[27], line 13, in C.m1(self)  
 12 def m1(self):  
---> 13     val = self.m1() + 20 # infinite recursion , obj3.m1()=self.m1()  
 14     return val  
  
Cell In[27], line 13, in C.m1(self)  
 12 def m1(self):  
---> 13     val = self.m1() + 20 # infinite recursion , obj3.m1()=self.m1()  
 14     return val  
  
[... skipping similar frames: C.m1 at line 13 (2974 times)]  
  
Cell In[27], line 13, in C.m1(self)  
 12 def m1(self):  
---> 13     val = self.m1() + 20 # infinite recursion , obj3.m1()=self.m1()  
 14     return val  
  
RecursionError: maximum recursion depth exceeded
```

1. In Python, polymorphism can indeed be achieved through method overriding, method overloading, and operator overloading,
2. Method overloading is achieved by providing different inputs to same method to induce different behavior
3. So, while you can indeed create methods in Python that behave differently based on the inputs provided,
4. it's not achieved through traditional method overloading as seen in languages like Java. Instead, it's more about writing flexible methods that can handle different inputs gracefully.

```
1 # this code will work perfectly well in java but not work in  
2 # In python, we can not define multiple methods with same, p  
3 #to behave differently  
4 # this code will work in java, here the second area method h  
5 # Technically, method overlaoding does not work in python  
6 class Geometry:  
7     def area(self, radius):  
        return 3.14*radius * radius  
9  
10    def area(self, l, b):
```

```
11         return l*b  
12  
13 obj= Geometry()  
14 obj.area(3)
```

→ 9

```
1 obj.area(3,4)
```

→ 12

1. Technically speaking, traditional method overloading, as seen in languages like Java or C++, where you can define multiple methods with the same name but different parameter lists, does not exist in Python.
2. In Python, if you define multiple methods with the same name, the last method definition will overwrite any previous ones with the same name, and only the latest method definition will be retained.
3. However, you can achieve similar functionality by using default argument values or variable-length argument lists (*args and \*kwargs*). This allows you to define a single method that can handle different numbers or types of arguments gracefully.
4. OR this allow u to define a single method that will behave differently on different inputs.
5. While this is not traditional method overloading, it serves a similar purpose in allowing you to create flexible methods that behave differently based on the inputs provided.
6. So, in summary, while traditional method overloading does not exist in Python, you can achieve similar behavior using other techniques available in the language.

```
1 #method overloading acheived through default argument  
2 class Geometry:  
3  
4     def area(self, l, b=0):  
5         if b==0:  
6             print("Area of circle", 3.14*l*l)  
7         else:  
8             print("Area of Rectangle", l*b)  
9  
10 obj= Geometry()
```

```
11 obj.area(3)
12 obj.area(3,3)
```

```
→ Area of circle 28.25999999999998
Area of Rectangle 9
```

1. {}: This is a placeholder for the value that will be inserted into the formatted string.
2. : This indicates the start of the format specification.
3. .2: This specifies the precision of the floating-point number, i.e., the number of decimal places to display. In this case, it's 2, indicating that we want to display the number with two decimal places.
4. f: This specifies that the placeholder should be filled with a floating-point number. So, "{:.2f}" is a format string that tells Python to format the inserted value as a floating-point number with two decimal places.

```
1 class Geometry:
2     def area(self, *args):
3
4         if len(args) == 1:
5             if isinstance(args[0], (float)) and args[0] > 0:
6                 # Circle: area = π * r^2
7                 radius = args[0]
8                 # Format the area to two decimal places
9                 #formatted_circle_area = "{:.2f}".format(cir
10                #{:.2}
11
12                print("Area of circle:", "{:.3f}".format(3.14159*radius**2))
13            else:
14                # Square: area = side * side
15                side = args[0]
16                print("Area of Square:", side * side)
17
18            elif len(args) == 2:
19                # Rectangle: area = length * breadth
20                length, breadth = args
21
22                print("Area of Rectangle:", length * breadth)
23
24            #elif len(args) == 3:
```

```
25         # Triangle: area = (base * height) / 2
26         # base, height = args[:2]
27         #print("Area of Triangle:", (base * height) / 2)
28     else:
29         print("Invalid number of arguments provided")
30
31 obj = Geometry()
32 obj.area(3.0) # Area of circle
33 obj.area(4, 5) # Area of Rectangle
34 #obj.area(3, 4, 5) # Area of Triangle
35 obj.area(4) # Area of Square
36
```

→ Area of circle: 28.260  
Area of Rectangle: 20  
Area of Square: 16

1. In Python, we can change the way operators work for user-defined types.
2. For example, the + operator will perform arithmetic addition on two numbers, merge two lists, or concatenate two strings.
3. This feature in Python that allows the same operator to have different meaning according to the context is called operator overloading.

1 Start coding or generate with AI.

## ▼ Notebook: 14) Numpy\_lec1-part-a.ipynb

6 different methods for creating numpy array

```
1 import numpy as np
```

```
1 !pip install numpy
```

```
1 # numpy is an external moduel, u need to install it, it does
2 # to include numpy capabilities in your python code
```

```
3 import numpy as np
```

```
1 arr1= np.array([1,2,3,4]) # array method converts list into  
2 arr1 # vector of 4 elements
```

```
→ array([1, 2, 3, 4])
```

```
1 type(arr1) # ndarray type object inside numpy library
```

```
→ numpy.ndarray
```

```
1 #create 2 dimensional n-d array by passing 2 d list  
2 arr2=np.array([[1,2,3],[4,5,6]])  
3 arr2
```

```
→ array([[1, 2, 3],  
[4, 5, 6]])
```

```
1 type(arr2)
```

```
→ numpy.ndarray
```

```
1 # if u pass the shape in the form of tuple, so zeros function  
2 # with all elements initialized to zero, quickly create a ar  
3 arr3=np.zeros((4,4))  
4 arr3
```

```
→ array([[0., 0., 0., 0.],  
[0., 0., 0., 0.],  
[0., 0., 0., 0.],  
[0., 0., 0., 0.]])
```

```
1 arr4=np.ones((3,3)) # creates an N-D array of size 3 by 3 wi  
2 arr4  
3 # the type of all elements here in this array are float. mat  
4 # captures more scenarios
```

```
→ array([[1., 1., 1.],  
[1., 1., 1.],  
[1., 1., 1.]])
```

```
1 # creates an identity matrix of order 5 (a matrix with 5 row  
2 #and rest of elements are zero  
3 arr5= np.identity(6)  
4 arr5
```

```
→ array([[1., 0., 0., 0., 0.],  
         [0., 1., 0., 0., 0.],  
         [0., 0., 1., 0., 0.],  
         [0., 0., 0., 1., 0.],  
         [0., 0., 0., 0., 1.],  
         [0., 0., 0., 0., 0.]])
```

1 Start coding or generate with AI.

```
1 # range function generates a list in a given range  
2 #Its counter function in numpy is arange which creates an ar  
3 arr5=np.arange(10)  
4 arr5
```

```
→ array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
1 arr5=np.arange(10,26) # creates an array with elements 10 to  
2 arr5
```

```
→ array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25])
```

```
1 arr6=np.arange(4,51,2) # creates an array with elements 5 to  
2 arr6
```

```
→ array([ 4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36,  
        38, 40, 42, 44, 46, 48, 50])
```

1. The np.linspace function calculates the step size required to generate the specified number of values between the start and stop values.

- 2.
- 3.
- 4.

```
1 arr7= np.linspace(10,20,10) ## start at 10, end at 20, gener  
2 arr7
```

```
→ array([10. , 11.111111, 12.2222222, 13.3333333, 14.44444444,  
       15.55555556, 16.66666667, 17.77777778, 18.88888889, 20. ])
```

```
1 arr8= np.linspace(20,30,5) ## start at 10, end at 20, genera  
2 arr8
```

```
→ array([20. , 22.5, 25. , 27.5, 30. ])
```

```
1 arr8=arr7.copy() # creates copy of numpy array  
2 arr8
```

```
→ array([20. , 22.5, 25. , 27.5, 30. ])
```

Numpy is basically a class so it will have properties and attributes 

```
1 ar1=np.array([1,2,3,4])  
2 print(ar1)  
3 print(ar1.shape) # shape is an attribute  
4  
5 #The output (4,) indicates that ar1 is a 1-dimensional array  
6 #The shape of an array is a tuple that describes the size of  
7 #Since the array is 1-dimensional, so the tuple has a single
```

```
→ [1 2 3 4]  
(4,)
```

```
1 ar1.ndim # returns number of dimension of array, 1 in this
```

```
→ 1
```

```
1 ar2=np.array([[1, 2, 3, 4], [5, 6, 7, 8]]) # creates 2d arra  
2 print(ar2)  
3 print(ar2.shape) # shape is (2, 4) indicating 2 rows and 4 c
```

```
→ [[1 2 3 4]  
 [5 6 7 8]]  
(2, 4)
```

```
1 ar3=np.array([[1, 2, 3, 4], [5, 6, 7, 8]]) # creates 2d arra  
2 print(ar3)  
3 print(ar3.shape)
```

```
→ [[1 2 3 4]
 [5 6 7 8]]
(2, 4)
```

```
1 ar2.ndim
```

```
→ 2
```

```
image.png image.png
```

```
1 ar3=np.array([[ [1, 2,5],[3, 4,5]], [[5, 6,1], [7, 8,9]]])
2 print(ar3)
3 print(ar3.shape)
```

```
→ [[[1 2 5]
 [3 4 5]]

 [[5 6 1]
 [7 8 9]]]
(2, 2, 3)
```

The issue with the code arises from the fact that the inner arrays within the outer array have inconsistent shapes. Specifically:

The first sub-array is `[[1, 2, 4], [3, 4, 5]]`, which is a 2x3 array (2 rows and 3 columns). The second sub-array is `[[5, 6], [7, 8]]`, which is a 2x2 array (2 rows and 2 columns). NumPy expects all the sub-arrays to have the same shape to form a proper multi-dimensional array. Since the shapes are different (2x3 and 2x2), NumPy cannot create a regular 3D array.

```
1 ar33=np.array([[ [1, 2,4],[3, 4,5]], [[5, 6], [7, 8]]])
2 print(ar33)
3 print(ar33.shape)
```

```
→ -----
ValueError                                                 Traceback (most recent call last)
Cell In[14], line 1
----> 1 ar33=np.array([[ [1, 2,4],[3, 4,5]], [[5, 6], [7, 8]]])
      2 print(ar33)
      3 print(ar33.shape)
```

**ValueError:** setting an array element with a sequence. The requested array has an inhomogeneous shape after 2 dimensions. The detected shape was (2, 2) + inhomogeneous part.

```
1 print(ar3)
2 ar3.ndim
```

```
→ [[1 2 5]
 [3 4 5]]
```

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

```
1 ar1
```

```
→ array([1, 2, 3, 4])
```

```
1 ar1.size # gives number of items in array
```

```
→ 4
```

```
1 ar2
```

```
→ array([[1, 2, 3, 4],
 [5, 6, 7, 8]])
```

```
1 ar2.size # size is property of ar2 object of numpy
```

```
→ 8
```

```
1 ar2.itemsize
```

```
→ 4
```

```
1 ar3
```

```
→ array([[[1, 2, 5],
 [3, 4, 5]],

 [[5, 6, 1],
 [7, 8, 9]]])
```

```
1 ar3.size
```

```
→ 8
```

image.png image.png

```
1 ar1_int32 = np.array([1, 2, 3, 4, 5, 6], dtype=np.int32)
2
3 print(ar1_int32.itemsize) # Should print 4
4
5 ar1_int64 = np.array([1, 2, 3, 4], dtype=np.int64)
6 print(ar1_int64.itemsize) # Should print 8
7
```

→ 4  
8

```
1 ar3=np.array([[ [1, 2],[3, 4]], [[5, 6], [7, 8]]])
2 print(ar3)
3 print(ar3.shape)
```

→ [[[1 2]
[3 4]]
[[5 6]
[7 8]]]
(2, 2, 2)

```
1 ar3.itemsize # Output the size (in bytes) of each element i
```

→ 4

```
1 ar3.dtype # Output the data type of the array elements
```

→ dtype('int32')

```
1 import numpy as np
2
3 ar1 = np.array([1, 2, 3, 4])
4 print(ar1.shape)      # Output the shape of the array
5 print(ar1.ndim)       # Output the number of dimensions of t
6 print(ar1.dtype)       #Output the data type of the array ele
7 print(ar1.itemsize)    # Output the size (in bytes) of each e
8
```

→ (4,)  
1  
int32  
4

```
1 arr5= np.identity(5)
2 arr5
```

```
→ array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

```
1
2 print(arr5.itemsize)
3 print(arr5.dtype)
```

```
→ 8
float64
```

```
1 arr5
```

```
→ array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

```
1 #Data Type (dtype): float64 indicates that each element is a
2 #Item Size (itemsize): 8 indicates that each float64 element
3
4 import numpy as np
5
6 # Create an array with specified elements and dtype=float64
7 ar_float64 = np.array([1, 2, 3, 4], dtype=np.float64)
8 print(ar_float64)
9 print(ar_float64.dtype)      # Output the data type of the array
10 print(ar_float64.itemsize)   # Output the size (in bytes) of
11
```

```
→ [1. 2. 3. 4.]
float64
8
```

```
1 import numpy as np
2
```

```
3 # Create an array with specified elements and dtype=float64
4 ar_float64 = np.array([1, 2, 3, 4], dtype=np.float64)
5 print(ar_float64)
6 print(ar_float64.dtype)      # Output the data type of the a
7 print(ar_float64.itemsize)   # Output the size (in bytes) of
8
```

```
→ [1. 2. 3. 4.]
float64
8
```

```
1 import numpy as np
2
3 # Create an array with specified elements and dtype=float64
4 ar_float = np.array([1.0, 2.0, 3.0, 4.0]) # by default, numpy
5 print(ar_float)
6 print(ar_float.dtype)      # Output the data type of the arr
7 print(ar_float.itemsize)
```

```
→ [1. 2. 3. 4.]
float64
8
```

```
1 ar3
```

```
→ array([[1, 2],
       [3, 4],
       [[5, 6],
        [7, 8]]])
```

image.png

```
1 #ar3 elements are of type int, we want to convert it into float
2 ar4=ar3.astype(float) #The astype method converts the data type
3 ar4
```

```
→ array([[1., 2.],
       [3., 4.],
       [[5., 6.],
        [7., 8.]]])
```

```
1 ar3
```

```
→ array([[[1, 2],  
          [3, 4]],  
  
          [[5, 6],  
          [7, 8]]])
```

```
1 ar4
```

```
→ array([[[1., 2.],  
          [3., 4.]],  
  
          [[5., 6.],  
          [7., 8.]]])
```

```
1 ar5=ar4.astype(int)
```

```
2 ar5
```

```
→ array([[[1, 2],  
          [3, 4]],  
  
          [[5, 6],  
          [7, 8]]])
```

Context: The text discusses a machine learning (ML) scenario involving data cleaning for a dataset of 100,000 (1 lac) customers. reduce the footprint(size) of the features

Problem: The age field in the dataset has floating point values (e.g., 27.5). Running an algorithm on this dataset can be time-consuming. Solution: Changing the data type of the age field from float to integer (e.g., converting 27.5 to 27) can reduce memory usage by half. This reduction in memory usage can, in turn, decrease the algorithm's execution time.

 python list has limited capability to perform machine learning. Given same operation on numbers in list and numpy array, numpy array will be faster in execution and will take less memory.



✗ Notebook: 15) Numpy\_lec2-3.ipynb

```
1 lista=range(100) # both list and array have 100 elements
2 nparr=np.arange(100)
```

```
1 import sys
2 print(sys.getsizeof(0))
```

→ 28

Fun Fact: Small Integer Caching (CPython)  
CPython pre-allocates and caches integers from -5 to 256 to optimize memory and speed:

```
1 import sys
2 print(sys.getsizeof(0) * len(lista)) #87 comes in a list, so
```

→ 2800

```
1 # t multiplies the size of each item (arra.itemsize) by the t
2 print(nparr.itemsize * nparr.size)
```

→ 400

```
1 # numpy array while storing 100 elements is saving 2400 byte
2 import sys
3 import numpy as np
4
5 # Create a Python list containing integers from 0 to 99
6 py_list = list(range(100))
7
8 # Create a NumPy array containing integers from 0 to 99
9 np_array = np.arange(100)
10
11 # Calculate memory usage of the Python list
12 py_list_memory = sys.getsizeof(0) * len(py_list)
13
14 # Calculate memory usage of the NumPy array
15 np_array_memory = np_array.itemsize * np_array.size
16
17 # Calculate memory saved by NumPy array
18 memory_saved = py_list_memory - np_array_memory
```

```
19
20 print("Memory saved by using NumPy array:", memory_saved, "bytes")
21
```

```
→ Memory saved by using NumPy array: 2400 bytes
```

To calculate the memory saved by using a NumPy array instead of a Python list for machine learning data (numbers in the range of lacs), we can follow these steps:

1. Calculate the memory usage of both the Python list and the NumPy array for the given number of elements.
2. Convert the memory usage from bytes to kilobytes.
3. Calculate the total memory saved by using NumPy instead of Python lists.

```
1 import sys
2 import numpy as np
3
4 # Number of elements (in lacs)
5 num_elements = 100000
6
7 # Create a Python list containing integers from 0 to (num_elements)
8 py_list = list(range(num_elements))
9
10 # Create a NumPy array containing integers from 0 to (num_elements)
11 np_array = np.arange(num_elements)
12
13 # Calculate memory usage of the Python list (in bytes)
14 py_list_memory = sys.getsizeof(0) * len(py_list)
15
16 # Calculate memory usage of the NumPy array (in bytes)
17 np_array_memory = np_array.itemsize * np_array.size
18
19 # Convert memory usage from bytes to kilobytes
20 py_list_memory_kb = py_list_memory / 1024
21 np_array_memory_kb = np_array_memory / 1024
22
23 # Calculate total memory saved by using NumPy array (in kilobytes)
24 memory_saved_kb = (py_list_memory - np_array_memory) / 1024
25
```

```
26 print("Memory usage of Python list:", py_list_memory_kb, "KB")
27 print("Memory usage of NumPy array:", np_array_memory_kb, "K")
28 print("Total memory saved by using NumPy array:", memory_saved)
29
```

```
→ Memory usage of Python list: 2734.375 KB
    Memory usage of NumPy array: 390.625 KB
        Total memory saved by using NumPy array: 2343.75 KB
```

## Numpy arrays are faster in comparison to python list

```
1 import time # time module
2 x=range(100000)
3 y=range(100000)
4 start=time.time()
5 c=[x+y  for x,y in zip(x,y)]
6 end=time.time()
7 print(f'Total execution time is {end- start}')
```

```
→ Total execution time is 0.0598301887512207
```

```
1 import time # time module
2 a=np.arange(100000)
3 b=np.arange(100000)
4 start=time.time()
5 c=a + b
6 end=time.time()
7 print(f'Total execution time is {end- start}')
```

```
→ Total execution time is 0.0020363330841064453
```

```
1 #List Comprehension ([x + y for x, y in zip(x, y)]):
2
3 #Iterates over the elements of x and y simultaneously using
4 #For each pair of corresponding elements from x and y, it ca
5 #Creates a new list c containing the results of the element-
6
7 import time # time module
8 x=range(100000000000)
9 y=range(100000000000)
```

```
10 start=time.time()
11 c=[x+y  for x,y in zip(x,y)] ## Perform element-wise addition
12 end=time.time()
13 print(f'Total execution time is {end- start}')
```

1.  $c = a + b$ : Performs element-wise addition of the corresponding elements of arrays  $a$  and  $b$ , resulting in a new NumPy array
2. Each element of  $c$  is the sum of the corresponding elements in  $a$  and  $b$ .
3. this code efficiently performs element-wise addition of two NumPy arrays using vectorized operations, leveraging the optimized capabilities of NumPy for numerical computations.
4. The resulting array  $c$  will contain the sum of each pair of corresponding elements from arrays  $a$  and  $b$ .

```
1 import numpy as np
2 import time # time module
3 a=np.arange(100000000)
4 b=np.arange(100000000)
5 start=time.time()
6 c=a + b
7 end=time.time()
8 print(f'Total execution time is {end- start}')
```

→ Total execution time is 0.18782806396484375

three things about numpy array have been proved: 1, less memory,

2. faster execution time 3. code is convenient like  $c=a+b$  in comparison with element wise addition of python lists

```
1 import numpy as np
```

```
1 #indexing, slicing and Iteration
2 # if we want to retrieve a single item, we use indexing, use
3
```

```
4 arr12=np.arange(24)
```

```
5 arr12
```

```
→ array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23])
```

```
1 arr12=np.arange(24).reshape(12,2)
```

```
2 arr12
```

```
→ array([[ 0,  1],
       [ 2,  3],
       [ 4,  5],
       [ 6,  7],
       [ 8,  9],
       [10, 11],
       [12, 13],
       [14, 15],
       [16, 17],
       [18, 19],
       [20, 21],
       [22, 23]])
```

```
1 arr12=np.arange(24).reshape(3,8)
```

```
2 arr12
```

```
→ array([[ 0,  1,  2,  3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20, 21, 22, 23]])
```

```
1 arr12=np.arange(24).reshape(3,8) # or reshape(8, 3) or resh
```

```
2 arr12
```

```
→ array([[ 0,  1,  2,  3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20, 21, 22, 23]])
```

```
1 arr13=np.arange(23).reshape(23,1) # or reshape(8, 3) or res
```

```
2 arr13
```

```
→ array([[ 0],
       [ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
```

```
[ 9],  
[10],  
[11],  
[12],  
[13],  
[14],  
[15],  
[16],  
[17],  
[18],  
[19],  
[20],  
[21],  
[22]])
```

```
1 #np.arange(24)--creates a 1-dimensional NumPy array with ele  
2 #reshape(6, 4) reshapes the 1-dimensional array into a 2-dim  
3  
4 arr12=np.arange(24).reshape(6,4) # Create a NumPy array with  
5 arr12
```

```
→ array([[ 0,  1,  2,  3],  
       [ 4,  5,  6,  7],  
       [ 8,  9, 10, 11],  
       [12, 13, 14, 15],  
       [16, 17, 18, 19],  
       [20, 21, 22, 23]])
```

```
1 arr12[4:5,:]
```

```
→ array([[16, 17, 18, 19]])
```

```
1 arr12[:,1:2]
```

```
→ array([[ 1],  
       [ 5],  
       [ 9],  
       [13],  
       [17],  
       [21]])
```

```
1 arr12[2:4,1:3]
```

```
→ array([[ 9, 10],  
       [13, 14]])
```

```
1 arr12[0:3,1:3]
```

```
→ array([[ 1,  2],  
       [ 5,  6],  
       [ 9, 10]])
```

```
1 arr12
```

```
→ array([[ 0,  1,  2,  3],  
       [ 4,  5,  6,  7],  
       [ 8,  9, 10, 11],  
       [12, 13, 14, 15],  
       [16, 17, 18, 19],  
       [20, 21, 22, 23]])
```

```
1 arr12[0,0:2]
```

```
→ array([0, 1])
```

```
1 #arr12[4,:]  
2 arr12[4]
```

```
→ array([16, 17, 18, 19])
```

```
1 arr12[0:2,0:2]
```

```
→ array([[0, 1],  
       [4, 5]])
```

```
1 arr12[:,2]
```

```
→ array([ 2,  6, 10, 14, 18, 22])
```

```
1 #By using arr12[2], you are selecting the entire 3rd row of  
2 #This operation is a common way to access specific rows or columns  
3  
4 arr12[2] #Fetches the 3rd row (remember, Python uses zero-based indexing)
```

```
→ array([4, 5])
```

```
1 arr12[:,1] #Fetches all rows, but only column 1 (the second  
2 #gives you the second column as a 1D array  
3 #: means all rows  
4 #1 means column at index 1 (the second column)
```

```
→ array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23])
```

```
1 arr12[:,1:] #Fetches all rows, and from column index 1 onward
2 #This returns the second column but as a 2D array of shape (12,1)
3 #: means all rows
4 #1: means start from column 1 to the end (only column 1 in the result)
```

```
→ array([[ 1],
       [ 3],
       [ 5],
       [ 7],
       [ 9],
       [11],
       [13],
       [15],
       [17],
       [19],
       [21],
       [23]])
```

image.png

```
1 c=np.arange(24).reshape(12,2)
2 c
3
```

```
→ array([[ 0,  1],
       [ 2,  3],
       [ 4,  5],
       [ 6,  7],
       [ 8,  9],
       [10, 11],
       [12, 13],
       [14, 15],
       [16, 17],
       [18, 19],
       [20, 21],
       [22, 23]])
```

```
1 c[0][-1]
```

```
→ 1
```

```
1 c[:4]
```

```
→ array([[ 0,  1],
       [ 2,  3],
```

```
[4, 5],  
[6, 7]))
```

```
1 c[:,1:]
```

```
→ array([[ 1],  
         [ 3],  
         [ 5],  
         [ 7],  
         [ 9],  
         [11],  
         [13],  
         [15],  
         [17],  
         [19],  
         [21],  
         [23]])
```

```
1 c[:,1]
```

```
→ array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23])
```

```
1 arr12[:2] # returns first 2 rows of 2d array
```

```
→ array([[0, 1, 2, 3],  
         [4, 5, 6, 7]])
```

```
1 arr12[:1, :] #return 1st row as 2d slice
```

```
→ array([[0, 1]])
```

```
1 arr12[0, :] ##return 1st row as 1d slice
```

```
→ array([0, 1])
```

```
1 arr12[:2]# returns first 2 rows of 2d array
```

```
→ array([[0, 1],  
         [2, 3]])
```

```
1 arr12
```

```
→ array([[ 0,  1],  
         [ 2,  3],  
         [ 4,  5],  
         [ 6,  7],  
         [ 8,  9],
```

```
[10, 11],  
[12, 13],  
[14, 15],  
[16, 17],  
[18, 19],  
[20, 21],  
[22, 23]])
```

```
1 arr12[:1]
```

```
→ array([[0, 1, 2, 3]])
```

```
1 arr12[:1][0][-1]
```

```
→ 3
```

```
1 #arr12[:2] uses slicing to access the first 2 rows of the ar  
2 #The slice :2 means "from the start up to, but not including  
3 #In this context, it selects the rows at indices 0 and 1.  
4  
5 arr12[:2] # returns first 2 rows of 2d array
```

```
→ array([[0, 1, 2, 3],  
         [4, 5, 6, 7]])
```

```
1 arr12
```

```
→ array([[ 0,  1],  
        [ 2,  3],  
        [ 4,  5],  
        [ 6,  7],  
        [ 8,  9],  
        [10, 11],  
        [12, 13],  
        [14, 15],  
        [16, 17],  
        [18, 19],  
        [20, 21],  
        [22, 23]])
```

```
1 arr12[:2,1]
```

```
→ array([1, 3])
```

```
1 arr12[0:2,1]
```

```
→ array([1, 3])
```

```
1 arr12[:,2:4]
```

```
→ array([[ 2,  3],  
         [ 6,  7],  
         [10, 11],  
         [14, 15],  
         [18, 19],  
         [22, 23]])
```

```
1 p=np.arange(24).reshape(6,4)
```

```
2 p
```

```
→ array([[ 0,  1,  2,  3],  
         [ 4,  5,  6,  7],  
         [ 8,  9, 10, 11],  
         [12, 13, 14, 15],  
         [16, 17, 18, 19],  
         [20, 21, 22, 23]])
```

```
1 #The notation arr12[:, 2] is used to access the entire 3rd column
```

```
2 #The slice : means "all rows", and 2 specifies the 3rd column
```

```
3
```

```
4 arr12[:,2]
```

```
→ array([ 2, 10, 18])
```

```
1 arr12=np.arange(24).reshape(6,4)
```

```
2 arr12
```

```
→ array([[ 0,  1,  2,  3],  
         [ 4,  5,  6,  7],  
         [ 8,  9, 10, 11],  
         [12, 13, 14, 15],  
         [16, 17, 18, 19],  
         [20, 21, 22, 23]])
```

```
1 #Access 2nd and 3rd columns
```

```
2 arr12[:,2:4]
```

```
→ array([[ 2,  3],  
         [ 6,  7],  
         [10, 11],  
         [14, 15],
```

```
[18, 19],  
[22, 23]])
```

```
1 e=np.empty((4,))  
2 e
```

```
→ array([0.000000e+000, 1.2780773e-311, 1.2780773e-311, 1.2780773e-311])
```

```
1 np.random.random((2,2))
```

```
→ array([[0.45247041, 0.25926956],  
         [0.03223025, 0.45217967]])
```

```
1 #exract 9 10  
2 #          13 14  
3  
4 arr12[2:4:,1:3]
```

```
→ array([[ 9, 10],  
         [13, 14]])
```

```
1 arr12
```

```
→ array([[ 0,  1,  2,  3],  
         [ 4,  5,  6,  7],  
         [ 8,  9, 10, 11],  
         [12, 13, 14, 15],  
         [16, 17, 18, 19],  
         [20, 21, 22, 23]])
```

```
1 #exract 18 19  
2 #          22 23  
3 arr12[4:,2:]
```

```
→ array([[18, 19],  
         [22, 23]])
```

```
1 ar11=np.array([1,2,3,4,5,6])  
2 ar11
```

```
→ array([1, 2, 3, 4, 5, 6])
```

```
1 ar11[3]
```

```
1 #slice notation Slices the array to extract the elements at
2 ar11[2:4] ## Slice the array to get elements at positions 2
```

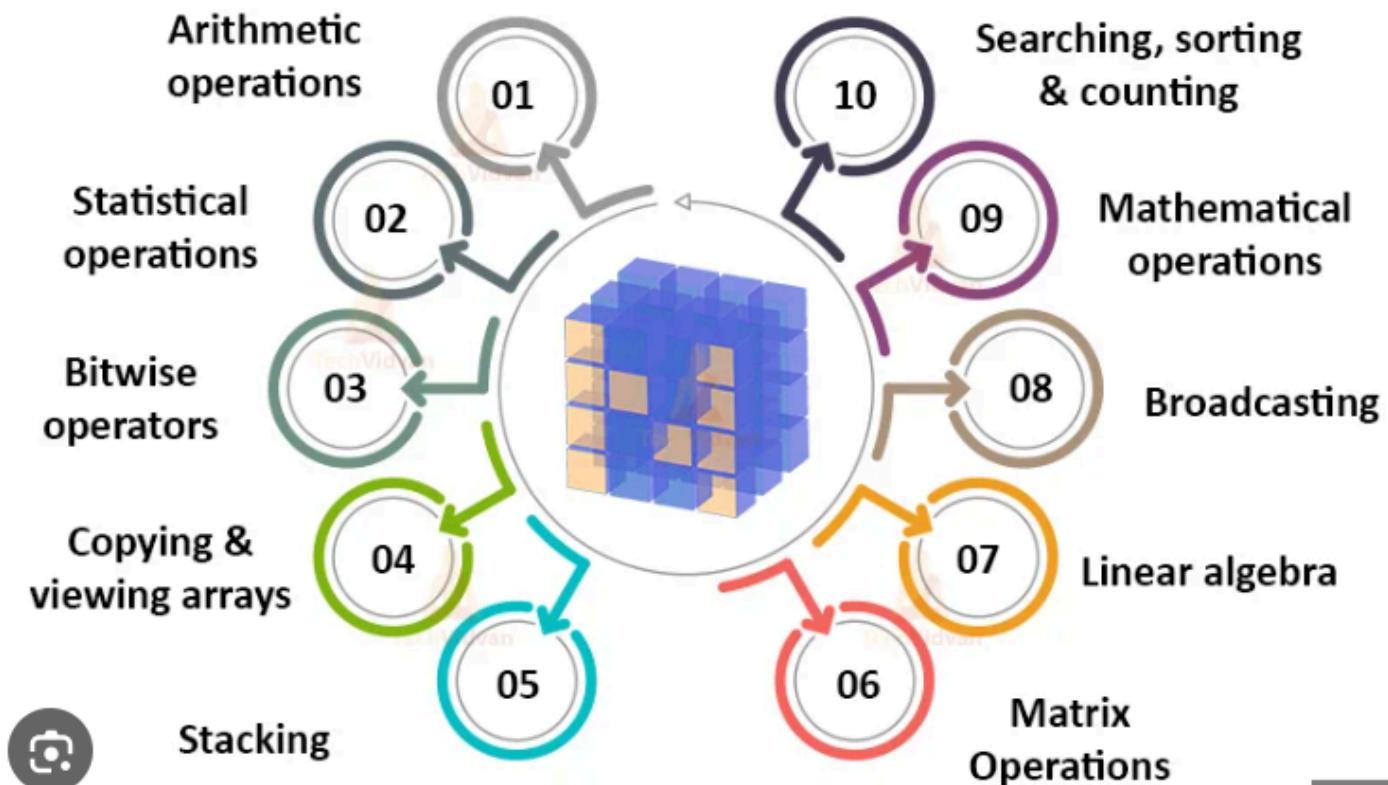
```
→ array([3, 4])
```

```
1 arr12
```

```
→ array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19],
       [20, 21, 22, 23]])
```

▼ Notebook: 16) Numpy\_Functions.ipynb

# Uses of NumPy



## ▼ 1. Basics of NumPy

Installing and Importing NumPy

```
1 # Install NumPy (if not already installed)
2 !pip install numpy
```

→ Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)

```
1 # Import the library
2 import numpy as np
```

1 Start coding or generate with AI.

## ▼ Creating Arrays

An array is a grid of value and it contain information about raw data how to locate an element, and how to interpret an element.

4	5	9
---	---	---

8	2	9
8	7	5
4	1	2
9	1	9

1

```
my_np.array([4,5,9])
```

2

```
my_np.array([[8,2,9],
             [8,7,5],
             [4,1,2],
             [9,1,9]])
```

3

8	2	9
8	7	5
4	1	2
9	1	9

9		
8		
5		
4		
5		

4	6	2
1	2	5

```
my_np.array([[8,2,9],
             [8,7,5],[4,1,2],[9,1,9]],
            [[4,5,9],[4,2,8],
             [2,1,9],[4,6,2]],
            [[6,8,8],[2,4,5],
             [7,8,4],[1,2,5]]])
```

```
1 # Difference between array and list
2 list = [1, 2, 3, 4, 5]
3 print(list)
4 print(type(list))
5
```

```
→ [1, 2, 3, 4, 5]
<class 'list'>
```

```
1 type(list)
```

```
→ list
```

```
1 list
```

```
→ [1, 2, 3, 4, 5]
```

```
1 array = np.array([1, 2, 3, 4, 5])
2 print(array)
3 print(type(array))
```

```
→ [1 2 3 4 5]
<class 'numpy.ndarray'>
```

```
1 # Create a 1D array from a list
2 arr1 = np.array([1, 2, 3, 4, 5, 6])
3 print("1D Array:", arr1)
```

```
→ 1D Array: [1 2 3 4 5 6]
```

```
1 type(arr1)
```

```
→ numpy.ndarray
```

```
1 # Create a 2D array (matrix)
2 arr2 = np.array([[1, 2, 3], [4, 5, 6]])
3 print("2D Array:\n", arr2)
```

```
→ 2D Array:
[[1 2 3]
 [4 5 6]]
```

```
1 a = np.array([1, 2, 3, 4, 5])
```

```
1 a
```

```
→ array([1, 2, 3, 4, 5])
```

```
1 # Create a 3D array (matrix)
```

```
2 arr3 = np.array([[1, 2, 3], [4, 5, 6],[7,8,9]])
```

```
3 print("3D Array:\n", arr3)
```

```
→ 3D Array:
```

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

## ▼ Array attributes

This section covers the **ndim**, **shape**, **size**, and **dtype** attributes of an array.

```
1 arr2.ndim # tell the number of dimensions like 1d,2d,.....
```

```
→ 2
```

```
1 print(arr2.ndim)
```

```
→ 2
```

```
1 arr3.shape #
```

```
→ (1, 3, 3)
```

```
1 arr3.size # total number of elements in array is contained in
```

```
→ 9
```

```
1 arr3.dtype
```

```
→ dtype('int64')
```

```
1 # Arrays are typically “homogeneous”, meaning that they contain
```

```
2 # The data type is recorded in the dtype attribute.
```

```
3 arr3.dtype  
4 a.dtype
```

```
→ dtype('int64')
```

## ✓ How to create a basic array

This section covers `np.zeros()`, `np.ones()`, `np.empty()`, `np.arange()`, `np.linspace()`

```
1 a = np.zeros(4)  
2 print(a)
```

```
→ [0. 0. 0. 0.]
```

```
1 # Create an array of zeros (shape = 3x3)  
2 zeros = np.zeros((3, 3))  
3 print("Zeros:\n", zeros)
```

```
→ Zeros:  
[[0. 0. 0.]  
[0. 0. 0.]  
[0. 0. 0.]]
```

```
1 # Create an array of ones (shape = 2x4)  
2 ones = np.ones((2, 4))  
3 print("Ones:\n", ones)
```

```
→ Ones:  
[[1. 1. 1. 1.]  
[1. 1. 1. 1.]]
```

```
1 # Create an empty array with 2 elements  
2 a= np.empty(5)
```

```
1 a
```

```
→ array([2.49044248e-316, 0.00000000e+000, 7.21908161e+159, 4.63461826e+228,  
6.32404027e-322])
```

```
1 np.arange(5)
```

```
→ array([0, 1, 2, 3, 4])
```

```
1 np.arange(2, 9, 2)
```

```
→ array([2, 4, 6, 8])
```

```
1 a
```

```
→ array([2.49044248e-316, 0.00000000e+000, 7.21908161e+159, 4.63461826e+228,  
       6.32404027e-322])
```

```
1 a = np.arange(6)  
2 print(a)
```

```
→ [0 1 2 3 4 5]
```

```
1 # reshape the array  
2 b = a.reshape(3, 2)  
3 b  
4
```

```
→ array([[0, 1],  
        [2, 3],  
        [4, 5]])
```

```
1 x = np.reshape(a, (1, 4), order =)  
2
```

```
→     File "<ipython-input-43-a53f1083e4d8>", line 1  
         x = np.reshape(a, (1, 4), order =)  
               ^
```

```
SyntaxError: invalid syntax
```

```
1 # Create a range of values (start, end, step)  
2 range_arr = np.arange(0, 10, 2)  
3 print("Range Array:", range_arr)
```

```
→ Range Array: [0 2 4 6 8]
```

```
1 # Create equally spaced values (start, end, number_of_values)  
2 linspace = np.linspace(0, 20, 2)
```

```
3 print("Linspace:", linspace)
```

```
→ Linspace: [ 0. 20.]
```

```
1 x = np.ones(3, dtype=np.int64)
```

```
2 x
```

```
→ array([1, 1, 1])
```

## ✓ Adding, removing, and sorting elements

```
1 arr = np.array([2, 1, 5, 3, 7, 4, 6, 8])
```

```
2 np.sort(arr)
```

```
3
```

```
4
```

```
→ array([1, 2, 3, 4, 5, 6, 7, 8])
```

```
1 a = np.array([1, 2, 3, 4])
```

```
2 b = np.array([5, 6, 7, 8])
```

```
3 np.concatenate((a, b))
```

```
4
```

```
→ array([1, 2, 3, 4, 5, 6, 7, 8])
```

```
1 x = np.array([[1, 2], [3, 4]])
```

```
2 y = np.array([[5, 6]])
```

```
3 np.concatenate((x, y), axis=0)
```

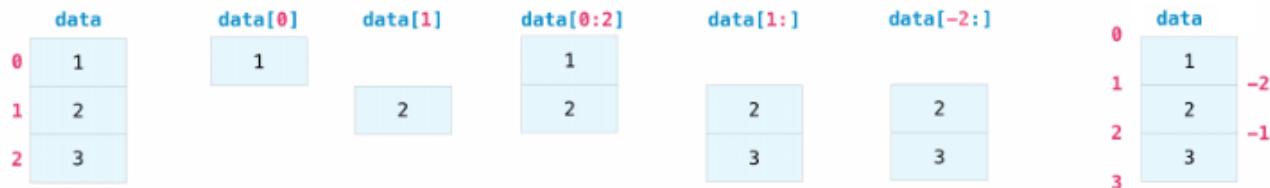
```
4
```

```
→ array([[1, 2],  
         [3, 4],  
         [5, 6]])
```

## ✓ Indexing and slicing

You can index and slice NumPy arrays in the same ways you can slice Python lists.

You can visualize it this way:



```
1 data = np.array([1, 2, 3, 5, 8, 9, 10])  
2  
3 data[1]
```

```
→ np.int64(2)
```

```
1 data[0:2]  
2
```

```
→ array([1, 2])
```

```
1 data[1:]  
2
```

```
→ array([ 2,  3,  5,  8,  9, 10])
```

```
1 data[-2:]
```

```
→ array([ 9, 10])
```

```
1 # indexing with condition  
2 a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
```

```
1 print(a[a < 5])
```

```
→ [1 2 3 4]
```

```
1 five_up = (a >= 5)
2 print(a[five_up])
```

→ [ 5 6 7 8 9 10 11 12]

```
1 divisible_by_2 = a[a%2==0]
2 print(divisible_by_2)
```

→ [ 2 4 6 8 10 12]

```
1 five_up = (a > 5) | (a == 5)
2 print(five_up)
```

→ [[False False False False]
 [ True True True True]
 [ True True True True]]

1 Start coding or generate with AI.

## ✓ How to create an array from existing data

This section covers **slicing and indexing**, **np.vstack()**, **np.hstack()**, **np.hsplit()**, **.view()**, **copy()**

```
1 arr.view()
```

→ array([2, 1, 5, 3, 7, 4, 6, 8])

```
1 a1 = np.array([[1, 1],
2                 [2, 2]])
3
4 a2 = np.array([[3, 3],
5                 [4, 4]])
```

```
1 np.vstack((a1, a2))
```

→ array([[1, 1],
 [2, 2],
 [3, 3],
 [4, 4]])

```
1 np.hstack((a1, a2))
```

```
→ array([[1, 1, 3, 3],  
        [2, 2, 4, 4]])
```

```
1 x = np.arange(1, 25).reshape(2, 12)
```

```
2 x
```

```
→ array([[ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12],  
        [13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]])
```

```
1 # If you want to split this array into three equally shaped
```

```
2 np.hsplit(x, 3)
```

```
→ [array([[ 1,  2,  3,  4],  
          [13, 14, 15, 16]]),  
  array([[ 5,  6,  7,  8],  
          [17, 18, 19, 20]]),  
  array([[ 9, 10, 11, 12],  
          [21, 22, 23, 24]])]
```

```
1 # If you want to split your array after the third and fourth
```

```
2 np.hsplit(x, (3, 4))
```

```
→ [array([[ 1,  2,  3],  
          [13, 14, 15]]),  
  array([[ 4],  
          [16]]),  
  array([[ 5,  6,  7,  8,  9, 10, 11, 12],  
          [17, 18, 19, 20, 21, 22, 23, 24]])]
```

```
1 x = x[0, :]
```

```
2 x
```

```
→ array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

```
1 # copy
```

```
2 y = x.copy()
```

## ✓ Basic array operations and the Use of Ramdom function

```
1 # Create Numpy Array with Random numbers
2 a = np.random.rand(4)
3 a
```

```
→ array([0.8660102 , 0.34744172, 0.62012667, 0.90615606])
```

```
1 b = np.random.rand(5)
2 b
```

```
→ array([0.53267847, 0.77861484, 0.74366546, 0.73076257, 0.14126357])
```

```
1 b= np.random.rand(4)
2 b
```

```
→ array([0.16996912, 0.23082763, 0.24779412, 0.47545867])
```

```
1 import numpy as np
2
3 # Generate an array with 5 rows and 2 columns containing random numbers
4 data = np.random.rand(5, 2)
5 print(data)
6
```

```
→ [[0.71315951 0.52111715]
 [0.50913142 0.19172324]
 [0.17026794 0.55856996]
 [0.20749783 0.96736684]
 [0.88244466 0.89888992]]
```

```
1 import numpy as np
2
3 # Generate a 5x2 array with random integers from 0 to 9
4 data = np.random.randint(0, 10, size=(5, 2))
5 print(data)
6
```

```
→ [[0 8]
 [3 4]
 [9 4]
 [3 2]
 [7 6]]
```

```
1 c = np.add(a,b)
```

```
2 c
```

```
→ array([1.03597933, 0.57826935, 0.86792078, 1.38161474])
```

```
1 c = a+b
```

```
2 c
```

```
→ array([1.03597933, 0.57826935, 0.86792078, 1.38161474])
```

```
1 c = np.add(a,b)
2 d = np.subtract(a,b)
3 e = np.multiply(a,b)
4 f = np.divide(a,b)
5 g = np.sum(a)
6 h = np.min(a)
7 i = np.max(a)
8 j = np.mod(a,b)
9 k = np.average(a)
10 l = np.median(a)
11 m = np.std(a)
12 n = np.var(a)
13 o = np.sort(a)
14 p = np.argsort(a)
15 q = np.sin(a)
16 r = np.cos(a)
17 s = np.tan(a)
18 t = np.arcsin(a)
19 u = np.arccos(a)
20 v = np.arctan(a)
21 w = np.log(a)
22 x = np.log2(a)
23 y = np.log10(a)
24 z = np.exp(a)
25
```

```
1 print(o)
```

```
2 g
```

```
→ [0.34744172 0.62012667 0.8660102 0.90615606]  
np.float64(2.7397346492108485)
```

```
1 data = np.array([1.0, 2.0])  
2 data * 1.6
```

```
→ array([1.6, 3.2])
```

```
1 arr2
```

```
→ array([[1, 2, 3],  
         [4, 5, 6]])
```

```
1 sum = np.sum([arr2, arr2])  
2 sum
```

```
→ np.int64(42)
```

```
1 sum = np.sum([arr1, arr2])  
2 sum
```

```
→ -----  
ValueError Traceback (most recent call last)  
<ipython-input-92-ec978590421e> in <cell line: 0>()  
----> 1 sum = np.sum([arr1, arr2])  
      2 sum
```

◆ 1 frames ◆

```
/usr/local/lib/python3.11/dist-packages/numpy/_core/fromnumeric.py in  
_wrapreduction(obj, ufunc, method, axis, dtype, out, **kwargs)  
    84             return reduction(axis=axis, out=out, **passkwargs)  
    85  
---> 86     return ufunc.reduce(obj, axis, dtype, out, **passkwargs)  
    87  
    88
```

**ValueError:** setting an array element with a sequence. The requested array has an inhomogeneous shape after 1 dimensions. The detected shape was (2,) + inhomogeneous part.

```
1 Start coding or generate with AI.
```

## ▼ Iterating

It is used to break the each element.

```
1 for i in arr1:  
2   print(i)  
3
```

```
→ 1  
2  
3  
4  
5  
6
```

```
1 arr2
```

```
→ array([[1, 2, 3],  
         [4, 5, 6]])
```

```
1 for i in arr2:  
2   for j in i:  
3     print(j)  
4
```

```
→ 1  
2  
3  
4  
5  
6
```

```
1 # use of nditer()  
2 for x in np.nditer(arr2):  
3   print(x)
```

```
→ 1  
2  
3  
4  
5  
6
```

```
1 # use of ndenumerate().....it give the index value with d  
2 for i, d in np.ndenumerate(arr2):  
3   print(i, d)
```

```
→ (0, 0) 1  
  (0, 1) 2
```

```
(0, 2) 3  
(1, 0) 4  
(1, 1) 5  
(1, 2) 6
```

1 Start coding or generate with AI.

```
1 import sys  
2 sys.getsizeof(a)
```

→ 144

```
1 np.ravel(arr3)
```

→ array([1, 2, 3, 4, 5, 6, 7, 8, 9])

```
1 h
```

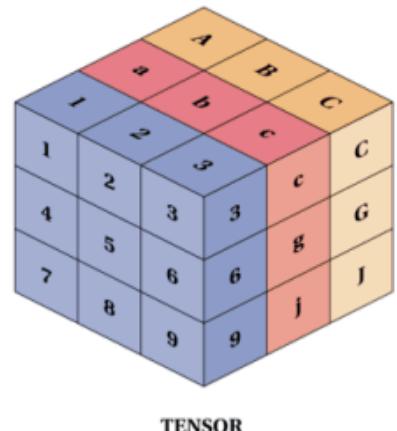
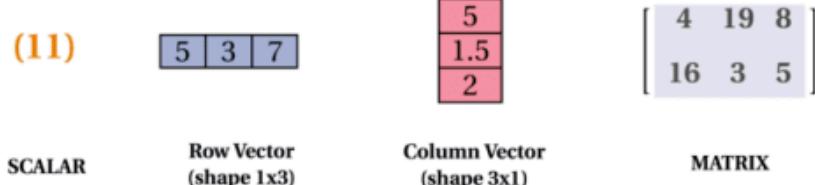
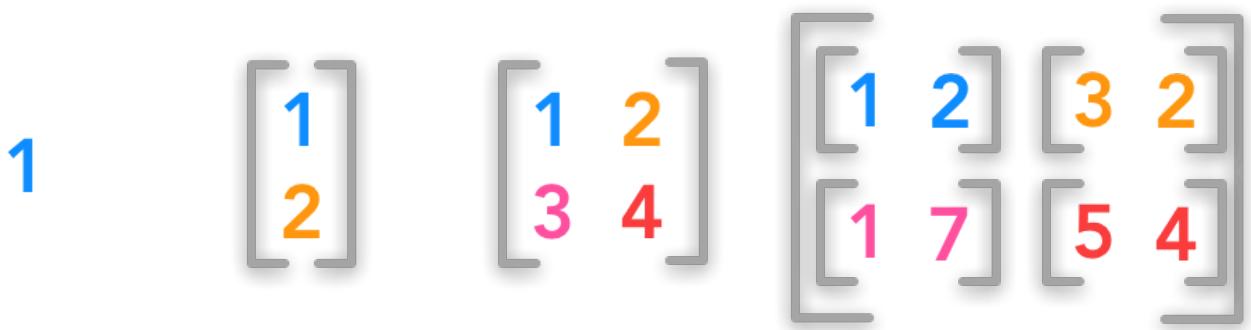
→ array([1, 2, 3, 4, 5, 6, 7, 8, 9])

1 Start coding or generate with AI.

## ▼ Linear Algebra

Data is stored in vectors, matrices, and tensors. It deals with algebra and mathematics. Linear algebra is used in preprocessing and transformation of data. It is the pillar of ML.

# Scalar Vector Matrix Tensor



## ▼ Basic functions of linear Algebra

```
1 #transpose means to convert the row into column ( You can no
2 arr
```

```
→ array([2, 1, 5, 3, 7, 4, 6, 8])
```

```
1 x = arr.T
2 x
```

```
→ array([2, 1, 5, 3, 7, 4, 6, 8])
```

```
1 a = np.matrix([1,2,3,4,5,6,7,8,9])
```

```
1 x = a.T
```

```
2 x
```

```
→ matrix([[1],  
          [2],  
          [3],  
          [4],  
          [5],  
          [6],  
          [7],  
          [8],  
          [9]])
```

```
1 b = a.copy()
```

```
2 b
```

```
→ matrix([[1, 2, 3, 4, 5, 6, 7, 8, 9]])
```

```
1 b.shape
```

```
→ (1, 9)
```

```
1 c = np.dot( arr1, arr1) # we will get the answer in scalar
```

```
2 c
```

```
→ np.int64(91)
```

```
1 d = np.diag(arr)
```

```
2 d
```

```
→ array([[2, 0, 0, 0, 0, 0, 0, 0],  
          [0, 1, 0, 0, 0, 0, 0, 0],  
          [0, 0, 5, 0, 0, 0, 0, 0],  
          [0, 0, 0, 3, 0, 0, 0, 0],  
          [0, 0, 0, 0, 7, 0, 0, 0],  
          [0, 0, 0, 0, 0, 4, 0, 0],  
          [0, 0, 0, 0, 0, 0, 6, 0],  
          [0, 0, 0, 0, 0, 0, 0, 8]])
```

```
1 s = np.identity(6)
```

```
2 s
```

```
→ array([[1., 0., 0., 0., 0., 0.],  
          [0., 1., 0., 0., 0., 0.],  
          [0., 0., 1., 0., 0., 0.],  
          [0., 0., 0., 1., 0., 0.],  
          [0., 0., 0., 0., 1., 0.],  
          [0., 0., 0., 0., 0., 1.]])
```

```
[0., 0., 0., 0., 1., 0.],  
[0., 0., 0., 0., 0., 1.]])
```

1 Start coding or generate with AI.

# Solving a Matrix Equation

$$AX = B$$

$$A = \begin{bmatrix} -1 & -11 & -3 \\ 1 & 1 & 0 \\ 2 & 5 & 1 \end{bmatrix} \quad B = \begin{bmatrix} -37 \\ -1 \\ 10 \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

1 a

```
→ matrix([[1, 2, 3, 4, 5, 6, 7, 8, 9]])
```

1 b

```
→ matrix([[1, 2, 3, 4, 5, 6, 7, 8, 9]])
```

```
1 # Solving equation by Cramer's rule  
2 x = np.linalg.solve(a,b)  
3 x
```

```
→ -----  
LinAlgError Traceback (most recent call last)  
<ipython-input-119-1a2266a8361b> in <cell line: 0>()  
      1 # Solving equation by Cramer's rule  
----> 2 x = np.linalg.solve(a,b)  
      3 x  
  
-----  
          1 frames -----  
/usr/local/lib/python3.11/dist-packages/numpy/linalg/_linalg.py in  
_assert_stacked_square(*arrays)  
    200     m, n = a.shape[-2:]  
    201     if m != n:  
--> 202         raise LinAlgError('Last 2 dimensions of the array must be square')  
    203  
 204 def _assert_finite(*arrays):  
  
LinAlgError: Last 2 dimensions of the array must be square
```

```
1 import numpy as np  
2  
3 # Define a coefficient matrix (A) and a right-hand side vector (b)  
4 A = np.array([[3, 1], [1, 2]]) # 2x2 matrix  
5 b = np.array([9, 8])           # 2-element vector  
6  
7 # Solve the linear system: A * x = b  
8 x = np.linalg.solve(A, b)       # x holds the solution of the system  
9  
10 print("Solution of A*x=b:", x) # Output the solution  
11
```

```
→ Solution of A*x=b: [2. 3.]
```

```
1 import numpy as np  
2  
3 # Define a square matrix  
4 matrix = np.array([[4, 7], [2, 6]])  
5  
6 # Compute its inverse using np.linalg.inv  
7 matrix_inv = np.linalg.inv(matrix) # Get the inverse of the matrix  
8  
9 print("Matrix Inverse:\n", matrix_inv) # Display the inverse matrix  
10
```

→ Matrix Inverse:

```
[[ 0.6 -0.7]
 [-0.2  0.4]]
```

```
1 import numpy as np
2
3 # Define a square matrix
4 matrix = np.array([[4, 7], [2, 6]])
5
6 # Calculate the determinant using np.linalg.det
7 det = np.linalg.det(matrix) # Compute the determinant
8
9 print("Determinant of the matrix:", det) # Output the deter
10
```

→ Determinant of the matrix: 10.000000000000002

```
1 import numpy as np
2
3 # Define a vector or matrix
4 vector = np.array([3, 4])
5
6 # Compute the Euclidean norm (L2 norm) of the vector using n
7 norm_value = np.linalg.norm(vector)
8
9 print("Euclidean Norm:", norm_value) # Expected output: 5,
10
```

→ Euclidean Norm: 5.0

```
1 import numpy as np
2
3 # Define a square matrix
4 matrix = np.array([[12, -51, 4],
5                   [6, 167, -68],
6                   [-4, 24, -41]])
7
8 # Compute QR decomposition of the matrix
9 Q, R = np.linalg.qr(matrix) # Decomposes matrix into orthog
10
```

```
11 print("Matrix Q (orthogonal):\n", Q)
12 print("Matrix R (upper triangular):\n", R)
13
```

```
→ Matrix Q (orthogonal):
 [[-0.85714286  0.39428571  0.33142857]
 [-0.42857143 -0.90285714 -0.03428571]
 [ 0.28571429 -0.17142857  0.94285714]]
Matrix R (upper triangular):
 [[ -14.   -21.    14.]
 [   0.  -175.    70.]
 [   0.     0.   -35.]]
```

```
1 import numpy as np
2
3 # Define a square matrix
4 matrix = np.array([[5, 4], [1, 2]])
5
6 # Compute eigenvalues and eigenvectors
7 eigenvalues, eigenvectors = np.linalg.eig(matrix) # Perform
8
9 print("Eigenvalues:", eigenvalues)                 # Output the eig
10 print("Eigenvectors:\n", eigenvectors)           # Output the e
11
```

```
→ Eigenvalues: [6. 1.]
Eigenvectors:
 [[ 0.9701425 -0.70710678]
 [ 0.24253563  0.70710678]]
```

```
1 np.linalg.
```

```
1 Start coding or generate with AI.
```

## ✓ Notebook: 17) 12-13-Pandas Intro.ipynb

```
1 import pandas as pd
2 import numpy as np
```

1. Pandas is the open source python library data analysis. Its a powerful library.

2. Python has great for manipulating data but pandas make it more powerful. Pandas is very fast and quick because it is based on spilled on the top of numpy. It has two data structures: series and data frames.
3. Data frames will be heavily used.
4. Series can be thought of as one dimensional array object.
5. It is one dimensional array or list or column in a table but it can have labels.
6. Labelled arrays or labelled columns or vectors are sort of one dimensional indexed array.
7. In addition to index, each element can have a name. so each element can be accessed either by index or name
8. Data frame is powerful data structure and comes in pandas package.
9. Data frames are 2 dimensional labelled data structure with columns of different types. They are 2 dimensional in a grid like format (like 2 dimensional array or Database tables or Excel sheet ).
10. They have labelled columns and rows and data can of different types. Data frames consists of components: 1. Data itself, index and the column

```
1 import numpy as np  
2 a=np.array([('ihsan',20,'lahore'),('femala',21,'shabe qadar'  
3 print(a)
```

```
→ [['ihsan' '20' 'lahore']  
 ['femala' '21' 'shabe qadar']  
 ['engin' '21' 'swabi']]
```

```
1 import pandas as pd
```

```
1 series=pd.Series(['atif',80,'Pehwar'])  
2 series
```

```
→ 0      atif  
 1      80  
 2    Pehwar  
dtype: object
```

```
1 series1=pd.Series(['ali',37,'Peshawar',5.5]) # create a ser  
2 series1
```

```
→ 0      ali  
 1      37  
 2    Peshawar  
 3      5.5  
dtype: object
```

```
1 series1 # series is one dimensional object array
```

```
→ 0      ali
  1      37
  2  Peshawar
  3      5.5
dtype: object
```

```
1 type(series1)
```

```
→ pandas.core.series.Series
```

```
1 ser1=pd.Series(['ali',37,'Peshawar',5.5] , index=['name','ag
2 ser1
```

```
→ name      ali
  age      37
  city    Peshawar
  hgt      5.5
dtype: object
```

```
1 #ser1['name']
2 ser1.iloc[0]
```

```
→ 'ali'
```

```
1 print(ser1[0])# elements of series object canbe accessed by
```

```
→ ali
```

```
1 ser1['name'] #ser1[1]      #ser1['age'] or ser1[1]
```

```
→ 'ali'
```

```
1 my_series = pd.Series({'London':10,'Tripoli':100,'Cairo':10}
```

```
1 my_series      # dictionary do not care about the order, we ca
```

```
→ London      10
  Tripoli    100
  Cairo      10
dtype: int64
```

```
1 my_series['Tripoli']
```

```
→ 100
```

```
1 my_series[my_series > 1] # filtering the dictionary
```

```
→ London    10  
  Tripoli   100  
  Cairo     10  
  dtype: int64
```

```
1 my_series[my_series == 10] # filtering the dictionary
```

```
→ London    10  
  Cairo     10  
  dtype: int64
```

```
1 my_series[my_series == 100] # filtering the dictionary
```

```
→ Tripoli   100  
  dtype: int64
```

```
1 Start coding or generate with AI.
```

## ▼ Notebook: 18) 13-Dataframes.ipynb

```
1 import pandas as pd
```

```
1 Two-dimensional size-mutable, potentially heterogeneous tabu  
2 structure with labeled axes (rows and columns)
```

```
1 import numpy as np
```

```
1 arr = np.array([[1, 2, 3], [4, 5, 6]]) #2 d numpy array  
2 arr
```

```
→ array([[1, 2, 3],  
         [4, 5, 6]])
```

```
1 arr = np.array([(1, 2, 3), (4, 5, 6)]) #2 d numpy array  
2 arr
```

→ array([[1, 2, 3],  
 [4, 5, 6]])

```
1 arr = np.array([[1, 2, 3], [4, 5, 6]]) #2 d numpy array  
2  
3 df = pd.DataFrame(arr) # create a data frame from the numpy  
4 # Pandas automatically gives them index  
5  
6 #df = pd.DataFrame(arr),columns=['col 1','col 2','col 3']
```

```
1 df           # first array or vector as 1st row in data frame,  
2  
3 df[2]
```

→

	0	1	2
0	1	2	3
1	4	5	6

```
1 df[2]
```

→

	0	3
0	1	6

Name: 2, dtype: int32

```
1 df.columns
```

→ RangeIndex(start=0, stop=3, step=1)

```
1 arr = np.array([[1, 2, 3], [4, 5, 6]]) #2 d numpy array  
2 df = pd.DataFrame(arr,columns=['col1','col2','col3'],index=[  
3 df
```

→

	col1	col2	col3
row1	1	2	3
row2	4	5	6

```
1 df['col3']# access the elements by using indices i.e access t
```

```
1 row1      3  
2 row2      6  
3 Name: col3, dtype: int32
```

```
1 df.columns # gives the names of columns in the form of range
```

```
1 Index(['col1', 'col2', 'col3'], dtype='object')
```

```
1 df['col2'] # columns can be labelled manually
```

```
1 row1      2  
2 row2      5  
3 Name: col2, dtype: int32
```

```
1 s1 = pd.Series(np.random.randn(5),index=['A','B','C','D','E'])
```

```
1 s1
```

```
1 A    -0.015466  
2 B    -0.909517  
3 C    -1.556067  
4 D    -0.275656  
5 E    0.229022  
6 dtype: float64
```

```
1 s2 = pd.Series(np.random.randn(5),index=['A','B','C','D','E'])
```

```
2 s2
```

```
1 A    0.465308  
2 B    -0.914711  
3 C    1.690892  
4 D    -0.703051  
5 E    0.458508  
6 dtype: float64
```

```
1 s2
```

```
1 A    0.483540  
2 B    0.742108  
3 C    -0.071070  
4 D    1.591000  
5 E    0.105240  
6 dtype: float64
```

```
1 #axis : {0/'index', 1/'columns'}, default 0 The axis to con  
2 df = pd.concat([s1, s2],axis=1)
```

```
3 df
```

	0	1
A	-0.015466	0.465308
B	-0.909517	-0.914711
C	-1.556067	1.690892
D	-0.275656	-0.703051
E	0.229022	0.458508

```
1 #axis : {0/'index', 1/'columns'}, default 0  The axis to con  
2 df = pd.concat([s1, s2],axis=1).reset_index(inplace=True)  
3 df
```

```
1 df
```

```
1 df[1]['A']
```

```
→ 1.4044485094500208
```

```
1 df[1]['E']
```

```
→ -0.15104872556876808
```

```
1 dp=s2.to_frame()  
2 print(dp)  
3 type(dp)
```

```
→          0  
A  1.404449  
B -1.133963  
C -1.873227  
D  0.448066  
E -0.151049  
pandas.core.frame.DataFrame
```

```
1 Start coding or generate with AI.
```

```
1 Start coding or generate with AI.
```

```
1 my_df = pd.DataFrame(np.random.randn(25).reshape((5,5)),index
```

```
1 my_df
```

	c1	c2	c3	c4	c5
a	0.582299	-0.465746	-0.706074	0.919884	-0.208221
b	1.109036	-0.556409	0.859343	-0.040539	0.777285
c	-0.321558	0.827996	-0.252036	1.176383	1.260723
d	-0.649170	-0.001465	1.409984	0.284063	0.805140
e	-0.356754	-1.673552	0.052488	-0.369870	0.406328

```
1 Drop specified labels from rows or columns.
```

```
2
```

```
3 Remove rows or columns by specifying label names and corresponding axis, or by specifying directly index or column names.
```

```
1 my_df.drop('c2',axis=1,inplace=False)
```

	c1	c3	c4	c5
a	0.582299	-0.706074	0.919884	-0.208221
b	1.109036	0.859343	-0.040539	0.777285
c	-0.321558	-0.252036	1.176383	1.260723
d	-0.649170	1.409984	0.284063	0.805140
e	-0.356754	0.052488	-0.369870	0.406328

```
1 my_df
```

	c1	c2	c3	c4	c5
a	0.582299	-0.465746	-0.706074	0.919884	-0.208221
b	1.109036	-0.556409	0.859343	-0.040539	0.777285
c	-0.321558	0.827996	-0.252036	1.176383	1.260723
d	-0.649170	-0.001465	1.409984	0.284063	0.805140
e	-0.356754	-1.673552	0.052488	-0.369870	0.406328

```
1 my_df.drop('c2',axis=1,inplace=True)
```

```
1 my_df
```

```
→
```

	c1	c3	c4	c5
a	0.582299	-0.706074	0.919884	-0.208221
b	1.109036	0.859343	-0.040539	0.777285
c	-0.321558	-0.252036	1.176383	1.260723
d	-0.649170	1.409984	0.284063	0.805140
e	-0.356754	0.052488	-0.369870	0.406328

```
1 my_df.columns = ['col1', 'col2', 'col3', 'col4']  
2 my_df.index=[ ]
```

```
1 my_df
```

```
→
```

	col1	col2	col3	col4
a	0.582299	-0.706074	0.919884	-0.208221
b	1.109036	0.859343	-0.040539	0.777285
c	-0.321558	-0.252036	1.176383	1.260723
d	-0.649170	1.409984	0.284063	0.805140
e	-0.356754	0.052488	-0.369870	0.406328

```
1 my_df.loc['a']['col2'] # accessing elements of dataframe tho
```

```
→ -0.7060742187474643
```

```
1 my_df.iloc[0][1] # integer based indexing to access elements  
2  
3 #Purely integer-location based indexing for selection by pos
```

```
→ -0.7060742187474643
```

```
1 my_df.iloc[0,1]
```

```
→ -0.7060742187474643
```

```
1 my_df['col1']['a']
```

→ 0.5822986346381396

```
1 my_df['col1'][0]
```

→ 0.5822986346381396

```
1 my_df
```

```
1 my_df
```

→

	col1	col2	col3	col4
a	0.582299	-0.706074	0.919884	-0.208221
b	1.109036	0.859343	-0.040539	0.777285
c	-0.321558	-0.252036	1.176383	1.260723
d	-0.649170	1.409984	0.284063	0.805140
e	-0.356754	0.052488	-0.369870	0.406328

```
1 my_df.sort_values(by=['col1','col4']) #axis : {0 or 'index',  
2 # sortint along rowwise by defualt
```

→

	col1	col2	col3	col4
d	-0.649170	1.409984	0.284063	0.805140
e	-0.356754	0.052488	-0.369870	0.406328
c	-0.321558	-0.252036	1.176383	1.260723
a	0.582299	-0.706074	0.919884	-0.208221
b	1.109036	0.859343	-0.040539	0.777285

```
1 my_df.sort_values(by='col1')# sorting along columnwise
```

```
→      col2      col4      col1      col3
a -0.706074 -0.208221  0.582299  0.919884
b  0.859343  0.777285  1.109036 -0.040539
c -0.252036  1.260723 -0.321558  1.176383
d  1.409984  0.805140 -0.649170  0.284063
e  0.052488  0.406328 -0.356754 -0.369870
```

```
1 my_df.sort_values(by='b',axis=1) # sorting along rowwise
```

```
→      col2      col4      col1      col3
a -0.706074 -0.208221  0.582299  0.919884
b  0.859343  0.777285  1.109036 -0.040539
c -0.252036  1.260723 -0.321558  1.176383
d  1.409984  0.805140 -0.649170  0.284063
e  0.052488  0.406328 -0.356754 -0.369870
```

```
1 data = {'first_name': ['Noureddin', 'Josef', 'Mike', 'Ali',
2                      'last_name': ['Sadawi', 'Baker', 'Brooks', 'Shah'],
3                      'year': [2015, 2009, 2013, 2016, 2014],
4                      'score': [422, 322, 233, 223, 401]
5                    }
6 df = pd.DataFrame(data, index = ['Tripoli', 'Birmingham', 'L
7 df
```

```
→      first_name  last_name  year  score
Tripoli        Noureddin   Sadawi  2015    422
Birmingham      Josef     Baker  2009    322
London          Mike     Brooks  2013    233
Islamabad       Ali      Shah  2016    223
New York        Luke     Brown  2014    401
```

```
1 df.sort_values(by='last_name', ascending=True) # by defult a
```

→

		first_name	last_name	year	score
Birmingham		Josef	Baker	2009	322
London		Mike	Brooks	2013	233
New York		Luke	Brown	2014	401
Tripoli		Noureddin	Sadawi	2015	422
Islamabad		Ali	Shah	2016	223

```
1 df.sort_values(by=['last_name','score'])
```

→

		first_name	last_name	year	score
Birmingham		Josef	Baker	2009	322
London		Mike	Brooks	2013	233
New York		Luke	Brown	2014	401
Tripoli		Noureddin	Sadawi	2015	422
Islamabad		Ali	Shah	2016	223

```
1 m = df[df < 240]
```

```
1 m
```

→

		first_name	last_name	year	score
Tripoli		Noureddin	Sadawi	NaN	NaN
Birmingham		Josef	Baker	NaN	NaN
London		Mike	Brooks	NaN	233.0
Islamabad		Ali	Shah	NaN	223.0
New York		Luke	Brown	NaN	NaN

```
1 m.dropna(how='all',axis=1,thresh=3)#Remove missing values.
```

```
2
```

```
3 #0, or 'index' : Drop rows which contain missing values.
```

```
4 #    * 1, or 'columns' : Drop columns which contain missing
```

```
5 #'any' : If any NA values are present, drop that row or column
```

```
6 #    * 'all' : If all values are NA, drop that row or column
```

```
7     # Keep columns with atleast 3 non-NA values  
8
```

→

	first_name	last_name	score
Tripoli	Noureddin	Sadawi	NaN
Birmingham	Josef	Baker	NaN
London	Mike	Brooks	233.0
Islamabad	Ali	Shah	223.0
New York	Luke	Brown	NaN

1 m

→

	first_name	last_name	year	score
Tripoli	Noureddin	Sadawi	NaN	NaN
Birmingham	Josef	Baker	NaN	NaN
London	Mike	Brooks	NaN	233.0
Islamabad	Ali	Shah	NaN	223.0
New York	Luke	Brown	NaN	NaN

1 m.fillna(1,inplace=True)

1 m

→

	first_name	last_name	year	score
Tripoli	Noureddin	Sadawi	1.0	1.0
Birmingham	Josef	Baker	1.0	1.0
London	Mike	Brooks	1.0	233.0
Islamabad	Ali	Shah	1.0	223.0
New York	Luke	Brown	1.0	1.0

1 Start coding or generate with AI.

## ✓ Notebook: 19) 13-Dataframes-updated.ipynb

```
1 import pandas as pd
```

```
1 Two-dimensional size-mutable, potentially heterogeneous tabu  
2 structure with labeled axes (rows and columns)
```

```
1 import numpy as np
```

```
1 arr = np.array([[1, 2, 3], [4, 5, 6]]) #2 d numpy array  
2 arr
```

```
→ array([[1, 2, 3],  
       [4, 5, 6]])
```

```
1 arr = np.array([(1, 2, 3), (4, 5, 6)]) #2 d numpy array  
2 arr
```

```
→ array([[1, 2, 3],  
       [4, 5, 6]])
```

```
1 arr = np.array([[1, 2, 3], [4, 5, 6]]) #2 d numpy array  
2 df = pd.DataFrame(arr) # create a data frame from the numpy  
3 # Pandas automatically gives them in  
4  
5 #df = pd.DataFrame(arr),columns=['col 1','col 2','col 3']
```

```
1 df # first array or vector as 1st row in data frame,
```

```
→      0  1  2  
      0  1  2  3  
      1  4  5  6
```

```
1 df[0]
```

```
→  0    1  
  1    4  
Name: 0, dtype: int32
```

```
1 df[1]
```

```
→ 0    2  
  1    5  
Name: 1, dtype: int32
```

```
1 df.columns
```

```
→ RangeIndex(start=0, stop=3, step=1)
```

```
1 df.index
```

```
→ RangeIndex(start=0, stop=2, step=1)
```

```
1 arr = np.array([[1, 2, 3], [4, 5, 6]]) #2 d numpy array  
2 df = pd.DataFrame(arr,columns=['col1','col2','col3'],index=[  
3 df
```

```
→      col1  col2  col3  
-----  
r1      1      2      3  
r2      4      5      6
```

```
1 df['col3']# acess the elements by using indices i.e acces t
```

```
→ r1    3  
  r2    6  
Name: col3, dtype: int32
```

```
1 df.columns # gives the names of columns in the form of range
```

```
→ Index(['col1', 'col2', 'col3'], dtype='object')
```

```
1 df['col2'] # columns can be labelled manually
```

```
→ r1    2  
  r2    5  
Name: col2, dtype: int32
```

```
1 s1 = pd.Series(np.random.randn(5),index=['A','B','C','D','E']
```

```
1 s1
```

```
→ A    1.968295
  B    0.235175
  C    1.041845
  D   -0.600387
  E   -0.497146
dtype: float64
```

```
1 s2 = pd.Series(np.random.randn(5),index=['A','B','C','D','E'])
```

```
1 s2
```

```
→ A   -0.123859
  B   -0.533384
  C   -1.709159
  D    1.311607
  E   -0.434962
dtype: float64
```

```
1 #axis : {0/'index', 1/'columns'}, default 0  The axis to con
```

```
2 df = pd.concat([s1, s2],axis=1)
```

```
3 df
```

```
→      0        1
  A -0.439239 -0.123859
  B  0.144203 -0.533384
  C  1.561591 -1.709159
  D -1.328464  1.311607
  E -0.672147 -0.434962
```

```
1 df[1]['E']
```

```
→ -0.43496156582130313
```

pd.concat([s1, s2], axis=1) joins the two Series side by side (columns).

reset\_index() moves the original index (A, B, C, D, E) into a new column called "index" and assigns a new default integer index (0, 1, 2, 3, 4).

```
1 #axis : {0/'index', 1/'columns'}, default 0  The axis to con  
2 df = pd.concat([s1, s2],axis=1).reset_index()  
3
```

```
1 df
```

```
→      index          0          1  
0      A   -0.439239  -0.123859  
1      B    0.144203  -0.533384  
2      C   1.561591  -1.709159  
3      D  -1.328464   1.311607  
4      E  -0.672147  -0.434962
```

```
1 df[1][0]
```

```
→ -0.12385923918840228
```

axis=1 → concat side-by-side (columns).

reset\_index(drop=True) → removes the old index without adding it as a new column.

Now you will get a DataFrame with integer indexes (0, 1, 2, 3, 4) and no "index" column.

```
1 df = pd.concat([s1, s2], axis=1).reset_index(drop=True)  
2 df
```

```
→      0          1  
0  -0.439239  -0.123859  
1   0.144203  -0.533384  
2   1.561591  -1.709159  
3  -1.328464   1.311607  
4  -0.672147  -0.434962
```

```
1 dp=s2.to_frame()  
2 print(dp)  
3 type(dp)
```

```
0
A 1.404449
B -1.133963
C -1.873227
D 0.448066
E -0.151049
pandas.core.frame.DataFrame
```

```
1 my_df = pd.DataFrame(np.random.randn(25).reshape((5,5)),inde
```

```
1 my_df
```

```
0
      c1      c2      c3      c4      c5
a  0.180907 -0.634018  0.583386 -0.458330 -0.850565
b  1.759507  0.100223 -0.128512 -0.943770  0.289280
c  0.966118  0.923943 -0.947233  1.461177  0.545747
d  0.431565  0.065300  1.034868  0.654516  0.543274
e -0.705384  0.429243 -0.430414  0.851109  0.476680
```

```
1 Drop specified labels from rows or columns.
```

```
2
```

```
3 Remove rows or columns by specifying label names and corresponding axis, or by specifying directly index or column names.
```

Drop the column c2.

Work on columns because axis=1.

Do not change my\_df permanently – instead, return a new DataFrame.

my\_df will remain unchanged unless you assign the result to a new variable.

```
1 my_df.drop('c2',axis=1,inplace=False)
```

1 my\_df

	c1	c3	c4	c5
a	0.582299	-0.706074	0.919884	-0.208221
b	1.109036	0.859343	-0.040539	0.777285
c	-0.321558	-0.252036	1.176383	1.260723
d	-0.649170	1.409984	0.284063	0.805140
e	-0.356754	0.052488	-0.369870	0.406328

1 my\_df

1 my\_df.drop('c2',axis=1,inplace=True)

	c1	c2	c3	c4	c5
a	0.582299	-0.465746	-0.706074	0.919884	-0.208221
b	1.109036	-0.556409	0.859343	-0.040539	0.777285
c	-0.321558	0.827996	-0.252036	1.176383	1.260723
d	-0.649170	-0.001465	1.409984	0.284063	0.805140
e	-0.356754	-1.673552	0.052488	-0.369870	0.406328

1 my\_df.drop('c2',axis=1,inplace=True)

1 my\_df

	col1	col2	col3	col4
a	0.180907	0.583386	-0.458330	-0.850565
b	1.759507	-0.128512	-0.943770	0.289280
c	0.966118	-0.947233	1.461177	0.545747
d	0.431565	1.034868	0.654516	0.543274
e	-0.705384	-0.430414	0.851109	0.476680

1 my\_df.columns = ['col1', 'col2', 'col3', 'col4']  
2 my\_df.index=[ 'rol1', 'rol2', 'rol3', 'rol4', '5' ]

1 my\_df

→

	col1	col2	col3	col4
a	0.180907	0.583386	-0.458330	-0.850565
b	1.759507	-0.128512	-0.943770	0.289280
c	0.966118	-0.947233	1.461177	0.545747
d	0.431565	1.034868	0.654516	0.543274
e	-0.705384	-0.430414	0.851109	0.476680

```
1 my_df.loc['a']['col2'] # accessing elements of dataframe tho
```

→ -0.7060742187474643

```
1 my_df.iloc[0][1] # integer based indexing to access elements
```

2

```
3 #Purely integer-location based indexing for selection by pos
```

→ -0.7060742187474643

```
1 my_df.iloc[0,1]
```

→ -0.7060742187474643

```
1 my_df['col1']['a']
```

→ 0.5822986346381396

```
1 my_df['col1'][0]
```

→ 0.5822986346381396

```
1 my_df
```

```
1 my_df
```

→

	col1	col2	col3	col4
a	0.582299	-0.706074	0.919884	-0.208221
b	1.109036	0.859343	-0.040539	0.777285
c	-0.321558	-0.252036	1.176383	1.260723
d	-0.649170	1.409984	0.284063	0.805140
e	-0.356754	0.052488	-0.369870	0.406328

```
1 my_df.sort_values(by=['col1','col4']) #axis : {0 or 'index',  
2 # sorint along rowwise by defualt
```

→

	col1	col2	col3	col4
d	-0.649170	1.409984	0.284063	0.805140
e	-0.356754	0.052488	-0.369870	0.406328
c	-0.321558	-0.252036	1.176383	1.260723
a	0.582299	-0.706074	0.919884	-0.208221
b	1.109036	0.859343	-0.040539	0.777285

```
1 my_df.sort_values(by='col1')# sorting along columnwise
```

→

	col2	col4	col1	col3
a	-0.706074	-0.208221	0.582299	0.919884
b	0.859343	0.777285	1.109036	-0.040539
c	-0.252036	1.260723	-0.321558	1.176383
d	1.409984	0.805140	-0.649170	0.284063
e	0.052488	0.406328	-0.356754	-0.369870

```
1 my_df.sort_values(by='b',axis=1)# sorting along rowsise
```

	col2	col4	col1	col3
a	-0.706074	-0.208221	0.582299	0.919884
b	0.859343	0.777285	1.109036	-0.040539
c	-0.252036	1.260723	-0.321558	1.176383
d	1.409984	0.805140	-0.649170	0.284063
e	0.052488	0.406328	-0.356754	-0.369870

```

1 data = {'first_name': ['Noureddin', 'Josef', 'Mike', 'Ali',
2                         'last_name': ['Sadawi', 'Baker', 'Brooks', 'Shah'],
3                         'year': [2015, 2009, 2013, 2016, 2014],
4                         'score': [422, 322, 233, 223, 401]
5                     }
6 df = pd.DataFrame(data, index = ['Tripoli', 'Birmingham', 'L
7 df

```

	first_name	last_name	year	score
Tripoli	Noureddin	Sadawi	2015	422
Birmingham	Josef	Baker	2009	322
London	Mike	Brooks	2013	233
Islamabad	Ali	Shah	2016	223
New York	Luke	Brown	2014	401

```
1 df.sort_values(by='last_name', ascending=True) # by defult a
```

	first_name	last_name	year	score
Birmingham	Josef	Baker	2009	322
London	Mike	Brooks	2013	233
New York	Luke	Brown	2014	401
Tripoli	Noureddin	Sadawi	2015	422
Islamabad	Ali	Shah	2016	223

```
1 df.sort_values(by=['last_name','score'])
```

→

	first_name	last_name	year	score
Birmingham	Josef	Baker	2009	322
London	Mike	Brooks	2013	233
New York	Luke	Brown	2014	401
Tripoli	Noureddin	Sadawi	2015	422
Islamabad	Ali	Shah	2016	223

```
1 m = df[df < 240]
```

1 m

→

	first_name	last_name	year	score
Tripoli	Noureddin	Sadawi	NaN	NaN
Birmingham	Josef	Baker	NaN	NaN
London	Mike	Brooks	NaN	233.0
Islamabad	Ali	Shah	NaN	223.0
New York	Luke	Brown	NaN	NaN

```
1 m.dropna(how='all',axis=1,thresh=3)#Remove missing values.  
2  
3 #0, or 'index' : Drop rows which contain missing values.  
4 # * 1, or 'columns' : Drop columns which contain missing  
5 #'any' : If any NA values are present, drop that row or column  
6 # * 'all' : If all values are NA, drop that row or column  
7 # Keep columns with atleast 3 non-NA values  
8
```

→

	first_name	last_name	score
Tripoli	Noureddin	Sadawi	NaN
Birmingham	Josef	Baker	NaN
London	Mike	Brooks	233.0
Islamabad	Ali	Shah	223.0
New York	Luke	Brown	NaN

1 m

→

	first_name	last_name	year	score
Tripoli	Noureddin	Sadawi	NaN	NaN
Birmingham	Josef	Baker	NaN	NaN
London	Mike	Brooks	NaN	233.0
Islamabad	Ali	Shah	NaN	223.0
New York	Luke	Brown	NaN	NaN

1 m.fillna(1,inplace=True)

1 m

→

	first_name	last_name	year	score
Tripoli	Noureddin	Sadawi	1.0	1.0
Birmingham	Josef	Baker	1.0	1.0
London	Mike	Brooks	1.0	233.0
Islamabad	Ali	Shah	1.0	223.0
New York	Luke	Brown	1.0	1.0

1 Start coding or generate with AI.

## ▼ Notebook: 2) Python Tuples.ipynb

Tuples in Python are similar to lists, but they are immutable, meaning their elements cannot be changed after creation. They are created using parentheses and can store different data types. Tuples are often used when the data should remain constant throughout the program. They're efficient #for representing fixed collections of values. list vs tuple

1. list use square bracket while tuple use parenthesis
2. creating one item tuple is different from one item list

1. Tuples
2. create Tuples
3. Access elements of Tuples
4. Edit tuples
4. Add elements to tuples
2. Delete Tuples
6. Operations on Tuple
7. Functions on Tuples

```
1 # ceate an empty tuple  
2 t= ()  
3 t
```

→ ()

```
1 # create a homogenous tuple, where items are of same data ty  
2 t2=(1,4,5,6,7)  
3 t2
```

→ (1, 4, 5, 6, 7)

```
1 # create a heterogenous tuple where some items are of differ  
2 t3=('Hello', 5,True)  
3 t3
```

→ ('Hello', 5, True)

```
1 # create a 2d tuple  
2 t4=(2,3,4,(5,6))  
3 t4[-1][-1]
```

→ 6

```
1 t3=("hello",)  
2 type(t3)
```

→ tuple

```
1 t4=(4,)  
2 type(t4)
```

→ tuple

1 Start coding or generate with AI.

1 Start coding or generate with AI.

1 Start coding or generate with AI.

```
1 # creating a single item tuple
2 t5=(5) # it is not a tuple, it is an integer
3 print(type(t5))
```

→ <class 'int'>

```
1 t5=(5,) # it is a tuple
2 print(type(t5))
```

→ <class 'tuple'>

```
1 t5=("hello") # it is not a tuple, it is a string
2 print(type(t5))
```

→ <class 'str'>

```
1 t5= ("hello",) # it is now a single item tuple
2 print(type(t5))
```

→ <class 'tuple'>

```
1 # create a tuple using type conversion
2 t5= tuple("helllo") # it converts string into tuple
3 t5
```

→ ('h', 'e', 'l', 'l', 'o')

```
1 l5= list("helllo") # it converts string into list
2 print(l5)
3 print(type(l5))
```

→ ['h', 'e', 'l', 'l', 'o']
<class 'list'>

```
1 t45=(1,2,4,[3,45,6], ("ziaullah", 'atiq'))
2 t45[3][1]
```

→ 45

```
1 # create a tuple from list
2 t6= tuple([4,5,6,7,9,10])
```

```
3 t6
```

```
→ (4, 5, 6, 7, 9, 10)
```

```
1 t0=('data science',)
2 print(id(t0))
3
4 print(t0)
5
```

```
→ 2207893042224
('data science',)
```

```
1 t0= t0 + ('XYZ', 'Peshawar')
2 print(t0)
3 print(id(t0))
```

```
→ ('data science', 'XYZ', 'Peshawar')
2207897821184
```

```
1 t11=()
2 lst= list(t11)
3 lst.append('zia')
4 lst.append('maqsood')
5 lst.append('uzair')
6 lst
7 t11=tuple(lst)
8 t11
```

```
→ ('zia', 'maqsood', 'uzair')
```

```
1 t11=("Muazzam",)
2 lst= list(t11)
3 lst.append('zia')
4 lst.append('maqsood')
5 lst.append('uzair')
6 lst
7 t11=tuple(lst)
8 t11
```

```
→ ('Muazzam', 'zia', 'maqsood', 'uzair')
```

```
1 # Access items from a tuple  
2 t11[0]
```

```
→ 'Muazzam'
```

```
1 t11[-1]
```

```
→ 'uzair'
```

```
1 t6[:4] # Access first 4 items from tuple
```

```
→ (4, 5, 6, 7)
```

```
1 t6
```

```
→ (4, 5, 6, 7, 9, 10)
```

```
1 del t6[0]
```

```
→ -----  
TypeError Traceback (most recent call last)  
Cell In[41], line 1  
----> 1 del t6[0]
```

```
TypeError: 'tuple' object doesn't support item deletion
```

```
1 t4
```

```
→ (4,)
```

```
1 t4[-1][1]
```

```
→ 6
```

```
1 t4
```

```
→ (4,)
```

```
1 # tuples just like string are immutable means tuples can not  
2 t4[0]
```

```
→ 4
```

```
1 # can you add new item to the existing tuple, the answer is |
```

```
1 t2=(2,3,4)
```

```
1 # deleting tuple  
2 del t2 # it wil delete the entire tuple
```

```
1 # we cannot delete a part or item of the tuple becos tuples  
2 del t4[-1]
```

```
→-----  
TypeError Traceback (most recent call last)  
Cell In[15], line 2  
      1 # we can delete a part or item of the tuple becos tuples are immutable  
----> 2 del t4[-1]  
  
TypeError: 'tuple' object doesn't support item deletion
```

```
1 # operations on tuples: concatenation, multipicaiton, loop v  
2 t4+t6
```

```
→ (4, 4, 5, 6, 7, 9, 10)
```

```
1 t6*3
```

```
→ (4, 5, 6, 7, 9, 10, 4, 5, 6, 7, 9, 10, 4, 5, 6, 7, 9, 10)
```

```
1 for i in t6:  
2     print(i)
```

```
→ 4  
5  
6  
7  
9  
10
```

```
1 Start coding or generate with AI.
```

```
1 8 in t6
```

```
False
```

```
1 # functions available for tuple  
2 len(t6)
```

```
6
```

```
1 min(t6)
```

```
4
```

```
1 max(t6)
```

```
10
```

```
1 sum(t6)
```

```
41
```

```
1 sorted(t6) # returns a list
```

```
[4, 5, 6, 7, 9, 10]
```

```
1 sorted(t6, reverse=True)
```

```
[10, 9, 7, 6, 5, 4]
```

```
1 # Tuples are read-only datatype and are used in scenarios wh  
2 for example: you are web developer publishing results of CS  
3 your brother got admission in stat but interested in CS.  
4 He was conditioned to get 3.8 in 1st semester in order to ge  
5 but unfortunately he could not. Now he requested  
6 brother to do something, if the data from database is in tup  
7 would not be able to change data as tuples are immutable  
8 otherwise editing in list is possible
```

Notebook: 20) 16-Pandas Dataframes-Summary

Statistics.ipynb

```
1 import pandas as pd
2 import numpy as np
3 my_df = pd.DataFrame(np.random.randn(25).reshape((5,5)),index=)
4 my_df
```

	c1	c2	c3	c4	c5
a	-0.448478	-0.497061	1.141220	0.268214	-2.195187
b	-1.678798	1.599904	1.782842	-1.333707	0.163318
c	0.077961	-0.501636	0.784316	0.134869	-0.985892
d	1.718908	-0.602497	0.754953	1.398193	0.486049
e	-0.269389	-0.613194	-0.007055	0.754934	-0.347967

```
1
2 my_df.describe() # analyze the data and gives various summa
3 # count values in column1, mean of column1 v
```

	c1	c2	c3	c4	c5
count	5.000000	5.000000	5.000000	5.000000	5.000000
mean	-0.119959	-0.122897	0.891255	0.244501	-0.575936
std	1.222788	0.964610	0.650843	1.011789	1.062581
min	-1.678798	-0.613194	-0.007055	-1.333707	-2.195187
25%	-0.448478	-0.602497	0.754953	0.134869	-0.985892
50%	-0.269389	-0.501636	0.784316	0.268214	-0.347967
75%	0.077961	-0.497061	1.141220	0.754934	0.163318
max	1.718908	1.599904	1.782842	1.398193	0.486049

```
1 my_df.transpose()
```

→

	a	b	c	d	e
<b>c1</b>	-0.448478	-1.678798	0.077961	1.718908	-0.269389
<b>c2</b>	-0.497061	1.599904	-0.501636	-0.602497	-0.613194
<b>c3</b>	1.141220	1.782842	0.784316	0.754953	-0.007055
<b>c4</b>	0.268214	-1.333707	0.134869	1.398193	0.754934
<b>c5</b>	-2.195187	0.163318	-0.985892	0.486049	-0.347967

```
1 my_df.transpose().describe() #gives various summary statis
```

→

	a	b	c	d	e
<b>count</b>	5.000000	5.000000	5.000000	5.000000	5.000000
<b>mean</b>	-0.346258	0.106712	-0.098076	0.751121	-0.096534
<b>std</b>	1.229013	1.605077	0.673600	0.902479	0.522743
<b>min</b>	-2.195187	-1.678798	-0.985892	-0.602497	-0.613194
<b>25%</b>	-0.497061	-1.333707	-0.501636	0.486049	-0.347967
<b>50%</b>	-0.448478	0.163318	0.077961	0.754953	-0.269389
<b>75%</b>	0.268214	1.599904	0.134869	1.398193	-0.007055
<b>max</b>	1.141220	1.782842	0.784316	1.718908	0.754934

```
1 #To get the summary statistics along the rows, you can trans
2 my_df.T.describe()
3
```

→

	a	b	c	d	e
<b>count</b>	5.000000	5.000000	5.000000	5.000000	5.000000
<b>mean</b>	-0.346258	0.106712	-0.098076	0.751121	-0.096534
<b>std</b>	1.229013	1.605077	0.673600	0.902479	0.522743
<b>min</b>	-2.195187	-1.678798	-0.985892	-0.602497	-0.613194
<b>25%</b>	-0.497061	-1.333707	-0.501636	0.486049	-0.347967
<b>50%</b>	-0.448478	0.163318	0.077961	0.754953	-0.269389
<b>75%</b>	0.268214	1.599904	0.134869	1.398193	-0.007055
<b>max</b>	1.141220	1.782842	0.784316	1.718908	0.754934

In the first column, the 25% value is 0.096394, which means 25% of the values are less than or equal to this value.

The 50% (median) value is 1.032588, meaning half of the values are below 1.032588.

The 75% value is 1.391746, indicating that 75% of the values are less than or equal to 1.391746.

```
1 my_df.apply(lambda x: x.describe(), axis=1)
2 #to get summary statistics along the rows.
```

	count	mean	std	min	25%	50%	75%	max
a	5.0	-0.346258	1.229013	-2.195187	-0.497061	-0.448478	0.268214	1.141220
b	5.0	0.106712	1.605077	-1.678798	-1.333707	0.163318	1.599904	1.782842
c	5.0	-0.098076	0.673600	-0.985892	-0.501636	0.077961	0.134869	0.784316
d	5.0	0.751121	0.902479	-0.602497	0.486049	0.754953	1.398193	1.718908
e	5.0	-0.096534	0.522743	-0.613194	-0.347967	-0.269389	-0.007055	0.754934

```
1 my_df.apply(lambda x: x.describe(), axis=0)
```

	c1	c2	c3	c4	c5
count	5.000000	5.000000	5.000000	5.000000	5.000000
mean	-0.119959	-0.122897	0.891255	0.244501	-0.575936
std	1.222788	0.964610	0.650843	1.011789	1.062581
min	-1.678798	-0.613194	-0.007055	-1.333707	-2.195187
25%	-0.448478	-0.602497	0.754953	0.134869	-0.985892
50%	-0.269389	-0.501636	0.784316	0.268214	-0.347967
75%	0.077961	-0.497061	1.141220	0.754934	0.163318
max	1.718908	1.599904	1.782842	1.398193	0.486049

```
1 my_df
```

→

	c1	c2	c3	c4	c5
a	-0.448478	-0.497061	1.141220	0.268214	-2.195187
b	-1.678798	1.599904	1.782842	-1.333707	0.163318
c	0.077961	-0.501636	0.784316	0.134869	-0.985892
d	1.718908	-0.602497	0.754953	1.398193	0.486049
e	-0.269389	-0.613194	-0.007055	0.754934	-0.347967

```
1 my_df.transpose()
```

→

	a	b	c	d	e
c1	-0.263026	0.996578	-2.159133	-0.342724	0.243077
c2	0.182714	-0.859663	0.431745	0.832285	-0.065601
c3	-0.308226	-0.152844	-0.424261	0.833454	0.032022
c4	1.026448	-0.752046	-0.734380	2.018358	-0.781409
c5	0.180764	1.522327	-1.241149	0.537955	-0.535210

```
1 my_df
```

→

	c1	c2	c3	c4	c5
a	-0.146897	1.114615	-2.127985	-0.751427	1.369291
b	1.136061	1.318517	-1.664691	-1.883622	-1.360831
c	-1.376439	2.393722	-0.723521	0.233385	1.623119
d	0.692717	-1.169075	1.790610	0.175946	0.011967
e	-0.262936	-2.152905	-0.799963	1.081096	1.511611

```
1 my_df.sum() # sum the columns
```

→

```
c1    -0.599796
c2    -0.614484
c3     4.456275
c4    1.222504
c5   -2.879678
dtype: float64
```

```
1 my_df.sum(axis=1) # sum the rows  
2 #my_df.transpose().sum()
```

```
→ a -1.731292  
    b 0.533559  
    c -0.490382  
    d 3.755607  
    e -0.482672  
dtype: float64
```

```
1 my_df.apply(lambda x: x.sum(), axis=1)
```

```
→ a -1.731292  
    b 0.533559  
    c -0.490382  
    d 3.755607  
    e -0.482672  
dtype: float64
```

```
1 my_df
```

```
→      c1      c2      c3      c4      c5  
a -0.146897  1.114615 -2.127985 -0.751427  1.369291  
b  1.136061  1.318517 -1.664691 -1.883622 -1.360831  
c -1.376439  2.393722 -0.723521  0.233385  1.623119  
d  0.692717 -1.169075  1.790610  0.175946  0.011967  
e -0.262936 -2.152905 -0.799963  1.081096  1.511611
```

```
1 my_df.cumsum(axis=0) # return commulative sum over a datafr
```

```
→      c1      c2      c3      c4      c5  
a -0.448478 -0.497061  1.141220  0.268214 -2.195187  
b -2.127276  1.102844  2.924061 -1.065493 -2.031869  
c -2.049315  0.601207  3.708377 -0.930623 -3.017760  
d -0.330407 -0.001290  4.463330  0.467570 -2.531711  
e -0.599796 -0.614484  4.456275  1.222504 -2.879678
```

The cumsum() method in pandas computes the cumulative sum of the elements along a specified axis.

When you write `my_df.cumsum(axis=0)`, it means you're calculating the cumulative sum column-wise (since axis=0 corresponds to columns in a DataFrame)

```
1 my_df.cumsum(axis=1) # return commulative sum along row ove
```

```
→      c1      c2      c3      c4      c5
a -0.448478 -0.945539 0.195681 0.463895 -1.731292
b -1.678798 -0.078894 1.703948 0.370241 0.533559
c  0.077961 -0.423675 0.360640 0.495510 -0.490382
d  1.718908  1.116411 1.871365 3.269557 3.755607
e -0.269389 -0.882583 -0.889639 -0.134704 -0.482672
```

```
1 my_df.min() # gives min value in each column
```

```
→ c1 -1.678798
c2 -0.613194
c3 -0.007055
c4 -1.333707
c5 -2.195187
dtype: float64
```

```
1 my_df.min(axis=1) # gives min value in each row
```

```
→ a -2.195187
b -1.678798
c -0.985892
d -0.602497
e -0.613194
dtype: float64
```

```
1 my_df.max()    # my_df.max(axis=1)
2 # my_df.min()
3 # my_df.min(axis=1)
```

```
→ c1  1.718908
c2  1.599904
c3  1.782842
c4  1.398193
c5  0.486049
dtype: float64
```

```
1 my_df
```

→

	c1	c2	c3	c4	c5
a	-0.448478	-0.497061	1.141220	0.268214	-2.195187
b	-1.678798	1.599904	1.782842	-1.333707	0.163318
c	0.077961	-0.501636	0.784316	0.134869	-0.985892
d	1.718908	-0.602497	0.754953	1.398193	0.486049
e	-0.269389	-0.613194	-0.007055	0.754934	-0.347967

```
1 my_df.idxmin() # gives index of the minimum value in each column
2 # for column1, minimum value at row a
```

→

c1	b
c2	e
c3	e
c4	b
c5	a

dtype: object

```
1 my_df
```

→

	c1	c2	c3	c4	c5
a	-0.448478	-0.497061	1.141220	0.268214	-2.195187
b	-1.678798	1.599904	1.782842	-1.333707	0.163318
c	0.077961	-0.501636	0.784316	0.134869	-0.985892
d	1.718908	-0.602497	0.754953	1.398193	0.486049
e	-0.269389	-0.613194	-0.007055	0.754934	-0.347967

```
1 my_df.idxmin(axis=1) # gives index of the minimum value in each column
```

→

a	c5
b	c1
c	c5
d	c2
e	c2

dtype: object

```
1 my_df.idxmax(axis=1) # gives index of the maximum value in each column
```

→

a	c3
b	c3
c	c3

```
d      c1  
e      c4  
dtype: object
```

```
1 my_df.idxmax(axis=0) # gives index of the maximum value in .
```

```
→ c1    b  
c2    c  
c3    b  
c4    b  
c5    b  
dtype: object
```

```
1 s = pd.Series([1, 2, 3])  
2 s
```

```
→ 0    1  
1    2  
2    3  
dtype: int64
```

```
1 s = pd.Series([1, 2, 4, 3, 5, 6, 7, 8, 9, 10])  
2 s.describe()  
3
```

```
→ count    10.00000  
mean      5.50000  
std       3.02765  
min       1.00000  
25%       3.25000  
50%       5.50000  
75%       7.75000  
max       10.00000  
dtype: float64
```

```
1 Start coding or generate with AI.
```

## ▼ Notebook: 21) 17-Selecting-Elements.ipynb

```
1 import pandas as pd # how to access the elements in datafram
```

```
1 raw_data = {  
2         "city": ["Tripoli", "Sydney", "Tripoli", "Rome", "R
```

```

3         "rank": ["1st", "2nd", "1st", "2nd", "1st", "2nd", "
4             "score1": [44, 48, 39, 41, 38, 44, 34, 54, 61],
5             "score2": [67, 63, 55, 70, 64, 75, 45, 66, 72]
6         }
7
8 df = pd.DataFrame(raw_data,
9                     index = pd.Index(['A','B','C','D','E','F'],
10                    columns = pd.Index(['city', 'rank','score1',
11

```

1 df

	attributes	city	rank	score1	score2
	letter				
A	Tripoli	1st	44	67	
B	Sydney	2nd	48	63	
C	Tripoli	1st	39	55	
D	Rome	2nd	41	70	
E	Rome	1st	38	64	
F	Tripoli	2nd	44	75	
G	Rome	1st	34	45	
H	Sydney	2nd	54	66	
I	Sydney	1st	61	72	

1 df.loc['A':'B']

	attributes	city	rank	score1	score2
	letter				
A	Tripoli	1st	44	67	
B	Sydney	2nd	48	63	

1 Allowed inputs for iloc method are:

2

3 - An integer, e.g. ``5``.

- 4 - A list or array of integers, e.g. ``[4, 3, 0]``.
- 5 - A slice object with ints, e.g. ``1:7``.

```
1 df.iloc[0:2]
```

```
→ attributes    city  rank  score1  score2
      letter
      A      Tripoli   1st     44      67
      B      Sydney    2nd     48      63
```

```
1 df.iloc[2] # returns the row labelled with index 2
```

```
→ attributes
  city      Tripoli
  rank      1st
  score1    39
  score2    55
Name: C, dtype: object
```

```
1 df
```

```
→ attributes    city  rank  score1  score2
      letter
      A      Tripoli   1st     44      67
      B      Sydney    2nd     48      63
      C      Tripoli   1st     39      55
      D      Rome      2nd     41      70
      E      Rome      1st     38      64
      F      Tripoli   2nd     44      75
      G      Rome      1st     34      45
      H      Sydney    2nd     54      66
      I      Sydney    1st     61      72
```

```
1 df.iloc[0,2]
```

```
→ 44
```

```
1 df.iloc[2,2] #Purely integer-location based indexing for se  
2                         #iloc method can be used to update the values
```

→ 39

```
1 df.loc['B','city']
```

→ 'Sydney'

```
1 df.loc['D','score1']  
2 #Single label for row and column  
3 #label-location based indexing for selection by label.Row D,  
4                         #loc method can be used to update the va
```

→ 41

```
1 df.loc['A'] # if you provide a single label, it will return
```

→ attributes

city	Tripoli
rank	1st
score1	44
score2	67
Name:	A, dtype: object

```
1 List of labels. Note using ``[[[]`` returns a DataFrame.
```

```
1 df.loc[['A','D']]# return group of rows
```

→ attributes city rank score1 score2

letter				
A	Tripoli	1st	44	67
D	Rome	2nd	41	70

image.png image.png

```
1 df.loc['A':'D']# return group of rows
```

```
→ attributes    city  rank  score1  score2  
      letter  
-----  
    A     Tripoli   1st     44     67  
    B     Sydney    2nd     48     63  
    C     Tripoli   1st     39     55  
    D     Rome      2nd     41     70
```

```
1 len(df)
```

```
→ 9
```

```
1 df.loc[[True, True, False, False, False, False, False, True, True]]  
2
```

```
→ attributes    city  rank  score1  score2  
      letter  
-----  
    A     Tripoli   1st     44     67  
    B     Sydney    2nd     48     63  
    H     Sydney    2nd     54     66  
    I     Sydney    1st     61     72
```

```
1 df
```

→ attributes city rank score1 score2

letter					
A	Tripoli	1st	44	67	
B	Sydney	2nd	48	63	
C	Tripoli	1st	39	55	
D	Rome	2nd	41	70	
E	Rome	1st	38	64	
F	Tripoli	2nd	44	75	
G	Rome	1st	34	45	
H	Sydney	2nd	54	66	
I	Sydney	1st	61	72	

```
1 df['score2']>70
```

→ letter

A	False
B	False
C	False
D	False
E	False
F	True
G	False
H	False
I	True

Name: score2, dtype: bool

```
1 df.loc[df['score2']>=70] #Conditional that returns a boolean
2 #type(df.loc[df['score1']>=41]) #dataframe
3
```

→ attributes score2

letter		
D	70	
F	75	
I	72	

```
1 df.loc[df['score2']>=70, ['score1', 'score2']]
```

```
→ attributes score1 score2
```

letter

D	41	70
F	44	75
I	61	72

image.png

```
1 df.loc[df['score1']>=41,['score1']] #Conditional that return
```

```
→ attributes score1
```

letter

A	44
B	48
D	41
F	44
H	54
I	61

```
1 df.loc[df['score1']>=41,['city','score1','score2']]
```

```
→ attributes city score1 score2
```

letter

A	Tripoli	44	67
B	Sydney	48	63
D	Rome	41	70
F	Tripoli	44	75
H	Sydney	54	66
I	Sydney	61	72

```
1 df.loc[['A','B']]
```

```
→ attributes city rank score1 score2  
letter  
A Tripoli 1st 44 67  
B Sydney 2nd 48 63
```

```
1 type(df.loc[['A','B']])
```

```
→ pandas.core.frame.DataFrame
```

```
1 Slice with labels for row and single label or multiple label  
2 above,  
3 note that both the start and stop of the slice are included.
```

image.png image.png

```
1 df.loc[['A','C','D'],['rank','score2']]  
2
```

```
→ attributes rank score2  
letter  
A 1st 67  
C 1st 55  
D 2nd 70
```

```
1 df
```

→ attributes city rank score1 score2

letter				
A	Tripoli	1st	44	67
B	Sydney	2nd	48	63
C	Tripoli	1st	39	55
D	Rome	2nd	41	70
E	Rome	1st	38	64
F	Tripoli	2nd	44	75
G	Rome	1st	34	45
H	Sydney	2nd	54	66
I	Sydney	1st	61	72

## Setting values

Set value for all items matching the list of labels

```
df.loc[['viper', 'sidewinder'], ['shield']] = 50
```

```
1 df.loc[['A', 'C', 'D'], ['score2']] = 100
2 df
```

→ attributes city rank score1 score2

letter				
A	Tripoli	1st	44	100
B	Sydney	2nd	48	63
C	Tripoli	1st	39	100
D	Rome	2nd	41	100
E	Rome	1st	38	64
F	Tripoli	2nd	44	75
G	Rome	1st	34	45
H	Sydney	2nd	54	66
I	Sydney	1st	61	72

```
1 df.loc['A','score1']=100
```

```
2 df
```

→ attributes city rank score1 score2

letter				
A	Tripoli	1st	100	100
B	Sydney	2nd	48	63
C	Tripoli	1st	39	100
D	Rome	2nd	41	100
E	Rome	1st	38	64
F	Tripoli	2nd	44	75
G	Rome	1st	34	45
H	Sydney	2nd	54	66
I	Sydney	1st	61	72

```
1 df
```

→ attributes city rank score1 score2

letter				
A	Tripoli	1st	100	100
B	Sydney	2nd	48	63
C	Tripoli	1st	39	100
D	Rome	2nd	41	100
E	Rome	1st	38	64
F	Tripoli	2nd	44	75
G	Rome	1st	34	45
H	Sydney	2nd	54	66
I	Sydney	1st	61	72

```
1 #Set value for an entire row
```

```
2 #df.loc['A']=10 # use when we have all numeric columns again
```

```
3 df.loc[['A'],['score1','score2']]=100
```

```
4 df
```

5

6

→ attributes city rank score1 score2

letter

A	Tripoli	1st	100	100
B	Sydney	2nd	48	63
C	Tripoli	1st	39	100
D	Rome	2nd	41	100
E	Rome	1st	38	64
F	Tripoli	2nd	44	75
G	Rome	1st	34	45
H	Sydney	2nd	54	66
I	Sydney	1st	61	72

```
1 df.loc['A', 'score1']=50  
2 df.loc['A', 'score2']=100  
3 df
```

→ attributes city rank score1 score2

letter

A	Tripoli	1st	50	100
B	Sydney	2nd	48	63
C	Tripoli	1st	39	100
D	Rome	2nd	41	100
E	Rome	1st	38	64
F	Tripoli	2nd	44	75
G	Rome	1st	34	45
H	Sydney	2nd	54	66
I	Sydney	1st	61	72

Set value for an entire column

```
df.loc[:, 'max_speed']=30 df
```

```
1 df.loc[:, 'score1']=100  
2 df
```

→ attributes city rank score1 score2

letter				
A	Tripoli	1st	100	100
B	Sydney	2nd	100	63
C	Tripoli	1st	100	100
D	Rome	2nd	100	100
E	Rome	1st	100	64
F	Tripoli	2nd	100	75
G	Rome	1st	100	45
H	Sydney	2nd	100	66
I	Sydney	1st	100	72

```
1 df
```

→ attributes city rank score1 score2

letter				
A	Tripoli	1st	100	100
B	Sydney	2nd	100	63
C	Tripoli	1st	100	100
D	Rome	2nd	100	100
E	Rome	1st	100	64
F	Tripoli	2nd	100	75
G	Rome	1st	100	45
H	Sydney	2nd	100	66
I	Sydney	1st	100	72

```
1 #Set value for rows matching callable condition
2 df.loc[df['score1'] < 200] = 0
3 df
```

```
→ attributes city rank score1 score2
   letter
   A      0    0     0     0
   B      0    0     0     0
   C      0    0     0     0
   D      0    0     0     0
   E      0    0     0     0
   F      0    0     0     0
   G      0    0     0     0
   H      0    0     0     0
   I      0    0     0     0
```

```
1 df['city']
```

```
→ letter
A    Tripoli
B    Sydney
C    Tripoli
D    Rome
E    Rome
F    Tripoli
G    Rome
H    Sydney
I    Sydney
Name: city, dtype: object
```

```
1 # create a dataframe from two dimensional array or list of 1
2 df = pd.DataFrame([[1, 2], [4, 5], [7, 8]],
3                   index=[7, 8, 9], columns=['max_speed', 'shield'])
4 df
```

```
→ max_speed shield
```

	max_speed	shield
7	1	2
8	4	5
9	7	8

```
1 df.loc['A','city']
```

```
→ 0
```

```
1 df.at['A','city'] # fast label based scalar accessor, gives  
2 #at method can be used to update the value
```

```
→ 0
```

```
1 df.at['A','city']=100  
2 df.at['A','city']
```

```
→ 100
```

```
1 Access a single value for a row/column label pair.  
2  
3 Similar to ``loc``, in that both provide label-based lookups  
4 ``at`` if you only need to get or set a single value in a Da  
5 or Series.
```

image.png

```
1  
2 df.iat[1,0] # fast integer location scalar accessor, element
```

```
→ 0
```

```
1 df.iloc[1,0]
```

```
→ 0
```

```
1 df['city'] # values of column
```

```
→ letter
A    Tripoli
B    Sydney
C    Tripoli
D      Rome
E      Rome
F    Tripoli
G      Rome
H    Sydney
I    Sydney
Name: city, dtype: object
```

image.png image.png

```
1 df['city']['D']
```

```
→ 0
```

```
1 df['city']['D'] #value at column city and row D
2                      # to access the element usng df, 1st specif:
```

```
→ 0
```

```
1 df['city'][3]  # we can pass index number (3) of the row in
2                      # we cannot pass index of the column
```

```
→ 'Rome'
```

```
1 df[:]  # all rows and all columns, all elements
```

→ attributes city rank score1 score2

letter				
A	Tripoli	1st	44	67
B	Sydney	2nd	48	63
C	Tripoli	1st	39	55
D	Rome	2nd	41	70
E	Rome	1st	38	64
F	Tripoli	2nd	44	75
G	Rome	1st	34	45
H	Sydney	2nd	54	66
I	Sydney	1st	61	72

```
1 df[1:3] # this will rows(1 and 2 excluding row 3)
```

→ attributes city rank score1 score2

letter				
B	Sydney	2nd	48	63
C	Tripoli	1st	39	55

1 Start coding or generate with AI.

## ▼ Notebook: 22) KNN.ipynb

```
1 from sklearn.datasets import make_blobs
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
```

```
1 inputs,target=make_blobs(n_samples=1000,
2                               centers=[(-3,3),(0,0),(2,2)],
3                               random_state=365)
```

```
1 inputs.shape,target.shape
```

```
→ ((1000, 2), (1000,))
```

```
1 data=pd.DataFrame(data=inputs,columns=[ 'Feature 1','Feature  
2 data[ 'Target']=target  
3 data
```

```
→
```

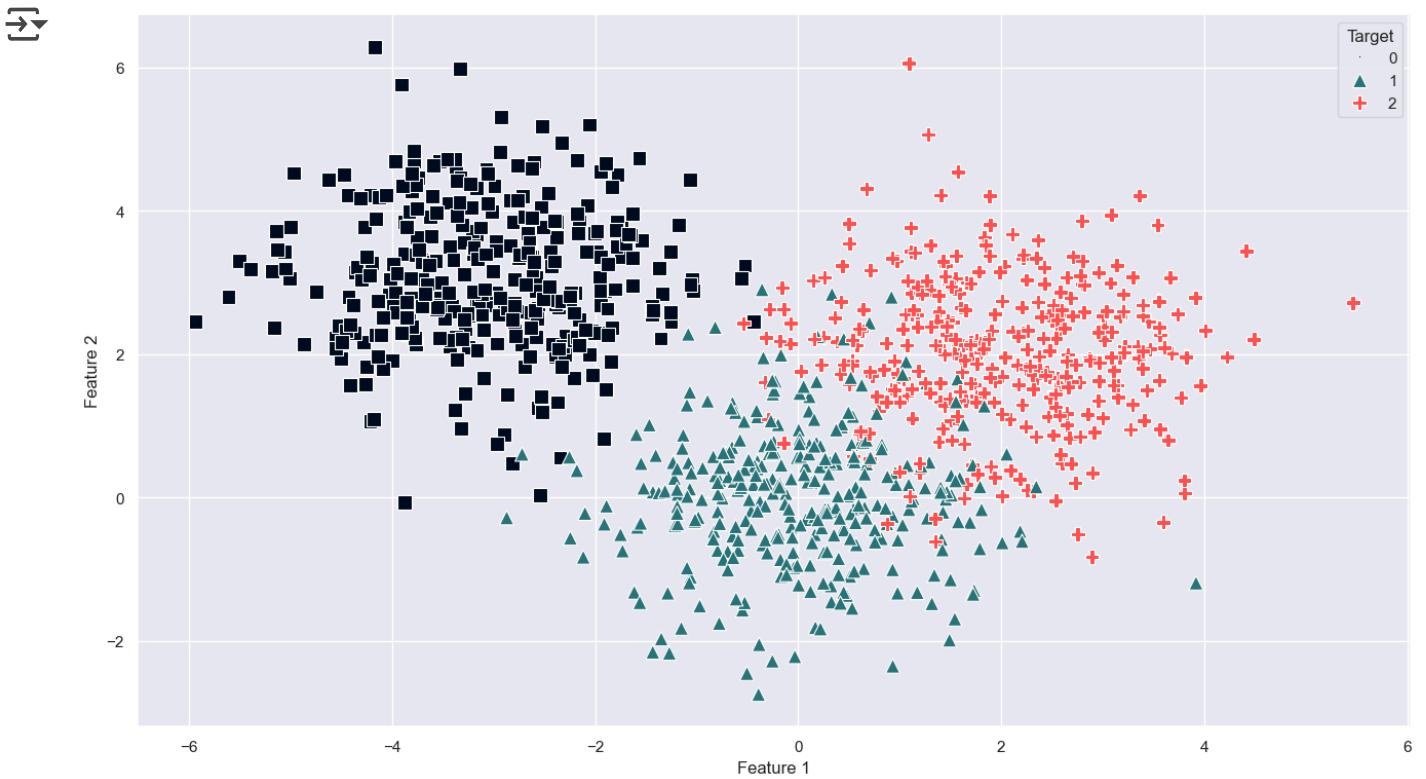
	Feature 1	Feature 2	Target
0	1.630460	2.094029	2
1	-2.811252	3.852241	0
2	0.501051	1.582531	2
3	-3.624112	3.325318	0
4	-3.278106	2.359416	0
...	...	...	...
995	-4.412271	2.420197	0
996	0.398022	-0.847863	1
997	-0.588974	0.317711	1
998	-2.328593	4.957489	0
999	-0.400935	-0.185953	1

1000 rows × 3 columns

```
1 knn_palette=sns.color_palette(['#000C1F','#29757A','#FF5050'  
2 knn_palette
```

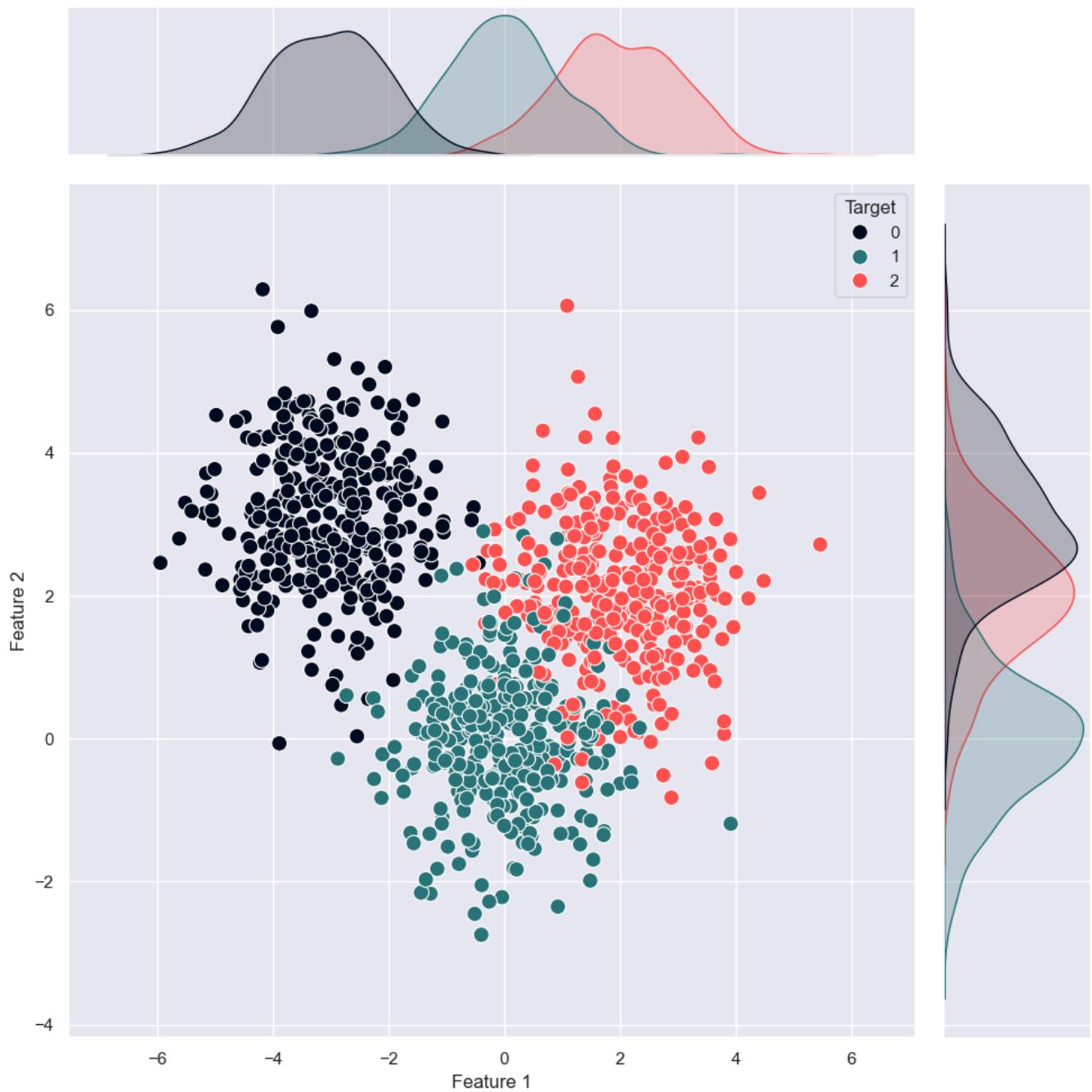
```
→
```

```
1 sns.set()  
2 plt.figure(figsize=(16,9))  
3 sns.scatterplot(x='Feature 1',y='Feature 2',  
4 data=data,  
5 hue='Target',palette=knn_palette,  
6 markers=[ ',', '^', 'P'],style='Target',  
7 s=100);
```



```
1 sns.set()
2 sns.jointplot(x='Feature 1',y='Feature 2',
3                 data=data,
4                 hue='Target',
5                 palette=knn_palette,
6                 height=10,
7                 s=100,
8                 legend=True)
```

→ <seaborn.axisgrid.JointGrid at 0x251be4a3750>



1 Start coding or generate with AI.

## ✓ Notebook: 23) Spam\_ClassificationP1-7-May-2025.ipynb

```
1 !pip cache purge # Sometimes the local cache may cause issue
```

→ Files removed: 1277

```
1 import numpy as np
2 import pandas as pd
3 import itertools
4 from sklearn.model_selection import train_test_split
5 from sklearn.feature_extraction.text import TfidfVectorizer
6 from sklearn.linear_model import PassiveAggressiveClassifier
7 from sklearn.linear_model import LogisticRegression
8 from sklearn.svm import SVC
9 from sklearn.ensemble import RandomForestClassifier
10 from sklearn.metrics import accuracy_score, confusion_matrix
```

```
1 #The issue you're encountering is due to the fact that the s
2 #Instead of pip install sklearn, you need to install scikit-
3 #!pip install sklearn
```

```
1 !pip install scikit-learn
```

```
2
```

→ Requirement already satisfied: scikit-learn in c:\users\dr. atif khan\appdata\local\program files\python\3.8\lib\site-packages  
Requirement already satisfied: numpy>=1.19.5 in c:\users\dr. atif khan\appdata\local\program files\python\3.8\lib\site-packages  
Requirement already satisfied: scipy>=1.6.0 in c:\users\dr. atif khan\appdata\local\program files\python\3.8\lib\site-packages  
Requirement already satisfied: joblib>=1.2.0 in c:\users\dr. atif khan\appdata\local\program files\python\3.8\lib\site-packages  
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\dr. atif khan\appdata\local\program files\python\3.8\lib\site-packages

[notice] A new release of pip is available: 24.0 -> 24.2

[notice] To update, run: python.exe -m pip install --upgrade pip

1 The error you're encountering, UnicodeDecodeError, typically

```
1 import pandas as pd
```

```
1 pwd
```

```
→ 'C:\\\\Users\\\\Dr. Atif Khan\\\\NAVTAC\\\\Ims'
```

```
1 df=pd.read_csv('spam.csv', encoding='latin1') # #Latin-1 is ...
2 df.shape                                     #The re...
3
```

```
→ (5572, 5)
```

```
1 #Get shape and head
2 print(df.head(10))
```

```
→      v1                               v2 Unnamed: 2 \
0  ham  Go until jurong point, crazy.. Available only ...    NaN
1  ham          Ok lar... Joking wif u oni...    NaN
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...    NaN
3  ham  U dun say so early hor... U c already then say...    NaN
4  ham  Nah I don't think he goes to usf, he lives aro...    NaN
5  spam  FreeMsg Hey there darling it's been 3 week's n...    NaN
6  ham  Even my brother is not like to speak with me. ...    NaN
7  ham  As per your request 'Melle Melle (Oru Minnamin...    NaN
8  spam  WINNER!! As a valued network customer you have...    NaN
9  spam  Had your mobile 11 months or more? U R entitle...    NaN

      Unnamed: 3 Unnamed: 4
0        NaN        NaN
1        NaN        NaN
2        NaN        NaN
3        NaN        NaN
4        NaN        NaN
5        NaN        NaN
6        NaN        NaN
7        NaN        NaN
8        NaN        NaN
9        NaN        NaN
```

```
1
2 pd.set_option('display.max_columns', None)
3 pd.set_option('display.expand_frame_repr', True) #True (defa...
4                                         #False: ...
5 pd.set_option('max_colwidth',None)   #Setting max_colwidth to
6 df.head(10)
```

→

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives around here though	NaN	NaN	NaN
5	spam	FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, £1.50 to rcv	NaN	NaN	NaN
6	ham	Even my brother is not like to speak with me. They treat me like aids patent.	NaN	NaN	NaN
7	ham	As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 to copy your friends Callertune	NaN	NaN	NaN
		WINNER!! As a valued network customer you have been			

1 pwd

→ 'C:\\\\Users\\\\Dr. Atif Khan\\\\NAVTAC\\\\Ims'

```
1 df = df.rename(columns={'v1': 'category', 'v2': 'text'})
2 df
3
```



## category

text Unnamed: 2 Unnamed: 3 Unnamed: 4

0	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives around here though	NaN	NaN	NaN
...	...	...	...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u. U have won the £750 Pound prize. 2 claim is easy, call 087187272008 NOW! Only 10p per minute. BT-national-rate.	NaN	NaN	NaN
5568	ham	Will I_b going to esplanade fr home?	NaN	NaN	NaN
5569	ham	Pity, * was in mood for that. So...any other suggestions?	NaN	NaN	NaN
5570	ham	The guy did some bitching but I acted like i'd be interested in buying something else next week	NaN	NaN	NaN

1 newdf=df[['text', 'category']]

2 newdf



text category

0	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	ham
1	Ok lar... Joking wif u oni...	ham
2	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	spam
3	U dun say so early hor... U c already then say...	ham
4	Nah I don't think he goes to usf, he lives around here though	ham
...	...	...
5567	This is the 2nd time we have tried 2 contact u. U have won the £750 Pound prize. 2 claim is easy, call 087187272008 NOW! Only 10p per minute. BT-national-rate.	spam
5568	Will i_ b going to esplanade fr home?	ham
5569	Pity, * was in mood for that. So...any other suggestions?	ham
5570	The guy did some bitching but I acted like i'd be interested in buying something else next week and he gave it to us for free	ham
5571	Rofl. Its true to its name	ham

5572 rows × 2 columns

## Step 2: Data Analysis

1. Check for Missing Values: It's essential to check if there are any missing values in your DataFrame.
2. Class Distribution: Check how many messages are "ham" and how many are "spam."

```
1 #number of missing values (NaN or None) in each column of th
2 print(newdf['text'].isnull().sum(axis=0))
```

→ 0

```
1 print(newdf.isnull().sum())
2
```

```
→ text      0
    category  0
    dtype: int64
```

```
1 #Class Distribution: Check how many messages are "ham" and h
2 #prints the frequency (count) of each unique value in the 'L
```

```
3 print(df['category'].value_counts())
4
```

```
→ category
ham    4825
spam    747
Name: count, dtype: int64
```

The ax object is used later to plot the bars using the ax.bar() method.

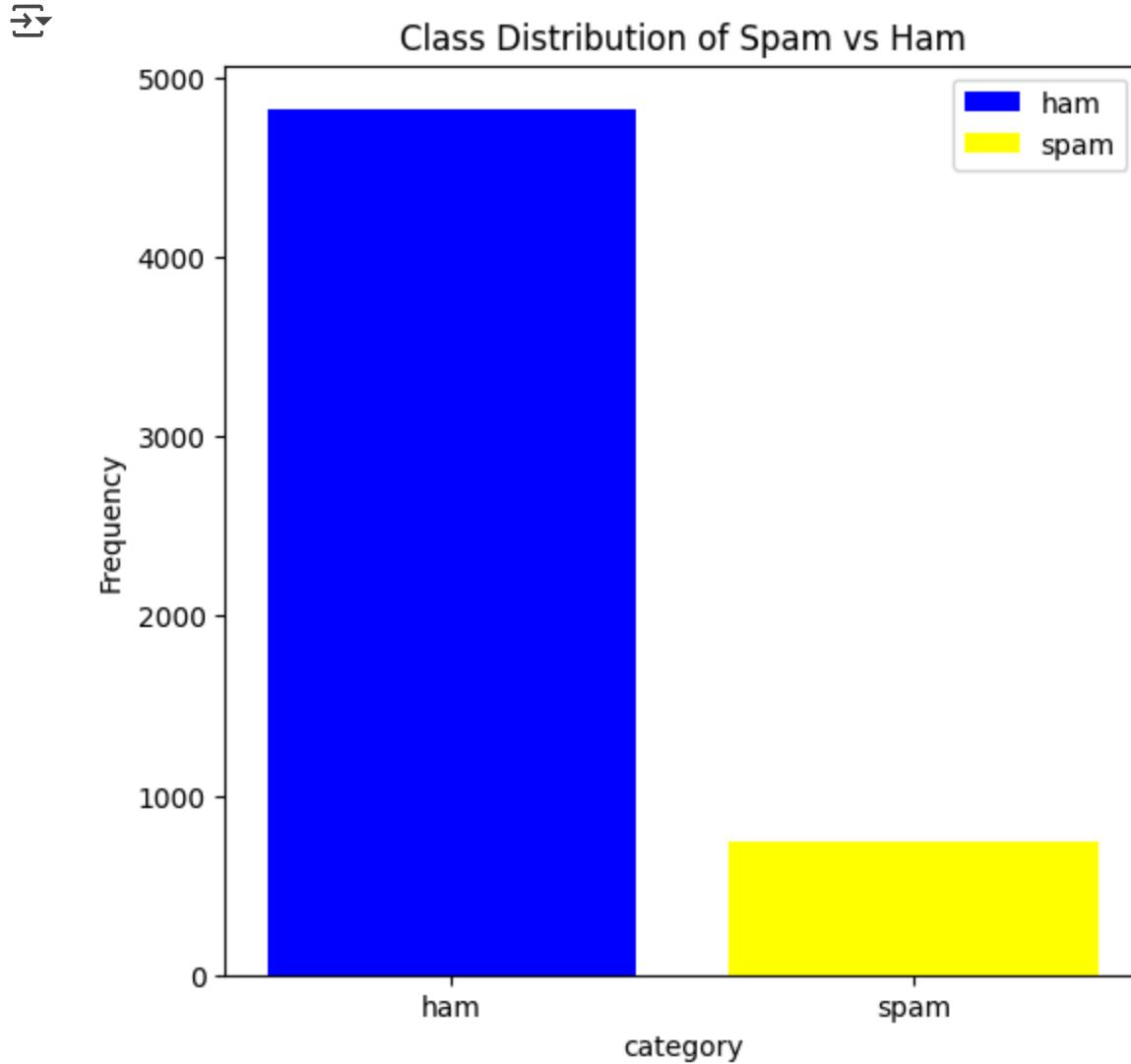
This creates a blank figure (canvas) and a set of axes (where the plot will appear).

figsize=(6, 6) sets the size of the plot (6 inches by 6 inches).

```
1 !pip install matplotlib
```

```
1 #The legend is the small box (usually on the side or top of
2 #showing the color associated with each class.
3
4 import matplotlib.pyplot as plt #library, which is used to c
5
6 # Bar Plot for Class Distribution
7 #This counts how many times each category (like "ham" and "s
8 class_distribution = newdf['category'].value_counts()
9
10 # Create a figure and axis
11 fig, ax = plt.subplots(figsize=(6, 6))
12
13 # Plot each bar separately to associate them with labels
14 #Draws a blue bar for the first category (likely 'ham') usin
15 ax.bar(class_distribution.index[0], class_distribution.value
16 #Draws a yellow bar for the first category (likely 'spam') u
17 ax.bar(class_distribution.index[1], class_distribution.value
18
19 # Add title and labels
20 ax.set_title('Class Distribution of Spam vs Ham')
21 ax.set_xlabel('category')
22 ax.set_ylabel('Frequency')
23
```

```
24 # Add a legend with the correct labels
25 ax.legend(loc='upper right')
26
27 # Show the plot
28 plt.show()
```



```
1 newdf['text'].head(10)
```

→ 0 Go until jurong point, crazy.. Available only ...
 1 Ok lar... Joking wif u oni...
 2 Free entry in 2 a wkly comp to win FA Cup fina...
 3 U dun say so early hor... U c already then say...
 4 Nah I don't think he goes to usf, he lives aro...
 5 FreeMsg Hey there darling it's been 3 week's n...
 6 Even my brother is not like to speak with me. ...
 7 As per your request 'Melle Melle (Oru Minnamin...
 8 WINNER!! As a valued network customer you have...

```
9     Had your mobile 11 months or more? U R entitle...
Name: text, dtype: object
```

```
1 ' '.join(newdf['text'].head(10))
```

```
→ "Go until jurong point, crazy.. Available only in bugis n great world la e buffet...
Cine there got amore wat... Ok lar... Joking wif u oni... Free entry in 2 a wkly comp
to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std
txt rate)T&C's apply 08452810075over18's U dun say so early hor... U c already then
say... Nah I don't think he goes to usf, he lives around here though FreeMsg Hey there
darling it's been 3 week's now and no word back! I'd like some fun you up for it still?
Tb ok! Xxx std chgs to send, ₹1.50 to rcv Even my brother is not like to speak with
me. They treat me like aids patient. As per your request 'Melle Melle (Oru
Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press
*9 to copy your friends Callertune WINNER!! As a valued network customer you have been
selected to receivea ₹900 prize reward! To claim call 09061701461. Claim code KL341.
Valid 12 hours only. Had your mobile 11 months or more? U R entitled to Update to the
latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on
08002986030"
```

```
1 df.head(2)
```

	category	text	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN

```
1 ' '.join(newdf['text'].head(2))
```

```
→ 'Go until jurong point, crazy.. Available only in bugis n great world la e buffet...
Cine there got amore wat... Ok lar... Joking wif u oni...'
```

```
1 import matplotlib.pyplot as plt
2
3 from wordcloud import WordCloud
4
5 text_corpus = ' '.join(newdf['text'])
6
7 wordcloud = WordCloud(width=800, height=400, random_state=42
8
9 # #This creates a new figure (or canvas) for the plot, with
10 #This function is used to create a new figure(or the overall
11 plt.figure(figsize=(10, 5))
12
```

13

```
14 ##This displays the word cloud image. The wordcloud object c
15 #interpolation='bilinear': This smooths the image when it is
16 plt.imshow(wordcloud, interpolation='bilinear')
17
18 ##this removes the axis from the plot, so no axis lines or la
19 plt.axis('off')
20
21 ##Adds a title to the plot, in this case, "Word Cloud for Te
22 plt.title('Word Cloud for English')
23
24 # Displays the plot with the word cloud.
25 plt.show()
```



1. Step 3: Text Preprocessing
  2. Before you can use this data for modeling, you typically need to preprocess the text data.  
Common steps include:
  3. Lowercasing
  4. Removing punctuation

5. Removing stop words
6. Tokenization
7. Converting text to numerical features (e.g., using Bag of Words or TF-IDF)

```
1 s=r'\n'  
2 print(s)  
3 t= '\n'  
4 print(t)
```

→ \n

```
1 import re  
2 text='&&&&akhtar age is 14 and??? saeed age is 15 are %%fr  
3 text1=re.sub(r'^[a-zA-Z0-9\s]', '', text)  
4 text1=text1.split(" ")  
5 text1
```

→ ['akhtar',  
 'age',  
 'is',  
 '14',  
 'and',  
 'saeed',  
 'age',  
 'is',  
 '15',  
 'are',  
 'friends',  
 'but',  
 'they',  
 'usuallydisagree',  
 'on',  
 'certain',  
 'matters']

1 Start coding or generate with AI.

## ▼ Notebook: 24)Spam\_ClassificationP2-9-5-2025.ipynb

```
1 !pip cache purge # Sometimes the local cache may cause issue
```

```
→ Files removed: 1277
```

```
1 import numpy as np
2 import pandas as pd
3 import itertools
4 from sklearn.model_selection import train_test_split
5 from sklearn.feature_extraction.text import TfidfVectorizer
6 from sklearn.linear_model import PassiveAggressiveClassifier
7 from sklearn.linear_model import LogisticRegression
8 from sklearn.svm import SVC
9 from sklearn.ensemble import RandomForestClassifier
10 from sklearn.metrics import accuracy_score, confusion_matrix
```

```
1 #The issue you're encountering is due to the fact that the s
2 #Instead of pip install sklearn, you need to install scikit-
3 #!pip install sklearn
```

```
1 !pip install scikit-learn
2
```

```
→ Requirement already satisfied: scikit-learn in c:\users\dr. atif khan\appdata\local\prog
Requirement already satisfied: numpy>=1.19.5 in c:\users\dr. atif khan\appdata\local\pro
Requirement already satisfied: scipy>=1.6.0 in c:\users\dr. atif khan\appdata\local\pro
Requirement already satisfied: joblib>=1.2.0 in c:\users\dr. atif khan\appdata\local\pro
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\dr. atif khan\appdata\lc

[notice] A new release of pip is available: 24.0 -> 24.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
1 The error you're encountering, UnicodeDecodeError, typically
```

```
1 import pandas as pd
```

```
1 pwd
```

```
→ 'E:\\Python For Advance Application\\notebooks'
```

```
1 df=pd.read_csv('spam.csv', encoding='latin1') # #Latin-1 is
2 df.shape #The re
```

```
3
```

```
→ (5572, 5)
```

```
1 #Get shape and head  
2 print(df.head(10))
```

```
→ v1 v2 Unnamed: 2 \  
0 ham Go until jurong point, crazy.. Available only ... NaN  
1 ham Ok lar... Joking wif u oni... NaN  
2 spam Free entry in 2 a wkly comp to win FA Cup fina... NaN  
3 ham U dun say so early hor... U c already then say... NaN  
4 ham Nah I don't think he goes to usf, he lives aro... NaN  
5 spam FreeMsg Hey there darling it's been 3 week's n... NaN  
6 ham Even my brother is not like to speak with me. ... NaN  
7 ham As per your request 'Melle Melle (Oru Minnamin... NaN  
8 spam WINNER!! As a valued network customer you have... NaN  
9 spam Had your mobile 11 months or more? U R entitle... NaN  
  
Unnamed: 3 Unnamed: 4  
0      NaN      NaN  
1      NaN      NaN  
2      NaN      NaN  
3      NaN      NaN  
4      NaN      NaN  
5      NaN      NaN  
6      NaN      NaN  
7      NaN      NaN  
8      NaN      NaN  
9      NaN      NaN
```

```
1  
2 pd.set_option('display.max_columns', None)  
3 pd.set_option('display.expand_frame_repr', True) #True (defa  
4 #False: |  
5 pd.set_option('max_colwidth',None) #Setting max_colwidth to  
6 df.head(10)
```

→

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives around here though	NaN	NaN	NaN
5	spam	FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, £1.50 to rcv	NaN	NaN	NaN
6	ham	Even my brother is not like to speak with me. They treat me like aids patent.	NaN	NaN	NaN
7	ham	As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 to copy your friends Callertune	NaN	NaN	NaN
		WINNER!! As a valued network customer you have been			

1 pwd

→ 'E:\\Python For Advance Application\\notebooks'

```
1 df = df.rename(columns={'v1': 'category', 'v2': 'text'})
2 df
3
```



## category

text Unnamed: 2 Unnamed: 3 Unnamed: 4

0	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives around here though	NaN	NaN	NaN
...	...	...	...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u. U have won the £750 Pound prize. 2 claim is easy, call 087187272008 NOW! Only 10p per minute. BT-national-rate.	NaN	NaN	NaN
5568	ham	Will I_b going to esplanade fr home?	NaN	NaN	NaN
5569	ham	Pity, * was in mood for that. So...any other suggestions?	NaN	NaN	NaN
5570	ham	The guy did some bitching but I acted like i'd be interested in buying something else next week	NaN	NaN	NaN

1 newdf=df[['text', 'category']]

2 newdf



text category

0	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	ham
1	Ok lar... Joking wif u oni...	ham
2	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	spam
3	U dun say so early hor... U c already then say...	ham
4	Nah I don't think he goes to usf, he lives around here though	ham
...	...	...
5567	This is the 2nd time we have tried 2 contact u. U have won the £750 Pound prize. 2 claim is easy, call 087187272008 NOW1! Only 10p per minute. BT-national-rate.	spam
5568	Will I_ b going to esplanade fr home?	ham
5569	Pity, * was in mood for that. So...any other suggestions?	ham
5570	The guy did some bitching but I acted like i'd be interested in buying something else next week and he gave it to us for free	ham
5571	Rofl. Its true to its name	ham

5572 rows × 2 columns

```
1 CORPUS = [  
2 'the sky is blue',  
3 'sky is blue and sky is beautiful',  
4 'the beautiful sky is so blue',  
5 'i love blue cheese'  
6 ]  
7 new_doc = ['loving this blue sky today']
```

```
1 from sklearn.feature_extraction.text import CountVectorizer  
2  
3 # Define the corpus  
4 CORPUS = [  
5     'the sky is blue',  
6     'sky is blue and sky is beautiful',  
7     'the beautiful sky is so blue',  
8     'i love blue cheese'  
9 ]  
10 vectorizer = CountVectorizer(ngram_range=(1,1))  
11
```

```
12     # Fit on corpus and transform both corpus and new document
13
14 X_corpus = vectorizer.fit_transform(CORPUS)
15
16 #Vocabulary Lookup, You can see which word corresponds to what index
17 print(vectorizer.vocabulary_)
18
19 print(X_corpus)
20 # The output is a sparse matrix produced by countvectorizer
21 #Each line represents a non-zero entry in the term-document matrix
22 #In the tuple format (doc_index, word_index), the second value is the
23 # 2 → "blue"
24
25 # 8 → "the"
26
27 # 4 → "is"
28
29 # So in (0, 2) 1, it means the word "blue" appears once in document 0
```

```
→ { 'the': 8, 'sky': 6, 'is': 4, 'blue': 2, 'and': 0, 'beautiful': 1, 'so': 7, 'love': 5, 'brown': 3 }
    (0, 8)      1
    (0, 6)      1
    (0, 4)      1
    (0, 2)      1
    (1, 6)      2
    (1, 4)      2
    (1, 2)      1
    (1, 0)      1
    (1, 1)      1
    (2, 8)      1
    (2, 6)      1
    (2, 4)      1
    (2, 2)      1
    (2, 1)      1
    (2, 7)      1
    (3, 2)      1
    (3, 5)      1
    (3, 3)      1
```

```
1 from sklearn.feature_extraction.text import CountVectorizer
2
3 # Define the corpus
4 CORPUS = [
5     'the sky is blue',
```

```
6     'sky is blue and sky is beautiful',
7     'the beautiful sky is so blue',
8     'i love blue cheese'
9 ]
10
11 #new_doc = ['loving this blue sky today']
12
13 # Define a function to extract n-grams
14 def extract_ngrams(corpus, ngram_range=(1,1)):
15
16     vectorizer = CountVectorizer(ngram_range=ngram_range)
17
18     # Fit on corpus and transform both corpus and new docume
19
20     X_corpus = vectorizer.fit_transform(corpus)
21     #X_new_doc = vectorizer.transform(new_doc)
22
23
24
```

```
1 corpus_unigrams = extract_ngrams(CORPUS, ngram_range=(1, 1))
```

```
1 def extract_ngrams(corpus, ngram_range=(1,1)):
2
3     vectorizer = CountVectorizer(ngram_range=ngram_range)
4
5     # Fit on corpus and transform both corpus and new docume
6
7     X_corpus = vectorizer.fit_transform(corpus)
8     #X_new_doc = vectorizer.transform(new_doc)
9 # Get feature names (n-grams)
10    feature_names = vectorizer.get_feature_names_out()
11
12    # Convert the results to arrays
13    corpus_ngrams = X_corpus.toarray()
14    #new_doc_ngrams = X_new_doc.toarray()
15
16    return feature_names, corpus_ngrams
```

```
17
18 # Unigrams (1-gram)
19 unigrams, corpus_unigrams = extract_ngrams(CORPUS, ngram_range)
20 print("Unigrams:")
21 print(unigrams)
22 print("\n")
23 print("corpus matrix is\n", corpus_unigrams)
24 #print(new_doc_unigrams)
25
```

```
→ Unigrams:
['and' 'beautiful' 'blue' 'cheese' 'is' 'love' 'sky' 'so' 'the']
```

```
corpus matrix is
[[0 0 1 0 1 0 1 0 1]
 [1 1 1 0 2 0 2 0 0]
 [0 1 1 0 1 0 1 1 1]
 [0 0 1 1 0 1 0 0 0]]
```

```
1 # Bigrams (2-grams)
2 bigrams, corpus_bigrams = extract_ngrams(CORPUS, ngram_range)
3 print("\nBigrams:")
4 print(bigrams)
5 print(corpus_bigrams)
6 #print(new_doc_bigrams)
7
8 # Trigrams (3-grams)
9 trigrams, corpus_trigrams= extract_ngrams(CORPUS, ngram_range)
10 print("\nTrigrams:")
11 print(trigrams)
12 print(corpus_trigrams)
13 #print(new_doc_trigrams)
14
```

```
→
Bigrams:
['and sky' 'beautiful sky' 'blue and' 'blue cheese' 'is beautiful'
 'is blue' 'is so' 'love blue' 'sky is' 'so blue' 'the beautiful'
 'the sky']
[[0 0 0 0 1 0 0 1 0 0 1]
 [1 0 1 0 1 1 0 0 2 0 0]
 [0 1 0 0 0 0 1 0 1 1 0]
 [0 0 0 1 0 0 0 1 0 0 0]]
```

Trigrams:

```
[ 'and sky is' 'beautiful sky is' 'blue and sky' 'is blue and' 'is so blue'
  'love blue cheese' 'sky is beautiful' 'sky is blue' 'sky is so'
  'the beautiful sky' 'the sky is']
[[0 0 0 0 0 0 1 0 0 1]
 [1 0 1 1 0 0 1 1 0 0 0]
 [0 1 0 0 1 0 0 0 1 1 0]
 [0 0 0 0 0 1 0 0 0 0 0]]
```

```
1 import pandas as pd
2 from sklearn.feature_extraction.text import CountVectorizer
3
4 # Define the corpus and new document
5 CORPUS = [
6     'the sky is blue',
7     'sky is blue and sky is beautiful',
8     'the beautiful sky is so blue',
9     'i love blue cheese'
10 ]
11
12 new_doc = ['loving this blue sky today']
13
14 # Define a function to extract n-grams and return them as a DataFrame
15 def extract_ngrams_df(corpus, new_doc, ngram_range=(1,1)):
16     vectorizer = CountVectorizer(ngram_range=ngram_range)
17
18     # Fit on corpus and transform both corpus and new document
19     X_corpus = vectorizer.fit_transform(corpus)
20     print(type(X_corpus))
21     print(X_corpus)
22     X_new_doc = vectorizer.transform(new_doc)
23
24     # Get feature names (n-grams)
25     feature_names = vectorizer.get_feature_names_out()
26     print("\n\n features names are: ",feature_names)
27
28     features = X_corpus.toarray()                      #toarray() and toarray()
29     # Convert the results to arrays
30     print("*****")
31     print("\n features are ",X_corpus.todense()) # dense matrix
32     print("\n features are\n",features) # dense matrix representation
```

```
33     print("\n\n")
34
35     corpus_ngrams = X_corpus.toarray().sum(axis=0) # Sum co
36     #new_doc_ngrams = X_new_doc.toarray().flatten() # Flatt
37
38     # Create a DataFrame for better visualization
39     df1 = pd.DataFrame(data=features,
40                         columns=feature_names)
41
42     df2 = pd.DataFrame({
43         'N-gram': feature_names,
44         'Corpus Count': corpus_ngrams,
45
46     })
47
48
49
50     return df1, df2
51
52 # Unigrams (1-gram)
53 df_unigrams1, df_unigrams2 = extract_ngrams_df(CORPUS, new_d
54 print("\n\nUnigrams1 DataFrame:")
55 print(df_unigrams1)
56 print("\n\nUnigrams2 DataFrame:")
57 print(df_unigrams2)
58
59
```

```
→ <class 'scipy.sparse._csr.csr_matrix'>
(0, 8)      1
(0, 6)      1
(0, 4)      1
(0, 2)      1
(1, 6)      2
(1, 4)      2
(1, 2)      1
(1, 0)      1
(1, 1)      1
(2, 8)      1
(2, 6)      1
(2, 4)      1
(2, 2)      1
(2, 1)      1
(2, 7)      1
```

```
(3, 2)      1
(3, 5)      1
(3, 3)      1
```

```
features names are:  ['and' 'beautiful' 'blue' 'cheese' 'is' 'love' 'sky' 'so' 'the'
*****
*****
```

```
features are [[0 0 1 0 1 0 1 0 1]
[1 1 1 0 2 0 2 0 0]
[0 1 1 0 1 0 1 1 1]
[0 0 1 1 0 1 0 0 0]]
```

```
features are
[[0 0 1 0 1 0 1 0 1]
[1 1 1 0 2 0 2 0 0]
[0 1 1 0 1 0 1 1 1]
[0 0 1 1 0 1 0 0 0]]
```

Unigrams1 DataFrame:

	and	beautiful	blue	cheese	is	love	sky	so	the
0	0	0	1	0	1	0	1	0	1
1	1	1	1	0	2	0	2	0	0
2	0	1	1	0	1	0	1	1	1
3	0	0	1	1	0	1	0	0	0

Unigrams2 DataFrame:

	N-gram	Corpus	Count
0	and	1	
1	beautiful	2	
2	blue	4	
3	cheese	1	
4	is	4	
5	love	1	
6	sky	4	
-		-	-

```
1 # Define a function to extract n-grams and return them as a |
2 def extract_ngrams_df(corpus, new_doc, ngram_range=(1,1)):
3     vectorizer = CountVectorizer(ngram_range=ngram_range)
4
5     # Fit on corpus and transform both corpus and new docume
6     X_corpus = vectorizer.fit_transform(corpus)
7     print(type(X_corpus))
8     print(X_corpus)
9     X_new_doc = vectorizer.transform(new_doc)
```

```

10
11     # Get feature names (n-grams)
12     feature_names = vectorizer.get_feature_names_out()
13     print("\n\n features names are: ",feature_names)
14
15     features = X_corpus.toarray()                      #toarray() and t
16     # Convert the results to arrays
17     print("*****")
18     print("\n features are ",X_corpus.todense()) # dense mat
19     print("\n features are\n",features) # dense matrix repre
20     print("\n\n")
21
22     corpus_ngrams = X_corpus.toarray().sum(axis=0) # Sum co
23     #new_doc_ngrams = X_new_doc.toarray().flatten() # Flatt
24
25     # Create a DataFrame for better visualization
26     df1 = pd.DataFrame(data=features,
27                         columns=feature_names)
28
29     df2 = pd.DataFrame({
30         'N-gram': feature_names,
31         'Corpus Count': corpus_ngrams,
32
33     })
34
35
36
37     return df1, df2
38

```

```

1 # Bigrams (2-grams)
2 df_bigrams1, df_bigrams2 = extract_ngrams_df(CORPUS, new_doc
3 print("\nBigrams1 DataFrame:")
4 print(df_bigrams1)
5
6 print("\nBigrams2 DataFrame:")
7 print(df_bigrams2)
8

```

9

10

```
→ <class 'scipy.sparse._csr.csr_matrix'>
  (0, 11)      1
  (0, 8)       1
  (0, 5)       1
  (1, 8)       2
  (1, 5)       1
  (1, 2)       1
  (1, 0)       1
  (1, 4)       1
  (2, 8)       1
  (2, 10)      1
  (2, 1)       1
  (2, 6)       1
  (2, 9)       1
  (3, 7)       1
  (3, 3)       1
```

features names are: ['and sky' 'beautiful sky' 'blue and' 'blue cheese' 'is beautiful'  
'is blue' 'is so' 'love blue' 'sky is' 'so blue' 'the beautiful'  
'the sky']

\*\*\*\*\*

```
features are [[0 0 0 0 0 1 0 0 1 0 0 1]
[1 0 1 0 1 1 0 0 2 0 0 0]
[0 1 0 0 0 0 1 0 1 1 1 0]
[0 0 0 1 0 0 0 1 0 0 0 0]]
```

```
features are
[[0 0 0 0 0 1 0 0 1 0 0 1]
[1 0 1 0 1 1 0 0 2 0 0 0]
[0 1 0 0 0 0 1 0 1 1 1 0]
[0 0 0 1 0 0 0 1 0 0 0 0]]
```

Bigrams1 DataFrame:

	and sky	beautiful sky	blue and	blue cheese	is beautiful	is blue	is so	love
0	0	0	0	0	0	0	1	0
1	1	0	1	0	1	1	1	0
2	0	1	0	0	0	0	0	1
3	0	0	0	1	0	0	0	0

Bigrams2 DataFrame:

	N-gram	Corpus	Count
0	and sky		1
1	beautiful sky		1
2	blue and		1
3	blue cheese		1
4	is beautiful		1
5	is blue		2

```
6      is so          1
7      love blue      1
8      sky is          4
9      so blue         1
-- ... . . . . .
```

```
1 # Trigrams (3-grams)
2 df_trigrams = extract_ngrams_df(CORPUS, new_doc, ngram_range
3 print("\nTrigrams DataFrame:")
4 print(df_trigrams)
```

```
1 arr=[
2     [1, 1, 1, 1],  # Document 1
3     [0, 2, 1, 1],  # Document 2
4     [1, 1, 1, 1]   # Document 3
5 ]
6
```

```
1 arr.torray().sum(axis=0) #will sum the counts for each unigr
```

```
→ -----  
AttributeError                                     Traceback (most recent call last)  
Cell In[25], line 1  
----> 1 arr.torray().sum(axis=0) #will sum the counts for each unigram (column-wise)  
across all documents:  
  
AttributeError: 'list' object has no attribute 'torray'
```

```
1 from scipy.sparse import csr_matrix
2 import numpy as np
3
4 # Define non-zero values and their coordinates
5 data = np.array([3, 4, 5])                      # Non-zero values
6 row_indices = np.array([0, 1, 2])                # Row indices
7 col_indices = np.array([2, 0, 1])                # Column indices
8
9 # Create a sparse matrix
10 sparse_matrix = csr_matrix((data, (row_indices, col_indices))
11
12 # Display the sparse matrix
```

```
13 print(sparse_matrix)
```

```
14
```

```
→ (0, 2)      3  
  (1, 0)      4  
  (2, 1)      5
```

```
1 from scipy.sparse import csr_matrix  
2  
3 # Define a dense 2D list (matrix)  
4 dense_array = [  
5     [0, 0, 3],  
6     [4, 0, 0],  
7     [0, 5, 0]  
8 ]  
9  
10 # Create a sparse matrix  
11 sparse_matrix = csr_matrix(dense_array)  
12  
13 # Display the sparse matrix  
14 print(sparse_matrix)  
15
```

```
→ (0, 2)      3  
  (1, 0)      4  
  (2, 1)      5
```

```
1 sparse_matrix.toarray()
```

```
→ array([[0, 0, 3],  
        [4, 0, 0],  
        [0, 5, 0]], dtype=int32)
```

```
1 sparse_matrix.toarray().sum(axis=0)
```

```
→ array([4, 5, 3], dtype=int32)
```

```
1 Start coding or generate with AI.
```

```
1
```

```
2 import pandas as pd
```

```
3 df=pd.read_csv('spam.csv', encoding='latin1') # #Latin-1 is ...
4 df.shape
```

→ (5572, 5)

```
1 df = df.rename(columns={'v1': 'category', 'v2': 'text'})
2 newdf=df[['text', 'category']]
3 newdf
```

→

		text category
0	Go until jurong point, crazy.. Available only ...	ham
1	Ok lar... Joking wif u oni...	ham
2	Free entry in 2 a wkly comp to win FA Cup fina...	spam
3	U dun say so early hor... U c already then say...	ham
4	Nah I don't think he goes to usf, he lives aro...	ham
...	...	...
5567	This is the 2nd time we have tried 2 contact u...	spam
5568	Will l_b going to esplanade fr home?	ham
5569	Pity, * was in mood for that. So...any other s...	ham
5570	The guy did some bitching but I acted like i'd...	ham
5571	Rofl. Its true to its name	ham

5572 rows × 2 columns

## Step 2: Data Analysis

1. Check for Missing Values: It's essential to check if there are any missing values in your DataFrame.
2. Class Distribution: Check how many messages are "ham" and how many are "spam."

```
1 #number of missing values (NaN or None) in each column of the
2 print(newdf['text'].isnull().sum(axis=0))
```

→ 0

```
1 print(newdf.isnull().sum())
2
```

```
→ text      0  
category   0  
dtype: int64
```

```
1 #Class Distribution: Check how many messages are "ham" and h  
2 #prints the frequency (count) of each unique value in the 'L  
3 print(df['category'].value_counts())  
4
```

```
→ category  
ham      4825  
spam     747  
Name: count, dtype: int64
```

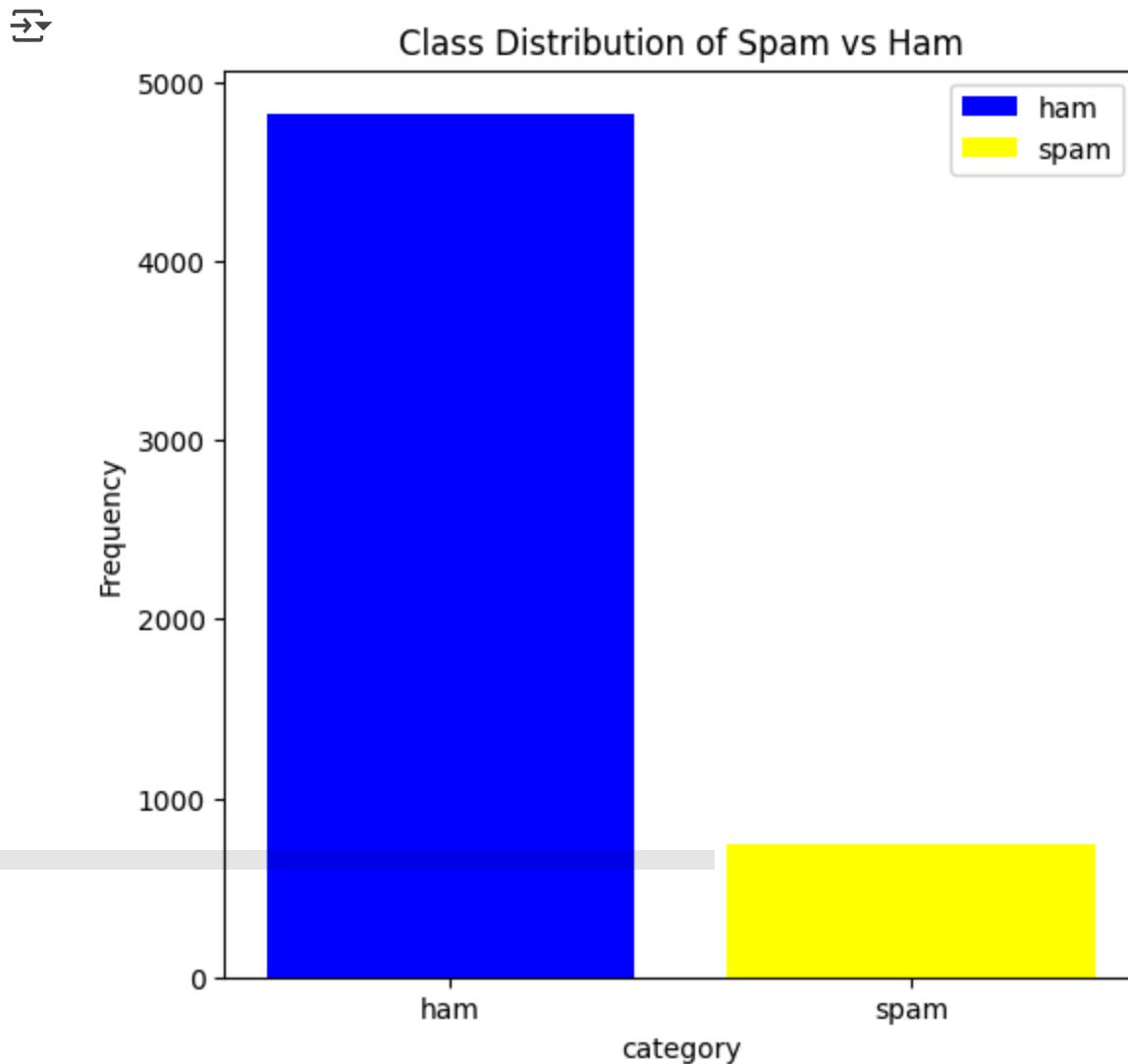
The ax object is used later to plot the bars using the ax.bar() method.

This creates a blank figure (canvas) and a set of axes (where the plot will appear).  
figsize=(6, 6) sets the size of the plot (6 inches by 6 inches).

```
1 !pip install matplotlib
```

```
1 #The legend is the small box (usually on the side or top of  
2 #showing the color associated with each class.  
3  
4 import matplotlib.pyplot as plt #library, which is used to c  
5  
6 # Bar Plot for Class Distribution  
7 #This counts how many times each category (like "ham" and "s  
8 class_distribution = newdf['category'].value_counts()  
9  
10 # Create a figure and axis  
11 fig, ax = plt.subplots(figsize=(6, 6))  
12  
13 # Plot each bar separately to associate them with labels  
14 #Draws a blue bar for the first category (likely 'ham') usin  
15 ax.bar(class_distribution.index[0], class_distribution.value  
16 #Draws a yellow bar for the first category (likely 'spam') u  
17 ax.bar(class_distribution.index[1], class_distribution.value
```

```
18
19 # Add title and labels
20 ax.set_title('Class Distribution of Spam vs Ham')
21 ax.set_xlabel('category')
22 ax.set_ylabel('Frequency')
23
24 # Add a legend with the correct labels
25 ax.legend(loc='upper right')
26
27 # Show the plot
28 plt.show()
```



```
1 newdf['text'].head(10)
```

```
→ 0 Go until jurong point, crazy.. Available only ...
    1 Ok lar... Joking wif u oni...
    2 Free entry in 2 a wkly comp to win FA Cup fina...
    3 U dun say so early hor... U c already then say...
    4 Nah I don't think he goes to usf, he lives aro...
    5 FreeMsg Hey there darling it's been 3 week's n...
    6 Even my brother is not like to speak with me. ...
    7 As per your request 'Melle Melle (Oru Minnamin...
    8 WINNER!! As a valued network customer you have...
    9 Had your mobile 11 months or more? U R entitle...

Name: text, dtype: object
```

```
1 ' '.join(newdf['text'].head(10))
```

```
→ "Go until jurong point, crazy.. Available only in bugis n great world la e buffet...
Cine there got amore wat... Ok lar... Joking wif u oni... Free entry in 2 a wkly comp
to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std
txt rate)T&C's apply 08452810075over18's U dun say so early hor... U c already then
say... Nah I don't think he goes to usf, he lives around here though FreeMsg Hey there
darling it's been 3 week's now and no word back! I'd like some fun you up for it still?
Tb ok! XxX std chgs to send, £1.50 to rcv Even my brother is not like to speak with
me. They treat me like aids patient. As per your request 'Melle Melle (Oru
Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press
*9 to copy your friends Callertune WINNER!! As a valued network customer you have been
selected to receivea £900 prize reward! To claim call 09061701461. Claim code KL341.
Valid 12 hours only. Had your mobile 11 months or more? U R entitled to Update to the
latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on
08002986030"
```

```
1 df.head(2)
```

```
→
```

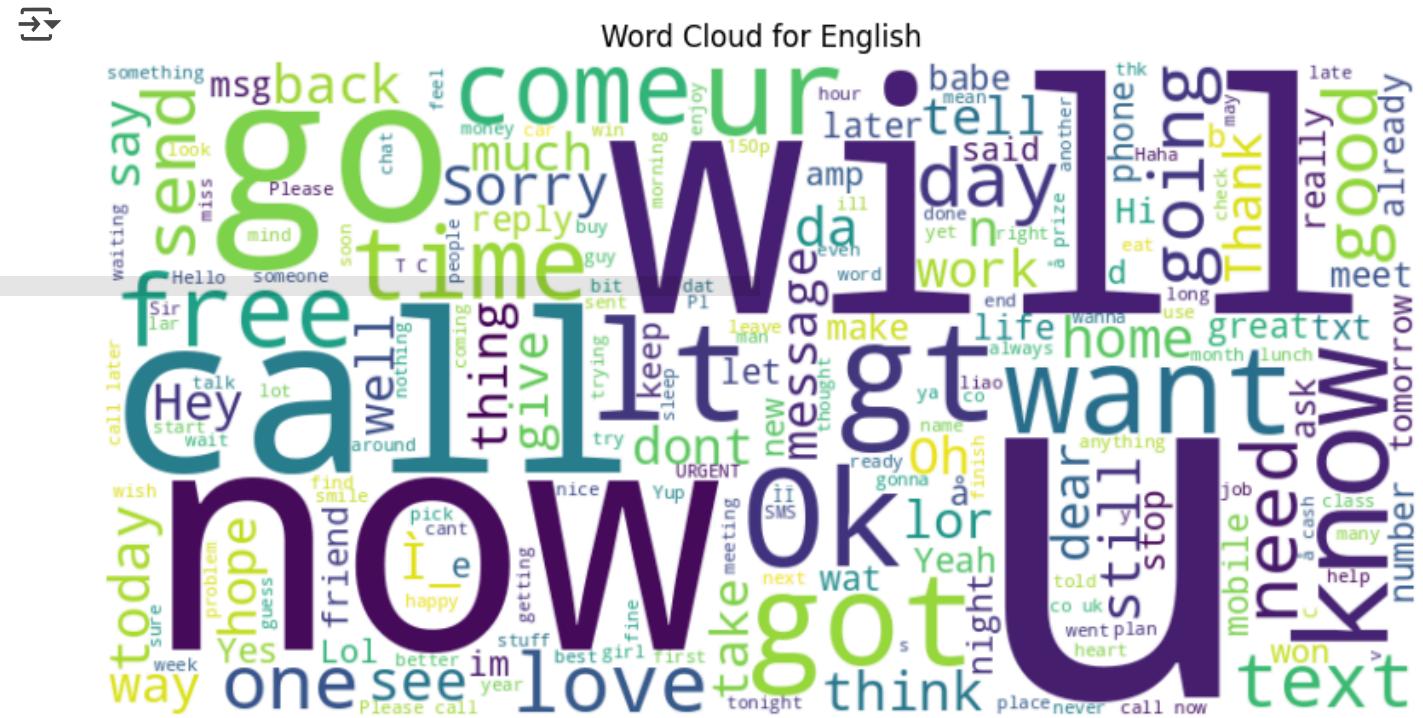
	category	text	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN

```
1 ' '.join(newdf['text'].head(2))
```

```
→ 'Go until jurong point, crazy.. Available only in bugis n great world la e buffet...
Cine there got amore wat... Ok lar... Joking wif u oni...'
```

```
1 import matplotlib.pyplot as plt
2
3 from wordcloud import WordCloud
4
5 text_corpus = ' '.join(newdf['text'])
6
```

```
7 wordcloud = WordCloud(width=800, height=400, random_state=42
8
9 # #This creates a new figure (or canvas) for the plot, with
10 #This function is used to create a new figure(or the overall
11 plt.figure(figsize=(10, 5))
12
13
14 ##This displays the word cloud image. The wordcloud object c
15 #interpolation='bilinear': This smooths the image when it is
16 plt.imshow(wordcloud, interpolation='bilinear')
17
18 ##this removes the axis from the plot, so no axis lines or la
19 plt.axis('off')
20
21 ##Adds a title to the plot, in this case, "Word Cloud for Te
22 plt.title('Word Cloud for English')
23
24 # Displays the plot with the word cloud.
25 plt.show()
```



2. Before you can use this data for modeling, you typically need to preprocess the text data.

Common steps include:

3. Lowercasing

4. Removing punctuation

5. Removing stop words

6. Tokenization

7. Converting text to numerical features (e.g., using Bag of Words or TF-IDF)

```
1 text="YES Sir"  
2 print(text.lower())  
3
```

→ yes sir

```
1 s=r'\n Yes sir'  
2 print(s)  
3 #t= '\n'  
4 #print(t)
```

→ \n Yes sir

```
1 t= '\n yes sir, we are sleeping'  
2 print(t)
```

→ yes sir, we are sleeping

```
1 import re  
2 text='&&&&akhtar age is 14 and??? !!!;;;;,,saeed age is 15  
3  
4 text1=re.sub(r'[^a-zA-Z0-9\s]', '', text)  
5  
6 text1
```

→ 'akhtar age is 14 and saeed age is 15 are friends but they usually disagree on certain matters'

```
1 text1=text1.split(" ")
2 text1
```

→ ['akhtar',
'age',
'is',
'14',
'and',
'saeed',
'age',
'is',
'15',
'are',
'friends',
'but',
'they',
'usually',
'disagree',
'on',
'certain',
'matters']

1 Start coding or generate with AI.

```
1 from nltk.corpus import stopwords
2 stop = stopwords.words('english')
3 len(stop)
4
5
```

→ 179

```
1 text1
```

→ ['akhtar',
'age',
'is',
'14',
'and',
'saeed',
'age',
'is',
'15',
'are',
'friends',
'but',
'they',
'usually',
'disagree',

```
'on',
'certain',
'matters']
```

```
1 cleantext=[x for x in text1 if x not in stop]
2 cleantext
```

```
→ ['akhtar',
'age',
'14',
'saeed',
'age',
'15',
'friends',
'usually',
'disagree',
'certain',
'matters']
```

```
1 text= ['akhtar', 'age', 'is', '14', 'and', '15']
2 print(" ".join(text))
```

```
→ akhtar age is 14 and 15
```

```
1 newdf.head(10)
```

		text category
0	Go until jurong point, crazy.. Available only ...	ham
1	Ok lar... Joking wif u oni...	ham
2	Free entry in 2 a wkly comp to win FA Cup fina...	spam
3	U dun say so early hor... U c already then say...	ham
4	Nah I don't think he goes to usf, he lives aro...	ham
5	FreeMsg Hey there darling it's been 3 week's n...	spam
6	Even my brother is not like to speak with me. ...	ham
7	As per your request 'Melle Melle (Oru Minnamin...	ham
8	WINNER!! As a valued network customer you have...	spam
9	Had your mobile 11 months or more? U R entitle...	spam

```
1 import re
2 from sklearn.model_selection import train_test_split
```

```
3 from sklearn.feature_extraction.text import CountVectorizer
4
5 # Function to preprocess text
6 def preprocess_text(text):
7     # Lowercase the text
8     text = text.lower()
9     # Remove punctuation and non-alphanumeric characters
10    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
11    text = text.split(" ")
12
13    text = [x for x in text if x not in stop]
14    return " ".join(text)
15
16 # Apply preprocessing to the Message column
17 newdf['Cleaned_Text'] = newdf['text'].apply(preprocess_text)
18
19 # Display the cleaned messages
20 print(newdf[['text', 'Cleaned_Text']].head(6))
21
```

→

	text \	Cleaned_Text
0	Go until jurong point, crazy.. Available only ...	go jurong point crazy available bugis n great ...
1	Ok lar... Joking wif u oni...	ok lar joking wif u oni
2	Free entry in 2 a wkly comp to win FA Cup fina...	free entry 2 wkly comp win fa cup final tkts 2...
3	U dun say so early hor... U c already then say...	u dun say early hor u c already say
4	Nah I don't think he goes to usf, he lives aro...	nah dont think goes usf lives around though
5	FreeMsg Hey there darling it's been 3 week's n...	freemsg hey darling 3 weeks word back id like ...

C:\Users\Dr. Atif Khan\AppData\Local\Temp\ipykernel\_13412\2816856476.py:17: SettingWithC  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#inplace-operations](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#inplace-operations)

◀ ━━━━━━ ▶

```
1 # Display the cleaned messages
2 print(df[['Cleaned_Text']].head(2))
```

```
→ Cleaned_Text  
0 go jurong point crazy available bugs n great ...  
1 ok lar joking wif u oni
```

## ✗ Notebook: 25) Spam\_ClassificationP3-14-May.ipynb

```
1 !pip cache purge # Sometimes the local cache may cause issue
```

```
→ ERROR: Too many arguments
```

```
1 import numpy as np  
2 import pandas as pd  
3 import itertools  
4 from sklearn.model_selection import train_test_split  
5 from sklearn.feature_extraction.text import TfidfVectorizer  
6 from sklearn.linear_model import PassiveAggressiveClassifier  
7 from sklearn.linear_model import LogisticRegression  
8 from sklearn.svm import SVC  
9 from sklearn.ensemble import RandomForestClassifier  
10 from sklearn.metrics import accuracy_score, confusion_matrix
```

```
1 #The issue you're encountering is due to the fact that the s  
2 #Instead of pip install sklearn, you need to install scikit-  
3 #!pip install sklearn
```

```
1 !pip install scikit-learn  
2
```

```
→ Requirement already satisfied: scikit-learn in c:\users\dr. atif khan\appdata\local\prog  
Requirement already satisfied: numpy>=1.19.5 in c:\users\dr. atif khan\appdata\local\prc  
Requirement already satisfied: scipy>=1.6.0 in c:\users\dr. atif khan\appdata\local\prog  
Requirement already satisfied: joblib>=1.2.0 in c:\users\dr. atif khan\appdata\local\prc  
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\dr. atif khan\appdata\lc
```

```
[notice] A new release of pip is available: 24.0 -> 25.1.1  
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
1 The error you're encountering, UnicodeDecodeError, typically
```

→ Cell In[22], line 1

The error you're encountering, `UnicodeDecodeError`, typically occurs when the CSV file contains characters that aren't properly decoded using the default encoding (`utf-8`).

SyntaxError: invalid syntax

```
1 import pandas as pd
```

```
1 pwd
```

```
1 df=pd.read_csv('spam.csv', encoding='latin1') # #Latin-1 is ...
2 df.shape #The re...
3
```

→ (5572, 5)

```
1 #Get shape and head
2 print(df.head(10))
```

	v1	v2	Unnamed: 2	\
0	ham Go until jurong point, crazy.. Available only ...		NaN	
1	ham Ok lar... Joking wif u oni...		NaN	
2	spam Free entry in 2 a wkly comp to win FA Cup fina...		NaN	
3	ham U dun say so early hor... U c already then say...		NaN	
4	ham Nah I don't think he goes to usf, he lives aro...		NaN	
5	spam FreeMsg Hey there darling it's been 3 week's n...		NaN	
6	ham Even my brother is not like to speak with me. ...		NaN	
7	ham As per your request 'Melle Melle (Oru Minnamin...		NaN	
8	spam WINNER!! As a valued network customer you have...		NaN	
9	spam Had your mobile 11 months or more? U R entitle...		NaN	

	Unnamed: 3	Unnamed: 4
0	NaN	NaN
1	NaN	NaN
2	NaN	NaN
3	NaN	NaN
4	NaN	NaN
5	NaN	NaN
6	NaN	NaN
7	NaN	NaN
8	NaN	NaN
9	NaN	NaN

```
1
```

```
2 pd.set_option('display.max_columns', None)
3 pd.set_option('display.expand_frame_repr', True) #True (defa
```

```
4 #False: |  
5 pd.set_option('max_colwidth',None) #Setting max_colwidth to  
6 df.head(10)
```

```
1 pwd
```

```
1 df = df.rename(columns={'v1': 'category', 'v2': 'text'})  
2 df  
3
```

	category	text	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN
...	...	...	...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...	NaN	NaN	NaN
5568	ham	Will l_b going to esplanade fr home?	NaN	NaN	NaN
5569	ham	Pity, * was in mood for that. So...any other	NaN	NaN	NaN

```
1 newdf=df[['text', 'category']]  
2 newdf
```



		text	category
0		Go until jurong point, crazy.. Available only ...	ham
1		Ok lar... Joking wif u oni...	ham
2		Free entry in 2 a wkly comp to win FA Cup fina...	spam
3		U dun say so early hor... U c already then say...	ham
4		Nah I don't think he goes to usf, he lives aro...	ham
...		...	...
5567		This is the 2nd time we have tried 2 contact u...	spam
5568		Will l_b going to esplanade fr home?	ham
5569		Pity, * was in mood for that. So...any other s...	ham
5570		The guy did some bitching but I acted like i'd...	ham
5571		Rofl. Its true to its name	ham

5572 rows × 2 columns

```
1 CORPUS = [
2 'the sky is blue',
3 'sky is blue and sky is beautiful',
4 'the beautiful sky is so blue',
5 'i love blue cheese'
6 ]
7 new_doc = ['loving this blue sky today']
```

```
1 from sklearn.feature_extraction.text import CountVectorizer
2
3 # Define the corpus
4 CORPUS = [
5     'the sky is blue',
6     'sky is blue and sky is beautiful',
7     'the beautiful sky is so blue',
8     'i love blue cheese'
9 ]
10 vectorizer = CountVectorizer(ngram_range=(1,1))
11
12     # Fit on corpus and transform both corpus and new docume
13
```

```
14 X_corpus = vectorizer.fit_transform(CORPUS)
15
16 #Vocabulary Lookup, You can see which word corresponds to wh
17 print(vectorizer.vocabulary_)
18
19 print(X_corpus)
20 # The output is a sparse matrix produced by countvectorizer
21 #Each line represents a non-zero entry in the term-document i
22 #In the tuple format (doc_index, word_index), the second val
23 # 2 → "blue"
24
25 # 8 → "the"
26
27 # 4 → "is"
28
29 # So in (0, 2) 1, it means the word "blue" appears once in d
```

```
→ { 'the': 8, 'sky': 6, 'is': 4, 'blue': 2, 'and': 0, 'beautiful': 1, 'so': 7, 'love': 5, '
  (0, 8)      1
  (0, 6)      1
  (0, 4)      1
  (0, 2)      1
  (1, 6)      2
  (1, 4)      2
  (1, 2)      1
  (1, 0)      1
  (1, 1)      1
  (2, 8)      1
  (2, 6)      1
  (2, 4)      1
  (2, 2)      1
  (2, 1)      1
  (2, 7)      1
  (3, 2)      1
  (3, 5)      1
  (3, 3)      1
```



```
1 from sklearn.feature_extraction.text import CountVectorizer
2
3 # Define the corpus
4 CORPUS = [
5     'the sky is blue',
6     'sky is blue and sky is beautiful',
7     'the beautiful sky is so blue',
```

```
8     'i love blue cheese'
9 ]
10
11 #new_doc = ['loving this blue sky today']
12
13 # Define a function to extract n-grams
14 def extract_ngrams(corpus, ngram_range=(1,1)):
15
16     vectorizer = CountVectorizer(ngram_range=ngram_range)
17
18     # Fit on corpus and transform both corpus and new docume
19
20     X_corpus = vectorizer.fit_transform(corpus)
21     #X_new_doc = vectorizer.transform(new_doc)
22
23
24
```

```
1 corpus_unigrams = extract_ngrams(CORPUS, ngram_range=(1, 1))
```

```
1 def extract_ngrams(corpus, ngram_range=(1,1)):
2
3     vectorizer = CountVectorizer(ngram_range=ngram_range)
4
5     # Fit on corpus and transform both corpus and new docume
6
7     X_corpus = vectorizer.fit_transform(corpus)
8     #X_new_doc = vectorizer.transform(new_doc)
9 # Get feature names (n-grams)
10    feature_names = vectorizer.get_feature_names_out()
11
12    # Convert the results to arrays
13    corpus_ngrams = X_corpus.toarray()
14    #new_doc_ngrams = X_new_doc.toarray()
15
16    return feature_names, corpus_ngrams
17
18 # Unigrams (1-gram)
```

```
19 unigrams, corpus_unigrams = extract_ngrams(CORPUS, ngram_range)
20 print("Unigrams:")
21 print(unigrams)
22 print("\n")
23 print("corpus matrix is\n", corpus_unigrams)
24 #print(new_doc_unigrams)
25
```

→ Unigrams:  
['and' 'beautiful' 'blue' 'cheese' 'is' 'love' 'sky' 'so' 'the']

```
corpus matrix is
[[0 0 1 0 1 0 1 0 1]
[1 1 1 0 2 0 2 0 0]
[0 1 1 0 1 0 1 1 1]
[0 0 1 1 0 1 0 0 0]]
```

```
1 # Bigrams (2-grams)
2 bigrams, corpus_bigrams = extract_ngrams(CORPUS, ngram_range)
3 print("\nBigrams:")
4 print(bigrams)
5 print(corpus_bigrams)
6 #print(new_doc_bigrams)
7
8 # Trigrams (3-grams)
9 trigrams, corpus_trigrams= extract_ngrams(CORPUS, ngram_range)
10 print("\nTrigrams:")
11 print(trigrams)
12 print(corpus_trigrams)
13 #print(new_doc_trigrams)
14
```

→ Bigrams:  
['and sky' 'beautiful sky' 'blue and' 'blue cheese' 'is beautiful'
 'is blue' 'is so' 'love blue' 'sky is' 'so blue' 'the beautiful'
 'the sky']  
[[0 0 0 0 0 1 0 0 1 0 0 1]
[1 0 1 0 1 1 0 0 2 0 0 0]
[0 1 0 0 0 0 1 0 1 1 1 0]
[0 0 0 1 0 0 0 1 0 0 0 0]]

Trigrams:  
['and sky is' 'beautiful sky is' 'blue and sky' 'is blue and' 'is so blue'
 'love blue cheese' 'sky is beautiful' 'sky is blue' 'sky is so'

```
'the beautiful sky' 'the sky is']
[[0 0 0 0 0 0 1 0 0 1]
 [1 0 1 1 0 0 1 1 0 0 0]
 [0 1 0 0 1 0 0 0 1 1 0]
 [0 0 0 0 1 0 0 0 0 0]]
```

```
1 import pandas as pd
2 from sklearn.feature_extraction.text import CountVectorizer
3
4 # Define the corpus and new document
5 CORPUS = [
6     'the sky is blue',
7     'sky is blue and sky is beautiful',
8     'the beautiful sky is so blue',
9     'i love blue cheese'
10 ]
11
12 new_doc = ['loving this blue sky today']
13
14 # Define a function to extract n-grams and return them as a DataFrame
15 def extract_ngrams_df(corpus, new_doc, ngram_range=(1,1)):
16     vectorizer = CountVectorizer(ngram_range=ngram_range)
17
18     # Fit on corpus and transform both corpus and new document
19     X_corpus = vectorizer.fit_transform(corpus)
20     print(type(X_corpus))
21     print(X_corpus)
22     X_new_doc = vectorizer.transform(new_doc)
23
24     # Get feature names (n-grams)
25     feature_names = vectorizer.get_feature_names_out()
26     print("\n\n features names are: ",feature_names)
27
28     features = X_corpus.toarray()                      #toarray() and toarray()
29     # Convert the results to arrays
30     print("*****")
31     print("\n features are ",X_corpus.todense()) # dense matrix
32     print("\n features are\n",features) # dense matrix representation
33     print("\n\n")
```

```
35     corpus_ngrams = X_corpus.toarray().sum(axis=0) # Sum co
36     #new_doc_ngrams = X_new_doc.toarray().flatten() # Flatt
37
38     # Create a DataFrame for better visualization
39     df1 = pd.DataFrame(data=features,
40                         columns=feature_names)
41
42     df2 = pd.DataFrame({
43         'N-gram': feature_names,
44         'Corpus Count': corpus_ngrams,
45
46     })
47
48
49
50     return df1, df2
51
52 # Unigrams (1-gram)
53 df_unigrams1, df_unigrams2 = extract_ngrams_df(CORPUS, new_d
54 print("\n\nUnigrams1 DataFrame:")
55 print(df_unigrams1)
56 print("\n\nUnigrams2 DataFrame:")
57 print(df_unigrams2)
58
59
```

```
→ <class 'scipy.sparse._csr.csr_matrix'>
(0, 8)      1
(0, 6)      1
(0, 4)      1
(0, 2)      1
(1, 6)      2
(1, 4)      2
(1, 2)      1
(1, 0)      1
(1, 1)      1
(2, 8)      1
(2, 6)      1
(2, 4)      1
(2, 2)      1
(2, 1)      1
(2, 7)      1
(3, 2)      1
(3, 5)      1
(3, 3)      1
```

```

features names are: ['and' 'beautiful' 'blue' 'cheese' 'is' 'love' 'sky' 'so' 'the'
*****
features are [[0 0 1 0 1 0 1 0 1]
[1 1 1 0 2 0 2 0 0]
[0 1 1 0 1 0 1 1 1]
[0 0 1 1 0 1 0 0 0]]

features are
[[0 0 1 0 1 0 1 0 1]
[1 1 1 0 2 0 2 0 0]
[0 1 1 0 1 0 1 1 1]
[0 0 1 1 0 1 0 0 0]]

```

Unigrams1 DataFrame:

	and	beautiful	blue	cheese	is	love	sky	so	the	
0	0		0	1	0	1	0	1	0	1
1	1		1	1	0	2	0	2	0	0
2	0		1	1	0	1	0	1	1	1
3	0		0	1	1	0	1	0	0	0

Unigrams2 DataFrame:

	N-gram	Corpus	Count
0	and		1
1	beautiful		2
2	blue		4
3	cheese		1
4	is		4
5	love		1
6	sky		4
7	--		1

```

1 # Define a function to extract n-grams and return them as a |
2 def extract_ngrams_df(corpus, new_doc, ngram_range=(1,1)):
3     vectorizer = CountVectorizer(ngram_range=ngram_range)
4
5     # Fit on corpus and transform both corpus and new docume
6     X_corpus = vectorizer.fit_transform(corpus)
7     print(type(X_corpus))
8     print(X_corpus)
9     X_new_doc = vectorizer.transform(new_doc)
10
11    # Get feature names (n-grams)

```

```

12     feature_names = vectorizer.get_feature_names_out()
13     print("\n\n features names are: ",feature_names)
14
15     features = X_corpus.toarray()                      #toarray() and t
16     # Convert the results to arrays
17     print("*****")
18     print("\n features are ",X_corpus.todense()) # dense mat
19     print("\n features are\n",features) # dense matrix repre
20     print("\n\n")
21
22     corpus_ngrams = X_corpus.toarray().sum(axis=0) # Sum co
23     #new_doc_ngrams = X_new_doc.toarray().flatten() # Flatt
24
25     # Create a DataFrame for better visualization
26     df1 = pd.DataFrame(data=features,
27                         columns=feature_names)
28
29     df2 = pd.DataFrame({
30         'N-gram': feature_names,
31         'Corpus Count': corpus_ngrams,
32
33     })
34
35
36
37     return df1, df2
38

```

```

1 # Bigrams (2-grams)
2 df_bigrams1, df_bigrams2 = extract_ngrams_df(CORPUS, new_doc
3 print("\nBigrams1 DataFrame:")
4 print(df_bigrams1)
5
6 print("\nBigrams2 DataFrame:")
7 print(df_bigrams2)
8
9
10

```

```

→ <class 'scipy.sparse._csr.csr_matrix'>
(0, 11)      1
(0, 8)       1
(0, 5)       1
(1, 8)       2
(1, 5)       1
(1, 2)       1
(1, 0)       1
(1, 4)       1
(2, 8)       1
(2, 10)      1
(2, 1)       1
(2, 6)       1
(2, 9)       1
(3, 7)       1
(3, 3)       1

features names are: ['and sky' 'beautiful sky' 'blue and' 'blue cheese' 'is beautif
'is blue' 'is so' 'love blue' 'sky is' 'so blue' 'the beautiful'
'the sky']
*****
features are [[0 0 0 0 0 1 0 0 1 0 0 1]
[1 0 1 0 1 1 0 0 2 0 0 0]
[0 1 0 0 0 0 1 0 1 1 1 0]
[0 0 0 1 0 0 0 1 0 0 0 0]]
features are
[[0 0 0 0 0 1 0 0 1 0 0 1]
[1 0 1 0 1 1 0 0 2 0 0 0]
[0 1 0 0 0 0 1 0 1 1 1 0]
[0 0 0 1 0 0 0 1 0 0 0 0]]

```

**Bigrams1 DataFrame:**

	and	sky	beautiful	sky	blue	and	blue	cheese	is	beautiful	is	blue	\
0	0		0		0		0	0	0	0	0	1	
1	1		0		1		0	0	0	1	1	1	
2	0		1		0		0	0	0	0	0	0	
3	0		0		0		0	1	0	0	0	0	

	is	so	love	blue	sky	is	so	blue	the	beautiful	the	sky
0	0		0	1	0		0	0	0	0	1	
1	0		0	2	0		0	0	0	0	0	
2	1		0	1	1		1	1	1	0	0	
3	0		1	0	0		0	0	0	0	0	

**Bigrams2 DataFrame:**

	N-gram	Corpus	Count
0	and sky		1
1	beautiful sky		1
2	blue and		1
3	blue cheese		1

```
4    is beautiful
```

```
1
```

```
1 # Trigrams (3-grams)
2 df_trigrams = extract_ngrams_df(CORPUS, new_doc, ngram_range
3 print("\nTrigrams DataFrame:")
4 print(df_trigrams)
```

```
→ <class 'scipy.sparse._csr.csr_matrix'>
(0, 10)      1
(0, 7)       1
(1, 7)       1
(1, 3)       1
(1, 2)       1
(1, 0)       1
(1, 6)       1
(2, 9)       1
(2, 1)       1
(2, 8)       1
(2, 4)       1
(3, 5)       1
```

```
features names are: ['and sky is' 'beautiful sky is' 'blue and sky' 'is blue and' '
'love blue cheese' 'sky is beautiful' 'sky is blue' 'sky is so'
'the beautiful sky' 'the sky is']
```

```
*****
```

```
features are [[0 0 0 0 0 0 0 1 0 0 1]
[1 0 1 1 0 0 1 1 0 0 0]
[0 1 0 0 1 0 0 0 1 1 0]
[0 0 0 0 0 1 0 0 0 0 0]]
```

```
features are
[[0 0 0 0 0 0 0 1 0 0 1]
[1 0 1 1 0 0 1 1 0 0 0]
[0 1 0 0 1 0 0 0 1 1 0]
[0 0 0 0 0 1 0 0 0 0 0]]
```

```
Trigrams DataFrame:
```

```
( and sky is beautiful sky is blue and sky is blue and is so blue \
0      0          0          0          0          0
1      1          0          1          1          0
2      0          1          0          0          1
3      0          0          0          0          0
```

```
love blue cheese sky is beautiful sky is blue sky is so \
0            0          0          1          0
1            0          1          1          0
2            0          0          0          1
3            1          0          0          0
```

```
the beautiful sky  the sky is
0                  0      1
1                  0      0
2                  1      0
3                  0      0 ,          N-gram  Corpus Count
0      and sky is      1
1  beautiful sky is      1
2      blue and sky      1
3      is blue and      1
4      :s -s h1...      1
```

```
1 arr=[
2     [1, 1, 1, 1],  # Document 1
3     [0, 2, 1, 1],  # Document 2
4     [1, 1, 1, 1]   # Document 3
5 ]
6
```

```
1 arr.torray().sum(axis=0) #will sum the counts for each unigr
```

```
→ -----  
AttributeError                                     Traceback (most recent call last)
Cell In[67], line 1
----> 1 arr.torray().sum(axis=0) #will sum the counts for each unigram (column-wise)
across all documents:  
  
AttributeError: 'list' object has no attribute 'torray'
```

```
1 from scipy.sparse import csr_matrix
2 import numpy as np
3
4 # Define non-zero values and their coordinates
5 data = np.array([3, 4, 5])                      # Non-zero values
6 row_indices = np.array([0, 1, 2])                # Row indices
7 col_indices = np.array([2, 0, 1])                # Column indices
8
9 # Create a sparse matrix
10 sparse_matrix = csr_matrix((data, (row_indices, col_indices))
11
12 # Display the sparse matrix
13 print(sparse_matrix)
14
```

```
1 from scipy.sparse import csr_matrix
2
3 # Define a dense 2D list (matrix)
4 dense_array = [
5     [0, 0, 3],
6     [4, 0, 0],
7     [0, 5, 0]
8 ]
9
10 # Create a sparse matrix
11 sparse_matrix = csr_matrix(dense_array)
12
13 # Display the sparse matrix
14 print(sparse_matrix)
15
```

```
→ (0, 2)      3
    (1, 0)      4
    (2, 1)      5
```

```
1 sparse_matrix.toarray()
```

```
→ array([[0, 0, 3],
          [4, 0, 0],
          [0, 5, 0]], dtype=int32)
```

```
1 sparse_matrix.toarray().sum(axis=0)
```

```
→ array([4, 5, 3], dtype=int32)
```

```
1 Start coding or generate with AI.
```

```
1
2 import pandas as pd
3 df=pd.read_csv('spam.csv', encoding='latin1') # #Latin-1 is ...
4 df.shape
```

```
→ (5572, 5)
```

```
1 df = df.rename(columns={'v1': 'category', 'v2': 'text'})
2 newdf=df[['text', 'category']]
```

### 3 newdf

→

		text	category
0		Go until jurong point, crazy.. Available only ...	ham
1		Ok lar... Joking wif u oni...	ham
2		Free entry in 2 a wkly comp to win FA Cup fina...	spam
3		U dun say so early hor... U c already then say...	ham
4		Nah I don't think he goes to usf, he lives aro...	ham
...		...	...
5567		This is the 2nd time we have tried 2 contact u...	spam
5568		Will l_b going to esplanade fr home?	ham
5569		Pity, * was in mood for that. So...any other s...	ham
5570		The guy did some bitching but I acted like i'd...	ham
5571		Rofl. Its true to its name	ham

5572 rows × 2 columns

### Step 2: Data Analysis

1. Check for Missing Values: It's essential to check if there are any missing values in your DataFrame.
2. Class Distribution: Check how many messages are "ham" and how many are "spam."

```
1 #number of missing values (NaN or None) in each column of the
2 print(newdf['text'].isnull().sum(axis=0))
```

→ 0

```
1 print(newdf.isnull().sum())
2
```

→ text 0
category 0
dtype: int64

```
1 #Class Distribution: Check how many messages are "ham" and h
2 #prints the frequency (count) of each unique value in the 'L
```

```
3 print(df['category'].value_counts())
4
```

```
→ category
ham      4825
spam     747
Name: count, dtype: int64
```

The ax object is used later to plot the bars using the ax.bar() method.

This creates a blank figure (canvas) and a set of axes (where the plot will appear).

figsize=(6, 6) sets the size of the plot (6 inches by 6 inches).

```
1 !pip install matplotlib
```

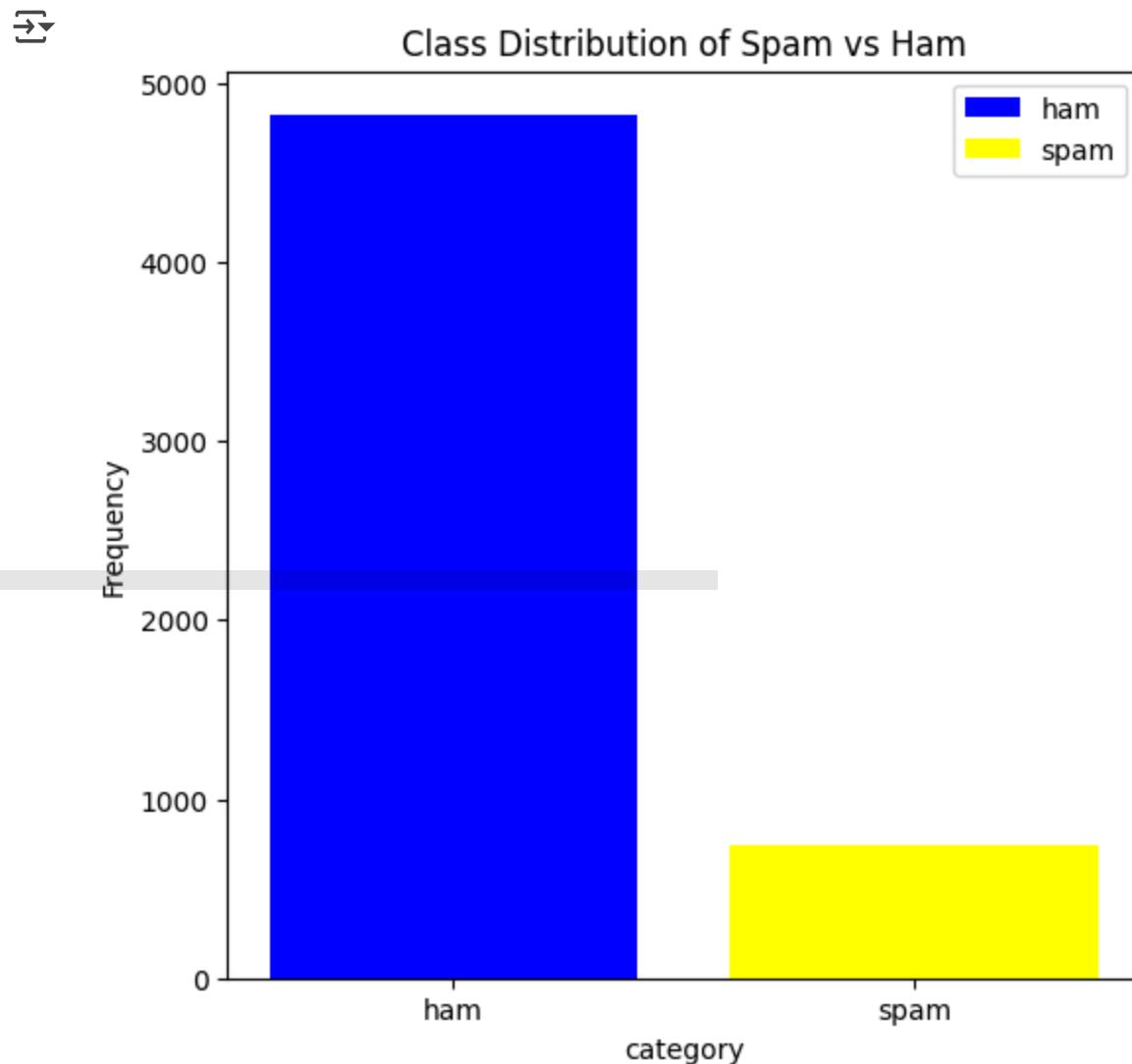
```
→ Requirement already satisfied: matplotlib in c:\users\dr. atif khan\appdata\local\programs\python\python37\lib\site-packages
Requirement already satisfied: contourpy>=1.0.1 in c:\users\dr. atif khan\appdata\local\programs\python\python37\lib\site-packages\contourpy\_
Requirement already satisfied: cycler>=0.10 in c:\users\dr. atif khan\appdata\local\programs\python\python37\lib\site-packages\cycl_
Requirement already satisfied: fonttools>=4.22.0 in c:\users\dr. atif khan\appdata\local\programs\python\python37\lib\site-pac_
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\dr. atif khan\appdata\local\programs\python\python37\lib\site-pac_
Requirement already satisfied: numpy<2,>=1.21 in c:\users\dr. atif khan\appdata\local\programs\python\python37\lib\site-pac_
Requirement already satisfied: packaging>=20.0 in c:\users\dr. atif khan\appdata\local\programs\python\python37\lib\site-pac_
Requirement already satisfied: pillow>=8 in c:\users\dr. atif khan\appdata\local\programs\python\python37\lib\site-pac_
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\dr. atif khan\appdata\local\programs\python\python37\lib\site-pac_
Requirement already satisfied: python-dateutil>=2.7 in c:\users\dr. atif khan\appdata\local\programs\python\python37\lib\site-pac_
Requirement already satisfied: six>=1.5 in c:\users\dr. atif khan\appdata\local\programs\python\python37\lib\site-pac
```

[notice] A new release of pip is available: 24.0 -> 25.1.1

[notice] To update, run: python.exe -m pip install --upgrade pip

```
1 #The legend is the small box (usually on the side or top of the plot)
2 #showing the color associated with each class.
3
4 import matplotlib.pyplot as plt #library, which is used to create plots
5
6 # Bar Plot for Class Distribution
7 #This counts how many times each category (like "ham" and "spam") appears
8 class_distribution = newdf['category'].value_counts()
9
10 # Create a figure and axis
11 fig, ax = plt.subplots(figsize=(6, 6))
12
```

```
13 # Plot each bar separately to associate them with labels
14 #Draws a blue bar for the first category (likely 'ham') usin
15 ax.bar(class_distribution.index[0], class_distribution.value
16 #Draws a yellow bar for the first category (likely 'spam') u
17 ax.bar(class_distribution.index[1], class_distribution.value
18
19 # Add title and labels
20 ax.set_title('Class Distribution of Spam vs Ham')
21 ax.set_xlabel('category')
22 ax.set_ylabel('Frequency')
23
24 # Add a legend with the correct labels
25 ax.legend(loc='upper right')
26
27 # Show the plot
28 plt.show()
```





```
1 newdf['text'].head(10)
```

```
→ 0 Go until jurong point, crazy.. Available only ...
  1 Ok lar... Joking wif u oni...
  2 Free entry in 2 a wkly comp to win FA Cup fina...
  3 U dun say so early hor... U c already then say...
  4 Nah I don't think he goes to usf, he lives aro...
  5 FreeMsg Hey there darling it's been 3 week's n...
  6 Even my brother is not like to speak with me. ...
  7 As per your request 'Melle Melle (Oru Minnamin...
  8 WINNER!! As a valued network customer you have...
  9 Had your mobile 11 months or more? U R entitle...
Name: text, dtype: object
```

```
1 ' '.join(newdf['text'].head(10))
```

```
→ "Go until jurong point, crazy.. Available only in bugis n great world la e buffet...
Cine there got amore wat... Ok lar... Joking wif u oni... Free entry in 2 a wkly comp
to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std
txt rate)T&C's apply 08452810075over18's U dun say so early hor... U c already then
say... Nah I don't think he goes to usf, he lives around here though FreeMsg Hey there
darling it's been 3 week's now and no word back! I'd like some fun you up for it still?
Tb ok! XxX std chgs to send, £1.50 to rcv Even my brother is not like to speak with
me. They treat me like aids patent. As per your request 'Melle Melle (Oru
Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press
*9 to copy your friends Callertune WINNER!! As a valued network customer you have been
selected to receivea £900 prize reward! To claim call 09061701461. Claim code KL341.
Valid 12 hours only. Had your mobile 11 months or more? U R entitled to Update to the
latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on
08002986030"
```

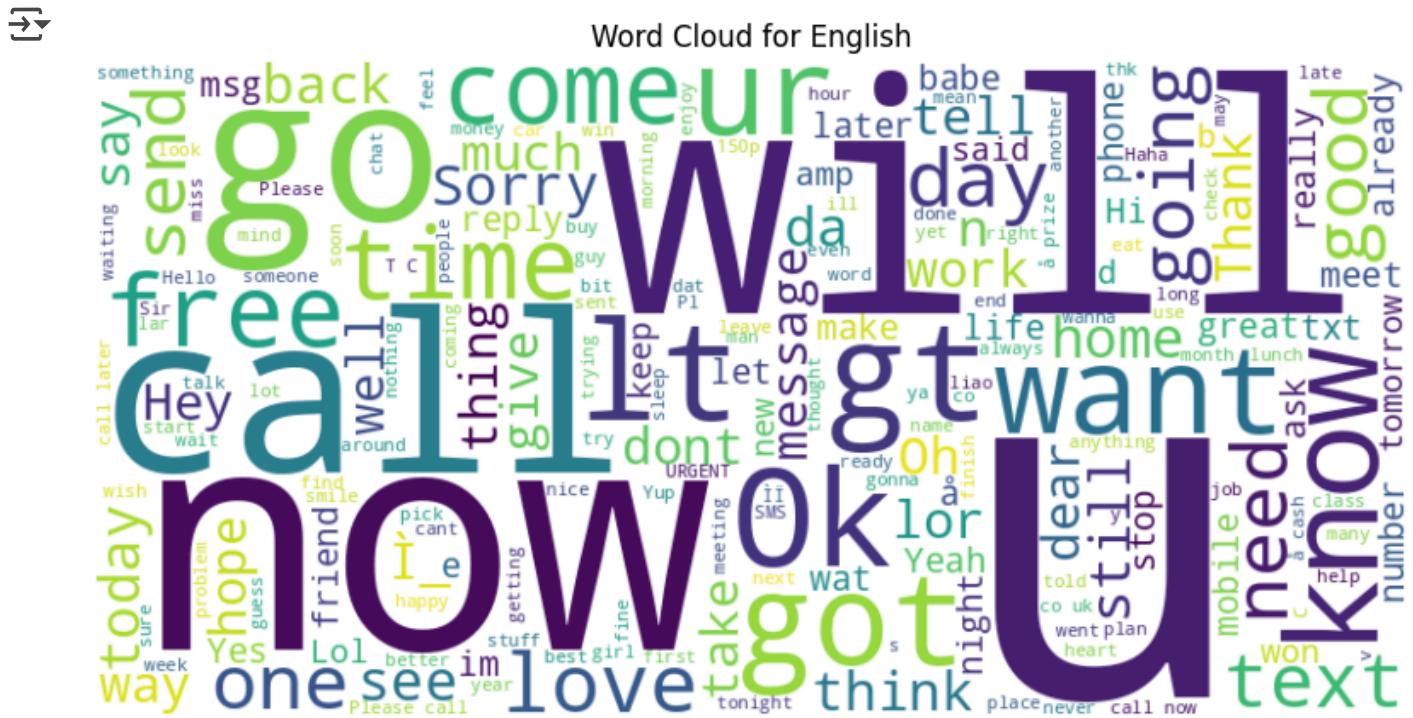
```
1 df.head(2)
```

	category	text	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN

```
1 ' '.join(newdf['text'].head(2))
```

```
→ 'Go until jurong point, crazy.. Available only in bugis n great world la e buffet...
Cine there got amore wat... Ok lar... Joking wif u oni...'
```

```
1 import matplotlib.pyplot as plt
2
3 from wordcloud import WordCloud
4
5 text_corpus = ' '.join(newdf['text'])
6
7 wordcloud = WordCloud(width=800, height=400, random_state=42
8
9 # #This creates a new figure (or canvas) for the plot, with
10 #This function is used to create a new figure(or the overall
11 plt.figure(figsize=(10, 5))
12
13
14 ##This displays the word cloud image. The wordcloud object c
15 #interpolation='bilinear': This smooths the image when it is
16 plt.imshow(wordcloud, interpolation='bilinear')
17
18 ##his removes the axis from the plot, so no axis lines or la
19 plt.axis('off')
20
21 ##Adds a title to the plot, in this case, "Word Cloud for Te
22 plt.title('Word Cloud for English')
23
24 # Displays the plot with the word cloud.
25 plt.show()
```



## 1. Step 3: Text Preprocessing

2. Before you can use this data for modeling, you typically need to preprocess the text data.

Common steps include:

### 3. Lowercasing

## 4. Removing punctuation

## 5. Removing stop words

## 6. Tokenization

7. Converting text to numerical features (e.g., using Bag of Words or TF-IDF)

```
1 text="YES Sir"  
2 print(text.lower())  
3
```

→ yes sir

```
1 s=r'\n Yes sir'  
2 print(s)
```

```
3 #t= '\n'  
4 #print(t)
```

→ \n Yes sir

```
1 t= '\n yes sir, we are sleeping'  
2 print(t)
```

→ yes sir, we are sleeping

```
1 import re  
2 text='&&&&&akhtar age is 14 and??? !!!;;;;,saeed age is 15  
3  
4 text1=re.sub(r'^[a-zA-Z0-9\s]', '', text)  
5  
6 text1
```

→ 'akhtar age is 14 and saeed age is 15 are friends but they usually disagree on certain matters'

```
1 text1=text1.split(" ")  
2 text1
```

→ ['akhtar',  
 'age',  
 'is',  
 '14',  
 'and',  
 'saeed',  
 'age',  
 'is',  
 '15',  
 'are',  
 'friends',  
 'but',  
 'they',  
 'usually',  
 'disagree',  
 'on',  
 'certain',  
 'matters']

1 Start coding or generate with AI.

```
1 from nltk.corpus import stopwords  
2 stop = stopwords.words('english')  
3 len(stop)  
4  
5
```

→ 198

```
1 text1
```

→ [ 'akhtar',  
 'age',  
 'is',  
 '14',  
 'and',  
 'saeed',  
 'age',  
 'is',  
 '15',  
 'are',  
 'friends',  
 'but',  
 'they',  
 'usually',  
 'disagree',  
 'on',  
 'certain',  
 'matters' ]

```
1 cleantext=[x for x in text1 if x not in stop]  
2 cleantext
```

→ [ 'akhtar',  
 'age',  
 '14',  
 'saeed',  
 'age',  
 '15',  
 'friends',  
 'usually',  
 'disagree',  
 'certain',  
 'matters' ]

```
1 text= ['akhtar', 'age', 'is', '14', 'and', '15']  
2 print(" ".join(text))
```

→ akhtar age is 14 and 15

```
1 newdf.head(10)
```

		text category
0	Go until jurong point, crazy.. Available only ...	ham
1	Ok lar... Joking wif u oni...	ham
2	Free entry in 2 a wkly comp to win FA Cup fina...	spam
3	U dun say so early hor... U c already then say...	ham
4	Nah I don't think he goes to usf, he lives aro...	ham
5	FreeMsg Hey there darling it's been 3 week's n...	spam
6	Even my brother is not like to speak with me. ...	ham
7	As per your request 'Melle Melle (Oru Minnamin...	ham
8	WINNER!! As a valued network customer you have...	spam
9	Had your mobile 11 months or more? U R entitle...	spam

## ▼ newdf.loc[:, 'Cleaned\_Text']

loc is used for label-based indexing.

: means all rows.

'Cleaned\_Text' is the name of the column you're assigning values to. select all rows in the column named 'Cleaned\_Text'.

```
1 import re
2 from sklearn.model_selection import train_test_split
3 from sklearn.feature_extraction.text import CountVectorizer
4
5 # Function to preprocess text
6 def preprocess_text(text):
7     # Lowercase the text
8     text = text.lower()
9     # Remove punctuation and non-alphanumeric characters
10    text = re.sub(r'^[a-zA-Z0-9\s]', ' ', text)
11    text = text.split(" ")
12
13    text = [x for x in text if x not in stop]
```

```

14     return " ".join(text)
15
16 # Apply preprocessing to the Message column
17 newdf.loc[:, 'Cleaned_Text'] = newdf['text'].apply(preprocess_text)
18
19 #newdf['Cleaned_Text'] = newdf['text'].apply(preprocess_text)
20
21 # Display the cleaned messages
22 print(newdf[['text', 'Cleaned_Text']].head(6))
23

```

```

→                                     text \
0 Go until jurong point, crazy.. Available only ...
1                      Ok lar... Joking wif u oni...
2 Free entry in 2 a wkly comp to win FA Cup fina...
3 U dun say so early hor... U c already then say...
4 Nah I don't think he goes to usf, he lives aro...
5 FreeMsg Hey there darling it's been 3 week's n...

                                     Cleaned_Text
0 go jurong point crazy available bugis n great ...
1                      ok lar joking wif u oni
2 free entry 2 wkly comp win fa cup final tkts 2...
3                      u dun say early hor u c already say
4                      nah dont think goes usf lives around though
5 freemsg hey darling 3 weeks word back id like ...

```

```

1 # Display the cleaned messages
2 print(newdf[['Cleaned_Text']].head(2))

```

```

→                                     Cleaned_Text
0 go jurong point crazy available bugis n great ...
1                      ok lar joking wif u oni

```

Step 4: Feature Extraction You can use CountVectorizer to convert the cleaned text data into numerical format.

```

1 # Create features and labels
2 X = newdf['Cleaned_Text']
3 y = newdf['category']
4
5 # Split data into training and testing sets
6 X_train, X_test, y_train, y_test = train_test_split(X, y, te
7

```

```
1 # Vectorize the text data
2 vectorizer = CountVectorizer()
3 X_train_vectorized = vectorizer.fit_transform(X_train)
4
```

```
1 X_test_vectorized = vectorizer.transform(X_test)
2
3 print(X_train_vectorized.shape)
4 print(X_test_vectorized.shape)
5 print(X_train_vectorized[0].toarray())
```

```
→ (4457, 8218)
(1115, 8218)
[[0 0 0 ... 0 0 0]]
```

```
1 from sklearn.naive_bayes import MultinomialNB
2 from sklearn.metrics import classification_report
3
4 # Initialize and train the classifier
5 classifier = MultinomialNB()
6 classifier.fit(X_train_vectorized, y_train)
7
8 # Make predictions
9 y_pred = classifier.predict(X_test_vectorized)
10
11 # Evaluate the model
12 # Evaluate the model
13 report=classification_report(y_test, y_pred)
14
15 print(report)
16 print("#####")
17 #conf_matrix = confusion_matrix(y_test, y_pred)
18 #print(conf_matrix)
```

```
→
```

	precision	recall	f1-score	support
ham	0.98	1.00	0.99	965
spam	0.97	0.88	0.92	150
accuracy			0.98	1115
macro avg	0.98	0.94	0.96	1115

```
weighted avg      0.98      0.98      0.98      1115
```

```
#####
#####
```

```
1 conf_matrix = confusion_matrix(y_test, y_pred)
2 print(conf_matrix)
```

```
→ [[961  4]
 [ 18 132]]
```

```
1 # Calculate metrics
2 report = classification_report(y_test, y_pred, output_dict=True)
3 print(report)
4 print("\n*****")
5 accuracy = accuracy_score(y_test, y_pred)
```

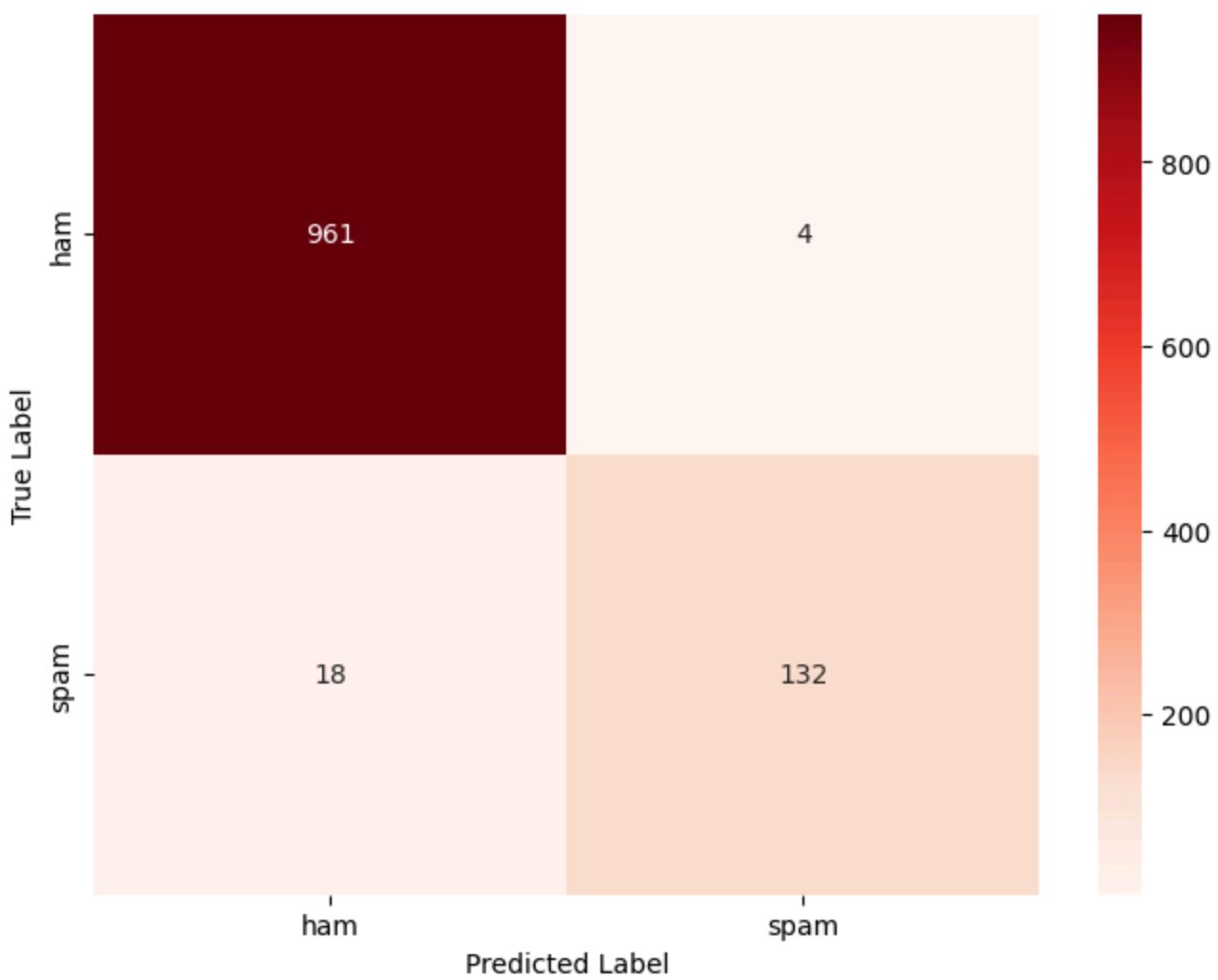
```
→ {'ham': {'precision': 0.9816138917262512, 'recall': 0.9958549222797928, 'f1-score': 0.9816138917262512}}
```



```
1 # Generate the confusion matrix
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 conf_matrix = confusion_matrix(y_test, y_pred)
5
6 # Create a heatmap for the confusion matrix
7 plt.figure(figsize=(8, 6))
8 sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Reds', xticklabels=['Predicted Label'],
9 xticklabels=['Predicted Label'])
10 plt.ylabel('True Label')
11 plt.title('Confusion Matrix')
12 plt.show()
```



Confusion Matrix

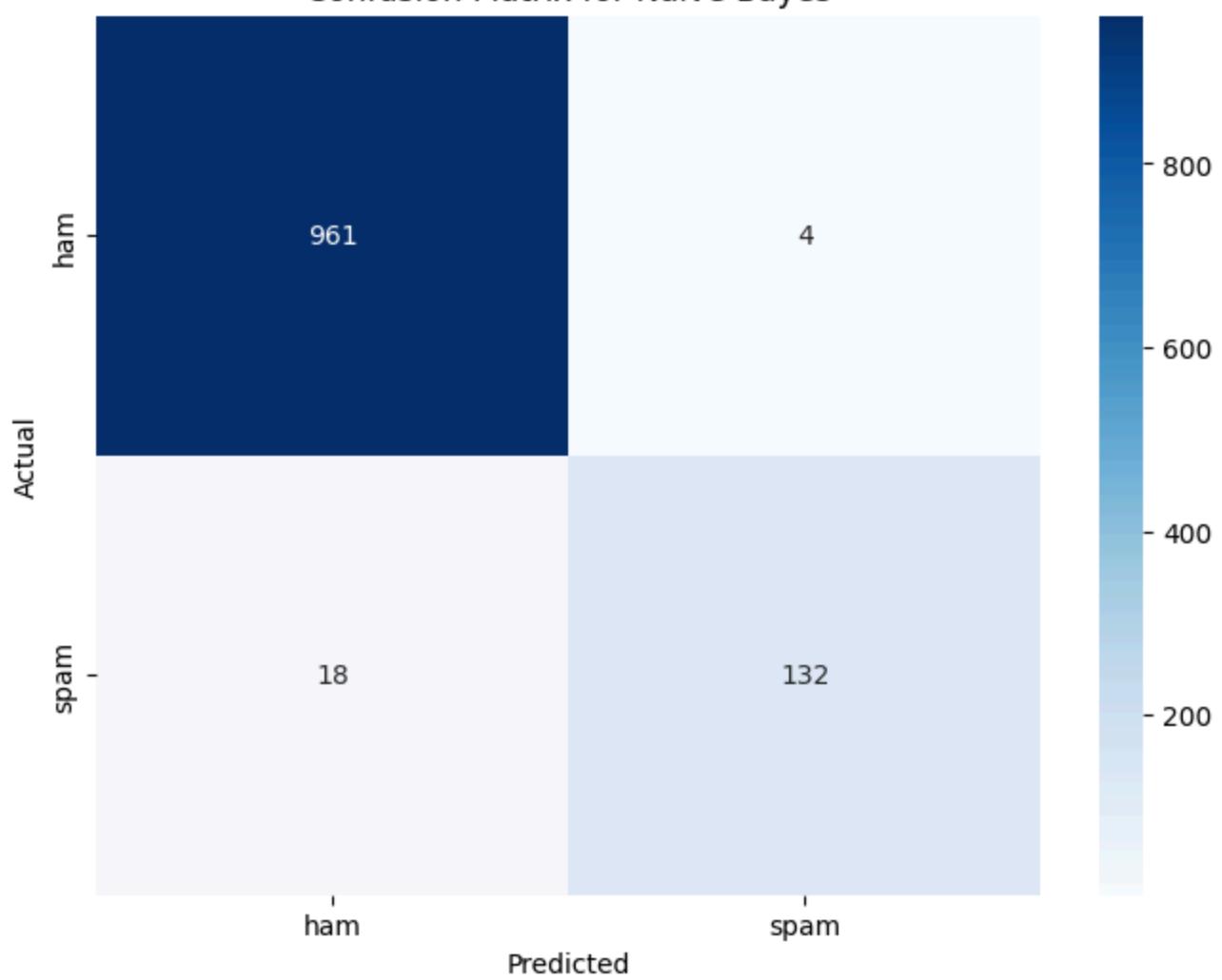


```
1
2
3 # Classifiers to evaluate
4 classifiers = {
5     'Naive Bayes': MultinomialNB(),
6     'Logistic Regression': LogisticRegression(),
7     'Support Vector Machine': SVC(probability=True),
8     'Random Forest': RandomForestClassifier()
9 }
10
11 # Store metrics for plotting
12 all_metrics = {}
13
14 for name, classifier in classifiers.items():
15     # Initialize and train the classifier
```

```
16     classifier.fit(X_train_vectorized, y_train)
17
18     # Make predictions
19     y_pred = classifier.predict(X_test_vectorized)
20
21     # Calculate metrics
22     report = classification_report(y_test, y_pred, output_di
23     accuracy = accuracy_score(y_test, y_pred)
24
25     # Prepare data for the bar chart
26     metrics = {
27         'Precision': report['spam']['precision'],
28         'Recall': report['spam']['recall'],
29         'F1 Score': report['spam']['f1-score'],
30         'Accuracy': accuracy
31     }
32
33     # Store metrics for each classifier
34     all_metrics[name] = metrics
35
36     # Create confusion matrix
37     cm = confusion_matrix(y_test, y_pred, labels=['ham', 'sp
38
39     # Create heatmap for confusion matrix
40     plt.figure(figsize=(8, 6))
41     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xtick
42     plt.ylabel('Actual')
43     plt.xlabel('Predicted')
44     plt.title(f'Confusion Matrix for {name}')
45     plt.show()
46
47     # Print the classification report for each classifier
48     print(f"\n{name} Classification Report:")
49     print(classification_report(y_test, y_pred))
50
51 print("dictionary ",all_metrics)
52
```



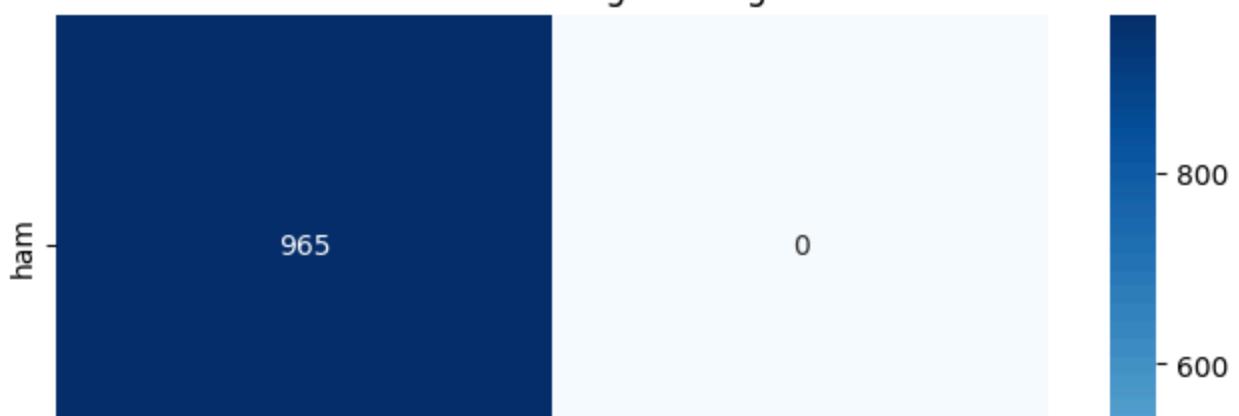
Confusion Matrix for Naive Bayes

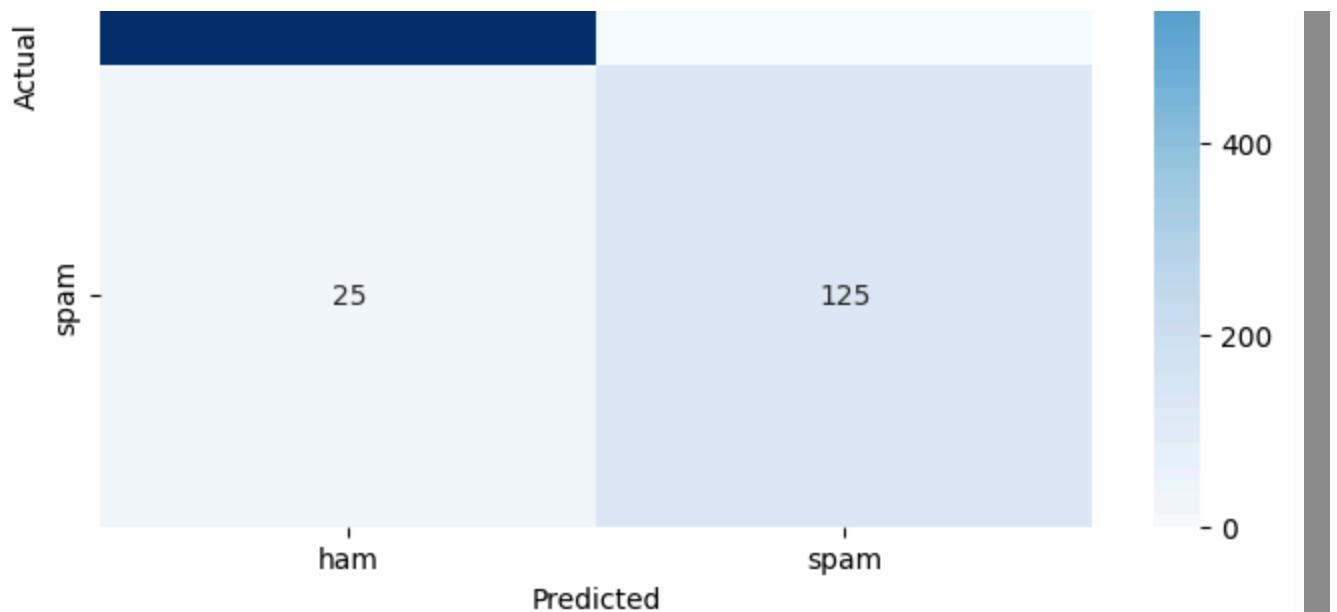


Naive Bayes Classification Report:

	precision	recall	f1-score	support
ham	0.98	1.00	0.99	965
spam	0.97	0.88	0.92	150
accuracy			0.98	1115
macro avg	0.98	0.94	0.96	1115
weighted avg	0.98	0.98	0.98	1115

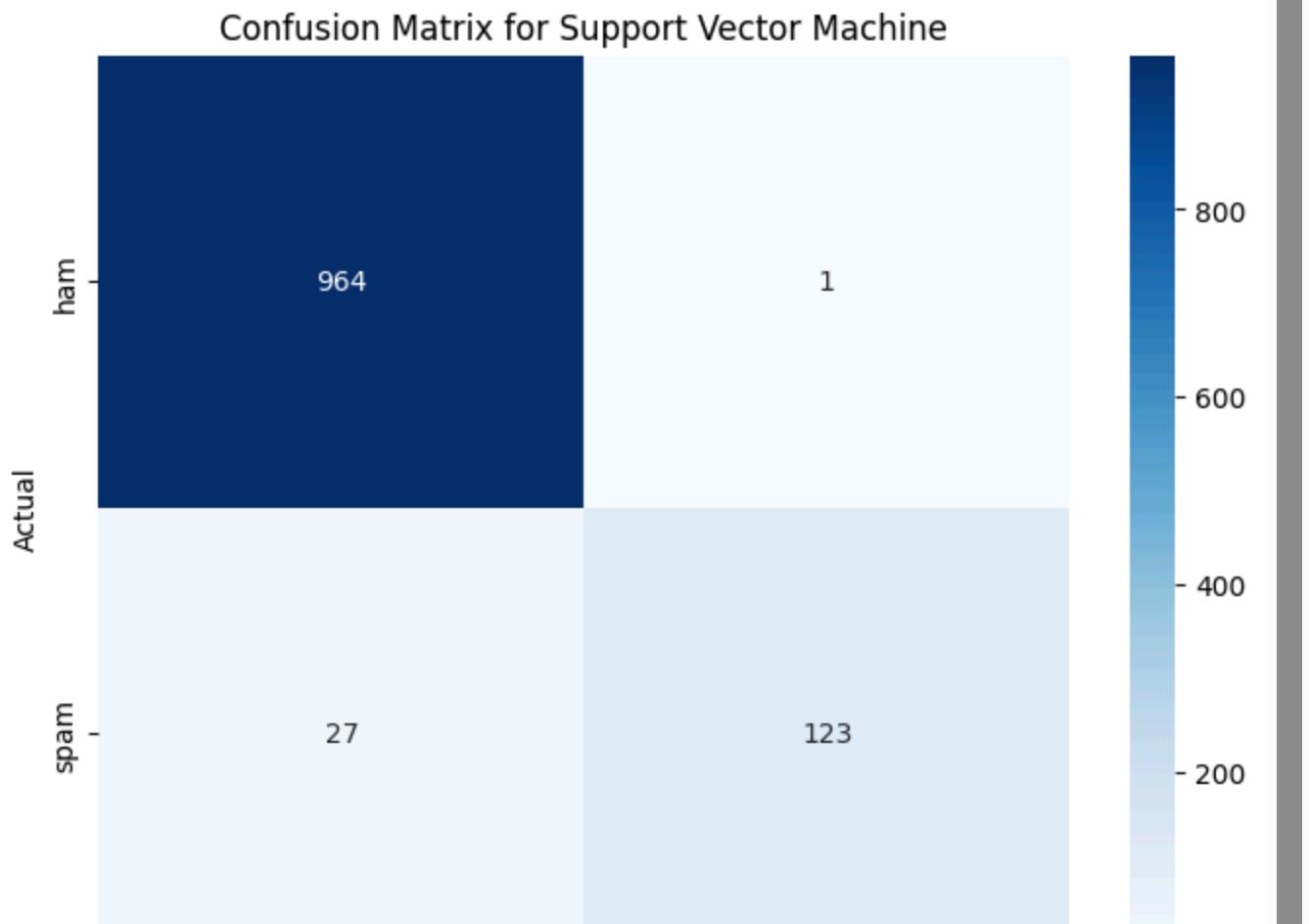
Confusion Matrix for Logistic Regression





Logistic Regression Classification Report:

	precision	recall	f1-score	support
ham	0.97	1.00	0.99	965
spam	1.00	0.83	0.91	150
accuracy			0.98	1115
macro avg	0.99	0.92	0.95	1115
weighted avg	0.98	0.98	0.98	1115



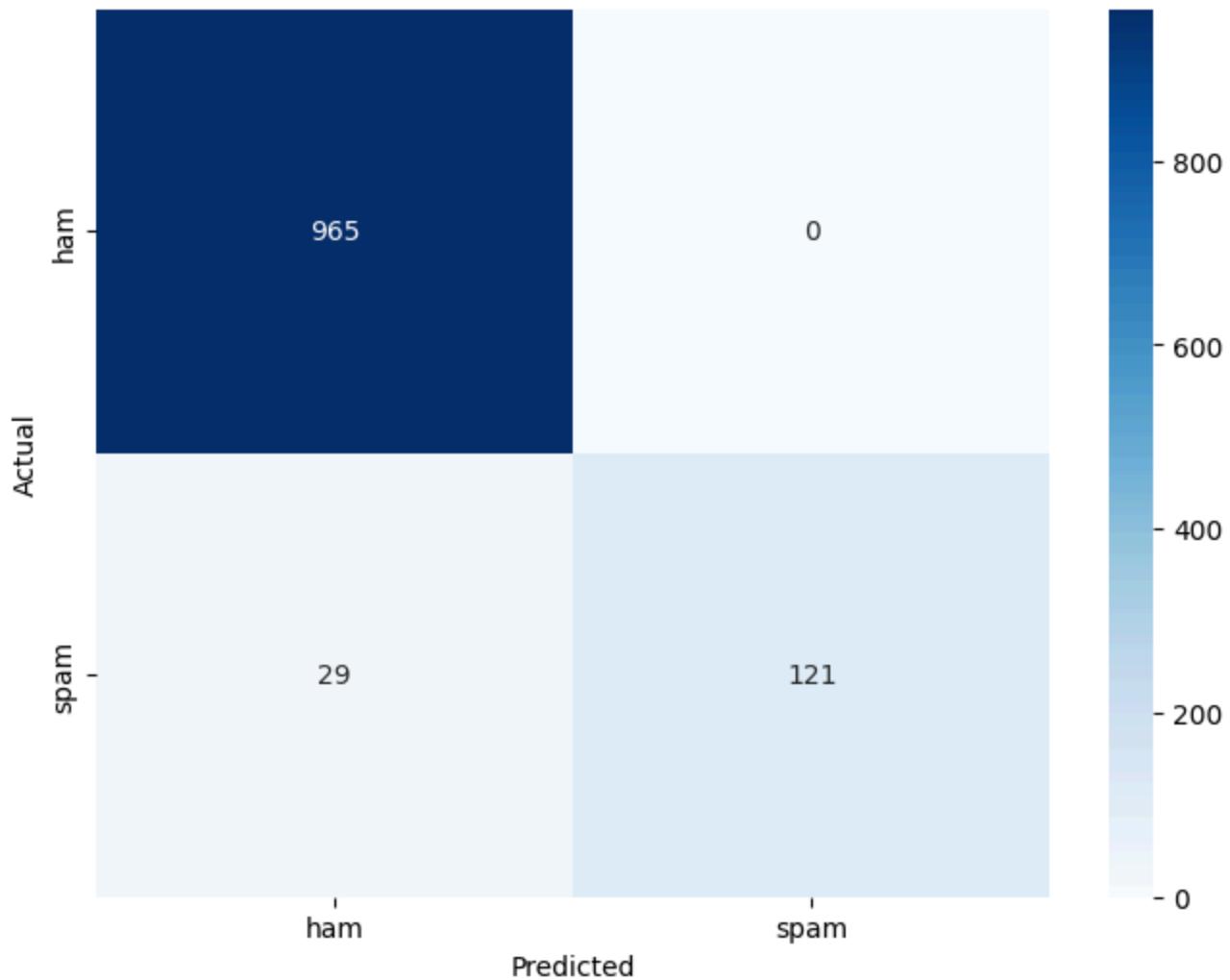
ham spam

Predicted

Support Vector Machine Classification Report:

	precision	recall	f1-score	support
ham	0.97	1.00	0.99	965
spam	0.99	0.82	0.90	150
accuracy			0.97	1115
macro avg	0.98	0.91	0.94	1115
weighted avg	0.98	0.97	0.97	1115

Confusion Matrix for Random Forest



Random Forest Classification Report:

	precision	recall	f1-score	support
ham	0.97	1.00	0.99	965
spam	1.00	0.81	0.89	150
accuracy			0.97	1115
macro avg	0.99	0.90	0.94	1115
weighted avg	0.97	0.97	0.97	1115

```
dictionary {'Naive Bayes': {'Precision': 0.9705882352941176, 'Recall': 0.88, 'F1 Scc
```



1 Start coding or generate with AI.

## ▼ Notebook: 26) Spam\_Classification-16-May.ipynb

```
1 !pip cache purge # Sometimes the local cache may cause issue
```

→ ERROR: Too many arguments

```
1 import numpy as np
2 import pandas as pd
3 import itertools
4 from sklearn.model_selection import train_test_split
5 from sklearn.feature_extraction.text import TfidfVectorizer
6 from sklearn.linear_model import PassiveAggressiveClassifier
7 from sklearn.linear_model import LogisticRegression
8 from sklearn.svm import SVC
9 from sklearn.ensemble import RandomForestClassifier
10 from sklearn.metrics import accuracy_score, confusion_matrix
```

```
1 #The issue you're encountering is due to the fact that the s
2 #Instead of pip install sklearn, you need to install scikit-
3 #!pip install sklearn
```

```
1 !pip install scikit-learn
2
```

→ Requirement already satisfied: scikit-learn in c:\users\dell\anaconda3\lib\site-packages
Requirement already satisfied: numpy>=1.17.3 in c:\users\dell\anaconda3\lib\site-packages
Requirement already satisfied: scipy>=1.5.0 in c:\users\dell\anaconda3\lib\site-packages
Requirement already satisfied: joblib>=1.1.1 in c:\users\dell\anaconda3\lib\site-packages
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\dell\anaconda3\lib\site-

1 The error you're encountering, UnicodeDecodeError, typically

→ Cell In[13], line 1

The error you're encountering, `UnicodeDecodeError`, typically occurs when the CSV file contains characters that aren't properly decoded using the default encoding (`utf-8`).

SyntaxError: invalid syntax

```
1 import pandas as pd
```

```
1 pwd
```

→ 'E:\\Python For Advance Application\\notebooks'

```
1 df=pd.read_csv('spam.csv', encoding='latin1') # #Latin-1 is ...
2 df.shape #The re...
3
```

→ (5572, 5)

```
1 #Get shape and head
2 print(df.head(10))
```

→

	v1	v2	Unnamed: 2	\
0	ham Go until jurong point, crazy.. Available only ...	NaN	NaN	
1	ham Ok lar... Joking wif u oni...	NaN	NaN	
2	spam Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	
3	ham U dun say so early hor... U c already then say...	NaN	NaN	
4	ham Nah I don't think he goes to usf, he lives aro...	NaN	NaN	
5	spam FreeMsg Hey there darling it's been 3 week's n...	NaN	NaN	
6	ham Even my brother is not like to speak with me. ...	NaN	NaN	
7	ham As per your request 'Melle Melle (Oru Minnamin...	NaN	NaN	
8	spam WINNER!! As a valued network customer you have...	NaN	NaN	
9	spam Had your mobile 11 months or more? U R entitle...	NaN	NaN	

Unnamed: 3 Unnamed: 4

0	NaN	NaN
1	NaN	NaN
2	NaN	NaN
3	NaN	NaN
4	NaN	NaN
5	NaN	NaN
6	NaN	NaN
7	NaN	NaN
8	NaN	NaN
9	NaN	NaN

```
1
2 pd.set_option('display.max_columns', None)
3 pd.set_option('display.expand_frame_repr', True) #True (defa
4                                         #False: |
5 pd.set_option('max_colwidth',None)  #Setting max_colwidth to
6 df.head(10)
```

→

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives around here though	NaN	NaN	NaN
5	spam	FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, £1.50 to rcv	NaN	NaN	NaN
6	ham	Even my brother is not like to speak with me. They treat me like aids patient.	NaN	NaN	NaN
7	ham	As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 to copy your friends Callertune	NaN	NaN	NaN
		WINNER!! As a valued network customer you have been			

```
1 pwd
```

→ 'E:\\Python For Advance Application\\notebooks'

```
1 df = df.rename(columns={'v1': 'category', 'v2': 'text'})
2 df
3
```



## category

text Unnamed: 2 Unnamed: 3 Unnamed: 4

0	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives around here though	NaN	NaN	NaN
...	...	...	...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u. U have won the £750 Pound prize. 2 claim is easy, call 087187272008 NOW! Only 10p per minute. BT-national-rate.	NaN	NaN	NaN
5568	ham	Will I_b going to esplanade fr home?	NaN	NaN	NaN
5569	ham	Pity, * was in mood for that. So...any other suggestions?	NaN	NaN	NaN
5570	ham	The guy did some bitching but I acted like i'd be interested in buying something else next week	NaN	NaN	NaN

1 newdf=df[['text', 'category']]

2 newdf



text category

0	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	ham
1	Ok lar... Joking wif u oni...	ham
2	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	spam
3	U dun say so early hor... U c already then say...	ham
4	Nah I don't think he goes to usf, he lives around here though	ham
...	...	...
5567	This is the 2nd time we have tried 2 contact u. U have won the £750 Pound prize. 2 claim is easy, call 087187272008 NOW! Only 10p per minute. BT-national-rate.	spam
5568	Will I_ b going to esplanade fr home?	ham
5569	Pity, * was in mood for that. So...any other suggestions?	ham
5570	The guy did some bitching but I acted like i'd be interested in buying something else next week and he gave it to us for free	ham
5571	Rofl. Its true to its name	ham

5572 rows × 2 columns

```
1 CORPUS = [
2 'the sky is blue',
3 'sky is blue and sky is beautiful',
4 'the beautiful sky is so blue',
5 'i love blue cheese'
6 ]
7 new_doc = ['loving this blue sky today']
```

```
1 from sklearn.feature_extraction.text import CountVectorizer
2
3 # Define the corpus
4 CORPUS = [
5     'the sky is blue',
6     'sky is blue and sky is beautiful',
7     'the beautiful sky is so blue',
8     'i love blue cheese'
9 ]
10 vectorizer = CountVectorizer(ngram_range=(1,1))
11
```

```
12     # Fit on corpus and transform both corpus and new document
13
14 X_corpus = vectorizer.fit_transform(CORPUS)
15
16 #Vocabulary Lookup, You can see which word corresponds to what index
17 print(vectorizer.vocabulary_)
18
19 print(X_corpus)
20 # The output is a sparse matrix produced by countvectorizer
21 #Each line represents a non-zero entry in the term-document matrix
22 #In the tuple format (doc_index, word_index), the second value is the
23 # 2 → "blue"
24
25 # 8 → "the"
26
27 # 4 → "is"
28
29 # So in (0, 2) 1, it means the word "blue" appears once in document 0
```

```
→ { 'the': 8, 'sky': 6, 'is': 4, 'blue': 2, 'and': 0, 'beautiful': 1, 'so': 7, 'love': 5, 'brown': 3 }
    (0, 8)      1
    (0, 6)      1
    (0, 4)      1
    (0, 2)      1
    (1, 6)      2
    (1, 4)      2
    (1, 2)      1
    (1, 0)      1
    (1, 1)      1
    (2, 8)      1
    (2, 6)      1
    (2, 4)      1
    (2, 2)      1
    (2, 1)      1
    (2, 7)      1
    (3, 2)      1
    (3, 5)      1
    (3, 3)      1
```

```
1 from sklearn.feature_extraction.text import CountVectorizer
2
3 # Define the corpus
4 CORPUS = [
5     'the sky is blue',
```

```
6     'sky is blue and sky is beautiful',
7     'the beautiful sky is so blue',
8     'i love blue cheese'
9 ]
10
11 #new_doc = ['loving this blue sky today']
12
13 # Define a function to extract n-grams
14 def extract_ngrams(corpus, ngram_range=(1,1)):
15
16     vectorizer = CountVectorizer(ngram_range=ngram_range)
17
18     # Fit on corpus and transform both corpus and new docume
19
20     X_corpus = vectorizer.fit_transform(corpus)
21     #X_new_doc = vectorizer.transform(new_doc)
22
23
24
```

```
1 corpus_unigrams = extract_ngrams(CORPUS, ngram_range=(1, 1))
```

```
1 def extract_ngrams(corpus, ngram_range=(1,1)):
2
3     vectorizer = CountVectorizer(ngram_range=ngram_range)
4
5     # Fit on corpus and transform both corpus and new docume
6
7     X_corpus = vectorizer.fit_transform(corpus)
8     #X_new_doc = vectorizer.transform(new_doc)
9 # Get feature names (n-grams)
10    feature_names = vectorizer.get_feature_names_out()
11
12    # Convert the results to arrays
13    corpus_ngrams = X_corpus.toarray()
14    #new_doc_ngrams = X_new_doc.toarray()
15
16    return feature_names, corpus_ngrams
```

```
17
18 # Unigrams (1-gram)
19 unigrams, corpus_unigrams = extract_ngrams(CORPUS, ngram_range)
20 print("Unigrams:")
21 print(unigrams)
22 print("\n")
23 print("corpus matrix is\n", corpus_unigrams)
24 #print(new_doc_unigrams)
25
```

```
→ Unigrams:
['and' 'beautiful' 'blue' 'cheese' 'is' 'love' 'sky' 'so' 'the']
```

```
corpus matrix is
[[0 0 1 0 1 0 1 0 1]
 [1 1 1 0 2 0 2 0 0]
 [0 1 1 0 1 0 1 1 1]
 [0 0 1 1 0 1 0 0 0]]
```

```
1 # Bigrams (2-grams)
2 bigrams, corpus_bigrams = extract_ngrams(CORPUS, ngram_range)
3 print("\nBigrams:")
4 print(bigrams)
5 print(corpus_bigrams)
6 #print(new_doc_bigrams)
7
8 # Trigrams (3-grams)
9 trigrams, corpus_trigrams= extract_ngrams(CORPUS, ngram_range)
10 print("\nTrigrams:")
11 print(trigrams)
12 print(corpus_trigrams)
13 #print(new_doc_trigrams)
14
```

```
→
Bigrams:
['and sky' 'beautiful sky' 'blue and' 'blue cheese' 'is beautiful'
 'is blue' 'is so' 'love blue' 'sky is' 'so blue' 'the beautiful'
 'the sky']
[[0 0 0 0 1 0 0 1 0 0 1]
 [1 0 1 0 1 1 0 0 2 0 0]
 [0 1 0 0 0 0 1 0 1 1 0]
 [0 0 0 1 0 0 0 1 0 0 0]]
```

Trigrams:

```
['and sky is' 'beautiful sky is' 'blue and sky' 'is blue and' 'is so blue'
 'love blue cheese' 'sky is beautiful' 'sky is blue' 'sky is so'
 'the beautiful sky' 'the sky is']
[[0 0 0 0 0 0 1 0 0 1]
 [1 0 1 1 0 0 1 1 0 0 0]
 [0 1 0 0 1 0 0 0 1 1 0]
 [0 0 0 0 0 1 0 0 0 0 0]]
```

```
1 import pandas as pd
2 from sklearn.feature_extraction.text import CountVectorizer
3
4 # Define the corpus and new document
5 CORPUS = [
6     'the sky is blue',
7     'sky is blue and sky is beautiful',
8     'the beautiful sky is so blue',
9     'i love blue cheese'
10 ]
11
12 new_doc = ['loving this blue sky today']
13
14 # Define a function to extract n-grams and return them as a DataFrame
15 def extract_ngrams_df(corpus, new_doc, ngram_range=(1,1)):
16     vectorizer = CountVectorizer(ngram_range=ngram_range)
17
18     # Fit on corpus and transform both corpus and new document
19     X_corpus = vectorizer.fit_transform(corpus)
20     print(type(X_corpus))
21     print(X_corpus)
22     X_new_doc = vectorizer.transform(new_doc)
23
24     # Get feature names (n-grams)
25     feature_names = vectorizer.get_feature_names_out()
26     print("\n\n features names are: ",feature_names)
27
28     features = X_corpus.toarray()                      #toarray() and toarray()
29     # Convert the results to arrays
30     print("*****")
31     print("\n features are ",X_corpus.todense()) # dense matrix
32     print("\n features are\n",features) # dense matrix representation
```

```
33     print("\n\n")
34
35     corpus_ngrams = X_corpus.toarray().sum(axis=0) # Sum co
36     #new_doc_ngrams = X_new_doc.toarray().flatten() # Flatt
37
38     # Create a DataFrame for better visualization
39     df1 = pd.DataFrame(data=features,
40                         columns=feature_names)
41
42     df2 = pd.DataFrame({
43         'N-gram': feature_names,
44         'Corpus Count': corpus_ngrams,
45
46     })
47
48
49
50     return df1, df2
51
52 # Unigrams (1-gram)
53 df_unigrams1, df_unigrams2 = extract_ngrams_df(CORPUS, new_d
54 print("\n\nUnigrams1 DataFrame:")
55 print(df_unigrams1)
56 print("\n\nUnigrams2 DataFrame:")
57 print(df_unigrams2)
58
59
```

```
→ <class 'scipy.sparse._csr.csr_matrix'>
(0, 8)      1
(0, 6)      1
(0, 4)      1
(0, 2)      1
(1, 6)      2
(1, 4)      2
(1, 2)      1
(1, 0)      1
(1, 1)      1
(2, 8)      1
(2, 6)      1
(2, 4)      1
(2, 2)      1
(2, 1)      1
(2, 7)      1
```

```
(3, 2)      1
(3, 5)      1
(3, 3)      1
```

```
features names are:  ['and' 'beautiful' 'blue' 'cheese' 'is' 'love' 'sky' 'so' 'the'
*****
*****
```

```
features are [[0 0 1 0 1 0 1 0 1]
[1 1 1 0 2 0 2 0 0]
[0 1 1 0 1 0 1 1 1]
[0 0 1 1 0 1 0 0 0]]
```

```
features are
[[0 0 1 0 1 0 1 0 1]
[1 1 1 0 2 0 2 0 0]
[0 1 1 0 1 0 1 1 1]
[0 0 1 1 0 1 0 0 0]]
```

Unigrams1 DataFrame:

	and	beautiful	blue	cheese	is	love	sky	so	the
0	0	0	1	0	1	0	1	0	1
1	1	1	1	0	2	0	2	0	0
2	0	1	1	0	1	0	1	1	1
3	0	0	1	1	0	1	0	0	0

Unigrams2 DataFrame:

	N-gram	Corpus	Count
0	and	1	
1	beautiful	2	
2	blue	4	
3	cheese	1	
4	is	4	
5	love	1	
6	sky	4	
-		-	-

```
1 # Define a function to extract n-grams and return them as a |
2 def extract_ngrams_df(corpus, new_doc, ngram_range=(1,1)):
3     vectorizer = CountVectorizer(ngram_range=ngram_range)
4
5     # Fit on corpus and transform both corpus and new docume
6     X_corpus = vectorizer.fit_transform(corpus)
7     print(type(X_corpus))
8     print(X_corpus)
9     X_new_doc = vectorizer.transform(new_doc)
```

```

10
11     # Get feature names (n-grams)
12     feature_names = vectorizer.get_feature_names_out()
13     print("\n\n features names are: ",feature_names)
14
15     features = X_corpus.toarray()                      #toarray() and t
16     # Convert the results to arrays
17     print("*****")
18     print("\n features are ",X_corpus.todense()) # dense mat
19     print("\n features are\n",features) # dense matrix repre
20     print("\n\n")
21
22     corpus_ngrams = X_corpus.toarray().sum(axis=0) # Sum co
23     #new_doc_ngrams = X_new_doc.toarray().flatten() # Flatt
24
25     # Create a DataFrame for better visualization
26     df1 = pd.DataFrame(data=features,
27                         columns=feature_names)
28
29     df2 = pd.DataFrame({
30         'N-gram': feature_names,
31         'Corpus Count': corpus_ngrams,
32
33     })
34
35
36
37     return df1, df2
38

```

```

1 # Bigrams (2-grams)
2 df_bigrams1, df_bigrams2 = extract_ngrams_df(CORPUS, new_doc
3 print("\nBigrams1 DataFrame:")
4 print(df_bigrams1)
5
6 print("\nBigrams2 DataFrame:")
7 print(df_bigrams2)
8

```

9

10

```
→ <class 'scipy.sparse._csr.csr_matrix'>
(0, 11)      1
(0, 8)       1
(0, 5)       1
(1, 8)       2
(1, 5)       1
(1, 2)       1
(1, 0)       1
(1, 4)       1
(2, 8)       1
(2, 10)      1
(2, 1)       1
(2, 6)       1
(2, 9)       1
(3, 7)       1
(3, 3)       1
```

features names are: ['and sky' 'beautiful sky' 'blue and' 'blue cheese' 'is beautiful'  
'is blue' 'is so' 'love blue' 'sky is' 'so blue' 'the beautiful'  
'the sky']

\*\*\*\*\*

```
features are [[0 0 0 0 0 1 0 0 1 0 0 1]
[1 0 1 0 1 1 0 0 2 0 0 0]
[0 1 0 0 0 0 1 0 1 1 1 0]
[0 0 0 1 0 0 0 1 0 0 0 0]]
```

```
features are
[[0 0 0 0 0 1 0 0 1 0 0 1]
[1 0 1 0 1 1 0 0 2 0 0 0]
[0 1 0 0 0 0 1 0 1 1 1 0]
[0 0 0 1 0 0 0 1 0 0 0 0]]
```

Bigrams1 DataFrame:

	and sky	beautiful sky	blue and	blue cheese	is beautiful	is blue	\
0	0	0	0	0	0	0	1
1	1	0	1	0	1	1	1
2	0	1	0	0	0	0	0
3	0	0	0	1	0	0	0

	is so	love blue	sky is	so blue	the beautiful	the sky
0	0	0	1	0	0	1
1	0	0	2	0	0	0
2	1	0	1	1	1	0
3	0	1	0	0	0	0

Bigrams2 DataFrame:

N-gram Corpus Count

```
0      and sky      1
1  beautiful sky    1
2      blue and    1
3  blue cheese     1
.
```

```
1 # Trigrams (3-grams)
2 df_trigrams = extract_ngrams_df(CORPUS, new_doc, ngram_range
3 print("\nTrigrams DataFrame:")
4 print(df_trigrams)
```

```
→ <class 'scipy.sparse._csr.csr_matrix'>
(0, 10)      1
(0, 7)       1
(1, 7)       1
(1, 3)       1
(1, 2)       1
(1, 0)       1
(1, 6)       1
(2, 9)       1
(2, 1)       1
(2, 8)       1
(2, 4)       1
(3, 5)       1
```

```
features names are: ['and sky is' 'beautiful sky is' 'blue and sky' 'is blue and'
 'love blue cheese' 'sky is beautiful' 'sky is blue' 'sky is so'
 'the beautiful sky' 'the sky is']
```

```
*****
```

```
features are [[0 0 0 0 0 0 0 1 0 0 1]
[1 0 1 1 0 0 1 1 0 0 0]
[0 1 0 0 1 0 0 0 1 1 0]
[0 0 0 0 0 1 0 0 0 0 0]]
```

```
features are
[[0 0 0 0 0 0 0 1 0 0 1]
[1 0 1 1 0 0 1 1 0 0 0]
[0 1 0 0 1 0 0 0 1 1 0]
[0 0 0 0 0 1 0 0 0 0 0]]
```

```
Trigrams DataFrame:
(  and sky is  beautiful sky is  blue and sky  is blue and  is so blue \
0      0          0          0          0          0
1      1          0          1          1          0
2      0          1          0          0          1
3      0          0          0          0          0
```

```
love blue cheese  sky is beautiful  sky is blue  sky is so \
```

```

0          0          0          1          0
1          0          1          1          0
2          0          0          0          1
3          1          0          0          0

the beautiful sky  the sky is
0          0          1
1          0          0
2          1          0
3          0          0 ,           N-gram  Corpus Count
0          and sky is    1
1          beautiful sky is 1
2          blue and sky   1
3          is blue and     1

```

```

1 arr=[
2     [1, 1, 1, 1],  # Document 1
3     [0, 2, 1, 1],  # Document 2
4     [1, 1, 1, 1]   # Document 3
5 ]
6

```

```
1 arr.torray().sum(axis=0) #will sum the counts for each unigr
```

AttributeError Traceback (most recent call last)  
Cell In[107], line 1  
----> 1 arr.torray().sum(axis=0)

AttributeError: 'list' object has no attribute 'torray'

```

1 from scipy.sparse import csr_matrix
2 import numpy as np
3
4 # Define non-zero values and their coordinates
5 data = np.array([3, 4, 5])          # Non-zero values
6 row_indices = np.array([0, 1, 2])    # Row indices
7 col_indices = np.array([2, 0, 1])    # Column indices
8
9 # Create a sparse matrix
10 sparse_matrix = csr_matrix((data, (row_indices, col_indices))
11
12 # Display the sparse matrix

```

```
13 print(sparse_matrix)
```

```
14
```

```
→ (0, 2)      3  
  (1, 0)      4  
  (2, 1)      5
```

```
1 from scipy.sparse import csr_matrix  
2  
3 # Define a dense 2D list (matrix)  
4 dense_array = [  
5     [0, 0, 3],  
6     [4, 0, 0],  
7     [0, 5, 0]  
8 ]  
9  
10 # Create a sparse matrix  
11 sparse_matrix = csr_matrix(dense_array)  
12  
13 # Display the sparse matrix  
14 print(sparse_matrix)  
15
```

```
→ (0, 2)      3  
  (1, 0)      4  
  (2, 1)      5
```

```
1 sparse_matrix.toarray()
```

```
→ array([[0, 0, 3],  
        [4, 0, 0],  
        [0, 5, 0]], dtype=int32)
```

```
1 sparse_matrix.toarray().sum(axis=0)
```

```
→ array([4, 5, 3], dtype=int32)
```

```
1 Start coding or generate with AI.
```

```
1
```

```
2 import pandas as pd
```

```
3 df=pd.read_csv('spam.csv', encoding='latin1') # #Latin-1 is ...
4 df.shape
```

→ (5572, 5)

```
1 df = df.rename(columns={'v1': 'category', 'v2': 'text'})
2 newdf=df[['text', 'category']]
3 newdf
```

→

		text	category
0	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...		ham
1	Ok lar... Joking wif u oni...		ham
2	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's		spam
3	U dun say so early hor... U c already then say...		ham
4	Nah I don't think he goes to usf, he lives around here though		ham
...	...	...	...
5567	This is the 2nd time we have tried 2 contact u. U have won the £750 Pound prize. 2 claim is easy, call 087187272008 NOW! Only 10p per minute. BT-national-rate.		spam
5568	Will i_b going to esplanade fr home?		ham
5569	Pity, * was in mood for that. So...any other suggestions?		ham
5570	The guy did some bitching but I acted like i'd be interested in buying something else next week and he gave it to us for free		ham
5571	Rofl. Its true to its name		ham

5572 rows × 2 columns

## Step 2: Data Analysis

1. Check for Missing Values: It's essential to check if there are any missing values in your DataFrame.
2. Class Distribution: Check how many messages are "ham" and how many are "spam."

```
1 #number of missing values (NaN or None) in each column of th...
2 print(newdf['text'].isnull().sum(axis=0))
```

→ 0

```
1 print(newdf.isnull().sum())
```

```
2
```

```
→ text      0  
category   0  
dtype: int64
```

```
1 #Class Distribution: Check how many messages are "ham" and h  
2 #prints the frequency (count) of each unique value in the 'L  
3 print(df['category'].value_counts())
```

```
4
```

```
→ category  
ham      4825  
spam     747  
Name: count, dtype: int64
```

The ax object is used later to plot the bars using the ax.bar() method.

This creates a blank figure (canvas) and a set of axes (where the plot will appear).

figsize=(6, 6) sets the size of the plot (6 inches by 6 inches).

```
1 !pip install matplotlib
```

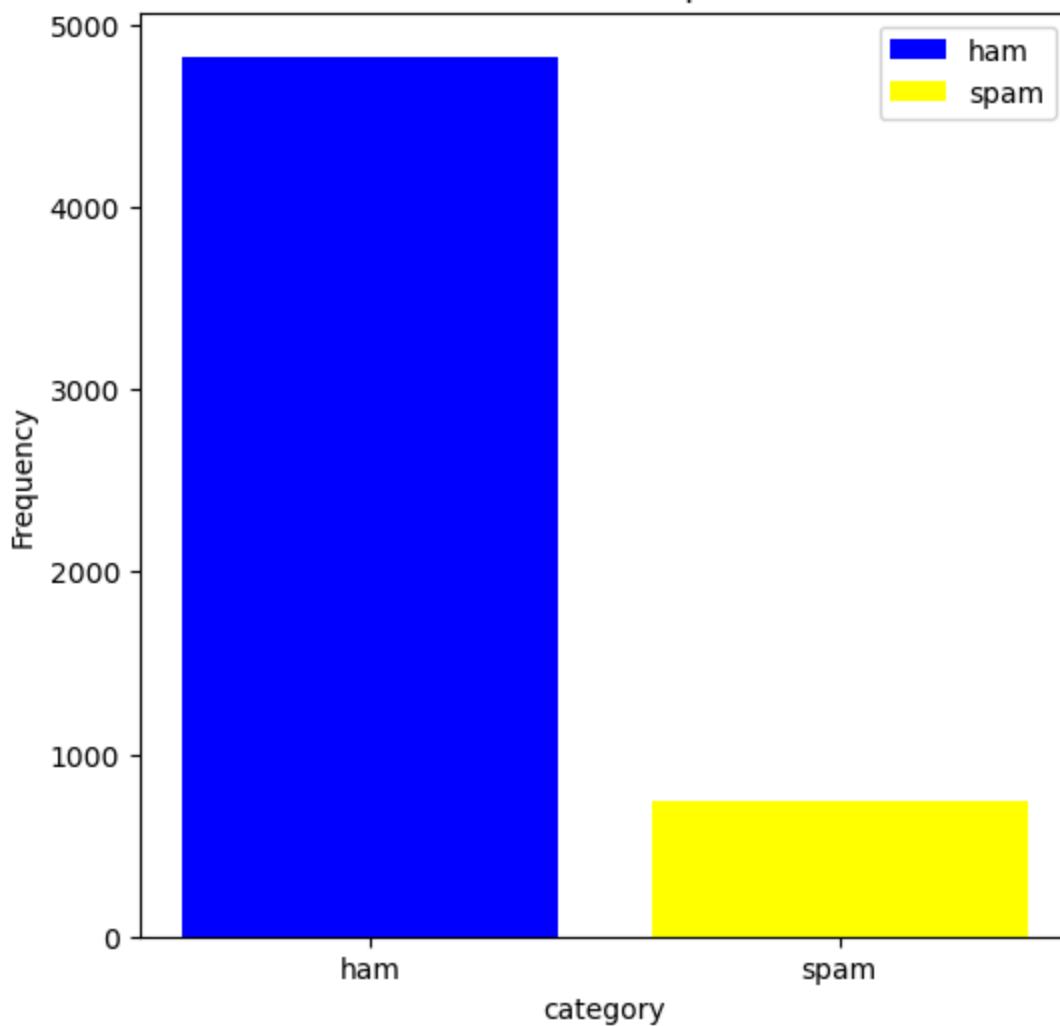
```
→ Requirement already satisfied: matplotlib in c:\users\dell\anaconda3\lib\site-packages (Requirement already satisfied: contourpy>=1.0.1 in c:\users\dell\anaconda3\lib\site-packages  
Requirement already satisfied: cycler>=0.10 in c:\users\dell\anaconda3\lib\site-packages  
Requirement already satisfied: fonttools>=4.22.0 in c:\users\dell\anaconda3\lib\site-packages  
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\dell\anaconda3\lib\site-packages  
Requirement already satisfied: numpy>=1.20 in c:\users\dell\anaconda3\lib\site-packages  
Requirement already satisfied: packaging>=20.0 in c:\users\dell\anaconda3\lib\site-packages  
Requirement already satisfied: pillow>=6.2.0 in c:\users\dell\anaconda3\lib\site-packages  
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in c:\users\dell\anaconda3\lib\site-packages  
Requirement already satisfied: python-dateutil>=2.7 in c:\users\dell\anaconda3\lib\site-packages  
Requirement already satisfied: six>=1.5 in c:\users\dell\anaconda3\lib\site-packages (fr
```

```
1 #The legend is the small box (usually on the side or top of  
2 #showing the color associated with each class.  
3  
4 import matplotlib.pyplot as plt #library, which is used to c  
5  
6 # Bar Plot for Class Distribution
```

```
7 #This counts how many times each category (like "ham" and "s
8 class_distribution = newdf['category'].value_counts()
9
10 # Create a figure and axis
11 fig, ax = plt.subplots(figsize=(6, 6))
12
13 # Plot each bar separately to associate them with labels
14 #Draws a blue bar for the first category (likely 'ham') usin
15 ax.bar(class_distribution.index[0], class_distribution.value
16 #Draws a yellow bar for the first category (likely 'spam') u
17 ax.bar(class_distribution.index[1], class_distribution.value
18
19 # Add title and labels
20 ax.set_title('Class Distribution of Spam vs Ham')
21 ax.set_xlabel('category')
22 ax.set_ylabel('Frequency')
23
24 # Add a legend with the correct labels
25 ax.legend(loc='upper right')
26
27 # Show the plot
28 plt.show()
```



Class Distribution of Spam vs Ham



```
1 newdf['text'].head(10)
```



```
0 Go until jurong point, crazy..  
Available only in bugis n great world la e buffet... Cine there got amore wat...  
1  
Ok lar... Joking wif u oni...  
2 Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA  
to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's  
3  
U dun say so early hor... U c already then say...  
4  
Nah I don't think he goes to usf, he lives around here though  
5 FreeMsg Hey there darling it's been 3 week's now and no word back! I'd  
like some fun you up for it still? Tb ok! XxX std chgs to send, £1.50 to rcv  
6  
Even my brother is not like to speak with me. They treat me like aids patient.  
7 As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been  
set as your callertune for all Callers. Press *9 to copy your friends Callertune  
8 WINNER!! As a valued network customer you have been selected to receive a £900
```

```
prize reward! To claim call 09061701461. Claim code KL341. Valid 12 hours only.  
9 Had your mobile 11 months or more? U R entitled to Update to the latest  
colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030  
Name: text, dtype: object
```

```
1 ' '.join(newdf['text'].head(10))
```

```
→ "Go until jurong point, crazy.. Available only in bugis n great world la e buffet...  
Cine there got amore wat... Ok lar... Joking wif u oni... Free entry in 2 a wkly comp  
to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std  
txt rate)T&C's apply 08452810075over18's U dun say so early hor... U c already then  
say... Nah I don't think he goes to usf, he lives around here though FreeMsg Hey there  
darling it's been 3 week's now and no word back! I'd like some fun you up for it still?  
Tb ok! XxX std chgs to send, £1.50 to rcv Even my brother is not like to speak with  
me. They treat me like aids patent. As per your request 'Melle Melle (Oru  
Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press  
*9 to copy your friends Callertune WINNER!! As a valued network customer you have been  
selected to receive a £900 prize reward! To claim call 09061701461. Claim code KL341.  
Valid 12 hours only. Had your mobile 11 months or more? U R entitled to Update to the  
latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on  
08002986030"
```

```
1 df.head(2)
```

	category	text	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN

```
1 ' '.join(newdf['text'].head(2))
```

```
→ 'Go until jurong point, crazy.. Available only in bugis n great world la e buffet...  
Cine there got amore wat... Ok lar... Joking wif u oni...'
```

```
1 import matplotlib.pyplot as plt  
2  
3 from wordcloud import WordCloud  
4  
5 text_corpus = ' '.join(newdf['text'])  
6  
7 wordcloud = WordCloud(width=800, height=400, random_state=42  
8  
9 # #This creates a new figure (or canvas) for the plot, with
```

```
10 #This function is used to create a new figure(or the overall
11 plt.figure(figsize=(10, 5))
12
13
14 ##This displays the word cloud image. The wordcloud object c
15 #interpolation='bilinear': This smooths the image when it is
16 plt.imshow(wordcloud, interpolation='bilinear')
17
18 ##this removes the axis from the plot, so no axis lines or la
19 plt.axis('off')
20
21 ##Adds a title to the plot, in this case, "Word Cloud for Te
22 plt.title('Word Cloud for English')
23
24 # Displays the plot with the word cloud.
25 plt.show()
```



- ## 1. Step 3: Text Preprocessing

2. Before you can use this data for modeling, you typically need to preprocess the text data.

Common steps include:

3. Lowercasing
4. Removing punctuation
5. Removing stop words
6. Tokenization
7. Converting text to numerical features (e.g., using Bag of Words or TF-IDF)

```
1 text="YES Sir"
2 print(text.lower())
3
```

→ yes sir

```
1 s=r'\n Yes sir'
2 print(s)
3 #t= '\n'
4 #print(t)
```

→ \n Yes sir

```
1 t= '\n yes sir, we are sleeping'
2 print(t)
```

→ yes sir, we are sleeping

```
1 import re
2 text='&&&&akhtar age is 14 and??? !!!;;;;,,saeed age is 15
3
4 text1=re.sub(r'^[a-zA-Z0-9\s]', '', text)
5
6 text1
```

→ 'akhtar age is 14 and saeed age is 15 are friends but they usually disagree on certain matters'

```
1 text1=text1.split(" ")
2 text1
```

→ ['akhtar',
'age',

```
'is',
'14',
'and',
'saeed',
'age',
'is',
'15',
'are',
'friends',
'but',
'they',
'usually',
'disagree',
'on',
'certain',
'matters']
```

1 Start coding or generate with AI.

```
1 from nltk.corpus import stopwords
2 stop = stopwords.words('english')
3 len(stop)
4
5
```

→ 198

1 text1

→ ['akhtar',
'age',
'is',
'14',
'and',
'saeed',
'age',
'is',
'15',
'are',
'friends',
'but',
'they',
'usually',
'disagree',
'on',
'certain',
'matters']

```
1 cleantext=[x for x in text1 if x not in stop]
2 cleantext
```

```
→ ['akhtar',
 'age',
 '14',
 'saeed',
 'age',
 '15',
 'friends',
 'usually',
 'disagree',
 'certain',
 'matters']
```

```
1 text= ['akhtar', 'age', 'is', '14', 'and', '15']
2 print(" ".join(text))
```

```
→ akhtar age is 14 and 15
```

```
1 newdf.head(10)
```

		text	category
0		Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	ham
1		Ok lar... Joking wif u oni...	ham
2		Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	spam
3		U dun say so early hor... U c already then say...	ham
4		Nah I don't think he goes to usf, he lives around here though	ham
5		FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, £1.50 to rcv	spam
6		Even my brother is not like to speak with me. They treat me like aids patent.	ham
7		As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 to copy your friends Callertune	ham
8		WINNER!! As a valued network customer you have been selected to receive a £900 prize reward! To claim call 09061701461. Claim code KL341. Valid 12 hours only.	spam
9		Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030	spam

✓ newdf.loc[:, 'Cleaned\_Text']

loc is used for label-based indexing.

: means all rows.

'Cleaned\_Text' is the name of the column you're assigning values to. select all rows in the column named 'Cleaned\_Text'.

```
1 import re
2 from sklearn.model_selection import train_test_split
3 from sklearn.feature_extraction.text import CountVectorizer
4
5 # Function to preprocess text
6 def preprocess_text(text):
7     # Lowercase the text
8     text = text.lower()
9     # Remove punctuation and non-alphanumeric characters
10    text = re.sub(r'^[a-zA-Z0-9\s]', '', text)
11    text = text.split(" ")
12
13    text = [x for x in text if x not in stop]
14    return " ".join(text)
15
16 # Apply preprocessing to the Message column
17 newdf.loc[:, 'Cleaned_Text'] = newdf['text'].apply(preprocess_text)
18
19 #newdf['Cleaned_Text'] = newdf['text'].apply(preprocess_text)
20
21 # Display the cleaned messages
22 print(newdf[['text', 'Cleaned_Text']].head(6))
23
```



```
0                               Go until jurong point, crazy.. Available
1
2 Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121
3
4
5      FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like sc
```

```
0                               go jurong point crazy available
1
2 free entry 2 wkly comp win fa cup final tkts 21st may 2005 text fa 87121 receive entr
3
```

```
4                                     freemsg hey darling 3 weeks word back
5
C:\Users\Dell\AppData\Local\Temp\ipykernel_11792\2084002667.py:17: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#inplace-operations
newdf.loc[:, 'Cleaned_Text'] = newdf['text'].apply(preprocess_text)
```

```
1 # Display the cleaned messages
2 print(newdf[['Cleaned_Text']].head(2))
```

```
→
0  go jurong point crazy available bugis n great world la e buffet cine got amore wat
1                               Cleaned_Text
ok lar joking wif u oni
```

Step 4: Feature Extraction You can use CountVectorizer to convert the cleaned text data into numerical format.

```
1 # Create features and labels
2 X = newdf['Cleaned_Text']
3 y = newdf['category']
4
5 # Split data into training and testing sets
6 X_train, X_test, y_train, y_test = train_test_split(X, y, te
7
```

```
1 # Vectorize the text data
2 vectorizer = CountVectorizer()
3 X_train_vectorized = vectorizer.fit_transform(X_train)
4
```

```
1 X_test_vectorized = vectorizer.transform(X_test)
2
3 print(X_train_vectorized.shape)
4 print(X_test_vectorized.shape)
5 print(X_train_vectorized[0].toarray())
```

```
→ (4457, 8218)
(1115, 8218)
[[0 0 0 ... 0 0 0]]
```

```

1 from sklearn.naive_bayes import MultinomialNB
2 from sklearn.metrics import classification_report
3
4 # Initialize and train the classifier
5 classifier = MultinomialNB()
6 classifier.fit(X_train_vectorized, y_train)
7
8 # Make predictions
9 y_pred = classifier.predict(X_test_vectorized)
10
11 # Evaluate the model
12 # Evaluate the model
13 report=classification_report(y_test, y_pred)
14
15 print(report)
16 print("#####")
17 #conf_matrix = confusion_matrix(y_test, y_pred)
18 #print(conf_matrix)

```

	precision	recall	f1-score	support
ham	0.98	1.00	0.99	965
spam	0.97	0.88	0.92	150
accuracy			0.98	1115
macro avg	0.98	0.94	0.96	1115
weighted avg	0.98	0.98	0.98	1115
#####				

```

1 conf_matrix = confusion_matrix(y_test, y_pred)
2 print(conf_matrix)

```

```

→ [[961  4]
 [ 18 132]]

```

```

1 # Calculate metrics
2 report = classification_report(y_test, y_pred, output_dict=True)
3 print(report['spam']['f1-score'])
4 print("\n*****")
5 print("\n*****")

```

```
6 print("\n*****")
7 accuracy = accuracy_score(y_test, y_pred)
```

→ 0.9230769230769231

```
*****
```

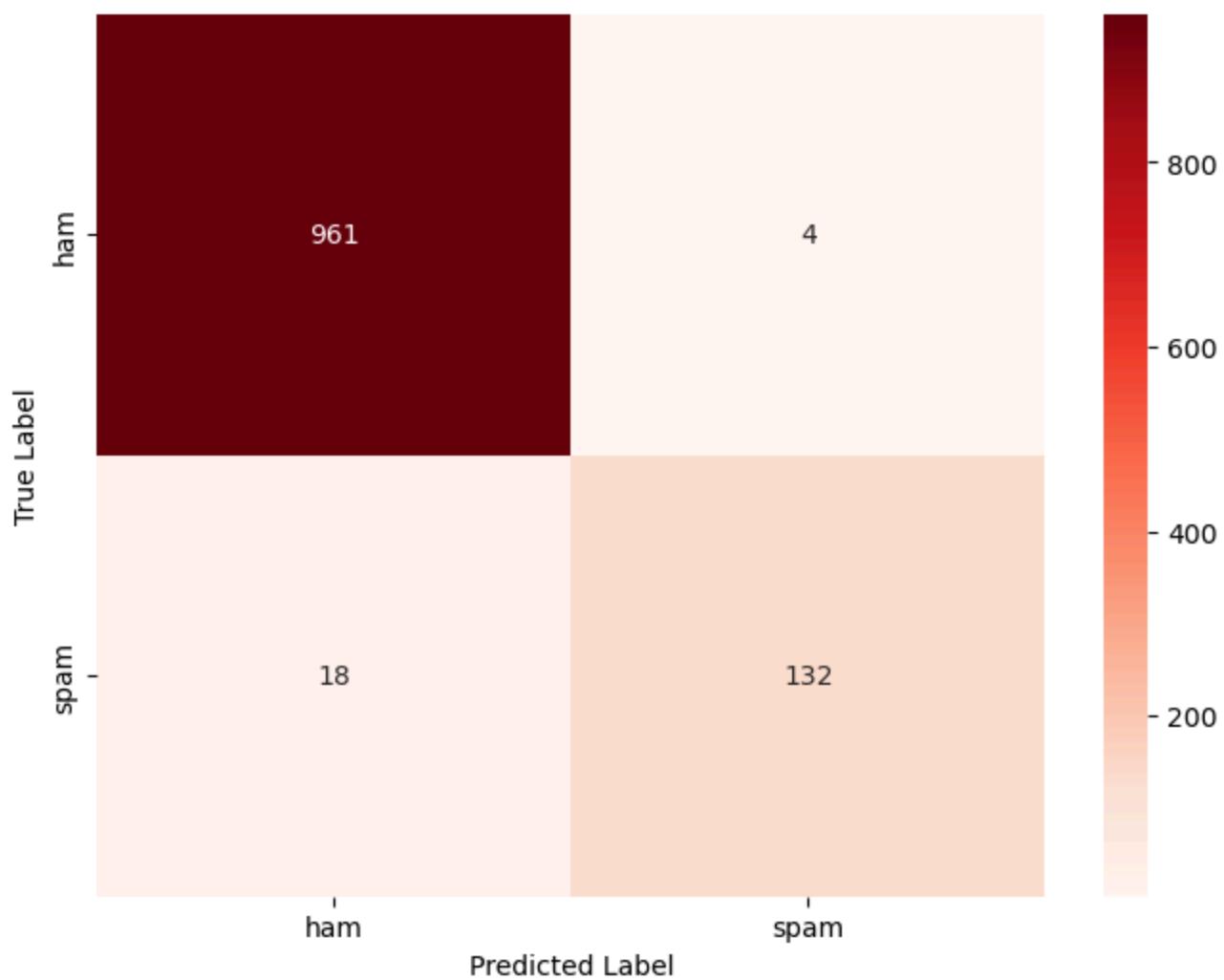
```
*****
```

```
*****
```

```
1 # Generate the confusion matrix
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 conf_matrix = confusion_matrix(y_test, y_pred)
5
6 # Create a heatmap for the confusion matrix
7 plt.figure(figsize=(8, 6))
8 sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Reds', x
9 plt.xlabel('Predicted Label')
10 plt.ylabel('True Label')
11 plt.title('Confusion Matrix')
12 plt.show()
```



Confusion Matrix

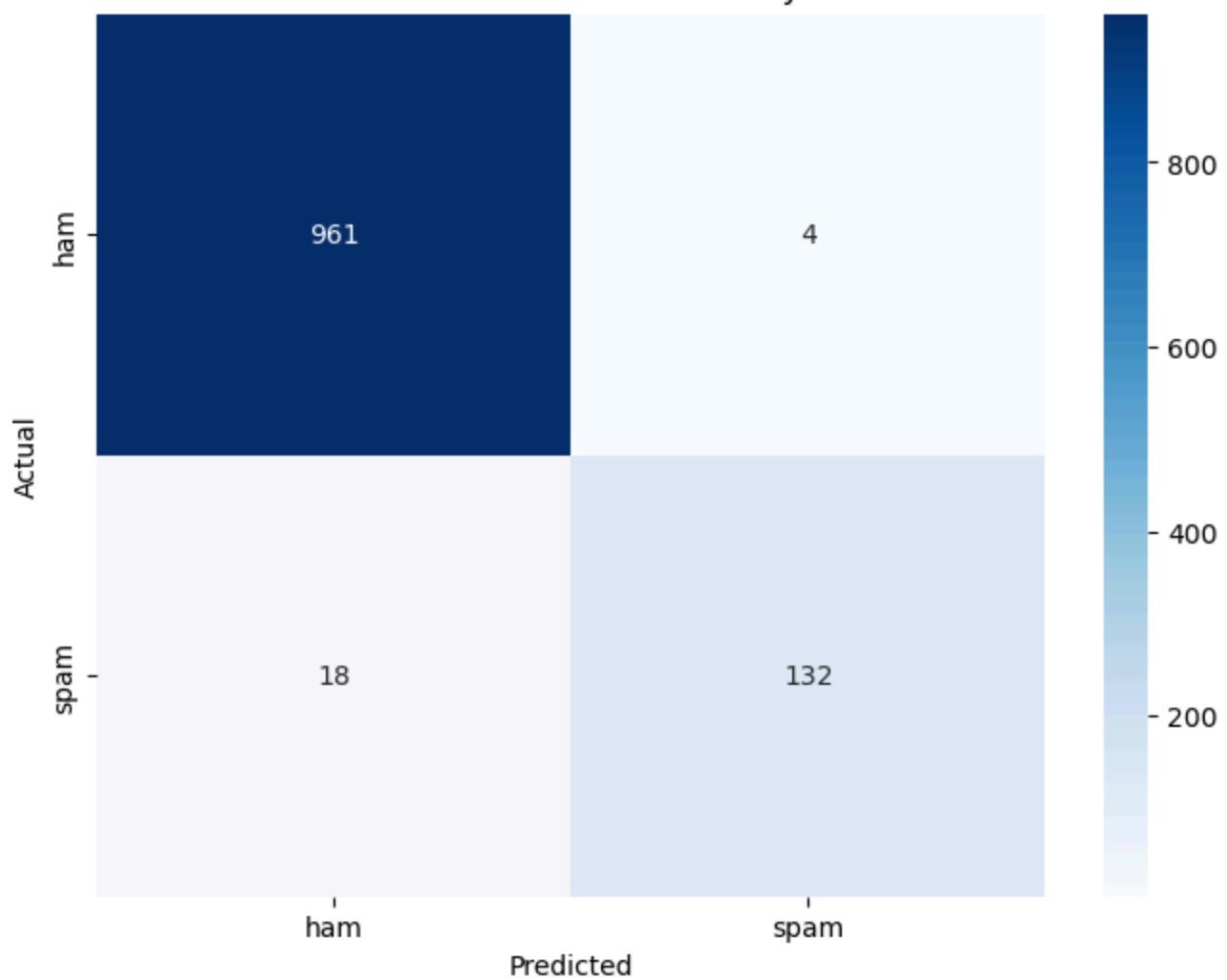


```
1 # machine learning algorithms--classification algorithms
2 # Classifiers to evaluate
3 classifiers = {
4     'Naive Bayes': MultinomialNB(),
5     'Logistic Regression': LogisticRegression(),
6     'Support Vector Machine': SVC(probability=True),
7     'Random Forest': RandomForestClassifier()
8 }
9
10 # Store metrics for plotting
11 all_metrics = {}
12
13 for name, classifier in classifiers.items():
14     # Initialize and train the classifier
15     classifier.fit(X_train_vectorized, y_train)
```

```
16
17     # Make predictions
18     y_pred = classifier.predict(X_test_vectorized)
19
20     # Calculate metrics
21     report = classification_report(y_test, y_pred, output_dict=True)
22     accuracy = accuracy_score(y_test, y_pred)
23
24     # Prepare data for the bar chart
25     metrics = {
26         'Precision': report['spam']['precision'],
27         'Recall': report['spam']['recall'],
28         'F1 Score': report['spam']['f1-score'],
29         'Accuracy': accuracy
30     }
31
32     # Store metrics for each classifier
33     all_metrics[name] = metrics
34
35     # Create confusion matrix
36     cm = confusion_matrix(y_test, y_pred, labels=['ham', 'spam'])
37
38     # Create heatmap for confusion matrix
39     plt.figure(figsize=(8, 6))
40     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['ham', 'spam'], yticklabels=['ham', 'spam'])
41     plt.ylabel('Actual')
42     plt.xlabel('Predicted')
43     plt.title(f'Confusion Matrix for {name}')
44     plt.show()
45
46     # Print the classification report for each classifier
47     print(f"\n{name} Classification Report:")
48     print(classification_report(y_test, y_pred))
49
50 print("dictionary ",all_metrics)
51
```



Confusion Matrix for Naive Bayes

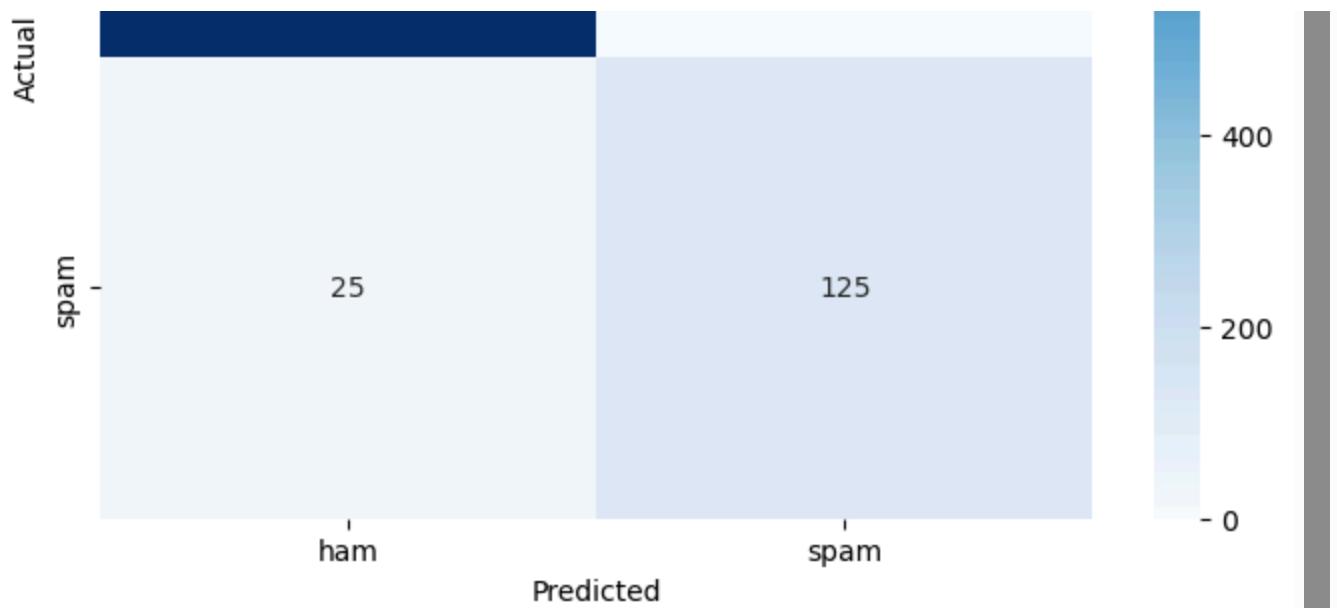


Naive Bayes Classification Report:

	precision	recall	f1-score	support
ham	0.98	1.00	0.99	965
spam	0.97	0.88	0.92	150
accuracy			0.98	1115
macro avg	0.98	0.94	0.96	1115
weighted avg	0.98	0.98	0.98	1115

Confusion Matrix for Logistic Regression

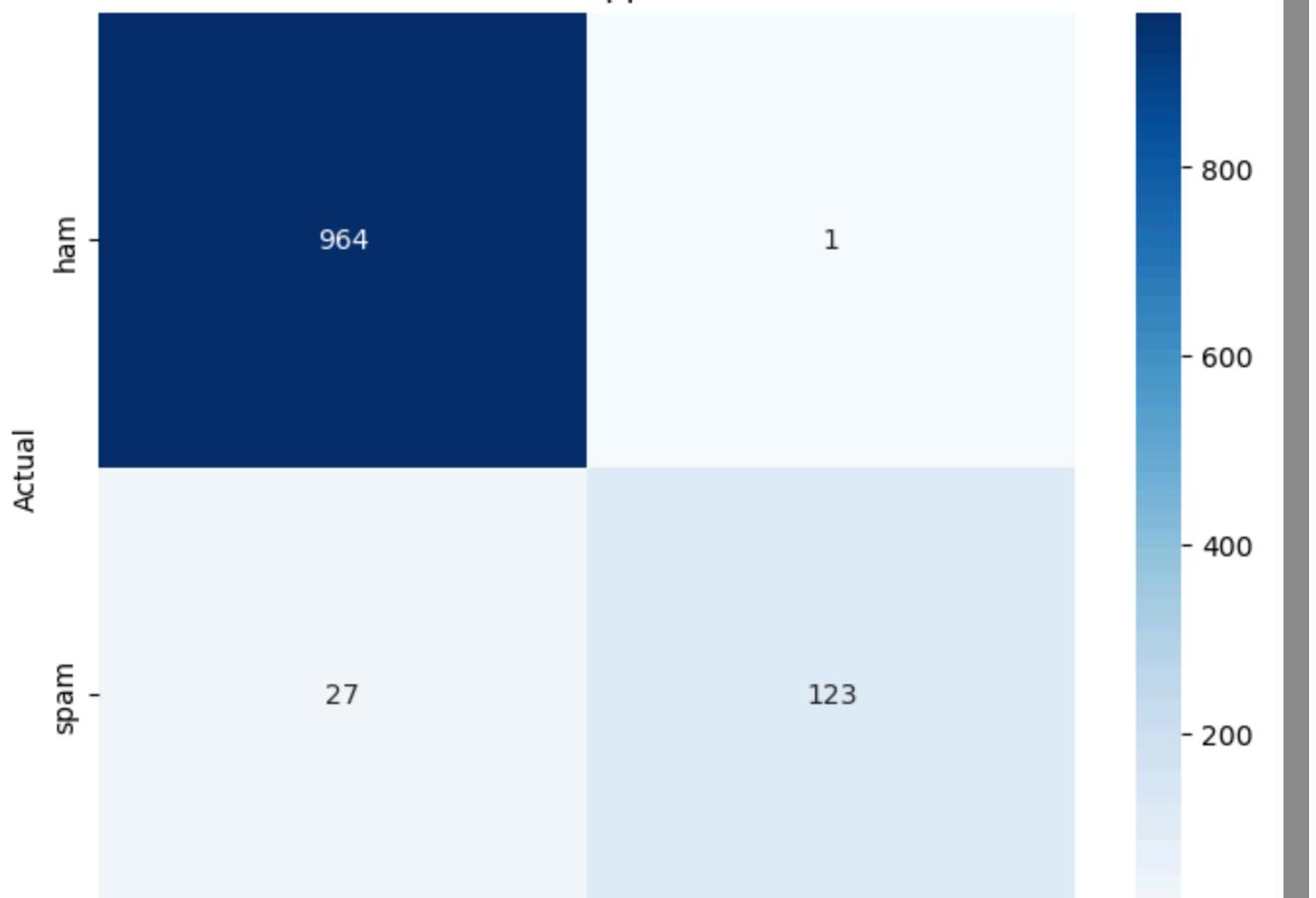




Logistic Regression Classification Report:

	precision	recall	f1-score	support
ham	0.97	1.00	0.99	965
spam	1.00	0.83	0.91	150
accuracy			0.98	1115
macro avg	0.99	0.92	0.95	1115
weighted avg	0.98	0.98	0.98	1115

Confusion Matrix for Support Vector Machine

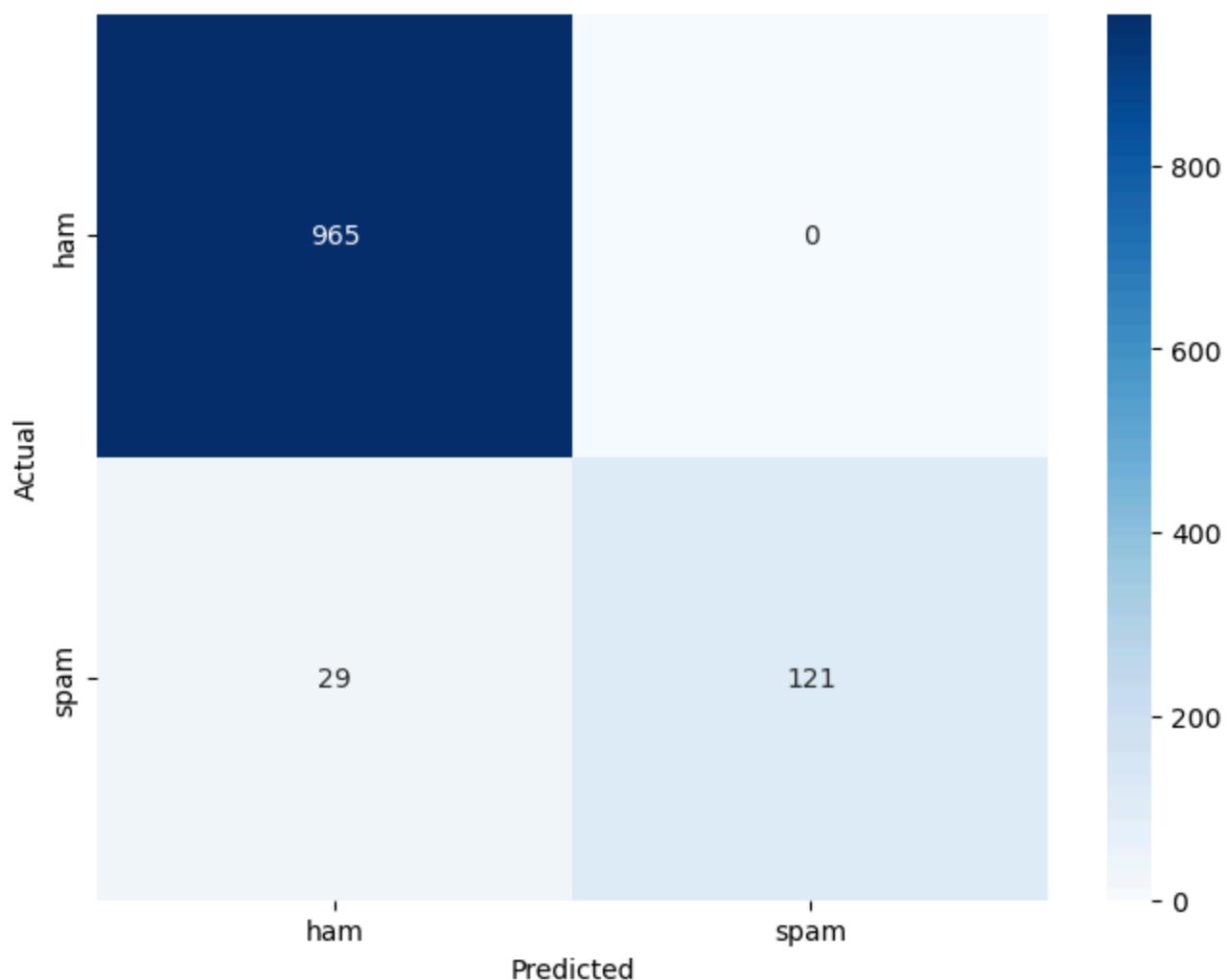


	ham	spam
	Predicted	

#### Support Vector Machine Classification Report:

	precision	recall	f1-score	support
ham	0.97	1.00	0.99	965
spam	0.99	0.82	0.90	150
accuracy			0.97	1115
macro avg	0.98	0.91	0.94	1115
weighted avg	0.98	0.97	0.97	1115

Confusion Matrix for Random Forest



#### Random Forest Classification Report:

	precision	recall	f1-score	support
ham	0.97	1.00	0.99	965
spam	1.00	0.81	0.89	150
accuracy			0.97	1115
macro avg	0.99	0.90	0.94	1115
weighted avg	0.97	0.97	0.97	1115

```
dictionary {'Naive Bayes': {'Precision': 0.9705882352941176, 'Recall': 0.88, 'F1 Score': 0.927}}
```



```
1 # Prepare data for grouped bar chart
2 metrics_df = pd.DataFrame(all_metrics)
3 #metrics_df = metrics_df[['Accuracy', 'Precision', 'Recall',
4 metrics_df
```

	Naive Bayes	Logistic Regression	Support Vector Machine	Random Forest
Precision	0.970588	1.000000	0.991935	1.000000
Recall	0.880000	0.833333	0.820000	0.806667
F1 Score	0.923077	0.909091	0.897810	0.892989
Accuracy	0.980269	0.977578	0.974888	0.973991

```
1 metrics_df=metrics_df.T
2 metrics_df
```

	Precision	Recall	F1 Score	Accuracy
Naive Bayes	0.970588	0.880000	0.923077	0.980269
Logistic Regression	1.000000	0.833333	0.909091	0.977578
Support Vector Machine	0.991935	0.820000	0.897810	0.974888
Random Forest	1.000000	0.806667	0.892989	0.973991

1 Start coding or [generate](#) with AI.

## ▼ Notebook: 27) Spam\_Classification-bigrams.ipynb

```
1 import numpy as np
2 import pandas as pd
3 import itertools
4 from sklearn.model_selection import train_test_split
5 from sklearn.feature_extraction.text import TfidfVectorizer
6 from sklearn.linear_model import PassiveAggressiveClassifier
7 from sklearn.linear_model import LogisticRegression
8 from sklearn.svm import SVC
9 from sklearn.ensemble import RandomForestClassifier
10 from sklearn.metrics import accuracy_score, confusion_matrix
```

```
1 #The issue you're encountering is due to the fact that the s  
2 #Instead of pip install sklearn, you need to install scikit-  
3 #!pip install sklearn
```

```
1 !pip install scikit-learn  
2
```

→ Requirement already satisfied: scikit-learn in c:\users\dell\anaconda3\lib\site-packages  
Requirement already satisfied: numpy>=1.17.3 in c:\users\dell\anaconda3\lib\site-packages  
Requirement already satisfied: scipy>=1.5.0 in c:\users\dell\anaconda3\lib\site-packages  
Requirement already satisfied: joblib>=1.1.1 in c:\users\dell\anaconda3\lib\site-packages  
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\dell\anaconda3\lib\site-

1 The error you're encountering, UnicodeDecodeError, typically

→ Cell In[22], line 1

The error you're encountering, UnicodeDecodeError, typically occurs when the CSV file contains characters that aren't properly decoded using the default encoding (utf-8).

SyntaxError: invalid syntax

```
1 df=pd.read_csv('spam.csv', encoding='latin1') # #Latin-1 is ...  
2 df.shape #The re...  
3
```

→ (5572, 5)

```
1 #Get shape and head  
2 print(df.head(10))
```

	v1	v2	Unnamed: 2	\\
0	ham Go until jurong point, crazy.. Available only ...		NaN	
1	ham Ok lar... Joking wif u oni...		NaN	
2	spam Free entry in 2 a wkly comp to win FA Cup fina...		NaN	
3	ham U dun say so early hor... U c already then say...		NaN	
4	ham Nah I don't think he goes to usf, he lives aro...		NaN	
5	spam FreeMsg Hey there darling it's been 3 week's n...		NaN	
6	ham Even my brother is not like to speak with me. ...		NaN	
7	ham As per your request 'Melle Melle (Oru Minnamin...		NaN	
8	spam WINNER!! As a valued network customer you have...		NaN	
9	spam Had your mobile 11 months or more? U R entitle...		NaN	
	Unnamed: 3	Unnamed: 4		
0	NaN	NaN		

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

```
1
2 pd.set_option('display.max_columns', None)
3 pd.set_option('display.expand_frame_repr', True) #True (defa
4                                         #False: |
5 pd.set_option('max_colwidth',None)  #Setting max_colwidth to
6 df.head(10)
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives around here though	NaN	NaN	NaN
5	spam	FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, £1.50 to rcv	NaN	NaN	NaN
6	ham	Even my brother is not like to speak with me. They treat me like aids patent.	NaN	NaN	NaN
7	ham	As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 to copy your friends Callertune	NaN	NaN	NaN
		WINNER!! As a valued network customer you have been			

```
1 pwd
```

```
→ 'E:\\Python For Advance Application\\notebooks'
```

```
1 df = df.rename(columns={'v1': 'category', 'v2': 'text'})  
2 df  
3
```

	category	text	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives around here though	NaN	NaN	NaN
...	...	...	...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u. U have won the £750 Pound prize. 2 claim is easy, call 087187272008 NOW! Only 10p per minute. BT-national-rate.	NaN	NaN	NaN
5568	ham	Will l_b going to esplanade fr home?	NaN	NaN	NaN
5569	ham	Pity, * was in mood for that. So...any other suggestions?	NaN	NaN	NaN
5570	ham	The guy did some bitching but I acted like i'd be interested in buying something else next week	NaN	NaN	NaN

```
1 newdf=df[['text', 'category']]  
2 newdf
```



text category

0	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...	ham
1	Ok lar... Joking wif u oni...	ham
2	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's	spam
3	U dun say so early hor... U c already then say...	ham
4	Nah I don't think he goes to usf, he lives around here though	ham
...	...	...
5567	This is the 2nd time we have tried 2 contact u. U have won the £750 Pound prize. 2 claim is easy, call 087187272008 NOW! Only 10p per minute. BT-national-rate.	spam
5568	Will i_ b going to esplanade fr home?	ham
5569	Pity, * was in mood for that. So...any other suggestions?	ham
5570	The guy did some bitching but I acted like i'd be interested in buying something else next week and he gave it to us for free	ham
5571	Rofl. Its true to its name	ham

5572 rows × 2 columns

## Step 2: Data Analysis

1. Check for Missing Values: It's essential to check if there are any missing values in your DataFrame.
2. Class Distribution: Check how many messages are "ham" and how many are "spam."

```
1 #number of missing values (NaN or None) in each column of th
2 print(newdf['text'].isnull().sum(axis=0))
```

→ 0

```
1 print(newdf.isnull().sum())
2
```

```
→ text      0
    category  0
    dtype: int64
```

```
1 #Class Distribution: Check how many messages are "ham" and h
2 #prints the frequency (count) of each unique value in the 'L
```

```
3 print(df['category'].value_counts())
4
```

```
→ category
ham    4825
spam    747
Name: count, dtype: int64
```

The ax object is used later to plot the bars using the ax.bar() method.

This creates a blank figure (canvas) and a set of axes (where the plot will appear).

figsize=(6, 6) sets the size of the plot (6 inches by 6 inches).

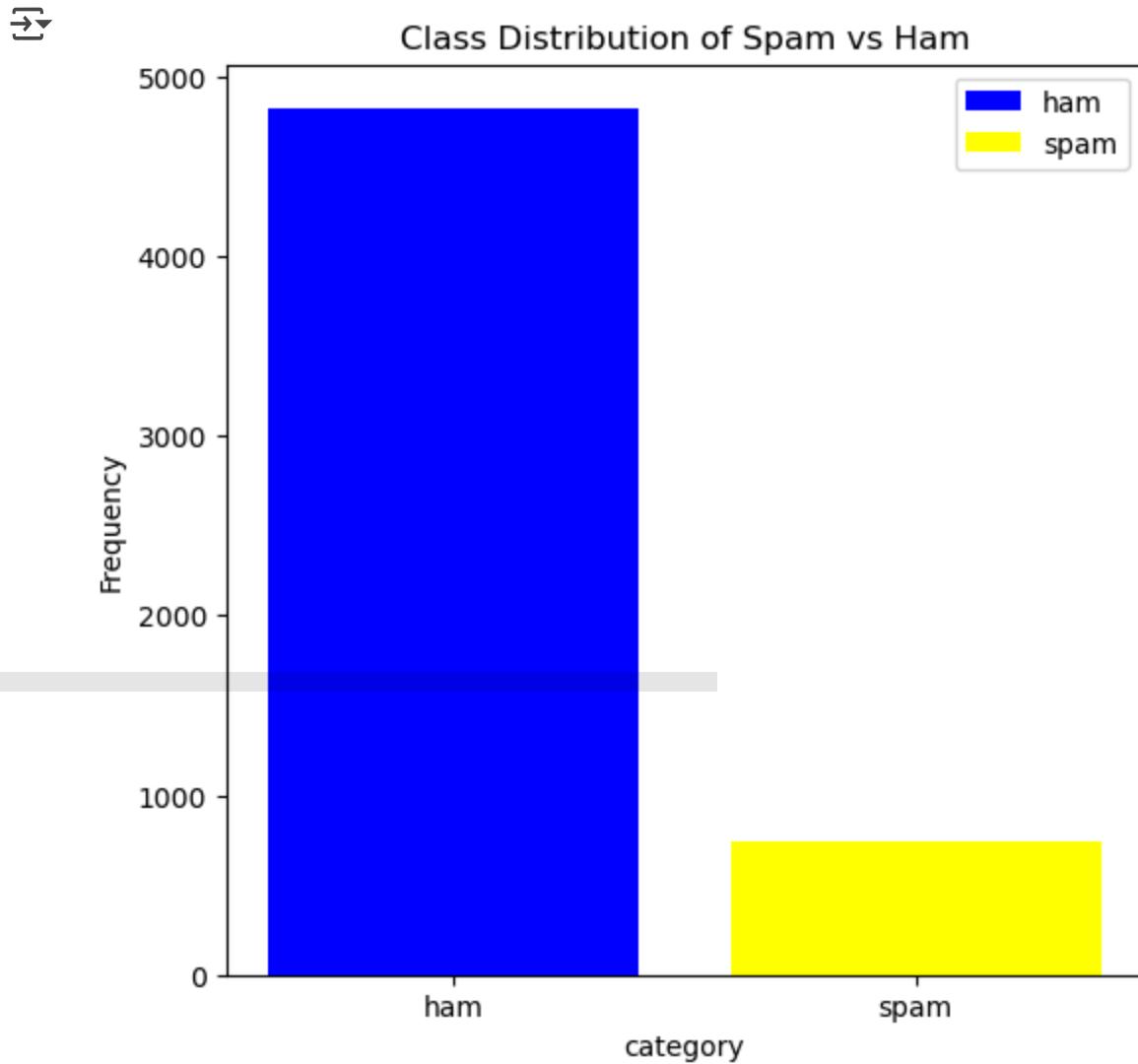
```
1 !pip install matplotlib
```

```
→ Requirement already satisfied: matplotlib in c:\users\dell\anaconda3\lib\site-packages
Requirement already satisfied: contourpy>=1.0.1 in c:\users\dell\anaconda3\lib\site-packages
Requirement already satisfied: cycler>=0.10 in c:\users\dell\anaconda3\lib\site-packages
Requirement already satisfied: fonttools>=4.22.0 in c:\users\dell\anaconda3\lib\site-packages
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\dell\anaconda3\lib\site-packages
Requirement already satisfied: numpy>=1.20 in c:\users\dell\anaconda3\lib\site-packages
Requirement already satisfied: packaging>=20.0 in c:\users\dell\anaconda3\lib\site-packages
Requirement already satisfied: pillow>=6.2.0 in c:\users\dell\anaconda3\lib\site-packages
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in c:\users\dell\anaconda3\lib\site-packages
Requirement already satisfied: python-dateutil>=2.7 in c:\users\dell\anaconda3\lib\site-packages
Requirement already satisfied: six>=1.5 in c:\users\dell\anaconda3\lib\site-packages (from matplotlib>=3.3.2)
Requirement already satisfied: typing-extensions>=3.7.4.3 in c:\users\dell\anaconda3\lib\site-packages
```



```
1 #The legend is the small box (usually on the side or top of
2 #showing the color associated with each class.
3
4 import matplotlib.pyplot as plt #library, which is used to c
5
6 # Bar Plot for Class Distribution
7 #This counts how many times each category (like "ham" and "s
8 class_distribution = newdf['category'].value_counts()
9
10 # Create a figure and axis
11 fig, ax = plt.subplots(figsize=(6, 6))
12
13 # Plot each bar separately to associate them with labels
14 #Draws a blue bar for the first category (likely 'ham') usin
```

```
15 ax.bar(class_distribution.index[0], class_distribution.value
16 #Draws a yellow bar for the first category (likely 'spam') u
17 ax.bar(class_distribution.index[1], class_distribution.value
18
19 # Add title and labels
20 ax.set_title('Class Distribution of Spam vs Ham')
21 ax.set_xlabel('category')
22 ax.set_ylabel('Frequency')
23
24 # Add a legend with the correct labels
25 ax.legend(loc='upper right')
26
27 # Show the plot
28 plt.show()
```



```
1 import matplotlib.pyplot as plt
2
3 from wordcloud import WordCloud
4
5 text_corpus = ' '.join(newdf['text'])
6
7 wordcloud = WordCloud(width=800, height=400, random_state=42
8
9 # #This creates a new figure (or canvas) for the plot, with
10 #This function is used to create a new figure(or the overall
11 plt.figure(figsize=(10, 5))
12
13
14 ##This displays the word cloud image. The wordcloud object c
15 #interpolation='bilinear': This smooths the image when it is
16 plt.imshow(wordcloud, interpolation='bilinear')
17
18 ##his removes the axis from the plot, so no axis lines or la
19 plt.axis('off')
20
21 ##Adds a title to the plot, in this case, "Word Cloud for Te
22 plt.title('Word Cloud for English')
23
24 # Displays the plot with the word cloud.
25 plt.show()
```



## 1. Step 3: Text Preprocessing

2. Before you can use this data for modeling, you typically need to preprocess the text data.

Common steps include:

### 3. Lowercasing

## 4. Removing punctuation

## 5. Removing stop words

## 6. Tokenization

## 7. Converting text to numerical features (e.g., using Bag of Words or TF-IDF)

1 Start coding or generate with AI.

```
1 from nltk.corpus import stopwords  
2 stop = stopwords.words('english')  
3 len(stop)  
4  
5
```

## newdf.loc[:, 'Cleaned\_Text']

loc is used for label-based indexing.

: means all rows.

'Cleaned\_Text' is the name of the column you're assigning values to. select all rows in the column named 'Cleaned\_Text'.

```
1 import re
2 from sklearn.model_selection import train_test_split
3 from sklearn.feature_extraction.text import CountVectorizer
4
5 # Function to preprocess text
6 def preprocess_text(text):
7     # Lowercase the text
8     text = text.lower()
9     # Remove punctuation and non-alphanumeric characters
10    text = re.sub(r'^[a-zA-Z0-9\s]', '', text)
11    text = text.split(" ")
12
13    text = [x for x in text if x not in stop]
14    return " ".join(text)
15
16 # Apply preprocessing to the Message column
17 newdf.loc[:, 'Cleaned_Text'] = newdf['text'].apply(preprocess_text)
18
19 #newdf['Cleaned_Text'] = newdf['text'].apply(preprocess_text)
20
21 # Display the cleaned messages
22 print(newdf[['text', 'Cleaned_Text']].head(6))
23
```



```
0                               Go until jurong point, crazy.. Available
1
2 Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121
3
4
5      FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like sc
```

```
0 go jurong point crazy available
1
2 free entry 2 wkly comp win fa cup final tkts 21st may 2005 text fa 87121 receive entr
3
4
5 freemsg hey darling 3 weeks word back
C:\Users\Dell\AppData\Local\Temp\ipykernel_1112\2084002667.py:17: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#inplace-operations
newdf.loc[:, 'Cleaned_Text'] = newdf['text'].apply(preprocess_text)
```

```
1 # Display the cleaned messages
2 print(newdf[['Cleaned_Text']].head(2))
```

```
→ Cleaned_Text
0 go jurong point crazy available bugis n great world la e buffet cine got amore wat
1 ok lar joking wif u oni
◀ ━━━━ ▶
```

Step 4: Feature Extraction You can use CountVectorizer to convert the cleaned text data into numerical format.

```
1 # Create features and labels
2 X = newdf['Cleaned_Text']
3 y = newdf['category']
4
5 # Split data into training and testing sets
6 X_train, X_test, y_train, y_test = train_test_split(X, y, te
7
```

```
1 # Vectorize the text data
2 vectorizer = CountVectorizer(ngram_range=(2,2))
3 X_train_vectorized = vectorizer.fit_transform(X_train)
4
```

```
1 X_test_vectorized = vectorizer.transform(X_test)
2
3 print(X_train_vectorized.shape)
4 print(X_test_vectorized.shape)
5 print(X_train_vectorized[0].toarray())
```

```
→ (4457, 26355)
  (1115, 26355)
  [[0 0 0 ... 0 0 0]]
```

```
1 from sklearn.naive_bayes import MultinomialNB
2 from sklearn.metrics import classification_report
3
4 # Initialize and train the classifier
5 classifier = MultinomialNB()
6 classifier.fit(X_train_vectorized, y_train)
7
8 # Make predictions
9 y_pred = classifier.predict(X_test_vectorized)
10
11 # Evaluate the model
12 # Evaluate the model
13 report=classification_report(y_test, y_pred)
14
15 print(report)
16 print("#####")
17 #conf_matrix = confusion_matrix(y_test, y_pred)
18 #print(conf_matrix)
```

```
→          precision    recall  f1-score   support
            ham       0.98     1.00      0.99      965
            spam       0.98     0.85      0.91      150
            accuracy                           0.98      1115
            macro avg       0.98     0.92      0.95      1115
            weighted avg       0.98     0.98      0.98      1115
            #####

```

```
1 conf_matrix = confusion_matrix(y_test, y_pred)
2 print(conf_matrix)
```

```
→ [[963  2]
  [ 23 127]]
```

```
1 # Calculate metrics
2 report = classification_report(y_test, y_pred, output_dict=T
```

```
3 print(report['spam']['f1-score'])
4 print("\n*****")
5 print("\n*****")
6 print("\n*****")
7 accuracy = accuracy_score(y_test, y_pred)
```

→ 0.910394265232975

```
*****
```

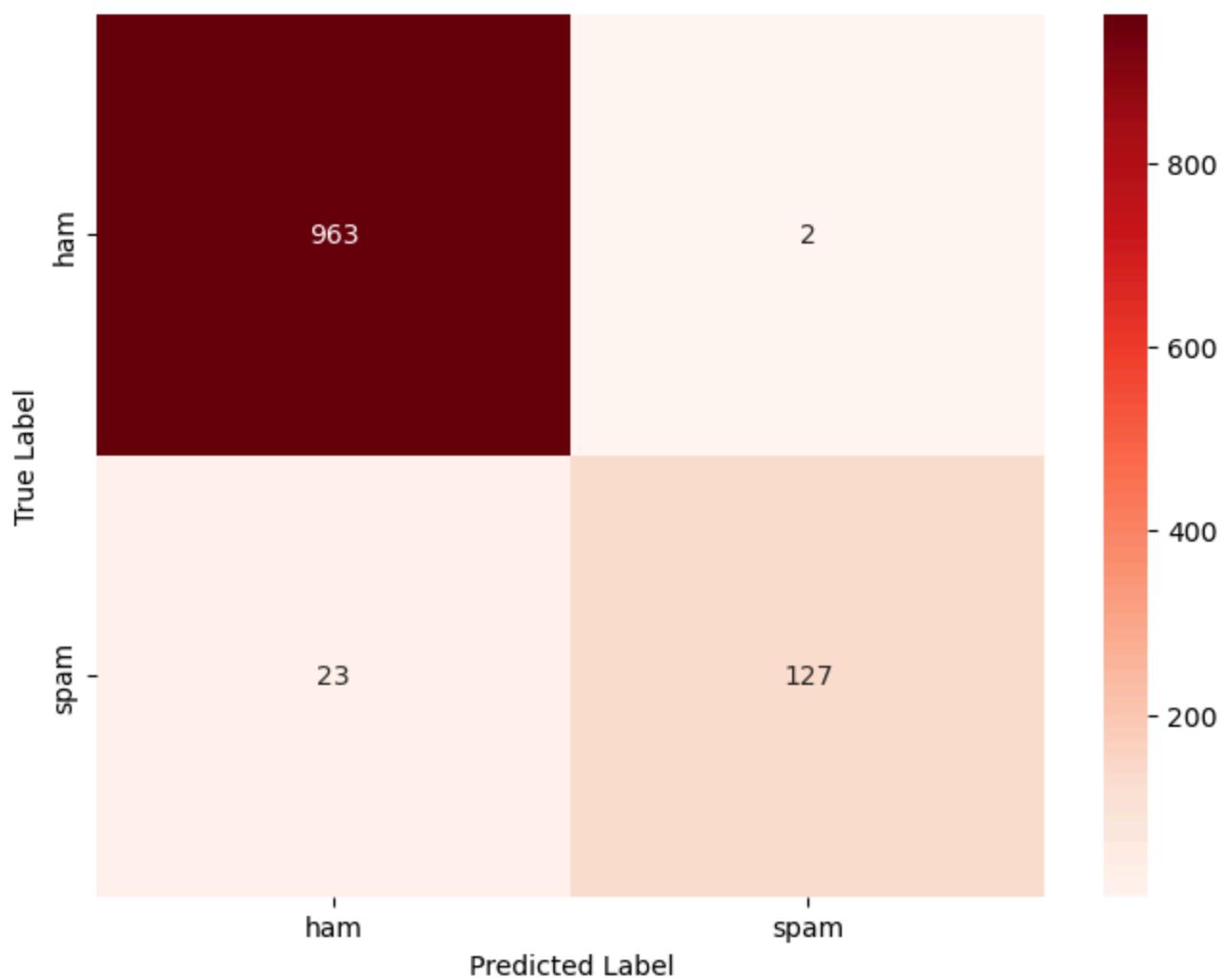
```
*****
```

```
*****
```

```
1 # Generate the confusion matrix
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 conf_matrix = confusion_matrix(y_test, y_pred)
5
6 # Create a heatmap for the confusion matrix
7 plt.figure(figsize=(8, 6))
8 sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Reds', x
9 plt.xlabel('Predicted Label')
10 plt.ylabel('True Label')
11 plt.title('Confusion Matrix')
12 plt.show()
```



Confusion Matrix

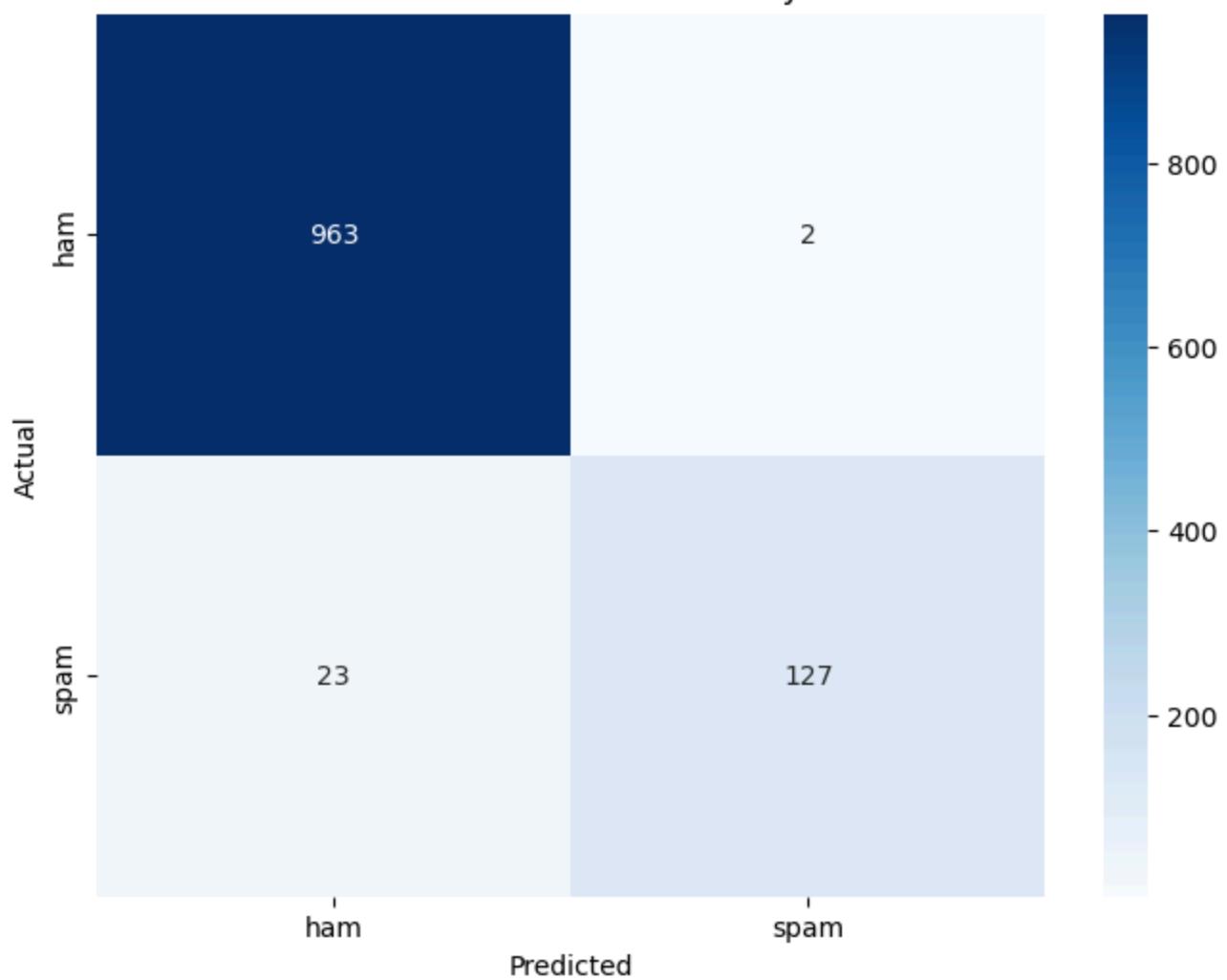


```
1 # machine learning algorithms--classification algorithms
2 # Classifiers to evaluate
3 classifiers = {
4     'Naive Bayes': MultinomialNB(),
5     'Logistic Regression': LogisticRegression(),
6     'Support Vector Machine': SVC(probability=True),
7     'Random Forest': RandomForestClassifier()
8 }
9
10 # Store metrics for plotting
11 all_metrics = {}
12
13 for name, classifier in classifiers.items():
14     # Initialize and train the classifier
15     classifier.fit(X_train_vectorized, y_train)
```

```
16
17     # Make predictions
18     y_pred = classifier.predict(X_test_vectorized)
19
20     # Calculate metrics
21     report = classification_report(y_test, y_pred, output_di
22     accuracy = accuracy_score(y_test, y_pred)
23
24     # Prepare data for the bar chart
25     metrics = {
26         'Precision': report['spam']['precision'],
27         'Recall': report['spam']['recall'],
28         'F1 Score': report['spam']['f1-score'],
29         'Accuracy': accuracy
30     }
31
32     # Store metrics for each classifier
33     all_metrics[name] = metrics
34
35     # Create confusion matrix
36     cm = confusion_matrix(y_test, y_pred, labels=['ham', 'sp
37
38     # Create heatmap for confusion matrix
39     plt.figure(figsize=(8, 6))
40     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xtick
41     plt.ylabel('Actual')
42     plt.xlabel('Predicted')
43     plt.title(f'Confusion Matrix for {name}')
44     plt.show()
45
46     # Print the classification report for each classifier
47     print(f"\n{name} Classification Report:")
48     print(classification_report(y_test, y_pred))
49
50 print("dictionary ",all_metrics)
51
```



Confusion Matrix for Naive Bayes

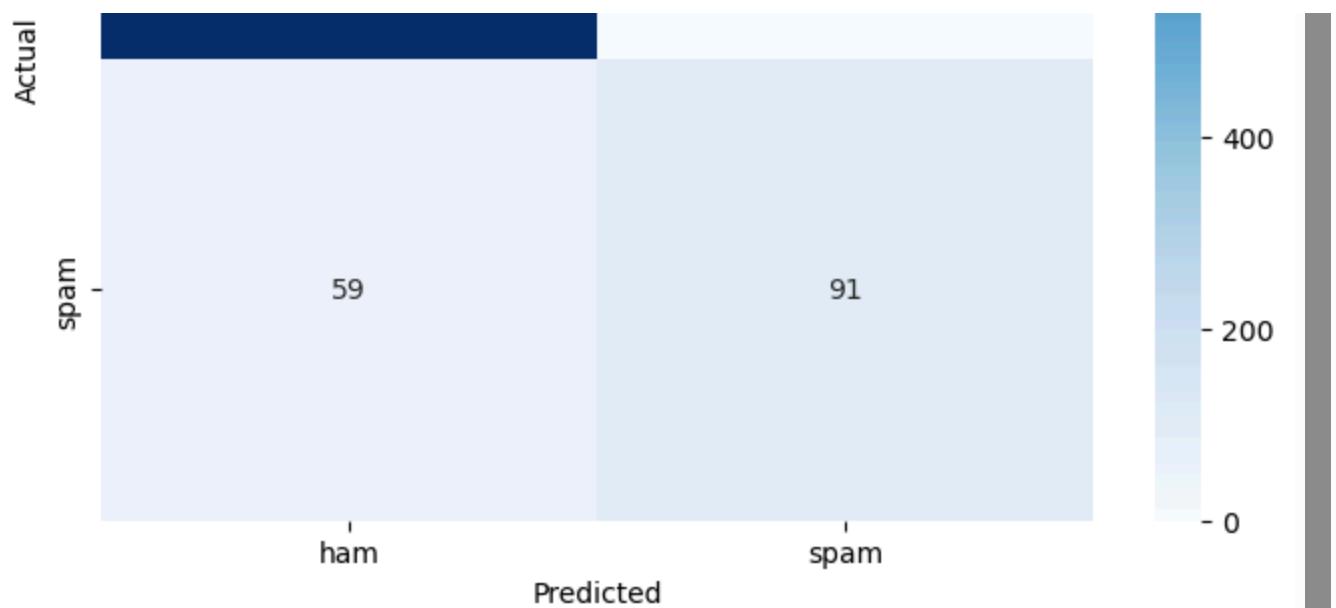


Naive Bayes Classification Report:

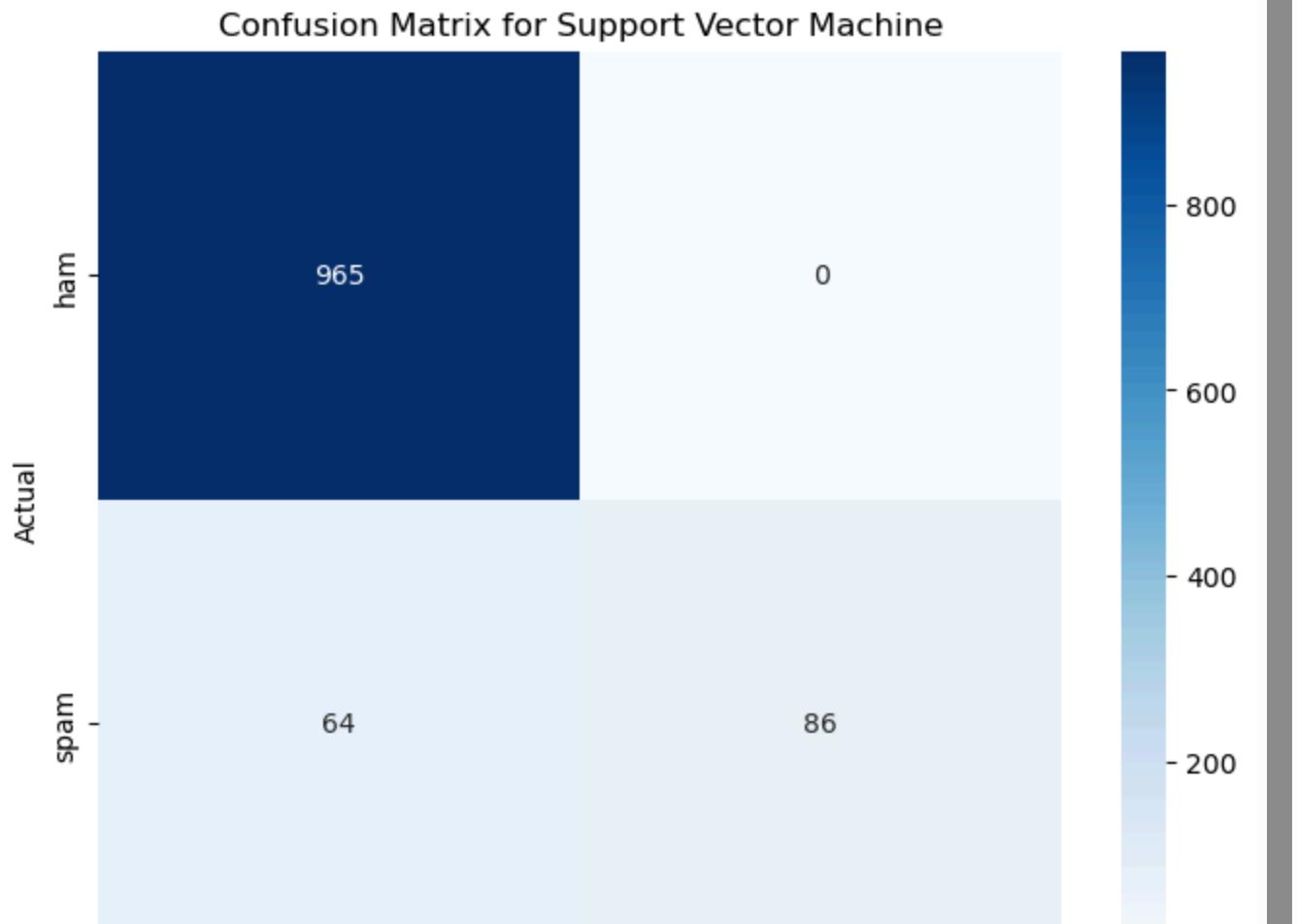
	precision	recall	f1-score	support
ham	0.98	1.00	0.99	965
spam	0.98	0.85	0.91	150
accuracy			0.98	1115
macro avg	0.98	0.92	0.95	1115
weighted avg	0.98	0.98	0.98	1115

Confusion Matrix for Logistic Regression



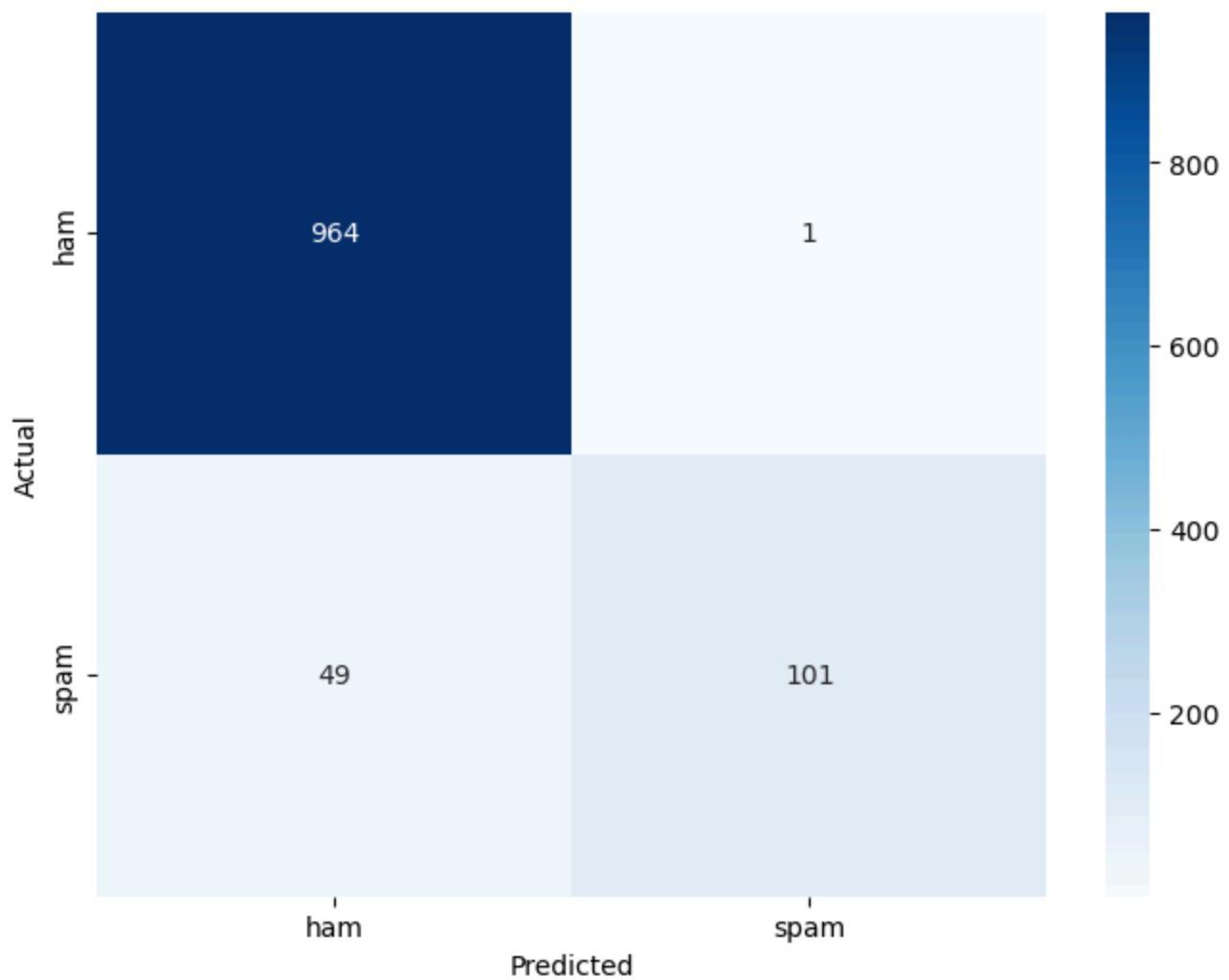


Logistic Regression Classification Report:				
	precision	recall	f1-score	support
ham	0.94	1.00	0.97	965
spam	1.00	0.61	0.76	150
accuracy			0.95	1115
macro avg	0.97	0.80	0.86	1115
weighted avg	0.95	0.95	0.94	1115



	ham	spam	- 0
	Predicted		
	precision	recall	f1-score
ham	0.94	1.00	0.97
spam	1.00	0.57	0.73
accuracy			0.94
macro avg	0.97	0.79	0.85
weighted avg	0.95	0.94	0.94

Confusion Matrix for Random Forest



	ham	spam	- 0
	Predicted		
	precision	recall	f1-score
ham	0.95	1.00	0.97
spam	0.99	0.67	0.80
accuracy			0.96
macro avg	0.97	0.84	0.89
weighted avg	0.96	0.96	0.95

```
dictionary {'Naive Bayes': {'Precision': 0.9844961240310077, 'Recall': 0.8466666666666667}
```



```
1 # Prepare data for grouped bar chart
2 metrics_df = pd.DataFrame(all_metrics)
3 #metrics_df = metrics_df[['Accuracy', 'Precision', 'Recall',
4 metrics_df
```

	Naive Bayes	Logistic Regression	Support Vector Machine	Random Forest
Precision	0.984496	1.000000	1.000000	0.990196
Recall	0.846667	0.606667	0.573333	0.673333
F1 Score	0.910394	0.755187	0.728814	0.801587
Accuracy	0.977578	0.947085	0.942601	0.955157

```
1 metrics_df=metrics_df.T
2 metrics_df
```

	Precision	Recall	F1 Score	Accuracy
Naive Bayes	0.984496	0.846667	0.910394	0.977578
Logistic Regression	1.000000	0.606667	0.755187	0.947085
Support Vector Machine	1.000000	0.573333	0.728814	0.942601
Random Forest	0.990196	0.673333	0.801587	0.955157

```
1 CORPUS12 = ' '.join(newdf['text'])
2 CORPUS12
```

→ 'Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat... Ok lar... Joking wif u oni... Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's U dun say so early hor... U c already then say... Nah I don't think he goes to usf, he lives around here though FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, £1.50 to rcv Even my brother is not like to speak with me. They treat me like aids patient. As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press \*9 to copy your friends Callertune WINNER!! As a valued network customer you have been selected to receivea £900 prize reward! To claim call 09061701461. Claim code KL341. Valid 12 hours only. Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030 I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today. SIX chances to win CASH! From 100 to 20,000 pounds txt> CSH11 and send to 87575. Cost 150p/day, 6days, 16+ TsandCs apply Reply HL 4 info URGENT! You have won a 1 week FREE membership in our £100,000 Prize Jackpot! Txt the word: CLAIM to No: 81010 T&C [www.dbuk.net](http://www.dbuk.net) LCCLTD POBOX 4403LDNW1A7RW18 I've been searching for the right words

to thank you for this breather. I promise i wont take your help for granted and will fulfil my promise. You have been wonderful and a blessing at all times. I HAVE A DATE ON SUNDAY WITH WILL!! XXXMobileMovieClub: To use your credit, click the WAP link in the next txt message or click here>> <http://wap>. xxxmobilemovieclub.com? n=QJKGIGHJJGCBL Oh k...i'm watching here:) Eh u remember how 2 spell his name... Yes i did. He v naughty make until i v wet. Fine if that's the way u feel. That's the way its gotta b England v Macedonia - dont miss the goals/team news. Txt ur national team to 87077 eg ENGLAND to 87077 Try:WALES, SCOTLAND 4txt/£1.20 POBOXox36504W45WQ 16+ Is that seriously how you spell his name? I'm going to try for 2 months ha ha only joking So I\_ pay first lar... Then when is da stock comin... Aft i finish my lunch then i go str down lor. Ard 3 smth lor. U finish ur lunch already? FFFFFFFF. Alright no way I can meet up with you sooner? Just forced myself to eat a slice. I'm really not hungry tho. This sucks. Mark is getting worried. He knows I'm sick when I turn down pizza. Lol Lol your always so convincing. Did you catch the bus ? Are you frying an egg ? Did you make a tea? Are you eating your mom's left over dinner ? Do you feel my Love ? I'm back &; we're packing the car now, I'll let you know if there's room Ahhh. Work. I vaguely remember that! What does it feel like? Lol Wait that's still not all that clear, were you not sure about me being sarcastic or that that's why x doesn't want to live with us Yeah he got in at 2 and was v apologetic. n had fallen out and she was actin like spoilt child and he got caught up in that. Till 2! But we won't go there! Not doing too badly cheers. You? K tell me anything about you. For fear of fainting with the of all that housework you just did? Quick have a cuppa Thanks for your subscription to Ringtone UK your mobile will be charged £5/month Please confirm by replying YES or NO. If you reply NO you will not be charged Yup... Ok i go home look at the timings then i msg I\_ again... Xuhui going to learn on 2nd may too but her lesson is at 8am Oops, I'll let you know when my roommate's done I see the letter B on my car Anything lor... U decide... Hello! How's you and how did saturday go? I was just texting to see if you'd decided to do anything tomo. Not that i'm trying to invite myself or anything! Pls go ahead with watts. I just wanted to be sure. Do have a great weekend. Abiola Did I forget to tell you ? I want you , I need you, I crave you ... But most of all ... I love you my sweet Arabian steed ... Mmmmmm ... Yummy 07732584351 - Rodger Burns - MSG = We tried to call you re your reply to our sms for a free nokia mobile + free camcorder. Please call now 08000930705 for delivery tomorrow WHO ARE YOU SEEING? Great! I hope you like your man well endowed. I am <#> inches... No calls..messages..missed calls Didn't you get hep b immunisation in nigeria. Fair enough, anything going on? Yeah hopefully, if tyler can't do it I could maybe ask around a bit U don't know how stubborn T am T didn't even want to go to the hospital T kent telling Mark T'm

```
1 import re
2 #text='&&&&&akhtar age is 14 and??? !!!;;;;,saeed age is 15
3
4 corpus=re.sub(r'[^a-zA-Z0-9\s]', '', CORPUS12)
5
6 corpus
```

→ 'Go until jurong point crazy Available only in bugis n great world la e buffet Cine there got amore wat Ok lar Joking wif u oni Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005 Text FA to 87121 to receive entry questionstd txt rateTCs apply 08452810075over18s U dun say so early hor U c already then say Nah I dont think he goes to usf he lives around here though FreeMsg Hey there darling its been 3 weeks now and no word back Id like some fun you up for it still Tb ok XxX std

chgs to send 150 to rcv Even my brother is not like to speak with me They treat me like aids patent As per your request Melle Melle Oru Minnaminunginte Nurungu Vettam has been set as your callertune for all Callers Press 9 to copy your friends Callertune WINNER As a valued network customer you have been selected to receive a 900 prize reward To claim call 09061701461 Claim code KL341 Valid 12 hours only Had your mobile 11 months or more U R entitled to Update to the latest colour mobiles with camera for Free Call The Mobile Update Co FREE on 08002986030 Im gonna be home soon and i dont want to talk about this stuff anymore tonight k Ive cried enough today SIX chances to win CASH From 100 to 20000 pounds txt CSH11 and send to 87575 Cost 150pday 6days 16 TsandCs apply Reply HL 4 info URGENT You have won a 1 week FREE membership in our 100000 Prize Jackpot Txt the word CLAIM to No 81010 TC wwwdbuknet LCCLTD POBOX 4403LDNW1A7RW18 Ive been searching for the right words to thank you for this breather I promise i wont take your help for granted and will fulfil my promise You have been wonderful and a blessing at all times I HAVE A DATE ON SUNDAY WITH WILL XXXMobileMovieClub To use your credit click the WAP link in the next txt message or click here httpwap xxxmobilemovieclubcommQJKGIGHJJGCBL Oh kim watching here Eh u remember how 2 spell his name Yes i did He v naughty make until i v wet Fine if thats the way u feel Thats the way its gotta b England v Macedonia dont miss the goalsteam news Txt ur national team to 87077 eg ENGLAND to 87077 TryWALES SCOTLAND 4txt120 POBOXox36504W45WQ 16 Is that seriously how you spell his name Im going to try for 2 months ha ha only joking So pay first lar Then when is da stock comin Aft i finish my lunch then i go str down lor Ard 3 smth lor U finish ur lunch already Ffffffff Alright no way I can meet up with you sooner Just forced myself to eat a slice Im really not hungry tho This sucks Mark is getting worried He knows Im sick when I turn down pizza Lol Lol your always so convincing Did you catch the bus Are you frying an egg Did you make a tea Are you eating your moms left over dinner Do you feel my Love Im back amp were packing the car now Ill let you know if theres room Ahhh Work I vaguely remember that What does it feel like Lol Wait thats still not all that clear were you not sure about me being sarcastic or that thats why x doesnt want to live with us Yeah he got in at 2 and was v apologetic n had fallen out and she was actin like spoilt child and he got caught up in that Till 2 But we wont go there Not doing too badly cheers You K tell me anything about you For fear of fainting with the of all that housework you just did Quick have a cuppa Thanks for your subscription to Ringtone UK your mobile will be charged 5month Please confirm by replying YES or NO If you reply NO you will not be charged Yup Ok i go home look at the timings then i msg again Xuhui going to learn on 2nd may too but her lesson is at 8am Oops Ill let you know when my roommates done I see the letter B on my car Anything lor U decide Hello Hows you and how did saturday go I was just texting to see if youd decided to do anything tomo Not that im trying to invite myself or anything Pls go ahead with watts I just wanted to be sure Do have a great weekend Abiola Did I forget to tell you I want you I need you I crave you But most of all I love you my sweet Arabian steed Mmmmmm Yummy 07732584351 Rodger Burns MSG We tried to call you re your reply to our sms for a free nokia mobile free camcorder Please call now 08000930705 for delivery tomorrow WHO ARE YOU SEEING Great I hope you like your man well endowed I am ltgt inches No callsmessagesmissed calls Didnt you get hep b immunisation in nigeria Fair enough anything going on Yeah hopefully if tyler cant do it I could maybe ask around a bit U dont know how stubborn I am I didnt even want to go to the hospital I kept telling Mark Im not a weak sucker Hospitals are for weak suckers What you thought about me First time you saw me in class A gram usually runs like ltgt a half eighth is smarter though and gets you almost a whole second gram for ltgt K fyi x has a ride

```
1 def extract_ngrams(corpus, ngram_range=(1,1)):
```

```
2
```

```
3     vectorizer = CountVectorizer(ngram_range=ngram_range)
4
5     # Fit on corpus and transform both corpus and new docume
6
7     X_corpus = vectorizer.fit_transform(corpus)
8     #X_new_doc = vectorizer.transform(new_doc)
9 # Get feature names (n-grams)
10    feature_names = vectorizer.get_feature_names_out()
11
12    # Convert the results to arrays
13    corpus_ngrams = X_corpus.toarray()
14    #new_doc_ngrams = X_new_doc.toarray()
15
16    return feature_names, corpus_ngrams
17
18 # Unigrams (1-gram)
19 unigrams, corpus_unigrams = extract_ngrams(CORPUS, ngram_ran
20 print("Unigrams:")
21 print(unigrams)
22 print("\n")
23 print("corpus matrix is\n", corpus_unigrams)
24 #print(new_doc_unigrams)
25
```

ValueError Traceback (most recent call last)

```

Cell In[82], line 19
    16     return feature_names, corpus_ngrams
    18 # Unigrams (1-gram)
--> 19 unigrams, corpus_unigrams = extract_ngrams(CORPUS, ngram_range=(1, 1))
    20 print("Unigrams:")
    21 print(unigrams)

Cell In[82], line 7, in extract_ngrams(corpus, ngram_range)
    3     vectorizer = CountVectorizer(ngram_range=ngram_range)
    5     # Fit on corpus and transform both corpus and new document
----> 7     X_corpus = vectorizer.fit_transform(corpus)
    8     #X_new_doc = vectorizer.transform(new_doc)
    9 # Get feature names (n-grams)
   10     feature_names = vectorizer.get_feature_names_out()

File ~/anaconda3\Lib\site-packages\sklearn\base.py:1151, in _fit_context.
<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
    1144     estimator._validate_params()
    1146 with config_context(
    1147     skip_parameter_validation=(
    1148         prefer_skip_nested_validation or global_skip_validation
    1149     )
    1150 ):
-> 1151     return fit_method(estimator, *args, **kwargs)

File ~/anaconda3\Lib\site-packages\sklearn\feature_extraction\text.py:1361, in CountVectorizer.fit_transform(self, raw_documents, y)
    1357 # We intentionally don't call the transform method to make
    1358 # fit_transform overridable without unwanted side effects in
    1359 # TfidfVectorizer.
    1360 if isinstance(raw_documents, str):
-> 1361     raise ValueError(
    1362         "Iterable over raw text documents expected, string object received."
    1363     )
    1365 self._validate_ngram_range()
    1366 self._warn_for_unused_params()

```

ValueError: Iterable over raw text documents expected, string object received.

1 Start coding or generate with AI.

## ▼ Notebook: 28) n\_gram of spam classification.ipynb

```

1 import pandas as pd
2 import numpy as np
3 import seaborn as sns

```

```
4 import matplotlib.pyplot as plt
5 from sklearn.feature_extraction.text import CountVectorizer
6 from sklearn.naive_bayes import MultinomialNB
7 from sklearn.linear_model import LogisticRegression
8 from sklearn.svm import SVC
9 from sklearn.ensemble import RandomForestClassifier
10 from sklearn.model_selection import train_test_split
11 from sklearn.metrics import (accuracy_score, precision_score
12                                         recall_score, f1_score, confusio
```

```
1
2 # Load data
3 df = pd.read_csv('spam.csv', encoding='latin-1')[['v1', 'v2']
4 df.columns = ['label', 'text']
5 df['label'] = df['label'].map({'ham': 0, 'spam': 1})
6
```

```
1 # Split data
2 train_texts, test_texts, train_labels, test_labels = train_t
3     df['text'], df['label'], test_size=0.2, random_state=42
4 )
```

## ▼ Unigram

```
1
2 # Classifiers to compare
3 classifiers = {
4     'Naive Bayes': MultinomialNB(),
5     'Logistic Regression': LogisticRegression(max_iter=1000)
6     'SVM': SVC(kernel='linear', probability=True),
7     'Random Forest': RandomForestClassifier(n_estimators=100
8 }
9 # N-gram combinations
10 ngram_configs = {
11     'Unigram (U)': (1, 1)
12
13 }
```

```
14
15 # Store results
16 results = []
17
18 # Process each classifier and n-gram
19 for clf_name, clf in classifiers.items():
20     for ngram_name, ngram_range in ngram_configs.items():
21         # Vectorize text
22         vectorizer = CountVectorizer(ngram_range=ngram_range)
23         X_train = vectorizer.fit_transform(train_texts)
24         clf.fit(X_train, train_labels)
25
26         # Predictions
27         X_test = vectorizer.transform(test_texts)
28         y_pred = clf.predict(X_test)
29
30         # Metrics
31         accuracy = accuracy_score(test_labels, y_pred)
32         precision = precision_score(test_labels, y_pred)
33         recall = recall_score(test_labels, y_pred)
34         f1 = f1_score(test_labels, y_pred)
35
36         # Store results
37         results.append({
38             'Classifier': clf_name,
39             'N-Gram': ngram_name,
40             'Accuracy': accuracy,
41             'Precision': precision,
42             'Recall': recall,
43             'F1': f1
44         })
45
46 # Convert to DataFrame
47 results_df = pd.DataFrame(results)
48
49 # Print top models
50 print("\n==== Top 4 Models of Unigram ===")
51 print(results_df.sort_values(by='F1', ascending=False).head()
52
```

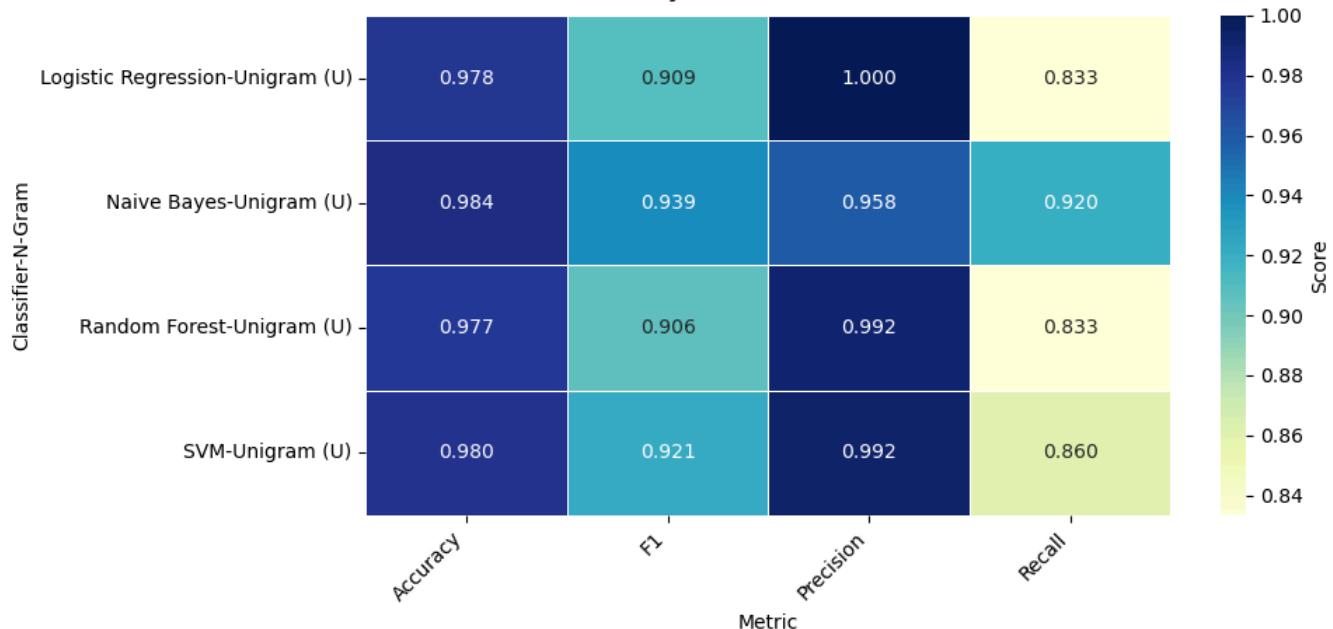
```
53 # Melt for visualization
54 melted_df = pd.melt(results_df,
55                      id_vars=['Classifier', 'N-Gram'],
56                      value_vars=['Accuracy', 'Precision', 'Re-
57                      var_name='Metric',
58                      value_name='Score')
59
60 # Create heatmap
61 plt.figure(figsize=(10, 5))
62 heatmap_data = melted_df.pivot_table(
63     index=['Classifier', 'N-Gram'],
64     columns='Metric',
65     values='Score'
66 )
67
68 sns.heatmap(
69     heatmap_data,
70     annot=True,
71     fmt=".3f",
72     cmap="YlGnBu",
73     linewidths=0.5,
74     cbar_kws={'label': 'Score'}
75 )
76
77 plt.title('Performance Metrics by Classifier and N-Gram Comb
78 plt.xticks(rotation=45, ha='right')
79 plt.yticks(rotation=0)
80 plt.tight_layout()
81 plt.show()
82
83 # Optional: Save results to CSV
84 results_df.to_csv('spam_detection_metrics.csv', index=False)
```



== Top 4 Models of Unigram ==

Classifier	N-Gram	Accuracy	Precision	Recall	F1
Naive Bayes	Unigram (U)	0.983857	0.958333	0.92	0.938776
SVM	Unigram (U)	0.980269	0.992308	0.86	0.921429
Logistic Regression	Unigram (U)	0.977578	1	0.833333	0.909091
Random Forest	Unigram (U)	0.976682	0.992063	0.833333	0.905797

Performance Metrics by Classifier and N-Gram Combination



## ▼ Bigram

```
1
2 # Classifiers to compare
3 classifiers = {
4     'Naive Bayes': MultinomialNB(),
5     'Logistic Regression': LogisticRegression(max_iter=1000)
6     'SVM': SVC(kernel='linear', probability=True),
7     'Random Forest': RandomForestClassifier(n_estimators=100
8 }
9 # N-gram combinations
10 ngram_configs = {
11     'Bigram (B)': (2, 2)
```

```
12
13 }
14
15 # Store results
16 results = []
17
18 # Process each classifier and n-gram
19 for clf_name, clf in classifiers.items():
20     for ngram_name, ngram_range in ngram_configs.items():
21         # Vectorize text
22         vectorizer = CountVectorizer(ngram_range=ngram_range)
23         X_train = vectorizer.fit_transform(train_texts)
24         clf.fit(X_train, train_labels)
25
26         # Predictions
27         X_test = vectorizer.transform(test_texts)
28         y_pred = clf.predict(X_test)
29
30         # Metrics
31         accuracy = accuracy_score(test_labels, y_pred)
32         precision = precision_score(test_labels, y_pred)
33         recall = recall_score(test_labels, y_pred)
34         f1 = f1_score(test_labels, y_pred)
35
36         # Store results
37         results.append({
38             'Classifier': clf_name,
39             'N-Gram': ngram_name,
40             'Accuracy': accuracy,
41             'Precision': precision,
42             'Recall': recall,
43             'F1': f1
44         })
45
46 # Convert to DataFrame
47 results_df = pd.DataFrame(results)
48
49 # Print top models
50 print("\n==== Top 4 Models of Bigram ===")
```

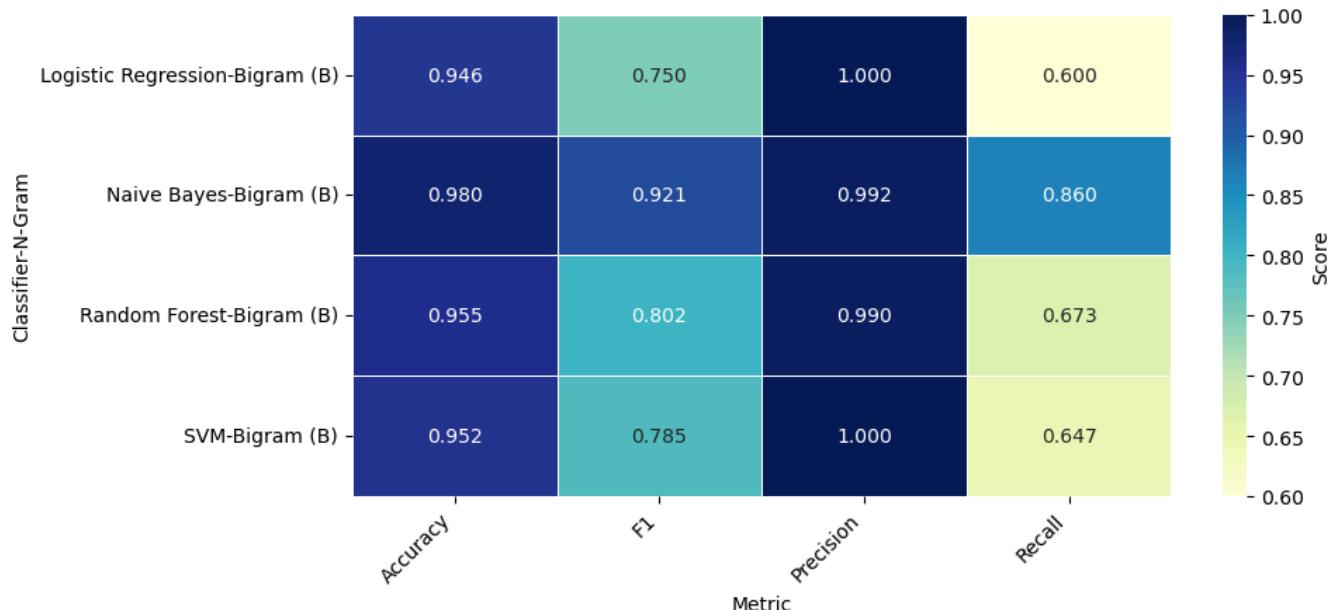
```
51 print(results_df.sort_values(by='F1', ascending=False).head()
52
53 # Melt for visualization
54 melted_df = pd.melt(results_df,
55                      id_vars=['Classifier', 'N-Gram'],
56                      value_vars=['Accuracy', 'Precision', 'Re
57                      var_name='Metric',
58                      value_name='Score')
59
60 # Create heatmap
61 plt.figure(figsize=(10, 5))
62 heatmap_data = melted_df.pivot_table(
63     index=['Classifier', 'N-Gram'],
64     columns='Metric',
65     values='Score'
66 )
67
68 sns.heatmap(
69     heatmap_data,
70     annot=True,
71     fmt=".3f",
72     cmap="YlGnBu",
73     linewidths=0.5,
74     cbar_kws={'label': 'Score'}
75 )
76
77 plt.title('Performance Metrics by Classifier and N-Gram Comb
78 plt.xticks(rotation=45, ha='right')
79 plt.yticks(rotation=0)
80 plt.tight_layout()
81 plt.show()
82
83 # Optional: Save results to CSV
84 results_df.to_csv('spam_detection_metrics.csv', index=False)
```



== Top 4 Models of Bigram ==

Classifier	N-Gram	Accuracy	Precision	Recall	F1
Naive Bayes	Bigram (B)	0.980269	0.992308	0.86	0.921429
Random Forest	Bigram (B)	0.955157	0.990196	0.673333	0.801587
SVM	Bigram (B)	0.952466	1	0.646667	0.785425
Logistic Regression	Bigram (B)	0.946188	1	0.6	0.75

Performance Metrics by Classifier and N-Gram Combination



## ▼ Trigram

```
1
2 # Classifiers to compare
3 classifiers = {
4     'Naive Bayes': MultinomialNB(),
5     'Logistic Regression': LogisticRegression(max_iter=1000)
6     'SVM': SVC(kernel='linear', probability=True),
7     'Random Forest': RandomForestClassifier(n_estimators=100
8 }
9 # N-gram combinations
10 ngram_configs = {
11     'Trigram (T)': (3, 3)
```

```
12
13 }
14
15 # Store results
16 results = []
17
18 # Process each classifier and n-gram
19 for clf_name, clf in classifiers.items():
20     for ngram_name, ngram_range in ngram_configs.items():
21         # Vectorize text
22         vectorizer = CountVectorizer(ngram_range=ngram_range)
23         X_train = vectorizer.fit_transform(train_texts)
24         clf.fit(X_train, train_labels)
25
26         # Predictions
27         X_test = vectorizer.transform(test_texts)
28         y_pred = clf.predict(X_test)
29
30         # Metrics
31         accuracy = accuracy_score(test_labels, y_pred)
32         precision = precision_score(test_labels, y_pred)
33         recall = recall_score(test_labels, y_pred)
34         f1 = f1_score(test_labels, y_pred)
35
36         # Store results
37         results.append({
38             'Classifier': clf_name,
39             'N-Gram': ngram_name,
40             'Accuracy': accuracy,
41             'Precision': precision,
42             'Recall': recall,
43             'F1': f1
44         })
45
46 # Convert to DataFrame
47 results_df = pd.DataFrame(results)
48
49 # Print top models
50 print("\n==== Top 4 Models of Trigram ===")
```

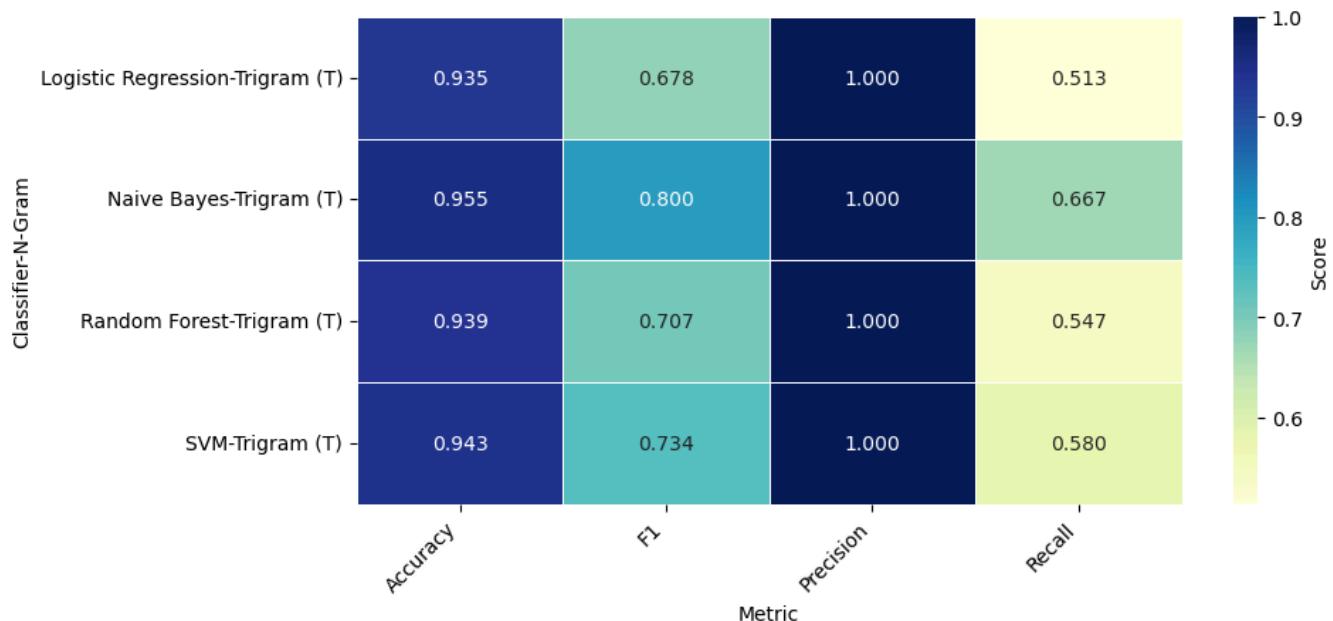
```
51 print(results_df.sort_values(by='F1', ascending=False).head()
52
53 # Melt for visualization
54 melted_df = pd.melt(results_df,
55                      id_vars=['Classifier', 'N-Gram'],
56                      value_vars=['Accuracy', 'Precision', 'Re
57                      var_name='Metric',
58                      value_name='Score')
59
60 # Create heatmap
61 plt.figure(figsize=(10, 5))
62 heatmap_data = melted_df.pivot_table(
63     index=['Classifier', 'N-Gram'],
64     columns='Metric',
65     values='Score'
66 )
67
68 sns.heatmap(
69     heatmap_data,
70     annot=True,
71     fmt=".3f",
72     cmap="YlGnBu",
73     linewidths=0.5,
74     cbar_kws={'label': 'Score'}
75 )
76
77 plt.title('Performance Metrics by Classifier and N-Gram Comb
78 plt.xticks(rotation=45, ha='right')
79 plt.yticks(rotation=0)
80 plt.tight_layout()
81 plt.show()
82
83 # Optional: Save results to CSV
84 results_df.to_csv('spam_detection_metrics.csv', index=False)
```



== Top 4 Models of Trigram ==

Classifier	N-Gram	Accuracy	Precision	Recall	F1
Naive Bayes	Trigram (T)	0.955157	1	0.666667	0.8
SVM	Trigram (T)	0.943498	1	0.58	0.734177
Random Forest	Trigram (T)	0.939013	1	0.546667	0.706897
Logistic Regression	Trigram (T)	0.934529	1	0.513333	0.678414

Performance Metrics by Classifier and N-Gram Combination



## ▼ Unigram + Bigram

```
1
2 # Classifiers to compare
3 classifiers = {
4     'Naive Bayes': MultinomialNB(),
5     'Logistic Regression': LogisticRegression(max_iter=1000)
6     'SVM': SVC(kernel='linear', probability=True),
7     'Random Forest': RandomForestClassifier(n_estimators=100
8 }
9 # N-gram combinations
10 ngram_configs = {
11     'U+B': (1, 2)
```

```
12
13 }
14
15 # Store results
16 results = []
17
18 # Process each classifier and n-gram
19 for clf_name, clf in classifiers.items():
20     for ngram_name, ngram_range in ngram_configs.items():
21         # Vectorize text
22         vectorizer = CountVectorizer(ngram_range=ngram_range)
23         X_train = vectorizer.fit_transform(train_texts)
24         clf.fit(X_train, train_labels)
25
26         # Predictions
27         X_test = vectorizer.transform(test_texts)
28         y_pred = clf.predict(X_test)
29
30         # Metrics
31         accuracy = accuracy_score(test_labels, y_pred)
32         precision = precision_score(test_labels, y_pred)
33         recall = recall_score(test_labels, y_pred)
34         f1 = f1_score(test_labels, y_pred)
35
36         # Store results
37         results.append({
38             'Classifier': clf_name,
39             'N-Gram': ngram_name,
40             'Accuracy': accuracy,
41             'Precision': precision,
42             'Recall': recall,
43             'F1': f1
44         })
45
46 # Convert to DataFrame
47 results_df = pd.DataFrame(results)
48
49 # Print top models
50 print("\n==== Top 4 Models of Unigram + Bigram ===")
```

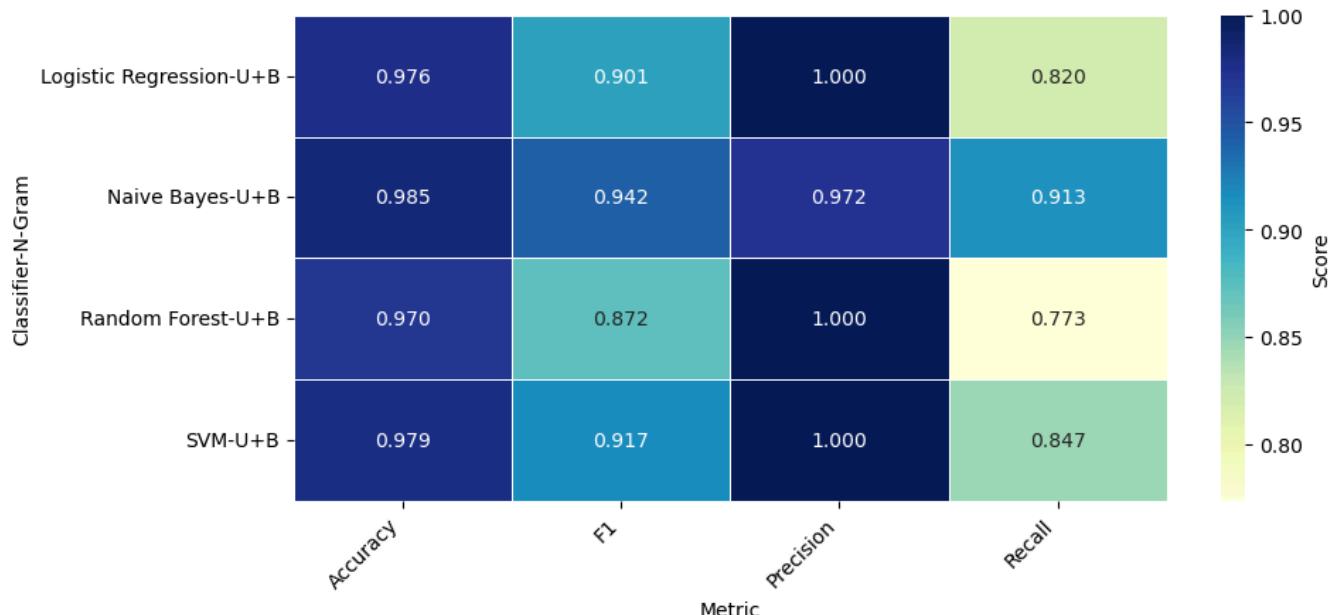
```
51 print(results_df.sort_values(by='F1', ascending=False).head()
52
53 # Melt for visualization
54 melted_df = pd.melt(results_df,
55                      id_vars=['Classifier', 'N-Gram'],
56                      value_vars=['Accuracy', 'Precision', 'Re
57                      var_name='Metric',
58                      value_name='Score')
59
60 # Create heatmap
61 plt.figure(figsize=(10, 5))
62 heatmap_data = melted_df.pivot_table(
63     index=['Classifier', 'N-Gram'],
64     columns='Metric',
65     values='Score'
66 )
67
68 sns.heatmap(
69     heatmap_data,
70     annot=True,
71     fmt=".3f",
72     cmap="YlGnBu",
73     linewidths=0.5,
74     cbar_kws={'label': 'Score'}
75 )
76
77 plt.title('Performance Metrics by Classifier and N-Gram Comb
78 plt.xticks(rotation=45, ha='right')
79 plt.yticks(rotation=0)
80 plt.tight_layout()
81 plt.show()
82
83 # Optional: Save results to CSV
84 results_df.to_csv('spam_detection_metrics.csv', index=False)
```



== Top 4 Models of Unigram + Bigram ==

Classifier	N-Gram	Accuracy	Precision	Recall	F1
Naive Bayes	U+B	0.984753	0.971631	0.913333	0.941581
SVM	U+B	0.979372	1	0.846667	0.916968
Logistic Regression	U+B	0.975785	1	0.82	0.901099
Random Forest	U+B	0.969507	1	0.773333	0.87218

Performance Metrics by Classifier and N-Gram Combination



## ▼ Unigram + Trigram

```
1
2 # Classifiers to compare
3 classifiers = {
4     'Naive Bayes': MultinomialNB(),
5     'Logistic Regression': LogisticRegression(max_iter=1000)
6     'SVM': SVC(kernel='linear', probability=True),
7     'Random Forest': RandomForestClassifier(n_estimators=100
8 }
9 # N-gram combinations
10 ngram_configs = {
11     'U+T': (1, 3)
```

```
12
13 }
14
15 # Store results
16 results = []
17
18 # Process each classifier and n-gram
19 for clf_name, clf in classifiers.items():
20     for ngram_name, ngram_range in ngram_configs.items():
21         # Vectorize text
22         vectorizer = CountVectorizer(ngram_range=ngram_range)
23         X_train = vectorizer.fit_transform(train_texts)
24         clf.fit(X_train, train_labels)
25
26         # Predictions
27         X_test = vectorizer.transform(test_texts)
28         y_pred = clf.predict(X_test)
29
30         # Metrics
31         accuracy = accuracy_score(test_labels, y_pred)
32         precision = precision_score(test_labels, y_pred)
33         recall = recall_score(test_labels, y_pred)
34         f1 = f1_score(test_labels, y_pred)
35
36         # Store results
37         results.append({
38             'Classifier': clf_name,
39             'N-Gram': ngram_name,
40             'Accuracy': accuracy,
41             'Precision': precision,
42             'Recall': recall,
43             'F1': f1
44         })
45
46 # Convert to DataFrame
47 results_df = pd.DataFrame(results)
48
49 # Print top models
50 print("\n==== Top 4 Models of Unigram + Trigram ===")
```

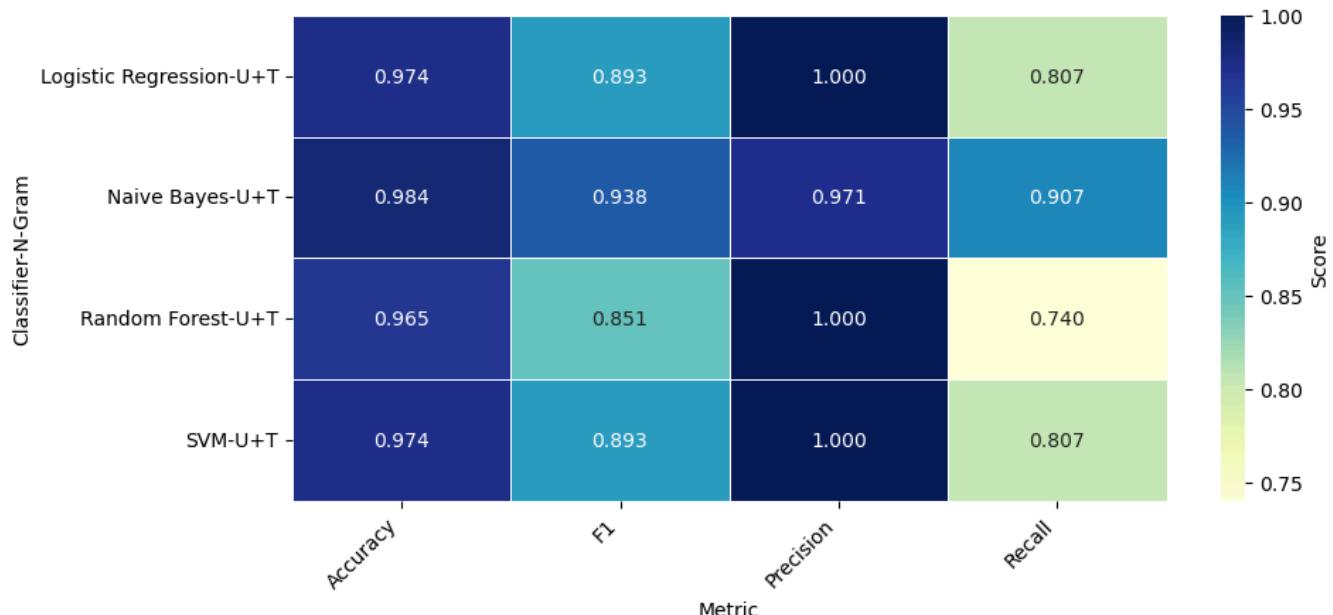
```
51 print(results_df.sort_values(by='F1', ascending=False).head()
52
53 # Melt for visualization
54 melted_df = pd.melt(results_df,
55                      id_vars=['Classifier', 'N-Gram'],
56                      value_vars=['Accuracy', 'Precision', 'Re
57                      var_name='Metric',
58                      value_name='Score')
59
60 # Create heatmap
61 plt.figure(figsize=(10, 5))
62 heatmap_data = melted_df.pivot_table(
63     index=['Classifier', 'N-Gram'],
64     columns='Metric',
65     values='Score'
66 )
67
68 sns.heatmap(
69     heatmap_data,
70     annot=True,
71     fmt=".3f",
72     cmap="YlGnBu",
73     linewidths=0.5,
74     cbar_kws={'label': 'Score'}
75 )
76
77 plt.title('Performance Metrics by Classifier and N-Gram Comb
78 plt.xticks(rotation=45, ha='right')
79 plt.yticks(rotation=0)
80 plt.tight_layout()
81 plt.show()
82
83 # Optional: Save results to CSV
84 results_df.to_csv('spam_detection_metrics.csv', index=False)
```



== Top 4 Models of Unigram + Trigram ==

Classifier	N-Gram	Accuracy	Precision	Recall	F1
Naive Bayes	U+T	0.983857	0.971429	0.906667	0.937931
Logistic Regression	U+T	0.973991	1	0.806667	0.892989
SVM	U+T	0.973991	1	0.806667	0.892989
Random Forest	U+T	0.965022	1	0.74	0.850575

Performance Metrics by Classifier and N-Gram Combination



## ▼ Bigram + Trigram

```
1
2 # Classifiers to compare
3 classifiers = {
4     'Naive Bayes': MultinomialNB(),
5     'Logistic Regression': LogisticRegression(max_iter=1000)
6     'SVM': SVC(kernel='linear', probability=True),
7     'Random Forest': RandomForestClassifier(n_estimators=100
8 }
9 # N-gram combinations
10 ngram_configs = {
11     'B+T': (2, 3)
```

```
12
13 }
14
15 # Store results
16 results = []
17
18 # Process each classifier and n-gram
19 for clf_name, clf in classifiers.items():
20     for ngram_name, ngram_range in ngram_configs.items():
21         # Vectorize text
22         vectorizer = CountVectorizer(ngram_range=ngram_range)
23         X_train = vectorizer.fit_transform(train_texts)
24         clf.fit(X_train, train_labels)
25
26         # Predictions
27         X_test = vectorizer.transform(test_texts)
28         y_pred = clf.predict(X_test)
29
30         # Metrics
31         accuracy = accuracy_score(test_labels, y_pred)
32         precision = precision_score(test_labels, y_pred)
33         recall = recall_score(test_labels, y_pred)
34         f1 = f1_score(test_labels, y_pred)
35
36         # Store results
37         results.append({
38             'Classifier': clf_name,
39             'N-Gram': ngram_name,
40             'Accuracy': accuracy,
41             'Precision': precision,
42             'Recall': recall,
43             'F1': f1
44         })
45
46 # Convert to DataFrame
47 results_df = pd.DataFrame(results)
48
49 # Print top models
50 print("\n==== Top 4 Models of Bigram + Trigram ===")
```

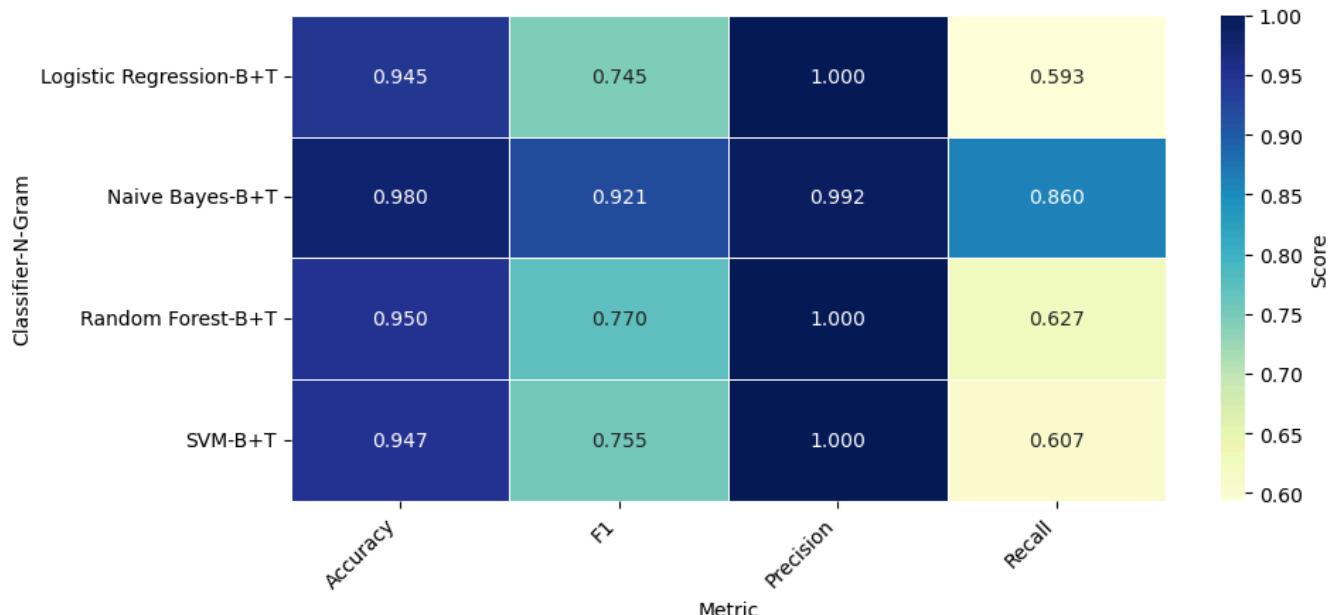
```
51 print(results_df.sort_values(by='F1', ascending=False).head()
52
53 # Melt for visualization
54 melted_df = pd.melt(results_df,
55                      id_vars=['Classifier', 'N-Gram'],
56                      value_vars=['Accuracy', 'Precision', 'Re
57                      var_name='Metric',
58                      value_name='Score')
59
60 # Create heatmap
61 plt.figure(figsize=(10, 5))
62 heatmap_data = melted_df.pivot_table(
63     index=['Classifier', 'N-Gram'],
64     columns='Metric',
65     values='Score'
66 )
67
68 sns.heatmap(
69     heatmap_data,
70     annot=True,
71     fmt=".3f",
72     cmap="YlGnBu",
73     linewidths=0.5,
74     cbar_kws={'label': 'Score'}
75 )
76
77 plt.title('Performance Metrics by Classifier and N-Gram Comb
78 plt.xticks(rotation=45, ha='right')
79 plt.yticks(rotation=0)
80 plt.tight_layout()
81 plt.show()
82
83 # Optional: Save results to CSV
84 results_df.to_csv('spam_detection_metrics.csv', index=False)
```



== Top 4 Models of Bigram + Trigram ==

Classifier	N-Gram	Accuracy	Precision	Recall	F1
Naive Bayes	B+T	0.980269	0.992308	0.86	0.921429
Random Forest	B+T	0.949776	1	0.626667	0.770492
SVM	B+T	0.947085	1	0.606667	0.755187
Logistic Regression	B+T	0.945291	1	0.593333	0.74477

Performance Metrics by Classifier and N-Gram Combination



## ▼ Unigram + Bigram + Trigram

```
1
2 # Classifiers to compare
3 classifiers = {
4     'Naive Bayes': MultinomialNB(),
5     'Logistic Regression': LogisticRegression(max_iter=1000)
6     'SVM': SVC(kernel='linear', probability=True),
7     'Random Forest': RandomForestClassifier(n_estimators=100
8 }
9 # N-gram combinations
10 ngram_configs = {
11     'U+B+T': (1,3)
```

```
12
13 }
14
15 # Store results
16 results = []
17
18 # Process each classifier and n-gram
19 for clf_name, clf in classifiers.items():
20     for ngram_name, ngram_range in ngram_configs.items():
21         # Vectorize text
22         vectorizer = CountVectorizer(ngram_range=ngram_range)
23         X_train = vectorizer.fit_transform(train_texts)
24         clf.fit(X_train, train_labels)
25
26         # Predictions
27         X_test = vectorizer.transform(test_texts)
28         y_pred = clf.predict(X_test)
29
30         # Metrics
31         accuracy = accuracy_score(test_labels, y_pred)
32         precision = precision_score(test_labels, y_pred)
33         recall = recall_score(test_labels, y_pred)
34         f1 = f1_score(test_labels, y_pred)
35
36         # Store results
37         results.append({
38             'Classifier': clf_name,
39             'N-Gram': ngram_name,
40             'Accuracy': accuracy,
41             'Precision': precision,
42             'Recall': recall,
43             'F1': f1
44         })
45
46 # Convert to DataFrame
47 results_df = pd.DataFrame(results)
48
49 # Print top models
50 print("\n==== Top 4 Models of Unigram + Bigram + Trigram ==="
```

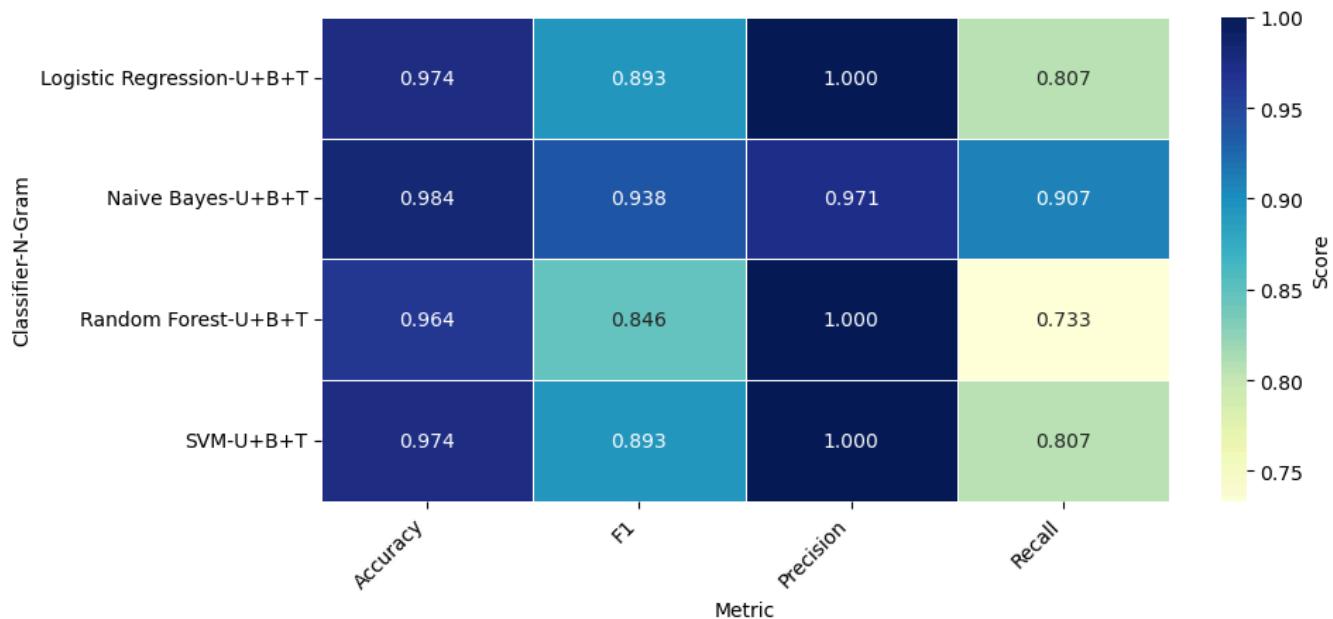
```
51 print(results_df.sort_values(by='F1', ascending=False).head()
52
53 # Melt for visualization
54 melted_df = pd.melt(results_df,
55                      id_vars=['Classifier', 'N-Gram'],
56                      value_vars=['Accuracy', 'Precision', 'Re
57                      var_name='Metric',
58                      value_name='Score')
59
60 # Create heatmap
61 plt.figure(figsize=(10, 5))
62 heatmap_data = melted_df.pivot_table(
63     index=['Classifier', 'N-Gram'],
64     columns='Metric',
65     values='Score'
66 )
67
68 sns.heatmap(
69     heatmap_data,
70     annot=True,
71     fmt=".3f",
72     cmap="YlGnBu",
73     linewidths=0.5,
74     cbar_kws={'label': 'Score'}
75 )
76
77 plt.title('Performance Metrics by Classifier and N-Gram Comb
78 plt.xticks(rotation=45, ha='right')
79 plt.yticks(rotation=0)
80 plt.tight_layout()
81 plt.show()
82
83 # Optional: Save results to CSV
84 results_df.to_csv('spam_detection_metrics.csv', index=False)
```



== Top 4 Models of Unigram + Bigram + Trigram ==

Classifier	N-Gram	Accuracy	Precision	Recall	F1
Naive Bayes	U+B+T	0.983857	0.971429	0.906667	0.937931
Logistic Regression	U+B+T	0.973991	1	0.806667	0.892989
SVM	U+B+T	0.973991	1	0.806667	0.892989
Random Forest	U+B+T	0.964126	1	0.733333	0.846154

Performance Metrics by Classifier and N-Gram Combination



1 Start coding or generate with AI.

All in One N\_Grams (Unigram , Bigram , Trigram , U+B , U+B+T , B+T , U+B+T )

```
1
2
3 # Load data
4 df = pd.read_csv('spam.csv', encoding='latin-1')[['v1', 'v2']
5 df.columns = ['label', 'text']
6 df['label'] = df['label'].map({'ham': 0, 'spam': 1})
7
```

```
8 # Split data
9 train_texts, test_texts, train_labels, test_labels = train_
10     df['text'], df['label'], test_size=0.2, random_state=42
11 )
12
13 # Classifiers to compare
14 classifiers = {
15     'Naive Bayes': MultinomialNB(),
16     'Logistic Regression': LogisticRegression(max_iter=1000
17     'SVM': SVC(kernel='linear', probability=True),
18     'Random Forest': RandomForestClassifier(n_estimators=10
19 }
20
21 # N-gram combinations
22 ngram_configs = {
23     'Unigram (U)': (1, 1),
24     'Bigram (B)': (2, 2),
25     'Trigram (T)': (3, 3),
26     'U+B': (1, 2),
27     'U+T': (1, 3),
28     'B+T': (2, 3),
29     'U+B+T': (1, 3)
30 }
31
32 # Store results
33 results = []
34
35 # Process each classifier and n-gram
36 for clf_name, clf in classifiers.items():
37     for ngram_name, ngram_range in ngram_configs.items():
38         # Vectorize text
39         vectorizer = CountVectorizer(ngram_range=ngram_range)
40         X_train = vectorizer.fit_transform(train_texts)
41         clf.fit(X_train, train_labels)
42
43         # Predictions
44         X_test = vectorizer.transform(test_texts)
45         y_pred = clf.predict(X_test)
46
```

```
47     # Metrics
48     accuracy = accuracy_score(test_labels, y_pred)
49     precision = precision_score(test_labels, y_pred)
50     recall = recall_score(test_labels, y_pred)
51     f1 = f1_score(test_labels, y_pred)
52
53     # Store results
54     results.append({
55         'Classifier': clf_name,
56         'N-Gram': ngram_name,
57         'Accuracy': accuracy,
58         'Precision': precision,
59         'Recall': recall,
60         'F1': f1
61     })
62
63 # Convert to DataFrame
64 results_df = pd.DataFrame(results)
65
66 # Print top models
67 print("\n==== Top 5 Models ===")
68 print(results_df.sort_values(by='F1', ascending=False).head(5))
69
70 # Melt for visualization
71 melted_df = pd.melt(results_df,
72                     id_vars=['Classifier', 'N-Gram'],
73                     value_vars=['Accuracy', 'Precision', 'Recall'],
74                     var_name='Metric',
75                     value_name='Score')
76
77 # Create heatmap
78 plt.figure(figsize=(10, 5))
79 heatmap_data = melted_df.pivot_table(
80     index=['Classifier', 'N-Gram'],
81     columns='Metric',
82     values='Score'
83 )
84
85 sns.heatmap(
```

```
86     heatmap_data,
87     annot=True,
88     fmt=".3f",
89     cmap="YlGnBu",
90     linewidths=0.5,
91     cbar_kws={'label': 'Score'}
92 )
93
94 plt.title('Performance Metrics by Classifier and N-Gram Com
95 plt.xticks(rotation=45, ha='right')
96 plt.yticks(rotation=0)
97 plt.tight_layout()
98 plt.show()
99
100 # Optional: Save results to CSV
101 results_df.to_csv('spam_detection_metrics.csv', index=False)
```

## ▼ 1. Performance Summary Statistics

```
1 # Calculate mean metrics across all n-grams for each classif
2 summary_stats = results_df.groupby('Classifier').agg({
3     'Accuracy': 'mean',
4     'Precision': 'mean',
5     'Recall': 'mean',
6     'F1': 'mean'
7 }).sort_values(by='F1', ascending=False)
8
9 print("\n==== Classifier Performance Summary ===")
10 print(summary_stats.to_markdown(floatfmt=".3f"))
```

1 Start coding or generate with AI.

## ▼ Best Model Identification

```
1 best_overall = summary_stats.idxmax()
2 print("\n==== Best Classifier by Metric ===")
```

```
3 print(best_overall.to_markdown())
```

1 Start coding or generate with AI.

1 Start coding or generate with AI.

## ✗ Notebook: 29) n\_gram of urdu dataset.ipynb

```
1 import pandas as pd
2 import numpy as np
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5 from sklearn.feature_extraction.text import CountVectorizer
6 from sklearn.naive_bayes import MultinomialNB
7 from sklearn.linear_model import LogisticRegression
8 from sklearn.svm import SVC
9 from sklearn.ensemble import RandomForestClassifier
10 from sklearn.model_selection import train_test_split
11 from sklearn.metrics import (accuracy_score, precision_score,
12                               recall_score, f1_score, confusio
```

```
1
2 # Load data
3 df=pd.read_csv('imdb_urdu_reviews_test.csv')
4 df
```



## review sentiment

0	... یہ بے گھر خواتین کے بارے میں ایک دستاویزی فلم	negative
1	... بالکل بھی اچھے ، یہ کام نہیں کیا گیا ، پوری فلم ص	negative
2	... یہ عجیب بات ہے کہ کچھ لوگوں کا کیا حشر ہوتا ہے	negative
3	... اور یہ خاص طور پر وکیلوں اور پولیس اہلکاروں کے	positive
4	... پہلے ، ایک وضاحت: میری سرخی کے باوجود ، میں اس	positive
...	...	...
9995	اگر آپ چیخنا چاہتے ہو یا بڑے استوڈیو بار براڈ	positive
9996	... براہ راست یہ ایک محض ایک چھوٹی سی چھوٹی چھوٹی	positive
9997	... میں نے اس فلم کو کل رات آدھی رات کو چکرے سے پہ	negative
9998	... دیکھنا کوئی آسان فلم نہیں ہے - یہ ساڑھے تین گھے	positive
9999	... ناگرا قدامت پسند ہندوستانی خاندان سے تعلق رکھت	positive

10000 rows × 2 columns

## 1 df.columns

Index(['review', 'sentiment'], dtype='object')

1 df['sentiment'] = df['sentiment'].map({'positive': 0, 'negat

## 1 df



## review sentiment

0	... یہ بے گھر خواتین کے بارے میں ایک دستاویزی فلم	1
1	... بالکل بھی اچھے ، یہ کام نہیں کیا گیا ، پوری فلم ص	1
2	... یہ عجیب بات ہے کہ کچھ لوگوں کا کیا حشر ہوتا ہے	1
3	... اور یہ خاص طور پر وکیلوں اور پولیس اہلکاروں کے	0
4	... پہلے ، ایک وضاحت: میری سرخی کے باوجود ، میں اس	0
...	...	...
9995	... اگر آپ چیخنا چاہتے ہو یا بڑے استوڈیو بار براڈ	0
9996	... براہ راست یہ ایک محض ایک چھوٹی سی چھوٹی چھوٹی	0
9997	... میں نے اس فلم کو کل رات آدھی رات کو چکے سے پہ	1
9998	... دیکھنا کوئی آسان فلم نہیں ہے - یہ ساڑھے تین گھ	0
9999	... ناگرا قدامت پسند ہندوستانی خاندان سے تعلق رکھت	0

10000 rows × 2 columns

```

1 # Split data
2 train_texts, test_texts, train_labels, test_labels = train_t
3     df['review'], df['sentiment'], test_size=0.2, random_st
4 )

```

## ▼ Unigram

```

1
2 # Classifiers to compare
3 classifiers = {
4     'Naive Bayes': MultinomialNB(),
5     'Logistic Regression': LogisticRegression(max_iter=1000)
6     'SVM': SVC(kernel='linear', probability=True),
7     'Random Forest': RandomForestClassifier(n_estimators=100
8 }
9 # N-gram combinations
10 ngram_configs = {
11     'Unigram (U)': (1, 1)
12

```

```
13 }
14
15 # Store results
16 results = []
17
18 # Process each classifier and n-gram
19 for clf_name, clf in classifiers.items():
20     for ngram_name, ngram_range in ngram_configs.items():
21         # Vectorize text
22         vectorizer = CountVectorizer(ngram_range=ngram_range)
23         X_train = vectorizer.fit_transform(train_texts)
24         clf.fit(X_train, train_labels)
25
26         # Predictions
27         X_test = vectorizer.transform(test_texts)
28         y_pred = clf.predict(X_test)
29
30         # Metrics
31         accuracy = accuracy_score(test_labels, y_pred)
32         precision = precision_score(test_labels, y_pred)
33         recall = recall_score(test_labels, y_pred)
34         f1 = f1_score(test_labels, y_pred)
35
36         # Store results
37         results.append({
38             'Classifier': clf_name,
39             'N-Gram': ngram_name,
40             'Accuracy': accuracy,
41             'Precision': precision,
42             'Recall': recall,
43             'F1': f1
44         })
45
46 # Convert to DataFrame
47 results_df = pd.DataFrame(results)
48
49 # Print top models
50 print("\n==== Top 4 Models of Unigram ===")
51 print(results_df.sort_values(by='F1', ascending=False).head(
```

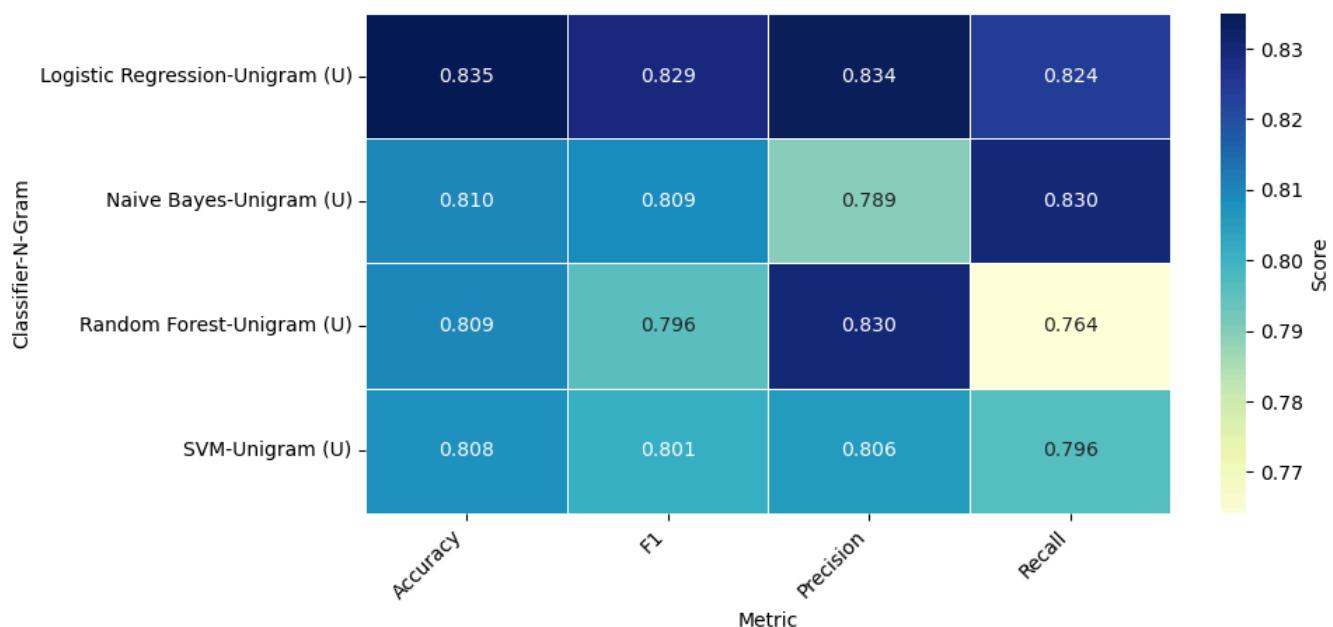
```
52
53 # Melt for visualization
54 melted_df = pd.melt(results_df,
55                      id_vars=['Classifier', 'N-Gram'],
56                      value_vars=[ 'Accuracy', 'Precision', 'Re
57                      var_name='Metric',
58                      value_name='Score')
59
60 # Create heatmap
61 plt.figure(figsize=(10, 5))
62 heatmap_data = melted_df.pivot_table(
63     index=['Classifier', 'N-Gram'],
64     columns='Metric',
65     values='Score'
66 )
67
68 sns.heatmap(
69     heatmap_data,
70     annot=True,
71     fmt=".3f",
72     cmap="YlGnBu",
73     linewidths=0.5,
74     cbar_kws={'label': 'Score'}
75 )
76
77 plt.title('Performance Metrics by Classifier and N-Gram Comb
78 plt.xticks(rotation=45, ha='right')
79 plt.yticks(rotation=0)
80 plt.tight_layout()
81 plt.show()
82
83 # Optional: Save results to CSV
84 results_df.to_csv('spam_detection_metrics.csv', index=False)
```



== Top 4 Models of Unigram ==

Classifier	N-Gram	Accuracy	Precision	Recall	F1
Logistic Regression	Unigram (U)	0.835	0.834202	0.823893	0.829016
Naive Bayes	Unigram (U)	0.81	0.789422	0.830072	0.809237
SVM	Unigram (U)	0.808	0.806048	0.796087	0.801036
Random Forest	Unigram (U)	0.8095	0.829978	0.764161	0.79571

Performance Metrics by Classifier and N-Gram Combination



## ▼ Bigram

```
1
2 # Classifiers to compare
3 classifiers = {
4     'Naive Bayes': MultinomialNB(),
5     'Logistic Regression': LogisticRegression(max_iter=1000)
6     'SVM': SVC(kernel='linear', probability=True),
7     'Random Forest': RandomForestClassifier(n_estimators=100
8 }
9 # N-gram combinations
10 ngram_configs = {
11     'Bigram (B)': (2, 2)
```

```
12
13 }
14
15 # Store results
16 results = []
17
18 # Process each classifier and n-gram
19 for clf_name, clf in classifiers.items():
20     for ngram_name, ngram_range in ngram_configs.items():
21         # Vectorize text
22         vectorizer = CountVectorizer(ngram_range=ngram_range)
23         X_train = vectorizer.fit_transform(train_texts)
24         clf.fit(X_train, train_labels)
25
26         # Predictions
27         X_test = vectorizer.transform(test_texts)
28         y_pred = clf.predict(X_test)
29
30         # Metrics
31         accuracy = accuracy_score(test_labels, y_pred)
32         precision = precision_score(test_labels, y_pred)
33         recall = recall_score(test_labels, y_pred)
34         f1 = f1_score(test_labels, y_pred)
35
36         # Store results
37         results.append({
38             'Classifier': clf_name,
39             'N-Gram': ngram_name,
40             'Accuracy': accuracy,
41             'Precision': precision,
42             'Recall': recall,
43             'F1': f1
44         })
45
46 # Convert to DataFrame
47 results_df = pd.DataFrame(results)
48
49 # Print top models
50 print("\n==== Top 4 Models of Bigram ===")
```

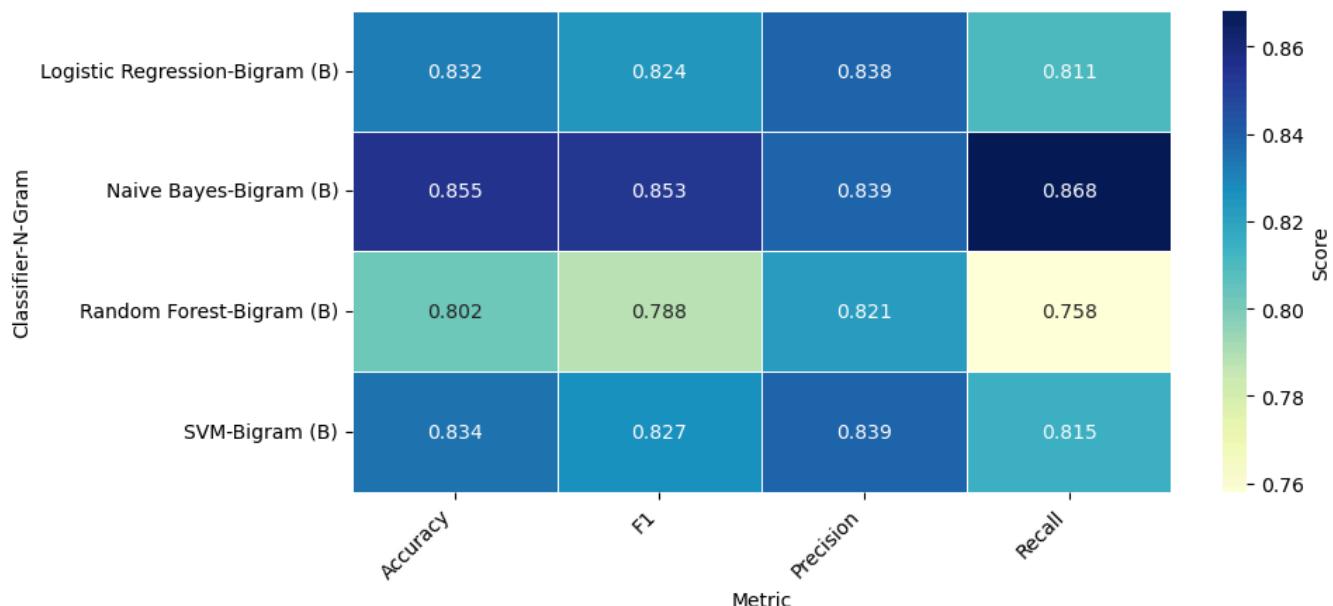
```
51 print(results_df.sort_values(by='F1', ascending=False).head()
52
53 # Melt for visualization
54 melted_df = pd.melt(results_df,
55                      id_vars=['Classifier', 'N-Gram'],
56                      value_vars=['Accuracy', 'Precision', 'Re
57                      var_name='Metric',
58                      value_name='Score')
59
60 # Create heatmap
61 plt.figure(figsize=(10, 5))
62 heatmap_data = melted_df.pivot_table(
63     index=['Classifier', 'N-Gram'],
64     columns='Metric',
65     values='Score'
66 )
67
68 sns.heatmap(
69     heatmap_data,
70     annot=True,
71     fmt=".3f",
72     cmap="YlGnBu",
73     linewidths=0.5,
74     cbar_kws={'label': 'Score'}
75 )
76
77 plt.title('Performance Metrics by Classifier and N-Gram Comb
78 plt.xticks(rotation=45, ha='right')
79 plt.yticks(rotation=0)
80 plt.tight_layout()
81 plt.show()
82
83 # Optional: Save results to CSV
84 results_df.to_csv('spam_detection_metrics.csv', index=False)
```



== Top 4 Models of Bigram ==

Classifier	N-Gram	Accuracy	Precision	Recall	F1
Naive Bayes	Bigram (B)	0.855	0.838806	0.868177	0.853239
SVM	Bigram (B)	0.834	0.838812	0.814624	0.826541
Logistic Regression	Bigram (B)	0.832	0.838126	0.810505	0.824084
Random Forest	Bigram (B)	0.8025	0.821429	0.757981	0.788431

Performance Metrics by Classifier and N-Gram Combination



## ▼ Trigram

```
1
2 # Classifiers to compare
3 classifiers = {
4     'Naive Bayes': MultinomialNB(),
5     'Logistic Regression': LogisticRegression(max_iter=1000)
6     'SVM': SVC(kernel='linear', probability=True),
7     'Random Forest': RandomForestClassifier(n_estimators=100
8 }
9 # N-gram combinations
10 ngram_configs = {
11     'Trigram (T)': (3, 3)
```

```
12
13 }
14
15 # Store results
16 results = []
17
18 # Process each classifier and n-gram
19 for clf_name, clf in classifiers.items():
20     for ngram_name, ngram_range in ngram_configs.items():
21         # Vectorize text
22         vectorizer = CountVectorizer(ngram_range=ngram_range)
23         X_train = vectorizer.fit_transform(train_texts)
24         clf.fit(X_train, train_labels)
25
26         # Predictions
27         X_test = vectorizer.transform(test_texts)
28         y_pred = clf.predict(X_test)
29
30         # Metrics
31         accuracy = accuracy_score(test_labels, y_pred)
32         precision = precision_score(test_labels, y_pred)
33         recall = recall_score(test_labels, y_pred)
34         f1 = f1_score(test_labels, y_pred)
35
36         # Store results
37         results.append({
38             'Classifier': clf_name,
39             'N-Gram': ngram_name,
40             'Accuracy': accuracy,
41             'Precision': precision,
42             'Recall': recall,
43             'F1': f1
44         })
45
46 # Convert to DataFrame
47 results_df = pd.DataFrame(results)
48
49 # Print top models
50 print("\n==== Top 4 Models of Trigram ===")
```

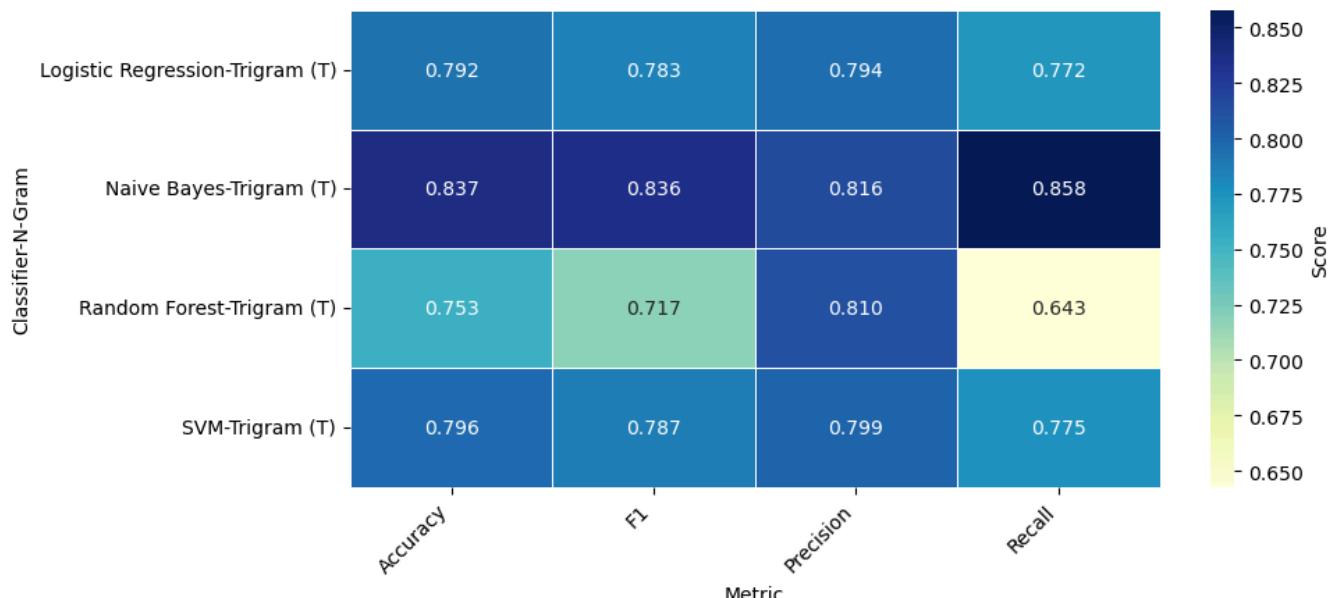
```
51 print(results_df.sort_values(by='F1', ascending=False).head()
52
53 # Melt for visualization
54 melted_df = pd.melt(results_df,
55                      id_vars=['Classifier', 'N-Gram'],
56                      value_vars=['Accuracy', 'Precision', 'Re
57                      var_name='Metric',
58                      value_name='Score')
59
60 # Create heatmap
61 plt.figure(figsize=(10, 5))
62 heatmap_data = melted_df.pivot_table(
63     index=['Classifier', 'N-Gram'],
64     columns='Metric',
65     values='Score'
66 )
67
68 sns.heatmap(
69     heatmap_data,
70     annot=True,
71     fmt=".3f",
72     cmap="YlGnBu",
73     linewidths=0.5,
74     cbar_kws={'label': 'Score'}
75 )
76
77 plt.title('Performance Metrics by Classifier and N-Gram Comb
78 plt.xticks(rotation=45, ha='right')
79 plt.yticks(rotation=0)
80 plt.tight_layout()
81 plt.show()
82
83 # Optional: Save results to CSV
84 results_df.to_csv('spam_detection_metrics.csv', index=False)
```



== Top 4 Models of Trigram ==

Classifier	N-Gram	Accuracy	Precision	Recall	F1
Naive Bayes	Trigram (T)	0.837	0.815867	0.857878	0.836345
SVM	Trigram (T)	0.796	0.798515	0.775489	0.786834
Logistic Regression	Trigram (T)	0.7925	0.794492	0.7724	0.78329
Random Forest	Trigram (T)	0.7535	0.81039	0.642636	0.716829

Performance Metrics by Classifier and N-Gram Combination



## ▼ Unigram + Bigram

```
1
2 # Classifiers to compare
3 classifiers = {
4     'Naive Bayes': MultinomialNB(),
5     'Logistic Regression': LogisticRegression(max_iter=1000)
6     'SVM': SVC(kernel='linear', probability=True),
7     'Random Forest': RandomForestClassifier(n_estimators=100
8 }
9 # N-gram combinations
10 ngram_configs = {
11     'U+B': (1, 2)
```

```
12
13 }
14
15 # Store results
16 results = []
17
18 # Process each classifier and n-gram
19 for clf_name, clf in classifiers.items():
20     for ngram_name, ngram_range in ngram_configs.items():
21         # Vectorize text
22         vectorizer = CountVectorizer(ngram_range=ngram_range)
23         X_train = vectorizer.fit_transform(train_texts)
24         clf.fit(X_train, train_labels)
25
26         # Predictions
27         X_test = vectorizer.transform(test_texts)
28         y_pred = clf.predict(X_test)
29
30         # Metrics
31         accuracy = accuracy_score(test_labels, y_pred)
32         precision = precision_score(test_labels, y_pred)
33         recall = recall_score(test_labels, y_pred)
34         f1 = f1_score(test_labels, y_pred)
35
36         # Store results
37         results.append({
38             'Classifier': clf_name,
39             'N-Gram': ngram_name,
40             'Accuracy': accuracy,
41             'Precision': precision,
42             'Recall': recall,
43             'F1': f1
44         })
45
46 # Convert to DataFrame
47 results_df = pd.DataFrame(results)
48
49 # Print top models
50 print("\n==== Top 4 Models of Unigram + Bigram ===")
```

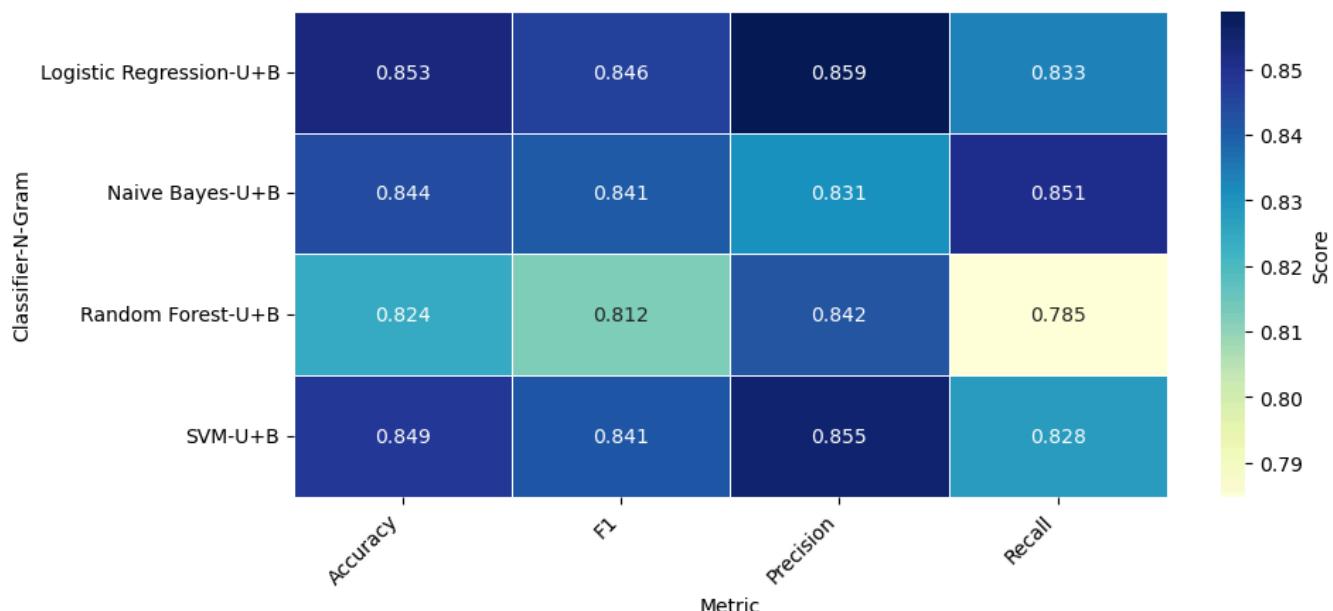
```
51 print(results_df.sort_values(by='F1', ascending=False).head()
52
53 # Melt for visualization
54 melted_df = pd.melt(results_df,
55                      id_vars=['Classifier', 'N-Gram'],
56                      value_vars=['Accuracy', 'Precision', 'Re
57                      var_name='Metric',
58                      value_name='Score')
59
60 # Create heatmap
61 plt.figure(figsize=(10, 5))
62 heatmap_data = melted_df.pivot_table(
63     index=['Classifier', 'N-Gram'],
64     columns='Metric',
65     values='Score'
66 )
67
68 sns.heatmap(
69     heatmap_data,
70     annot=True,
71     fmt=".3f",
72     cmap="YlGnBu",
73     linewidths=0.5,
74     cbar_kws={'label': 'Score'}
75 )
76
77 plt.title('Performance Metrics by Classifier and N-Gram Comb
78 plt.xticks(rotation=45, ha='right')
79 plt.yticks(rotation=0)
80 plt.tight_layout()
81 plt.show()
82
83 # Optional: Save results to CSV
84 results_df.to_csv('spam_detection_metrics.csv', index=False)
```



== Top 4 Models of Unigram + Bigram ==

Classifier	N-Gram	Accuracy	Precision	Recall	F1
Logistic Regression	U+B	0.8525	0.858811	0.833162	0.845792
SVM	U+B	0.8485	0.855319	0.828012	0.841444
Naive Bayes	U+B	0.8435	0.830986	0.850669	0.840712
Random Forest	U+B	0.824	0.841989	0.784758	0.812367

Performance Metrics by Classifier and N-Gram Combination



## ▼ Unigram + Trigram

```
1
2 # Classifiers to compare
3 classifiers = {
4     'Naive Bayes': MultinomialNB(),
5     'Logistic Regression': LogisticRegression(max_iter=1000)
6     'SVM': SVC(kernel='linear', probability=True),
7     'Random Forest': RandomForestClassifier(n_estimators=100
8 }
9 # N-gram combinations
10 ngram_configs = {
11     'U+T': (1, 3)
```

```
12
13 }
14
15 # Store results
16 results = []
17
18 # Process each classifier and n-gram
19 for clf_name, clf in classifiers.items():
20     for ngram_name, ngram_range in ngram_configs.items():
21         # Vectorize text
22         vectorizer = CountVectorizer(ngram_range=ngram_range)
23         X_train = vectorizer.fit_transform(train_texts)
24         clf.fit(X_train, train_labels)
25
26         # Predictions
27         X_test = vectorizer.transform(test_texts)
28         y_pred = clf.predict(X_test)
29
30         # Metrics
31         accuracy = accuracy_score(test_labels, y_pred)
32         precision = precision_score(test_labels, y_pred)
33         recall = recall_score(test_labels, y_pred)
34         f1 = f1_score(test_labels, y_pred)
35
36         # Store results
37         results.append({
38             'Classifier': clf_name,
39             'N-Gram': ngram_name,
40             'Accuracy': accuracy,
41             'Precision': precision,
42             'Recall': recall,
43             'F1': f1
44         })
45
46 # Convert to DataFrame
47 results_df = pd.DataFrame(results)
48
49 # Print top models
50 print("\n==== Top 4 Models of Unigram + Trigram ===")
```

```
51 print(results_df.sort_values(by='F1', ascending=False).head()
52
53 # Melt for visualization
54 melted_df = pd.melt(results_df,
55                      id_vars=['Classifier', 'N-Gram'],
56                      value_vars=['Accuracy', 'Precision', 'Re
57                      var_name='Metric',
58                      value_name='Score')
59
60 # Create heatmap
61 plt.figure(figsize=(10, 5))
62 heatmap_data = melted_df.pivot_table(
63     index=['Classifier', 'N-Gram'],
64     columns='Metric',
65     values='Score'
66 )
67
68 sns.heatmap(
69     heatmap_data,
70     annot=True,
71     fmt=".3f",
72     cmap="YlGnBu",
73     linewidths=0.5,
74     cbar_kws={'label': 'Score'}
75 )
76
77 plt.title('Performance Metrics by Classifier and N-Gram Comb
78 plt.xticks(rotation=45, ha='right')
79 plt.yticks(rotation=0)
80 plt.tight_layout()
81 plt.show()
82
83 # Optional: Save results to CSV
84 results_df.to_csv('spam_detection_metrics.csv', index=False)
```

## ▼ Bigram + Trigram

```
1
2 # Classifiers to compare
3 classifiers = {
4     'Naive Bayes': MultinomialNB(),
5     'Logistic Regression': LogisticRegression(max_iter=1000)
6     'SVM': SVC(kernel='linear', probability=True),
7     'Random Forest': RandomForestClassifier(n_estimators=100
8 }
9 # N-gram combinations
10 ngram_configs = {
11     'B+T': (2, 3)
12
13 }
14
15 # Store results
16 results = []
17
18 # Process each classifier and n-gram
19 for clf_name, clf in classifiers.items():
20     for ngram_name, ngram_range in ngram_configs.items():
21         # Vectorize text
22         vectorizer = CountVectorizer(ngram_range=ngram_range
23         X_train = vectorizer.fit_transform(train_texts)
24         clf.fit(X_train, train_labels)
25
26         # Predictions
27         X_test = vectorizer.transform(test_texts)
28         y_pred = clf.predict(X_test)
29
30         # Metrics
31         accuracy = accuracy_score(test_labels, y_pred)
32         precision = precision_score(test_labels, y_pred)
33         recall = recall_score(test_labels, y_pred)
34         f1 = f1_score(test_labels, y_pred)
35
36         # Store results
37         results.append({
38             'Classifier': clf_name,
39             'N-Gram': ngram_name,
```

```
40         'Accuracy': accuracy,
41         'Precision': precision,
42         'Recall': recall,
43         'F1': f1
44     })
45
46 # Convert to DataFrame
47 results_df = pd.DataFrame(results)
48
49 # Print top models
50 print("\n==== Top 4 Models of Bigram + Trigram ===")
51 print(results_df.sort_values(by='F1', ascending=False).head(
52
53 # Melt for visualization
54 melted_df = pd.melt(results_df,
55                     id_vars=['Classifier', 'N-Gram'],
56                     value_vars=['Accuracy', 'Precision', 'Re
57                     var_name='Metric',
58                     value_name='Score')
59
60 # Create heatmap
61 plt.figure(figsize=(10, 5))
62 heatmap_data = melted_df.pivot_table(
63     index=['Classifier', 'N-Gram'],
64     columns='Metric',
65     values='Score'
66 )
67
68 sns.heatmap(
69     heatmap_data,
70     annot=True,
71     fmt=".3f",
72     cmap="YlGnBu",
73     linewidths=0.5,
74     cbar_kws={'label': 'Score'}
75 )
76
77 plt.title('Performance Metrics by Classifier and N-Gram Comb
78 plt.xticks(rotation=45, ha='right')
```

```
79 plt.yticks(rotation=0)
80 plt.tight_layout()
81 plt.show()
82
83 # Optional: Save results to CSV
84 results_df.to_csv('spam_detection_metrics.csv', index=False)
```

## ▼ Unigram + Bigram + Trigram

```
1
2 # Classifiers to compare
3 classifiers = {
4     'Naive Bayes': MultinomialNB(),
5     'Logistic Regression': LogisticRegression(max_iter=1000)
6     'SVM': SVC(kernel='linear', probability=True),
7     'Random Forest': RandomForestClassifier(n_estimators=100
8 }
9 # N-gram combinations
10 ngram_configs = {
11     'U+B+T': (1,3)
12
13 }
14
15 # Store results
16 results = []
17
18 # Process each classifier and n-gram
19 for clf_name, clf in classifiers.items():
20     for ngram_name, ngram_range in ngram_configs.items():
21         # Vectorize text
22         vectorizer = CountVectorizer(ngram_range=ngram_range
23         X_train = vectorizer.fit_transform(train_texts)
24         clf.fit(X_train, train_labels)
25
26         # Predictions
27         X_test = vectorizer.transform(test_texts)
28         y_pred = clf.predict(X_test)
```

```
29
30     # Metrics
31     accuracy = accuracy_score(test_labels, y_pred)
32     precision = precision_score(test_labels, y_pred)
33     recall = recall_score(test_labels, y_pred)
34     f1 = f1_score(test_labels, y_pred)
35
36     # Store results
37     results.append({
38         'Classifier': clf_name,
39         'N-Gram': ngram_name,
40         'Accuracy': accuracy,
41         'Precision': precision,
42         'Recall': recall,
43         'F1': f1
44     })
45
46 # Convert to DataFrame
47 results_df = pd.DataFrame(results)
48
49 # Print top models
50 print("\n==== Top 4 Models of Unigram + Bigram + Trigram ===")
51 print(results_df.sort_values(by='F1', ascending=False).head(
52
53 # Melt for visualization
54 melted_df = pd.melt(results_df,
55                         id_vars=['Classifier', 'N-Gram'],
56                         value_vars=['Accuracy', 'Precision', 'Re-
57                         var_name='Metric',
58                         value_name='Score')
59
60 # Create heatmap
61 plt.figure(figsize=(10, 5))
62 heatmap_data = melted_df.pivot_table(
63     index=['Classifier', 'N-Gram'],
64     columns='Metric',
65     values='Score'
66 )
67
```

```
68 sns.heatmap(  
69     heatmap_data,  
70     annot=True,  
71     fmt=".3f",  
72     cmap="YlGnBu",  
73     linewidths=0.5,  
74     cbar_kws={'label': 'Score'}  
75 )  
76  
77 plt.title('Performance Metrics by Classifier and N-Gram Comb')  
78 plt.xticks(rotation=45, ha='right')  
79 plt.yticks(rotation=0)  
80 plt.tight_layout()  
81 plt.show()  
82  
83 # Optional: Save results to CSV  
84 results_df.to_csv('spam_detection_metrics.csv', index=False)
```

1 Start coding or generate with AI.

All in One N\_Grams (Unigram , Bigram , Trigram , U+B , U+T , B+T , U+B+T )

```
1  
2  
3 # Load data  
4 df = pd.read_csv('spam.csv', encoding='latin-1')[['v1', 'v2']]  
5 df.columns = ['label', 'text']  
6 df['label'] = df['label'].map({'ham': 0, 'spam': 1})  
7  
8 # Split data  
9 train_texts, test_texts, train_labels, test_labels = train_  
10     df['text'], df['label'], test_size=0.2, random_state=42  
11 )  
12  
13 # Classifiers to compare
```

```
14 classifiers = {
15     'Naive Bayes': MultinomialNB(),
16     'Logistic Regression': LogisticRegression(max_iter=1000),
17     'SVM': SVC(kernel='linear', probability=True),
18     'Random Forest': RandomForestClassifier(n_estimators=10)
19 }
20
21 # N-gram combinations
22 ngram_configs = {
23     'Unigram (U)': (1, 1),
24     'Bigram (B)': (2, 2),
25     'Trigram (T)': (3, 3),
26     'U+B': (1, 2),
27     'U+T': (1, 3),
28     'B+T': (2, 3),
29     'U+B+T': (1, 3)
30 }
31
32 # Store results
33 results = []
34
35 # Process each classifier and n-gram
36 for clf_name, clf in classifiers.items():
37     for ngram_name, ngram_range in ngram_configs.items():
38         # Vectorize text
39         vectorizer = CountVectorizer(ngram_range=ngram_range)
40         X_train = vectorizer.fit_transform(train_texts)
41         clf.fit(X_train, train_labels)
42
43         # Predictions
44         X_test = vectorizer.transform(test_texts)
45         y_pred = clf.predict(X_test)
46
47         # Metrics
48         accuracy = accuracy_score(test_labels, y_pred)
49         precision = precision_score(test_labels, y_pred)
50         recall = recall_score(test_labels, y_pred)
51         f1 = f1_score(test_labels, y_pred)
52
```

```
53     # Store results
54     results.append({
55         'Classifier': clf_name,
56         'N-Gram': ngram_name,
57         'Accuracy': accuracy,
58         'Precision': precision,
59         'Recall': recall,
60         'F1': f1
61     })
62
63 # Convert to DataFrame
64 results_df = pd.DataFrame(results)
65
66 # Print top models
67 print("\n==== Top 5 Models ===")
68 print(results_df.sort_values(by='F1', ascending=False).head
69
70 # Melt for visualization
71 melted_df = pd.melt(results_df,
72                     id_vars=['Classifier', 'N-Gram'],
73                     value_vars=['Accuracy', 'Precision', 'R
74                     var_name='Metric',
75                     value_name='Score')
76
77 # Create heatmap
78 plt.figure(figsize=(10, 5))
79 heatmap_data = melted_df.pivot_table(
80     index=['Classifier', 'N-Gram'],
81     columns='Metric',
82     values='Score'
83 )
84
85 sns.heatmap(
86     heatmap_data,
87     annot=True,
88     fmt=".3f",
89     cmap="YlGnBu",
90     linewidths=0.5,
91     cbar_kws={'label': 'Score'}
```

```

92 )
93
94 plt.title('Performance Metrics by Classifier and N-Gram Com
95 plt.xticks(rotation=45, ha='right')
96 plt.yticks(rotation=0)
97 plt.tight_layout()
98 plt.show()
99
100 # Optional: Save results to CSV
101 results_df.to_csv('spam_detection_metrics.csv', index=False)

```

## ▼ 1. Performance Summary Statistics

```

1 # Calculate mean metrics across all n-grams for each classif
2 summary_stats = results_df.groupby('Classifier').agg({
3     'Accuracy': 'mean',
4     'Precision': 'mean',
5     'Recall': 'mean',
6     'F1': 'mean'
7 }).sort_values(by='F1', ascending=False)
8
9 print("\n==== Classifier Performance Summary ===")
10 print(summary_stats.to_markdown(floatfmt=".3f"))

```



==== Classifier Performance Summary ===				
Classifier	Accuracy	Precision	Recall	F1
Naive Bayes	0.984	0.971	0.907	0.938
Logistic Regression	0.974	1.000	0.807	0.893
SVM	0.974	1.000	0.807	0.893
Random Forest	0.966	1.000	0.747	0.855

1 Start coding or generate with AI.

## ▼ Best Model Identification

```
1 best_overall = summary_stats.idxmax()  
2 print("\n==== Best Classifier by Metric ===")  
3 print(best_overall.to_markdown())
```

→

```
==== Best Classifier by Metric ===  
|          | 0  
|-----|-----|  
| Accuracy | Naive Bayes  
| Precision | Logistic Regression  
| Recall   | Naive Bayes  
| F1       | Naive Bayes
```

1 Start coding or [generate](#) with AI.

1 Start coding or [generate](#) with AI.

## ✓ Notebook: 3) Python Dictionaries.ipynb

Python dictionaries are powerful data structures or collection that store data in the form of key-value pairs. A dictionary is a collection which is ordered\*, changeable and do not allow duplicates. by ordered we mean, Items have a defined order, and that order will not change. Dictionary items are presented in key:value pairs, and can be referred to by using the key name. Think of them like real-world dictionaries, where you look up words to find their meanings. In Python, you use keys to access values, making it efficient for data retrieval. Dictionaries are versatile and widely used for tasks like data mapping, configuration settings, and more.

What is a Dictionary? 01:- Rules in Dictionary 02: - Mutability and Immutability 03: - Creating a Dictionary 04: - Accessing Items from a Dictionary 05: - Editing a Dictionary 06: - Using the .get function 07: - Adding new Key-Value Pairs 08: - Deleting from a dictionary 09: - Operations in a dictionary

Rules for Dictionary: R1: Dictionary has no indexing R2: Dictionary is a mutable data type R3: Keys--> immutable, values--> they can be mutable (changed) R4: keys should be unique

Mutable data types: Lists, sets and dictionaries

Immutable Data Types in Python: These are data types whose values cannot be modified after they are created. In Python, some common immutable data types include Strings, Tuples, Integers, Complex Numbers, and Floats.

**Cannot Change Values of Immutable Objects:** Once an immutable object is created, its value cannot be altered. For example, if you create a string 'hello', you cannot directly modify it to become 'hello world' without creating a new string object.

**New Memory Location for Modified Object:** If you attempt to modify an immutable object, Python will create a new object with the modified value in a different memory location. The original object remains unchanged. This behavior ensures data integrity and consistency.

**Mutable Data Types:** Mutable data types are those whose values can be modified after they are created. This means you can change the content of the object without necessarily creating a new object or changing its memory location.

**Examples of Mutable Data Types:** Some common mutable data types in Python include lists, dictionaries, sets, and custom objects.

**Ability to Modify Values:** Unlike immutable data types, where values cannot be changed after creation, mutable data types allow modifications to be made directly to the object's content.

**In-Place Modifications:** With mutable data types, you can perform in-place modifications, meaning you can alter the existing object without creating a new one.

```
1 # Immutable data types
2 # Strings
3 string1 = "hello"
4 # Tuples
5 tuple1 = (1, 2, 3)
6 # Integers
7 int1 = 5
8 # Complex Numbers
9 complex1 = 2 + 3j
10 # Floats
11 float1 = 3.14
12
13 print("Original String:", string1, id(string1))
14 # Attempting to modify immutable objects
15 # Modifying a string
16 # This will result in a new string object being created
17 string1 = string1 + " world"
18 # Printing the original and modified objects
19 print("Modified String:", string1, id(string1))
20 #print("Modified String:", string2)
21
```

```
→ Original String: hello 1769230247792
Modified String: hello world 1769251288304
```

```
1 print("Original integer:", int1, id(int1))
2 int1=int1+5
3 print("Modified integer:", int1, id(int1))
```

```
→ Original integer: 5 140729279484840
Modified integer: 10 140729279485000
```

```
1 # Modifying a tuple
2 # Tuples are immutable, so modification is not possible dire
3 # We need to create a new tuple with the modified values
4 tuple2 = tuple1 + (4,)
5
6 # Modifying an integer
7 # Integers are immutable, so modification is not possible di
8 # We need to create a new integer with the modified value
9 int2 = int1 + 1
10
11 # Modifying a complex number
12 # Complex numbers are immutable, so modification is not poss
13 # We need to create a new complex number with the modified v
14 complex2 = complex1 * 2
15
16 # Modifying a float
17 # Floats are immutable, so modification is not possible dire
18 # We need to create a new float with the modified value
19 float2 = float1 + 1.0
20
21 # Printing the original and modified objects
22
23
24 print("Original Tuple:", tuple1)
25 print("Modified Tuple:", tuple2)
26
27
28 print("Original Complex Number:", complex1)
29 print("Modified Complex Number:", complex2)
```

```
30
31 print("Original Float:", float1)
32 print("Modified Float:", float2)
33
```

```
→ Original Tuple: (1, 2, 3)
    Modified Tuple: (1, 2, 3, 4)
    Original Complex Number: (2+3j)
    Modified Complex Number: (4+6j)
    Original Float: 3.14
    Modified Float: 4.140000000000001
```

```
1 #creating an empty dictionary
2 d1={}
3 d1
```

```
→ {}
```

```
1 #creating a normal dictionary
2 d2={"name":"ALi", "Age":22, "Gender":"Male"}
3 d2
```

```
→ {'name': 'ALi', 'Age': 22, 'Gender': 'Male'}
```

```
1 # keys are immuatable
2 d3= {[1,2,3]: "Hassan"}
3 d3
```

```
→ -----
TypeError                                         Traceback (most recent call last)
Cell In[11], line 2
      1 # keys are immuatable
----> 2 d3= {[1,2,3]: "Hassan"}
      3 d3

TypeError: unhashable type: 'list'
```

```
1 # keys are immuatable
2 d3= {(1,2,3): "Hassan"}
3 d3
```

```
→ {(1, 2, 3): 'Hassan'}
```

```
1 #keys are unique, Dictionaries cannot have two items with the
2 d4= {"Name":"Ali", "Name":"Hassan"} # there will be single i
3 d4
```

```
→ {'Name': 'Hassan'}
```

```
1 #Duplicate values will overwrite existing values:
2 thisdict = {
3     "brand": "Toyota",
4     "model": "Vitz2004",
5     "year": 1964,
6     "year": 2020
7 }
8 print(thisdict)
```

```
→ {'brand': 'Toyota', 'model': 'Vitz2004', 'year': 2020}
```

```
1 #creating a 2d dictionary: dictionary within a dictionary
2 d5={"Name":"Basit", "University":"Abasyn", "Marks":{"DB":80,
3 d5
```

```
→ {'Name': 'Basit',
    'University': 'Abasyn',
    'Marks': {'DB': 80, 'OS': 90, 'DS': 90}}
```

```
1 d5[ 'Marks' ][ 'DB' ]
```

```
→ 80
```

```
1 d5[ 'Marks' ][ 'OS' ]
```

```
→ 90
```

```
1 d5[ 'Marks' ][ 'DB' ]
```

```
→ 80
```

## Python Dictionary fromkeys()

1. The `fromkeys()` method creates a dictionary from the given sequence of keys and values. The `fromkeys()` method can take two parameters:  
alphabets - are the keys that can be any iterables like string, set, list, etc.

numbers (Optional) - are the values that can be of any type or any iterables like string, set, list, etc. Note: The same value is assigned to all the keys of the dictionary.

### fromkeys() Return Value

2. The fromkeys() method returns: a new dictionary with the given sequence of keys and values

Note: If the value of the dictionary is not provided, None is assigned to the keys.

```
1 stds={10,8,7,6,5,2}  
2 stds
```

→ {2, 5, 6, 7, 8, 10}

```
1 # keys for the dictionary  
2 Stds = ['Ashiq', 'Zia', 'Asif']  
3  
4 # value for the dictionary  
5 Major = 'SE'  
6  
7 # creates a dictionary with keys and values  
8 dictionary = dict.fromkeys(Stds, Major)  
9  
10 print(dictionary)  
11  
12 # Output: {'a': 1, 'c': 1, 'b': 1}
```

→ {'Ashiq': 'SE', 'Zia': 'SE', 'Asif': 'SE'}

```
1 #All the keys of the dictionary are assigned with the same v  
2 #Python Dictionary fromkeys() with Key and Value  
3 # set of vowels  
4 keys = ('a', 'e', 'i', 'o', 'u' )  
5  
6 # assign string to the value  
7 value = 'vowel'  
8  
9 # creates a dictionary with keys and values  
10 vowels = dict.fromkeys(keys, value)  
11  
12 print(vowels)
```

```
→ { 'a': 'vowel', 'e': 'vowel', 'i': 'vowel', 'o': 'vowel', 'u': 'vowel'}
```

```
1 #Fromkeys() without Value
2 # list of numbers
3 keys = [1, 2, 4, 5, 6, 7, 8, 9 ]
4 value='integers'
5
6 # creates a dictionary with keys only
7 numbers = dict.fromkeys(keys,value)
8
9 print(numbers)
```

```
→ {1: 'integers', 2: 'integers', 4: 'integers', 5: 'integers', 6: 'integers', 7: 'integers'}
```



In the below example, we have used a list as the value for the dictionary. The iterables like list, dictionary, etc are the mutable objects, meaning they can be modified.

Here, when we update the list value using append, the keys are assigned with the new updated values. This is because each element is pointing to the same address in the memory. In Python, when you assign a mutable object (like a list) to multiple variables or use it as a value for multiple keys in a dictionary, all those variables or keys reference the same object in memory. They are essentially different labels or names pointing to the same object. So, when you modify the object through any of these references, the changes are reflected in all other references because they are all pointing to the same memory location.

```
1 #fromkeys() To Create A Dictionary From Mutable Object
2 # set of vowels
3 keys = {'a', 'e', 'i', 'o', 'u' }
4
5 # list of number
6 value = [1]
7
8 vowels = dict.fromkeys(keys, value) #{'a':[1], 'e':[1]}
9 print(vowels)
```

```
→ {'a': [1], 'o': [1], 'i': [1], 'u': [1], 'e': [1]}
```

```
1 # updates the list value
2 value.append(2)
```

```
3 value.append(3)
```

```
4
```

```
5 print(vowels)
```

```
→ { 'a': [1, 2, 3], 'o': [1, 2, 3], 'i': [1, 2, 3], 'u': [1, 2, 3], 'e': [1, 2, 3]}
```

```
1 #Accessing items from dictionary
```

```
1 d5[0] # No positive or negative indexing, nor slicing in dic
```

```
→ -----  
KeyError Traceback (most recent call last)  
Cell In[41], line 1  
----> 1 d5[0]
```

```
KeyError: 0
```

```
1 d5['Name'] #Provide a correct key for fetching a value from
```

```
→ 'Basit'
```

Return Value from get():

get() method returns:

the value for the specified key if key is in the dictionary. None if the key is not found and value is not specified. value if the key is not found and value is specified.

```
1 2/0
```

```
→ -----  
ZeroDivisionError Traceback (most recent call last)  
Cell In[46], line 1  
----> 1 2/0
```

```
ZeroDivisionError: division by zero
```

```
1 person = {'name': 'Uzair', 'age': 22}
2
3 print('Name: ', person.get('name'))
4
5 print('Age: ', person.get('age'))
6
```

```
→ Name: Uzair
    Age: 22
```

```
1 # value is not provided
2 print('Salary: ', person.get('salary'))
```

```
→ Salary: None
```

```
1 print('Salary: ', person['salary'])
```

```
→ -----
KeyError                                                 Traceback (most recent call last)
Cell In[54], line 1
----> 1 print('Salary: ', person['salary'])

KeyError: 'salary'
```

```
1 # value is provided
2 print('Salary: ', person.get('salary', 0.0))
```

```
→ Salary: 0.0
```

```
1 d5
```

```
→ {'Name': 'Basit',
    'University': 'Abasyn',
    'Marks': {'DB': 80, 'OS': 90, 'DS': 90}}
```

```
1 d5.get('Name') # can access value by providing a key to the dict
```

```
→ 'Basit'
```

```
1 d5.get('Marks').get('OS')
```

```
→ 90
```

Python get() method Vs dict[key] to Access Elements  
get() method returns a default value if the key is missing.

However, if the key is not found when you use dict[key], KeyError exception is raised.

```
1 person = {}  
2  
3 # Using get() results in None  
4 print('Salary: ', person.get('salary'))  
5 # Using [] results in KeyError  
6  
7
```

```
→ Salary: None
```

```
1 print(person['salary'])
```

```
→ -----  
KeyError Traceback (most recent call last)  
Cell In[66], line 1  
----> 1 print(person['salary'])  
  
KeyError: 'salary'
```

```
1 d5['Marks']
```

```
→ {'DB': 80, 'OS': 90, 'DS': 90}
```

```
1 d5['Marks']['OS']
```

```
→ 90
```

```
1 d5
```

```
→ {'Name': 'Basit',  
'University': 'Abasyn',  
'Marks': {'DB': 80, 'OS': 90, 'DS': 90}}
```

## Change Dictionary Items

Python dictionaries are mutable (changeable).

We can change the value of a dictionary element by referring to its key.

We can also use the update() method to add or change dictionary items.

```
1 #Edit particular item of dictionary
2 #Dictionaries are changeable, meaning that we can change, add
3 d5["Name"]="Basit"
4 d5
```

```
→ {'Name': 'Basit',
    'University': 'Abasyn',
    'Marks': {'DB': 80, 'OS': 90, 'DS': 90}}
```

```
1 d5[ 'Marks' ][ 'DB' ]=90
2 d5
```

```
→ {'Name': 'Basit',
    'University': 'Abasyn',
    'Marks': {'DB': 90, 'OS': 90, 'DS': 90}}
```

```
1 d5[ 'Marks' ][ 'OS' ]=99
2 d5
```

```
→ {'Name': 'Basit',
    'University': 'Abasyn',
    'Marks': {'DB': 90, 'OS': 99, 'DS': 90}}
```

```
1 d5[ 'address' ]='AUP'
2 d5
```

```
→ {'Name': 'Basit',
    'University': 'Abasyn',
    'Marks': {'DB': 90, 'OS': 99, 'DS': 90},
    'address': 'AUP'}
```

```
1 # Add new key-value pairs
2 d5[ 'Age' ]=25
3 d5
```

```
→ {'Name': 'Basit',
    'University': 'Abasyn',
    'Marks': {'DB': 90, 'OS': 99, 'DS': 90},
    'address': 'AUP',
    'Age': 25}
```

```
1 country_capitals = {
2     "Germany": "Berlin",
```

```
3 "Italy": "Naples",
4 "England": "London"
5 }
6
7 # change the value of "Italy" key to "Rome"
8 country_capitals["Italy"] = "Rome"
9
10 print(country_capitals)
```

```
→ {'Germany': 'Berlin', 'Italy': 'Rome', 'England': 'London'}
```

### Add Items to a Dictionary

We can add an item to a dictionary by assigning a value to a new key. For example,

```
1 country_capitals = {
2   "Germany": "Berlin",
3   "Canada": "Ottawa",
4 }
5
6
7 country_capitals["Pakistan"] = "Islamabad"
8
9 print(country_capitals)
```

```
→ {'Germany': 'Berlin', 'Canada': 'Ottawa', 'Pakistan': 'Islamabad'}
```

you can change the value of a specific item by referring to its key name:

d5['Marks']['AT']=70 where AT is a new key and 70 is new value

```
1 # Delete items from dictionary
2 del d2 # will delete the entire dictionary
```

```
1 d5
```

```
→ {'Name': 'Basit',
'University': 'Abasyn',
'Marks': {'DB': 90, 'OS': 99, 'DS': 90},
'address': 'AUP',
'Age': 25}
```

```
1 # Delete individual key-value pairs or delete the item having  
2 #del keyword gives a KeyError if the key is not present in the dictionary  
3 #We can also use the pop() method to remove an item from a dictionary  
4  
5 del d5['address']  
6 d5
```

```
→ { 'Name': 'Basit',  
    'University': 'Abasyn',  
    'Marks': {'DB': 90, 'OS': 99, 'DS': 90},  
    'Age': 25}
```

```
1 del d5['address']
```

```
→ -----  
KeyError Traceback (most recent call last)  
Cell In[96], line 1  
----> 1 del d5['address']  
  
KeyError: 'address'
```

```
1 d5.clear() # remove all items from dictionary  
2 d5
```

```
→ {}
```

```
1 d5
```

```
→ {}
```

```
1 #Just like sets, concatenation operation does not work on dictionaries  
2 d4+d5
```

```
→ -----  
TypeError Traceback (most recent call last)  
Cell In[102], line 2  
      1 #Just like sets, concatenation operation does not work on dictionaries  
----> 2 d4+d5  
  
TypeError: unsupported operand type(s) for +: 'dict' and 'dict'
```

```
1 t=(2,3,4)  
2 t*3
```

```
→ (2, 3, 4, 2, 3, 4, 2, 3, 4)
```

```
1 d4*3 # Multipication does not work on dictionaries  
2
```

```
→ -----  
TypeError Traceback (most recent call last)  
Cell In[106], line 1  
----> 1 d4*3  
  
TypeError: unsupported operand type(s) for *: 'dict' and 'int'
```

## Iterate Through a Dictionary

A dictionary is an ordered collection of items (starting from Python 3.7), therefore it maintains the order of its items.

We can iterate through dictionary keys one by one using a for loop.

```
1 country_capitals = {  
2     "Pakistan": "Islamabad",  
3     "Italy": "Rome"  
4 }  
5  
6 # print dictionary keys one by one  
7 for country in country_capitals:  
8     print(country, country_capitals[country])  
9 print()  
10  
11
```

```
→ Pakistan Islamabad  
    Italy Rome
```

```
1 # print dictionary values one by one  
2 for country in country_capitals:  
3     capital = country_capitals[country]  
4     print(capital)
```

```
→ Islamabad  
    Rome
```

```
1 d5={'Name':'Basit', 'Age':22, 'Univ':'UET'}
2 d5
```

```
→ {'Name': 'Basit', 'Age': 22, 'Univ': 'UET'}
```

```
1 for i in d5: # here loop over dictionary will give only key
2     print(i)
```

```
→ Name
  Age
  Univ
```

```
1 Start coding or generate with AI.
```

```
1 for i in d5: # here loop over dictionary will give only key
2     print(i, d5[i])
```

```
→ Name Basit
  Age 22
  Univ UET
```

```
1 # Membership operator work with dictionary and it checks the
2 'Univ' in d5
```

```
→ True
```

```
1 "Name" in d5
```

```
→ True
```

```
1 # Return the number of items or key-value pairs in dictionary
2 len(d5)
```

```
→ 3
```

```
1 min(d5) # return the min value based on lexicographic order,
```

```
→ 'Age'
```

```
1 max(d5)
```

```
→ 'Univ'
```

```
1 # sum() does not work as the key are strings but it worked i
```

```
1 sorted(d5)
```

```
→ ['Age', 'Name', 'Univ']
```

```
1 sorted(d5, reverse=True)
```

```
→ ['Univ', 'Name', 'Age']
```

▼ The keys() method returns:

a view object that displays the list of all the keys

For example, if the method returns dict\_keys([1, 2, 3]),

dict\_keys() is the view object [1, 2, 3] is the list of keys

```
1
```

```
2 d5.keys() # return list of keys
```

```
→ dict_keys(['Name', 'Age', 'Univ'])
```

```
1 employee = {'name': 'Esa', 'age': 22, 'salary': 3500.0}
```

```
2
```

```
3 # extracts the keys of the dictionary
```

```
4 dictionaryKeys = employee.keys()
```

```
5
```

```
6 print(dictionaryKeys)
```

```
→ dict_keys(['name', 'age', 'salary'])
```

```
1 #Update in dictionary updates the view object
```

```
2 employee = {'name': 'usman', 'age': 22, 'salary': 3500.0}
```

```
3
```

```
4 # extracts the dictionary keys
```

```
5 dictionaryKeys = employee.keys()
```

```
6 dictionaryvalues=employee.values()
```

```
7  
8 print('Before dictionary update:', dictionaryKeys)  
9 print('Before dictionary update:', dictionaryvalues)  
10  
11 # adds an element to the dictionary  
12 employee.update({'salary': 4500.0})  
13 employee.update({'address': 'UET'})  
14  
15 # prints the updated view object  
16 print('After dictionary update:', dictionaryKeys)  
17 print('After dictionary update:', dictionaryvalues)
```

```
→ Before dictionary update: dict_keys(['name', 'age', 'salary'])  
Before dictionary update: dict_values(['usman', 22, 3500.0])  
After dictionary update: dict_keys(['name', 'age', 'salary', 'address'])  
After dictionary update: dict_values(['usman', 22, 4500.0, 'UET'])
```

```
1 d5.values() # return list of values
```

```
→ dict_values(['Basit', 22, 'UET'])
```

Dictionary Items - Data Types The values in dictionary items can be of any data type:String, int, boolean, and list data types:

```
1 thisdict = {  
2     "brand": "Ford",  
3     "electric": False,  
4     "year": 1964,  
5     "colors": ["red", "white", "blue"]  
6 }  
7 thisdict
```

```
→ {'brand': 'Ford',  
  'electric': False,  
  'year': 1964,  
  'colors': ['red', 'white', 'blue']}
```

From Python's perspective, dictionaries are defined as objects with the data type 'dict': <class 'dict'>  
The dict() Constructor It is also possible to use the dict() constructor to make a dictionary.

```
1 thisdict = dict()  
2 print(thisdict)
```

→ {}

```
1 thisdict = dict(name = "Ali")  
2 print(thisdict)  
3
```

→ {'name': 'Ali'}

```
1 thisdict = dict(name = "Ali", age = 36, country = "Turkey")  
2 print(thisdict)  
3 print(type(thisdict))
```

→ {'name': 'Ali', 'age': 36, 'country': 'Turkey'}  
<class 'dict'>

```
1 #Make a change in the original dictionary, and see that the  
2 car = {  
3     "brand": "Ford",  
4     "model": "Mustang",  
5     "year": 1964  
6 }  
7  
8 x = car.values()  
9 print('before changes')  
10 print(x) #before the change  
11  
12 print('After changes')  
13 car["year"] = 2025  
14  
15 print(x) #after the change
```

→ before changes  
dict\_values(['Ford', 'Mustang', 1964])  
After changes  
dict\_values(['Ford', 'Mustang', 2025])

```
1 #Add a new item to the original dictionary, and see that the  
2 car = {
```

```
3 "brand": "Ford",
4 "model": "Mustang",
5 "year": 1964
6 }
7
8 x = car.values()
9
10 print(x) #before the change
11
12 car["color"] = "red"
13
14 print(x) #after the change
```

```
→ dict_values(['Ford', 'Mustang', 1964])
dict_values(['Ford', 'Mustang', 1964, 'red'])
```

```
1 #Get Items
2 #The items() method will return each item in a dictionary, a
3 #Get a list of the key:value pairs
4
5 item=car.items()
6 print(item)
```

```
→ dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964), ('color', 'red')])
```

```
1 car
```

```
→ {'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

```
1 ds={'name':"Azan", 'name':'DS'}
2 del ds['name']
3 ds
```

```
→ {}
```

```
1 car.items()
```

```
→ dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964), ('color', 'red')])
```

```
1 a,b= ('brand', 'Ford')
2 print(a,b)
```

→ brand Ford

```
1 for key, value in car.items():
2     print(key, value)
```

→ brand Ford
model Mustang
year 1964
color red

```
1 for key in car.keys():
2     print(key)
```

→ brand
model
year
color

```
1 for v in car.values():
2     print(v)
```

→ Ford
Mustang
1964
red

```
1 a,b,c=(4,5,6)
2 print(a,b,c)
```

→ 4 5 6

```
1 for k in car.items():
2     print(f'{k} ')
```

→ ('brand', 'Ford')
('model', 'Mustang')
('year', 1964)
('color', 'red')

```
1 #Check if Key Exists
2 #To determine if a specified key is present in a dictionary
```

```
3 thisdict = {  
4     "brand": "Ford",  
5     "model": "Mustang",  
6     "year": 1964  
7 }  
8 if "model" in thisdict:  
9     print("Yes, 'model' is one of the keys in the thisdict dic
```

→ Yes, 'model' is one of the keys in the thisdict dictionary

-The update() method will update the dictionary with the items from the given argument. --The argument must be a dictionary, or an iterable object with key:value pairs. --The update() method updates the dictionary with the elements from another dictionary object or from an iterable object of key/value pairs (generally tuples). If update() is called without passing parameters, the dictionary remains unchanged.

In Python, an iterable object is any object that can be iterated over, meaning you can traverse through its elements one by one. Examples of iterable objects in Python include lists, tuples, sets, dictionaries, strings, and more.

```
1 marks = {'Physics':67, 'Maths':87}  
2 internal_marks = {'Practical':40, 'CS':90}  
3 marks.update(internal_marks)  
4  
5 print(marks)  
6
```

→ {'Physics': 67, 'Maths': 87, 'Practical': 40, 'CS': 90}

```
1 #The update() method adds element(s) to the dictionary if th
2 #If the key is in the dictionary, it updates the key with th
3 d = {1: "one", 2: "three"}
4 d1 = {2: "two"}
5
6 # updates the value of key 2
7 d.update(d1)
8
9 print(d)
10
11
```

```
→ {1: 'one', 2: 'two'}
```

```
1 d1 = {3: "three"}
2
3 # adds element with key 3
4 d.update(d1)
5
6 print(d)
```

```
→ {1: 'one', 2: 'two', 3: 'three'}
```

Here, we have passed a list of tuples `[('y', 3), ('z', 0)]` to the `update()` function. In this case, the first element of tuple is used as the key and the second element is used as the value.

```
1 #we have passed a list of tuples [('y', 3), ('z', 0)] or ite
2 #the first element of tuple is used as the key and the secon
3
4 dictionary = {'x': 2}
5
6 dictionary.update([('y', 3), ('z', 0)])
7
8 #print(dictionary)
9 dc=dictionary.copy()
10 dc
```

```
→ {'x': 2, 'y': 3, 'z': 0}
```

```
1 # random sales dictionary
2 sales = { 'apple': 2, 'orange': 3, 'grapes': 4 }
3
4 values = sales.values()
5
6 print('Original items:', values)
7
8 # delete an item from dictionary
9 del sales['apple']
10
11 print('Updated items:', values)
```

→ Original items: dict\_values([2, 3, 4])  
Updated items: dict\_values([3, 4])

Python Dictionary pop(): The pop() method removes and returns an element from a dictionary having the given key.

pop() Parameters

pop() method takes two parameters:

key - key which is to be searched for removal

default - value which is to be returned when the key is not in the dictionary

Return value from pop() The pop() method returns:

If key is found - removed/popped element from the dictionary

If key is not found - value specified as the second argument (default)

If key is not found and default argument is not specified - KeyError exception is raised

```
1 # create a dictionary
2 marks = { 'Physics': 67, 'Chemistry': 72, 'Math': 89 }
3
4 #del marks['Chemistry']
5 element = marks.pop('Chemistry')
6
7 print('Popped Marks:', element)
8
9 # Output: Popped Marks: 72
```

→ Popped Marks: 72

```
1 #Pop an element from the dictionary
2 # random sales dictionary
3 sales = { 'apple': 2, 'orange': 3, 'grapes': 4 }
4
5 element = sales.pop('apple')
6
7 print('The popped element is:', element)
8 print('The dictionary is:', sales)
```

→ The popped element is: 2  
The dictionary is: {'orange': 3, 'grapes': 4}

```
1 #Pop an element not present from the dictionary
2 # random sales dictionary
3 sales = { 'apple': 2, 'orange': 3, 'grapes': 4 }
4
5 element = sales.pop('guava')
6
```

→ -----  
**KeyError** Traceback (most recent call last)  
Cell In[55], line 5  
 1 #Pop an element not present from the dictionary  
 2 # random sales dictionary  
 3 sales = { 'apple': 2, 'orange': 3, 'grapes': 4 }  
----> 5 element = sales.pop('guava')  
  
**KeyError**: 'guava'

```
1 #Pop an element not present from the dictionary, provided a default value
2 # random sales dictionary
3 sales = { 'apple': 2, 'orange': 3, 'grapes': 4 }
4
5 element = sales.pop('Mango', 'not found')
6
7 print('The popped element is:', element)
8 print('The dictionary is:', sales)
```

→ The popped element is: not found  
The dictionary is: {'apple': 2, 'orange': 3, 'grapes': 4}

Python Dictionary popitem() The popitem() doesn't take any parameters.

The popitem() method removes and returns the (key, value) pair from the dictionary in the Last In, First Out (LIFO) order.

Returns the latest inserted element (key,value) pair from the dictionary.

Removes the returned element pair from the dictionary.

The popitem() method raises a KeyError error if the dictionary is empty.

```
1 person = {'name': 'farhan', 'age': 22, 'salary': 3500.0}
2
3 # ('salary', 3500.0) is inserted at the last, so it is remov
4 result = person.popitem()
5
6 print('Return Value = ', result)
7 print('person = ', person)
8
9 # inserting a new element pair
10 person['profession'] = 'Developer'
11
12 # now ('profession', 'Developer') is the latest element
13 result = person.popitem()
14
15 print('Return Value = ', result)
16 print('person = ', person)
```

→ Return Value = ('salary', 3500.0)  
person = {'name': 'farhan', 'age': 22}  
Return Value = ('profession', 'Developer')  
person = {'name': 'farhan', 'age': 22}

The fromkeys() method in Python's dictionary class is not suitable for creating a dictionary where values are different for different keys.

This method is designed to create a new dictionary with keys from a sequence and set all the values to a single specified value.

Therefore, it does not allow for specifying different values for different keys directly through the fromkeys() method.

However, you can achieve the desired result by using other approaches, such as dictionary comprehension or by specifying values individually. Here's how you can do it using dictionary comprehension:

```
1 keys = ['a', 'b', 'c', 'd', 'e']
2 values = [1, 2, 3, 4, 5]
3 for i in zip(keys,values):
4     print(i)
```

```
→ ('a', 1)
  ('b', 2)
  ('c', 3)
  ('d', 4)
  ('e', 5)
```

```
1 keys = ['a', 'b', 'c', 'd', 'e']
2 values = [1, 2, 3, 4, 5]
3
4 # Create a dictionary using dictionary comprehension
5 result = {key: value for key, value in zip(keys, values)}
6
7 print(result)
8
```

```
→ {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
```

you modify the list value by appending 2 to it. Since each key in the vowels dictionary is referencing the same list value, modifying value affects all the values associated with the keys in vowels. So, when you print vowels again, you'll see that all values have been updated to [1, 2].

```
1 # vowels keys
2 keys = {'a', 'e', 'i', 'o', 'u' }
3 value = [1]
4
5 # creates dictionary using dictionary comprehension
6 vowels = { key : value for key in keys }
7
8 print(vowels)
9
10 # updates the value list
11 value.append(2)
12
13 print(vowels)
```

```
→ { 'i': [1], 'u': [1], 'a': [1], 'o': [1], 'e': [1] }
   { 'i': [1, 2], 'u': [1, 2], 'a': [1, 2], 'o': [1, 2], 'e': [1, 2] }
```

We can use dictionary comprehension and prevent updating the dictionary when the mutable object (list, dictionary, etc) is updated. For example,

In this code, `list(value)` ensures that each key in `vowels` is associated with a separate list object containing the same elements as value.

Thus, modifying value after creating the dictionary doesn't affect the values associated with the keys in the dictionary. Each key still refers to its own distinct list object.

```
1 # vowels keys
2 keys = {'a', 'e', 'i', 'o', 'u' }
3 value = [1]
4
5 # creates dictionary using dictionary comprehension
6 vowels = { key : list(value) for key in keys }
7
8 print(vowels)
9
10 # updates the value list
11 value.append(2)
12
13 print(vowels)
```

```
→ { 'i': [1], 'u': [1], 'a': [1], 'o': [1], 'e': [1] }
   { 'i': [1], 'u': [1], 'a': [1], 'o': [1], 'e': [1] }
```

1 Start coding or generate with AI.

By using `lst = list(value)`, you're creating a new list object `lst` that is a copy of the list `value`. This ensures that `lst` and `value` are two separate list objects in memory, even if they contain the same elements initially.

```
1 value = [2] # Original list
2 lst = list(value) # Creating a new list 'lst' by passing 'v
3 print(lst) # Output: [2]
```

```
→ [2]
```

Dictionary comprehension is an elegant and concise way to create dictionaries.

```
1 square_dict = dict()
2 for num in range(1, 11):
3     square_dict[num] = num*num
4 print(square_dict)
5
```

```
→ {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

```
1 #Now, let's create the dictionary in the above program using
2 # dictionary comprehension example
3 square_dict = {num: num*num for num in range(1, 11)}
4 print(square_dict)
```

```
→ {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

In both programs, we have created a dictionary `square_dict` with number-square key/value pair.

However, using dictionary comprehension allowed us to create a dictionary in a single line.



1 Start coding or generate with AI.

```
1 #we can see that we retrieved the item prices in dollars and
2 #Using dictionary comprehension makes this task much simpler
3
4 old_price = {'milk': 1.02, 'coffee': 2.5, 'bread': 2.5}
5
6 dollar_to_pound = 0.76
7 print(old_price)
8 new_price = {item: value*dollar_to_pound for (item, value) in
9 print(new_price)
10
11
```

```
→ {'milk': 1.02, 'coffee': 2.5, 'bread': 2.5}
→ {'milk': 0.7752, 'coffee': 1.9, 'bread': 1.9}
```

Conditionals in Dictionary Comprehension We can further customize dictionary comprehension by adding conditions to it. Let's look at an example. If Conditional Dictionary Comprehension only the items with even value have been added, because of the if clause in the dictionary comprehension.

```
1 original_dict = {'Farhan': 3.5, 'fazal': 4.8, 'Arsalan': 5.7
2
3 even_dict = {k: v for k, v in original_dict.items() if v > 4
4 print(even_dict)
5
```

```
→ {'fazal': 4.8, 'Arsalan': 5.7}
```

```
1 # add items in dictionary whose value is greater than 45
2 original_dict = {'jack': 38, 'michael': 48, 'guido': 57, 'jo
3
4 even_dict = {k: v for (k, v) in original_dict.items() if v >
5 print(even_dict)
```

```
→ {'michael': 48, 'guido': 57}
```

```
1 #Multiple if Conditional Dictionary Comprehension
2 #In this case, only the items with an odd value of less than
3 #It is because of the multiple if clauses in the dictionary
4 #They are equivalent to and operation where both conditions
5
6 original_dict = {'jack': 38, 'michael': 48, 'guido': 57, 'jo
7
8 new_dict = {k: v for (k, v) in original_dict.items() if v %
9 print(new_dict)
10
```

```
→ {'john': 33}
```

```
1 #if-else Conditional Dictionary Comprehension
2 original_dict = {'jack': 38, 'michael': 48, 'guido': 57, 'jo
3
4 new_dict_1 = {k: ('old' if v > 50 else 'Adult' if v>35 else
5     for (k, v) in original_dict.items())
6 print(new_dict_1)
```

```
7
```

```
8
```

```
→ { 'jack': 'Adult', 'michael': 'Adult', 'guido': 'old', 'john': 'Young', 'Alim': 'Young'}
```

```
1 original_dict = { 'jack': 3.8, 'michael': 3.4, 'guido': 2.4,  
2  
3 new_dict_1 = {k: ('A+' if v > 3.5 else 'B+' if v > 3.0 else '  
4     for (k, v) in original_dict.items())  
5 print(new_dict_1)
```

```
→ { 'jack': 'A+', 'michael': 'B+', 'guido': 'Failed', 'john': 'Failed', 'Alim': 'C'}
```

## ▼ multiples of 2, 3, 4 and 5

```
1 dic= { k: { v: v*k for v in range(1,6)} for k in range(2,6)  
2 print(dic)  
3
```

```
→ {2: {1: 2, 2: 4, 3: 6, 4: 8, 5: 10}, 3: {1: 3, 2: 6, 3: 9, 4: 12, 5: 15}, 4: {1: 4, 2: 8, 3: 12, 4: 16, 5: 20}}
```



```
1 dictionary = {  
2     k1: {k2: k1 * k2 for k2 in range(1, 6)} for k1 in range(1, 6)  
3 }  
4 print(dictionary)  
5
```

```
→ {2: {1: 2, 2: 4, 3: 6, 4: 8, 5: 10}, 3: {1: 3, 2: 6, 3: 9, 4: 12, 5: 15}, 4: {1: 4, 2: 8, 3: 12, 4: 16, 5: 20}}
```



Iterable:

An iterable is any object in Python that can be looped over using a loop (like a for loop) or consumed with functions and constructors that accept iterable objects (such as sum(), list(), tuple(), etc.). Examples of iterables include lists, tuples, sets, dictionaries, strings, and more. Essentially, anything that you can iterate over in a loop or pass to a function expecting multiple values is an iterable. Iterables provide a way to access the individual elements one at a time.

Iterator:

An iterator is a special object in Python that represents a stream of data. It provides a way to access the elements of an iterable one at a time and keeps track of the current state of iteration. Every iterator is an iterable, but not every iterable is an iterator. Iterators are required to implement two methods: **iter()** and **next()**. Iterators are typically created from iterables using the **iter()** function. Examples of iterators include objects returned by the **zip()**, **enumerate()**, and **range()** functions.

```
1 # Using range() function to create an iterator
2 iterator = iter(range(5))
3
4 # Iterating over the iterator using a for loop
5 for num in iterator:
6     print(num)
7
```

```
→ 0
1
2
3
4
```

The **zip()** function in Python takes iterables (such as lists, tuples, or sets) as input and returns an iterator that aggregates elements from each of the iterables. It pairs elements from each iterable together, creating tuples where the first element of each tuple comes from the first iterable, the second element comes from the second iterable, and so on.

```
1 name = [ "Atif", "Jamal", "obaid", "farhan", "farhan" ]
2 GPA = [ 4.0, 1.3, 3.4, 2.3 ,2.3]
3
4 # using zip() to map values
5 # creates an iterator that yields tuples containing correspo
6 mapped = zip(name, GPA)
7
8 #set() converts this iterator into a set.
9 #Since sets only contain unique elements, if there are dupli
10 #they will be removed, leaving only unique pairs in the set.
11
12 print(set(mapped))
13
```

```
→ {('obaid', 3.4), ('farhan', 2.3), ('Jamal', 1.3), ('Atif', 4.0)}
```

We have two lists:

stocks contains the names of stocks. prices contains the corresponding prices of the stocks. The dictionary comprehension:

`zip(stocks, prices)` pairs elements from the stocks and prices lists together, creating tuples of (stock, price). The dictionary comprehension iterates over these pairs, using stocks as keys and prices as values. For each pair (stock, price) generated by `zip(stocks, prices)`, a key-value pair is added to the dictionary `new_dict`. The resulting dictionary `new_dict` contains:

Keys: Stock names from the stocks list. Values: Corresponding prices from the prices list.

```
1 #Python zip() with Dictionary
2 stocks = ['AB', 'CD', 'EF']
3 prices = [2175, 1127, 2750]
4
5 new_dict = {stocks: prices for stocks,prices in zip(stocks,
6 print(new_dict)
7
```

```
→ {'AB': 2175, 'CD': 1127, 'EF': 2750}
```

```
1 # List of numbers
2 numbers = [1, 2, 3, 4, 5]
3
4 # Dictionary comprehension to create a dictionary where keys
5 squared_dict = {num: num**2 for num in numbers}
6
7 print(squared_dict) # Output: {1: 1, 2: 4, 3: 9, 4: 16, 5:
8
```

```
→ {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

`zip(names, ages):`

The `zip()` function takes two or more iterables and returns an iterator that generates tuples by pairing elements from the input iterables. In this case, it pairs corresponding elements from the names and ages lists.

`enumerate():`

The enumerate() function takes an iterable as input and returns an iterator that yields pairs of (index, value), where index is the index of the value in the iterable and value is the actual value. In this code, enumerate() is used to loop over the pairs of (name, age) generated by zip(names, ages) while also obtaining the index i of each pair.

```
1 names = ['Asif', 'Farhan', 'Adil']
2 ages = [24, 50, 18]
3
4 for i, (name, age) in enumerate(zip(names, ages)):
5     print(i, name, age)
6
```

```
→ 0 Asif 24
    1 Farhan 50
    2 Adil 18
```

When used with tuples, zip() works by pairing the elements from tuples based on their positions. The resulting iterable contains tuples where the i-th tuple contains the i-th element from each input tuple.

```
1 #Python zip() with Tuple
2 tuple1 = (1, 2, 3)
3 tuple2 = ('a', 'b', 'c')
4 zipped = zip(tuple1, tuple2)
5 print(zipped)
6 #print(dict(zipped))
7 result = list(zipped)
8 print(result)
9
```

```
→ <zip object at 0x000002BC85AFDB80>
    [(1, 'a'), (2, 'b'), (3, 'c')]
```

Converting to Dictionary:

The dict() function is used to convert the zip object zipped to a dictionary. Since the zip object is an iterable of tuples, each tuple contains a pair of elements from tuple1 and tuple2, which are used as key-value pairs in the resulting dictionary.

```
1 #Python zip() with Tuple
2 tuple1 = (1, 2, 3)
```

```
3 tuple2 = ('a', 'b', 'c')
4 zipped = zip(tuple1, tuple2)
5 print(zipped)
6 #print(dict(zipped))
7 result = dict(zipped)
8 print(result)
```

```
→ <zip object at 0x000002BC85AFC9C0>
{1: 'a', 2: 'b', 3: 'c'}
```

## Python zip() with Multiple Iterables

Python's `zip()` function can also be used to combine more than two iterables. It can take multiple iterables as input and return an iterable of tuples, where each tuple contains elements from the corresponding positions of the input iterables.

```
1 list1 = [1, 2, 3]
2 list2 = ['a', 'b', 'c']
3 list3 = ['x', 'y', 'z']
4 zipped = zip(list1, list2, list3)
5 result = list(zipped)
6 print(result)
7
```

```
→ [(1, 'a', 'x'), (2, 'b', 'y'), (3, 'c', 'z')]
```

## Unzipping Using `zip()`

Using `zip()` to Map Values:

The `zip()` function is used to combine corresponding elements from `name`, `roll_no`, and `marks` into tuples. Each tuple contains one element from each of the input lists, creating a mapping of values.

```
1 Mapped List of Tuples:
```

```
2
```

```
3 Before zip(*mapped) is used, mapped contains a list of tuple
4 zipped set of values from the original lists. <br>
```

```
5 For example, mapped might look like [(name1, roll_no1, mark1
```

```
6
```

```
7 Unzipping:<br>
```

```
8 The * operator is used to unpack the elements of mapped befo
```

```
9 This operation effectively "unzips" the list of tuples into
```

```
10 So, zip(*mapped) is equivalent to zip((name1, roll_no1, mark
11 Each tuple in mapped becomes a separate argument to zip()).<b
12
13 Zip Functionality:
14 The zip() function then pairs up corresponding elements from
15 In this case, it pairs up the first elements of each tuple,
16 Result:
17
18 The result of zip(*mapped) is a new iterable containing tuple
19 the first elements of each original tuple, the second element
20 This effectively "unzips" the original list of tuples back into
21 In summary, zip(*mapped) is a way to reverse the zipping operation
22 and then zipping their elements together.
```

```
1 # initializing lists
2 name = ["Atif", "Ali", "Fazal", "Obaid"]
3 roll_no = [4, 1, 3, 2]
4 marks = [40, 50, 60, 70]
5
6 # using zip() to map values
7 mapped = zip(name, roll_no, marks)
8
9 # converting values to print as list
10 mapped = list(mapped)
11
12 # printing resultant values
13 print("The zipped result is : ", end="")
14 print(mapped)
15
16 print("\n")
17
18 # unzipping values
19 namz, roll_noz, marksz = zip(*mapped)
20
21 print("The unzipped result: \n", end="")
22
23 # printing initial lists
24 print("The name list is : ", end="")
```

```
25 print(namz)
26
27 print("The roll_no list is : ", end="")
28 print(roll_noz)
29
30 print("The marks list is : ", end="")
31 print(marksz)
32
```

→ The zipped result is : [('Atif', 4, 40), ('Ali', 1, 50), ('Fazal', 3, 60), ('Obaid', 2,

The unzipped result:  
The name list is : ('Atif', 'Ali', 'Fazal', 'Obaid')  
The roll\_no list is : (4, 1, 3, 2)  
The marks list is : (40, 50, 60, 70)

```
1 # Python code to demonstrate the application of
2 # zip()
3
4 # initializing list of student.
5 students = ["Saeed", "Hassan", "Farhan", "Adnan", "Rahim"]
6
7 # initializing their gpa
8 scores = [4.0, 1.5, 3.7, 2.8, 4.0]
9
10 # printing students and scores.
11 for pl, sc in zip(students, scores):
12     print("Student : %s Score: %.1f" % (pl, sc))
13
```

→ Student : Saeed Score: 4.0  
Student : Hassan Score: 1.5  
Student : Farhan Score: 3.7  
Student : Adnan Score: 2.8  
Student : Rahim Score: 4.0

%-10s specifies a left-aligned string (student name) with a width of 10 characters. This ensures that each student name occupies at least 10 characters, providing consistent spacing. %.1f specifies a floating-point number (score) formatted with one digit after the decimal point. With this adjustment, the output will have consistent spacing between the student names and their scores.

```
1 students = ["Saeed", "Hassan", "Farhan", "Adnan", "Rahim"]
2 scores = [4.0, 1.5, 3.7, 2.8, 4.0]
3
4 for pl, sc in zip(students, scores):
5     print("Student : %-7s Score : %.1f" % (pl, sc))
6
```

```
→ Student : Saeed  Score : 4.0
Student : Hassan  Score : 1.5
Student : Farhan  Score : 3.7
Student : Adnan   Score : 2.8
Student : Rahim   Score : 4.0
```

What is Nested Dictionary in Python? In Python, a nested dictionary is a dictionary inside a dictionary. It's a collection of dictionaries into one single dictionary. nested\_dict = { 'dictA': {'key\_1': 'value\_1'}, 'dictB': {'key\_2': 'value\_2'}} Here, the nested\_dict is a nested dictionary with the dictionary dictA and dictB. They are two dictionary each having own key and value.

```
1 #Create a Nested Dictionary
2 #We're going to create dictionary of people within a diction
3 people = {1: {'name': 'John', 'age': '27', 'Gender': 'Male'}
4             2: {'name': 'Marie', 'age': '22', 'Gender': 'Femal
5
6 print(people)
```

```
→ {1: {'name': 'John', 'age': '27', 'Gender': 'Male'}, 2: {'name': 'Marie', 'age': '22', '
```



### Access elements of a Nested Dictionary

To access element of a nested dictionary, we use indexing [] syntax in Python.

```
1 people = {1: {'name': 'John', 'age': '27', 'gender': 'Male'}
2                 2: {'name': 'Marie', 'age': '22', 'gender': 'Femal
3
4 print(people[1]['name']) #we print the value of key name usi
5 print(people[1]['age'])
6 print(people[1]['gender'])
```

```
→ John
27
Male
```

Add element to a Nested Dictionary: How to change or add elements in a nested dictionary? we create an empty dictionary 3 inside the dictionary people.

Then, we add the key:value pair i.e people[3]['Name'] = 'Luna' inside the dictionary 3. Similarly, we do this for key age, gender and married one by one.

```
1 people = {1: {'name': 'John', 'age': '27', 'gender': 'Male'}
2                 2: {'name': 'Marie', 'age': '22', 'gender': 'Female'}
3
4 people[3] = {}
5
6 people[3]['name'] = 'Luna'
7 people[3]['age'] = '24'
8 people[3]['gender'] = 'Female'
9 people[3]['married'] = 'No'
10
11 #print(people[3])
12 print(people)
```

```
→ {1: {'name': 'John', 'age': '27', 'gender': 'Male'}, 2: {'name': 'Marie', 'age': '22', 'gender': 'Female'}, 3: {'name': 'Luna', 'age': '24', 'gender': 'Female'}
```

Add another dictionary to the nested dictionary: we assign a dictionary literal to people[4]. The literal have keys name, age and sex with respective values.

```
1 people = {1: {'name': 'John', 'age': '27', 'gender': 'Male'}
2                 2: {'name': 'Marie', 'age': '22', 'gender': 'Female'}
3                 3: {'name': 'Luna', 'age': '24', 'gender': 'Female'}
4
5 people[4] = {'name': 'Peter', 'age': '29', 'gender': 'Male',
6 #print(people[4])
7 print(people)
```

```
→ {1: {'name': 'John', 'age': '27', 'gender': 'Male'}, 2: {'name': 'Marie', 'age': '22', 'gender': 'Female'}, 3: {'name': 'Luna', 'age': '24', 'gender': 'Female'}, 4: {'name': 'Peter', 'age': '29', 'gender': 'Male'}
```

Delete elements from a Nested Dictionary In Python, we use “ del “ statement to delete elements from nested dictionary.

```
1 people = {1: {'name': 'John', 'age': '27', 'sex': 'Male'},  
2                 2: {'name': 'Marie', 'age': '22', 'sex': 'Female'}  
3                 3: {'name': 'Luna', 'age': '24', 'sex': 'Female'},  
4                 4: {'name': 'Peter', 'age': '29', 'sex': 'Male', '  
5  
6 #del people[3]['married']  
7 #del people[4]['married']  
8 del people[3]['married'], people[4]['married']  
9  
10 print(people[3])  
11 print(people[4])
```

```
→ { 'name': 'Luna', 'age': '24', 'sex': 'Female'}  
{'name': 'Peter', 'age': '29', 'sex': 'Male'}
```

How to delete dictionary from a nested dictionary?

```
1 people = {1: {'name': 'John', 'age': '27', 'sex': 'Male'},  
2                 2: {'name': 'Marie', 'age': '22', 'sex': 'Female'}  
3                 3: {'name': 'Luna', 'age': '24', 'sex': 'Female'},  
4                 4: {'name': 'Peter', 'age': '29', 'sex': 'Male'}}  
5  
6 del people[3], people[4]  
7 print(people)
```

```
→ {1: {'name': 'John', 'age': '27', 'sex': 'Male'}, 2: {'name': 'Marie', 'age': '22', 'se>
```



Iterating Through a Nested Dictionary Using the for loops, we can iterate through each elements in a nested dictionary.

```
1 people = {1: {'Name': 'John', 'Age': '27', 'Sex': 'Male'},  
2                 2: {'Name': 'Marie', 'Age': '22', 'Sex': 'Female'}  
3  
4 for p_id, p_info in people.items():  
5     print("\nPerson ID:", p_id)  
6  
7     for key in p_info:  
8         print(key + ':', p_info[key])
```



```
Person ID: 1
```

```
Name: John
```

```
Age: 27
```

```
Sex: Male
```

```
Person ID: 2
```

```
Name: Marie
```

```
Age: 22
```

```
Sex: Female
```

## 1 Nested Dictionary Comprehension

2 We can add dictionary comprehensions to dictionary comprehensions.  
3 nested dictionaries.

4 we have constructed a multiplication table in a nested dictionary.  
5 Whenever nested dictionary comprehension is used, Python first  
6 goes to the outer one and then goes to the inner one.

7 people is a nested dictionary. The internal dictionary 1 and  
8 Here, both the dictionary have key name, age , male with different values.

```
1 people = {1: {'Name': 'John', 'Age': '27', 'gender': 'Male'}  
2                 2: {'Name': 'Marie', 'Age': '22', 'gender': 'Female'}  
3  
4 dict1= {p_id:{key: p_info[key] for key in p_info} for p_id,  
5 print(dict1)
```



```
{1: {'Name': 'John', 'Age': '27', 'gender': 'Male'}, 2: {'Name': 'Marie', 'Age': '22', 'gender': 'Female'}
```



```
1 dictionary = {  
2     k1: {k2: k1 * k2 for k2 in range(1, 6)} for k1 in range(1, 6)  
3 }  
4 print(dictionary)  
5
```



```
{2: {1: 2, 2: 4, 3: 6, 4: 8, 5: 10}, 3: {1: 3, 2: 6, 3: 9, 4: 12, 5: 15}, 4: {1: 4, 2: 8, 3: 12, 4: 16, 5: 20}}
```



```
1 # The below code is equivalent to previous one given in cell  
2 dictionary = dict()  
3 for k1 in range(2, 5):  
4     dictionary[k1] = dict()
```

```
5     for k2 in range(1, 6):
6         dictionary[k1][k2] = k1*k2
7 print(dictionary)
8
9
```

→ {2: {1: 2, 2: 4, 3: 6, 4: 8, 5: 10}, 3: {1: 3, 2: 6, 3: 9, 4: 12, 5: 15}, 4: {1: 4, 2: 8, 3: 12, 4: 16, 5: 20}}



```
1 # The below code is equivalent to previous one given in cell
2 dictionary = dict()
3 for k1 in range(2, 5):
4     dictionary[k1] = {k2: k1*k2 for k2 in range(1, 6)}
5 print(dictionary)
6
```

→ {2: {1: 2, 2: 4, 3: 6, 4: 8, 5: 10}, 3: {1: 3, 2: 6, 3: 9, 4: 12, 5: 15}, 4: {1: 4, 2: 8, 3: 12, 4: 16, 5: 20}}



**Advantages of Using Dictionary Comprehension** As we can see, dictionary comprehension shortens the process of dictionary initialization by a lot. It makes the code more pythonic.

Using dictionary comprehension in our code can shorten the lines of code while keeping the logic intact.

## ▼ Warnings on Using Dictionary Comprehension

Even though dictionary comprehensions are great for writing elegant code that is easy to read, they are not always the right choice.

We must be careful while using them as :

They can sometimes make the code run slower and consume more memory. They can also decrease the readability of the code. We must not try to fit a difficult logic or a large number of dictionary comprehension inside them just for the sake of making the code single lined. In these cases, It is better to choose other alternatives like loops.

The outer dictionary has keys representing student names (in this case, "Esa"). The values corresponding to each student key are inner dictionaries. Each inner dictionary represents a subject taken by the student. The keys of the inner dictionaries are the subjects, and the values are dictionaries. The inner dictionaries contain assessment names as keys and their corresponding

scores as values. or Outer key: Student name (e.g., "Esa"). Outer value: Dictionary representing subjects and their corresponding assessments and scores. Inner keys: Subject names (e.g., "Math", "Science", "English"). Inner values: Dictionaries representing assessments and their scores for each subject.

```
1 ### Example data
2 students = ['farhan', 'Esa']
3 subjects = ['Math', 'Science', 'English']
4 num_assessments =3
5
6 # Nested dictionary to store grades
7 grading_system = {
8     student: {
9         subject: {f'Assessment_{i+1}': 0 for i in range(num_
10            for subject in subjects
11        }
12    for student in students
13 }
14
15 # Function to update grades
16 def update_grades(student, subject, assessment, grade):
17     print("entering2")
18     grading_system[student][subject][assessment] = grade
19
20 # Function to display grading system
21 def display_grading_system():
22     print("entering3")
23     for student, subjects in grading_system.items():
24         print(f"{student}:")
25         for subject, assessments in subjects.items():
26             print(f"  {subject}: {assessments}")
27
28 # Updating grades using for loop and input function
29 #We use nested for loops to iterate over each student, subje
30 #and prompt the user to input grades.
31 for student in students:
32     for subject in subjects:
33         for i in range(num_assessments):
34             assessment = f'Assessment_{i+1}'
```

```
35         grade = float(input(f"Enter grade for {student}"))
36         update_grades(student, subject, assessment, grad)
37
38 # Displaying updated grading system
39 print("Updated Grading System:")
40 display_grading_system()
41
```

```
→ Enter grade for farhan in Math Assessment_1: 33
entering2
Enter grade for farhan in Math Assessment_2: 33
entering2
Enter grade for farhan in Math Assessment_3: 33
entering2
Enter grade for farhan in Science Assessment_1: 33
entering2
Enter grade for farhan in Science Assessment_2: 33
entering2
Enter grade for farhan in Science Assessment_3: 33
entering2
Enter grade for farhan in English Assessment_1: 33
entering2
Enter grade for farhan in English Assessment_2: 33
entering2
Enter grade for farhan in English Assessment_3: 33
entering2
Enter grade for Esa in Math Assessment_1: 33
entering2
Enter grade for Esa in Math Assessment_2: 33
entering2
Enter grade for Esa in Math Assessment_3: 33
entering2
Enter grade for Esa in Science Assessment_1: 33
entering2
Enter grade for Esa in Science Assessment_2: 33
entering2
Enter grade for Esa in Science Assessment_3: 33
entering2
Enter grade for Esa in English Assessment_1: 33
entering2
Enter grade for Esa in English Assessment_2: 33
entering2
Enter grade for Esa in English Assessment_3: 33
entering2
Updated Grading System:
entering3
farhan:
    Math: {'Assessment_1': 33.0, 'Assessment_2': 33.0, 'Assessment_3': 33.0}
    Science: {'Assessment_1': 33.0, 'Assessment_2': 33.0, 'Assessment_3': 33.0}
    English: {'Assessment_1': 33.0, 'Assessment_2': 33.0, 'Assessment_3': 33.0}
Esa:
    Math: {'Assessment_1': 33.0, 'Assessment_2': 33.0, 'Assessment_3': 33.0}
    Science: {'Assessment_1': 33.0, 'Assessment_2': 33.0, 'Assessment_3': 33.0}
```

```
English: {'Assessment_1': 33.0, 'Assessment_2': 33.0, 'Assessment_3': 33.0}
```

The `dict.copy()` method returns a copy (shallow copy) of the dictionary.

The `copy()` method doesn't take any arguments.

This method returns a shallow copy of the dictionary. It doesn't modify the original dictionary.

```
1 original_marks = {'Physics':67, 'Maths':87}
2
3 copied_marks = original_marks.copy()
4
5
6 print('Original Marks:', original_marks)
7 print('Copied Marks:', copied_marks)
8
```

```
→ Original Marks: {'Physics': 67, 'Maths': 87}
Copied Marks: {'Physics': 67, 'Maths': 87}
```

#### Dictionary `copy()` Method Vs `=` Operator

When the `copy()` method is used, a new dictionary is created which is filled with a copy of the references from the original dictionary.

When the `=` operator is used, a new reference to the original dictionary is created.

```
1 original = {1:'one', 2:'two'}
2 new = original
3 # removing all elements from the list
4 new.clear()
5 print('new: ', new)
6 print('original: ', original)
7
8 #Here, when the new dictionary is cleared, the original dict
```

```
→ new: {}
original: {}
```

```
1 #Using copy() to Copy Dictionaries
2 #Here, when the new dictionary is cleared, the original dict
3 original = {1:'one', 2:'two'}
4 new = original.copy()
5
```

```
6
7 # removing all elements from the list
8 new.clear()
9
10 print('new: ', new)
11 print('original: ', original)
```

```
→ new: {}
original: {1: 'one', 2: 'two'}
```

## setdefault() Parameters

setdefault() takes a maximum of two parameters:

key - the key to be searched in the dictionary

default\_value (optional) - key with a value default\_value is inserted to the dictionary if the key is not in the dictionary.

If default value is not provided, the default\_value will be None.

Return Value from setdefault()

setdefault() returns:

value of the key if it is in the dictionary

None if the key is not in the dictionary and default\_value is not specified

default\_value if key is not in the dictionary and default\_value is specified

```
1 #How setdefault() works when key is in the dictionary?
2 person = {'name': 'hassan', 'age': 22}
3
4 age = person.setdefault('age')
5 print('person = ', person)
6 print('Age = ', age)
```

```
→ person = {'name': 'hassan', 'age': 22}
Age = 22
```

```
1 person = {'name': 'Hassan'}
2
3 # key is not in the dictionary
4 salary = person.setdefault('salary')
5 print('person = ', person)
6 print('salary = ', salary)
7
```

```
8 # key is not in the dictionary
9 # default_value is provided
10 age = person.setdefault('age', 22)
11 print('person = ', person)
12 print('age = ', age)
```

```
→ person = {'name': 'Hassan', 'salary': None}
salary = None
person = {'name': 'Hassan', 'salary': None, 'age': 22}
age = 22
```

1 Start coding or generate with AI.

1 Start coding or generate with AI.

## ✓ Notebook: 30) Deep\_learning.ipynb

```
1 conda install pytorch torchvision -c pytorch
```

```
→ Retrieving notices: ...working... done
Collecting package metadata (current_repodata.json): ...working... failed

Note: you may need to restart the kernel to use updated packages.
Retrying (Retry(total=2, connect=None, read=None, redirect=None, status=None)) after
Retrying (Retry(total=1, connect=None, read=None, redirect=None, status=None)) after
Retrying (Retry(total=0, connect=None, read=None, redirect=None, status=None)) after
```

```
ERROR conda.notices.fetch:get_channel_notice_response(68): Request error <HTTPSConne
Retrying (Retry(total=2, connect=None, read=None, redirect=None, status=None)) after
Retrying (Retry(total=1, connect=None, read=None, redirect=None, status=None)) after
```

```
1 import torch
2 print(torch.cuda.is_available())
3 print(torch.rand(2,2))
```

```
ModuleNotFoundError
Cell In[3], line 1
----> 1 import torch
      2 print(torch.cuda.is_available())
      3 print(torch.rand(2,2))

ModuleNotFoundError: No module named 'torch'
```

```
1 # Load library
2 import numpy as np
3 # Create a vector as a row
4 vector_row = np.array([1, 2, 3])
5 # Create a vector as a column
6 vector_column = np.array([[1],
```

```
7 [2],  
8 [3]])
```

```
1 vector_row
```

```
→ array([1, 2, 3])
```

```
1 vector_column
```

```
→ array([[1],  
         [2],  
         [3]])
```

```
1 # Load libraries  
2 import numpy as np  
3 from scipy import sparse  
4 # Create a matrix  
5 matrix = np.array([[0, 0],  
6 [0, 1],  
7 [3, 0]])  
8 # Create compressed sparse row (CSR) matrix  
9 matrix_sparse = sparse.csr_matrix(matrix)
```

```
1 matrix_sparse
```

```
→ <3x2 sparse matrix of type '<class 'numpy.intc'>'  
   with 2 stored elements in Compressed Sparse Row format>
```

```
1 matrix
```

```
→ array([[0, 0],  
         [0, 1],  
         [3, 0]])
```

```
1 print(matrix_sparse)
```

```
2
```

```
→ (1, 1)      1  
   (2, 0)      3
```

```
1 Start coding or generate with AI.
```

## ✓ Notebook: 4) Functions.ipynb

Function is an independent block of statements that perform a specific task. Function is a great concept, which makes programming easier. Suppose you are building a website and you are asked to insert data in database, you will write a function that will take correct data as input, and the function will insert data in the database and display a success message.

A function can be viewed as a black box – what matters to the user is the input it accepts and the output it produces, not how it processes the input internally. The main role of a function is to take given input, perform a specific task, and return the desired output.

Code inside the function matters to the developer implementing or maintaining it, even if it doesn't matter to the user of the function

Function follows two principles: Abstraction: It does not matter what code is inside the function but it matters how to execute the function means what input should be given to function and what expected output a function should produce. Procedural abstraction is a way of hiding the details of how a procedure is implemented from the user. This allows the user to perform actions without worrying about the specifics of how those actions are carried out. By using procedural abstraction, we can create procedures that are more reliable and easier to use.

Decomposition: Many small TVs combined to make a big TV and you see a big picture. similarly, when you build an application, you think of functions like login, logout and registration functions and all these functions constitute the application.

A function is a block of code that performs a specific task.

Suppose we need to create a program to make a circle and color it. We can create two functions to solve this problem:

function to create a circle

function to color the shape

Dividing a complex problem into smaller chunks makes our program easy to understand and reuse.

 Component of function can be seen above: is\_even is identifier and identifier can be variable name,

function name and class name. Argument is input to functions.

Difficulty level is the parameter while easy, medium, hard are the arguments

Brightness level in TV is parameter, while 0 to 1 values are arguments

A parameter is a variable in a function definition. It is a placeholder and hence does not have a

concrete value. An argument is a value passed during function invocation. In a way, arguments fill in the place the parameters have held for them.

  optional docstring is the reading manual of function. use triple inverted commas " doc string". for multiline string. IT contains function description, what inputs are required, what output will be retuned, created by, lasted Edited on. Function body: code logic of the function where the main job is done. return output of function function call/invoke

Python Function Arguments Arguments are inputs given to the function. Here, we passed 'John' as an argument to the greet() function.

We can pass different arguments in each call, making the function re-usable and dynamic.

```
1 #
2 def greet(name):
3     print("Hello", name)
4
5 # pass argument
6 greet("Yasir")
```

 Hello Yasir

## Parameters and Arguments Parameters

**Parameters** are the variables listed inside the parentheses in the function definition. They act like placeholders for the data the function can accept when we call them.

Think of parameters as the blueprint that outlines what kind of information the function expects to receive. In the following example, the print\_age() function takes age as its input. However, at this stage, the actual value is not specified. The age parameter is just a placeholder waiting for a specific value to be provided when the function is called.

**Arguments** Arguments are the actual values that we pass to the function when we call it.

Arguments replace the parameters when the function executes.

```
1 def print_age(age): # age is a parameter
2     print(age)
3 print(24)
```

 24

## The pass Statement

The pass statement serves as a placeholder for future code, preventing errors from empty code

blocks. It's typically used where code is planned but has yet to be written.

In Python programming, the pass statement is a null statement which can be used as a placeholder for future code. Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future. In such cases, we can use the pass statement. The syntax of the pass statement is: pass

```
1 def future_function():
2     pass
3
4 # this will execute without any action or error
5 future_function()
```

```
1 def future_function():
2     pass
3
4 # this will execute without any action or error
5 future_function()
```

```
1 # Use pass with conditional statement
2 n = 10
3
4 # use pass inside if statement
5 if n > 10:
6     pass
7
8 print('Hello')
9
10 #Here, notice that we have used the pass statement inside the if block
11 #However, nothing happens when the pass is executed. It results in an IndentationError.
```

→ Hello

Here, we will get an error message: IndentationError: expected an indented block Note: The difference between a comment and a pass statement in Python is that while the interpreter ignores a comment entirely, pass is not ignored.

```
1 #Suppose we didn't use pass or just put a comment as:
2 n = 10
```

```
3
4 if n > 10:
5     # write code later
6
7 print('Hello')
```

```
→ Cell In[6], line 7
    print('Hello')
    ^
IndentationError: expected an indented block after 'if' statement on line 4
```

Use of pass Statement inside Function or Class We can do the same thing in an empty function or class as well. For example,

```
1 def function(args):
2     pass
3 class Example:
4     pass
```

```
1 def is_even(num):
2     """
3         this function tells if the given number is odd or even
4         input: any valid integer
5         output: even or odd
6         created by: Atif
7         Last Edited on: 31 Jan 2024
8     """
9     if num % 2==0:
10         return "Even"
11     else:
12         return "Odd"
```

```
1 docstring----manual of function
```

```
1 is_even(2)
```

```
→ 'Even'
```

```
1 is_even(5)
```

→ 'Odd'

```
1 print(is_even.__doc__)
```

→

this function tells if the given number is odd or even  
input: any valid integer  
output: even or odd  
created by: Atif  
Last Edited on: 31 Jan 2024

```
1 print(is_even.__doc__)
```

→

this function tells if the given number is odd or even  
input: any valid integer  
output: even or odd  
created by: Atif  
Last Edited on: 31 Jan 2024

```
1 import math
```

```
1 print(math.sqrt.__doc__)
```

→ Return the square root of x.

```
1 print(print.__doc__)
```

→ Prints the values to a stream, or to sys.stdout by default.

```
sep
    string inserted between values, default a space.
end
    string appended after the last value, default a newline.
file
    a file-like object (stream); defaults to the current sys.stdout.
flush
    whether to forcibly flush the stream.
```

```
1 for i in range (1, 11):
2     print(f'{i} is {is_even(i)}')
```

```
→ 1 is Odd
  2 is Even
  3 is Odd
  4 is Even
  5 is Odd
  6 is Even
  7 is Odd
  8 is Even
  9 is Odd
  10 is Even
```

```
1 print(is_even.__doc__)
2
```

```
→
this function tells if the given number is odd or even
input: any valid integer
output: even or odd
created by: Atif
Last Edited on: 31 Jan 2024
```

```
1 print(is_even.__doc__)
```

```
→
this function tells if the given number is odd or even
input: any valid integer
output: even or odd
created by: Atif
Last Edited on: 31 Jan 2024
```

```
1 print(print.__doc__) # you can print documentation of any fu
```

```
→ Prints the values to a stream, or to sys.stdout by default.
```

```
sep
    string inserted between values, default a space.
end
    string appended after the last value, default a newline.
file
    a file-like object (stream); defaults to the current sys.stdout.
flush
    whether to forcibly flush the stream.
```

```
1 print(type.__doc__)
```

```
→ type(object) -> the object's type
type(name, bases, dict, **kwds) -> a new type
```

```
1 import math
```

```
1 print(math.sqrt(23))
```

→ 4.795831523312719

```
1 math.sqrt.__doc__
```

→ 'Return the square root of x.'

1 There are two point of views related to function. <br>  
2 One is point of view of the one who has written the function  
3 another point of view is of the person who uses the function  
4 Generally, senior programmer writes the function<br>  
5 and junior programmar uses the function.  
6 Usually, Both codes(senior programmer written code+ JP codes  
7

create a new file named as demo\_func in IDLE (copy even/odd code here), save it  
programs/python312/lib, so this will file will become a library file, run the file in IDLE, then write  
import demo\_func as fd --fd.is\_even(3) --fd.is\_even('hello') now you can import it just like other for  
instance random

image.png

```
1 import math
```

```
1 math.ceil(2.5)
```

→ 3

```
1 from math import sqrt
```

```
1 Start coding or generate with AI.
```

```
1 sqrt(25)
```

→ 5.0

```
1 from Alu import is_even
```

```
1 is_even(5)
```

→ 'Odd'

```
1 import Alu
```

```
1 Alu.div_five(25)
```

→ 25 is divisible by 5

```
1 Alu.is_even(5)
```

→ 'Odd'

```
1 Alu.is_even(50)
```

→ 'Even'

```
1 Alu.is_even("hello data scientist")
```

→ Please enter a valid integer

```
1 import demo_func
```

```
1 demo_func.is_even(4) # this is code written by junior programer  
2  
3 # and code inside the demo_func is written by senior programer  
4 #both are in separte files
```

→ 'Even'

```
1 import even_odd_lib as math_eo
```

```
1 math_eo.is_even(5)
```

→ 'Odd'

```
1 math_eo.is_even('strddd')
```

→ -----

TypeError Traceback (most recent call last)

Cell In[3], line 1  
----> 1 math\_eo.is\_even('strddd')

File ~\AppData\Local\Programs\Python\Python312\Lib\even\_odd\_lib.py:9, in is\_even(num)  
 1 def is\_even(num):  
 2 '''  
 3 *this function tells if the given number is odd or even*  
 4 *input: any valid integer*  
(...)  
 7 *Last Edited on: 31 Jan 2024*  
 8 '''  
----> 9 if num % 2==0:  
10 return "Even"  
11 else:

TypeError: not all arguments converted during string formatting

```
1 demo_func.is_even("helllo")
```

→ -----

TypeError Traceback (most recent call last)

Cell In[3], line 1  
----> 1 demo\_func.is\_even("helllo")

File ~\AppData\Local\Programs\Python\Python312\Lib\demo\_func.py:9, in is\_even(num)  
 1 def is\_even(num):  
 2 '''  
 3 *this function tells if the given number is odd or even*  
 4 *input: any valid integer*  
(...)  
 7 *Last Edited on: 31 Jan 2024*  
 8 '''  
----> 9 if num % 2==0:  
10 return "Even"  
11 else:

TypeError: not all arguments converted during string formatting

```
1 isinstance(2, int)
```

→ True

```
1 isinstance("python", int)
```

→ False

```
1 isinstance("python", str)
```

→ True

```
1 isinstance(2, str)
```

→ False

```
1 def is_even(num):
2
3     if not isinstance(num, int):
4         print("Please give an integer input.")
5         return None
6     if num % 2==0:
7         return "Even"
8     else:
9         return "Odd"
```

```
1 is_even('pyton')
```

→ Please give an integer input.

```
1 is_even(4)
```

→ 'Even'

```
1
2 if u pass a string 'hello' to the above function, the functio
3 It is only mistake of senior
4 programmer who wrote the function. becos the function can ha
5 nd can not deal with strings.
6 when developing functions, one would have to think all possi
7 and try to deal with those possibilites in to avoid crashing .
8 companies would not allow the above error. so there is More
9 So we discussed two point of views related to function.
10 the above program need to be fixed to handle strings.
```

```
1 #import demo_func as df
2 from demo_func import is_even
```

```
1 is_even(2)
```

→ 'Even'

```
1 !python --version
```

→ Python 3.12.2

<https://pythontutor.com/render.html#mode=display>

<https://pythontutor.com/visualize.html#mode=edit> python tutor where u can execute your code line by line. and can visually see the execution of a function. Global frame is a part of memory where the whole program is executed and program variables are created. when a function is called or executed in python, a separate frame is created where all the function code is executed and function local variables are created. when function finishes its job, the frame/block is destroyed.

```
1 def is_even(num):
2     """
3         this function tells if the given number is odd or even
4         input: any valid integer
5         output: even or odd
6         created by: Atif
7         Last Edited on: 31 Jan 2024
8     """
9     if num % 2==0:
10         return "Even"
11     else:
12         return "Odd"
13 x= is_even(4)
14 print(x)
```

→ Even

```
1 if the function is returning nothing, still it is returning
```

```
def is_even(num):
```

```
if num % 2==0:  
    print("Even")  
else:  
    print ("Odd")
```

x= is\_even(4) print(x)

Python has 4 different arguments:

```
1 def power(a, b):  
2     return a**b  
3
```

```
1 power(2,3)
```

→ 8

```
1 power(3,2)
```

→ 9

```
1 power(2) # function crashes becos it is expecting 2 argument  
2 #Good practices should not allow functions to crash. A solut
```

→

```
-----  
TypeError                                     Traceback (most recent call last)  
Cell In[3], line 1  
----> 1 power(2) # function crashes becos it is expecting 2 arguments and we supplied  
one.  
      2 #Good practices should not allow functions to crash. A solution is to use  
defualt arguments
```

```
TypeError: power() missing 1 required positional argument: 'b'
```

```
1 power()
```



```
TypeError  
Cell In[4], line 1  
----> 1 power()
```

Traceback (most recent call last)

```
TypeError: power() missing 2 required positional arguments: 'a' and 'b'
```

## Default Arguments in Functions

Python allows functions to have default argument values. Default arguments are used when no explicit values are passed to these parameters during a function call.

```
1 # here parameters are assigned default values. if a and b's v  
2 def power(a=1, b=1):  
3     return a**b
```

```
1 power(2,3)
```

→ 8

```
1 power(2) # here 2 will be assigned to a and b default value 1
```

→ 2

```
1 power()
```

→ 1

**Positional Arguments:** The order in which arguments are sent, the parameters will receive arguments/values in that order. means the first argument will be received by first parameter and second argument will be received by 2nd parameter.

keyword argument overrides the positional arguments. suppose there is a function with so many parameters and u donot remember the order of parameters, then in that case use keyword arguments without worrying about the position of parameters. Imagine 50 parameters, remember the names and order of parameters is quite tedious. In such case, pass the arguments or values by their parameter names. priority of Keyword argument is greater than positional arguments.

Keyword-only arguments mean whenever we pass the arguments(or value) by their parameter names at the time of calling the function in Python in which if you change the position of arguments then there will be no change in the output. In keyword arguments, arguments are assigned based on the name of the arguments. In such scenarios, the position of arguments

doesn't matter. Keyword-only arguments in Python refer to function arguments that can only be specified by parameter name, rather than by position. This means that you must provide the parameter name when calling the function, otherwise, you'll get a syntax error. When using keyword-only arguments, the order in which you pass arguments does not matter, as long as you specify the parameter names.

```
1 def power(a=1, b=1, c=1):  
2     return a**b
```

```
1 power(c=4, b=2, a=3) # keyword arguments ans: 3**2=9, accord
```

→ 9

```
1 power(b=2, a=5)
```

→ 25

```
1 # Changing the order of arguments has no effect because they  
2 power(a=3, b=3)
```

→ 27

```
1 def pers_details( name, fname, address):  
2     print(f"my name is {name}, father name is {fname}, and a  
3  
4 pers_details('Ali', 'salih','peshawar')
```

→ my name is Ali, father name is salih, and address is peshawar

```
1 def pers_details( name, fname, address):  
2     print(f"my name is {name}, father name is {fname}, and a  
3  
4 pers_details(fname='Ali', name='salih',address='peshawar')
```

→ my name is salih, father name is Ali, and address is peshawar

```
1 #print() is a flexible function that can handle any number o  
2 print(1)  
3 print(1,2)  
4 print(1,2,3,4,5,6,7,8,98,8,8,8,8,0,65,55,65,55,5,55,44,4)
```

```
5 # how to design such a function that can handle flexible no  
6
```

```
→ 1  
1 2  
1 2 3 4 5 6 7 8 98 8 8 8 8 0 65 55 65 55 5 55 44 4
```

## ▼ Python Function With Arbitrary Arguments

Sometimes, we do not know in advance the number of arguments that will be passed into a function. To handle this kind of situation, we can use arbitrary arguments in Python. Arbitrary arguments allow us to pass a varying number of values during a function call. We use an asterisk ( $*$ ) before the parameter name (args) to denote this kind of argument. and python interpreter understand and automatically convert this into tuple named args.

This feature is helpful in scenarios where you want your function to be versatile and capable of accepting different numbers of inputs without needing to specify each one individually. By using `*args`, you're essentially telling Python, "Hey, collect all the remaining positional arguments into this tuple named args."

```
1 # program to find a product of multiple numbers  
2  
3 def find_prod(*numbers):  
4     result = 1  
5     print(numbers)  
6     print(type(numbers))  
7  
8     for num in numbers:  
9         result = result * num  
10  
11    print("prod = ", result)  
12  
13 # function call with 3 arguments  
14 find_prod(1, 2, 3)  
15  
16 # function call with 2 arguments  
17 find_prod(4, 9)  
18 # function call with 4 arguments  
19 find_prod(4, 9, 10, 9, 10)
```

```
20
```

```
21 #print() function implement this type of technique to have f
```

```
→ (1, 2, 3)
<class 'tuple'>
prod = 6
(4, 9)
<class 'tuple'>
prod = 36
(4, 9, 10, 9, 10)
<class 'tuple'>
prod = 32400
```

```
1 # program to find sum of multiple numbers
2
3 def find_sum(*numbers):
4     result = 0
5
6     for num in numbers:
7         result = result + num
8
9     print("Sum = ", result)
10
11 # function call with 3 arguments
12 find_sum(1, 2, 3)
13
14 # function call with 2 arguments
15 find_sum(4, 9)
16 # function call with 4 arguments
17 find_sum(4, 9, 10, 9)
```

```
→ Sum = 6
Sum = 13
Sum = 32
```

```
1 def display_record(*args):
2     for item in args:
3         print(item, end=' ')
4
5 list_stds= [ (1, 'Ihsan', 'SE', 21, 'kark', 100), (2, 'Aslam
6 for record in list_stds:
7
```

```
8     display_record(record)
9
```

```
→ (1, 'Ihsan', 'SE', 21, 'kark', 100) (2, 'Aslam', 'CS', 21, 'kohat', 100)
```

```
1 def func_a():
2     print('inside func_a') #func_a() is defined: When called
3                         # Since there's no return stateme
4 def func_b(y):
5     print('inside func_b') #func_b(y) is defined: When calle
6     return y
7
8
9 print(func_a())
10
11 print(5+func_b(2))
12
```

```
→ inside func_a
None
inside func_b
7
```

```
1 def give_free_water():
2     print("Here's your free water!")
3     # No return value (like no bill for water)
4
5 def order_side_dish(price):
6     print("Preparing your side dish...")
7     return price # Returns the price you need to pay
8
9
10 # Scenario 1: You ask for free water and the waiter confirms
11 print(give_free_water())
12 # Output:
13 # Here's your free water!
14 # None (because water is free, no bill)
15
16 # Scenario 2: You order fries ($2) and add it to your main m
17 print(5 + order_side_dish(2))
```

```
18 # Output:  
19 # Preparing your side dish...  
20 # 7 (total bill: $5 main + $2 side = $7)
```

```
→ Here's your free water!  
None  
Preparing your side dish...  
7
```

## Python Variable Scope

In Python, we can declare variables in three different scopes: **local scope, global, and nonlocal scope**. A variable scope specifies the region where we can access a variable. For example,

```
def add_numbers():  
    sum = 5 + 4
```

Here, the sum variable is created inside the function, so it can only be accessed within it (local scope). This type of variable is called a local variable.

Based on the scope, we can classify Python variables into three types:

1. Local Variables
2. Global Variables
3. Nonlocal Variables

### Python Local Variables:

When we declare variables inside a function, these variables will have a local scope (within the function). We cannot access them outside the function. These types of variables are called local variables.

```
1 def greet():  
2  
3     # local variable  
4     message = 'Hello'  
5  
6     print('Local', message)  
7  
8 greet()  
9  
10 # try to access message variable  
11 # outside greet() function  
12  
13 print(message)  
14 #Here, the message variable is local to the greet() function
```

```
15 #That's why we get an error when we try to access it outside  
16 #To fix this issue, we can make the variable named message g
```

→ Local Hello

```
NameError  
Cell In[23], line 13  
    8 greet()  
    10 # try to access message variable  
    11 # outside greet() function  
--> 13 print(message)  
    14 #Here, the message variable is local to the greet() function, so it can only be  
accessed within the function.  
    15 #That's why we get an error when we try to access it outside the greet()  
function.  
    16 #To fix this issue, we can make the variable named message global.
```

```
NameError: name 'message' is not defined
```

## Python Global Variables:

In Python, a variable declared outside of the function or in global scope is known as a global variable. This means that a global variable can be accessed inside or outside of the function.

## Python Global Keyword

In Python, the global keyword allows us to modify the variable outside of the current scope. It is used to create a global variable and make changes to the variable in a local context.

## Access and Modify Python Global Variable

First let's try to access a global variable from the inside of a function,

```
1 c = 1 # global variable  
2  
3 def add():  
4     print('inside fun ',c)  
5  
6 add()  
7 print('outside func ',c)  
8 # Output: 1  
9 #Here, we can see that we have accessed a global variable fr
```

→ inside fun 1  
outside func 1

```
1 # declare global variable
2 message = 'Hello'
3
4 def greet():
5     # declare local variable
6     print('Local', message)
7
8 greet()
9
10 print('Global', message)
11 #This time we can access the message variable from outside o
12 #This is because we have created the message variable as the
13 #Now, message will be accessible from any scope (region) of
```

→ Local Hello  
Global Hello

```
1 # function local variable and main program global variable w
2 # A varialbe created inside a function is called local varia
3 # A varaible
4 def func(y):
5     x = 1 # local variable is created inside function
6     x = x+ 1
7     print(x)
8
9 x = 5
10 func(x)
11 print(x)
12
```

→ 2  
5

```
1 # return value is none
2 def g(y):
3     print(x) # we did not create x in a function, just print
4     print(x+1) # we did not change x, since x is immutable w
5
6 x = 5
7
```

```
8 g(x)
9 print(x)
10
11 # if function does not have local variabe x then it can acce
```

```
→ 5
6
5
```

```
1 #However, if we try to modify the global variable from insid
2 # global variable
3 c = 1
4
5 def add():
6     global c
7     # increment c by 2
8     c = c + 2
9
10    print(c)
11
12 add()
13
14 #This is because we can only access the global variable but
15 #The solution for this is to use the global keyword.
```

```
→ 3
```

```
1 # global variable
2 c = 1
3
4 def add():
5
6     # use of global keyword
7     global c
8
9     # increment c by 2
10    c = c + 2
11
12    print(c)
13
```

```
14 add()
15
16 # Output: 3
```

→ 3

```
1 def h(y):
2     x= x + 1 # attempting to change the value of global vari
3                     #which is not allowed
4
5 x = 5
6
7 h(x)
8 print(x)
9
10 #attempting to change the value of global variable x within
11 #Global variable might be accessed by other functions in a p
12 # in codes of other functions of main program
13
```

→

```
UnboundLocalError                                     Traceback (most recent call last)
Cell In[32], line 7
      3                     #which is not allowed
      5 x = 5
----> 7 h(x)
      8 print(x)
      10 #attempting to change the value of global variable x within a function, which
      is not allowed, its a logical.
      11 #Global variable might be accessed by other functions in a program, if one
      function shares the global varaiable, then issues might encounter
      12 # in codes of other functions of main program

Cell In[32], line 2, in h(y)
      1 def h(y):
----> 2     x= x + 1
```

**UnboundLocalError:** cannot access local variable 'x' where it is not associated with a value

Rule 1: If there is a global variable in the main program,  
then any function of the main program can access it

Rule 2: if a function does not have its own local variable, then it can use global varialbe of the main program

Rule 3: Any function in the program can just use the global variable, it cannot change it

```
1 # we can explicitly change the value of global variable x fr
2 #python has provided this feature to change global varialbe
3 def h(y):
4     global x
5     x += 1 # attempting to change the value of global variab
6
7
8 x = 5
9 h(x)
10 print(x)
```

→ 6

```
1 def f(x):
2     x = x + 1
3     print('in f(x): x =', x)
4     return x
5
6 x = 4
7 z = f(x)
8
9 print('in main program scope: z =', z)
10 print('in main program scope: x =', x)
```

→ in f(x): x = 5  
in main program scope: z = 5  
in main program scope: x = 4

1 Nested function: developing a function within another functi

```
1 def f():
2     print('inside f')
```

```
3
4     def g():
5         print('inside g')
6     g()
7
```

```
1 f()
```

→ inside f  
inside g

```
1 # if u call g() function directly, what would happen
2 g()
3 #main program does not know that g() function is defined.
4 #The concept is that nested/inner functions are hidden/abstr
5 #from the main program. So you cannot access g() without f()
```

→ -----  
**TypeError** Traceback (most recent call last)  
Cell In[35], line 2  
 1 # if u call g() function directly, what would happen  
----> 2 g()  
 3 #main program does not know that g() function is defined.  
 4 #The concept is that nested/inner functions are hidden/abstracted  
 5 #from the main program. So you cannot access g() without f(). some programmer  
hide code by specifying it inner functions

**TypeError:** g() missing 1 required positional argument: 'y'

## Global in Nested Functions

In Python, we can also use the global keyword in a nested function. For example,

```
1 def outer_function():
2     num = 20          #Local variable, exists only within the s
3
4     def inner_function():
5         global num  #global num declares num as a global var
6         num = 25
7
8     print("Before calling inner_function(): ", num) # num va
9     inner_function()
```

```
10     print("After calling inner_function(): ", num)
11
12 outer_function()
13 print("Outside both function: ", num) ## num variable printed
```

```
→ Before calling inner_function(): 20
    After calling inner_function(): 20
    Outside both function: 25
```

Rules of global Keyword The basic rules for global keyword in Python are:

1. When we create a variable inside a function, it is local by default.
2. When we define a variable outside of a function, it is global by default. You don't have to use the global keyword.
3. We use the global keyword to read and write a global variable inside a function.
4. Use of the global keyword outside a function has no effect.

In Python, the nonlocal keyword is used to declare that a variable inside a nested function is not local to it, nor is it global, but rather it belongs to the nearest enclosing scope. This feature is particularly useful when dealing with nested functions, where you want to modify variables from an outer scope within an inner function.

### Python Nonlocal Variables

In Python, nonlocal variables are used in nested functions whose local scope is not defined. This means that the variable can be neither in the local nor the global scope. We use the nonlocal keyword to create nonlocal variables. For example, This means that the message variable in inner() is referring to the same variable as the one defined in the enclosing outer() function. Therefore, when message is modified inside inner(), it affects the same message variable in outer().

```
1 # outside function
2 def outer():
3     message = 'local'
4
5     # nested function
6     def inner():
7         # declare nonlocal variable
8         nonlocal message
9
10    message = 'nonlocal'
11    print("inner:", message)
12
```

```
13     inner()
14     print("outer:", message)
15
16 outer()
17
18 #The inner() function is defined in the scope of another fun
19 # If we change the value of a nonlocal variable, the changes
20
```

```
→ inner: nonlocal
      outer: nonlocal
```

scenario where nonlocal variables are commonly used: Counter in a Recursive Function: When you have a recursive function and you need to maintain a counter or accumulator that persists across recursive calls, you can use a nonlocal variable.

```
1 def recursive_sum(n):
2     total = 0
3
4     def helper(x):
5         nonlocal total
6         if x == 0:
7             return
8         total += x
9         helper(x - 1)
10
11    helper(n)
12    return total
13
14 print(recursive_sum(5))  # Output: 15 (5 + 4 + 3 + 2 + 1)
15
```

```
→ 15
```

```
1 def f():
2     print('inside f')
3     def g():
4         print('inside g')
```

```
5         f()      # f() is called from function g and g is ca  
6     g()
```

```
1 f() # infinite loop, ffns calling each other, there is limit
```

```
1 def g(x):  
2     def h(): # inner function  
3         x = 'abc'  
4     x = x + 1  
5     print('in g(x): x =', x)  
6     h()  
7     return x  
8  
9 x = 3  
10 z = g(x)  
11 print(z)  
12
```

```
→ in g(x): x = 4  
    4
```

```
1 def g(x):  
2     def h(x):  
3         x = x+1  
4         print("in h(x): x = ", x)  
5     x = x + 1  
6     print('in g(x): x = ', x)  
7     h(x)  
8     return x  
9  
10 x = 3  
11 z = g(x)  
12 print('in main program scope: x = ', x)  
13 print('in main program scope: z = ', z)
```

```
→ in g(x): x = 4  
    in h(x): x = 5  
    in main program scope: x = 3  
    in main program scope: z = 4
```

## 1 Start coding or generate with AI.

Python Library Functions Python provides some built-in functions that can be directly used in our program.

We don't need to create the function, we just need to call them.

Some Python library functions are:

`print()` - prints the string inside the quotation marks   `sqrt()` - returns the square root of a number  
`pow()` - returns the power of a number These library functions are defined inside the module. And to use them, we must include the module inside our program.

For example, `sqrt()` is defined inside the `math` module. Here, we imported a `math` module to use the library functions `sqrt()` and `pow()`.

**User Defined Function Vs Standard Library Functions** In Python, functions are divided into two categories: user-defined functions and standard library functions. These two differ in several ways:

**User-Defined Functions** These are the functions we create ourselves. They're like our custom tools, designed for specific tasks we have in mind. They're not part of Python's standard toolbox, which means we have the freedom to tailor them exactly to our needs, adding a personal touch to our code.

**Standard Library Functions** Think of these as Python's pre-packaged gifts. They come built-in with Python, ready to use. These functions cover a wide range of common tasks such as mathematics, file operations, working with strings, etc. They've been tried and tested by the Python community, ensuring efficiency and reliability.

```
1 import math
2
3 # sqrt computes the square root
4 square_root = math.sqrt(4)
5
6 print("Square Root of 4 is",square_root)
7
8 # pow() computes the power
9 power = pow(2, 3)
10
11 print("2 to the power 3 is",power)
```

→ Square Root of 4 is 2.0  
2 to the power 3 is 8

```
1 def counter():
2     count = 0
3     print('hello message')
4
5     def increment():
6         nonlocal count
7         print('I am here')
8         count += 1
9         return count
10
11     return increment
12
13 counter1 = counter()
14 print(counter1()) # Output: 1
15 print(counter1()) # Output: 2
16
```

```
→ hello message
I am here
1
I am here
2
```

image.png

```
1 ## everything is python is object. Integers, strings are obj
2 def f(num):
3     return num*num
4
```

```
1 f(2)
```

```
→ 4
```

```
1 f(4)
```

```
→ 16
```

```
1 a=2
2 b=a
```

```
1 #Function ALiasing
2 #in Python, we can give another name of the function.
3 #For the existing function, we can give another name, which
4 x=f # just like we can do with integers, we can do the same
```

```
1 x(2) # you can use x like f, becos x is now a function
```

→ 4

```
1 del f # In python, we can delete function
```

```
1 print(type(x)) # function is behaving like a datatype
```

→ <class 'function'>

```
1 l=[2,4,5,5]
2 l
```

→ [2, 4, 5, 5]

```
1 # if integer can be stored in list, then function can be sto
2 l=[2,4,5,5,x] # the last item of list is object x of class
3 print(l)
```

→ [2, 4, 5, 5, <function f at 0x00000246C5FE7A60>]

```
1 l[-1](4) # -1 mean from reverse order , which is x and x is
2
3
```

→ 16

```
1 l[-1](3) # we can use function by l[-1] and pass the argume
```

→ 9

```
1 l=[2,4,5,5,x(5)] # In python, functions behaves like other d
2 l
```

→ [2, 4, 5, 5, 25]

you can pass function as an argument/input to another function

return of a function can be another function

```
1 def func_a():
2     print("inside function a")
3 def func_c(z):
4     print("inside function c")
5     return z()                      # here func_a is called, it will
6
7 print(func_c(func_a)) # func_c() is called, with func_a is
```

```
→ inside function c
inside function a
None
```

```
1 def func_a():
2     print("inside function a")
3 def func_c(z):
4     print("inside function c")
5     return z                      # here func_a is called, it will re-
6
7 print(func_c(func_a()))
```

```
1 def f():
2     def x(a,b):
3         return a+b
4     return x
5 # here u are performing nested calling, taking another call f
6 val= f()(4,5) # f is called and it is returning function x a-
7 print('value is ', val)
```

```
→ value is 9
```

Benefits of using functions:

1. Every code is self contained and used to break up the entire logic into pieces (code modularity).
2. works on a philosophy, write once use forever (code reusability)
3. code is organized and coherent (code readability)  
=> if you are building application, then you will write functions for login, registration and

dealing with database, so your application will be modular. You will be calling the same functions repeatedly with different arguments.

=> if your code crashes, you will see the code in a specific function rather than entire application, it becomes easy to debug the code

1 Start coding or generate with AI.

## ✓ Notebook: 5) Iterators.ipynb

### ✓ What is an Iteration

Iteration is a general term for taking each item of something, one after another. Any time you use a loop, explicit or implicit, to go over a group of items, that is iteration.

Iteration: Definition: Iteration is the process of going through a sequence (e.g., a list, tuple, string, or any other data structure) one element at a time.

Purpose: It allows you to perform operations on each element of a sequence, such as printing, modifying, or using the elements in computations.

Examples: Using a for loop to go through each element in a list or a while loop to iterate through a sequence of numbers.

1 difference between iteration, iterable and iterator

2 Iteration: The process of traversing items of object one by one

3 Iterable: It is an object upon which iteration can be performed

4 Iterator: It is an object with the help of which iteration is performed

1 # Example

2 num = [1,2,3,4,5,6, ]

3

4 for i in num:

5 print(i)

→ 1  
2  
3  
4  
5  
6

## ✓ What is Iterator

An Iterator is an object that allows the programmer to traverse through a sequence of data without having to store the entire data in the memory

```
1 # Example
2 L = [x for x in range(1,10000)] # list will store all numbers
3
4 #for i in L:                      # loop will go over the list and
5   # print(i*2,end=' ')
6
7 import sys # sys module
8
9 #print(sys.getsizeof(L)) # memory(in bytes) occupied by list
10 size_in_bytes=sys.getsizeof(L)
11
12 # Convert the size to kilobytes and megabytes
13 size_in_kb = size_in_bytes / 1024
14 size_in_mb = size_in_bytes / (1024 * 1024)
15
16 # Print the size in bytes, kilobytes, and megabytes
17 print()
18 print(f"Size of the list in bytes: {size_in_bytes} bytes")
19 print(f"Size of the list in kilobytes: {size_in_kb:.2f} KB")
20 print(f"Size of the list in megabytes: {size_in_mb:.5f} MB")
21
22 #print(sys.getsizeof(L)/64)
23
```



Size of the list in bytes: 85176 bytes  
Size of the list in kilobytes: 83.18 KB  
Size of the list in megabytes: 0.08123 MB

```
1 print(85176/9999)
2 print(9999*8.51845)
```



8.518451845184519  
85175.98155

The range function in Python creates an iterator that efficiently generates a sequence of numbers on demand. When you use the range function, it does not pre-allocate memory for the entire sequence. Instead, the range iterator only stores the start, stop, and step values for the range and calculates each item in the sequence as needed.

When you iterate over a range object, it does not load all the numbers into memory at once. Instead, it computes each number on demand and yields it. This is a form of lazy evaluation, where items are generated only when needed.

The range iterator does not load items from secondary storage (e.g., a hard drive or SSD). Instead, it uses arithmetic operations based on the start, stop, and step values to calculate each number in the sequence. Since the range function does not store the entire sequence in memory, it uses very little memory compared to a list.

When you use `sys.getsizeof(x)`, it measures the memory used by the range object itself, which is relatively small because it only stores a few integer values (the start, stop, and step values). It does not include the memory that would be used to store the entire sequence.

```
1 x = range(1,10000000)
2
3 #for i in x:
4     # print(i*2,end=' ')
5 print(sys.getsizeof(x))
6 print(sys.getsizeof(x)/1024)
7 print(sys.getsizeof(x)/(1024 *1024))
```

```
→ 48
0.046875
4.57763671875e-05
```

```
1 x = range(1,1000000000)
2
3 #for i in x:
4     #print(i*2)
5 print(sys.getsizeof(x))
6 print(sys.getsizeof(x)/1024)
```

```
→ 48
0.046875
```

## ❖ What is Iterable

Iterable is an object, which one can iterate over

It generates an Iterator when passed to iter() method. When you run a loop over an object to get its elements one by one, so that object is called iterable. list and range objects are iterables, becos u can run loop over the list and range object

```
1  
2 list1 = [1,2,3,4,5,7]  
3 print(type(list1))  
4 listiter= iter(list1)  
5 type(listiter)
```

```
→ <class 'list'>  
list_iterator
```

```
1 # Example  
2  
3 dic = {'name':'Asim','fname':'gul rahman'}  
4 for i in dic:  
5     print(dic[i])  
6 print(type(dic))  
7  
8
```

```
→ Asim  
gul rahman  
<class 'dict'>
```

```
1 # L is an iterable  
2 diciter=iter(dic)  
3 print(type(diciter))  
4  
5 # iter(L) --> iterator
```

```
→ <class 'dict_keyiterator'>
```

```
1 for x in range(1,10):  
2     print(x, end=" ")
```

## ▼ Point to remember

- Every **Iterator** is also an **Iterable**
- Not all **Iterables** are **Iterators**
- list is iterable but it is not iterator because list stores all elements in memory at once while Iterators generate each element on demand, one at a time, and do not store the entire sequence in memory. This makes iterators more memory-efficient, especially when dealing with large datasets. Iteration ) can be performed over iterator object or loop can be run over iterator object

1 List:

2 A list is an iterable, which means it can be used to create  
3 Lists store all elements in memory at once. This means that  
4 Lists are commonly used when you need to store and access al  
5 Iterator:

6 An iterator is an object that represents a stream of data an  
7 Iterators generate each element on demand, one at a time, an  
8 Iterators provide two key methods: `__iter__()` and `__next__()`  
9 In summary, while both lists and iterators allow you to iter

## ▼ Trick

- Every Iterable has an **iter function**
- Every Iterator has both **iter function** as well as a **next function**

1 # i have integer object, is it iterable, if loop can be run  
2 #otherwise not  
3 # Two ways to check whether object is iterable or not: for l  
4 a = 2  
5 a  
6  
7 for i in a:  
8 print(i)  
9  
10



TypeError

```
Cell In[10], line 7
  4 a = 2
  5 a
----> 7 for i in a:
     8     print(i)
```

Traceback (most recent call last)

```
TypeError: 'int' object is not iterable
```

```
1 dir(a)
```



```
['__abs__',  
 '__add__',  
 '__and__',  
 '__bool__',  
 '__ceil__',  
 '__class__',  
 '__delattr__',  
 '__dir__',  
 '__divmod__',  
 '__doc__',  
 '__eq__',  
 '__float__',  
 '__floor__',  
 '__floordiv__',  
 '__format__',  
 '__ge__',  
 '__getattribute__',  
 '__getnewargs__',  
 '__getstate__',  
 '__gt__',  
 '__hash__',  
 '__index__',  
 '__init__',  
 '__init_subclass__',  
 '__int__',  
 '__invert__',  
 '__le__',  
 '__lshift__',  
 '__lt__',  
 '__mod__',  
 '__mul__',  
 '__ne__',  
 '__neg__',  
 '__new__',  
 '__or__',  
 '__pos__',  
 '__pow__',  
 '__radd__',  
 '__rand__',  
 '__rdivmod__',  
 '__reduce__'
```

```
'__reduce_ex__',
'__repr__',
'__rfloordiv__',
'__rlshift__',
'__rmod__',
'__rmul__',
'__ror__',
'__round__',
'__rpow__',
'__rrshift__',
'__rshift__',
'__rsub__',
'__rtruediv__',
'__rxor__',
'__setattr__',
'__sizeof__',
'__str__'
```

```
1 str = 'Muazzam'
2 str
3
4 for i in str:
5     print(i)
```

```
→ M
 u
 a
 z
 z
 a
 m
```

```
1 dir(str)
```

```
→ [ '__add__',
  '__class__',
  '__contains__',
  '__delattr__',
  '__dir__',
  '__doc__',
  '__eq__',
  '__format__',
  '__ge__',
  '__getattribute__',
  '__getitem__',
  '__getnewargs__',
  '__getstate__',
  '__gt__',
  '__hash__',
  '__init__',
  '__init_subclass__',
  '__iter__',
  '__le__']
```

```
'__len__',
'__lt__',
'__mod__',
'__mul__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__rmod__',
'__rmul__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'capitalize',
'casefold',
'center',
'count',
'encode',
'endswith',
'expandtabs',
'find',
'format',
'format_map',
'index',
'isalnum',
'isalpha',
'isascii',
'isdecimal',
'isdigit',
'isidentifier',
'islower',
'isnumeric',
'isprintable',
'isspace',
'istitle',
'isupper',
'join',
```

```
1 dir(a) # another way (without for loop)to check whether python
2           # in a list of methods, if we find iter method, then
```

```
→ [ '__abs__',
  '__add__',
  '__and__',
  '__bool__',
  '__ceil__',
  '__class__',
  '__delattr__',
  '__dir__',
  '__divmod__',
  '__doc__',
  '__eq__',
  '__float__',
```

```
'__floor__',
 '__floordiv__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getnewargs__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__index__',
 '__init__',
 '__init_subclass__',
 '__int__',
 '__invert__',
 '__le__',
 '__lshift__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__neg__',
 '__new__',
 '__or__',
 '__pos__',
 '__pow__',
 '__radd__',
 '__rand__',
 '__rdivmod__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rfloordiv__',
 '__rlshift__',
 '__rmod__',
 '__rmul__',
 '__ror__',
 '__round__',
 '__rpow__',
 '__rrshift__',
 '__rshift__',
 '__rsub__',
 '__rtruediv__',
 '__rxor__',
 '__setattr__',
 '__sizeof__',
 '__str__',
```

1 Tp= (2,3,4)

2 dir(Tp)

→ [ '\_\_add\_\_',
 '\_\_class\_\_',
 '\_\_class\_getitem\_\_',
 '\_\_contains\_\_',
 '\_\_delattr\_\_',

```
'__dir__',  
'__doc__',  
'__eq__',  
'__format__',  
'__ge__',  
'__getattribute__',  
'__getitem__',  
'__getnewargs__',  
'__getstate__',  
'__gt__',  
'__hash__',  
'__init__',  
'__init_subclass__',  
'__iter__',  
'__le__',  
'__len__',  
'__lt__',  
'__mul__',  
'__ne__',  
'__new__',  
'__reduce__',  
'__reduce_ex__',  
'__repr__',  
'__rmul__',  
'__setattr__',  
'__sizeof__',  
'__str__',  
'__subclasshook__',  
'count',  
'index']
```

```
1 s= {2,3,4}  
2 dir(s)
```

```
[ '_and_',
  '_class_',
  '_class_getitem_',
  '_contains_',
  '_delattr_',
  '_dir_',
  '_doc_',
  '_eq_',
  '_format_',
  '_ge_',
  '_getattribute_',
  '_getstate_',
  '_gt_',
  '_hash_',
  '_iand_',
  '_init_',
  '_init_subclass_',
  '_ior_',
  '_isub_',
  '_iter_'],
```

```
'__ixor__',
'__le__',
'__len__',
'__lt__',
'__ne__',
'__new__',
'__or__',
'__rand__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__ror__',
'__rsub__',
'__rxor__',
'__setattr__',
'__sizeof__',
'__str__',
'__sub__',
'__subclasshook__',
'__xor__',
'add',
'clear',
'copy',
'difference',
'difference_update',
'discard',
'intersection',
'intersection_update',
'isdisjoint',
'issubset',
'issuperset',
'pop',
'remove',
'symmetric_difference',
'symmetric_difference_update',
'union',
'update']
```

```
1 T = {1:2,3:4}
2 dir(T)
```

```
→ ['__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getstate__',
```

```
'__gt__',  
'__hash__',  
'__init__',  
'__init_subclass__',  
'__ior__',  
'__iter__',  
'__le__',  
'__len__',  
'__lt__',  
'__ne__',  
'__new__',  
'__or__',  
'__reduce__',  
'__reduce_ex__',  
'__repr__',  
'__reversed__',  
'__ror__',  
'__setattr__',  
'__setitem__',  
'__sizeof__',  
'__str__',  
'__subclasshook__',  
'clear',  
'copy',  
'fromkeys',  
'get',  
'items',  
'keys',  
'pop',  
'popitem',  
'setdefault',  
'update',  
'values']
```

```
1 L = [1,2,3]
```

```
2 dir(L)
```

```
3
```

```
→ [ '__add__',  
  '__class__',  
  '__class_getitem__',  
  '__contains__',  
  '__delattr__',  
  '__delitem__',  
  '__dir__',  
  '__doc__',  
  '__eq__',  
  '__format__',  
  '__ge__',  
  '__getattribute__',  
  '__getitem__',  
  '__getstate__',  
  '__gt__',  
  '__hash__'
```

```
'__iadd__',
'__imul__',
'__init__',
'__init_subclass__',
'__iter__',
'__le__',
'__len__',
'__lt__',
'__mul__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__reversed__',
'__rmul__',
'__setattr__',
'__setitem__',
'__sizeof__',
'__str__',
'__subclasshook__',
'append',
'clear',
'copy',
'count',
'extend',
'index',
'insert',
'pop',
'remove',
'reverse',
'sort']
```

```
1 # to check whether the object is iterator or not, Pass the o
2 # L is iterator else L is not an iterator
3 L=[3,5,6,7,9]
4 iter_L = iter(L) # iterator can be generated from iterable
5 dir(iter_L)
6 # iter_L is an iterator
```

```
→ ['__class__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__rmul__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__']
```

```
'__init_subclass__',
'__iter__',
 '__le__',
 '__length_hint__',
 '__lt__',
 '__ne__',
 '__new__',
 '__next__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__setstate__',
 '__sizeof__',
 '__str__',
 '__subclasshook__']
```

### **difference between iteration, iterable and iterator**

Iteration: The process of traversing items of object one by one

Iterable: It is an object upon which iteration can be performed or loop can be run

Iterator: It is an object with the help of which iteration is performed.

## ▼ Understanding how for loop works

```
1 num = [1,2,3]
2
3 for i in num:
4     print(i)
```

```
→ 1
  2
  3
```

```
1 num = [1,2,3,4] # define iterable
2
3 # fetch the iterator
4 iter_num = iter(num) # first the for loop fetches iterator f
5
```

```
1 print(next(iter_num))
```

```
→ 1
```

```
1 print(next(iter_num))
```

→ 2

```
1 print(next(iter_num))
```

→ 3

```
1 print(next(iter_num))
```

→ 4

```
1 print(next(iter_num))
```

→

```
StopIteration                                     Traceback (most recent call last)
Cell In[22], line 1
----> 1 print(next(iter_num))
```

StopIteration:

```
1
2 # the next function of the iterator is called repeatedly to
3 # step2 --> next # Every iterator has next function showing
4 #since iterator is at 0th position, it will print 1
5 print(next(iter_num))
6 print(next(iter_num))
7 print(next(iter_num))
8 print(next(iter_num))
```

→ 1  
2  
3  
4

```
StopIteration                                     Traceback (most recent call last)
Cell In[5], line 12
    10 print(next(iter_num))
    11 print(next(iter_num))
--> 12 print(next(iter_num))
```

StopIteration:

## ✓ Making our own for loop

```
1 def my_for_loop(iterable):
2
3     iterator = iter(iterable)
4
5     while True:
6
7         try:
8             print(next(iterator))
9         except StopIteration:
10            break
```

```
1 a = [1,2,3,4,5,6,8,9]
2 my_for_loop(a)
```

```
→ 1
  2
  3
  4
  5
  6
  8
  9
```

```
1 dic = {'name':'Asim','fname':'gul rahman', 'add':'kohat'}
2 my_for_loop(dic)
```

```
→ name
  fname
  add
```

```
1 b = range(1,11)
2 my_for_loop(b)
```

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

```
1 C = range(1,11)
2 dir(C)
```

→ [ '\_\_bool\_\_',
 '\_\_class\_\_',
 '\_\_contains\_\_',
 '\_\_delattr\_\_',
 '\_\_dir\_\_',
 '\_\_doc\_\_',
 '\_\_eq\_\_',
 '\_\_format\_\_',
 '\_\_ge\_\_',
 '\_\_getattribute\_\_',
 '\_\_getitem\_\_',
 '\_\_getstate\_\_',
 '\_\_gt\_\_',
 '\_\_hash\_\_',
 '\_\_init\_\_',
 '\_\_init\_subclass\_\_',
 '\_\_iter\_\_',
 '\_\_le\_\_',
 '\_\_len\_\_',
 '\_\_lt\_\_',
 '\_\_ne\_\_',
 '\_\_new\_\_',
 '\_\_reduce\_\_',
 '\_\_reduce\_ex\_\_',
 '\_\_repr\_\_',
 '\_\_reversed\_\_',
 '\_\_setattr\_\_',
 '\_\_sizeof\_\_',
 '\_\_str\_\_',
 '\_\_subclasshook\_\_',
 'count',
 'index',
 'start',
 'step',
 'stop']

range(1, 11) is an iterable because you can loop over it.

But it is not an iterator because it doesn't have a **next()** method directly.

To get an iterator from it, you do this:

```
1 it = iter(C)
2 #Now it is an iterator, and you can use:
3 next(it)
```

→ 1

```
1 next(it)
```

→ 2

```
1 #You can explicitly convert the iterable C into an iterator
2 #In this case, iter(C) explicitly turns the range object into
3 #but the for loop will still handle the iteration process for you
4 C = range(1, 11)
5
6 for i in iter(C):
7     print(i)
8
```

→ 1  
2  
3  
4  
5  
6  
7  
8  
9  
10

The for loop in Python internally calls `iter()` on the iterable before starting the iteration. So when you use a for loop with an iterable like `range`, Python automatically converts it into an iterator and handles the iteration process for you.

Here's how it works internally:

**Iterable:** An object like `range(1, 11)` is an iterable.

**iter():** The for loop calls `iter(C)` behind the scenes to obtain an iterator.

**Iterator:** The iterator is then used to fetch elements one by one using `next()` until all elements are exhausted.

```
1 #you can definitely use a for loop over the range object because
2 C = range(1, 11)
3
4 for num in C:
```

```
5     print(num)
6
```

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



```
1
2
3 c = (1,2,3)
4 d = {1,2,3}
5 e = {'name':'Atif','addres':'Peshawar'}
6
7 my_for_loop(e)
8 #my_for_loop(b)
```

```
→ name
    addres
```

## ▼ A confusing point

1 Start coding or generate with AI.

```
1 num = [1,2,3]
2 iter_obj = iter(num) # you get iterator object, when run ite
3
4 print(id(iter_obj),'Address of iterator 1')
5
6 iter_obj2 = iter(iter_obj) # get another iterator object whe
7 print(id(iter_obj2),'Address of iterator 2')
```

```
→ 1793081604368 Address of iterator 1
  1793081604368 Address of iterator 2
```

## ▼ Let's create our own range() function

here i create an object that will behave like range function. we will create two classes

```
1 #when range function is called, we give two inputs: start and end
2 # Every iterable object has iter() magic method, so add this
3
4 class mera_range: # iterable class
5
6     def __init__(self,start,end):
7         self.start = start                 # user supplied value for start
8         self.end = end
9
10    def __iter__(self):
11        return mera_range_iterator(self)
```

```
1 class mera_range_iterator: # iterator class, its work is to iterate
2
3     def __init__(self,iterable_obj):
4         self.iterable = iterable_obj
5
6     def __iter__(self):
7         return self
8
9     def __next__(self):
10
11         if self.iterable.start >= self.iterable.end:
12             raise StopIteration
13
14         current = self.iterable.start
15         self.iterable.start+=1
16         return current
```

```
1 x = mera_range(1,11)
```

```
1 type(x)
```

```
→ __main__.mera_range
```

```
1 iter(x)
```

```
→ <__main__.mera_range_iterator at 0x2130fd362b0>
```

```
1 Start coding or generate with AI.
```

## ▼ Notebook: 6) List Comprehension.ipynb

Python List Comprehension List comprehension offers a concise way to create a new list based on the values of an existing list.

```
1 import math
```

```
1 math.sqrt(25)
```

```
→ 5.0
```

```
1 #Suppose we have a list of numbers and we desire to create a
2 numbers = [1, 2, 3, 4]
3
4 # list comprehension to create new list
5
6 doubled_numbers = [num ** 2 for num in numbers] #doubled_nu
7
8 print(doubled_numbers)
```

```
→ [1, 4, 9, 16]
```

```
1 #Suppose we have a list of numbers and we desire to create a
2 numbers = [16, 25, 64, 49]
3
4 # list comprehension to create new list
5
6 sqrt = [math.sqrt(num) for num in numbers] #doubled_numbers
```

```
7  
8 print(sqrt)
```

```
→ [4.0, 5.0, 8.0, 7.0]
```

for Loop vs. List Comprehension List comprehension makes the code cleaner and more concise than for loop.

Let's write a program to print the square of each list element using both for loop and list comprehension.

```
1 #for loop  
2 numbers = [1, 2, 3, 4, 5]  
3 square_numbers = []  
4  
5 # for loop to square each elements  
6 for num in numbers:  
7     square_numbers.append(num * 2)  
8  
9 print(square_numbers)  
10  
11 # Output: [1, 4, 9, 16, 25]
```

```
→ [1, 4, 9, 16, 25]
```

```
1 #List Comprehension  
2 numbers = [1, 2, 3, 4, 5]  
3  
4 # create a new list using list comprehension  
5 square_numbers = [num * num for num in numbers]  
6  
7 print(square_numbers)  
8  
9 # Output: [1, 4, 9, 16, 25]
```

```
→ [1, 4, 9, 16, 25]
```

## Conditionals in List Comprehension

List comprehensions can utilize conditional statements like if...else to filter existing lists.

Let's see an example of an if statement with list comprehension.

```
1 for x in range(1,100):  
2     print(x)
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45
```

```
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58
```

```
1 list(range(1,100))  
2
```

```
3 [1,  
4 2,  
5 3,  
6 4,  
7 5,  
8 6,  
9 7,  
10 8,  
11 9,  
12 10,  
13 11,  
14 12,  
15 13,  
16 14,  
17 15,  
18 16,  
19 17,  
20 18,  
21 19,  
22 20,  
23 21,  
24 22,  
25 23,  
26 24,  
27 25,  
28 26,  
29 27,  
30 28,  
31 29,  
32 30,  
33 31,  
34 32,  
35 33,  
36 34,  
37 35,  
38 36,
```

```
39,  
40,  
41,  
42,  
43,  
44,  
45,  
46,  
47,  
48,  
49,  
50,  
51,  
52,  
53,  
54,  
55,  
56,  
57,  
∞
```

```
1 # filtering even numbers from a list  
2 #Here, list comprehension checks if the number from range(1,  
3 even_numbers = [num for num in range(1, 51) if num % 2 == 0  
4  
5 print(even_numbers)  
6  
7 # Output: [2, 4, 6, 8]
```

```
→ [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46,
```

```
1 lst=[x for x in range(1,31) if x%2!=0]  
2 lst  
3
```

```
→ [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
```

```
1 lst=[x for x in range(1,31) if x%2==0]  
2 lst
```

```
→ [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]
```

```
1 lst=[x for x in range(1,31) if x%3==0]  
2 lst
```

```
→ [3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
```

```
1 # filtering odd numbers from a list
2 even_numbers = [num for num in range(1, 51) if num % 2 != 0]
3
4 print(even_numbers)
```

```
→ [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45,
```



### if...else With List Comprehension

Let's use if...else with list comprehension to find even and odd numbers.

```
1 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2
3 # find even and odd numbers
4 even_odd_list = ["Even" if i % 2 == 0 else "Odd" for i in numbers]
5
6 print(even_odd_list)
```

```
→ ['Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even']
```

```
1 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2
3 # find even and odd numbers
4 even_odd_list = ["Even" if i % 2 == 0 and i % 4 == 0 else "Odd"]
5
6 print(even_odd_list)
```

```
→ ['Odd', 'Odd', 'Odd', 'Even', 'Odd', 'Odd', 'Odd', 'Even', 'Odd', 'Odd']
```

```
1 #Extract vowels
2 text = "List comprehension is powerful!"
3 vowels = [char for char in text if char.lower() in "aeiou"]
4 print(vowels)
5
```

```
→ ['i', 'o', 'e', 'e', 'i', 'o', 'i', 'o', 'e', 'u']
```

```
1 text = "List comprehension is powerful!"
2 for char in text:
```

```
3     print(type(char), end=',')  
→ <class 'str'>, <class 'str'>, <class 'str'>, <class 'str'>, <class 'str'>, <class 'str'>, <class 'str'>
```



```
1 'e' in 'eiou'
```

```
→ True
```

```
1 #flatten a list  
2 nested_list = [[1, 2, 3], [4, 5], [6, 7, 8]]  
3 flattened_list = [num for sublist in nested_list for num in  
4 print(flattened_list)  
5 #[1, 2, 3, 4, 5, 6, 7, 8]  
6
```

```
1 len("hi students of low GPA".split())
```

```
→ 5
```

```
1 #Filter and Modify Strings  
2 sentences = ["hi students of low GPA", "hello engineers", "A  
3 filtered_upper = [sent for sent in sentences if len(sent.spl  
4 print(filtered_upper)  
5
```

```
→ ['hi students of low GPA', 'AI in boom']
```

```
1 #Find Common Elements in Two Lists  
2 list1 = [1, 2, 3, 4, 5]  
3 list2 = [3, 4, 5, 6, 7]  
4 common_elements = [num for num in list1 if num in list2]  
5 print(common_elements)  
6
```

```
1 sentence = "Python is fun"
```

```
2 sentence.split()
```

```
→ ['Python', 'is', 'fun']
```

```
1 #Reverse Words in a Sentence
2 sentence = "Python is fun"
3 reversed_words = [word[::-1] for word in sentence.split()]
4 print(reversed_words)
5
```

→ ['nohtyP', 'si', 'nuf']

```
1 #Create a Dictionary from Two Lists
2 keys = ["name", "age", "city"]
3 values = ["Alice", 25, "New York"]
4 dictionary = {keys[i]: values[i] for i in range(len(keys))} 
5 print(dictionary)
6
```

→ {'name': 'Alice', 'age': 25, 'city': 'New York'}

### Nested if With List Comprehension

Let's use nested if with list comprehension to find even numbers that are divisible by 5.

```
1 # find even numbers that are divisible by 5
2 #Here, list comprehension checks two conditions:
3 #if y is divisible by 2 or not.
4 #if yes, is y divisible by 5 or not.
5 #If y satisfies both conditions, the number appends to num_1
6
7 num_list = [y for y in range(100) if (y % 2 == 0) and (y %
8
9 print(num_list)
```

→ [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]

### Example: List Comprehension with String

We can also use list comprehension with iterables other than lists.

```
1 vowels='aeiou'
2 'y' in vowels
```

→ False

```
1 for i in 'vowels':  
2     print(i, end=' ')
```

→ vowels

```
1 list=[]  
2 zistr= 'ziaullah'  
3 list.append(zistr)  
4 list.extend('Ashiq')  
5 print(list)
```

→ ['ziaullah', 'A', 's', 'h', 'i', 'q']

```
1 lst=['ziaullah']  
2 lst[0]  
3
```

→ 'ziaullah'

```
1 #Here, we used list comprehension to find vowels in the string  
2 sentence = "Python is my favourite language"  
3 vowels = "aeiou"  
4  
5 # find vowel in the string "Python"  
6 result = [char for char in sentence if char in vowels]  
7  
8 print(result)  
9 print(len(result))  
10  
11 # Output: ['o']
```

→ ['o', 'i', 'a', 'o', 'u', 'i', 'e', 'a', 'u', 'a', 'e']  
11

## Nested List Comprehension

We can also use nested loops in list comprehension. Let's write code to compute a multiplication table.

```
1 multiplication = [[i * j for j in range(1, 11)] for i in ran  
2
```

```
3 print(multiplication)
```

```
→ [[2, 4, 6, 8, 10, 12, 14, 16, 18, 20], [3, 6, 9, 12, 15, 18, 21, 24, 27, 30], [4, 8, 12,
```

Here, the nested for loop generates the same output as the nested list comprehension. We can see that the code with list comprehension is much cleaner and concise.

```
1 #Let's see the equivalent code using nested for loop.  
2 #Equivalent Nested for Loop for multiplication table  
3 multiplication = []  
4  
5 for i in range(2, 5):  
6     row = []  
7     for j in range(1, 6):  
8         row.append(i * j)  
9     multiplication.append(row)  
10  
11 print(multiplication)
```

```
→ [[2, 4, 6, 8, 10], [3, 6, 9, 12, 15], [4, 8, 12, 16, 20]]
```

1 Start coding or generate with AI.

## List Comprehensions vs Lambda Functions

1. Along with list comprehensions, we also use lambda functions to work with lists.
2. While list comprehension is commonly used for filtering a list based on some conditions, lambda functions are commonly used with functions like map() and filter().
3. They are used for complex operations or when an anonymous function is required.

Let's look at an example.

```
1 numbers = [5, 6, 7, 8, 9]  
2  
3 # create a new list using a lambda function  
4 square_numbers = list(map(lambda num : num**2 , numbers))
```

```
5  
6 print(square_numbers)
```

→

```
TypeError Traceback (most recent call last)
Cell In[28], line 4
  1 numbers = [5, 6, 7, 8, 9]
  2 # create a new list using a Lambda function
----> 4 square_numbers = list(map(lambda num : num**2 , numbers))
  6 print(square_numbers)

TypeError: 'list' object is not callable
```

```
1 #We can achieve the same result using list comprehension by:
2
3 # create a new list using list comprehension
4 square_numbers = [num ** 2 for num in numbers]
5 print(square_numbers)
```

→ [25, 36, 49, 64, 81]

If we compare the two codes, list comprehension is straightforward and simpler to read and understand.

So unless we need to perform complex operations, we can stick to list comprehension.

1 Start coding or generate with AI.

## ▼ Notebook: 7) Lamda functions-Part1.ipynb

Lambda functions are also known as anonymous functions. these functions are written in one line. They do not span multi-line. they are quite useful in some scenarios

syntax of lambda functions:

lambda input: expression

lambda is a keyword

### Python Lambda/Anonymous Function

In Python, a lambda function is a special type of function without the function name. For example,

lambda : print('Hello World') Here, we have created a lambda function that prints 'Hello World'.



```
1 greet = lambda : print('Eid ul fitr Mubarak')
2
3 #Here, we have defined a lambda function and assigned it to
4
5 #To execute this lambda function, we need to call it. Here's
6
7 # call the lambda
8 greet()
9
10 #The lambda function above simply prints the text 'Hello Wor
11
12 #Note: This lambda function doesn't have any argument.
```

→ Eid ul fitr Mubarak

### Python lambda Function with an Argument

Similar to normal functions, a lambda function can also accept arguments. For example,

```
1 # lambda that accepts one argument
2 greet_user = lambda name : print(f'eid mubarak, {name}')
3
```

```
4 # lambda call
5 greet_user('Naseer')
6
7 # Output: Hey there, Atif
8 #Here, we have passed a string value 'Atif' to our lambda fu
9
10 #Finally, the statement inside the lambda function is execut
```

→ eid mubarak, Naseer

```
1 def square(x):
2     return x*x
3
4 square(5)
```

→ 25

```
1 print(type(square))
```

→ <class 'function'>

```
1 square = lambda x : x**3
2
3 # lambda call
4 square(5)
```

→ 125

```
1 print(type(square))
```

→ <class 'function'>

```
1 #create a lamda function that calcualte the square of the gi
2 # Define a lambda function to square a number
3 square = lambda x: x ** 2
4
5 # Use the lambda function
6 result = square(5) # Result: 16
7 print(result)
8
```

→ 25

```
1 lst= [2,4,5,6]
2 doublelst = lambda x: [i * 2 for i in lst]
3
4 # Use the lambda function
5 result = doublelst(lst) # Result: 16
6 print(result)
```

→ [4, 8, 10, 12]

```
1 #create a lamda function that take two numbers as input and
2 s1=lambda a,b: a+b           # here whole lambda fuction is re
3
4 s1(3,4)
```

→ 7

1 Start coding or generate with AI.

```
1 #create a lamda function that take a list of numbers as input
2 lst=[2,4,5,6,7]
3
4 result= lambda x: sum(x)           # here whole lambda fuction
5
6 result(lst)
```

→ 24

1 print(type(result))

→ <class 'function'>

1 print(type(x))

```
→ -----
NameError                                Traceback (most recent call last)
Cell In[6], line 1
----> 1 print(type(x))

NameError: name 'x' is not defined
```

## Difference between lambda function and normal function

1. lambda function has no return value, In fact, whole lambda function is returned. see above the type of x is a function. while function can return a value.
2. lambda function is written in a single line.
3. lambda functions are not used for code reusability while normal functions are used for code reusability. lambda functions are developed for one time use.
4. lambda function has no name while normal function has name like def square(): why lambda functions are used> In normal scenarios, we use normal functions. lambda functions are used along with higher order functions. Higher order functions require another function as input perform their job. or higher order function is a function that is returning another function. Lambda functions are used alongside higher-order functions. Higher-order functions are functions that accept other functions as arguments to perform their tasks

1. Return Value of a Lambda Function Always returns the result of the expression it evaluates. No need to write return – the expression's value is returned implicitly
2. Return Value of a Regular Function You must use the return keyword explicitly to return a value.

If you don't use return, the function returns None by default.

Python is not just about writing functions that perform simple tasks. It empowers developers with a powerful feature called “higher-order functions.” These functions take other functions as arguments or return functions as results. A higher-order function is a function that can accept one or more functions as arguments or return a function as its result.

A lambda function, also known as an anonymous function, is a small, unnamed function defined using the lambda keyword. It's a quick way to create small, throwaway functions for simple operations. Lambda functions are especially handy when you need a function for a short period and don't want to define a full def function.



1. List Concatenation using + Creates a new list with elements from both lists.

Original lists are not modified.

Slower for very large lists due to the creation of a new list.

```
1 a = [1, 2]
2 b = [3, 4]
3
4 c = a + b  # c = [1, 2, 3, 4]
5 print(c)
6 print(a)    # a is still [1, 2]
7
```

```
→ [1, 2, 3, 4]
[1, 2]
```

2. List extend() Method Modifies the original list in place.

Adds elements from another iterable (like a list) to the end of the list.

More memory-efficient and generally faster than concatenation.

```
1 # create a lambda function that check whether the first char
2 # lambda function can be used in a lot of scenarios
3 c= lambda x: x[0]=='a'
4
5 c('atif')
6
```

```
→ True
```

```
1 def square():
2     print("hello")
3 s=square()
4 print(s)
```

```
→ hello
None
```

```
1 lst1= [3,4,4,4]
2 lst2= [3,4,4,4]
3 result=lst1.extend(lst2)
4 print(result)
5 lst1
```

→ None  
[3, 4, 4, 4, 3, 4, 4, 4]

```
1 type('a')
```

→ str

```
1 # create a lambda function that check whether the first char
2 # lambda function can be used in a lot of scenarios
3 c= lambda x: type(x[0])==str
4
5 c('atif')
```

→ True

```
1 c('banana')
```

→ True

```
1 lst=[1,2]
```

```
1 c(lst)
```

→ False

```
1 # condition with lambda function, create a lambda function t
2 x= lambda n: "Even" if n% 2==0 else "Odd"
3 x(4)
```

→ 'Even'

```
1 x(5)
```

→ 'Odd'

```

1 # I want a function that give me three outputs, 1st output
2 # the second output should be the sum of odd numbers in list
3 # I need three values from the function
4
5 def return_sum(lst):
6     even_sum=0
7     odd_sum=0
8     div3_sum=0
9     for i in lst:
10         if i % 2==0:
11             even_sum= even_sum +i
12
13         if i % 2!=0:
14             odd_sum= odd_sum+i
15
16         if i % 3==0:
17             div3_sum= div3_sum + i
18
19     return (even_sum, odd_sum, div3_sum)
20
21
22 lst= [11,14, 21,23,56, 78,45, 29, 28]
23
24 even_sum, odd_sum, div3_sum= return_sum(lst)
25
26 print("sum of even no is ", even_sum)
27 print('sum of odd no is ', odd_sum)
28 print('sum of nos divisible by 3 is', div3_sum)

```

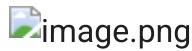
→ sum of even no is 176  
 sum of odd no is 129  
 sum of nos divisible by 3 is 144

## ▼ Notebook: 8) Lamda functions-Part2.ipynb

Lambda functions are also known as anonymous functions. these functions are written in one line. They do not span multi-line. they are quite useful in some scenarios syntax of lambda functions:  
 lambda input: expression lambda is a keyword

## Python Lambda/Anonymous Function

In Python, a lambda function is a special type of function without the function name. For example, `lambda : print('Hello World')` Here, we have created a lambda function that prints 'Hello World'.



```
1 output= lambda: print('python progarammers')
2 output()
```

→ python progarmmers

```
1 greet = lambda : print('Hello World')
2
3 #Here, we have defined a lambda function and assigned it to
4
5 #To execute this lambda function, we need to call it. Here's
6
7 # call the lambda
8 greet()
9
10 #The lambda function above simply prints the text 'Hello Wor
11
12 #Note: This lambda function doesn't have any argument.
```

→ Hello World

## Python lambda Function with an Argument

Similar to normal functions, a lambda function can also accept arguments. For example,

```
1 # lambda that accepts one argument
2 greet_user = lambda name : print('Hey there,', name)
3
4 # lambda call
5 greet_user('Waqas')
6
7 # Output: Hey there, Atif
8 #Here, we have passed a string value 'Delilah' to our lambda
```

```
9
```

```
10 #Finally, the statement inside the lambda function is execut
```

→ Hey there, Waqas

```
1 #create a lamda function that calcualte the square of the gi
2 # Define a lambda function to square a number
3 square = lambda x: x ** 2
4
5 # Use the lambda function
6 result = square(5) # Result: 16
7 print(result)
```

→ 25

```
1 #create a lamda function that calcualte the square of the gi
2 # Define a lambda function to square a number
3 lst=[2,3,4,5,6]
4 square = lambda x: [i ** 2 for i in x]
5
6 # Use the lambda function
7 result = square(lst) # Result: 16
8 print(result)
```

→ [4, 9, 16, 25, 36]

```
1 lst1=[2,3,4,5,6]
2 lst2=[12,13,14,15]
3 #lst3=[22,23,24,25]
4
5 addlist= lambda x1, x2: [ i + j for i,j in zip(x1, x2)]
6 addlist(lst1, lst2)
7
```

→ [14, 16, 18, 20]

```
1 for i, j in zip(lst1, lst2):
2     print(i,j)
```

→ 2 12
3 13

```
4 14  
5 15
```

1 None or [1,2,3,4]

→ [1, 2, 3, 4]

```
1 #create a lamda function that calcualte the square of the gi  
2 # Define a lambda function to square a number  
3 lst1=[2,3,4,5,6]  
4 lst2=[12,13,14,15]  
5 square = lambda x, y: x.extend(y) or x  
6  
7 # Use the lambda function  
8 result = square(lst1, lst2) # Result: 16  
9 print(result)
```

→ None

```
1 lst1=[2,3,4,5,6]  
2 lst2=[12,13,14,15,16]  
3 add = lambda x, y: [i+j for i, j in zip(x,y)]  
4  
5 # Use the lambda function  
6 result = add(lst1, lst2) # Result: 16  
7 print(result)
```

→ [14, 16, 18, 20, 22]

1 zip function is a built in function in python that allows yo  
2 simultaneously, it takes corresponding elements from each lis

```
1 lst1=[2,3,4,5,6]  
2 lst2=[12,13,14,15,16]  
3 add = lambda x, y: [i+j for i, j in zip(x,y)]  
4  
5 # Use the lambda function  
6 result = add(lst1, lst2) # Result: 16  
7 print(result)
```

```
→ [14, 16, 18, 20, 22]
```

```
1 # Define a lambda function to extend a list
2 lst1 = [2, 3, 4, 5, 6]
3 lst2 = [12, 13, 14, 15]
4
5 # Lambda function to extend the list
6 extend_lists = lambda x, y: x.extend(y) or x
7
8 # Use the lambda function
9 result = extend_lists(lst1, lst2)
10 print(result) # Output: [2, 3, 4, 5, 6, 12, 13, 14, 15]
11
```

```
→ [2, 3, 4, 5, 6, 12, 13, 14, 15]
```

```
1 d = {1:12, 2:13, 3:14, 4:15, 5: 16}
2 ds = lambda dict1: {i:j**2 for i,j in dict1.items()}
3 print(ds(d))
```

```
→ {1: 144, 2: 169, 3: 196, 4: 225, 5: 256}
```

```
1 #create a lamda function that take two numbers as input and
2 sum= lambda a,b: a+b           # here whole lambda fuction is
3
4 sum(3,4)
```

```
→ 12
```

```
1 #create a lamda function that take a list of numbers as input
2 l=[2,4,5,6,7]
3
4 result= lambda x: sum(x)           # here whole lambda fuction
5
6 result(l)
```

```
→ 24
```

```
1 print(type(result))
```

```
→ <class 'function'>
```

```
1 print(type(x))
```

```
→ <class 'function'>
```

Difference between lamda function and normal function

1. lambda function has no return value, In fact, whole lambda function is returned. see above the type of x is a function. while function can return a value.
2. lambda function is written in a single line.
3. lambda functions are not used for code reusability while normal functions are used for code reusability. lambda functions are developed for one time use.
4. lambda function has no name while normal function has name like def square(): why lambda functions are used> In normal scenarios, we use normal functions. lambda functions are used along with higher order functions. Higher order functions require another function as input perform their job. or higher order function is a function that is returning another function. Lambda functions are used alongside higher-order functions. Higher-order functions are functions that accept other functions as arguments to perform their tasks

```
1 Start coding or generate with AI.
```

Python is not just about writing functions that perform simple tasks. It empowers developers with a powerful feature called “higher-order functions.” These functions take other functions as arguments or return functions as results. A higher-order function is a function that can accept one or more functions as arguments or return a function as its result.

A lambda function, also known as an anonymous function, is a small, unnamed function defined using the lambda keyword. It's a quick way to create small, throwaway functions for simple operations. Lambda functions are especially handy when you need a function for a short period and don't want to define a full def function.

```
1 # create a lambda function that check whether the first char
2 # lambda function can be used in a lot of scenarios
3 c= lambda x: x[0]=='a'
4
5 c('atif')
6
```

```
→ True
```

```
1 c('banana')
```

```
→ False
```

```
1 # condition with lambda function, create a lambda function t  
2 x= lambda n: "Even" if n% 2==0 else "Odd"  
3 x(4)
```

```
→ 'Even'
```

```
1 x(5)
```

```
1 # I want a function that give me three outputs, 1st output  
2 # the second output should be the sum of odd numbers in list  
3 # I need three values from the function  
4 def return_sum(lst):  
5     even_sum=0  
6     odd_sum=0  
7     div3_sum=0  
8     for i in lst:  
9         if i % 2==0:  
10             even_sum= even_sum +i  
11  
12         if i % 2!=0:  
13             odd_sum= odd_sum+i  
14  
15         if i % 3==0:  
16             div3_sum= div3_sum + i  
17  
18     return (even_sum, odd_sum, div3_sum)  
19  
20  
21 lst= [11,14, 21,23,56, 78,45, 29, 28]  
22  
23 even_sum, odd_sum, div3_sum= return_sum(lst)  
24  
25 print("sum of even no is ", even_sum)
```

```
26 print('sum of odd no is ', odd_sum)
27 print('sum of nos divisibile by 3 is', div3_sum)
```

```
→ sum of even no is 176
    sum of odd no is 129
    sum of nos divisibile by 3 is 144
```

Generally, a function performs a specific task on a given input. The advantage of a higher-order function (HOF) is that it can perform various operations or behaviors using lambda functions on the given input. lambda functions are used extensively in python People create higher-order functions and customize/control their behavior using lambda functions

```
1 # we can improve the above normal function return_sum by co
2 # the high order function will expect a function as input al
3
4 def return_sum(func, lst):
5
6     result=0
7
8     for i in lst:
9
10         if func(i): # if func is x, then it is checking whet
11             result= result+i # if number is 11, then the co
12
13
14
15 lst= [11,14, 21,23,56, 78,45, 29, 28]
16
17 # create three lambda functions
18 x= lambda x: x%2==0
19
20 y= lambda y: y%2!=0
21
22 z= lambda z: z%3==0
23
24 print(return_sum(x, lst))
25
26 print(return_sum(y, lst))
```

```
27
```

```
28 print(return_sum(z, lst))
```

→ 176  
129  
144

1. In python, We will study special high order functions- Map, Filter, Reduce, that are built-in in python.
2. The built-in functions like map() and filter() allow us to write simpler, shorter, and more Pythonic code in place of using loops and branching.
3. Map is a function that expects two things: 1st is a lambda function and second is iterable (list, tuple, set, dictionary)
4. An iterable is an object in Python that can be iterated over, meaning you can loop through its elements one by one.
5. In simpler terms, an iterable is anything that you can loop over using a loop construct like a for loop.
6. The map function applies a specified operation/function to all items in an iterable (like a list) and returns a new iterable (usually a map object) with the results of applying the function to each item.

```
1 # suppose u want to double each item of list, you can esily  
2 lst= [1,2,3,4,5,6]  
3 list(map(lambda x: x**3, lst)) # returns a map object  
4  
5 #list(map(lambda x: x*2, lst)) # convert map object into lis  
6  
7
```

→ [1, 8, 27, 64, 125, 216]

```
1 import math
```

```
1 # suppose u want to double each item of list, you can esily  
2 lst= [4,16,32,63,55,125]  
3 list(map(lambda x: math.ceil(math.sqrt(x)), lst)) # returns  
4
```

→ [2, 4, 6, 8, 8, 12]

## 1 Start coding or generate with AI.

1. map(lambda x: x \*2, lst): This line uses the map() function to apply a lambda function to each element in the list lst. The lambda function lambda x: x \*2 takes an input x and returns x multiplied by 2.
  2. So, this line returns a map object that contains the result of applying the lambda function to each element in lst.
  3. However, this line itself doesn't execute the mapping operation; it just creates a map object.
  4. list(map(lambda x: x\*2, lst)): This line converts the map object into a list by passing it to the list() function.
  5. This actually executes the mapping operation, applying the lambda function to each element in lst. The result is a list where each element is the corresponding element from lst multiplied by 2.
  6. So, the resulting list would be [2, 4, 6, 8, 10, 12]. This line demonstrates how to use the map() function along with a lambda function to transform each element in a list.
- 
1. The line of code map(lambda x: x\*2, lst) creates a map object but does not execute the mapping operation immediately.
  2. It creates an **iterator** that will apply the lambda function (lambda x: x\*2) to each element of the iterable 'lst' when iterated over or when **another function like list() is called to consume the iterator**.
  3. **In Python, functions like map(), filter(), and zip() return iterator objects that represent a sequence of operations to be performed lazily (i.e., as needed). This lazy evaluation is efficient, especially for large datasets, as it avoids unnecessary computation until the results are actually needed.**

```
1 # using map, u can check whether each item in the list is od
2 lst= [1,2,3,4,5,6]
3 list(map(lambda x: if x%2==0, lst))
4 #list(map(lambda x: x%2==0, lst))
5
```

→ Cell In[51], line 3  
list(map(lambda x: x if x%2==0, lst))  
^  
SyntaxError: expected 'else' after 'if' expression

```
1 #This code uses the map function along with a lambda functio
2
3 students = [
4     {"name": "basit", "father": "abdus salam", "address": "p
5     {"name": "Bob", "father": "Michael", "address": "456 Elm
6     {"name": "Charlie", "father": "David", "address": "789 O
7 ]
8
9 list(map(lambda std:std["name"] , students))
10
11 #lambda std: std["name"]: This is a lambda function that tak
12 #and returns the value associated with the key "name" in tha
13
14 #students: This is the iterable (list of dictionaries) to wh
```

```
→ ['basit', 'Bob', 'Charlie']
```

The map function applies a given function to all items in an iterable and returns a new iterable with the results. It's a concise way to transform data.

or **How to use the lambda function with map()?**

**The map() function in Python takes in a function and an iterable (lists, tuples, and strings) as arguments. The function is called with all the items in the list, and a new list is returned, which contains items returned by that function for each item.**

```
1 # Define a function to square a number
2 def square(x):
3     return x ** 2
4
5 # Apply the square function to a list of numbers
6 numbers = [1, 2, 3, 4, 5]
7 squared = list(map(square, numbers))
8
9 print(squared)
10 # Result: [1, 4, 9, 16, 25]
```

```
→ [1, 4, 9, 16, 25]
```

1  
2

```
3 # In a map() function, we can also use a lambda function ins·
4
5 # Using a lambda function to square numbers
6 numbers = [1, 2, 3, 4, 5]
7 squared = list(map(lambda x: x ** 2, numbers))
8 print(squared)
9 # Result: [1, 4, 9, 16, 25]
```

→ [1, 4, 9, 16, 25]

```
1 #Using map to Convert Strings to Integers
2
3 # Convert a list of strings with numbers to integers using m
4 numbers_as_strings = ["1", "2", "3", "4", "5"]
5
6 numbers_as_integers = list(map(int, numbers_as_strings))
7 print(numbers_as_integers)
8 # Result: [1, 2, 3, 4, 5]
```

→ [1, 2, 3, 4, 5]

```
1 #The map() function executes a given function to each elemen·
2 numbers = [1,2,3,4]
3
4 # returns the cube of a number
5 def cube(number):
6     return number * number * number
7
8 # apply square() to each item of the numbers list
9 cubed_numbers = map(cube, numbers)
10
11 # converting to list for printing
12 result = list(cubed_numbers)
13 print(result)
14
15 # Output: [1,4,9,16]
```

→ [1, 8, 27, 64]

map() Syntax

map(function, iterables)

map() Arguments

The map() function takes two arguments:

function - a function that is applied to each element of an iterable.

iterables - iterables such as lists, tuples, etc.

Note: We can pass more than one iterable to the map() function.

map() Return Value

The map() function returns a map object, which can be easily converted to lists, tuples, etc.

```
1 def cube(n):
2     return n*n*n
3
4 numbers = (1, 2, 3, 4) #defined a tuple named numbers with 4
5 result = map(cube, numbers) #Here, the map() function square
6 print(result)
7
8 # converting the map object to set
9 result = set(result) #
10 print(result)
11
12 #The output is not in order because sets in Python are unord
13 #and do not preserve the original sequence of elements.
14 #-----
15
16 #we have directly used the lambda function to perform the sq
17
18 #Note: Use of lambda() function makes the code concise and e
19
20 numbers = (1, 2, 3, 4) #defined a tuple named numbers with 4
21 result = map(lambda x: x*x*x, numbers) #Here, the lambda fun
22 print(result)
23
24 # converting the map object to set
25 result = set(result) #
26 print(result)
```

→ <map object at 0x0000026D9E079930>
{8, 1, 27, 64}

```
<map object at 0x0000026D9DFE7040>
{8, 1, 27, 64}
```

## Add Multiple Lists using map() and lambda

We can use map() and lambda to add multiple lists in Python. For example,

```
1 num1 = [1, 2, 3]
2 num2 = [10, 20, 40, 50]
3
4 # add corresponding items from the num1 and num2 lists
5 result = map(lambda n1, n2: n1+n2, num1, num2) #Here, the la
6 print(list(result))
```

→ [11, 22, 43]

## String Modification using map()

We can use map() function to modify the string. For example,

```
1 list('eat')
```

→ ['e', 'a', 't']

```
1 # list of actions
2 actions=['eat', 'sleep', 'read']
3 # convert each string into list of individual characters
4 result= list(map(list,actions))
5 print(result)
```

→ [[‘e’, ‘a’, ‘t’], [‘s’, ‘l’, ‘e’, ‘e’, ‘p’], [‘r’, ‘e’, ‘a’, ‘d’]]

## The filter Function: Selecting Elements

1. The filter function filters elements from an iterable based on a given function (which returns True or False) and returns a new iterable with the elements that satisfy the condition.
2. or The filter() function selects elements from an iterable based on the result of a function.
3. The function takes two parameters:
4. function - a function that runs for each item of an iterable
5. iterable - a sequence that needs to be filtered like sets, lists, tuples, etc filter() Return Value  
The filter() function returns an iterator.

## or How to use the lambda function with filter()?

The filter() function in Python takes in a function and an iterable (lists, tuples, and strings) as arguments.

The function is called with all the items in the list, and a new list is returned, which contains items for which the function evaluates to True.

```
1
2 #Using filter with def
3
4 # Define a function to filter even numbers
5 def is_even(x):
6     return x % 2 == 0
7
8 # Filter even numbers from a list
9 numbers = [1, 2, 3, 4, 5]
10 evens = list(filter(is_even, numbers))
11 print(evens)
```

→ [2, 4]

```
1
2 # Result: [2, 4]
3
4 ****
5
6 #Using filter with Lambda Functions
7
8 # Using a lambda function to filter even numbers
9 numbers = [1, 2, 3, 4, 5]
10 evens = list(filter(lambda x: x % 2 == 0, numbers))
11 print(evens)
12 # Result: [2, 4]
```

→ <filter object at 0x00000214AF59BA30>
[2, 4]

## Explanation of the above code

In each iteration, filter() calls the is\_even function with one element from the numbers list, and then

it decides whether to include that element in the filtered result based on the return value of the is\_even function for that particular element.

Here's the process in detail:

filter() iterates over each element in the numbers list. For each element, it calls the is\_even function with that element as an argument. The is\_even function checks whether the current element is even or not, and it returns True if the element is even and False otherwise. Based on the return value of the is\_even function for each element, filter() decides whether to include that element in the filtered result. After iterating over all elements in the list, filter() returns a filtered iterable containing only the elements for which the is\_even function returned True.

```
1 letters = ['a', 'b', 'd', 'e', 'i', 'j', 'o']
2
3 # a function that returns True if letter is vowel
4 def filter_vowels(letter):
5     vowels = ['a', 'e', 'i', 'o', 'u']
6     if letter in vowels:
7         return True
8     else:
9         return False
10
11 # selects only vowel elements
12 filtered_vowels = filter(filter_vowels, letters)
13
14 # converting to tuple
15 vowels = tuple(filtered_vowels)
16 print(vowels)
17
18 # Output: ('a', 'e', 'i', 'o')
```

→ ('a', 'e', 'i', 'o')

```
1 letters = ['a', 'b', 'd', 'e', 'i', 'j', 'o']
2
3 # a function that returns True if letter is vowel
4
5 vowels = ['a', 'e', 'i', 'o', 'u']
6
7
```

```
8 # selects only vowel elements
9 #each element of letters is passed to the lambda function
10 #if it returns True, filter() selects the element
11
12 filtered_vowels = filter(lambda x: x in vowels, letters)
13
14 # converting to tuple
15 #Here, the program returns the iterator, which we converted
16 vowels = tuple(filtered_vowels)
17 print(vowels)
```

→ ('a', 'e', 'i', 'o')

```
1 # Using a lambda function to filter numbers greater than 4
2 numbers = [1, 24, 43, 34, 45, 7, 2, 3, ]
3 gt = list(filter(lambda x: x > 4, numbers))
4 print(gt)
5 # Result: [24, 43, 34, 45, 7]
```

→ [24, 43, 34, 45, 7]

```
1 fruits=["Apple", "orange", "mango", "guava"]
2 list(filter(lambda fruit: 'e' in fruit, fruits)) # for each
```

→ ['Apple', 'orange']

1. The reduce function, provided in the functools module in Python, is used to apply a function to an iterable (such as a list) cumulatively to reduce it to a single value.
2. In the context of the reduce function, "cumulatively" means that the function provided to reduce is applied repeatedly to the items of the iterable (in this case, the numbers list), where the result of each application becomes one of the inputs for the next application of the function.
3. Syntax: `functools.reduce(function, iterable[, initializer])`

Parameters:

3. **function**: The function to apply cumulatively to the items of the iterable. It should accept two arguments and return a single value. **[The function passed to reduce should accept two arguments. These arguments represent: The "accumulator": This is the cumulative result**

**obtained so far. The "current value": This is the next value from the iterable that needs to be processed.]**

4. iterable: The iterable sequence of items to be reduced.
5. initializer (optional): An optional value that is used as the initial value of the accumulator. If provided, reduce will return this value when the iterable is empty.

### Working:

6. The reduce function takes the first two elements of the iterable and applies the given function to them, producing a single result.
7. It then takes this result and the next element of the iterable and applies the function again, producing another result.
8. This process continues until all elements of the iterable have been processed, resulting in a single value. The final value is returned as the output of the reduce function.



```
1 from functools import reduce
2
3 # Example 1: Summing all elements of a list
4 numbers = [1, 2, 3, 4, 5]
5 total = reduce(lambda x, y: x + y, numbers)
6 print(total) # Output: 15 (1 + 2 + 3 + 4 + 5)
7
8 # Example 2: Finding the maximum value in a list
9 numbers = [11, 12, 56, 58, 32]
10 max_value = reduce(lambda x, y: x if x > y else y, numbers)
11 print(max_value) # Output: 58
12
```

→ 15  
58



```
1 # Example 2: Finding the maximum value in a list
2 numbers = [11, 12, 56, 58, 32]
3 max_value = reduce(lambda x, y: x if x < y else y, numbers)
4 print(max_value) # Output: 30
```

```
1 #The add function takes two arguments (x and y) and returns
2 #The reduce function applies the add function cumulatively to
3 #resulting in the sum of all numbers.
4
5 from functools import reduce
6
7 numbers = [1, 2, 3, 4, 5]
8
9 # Define a function to add two numbers
10 def add(x, y):
11     return x + y
12
13 # Use reduce to find the sum of all numbers
14 total = reduce(add, numbers)
15 print(total) # Output: 15 (1 + 2 + 3 + 4 + 5)
16
```

```
1
2 numbers = [11, 12, 56, 58, 32]
3 max_value = reduce(lambda x, y: x if x > y else y, numbers)
4 print(max_value) # Output: 30
5
6 ######
7 from functools import reduce
8
9 def find_max(x, y):
10     if x > y:
11         return x
12     else:
13         return y
14
15 numbers = [11, 12, 56, 58, 32]
16 max_value = reduce(find_max, numbers)
17 print(max_value) # Output: 58
18
```

→ 58

```
1 from functools import reduce
2
3 # Define a function to find the product of two numbers
4 def multiply(x, y):
5     return x * y
6
7 # Find the product of all numbers in a list
8 numbers = [1, 2, 3, 4, 5]
9 product = reduce(multiply, numbers)
10 print(product)
11 # Result: 120 (1 * 2 * 3 * 4 * 5)
```

→ 120

```
1 # Find the product of all numbers in a list using lambda fun
2 numbers = [1, 2, 3, 4, 5]
3 product = reduce(lambda x,y: x * y, numbers)
4 print(product)
```

→ 120

```
1 n= int(input('enter a number to find its factorial'))
2 product = reduce(lambda x,y: x * y,range(1,n+1))
3 print(product)
```

→ enter a number to find its factorial 5  
120