Arrays and Vectorized Computation

1.Numpy module:

NumPy arrays can be created from various Python data structures like lists and tuples, or by using intrinsic NumPy functions such as `zeros`, `ones`, `arange`, and `linspace`. Additionally, random arrays can be generated using functions like `rand`, `randint`, and `randn`. Once created, these arrays can be manipulated through indexing to access individual elements, slicing to extract subarrays, reshaping to alter their shape, and using functions for joining or splitting arrays. Computation on arrays is made efficient with NumPy's universal functions (ufuncs), which allow for fast element-wise operations such as arithmetic calculations, trigonometric evaluations, and exponential or logarithmic transformations. Data can also be imported into NumPy arrays from external files, enabling the application of statistical methods (like mean, median, and standard deviation) and mathematical operations (such as sum, min, and max), along with comparison functions for element-wise conditions. NumPy also offers sorting functionality, provides methods to find unique elements, and supports set operations like union, intersection, and difference. Beyond numerical data, NumPy can be used to manipulate images by loading them as arrays, allowing for operations like cropping and flipping through array indexing, making it an essential tool for image processing tasks.

1. Create Numpy Arrays from PythonDataStructures,Intrinsic Numpy Objects and Random

Functions

```
 # Importing the NumPy library, which is used for numerical operations
import numpy as np
# Defining a Python list with five integer element
a = [1, 2, 3, 4, 5] b = np.array(a) print(a)
```

1.1 Arrays from python datastructures

In [1]:

[1, 2, 3, 4, 5] In

[2]:

```
#creates two lists x and y, combines them into a 2D NumPy array z.
import numpy as np x=[1,2,3] y=[3,4,5] z=np.array((x,y)) print(z)
```

[[1 2 3]
 [3 4 5]] In

[3]:

```
# creates two tuples a and b, combines them into a 2D NumPy array
c. import numpy as np a=(1,2,3,4,5) b=(6,7,8,9,1) c=np.array((a,b))
print(c)
```

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

[4]:

Out[4]:

```
# list `a`, converts it into a set `c`, and then attempts to convert `c` into a NumPy
array. a=[1,2,3,4,5] c=set(a) np.array(c)
```

```
#converts a dictionary dict into a 2D NumPy array z containing its key-value pairs, prints z, converts
th import numpy as np dict={'a':1,'b':2,'c':3} z=np.array(list(dict.items())) print(z)
a=np.array(list(dict.keys())) print(a)
```

array({1, 2, 3, 4, 5}, dtype=object) In [5]:

```
[['a' '1']
['b' '2']
 ['c' '3']]
['a' 'b' 'c']
```

1.2 Intrinsic Numpy Objects

Intrinsic NumPy objects are fundamental data structures provided by the NumPy library, which are optimized for numerical computations and provide efficient operations on large datasets.

In [6]:
```
#The code creates a NumPy array `a` containing values from 0 to 8 (inclusive) using
`np.arange(9)`. a=np.array(np.arange(9)) print(a)
```

```
#creates a NumPy array a of length 3 filled with zeros using
np.zeros(3). a=np.zeros(3) print(a)
```

```
[0 1 2 3 4 5 6 7 8] In
```
[7]:

```
[0. 0. 0.] In
```
[8]:
```
#The code creates a NumPy array `a` of length 4 filled with ones using
`np.ones(4)`. b=np.zeros([3,3]) print(b)
```

```
[[0. 0. 0.]
   [0.       0. 0.]
        [0.  0. 0.]] In [9]:
```

```
a=np.ones(4) print(a)
```

```
        [1.  1. 1. 1.] In [10]:
```

```
b=np.ones([3,3]) print(b)
```

```
[[1. 1. 1.]
[1. 1. 1.]
   [1.       1. 1.]] In [11]:
```

```
# creates a 3x3 identity matrix a using np.eye(3) and then prints the matrix. The identity matrix has
one a=np.eye(3) print(a)
```

```
[[1. 0. 0.]
[0. 1. 0.]
 [0. 0. 1.]] In
```
[12]:
```
#code creates a 3x3 matrix c using np.eye(3, k=1) with ones on the diagonal just above the main
diagonal c=np.eye(3,k=1) print(c)
```

```
[[0. 1. 0.]
[0. 0. 1.]
```

```
#code creates a 3x3 identity matrix a using np.identity(3) and The identity matrix has ones on the
main d a=np.identity(3) print(a)
```

```
[0. 0. 0.]]
```
In [13]:

```
[[1. 0. 0.]
[0. 1. 0.]
 [0. 0. 1.]] In
```

[14]:
```
d=np.full((2,2),7) print(d)
```

```
[[7 7]
 [7 7]] In
```

[15]:
```
a=np.empty((2,3)) print(a)
```

```
[[6.23042070e-307 4.67296746e-307 1.69121096e-306]
 [7.56598449e-307 1.89146896e-307 7.56571288e-307]] In
```

[16]:
```
np.diag([1,2,3,4])
```

Out[16]:
```
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
              [0, 0, 3, 0],
              [0, 0, 0, 4]]) In
```

[17]:
```
#o create two 2D arrays x and y from the 1D arrays x and y, allowing for a grid of coordinates.
x=np.array([1,2,3])
y=np.array([4,5,6])
x,y=np.meshgrid(x,y) print(x)
print(y)
```

```
[[1 2 3]
[1 2 3]
 [1 2 3]]
[[4 4 4]
[5 5 5]
 [6 6 6]]
```

1.3 Random Functions

The random functions in NumPy are essential for simulations, statistical sampling, and generating synthetic data. They help facilitate various operations in scientific computing, machine learning, and data analysis.

In [18]:
```
#code imports the random module from NumPy and generates a random integer x between 0 and
99 from numpy import random x = random.randint(100) print(x)
```

```
52
```

In [19]:
```
y=np.random.bytes(7) print(y)
a=np.random.choice(['true','false'],size=(2,3)) print(a)
```

```
b'V\x88\x8b\xads\xd2\x91'
[['true' 'true' 'true']
 ['false' 'false' 'false']] In
```

[20]:
```
# a complex number from two random values, and separately prints its real and imaginary components.
x = random.rand(1) + random.rand(1)*1j print (x) print(x.real) print(x.imag)
```

```
[0.53509954+0.55378113j]
[0.53509954]
```

[0.55378113] In

[21]:
```
#complex 2D array with one row and five columns, where each element consists of a random real part
and a x = random.rand(1,5) + random.rand(1,5)*1j print (x)
```

```
[[0.80585931+0.2420614j  0.54479211+0.70111829j 0.75242834+0.94709726j
  0.3909485 +0.01541345j 0.01562967+0.27551819j]]
```

In [22]:
```
#creates a 2D array of complex numbers with dimensions 2x2, where each element consists of a random
real np.random.random(size=(2,2))+1j*np.random.random(size=(2,2))
```

Out[22]:
```
array([[0.97009908+0.40755078j, 0.93247886+0.84556959j],
       [0.88352279+0.20473193j, 0.78605216+0.85974768j]])
```

In [23]:
```
np.random.permutation(5)
```

Out[23]:
```
array([0, 4, 1, 3, 2])
```

In [24]:
```
a=np.array(5)
b=np.random.choice(a,size=5,p=[0.1,0.2,0.3,0.2,0.2])

print(b)
```

```
[2     4 1
```

```
np.random.randint(1,5)
```

```
1     4] In
```

[25]: Out[25]: 1 In

[26]:
```
a=np.random.randn(1,10) print(a)
```

```
[[-0.35025419 -0.64943739 -0.83090508  0.35285294 -1.17989692 -0.17760276
-
```

```
#code creates a NumPy array of fruit names and randomly selects one fruit name to
print a=np.array(['apple','bananaa','cherry']) b=np.random.choice(a) print(b)
```

```
0.35255731 -0.45490048  1.43300062  0.67521065]] In [27]:
```

```
bananaa In
```

[28]:
```
#shuffle function to randomly reorder the elements of the array a
np.random.shuffle(a) print(a)
```

```
['bananaa' 'cherry' 'apple']
```

2.Manipulation Of Numpy Arrays

2.1 Indexing

Indexing in NumPy refers to accessing individual elements or groups of elements within an array

In [29]:
```
#elements from a NumPy array using advanced indexing, and prints a specific element based on row and
colu import numpy as np x = np.array([[1, 2], [3, 4], [5, 6]]) y = x[[0,1,2], [0,1,0]] print(x[0,1])
```

2 In [30]:

```
a=[3,4,5,6,7] print(a[0])
```

3

In [31]:

```
arr3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]]) arr3d[0]
```

Out[31]:

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

[32]:

```
#the first "slice" of a 3D NumPy array by replacing it with the value
42 old_values = arr3d[0].copy() arr3d[0] = 42 print(arr3d)
```

```
[[[42 42 42]
  [42 42 42]]

 [[ 7  8  9]
  [10 11 12]]] In
```

[33]:

```
import numpy as np arr =
np.array([1, 2, 3, 4]) print(arr[2]
+ arr[3])
```

```
import numpy as np arr =
np.array([[1,2,3,4,5], [6,7,8,9,10]]) print(
arr[0, 1])
```

7 In

[34]:

2

In [35]:

```
import numpy as np arr =
np.array([[1,2,3,4,5], [6,7,8,9,10]]) print(
arr[1, 4])
```

10

In [36]:

```
import numpy as np arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8,
9], [10, 11, 12]]]) print(arr[0, 1, 2])
```

```
#to access specific elements within a 2D NumPy array using indexing.
import numpy as np arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print( arr[1, -1])
```

6 In

[37]:

10

2.2 Slicing

Slicing in NumPy refers to the process of selecting a specific subset of elements from an array. It allows you to create a new view of the original data without copying it, which can be very efficient in terms of memory usage.

In [38]:
```
#ode creates a 1D NumPy array and prints the elements from index 1 to index
2 import numpy as np arr=np.array([5,6,7,8,9]) print(arr[1:3])
```

```
[6 7]
```

In [39]:
```
arr=np.array([5,6,7,3,6,8,9]) print(arr[1:])
```

```
[6 7 3 6 8 9] In
```

[40]:
```
arr=np.array([5,6,7,3,6,8,9]) print(arr[1:])
```

```
arr=np.array([5,6,7,8,9])
print(arr[:3])
```

```
 6 7]
```

```
arr=np.array([5,6,7,8,9]) print(arr[-3:-
1])
```

```
 8]
```

```
arr=np.array([5,6,7,8,9])
print(arr[:3])
```

```
 6 7]
```

```
arr=np.array([5,6,7,8,9])
print(arr[:3 ])
```

```
 6 7]
```

```
arr=np.array([5,6,7,8,9]) print(arr[-3:-
1])
```

```
 8]
```

```
arr=np.array([5,6,7,8,4,5,6,7,9])
print(arr[1:5:2])
```

```
 8]
```

```
arr=np.array([5,6,7, 8,4,5,6,7,9]) pr nt(arr[-1:-5:-
1])
```

```
 7 6 5]
```

```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
```

```
 8 9]
```

```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:2, 2])
```

```
 8]
```

```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:2, 1:4])
```

```
[6 7 3 6 8 9] In
```

[41]:

[5 I

n

[

4

2

]

:

[7

In [43]:

[5

In [44]:

[5

In [45]:

[7

In [46]:

[6 I

n

[

4

7

]

:

[9

In [48]:

[7 I

n

[

4

9

]

:

[3

In [50]:

[[2 3 4]
 [7 8 9]] In

[51]:

```
b = "Hello, World!" print(b[2:5])
```

```
b = "Hello, World!"
print(b[:5])
```

llo In

[52]:

Hello In

[53]:
```
b = "Hello, World!" print(b[2:])
```

llo, World!

2.3 Re-Shaping

Reshaping in NumPy is the process of changing the shape (i.e., dimensions) of an existing array without altering the data. This is particularly useful when you need to transform an array to fit a certain shape for further operations, such as machine learning or data processing task

In [54]:
```
import numpy as np arr = np.array([[1, 2, 3,
4], [5, 6, 7, 8]]) print(arr.shape)
```

(2, 4)

In [55]:
```
arr = np.array([1, 2, 3, 4], ndmin=5)
print(arr) print('shape of array :',
arr.shape)
```

[[[[[1 2 3 4]]]]] shape
of array
: (1, 1,
1, 1, 4)

In [56]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
arr1= arr.reshape(4, 3) print(arr1)
```

[[1 2 3]
[4 5 6]
 [7 8 9]
 [10 11 12]] In

[57]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
arr1 = arr.reshape(2, 2, 3) print(arr1)
```

[[[1 2 3]
  [4 5 6]]

 [[7 8 9]
  [10 11 12]]] In

[58]:
```
a=np.arange(8) print(a.reshape(4,2))
```

[[0 1]
[2 3]
 [4 5]
 [6 7]] In

[59]:
```
a=np.arange(12).reshape(4,3) print(a)
```

[[0 1 2]
[3 4 5]
 [6 7 8]
 [9 10 11]]

2.4 Joining Arrays

Joining arrays in NumPy is a way of combining two or more arrays into a single array. There are several ways to join arrays, depending on the desired result and the shape of the input arrays.

In [60]:
```
a1=np.arange(6).reshape(3,2) a2=np.arange(6).reshape(3,2)
print(np.concatenate((a1,a2),axis=1))
```

```
[[0 1 0 1]
[2 3 2 3]
 [4 5 4 5]] In
```

[61]:
```
#numpy.hstack and vstack a
= np.array([[1,2],[3,4]]) b
=
np.array([[5,6],[7,8]]) print(np.stack((a,b)))
```

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

 [[5 6]
  [7 8]]]
```

In [62]:
```
print(np.stack((a,b),axis=0))
```

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

 [[5 6]
  [7 8]]]
```

In [63]:
```
print(np.stack((a,b),axis=1))
```

```
[[[1 2]
  [5 6]]

 [[3 4]
  [7 8]]] In
```

[64]:
```
ch = np.hstack((a,b)) print(ch)
```

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

[65]:
```
ch = np.vstack((a,b)) print(ch)
```

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

2.5 Splitting

Splitting in NumPy involves dividing an array into multiple sub-arrays. This can be useful when you need to partition data for different processing purposes or when dealing with chunks of data in a structured way.

In [66]:
```
import numpy as np a
= np.arange(9)
print(a)
```

```
[0 1 2 3 4 5 6 7 8] In
```

[67]:
```
b = np.split(a,3) print(b)
```

```
a =
np.arange(12).reshape(4,3) b=np.hsplit(a,3)
print(b)
```

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

In [68]:

```
[array([[0],
       [3],
       [6],
       [9]]), array([[ 1],
       [ 4],
       [ 7],
       [10]]), array([[ 2],
       [ 5],
       [ 8],
       [11]])]
```

```
b=np.vsplit(a,2)
print(b)
```

In [69]:

```
[array([[0, 1, 2],
       [3, 4, 5]]), array([[ 6,  7,  8],
       [ 9, 10, 11]])]
```

3.Computation On Numpy Arrays Using Universal Functions

3.1 Unary Universal Functions

Unary Universal Functions (also known as unary ufuncs) in NumPy are mathematical functions that operate on a single input array element-wise. These functions apply a specific mathematical operation to each element of an array independently, resulting in an output array of the same shape.

In [70]:

```
arr = np.arange(10) print(arr)
```

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

In [71]:

```
np.sqrt(arr)
```

Out[71]:

```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])
```

In [72]:

```
np.exp(arr)
```

Out[72]:

```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
       5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
       2.98095799e+03, 8.10308393e+03])
```

In [73]:

```
np.min(arr)
```

Out[73]:

```
0
```

In [74]:

```
np.max(arr)
```

Out[74]:

```
9
```

In [75]:

```
np.average(arr)
```

Out[75]:

```
4.5
```

In [76]:

```
print(np.abs(arr))
```

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

In [77]:

```
arr=np.arange(0,-5,-0.5) print(np.fabs(arr))
```

```
[0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5]
```

3.2 Binary Universal Functions

Binary Universal Functions (also known as binary ufuncs) operate on two input arrays element-wise. These functions require two arrays (or one array and one scalar) and perform a mathematical operation between corresponding elements.

```
In [78]: x = np.random.randn(8) y
         = np.random.randn(8)
         print(x)
```

```
[-0.43340117  0.47393419  0.25304006 -1.31141943 -0.89094493  0.13425514
  1.25837246 -0.17732649]
```

```
In [79]: print(y)
```

```
[ 1.69294280e+00  4.99938563e-01  1.40610787e+00  6.60917678e-01
```

```
         np.maximum(x, y)
```

```
-2.97195702e-01 -1.26245285e-03 -1.24439307e+00 -7.85070022e-02]
In [80]:
```

```
Out[80]: array([ 1.6929428 ,  0.49993856,  1.40610787,  0.66091768, -0.2971957 ,
0.13425514,  1.25837246, -0.078507  ]) In [81]:
```

```
         arr = np.random.randn(7) * 5 remainder,
         whole_part = np.modf(arr)
         print(remainder)
```

```
[ 0.25638236 -0.36051531  0.06918192 -0.3267168  -0.66199236  0.62564871
0.59405867]
```

```
In [82]: print(whole_part)
```

```
[ 7. -3.  5. -4. -0.  7.  1.] In
```

```
[83]:
         import numpy as np a = np.arange(9).reshape(3,3)
         b =
         np.array([[10,10,10],[10,10,10],[10,10,10]]) print(np.add(a,b))
```

```
[[10 11 12]
[13 14 15]
 [16 17 18]]
```

```
In [84]: np.subtract(a,b)
Out[84]:      array([[-10,  -9,  -
             8],        [ -7,  -6,
                              -5],
                    [ -4,  -3,  -2]])
In [85]: np.multiply(a,b)
Out[85]:        array([[ 0, 10,
                20],        [30,
                       40, 50],
                  [60,      70,
              80]])
In [86]: np.divide(a,b)
Out[86]:         array([[0. , 0.1,
                0.2],        [0.3,
                       0.4, 0.5],
                  [0.6,       0.7,
              0.8]])
```

```
In [87]: import numpy as np a =
         np.array([10,100,1000]) np.power(a,2)
```

```
Out[87]: array([    100,   10000, 1000000], dtype=int32)
```

4.Compute Statistical and Mathematical Methods and Comparison Operations on rows/columns

4.1 Mathematical and Statistical methods on Numpy Arrays

NumPy provides a variety of mathematical and statistical methods to perform operations on arrays.

```
In [88]: a = np.array([[3,7,5],[8,4,3],[2,4,9]])
a
```

```
Out[88]:       array([[3, 7,
              5],        [8,
                     4, 3],
                    [2, 4, 9]])
```

```
In [89]: a.sum() Out[89]: 45
In [90]: import numpy as np
         a = np.array([[30,40,70],[80,20,10],[50,90,60]])
         np.percentile(a,90)
```

```
Out[90]: 82.0
In [91]: arr = np.random.randn(5, 4)

In [92]: arr.mean()
Out[92]: 0.2747706750708369
In [93]: arr.mean(axis=1)

Out[93]: array([ 0.32267986,  1.25527755,  0.47312816, -0.25985255, -0.41737964])

In [94]: np.median(arr)
Out[94]: 0.053769781267203304
In [95]: np.std(arr)
Out[95]: 1.0838958196039732
In [96]: np.var(arr)
Out[96]: 1.1748301477549687
In [97]: arr.sum(axis=0)

Out[97]: array([ 0.20844111, -2.23172056,  6.05887522,  1.45981773])

In [98]: arr = np.array([0, 1, 2, 3, 4, 5, 6, 7]) print(arr.cumsum())
         [ 0  1  3  6 10 15 21 28]
In [99]:
         arr = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]]) print(arr.cumsum(axis=0))
         [[ 0  1  2]
         [ 3  5  7]
          [ 9 12 15]]
In [100…  print(arr.cumprod(axis=1))
         [[  0   0   0]
         [  3  12  60]
          [  6  42 336]]
```

4.2 Comparison Operations

Comparison operations in NumPy allow element-wise comparison between arrays or with scalars.

```
In [101…  a=np.array([[1,2],[3,4]]) b=np.array([[1,2],[3,4]])
          print(np.array_equal(a,b))
          True
```

In [102…
```
a=np.array([1,15,6,8]) b=np.array([11,12,6,4])
```

In [103…
```
print(np.greater(a,b))
```

```
[False  True False  True]
```

In [104…
```
print(np.greater(a[0],b[2]))
```

```
False
```

In [105…
```
print(np.greater_equal(a,b))
```

```
print(np.less(a[0],b[2]))
```

```
[False  True  True  True]
```

In [106…

```
True
```

In [107…
```
print(np.less(a,b))
```

```
print(np.less_equal(a,b))
```

```
[ True False False False]
```

In [108…

```
[ True False  True False]
```

5.Computation on Numpy Arrays using Sorting,unique and Set Operations

5.1 Sorting

Sorting helps to arrange elements of an array in a particular order.

In [109…
```
import numpy as np a =
np.array([[3,7],[9,1]]) print(a)
```

```
[[3 7]
 [9 1]]
```

In [110… np.sort(a) Out[110…
```
array([[3, 7],
                [1,
       9]]) In [111…
np.sort(a,axis=0) Out[111…
      array([[3, 1],
                [9,
       7]]) In [112…
np.sort(a,axis=1) Out[112…
      array([[3, 7],
                [1,
         9]])
```

In [113… arr = np.random.randn(5, 3) print(arr)

```
[[-0.44159914 -1.56972001  1.15266243]
 [ 2.18535348  0.33599058  1.32702012]
 [-1.85367937  0.73952353  0.58621683]
 [ 2.0075274  -1.32156048  0.64298278]
 [-0.29090408 -0.52250952 -0.75770417]]
```

In [114…
```
arr.sort(1) print(arr)
```

```
[[-1.56972001 -0.44159914  1.15266243]
[ 0.33599058  1.32702012  2.18535348]
 [-1.85367937  0.58621683  0.73952353]
 [-1.32156048  0.64298278  2.0075274 ]  [-
0.75770417 -0.52250952 -0.29090408]]
```

5.2 Unique Operation

The np.unique() function is used to find the unique elements of an array.

In [115…
```
names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe']) print(np.unique(names))
```
```
['Bob' 'Joe' 'Will']
```

In [116…
```
# Contrast np.unique with the pure Python alternative: sorted(set(names))
```
Out[116…
```
['Bob',
'Joe', 'Will'] In
```

[117…
```
ints = np.array([3, 3, 3, 2, 2, 1, 1, 4, 4]) print(np.unique(ints))
```
```
[1 2 3 4]
```

5.3 Set Operations

NumPy provides functions that perform set operations on arrays

In [118…
```
import numpy as np values = np.array([6,
0, 0, 3, 2, 5, 6]) print(np.in1d(values,
[2, 3, 6]))
```
```
[ True False False  True  True False  True]
```

In [119…
```
arr1=np.array([1,2,3,4]) arr2=np.array([3,4,5,6])
```

In [120…
```
print(np.union1d(arr1,arr2))
```
```
[1 2 3 4 5 6]
```

In [121…
```
print(np.intersect1d(arr1,arr2))
```
```
[3 4]
```

In [122…
```
print(np.setdiff1d(arr1,arr2))
```
```
[1 2]
```

In [123…
```
print(np.setxor1d(arr1,arr2))
```
```
[1 2 5 6]
```

6.Load an image file and do crop and flip operation using Numpy indexing

To load and manipulate images with NumPy, you can use the Pillow (PIL) library to load an image and convert it into a NumPy array.

```
In [11]: from PIL import Image
         img=Image.open("img.jpeg") img.format
```
```
Out[11]: 'JPEG'
```
```
In [12]: import numpy as np
         a=np.array(img) print(a)
```
```
[[[249 253 254]
```

```
[249 253 254]
[249 253 254]
...
  [248 253 255]
[248  253 255]
[248 253 255]]

 [[249 253 254]
[249  253 254]
[249 253 254]
...
  [248 253 255]
[248  253 255]
[248 253 255]]

 [[249 253 254]
[249  253 254]
[249 253 254]
...
  [248 253 255]
  [248 253 255]
[248 253 255]]  ...
 [[245 249 252]
[245 249 252]
[245 249 252]
...
  [243 248 252]
  [243 248 252]
  [243 248 252]]

 [[245 249 252]
[245 249 252]
[245 249 252]
...
  [243 248 252]
  [243 248 252]
  [243 248 252]]

 [[245 249 252]
[245 249 252]
[245 249 252]
...
  [243 248 252]
  [243 248 252]
  [243 248 252]]] In
```

[29]:
```
cropped_img=a[0:400,218:400,:] img=Image.fromarray(a).show()
```

In [30]:
```
img=Image.fromarray(cropped_img).show()
```

In [20]:
```
from IPython.display import display
# Display the original, cropped and flipped images display(Image.fromarray(a))
```

In [21]:
```
display(Image.fromarray(cropped_img))
```

In [17]:
```
flipped_img=np.flipud(a)
img=Image.fromarray(flipped_img).show() display(Image.fromarray(flipped_img))
```

In [ ]:

2) Data Manipulation with Pandas

Data manipulation with Pandas starts with creating a Pandas Series, which can be done from a Python list, NumPy array, or dictionary. Once the Series is created, various operations can be performed, such as indexing, selecting, and filtering elements based on conditions. Arithmetic operations can also be applied element-wise on the Series, along with ranking and sorting values. Checking for null values is another important task, and you can concatenate multiple Series together for further analysis.

You can also create a DataFrame, which is a two-dimensional data structure, from lists or dictionaries. Pandas allows you to import data from various file formats (CSV, Excel, etc.) into a DataFrame, and once imported, you can manipulate the data in several ways. You can display the first five rows using the `head()` function and the last five rows with the `tail()` function. Pandas also enables you to get a detailed view of the DataFrame's shape, data types, null values, index, and columns. You can select or delete specific rows or columns based on conditions, and perform sorting and ranking operations within the DataFrame. Statistical operations like mean, median, and standard deviation can be applied to the numeric data in the DataFrame. For categorical data, Pandas offers functions to count occurrences and find the uniqueness of values. Additionally, you can rename single or multiple columns to improve the clarity of the DataFrame.

1.create pandas series from python List ,Numpy Arrays and Dictionary

```
import pandas as pd import
numpy as np data=[4,7,-
5,3] a=pd.Series(data)
print(a)
```

1.1Pandas Series From Python List

In [2]:

```
0    4
1    7
2   -5

3 3

dtype:
int64 In
```

[3]:

```
# import pandas lib. as pd import
pandas as pd

# create Pandas Series with define indexes  x = pd.Series([10,
20, 30, 40, 50], index =['a', 'b', 'c', 'd', 'e'])
# print the Series print(x)
```

```
a     10

b     20

c     30

d    40

e    50

dtype:
```

int64 In [4]:

```
import pandas as pd
   ind = [10, 20, 30, 40, 50, 60,
70]
   lst = ['G', 'h', 'i',
'j',              'k', 'l',
'm']
# create Pandas Series with define indexes x
= pd.Series(lst, index = ind)

# print the Series print(x)
```

```
10    G
20    h
30    i
40    j
50    k
60    l 70    m
```

```
dtype: object
```

```
import pandas as pd import
numpy as np

# numpy array  data = np.array(['a', 'b',
'c', 'd', 'e'])
# creating series s
=
pd.Series(data) print(s)
```

1.2 Pandas Series From Numpy arrays

```
In [5]:
0    a
1    b
2    c
3    d

4    e

dtype: object In
```

```
[6]:
# importing Pandas & numpy
import pandas as pd import
numpy as np

# numpy array  data = np.array(['a', 'b',
'c', 'd', 'e'])
# creating series
s = pd.Series(data, index =[1000, 1001, 1002, 1003, 1004]) print(s)
```

```
      1000    a
      1001    b
      1002    c
      1003    d1004    e dtype: object In [7]:
```

```
numpy_array = np.array([1, 2.8, 3.0, 2, 9, 4.2])
# Convert NumPy array to Series s
= pd.Series(numpy_array,
index=list('abcdef')) print("Output Series:") print(s)
```

```
Output Series:
a    1.0 b
2.8 c    3.0 d
2.0 e    9.0 f
4.2 dtype:
float64
```

1.3 Pandas Series From Dictionary

In [8]:
```
import pandas as pd

# create a dictionary dictionary = {'D':
10, 'B': 20, 'C': 30}

# create a series series =
pd.Series(dictionary)

print(series)
```

```
D    10
B    20

C    30dtyp e:
     int64 In
```

[9]:
```
# import the pandas lib as pd import
pandas as pd

# create a dictionary dictionary = {'A':
50, 'B': 10, 'C': 80}

# create a series series = pd.Series(dictionary,
index=['B','C','A'])

print(series)
```

```
B    10
C    80

       A     50dtype: int64 In [10]:
```

```
import pandas as pd

# create a dictionary dictionary = {'A':
50, 'B': 10, 'C': 80}

# create a series
series = pd.Series(dictionary, index=['B', 'C', 'D', 'A'])
print(series)
```

```
       B     10.0
       C     80.0
       D     NaNA    50.0 dtype: float64
```

2. Data Manipulation with Pandas Series

```
import pandas as pd import
numpy as np

# creating simple array
data = np.array(['s','p','a','n','d','a','n','a']) ser
=
pd.Series(data,index=[10,11,12,13,14,15,16,17]) print(ser[16])
```

2.1 Indexing

In [11]:

```
n
```

In [12]:
```
import pandas as pd

Date = ['1/1/2018', '2/1/2018', '3/1/2018',
'4/1/2018'] Index_name = ['Day 1', 'Day 2', 'Day 3', 'Day
4']  sr = pd.Series(data = Date,
index = Index_name   ) print(sr)
```

```
Day 1    1/1/2018
Day 2    2/1/2018
Day 3    3/1/2018
Day 4    4/1/2018 dtype:
object
```

In [13]:
```
print(sr['Day 1'])
```

```
1/1/2018 In
```

[14]:
```
import numpy as np import pandas as pd
s=pd.Series(np.arange(5.),index=['a','b','c','d','e']) print(s)
```

```
a    0.0 b
1.0 c    2.0 d
3.0 e    4.0
dtype:
float64
```

```
import numpy as np import pandas as pd
s=pd.Series(np.arange(5.),index=['a','b','c','d','e'])
print(s)
```

2.2 Selecting

In [15]:

```
a    0.0 b
1.0 c    2.0 d
3.0 e    4.0
dtype:
float64
```

In [16]:
```
s['b']
```

Out[16]:

```
1.0
```

In [17]:
```
s[['b','a','d']]
```

Out[17]:
```
b    1.0 a
 0.0 d    3.0
 dtype: float64
```

In [18]:
```
s['b':'e']
```

Out[18]:
```
b    1.0 c
 2.0 d    3.0 e
 4.0 dtype:
 float64
```

In [19]:
```
s[1]
```

```
C:\Users\sivav\AppData\Local\Temp\ipykernel_6176\878419959.py:1: FutureWarning: Series.__getitem__
treati ng keys as positions is deprecated. In a future version, integer keys will always be treated as
labels (c onsistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
s[1]
```

Out[19]:
```
1.0
```

In [20]:
```
s[2:4]
```

```
Out[20]: c    2.0 d
          3.0 dtype:
          float64
In [21]: s[[1,3]]
```

```
C:\Users\sivav\AppData\Local\Temp\ipykernel_6176\363386986.py:1: FutureWarning: Series.__getitem__
treati ng keys as positions is deprecated. In a future version, integer keys will always be treated as
labels (c onsistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
s[[1,3]]
```

```
Out[21]: b    1.0
          d    3.0 dtype:
          float64
```

```
In [22]: print(s[[0, 2, 4]])
```

```
a    0.0 c
2.0 e    4.0
dtype: float64
```

```
C:\Users\sivav\AppData\Local\Temp\ipykernel_6176\430931747.py:1: FutureWarning: Series.__getitem__
treati ng keys as positions is deprecated. In a future version, integer keys will always be treated as
labels (c onsistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
print(s[[0, 2, 4]])
```

```
import numpy as np import pandas as pd
s=pd.Series(np.arange(5.),index=['a','b','c','d','e'])
print(s)
```

2.3 Filtering

```
In [23]:
```

```
a      0.0b
1.0 c    2.0 d
3.0 e    4.0
dtype:
float64 In
```

```
[24]:
```

```
s[s<2]
```

```
Out[24]:
a    0.0
          b      1.0dty pe: float64
```

```
In [25]: s[s>2]
```

```
Out[25]:
d    3.0 e
          4.0 dtype:
          float64
In [26]: s[s!=2]
Out[26]: a    0.0 b
          1.0 d    3.0 e
          4.0 dtype:
          float64
In [27]: s[(s>2)&(s<5) ]
Out[27]: d    3.0 e
          4.0 dtype:
          float64
In [28]: s['b':'c']
Out[28]: b    1.0 c
          2.0 dtype:
          float64
In [29]: print(s[1:2]==5)
```

```
b    False
```

```
dtype: bool

In [30]: s[s.isin([2,4])]

Out[30]: c    2.0 e
          4.0 dtype:
          float64
```

2.4 Arithmetic Operations

```
In [31]: import pandas as pd series1 = pd.Series([1, 2, 3, 4, 5]) series2
= pd.Series([6, 7, 8, 9, 10])

In [32]: series3 = series1 + series2 print(series3)

0     7
1     9
2     11
3     13 4    15

dtype:
int64 In
```

[33]:

```
series3 = series1 - series2 print(series3)

       0     -5
       1     -5
       2     -5
       3     -5 4   -5 dtype: int64 In [34]:

series3 = series1 *series2 print(series3)

       0     6
       1     14
       2     24
       3     36 4    50 dtype: int64 In [35]:

series3 = series1 /series2 print(series3)

       0     0.166667
       1     0.285714
       2     0.375000
       3     0.444444 4    0.500000 dtype: float64 In [36]:

series3 = series1 %series2 print(series3)

0                                                                    11
1                                                                    20
2                                                                    20
3 4 4dtype: int64                                  Error! Bookmark not defined.
```

2.5 Ranking

```
In [37]: import pandas as pd
         s=pd.Series([121,211,153,214,115,116,237,118,219,1
         20])
s.rank(ascending=True)

Out[37]: 0     5.0

1     7.0

2     6.0

3     8.0

4     1.0
```

```
5     2.0
6     10.0
7     3.0
8     9.0
9     4.0
dtype: float64
```

```
In [38]: s.rank(ascending=False)
 Out[38]: 0       6.0
                1       4.0
                2       5.0
                3       3.0
                4       10.0
                5       9.0
                6       1.0
                7       8.0
                8       2.0 9
          7.0 dtype:
          float64
```

```
In [39]: s.rank(method='min')
 Out[39]: 0       5.0
                1       7.0
                2       6.0
                3       8.0
                4       1.0
                5       2.0
                6       10.0
                7       3.0
                8       9.0 9
          4.0 dtype:
          float64
```

```
In [40]: s.rank(method='max')
 Out[40]: 0       5.0
                1       7.0
                2       6.0
                3       8.0
                4       1.0
                5       2.0
                6       10.0
                7       3.0
                8       9.0 9
          4.0 dtype:
          float64
```

```
In [41]: s.rank(method='first')
 Out[41]: 0       5.0
                1       7.0
                2       6.0
                3       8.0
                4       1.0
                5       2.0
                6       10.0
                7       3.0
                8       9.0 9
          4.0 dtype: float64
```

2.6 Sorting

```
In [42]: import pandas as pd  sr = pd.Series([19.5, 16.8,
         22.78, 20.124, 18.1002]) print(sr)
        0    19.5000
        1    16.8000
        2    22.7800
        3    20.1240 4    18.1002 dtype: float64
```

```
In [43]: sr.sort_values(ascending = False)
Out[43]: 2    22.7800
         3    20.1240
         0    19.5000
         4    18.1002 1
         16.8000 dtype:
         float64
In [44]: sr.sort_values(ascending = True)
Out[44]: 1    16.8000 4
         18.1002
         0    19.5000
         3    20.1240 2
         22.7800 dtype:
         float64
In [45]: sr.sort_index()
```

```
Out[45]: 0    19.5000 1
         16.8000
          2      22.7800
          3      20.1240
          4      18.1002 dtype: float64 In [ ]:
```

2.7 checking null values

```
In [48]: s=pd.Series({'ohio':35000,'teyas':71000,'oregon':16000,'utah':5000}) print(s)
         states=['california','ohio','Texas','oregon'
         ] x=pd.Series(s,index=states) print(x)
```

```
ohio      35000
teyas     71000
oregon    16000 utah
5000 dtype: int64
california        NaN
ohio
35000.0 Texas
NaN oregon
16000.0 dtype: float64
```

```
In [49]: x.isnull()
Out[49]: california     True
ohio          False Texas
True oregon         False
dtype: bool In [50]:
x.notnull() Out[50]:
california     False ohio
True Texas          False
oregon          True dtype:
bool
```

2.8 Concatenation

```
In [51]: # creating the Series series1 =
         pd.Series([1, 2, 3]) series2 =
         pd.Series(['A', 'B', 'C'])
```

```
In [52]: # concatenating
display(pd.concat([series1, series2]))
```

```
0    1
1    2
2    3
0    A
1    B2    C
```

dtype:
object In

[53]:
```
display(pd.concat([series1, series2],                    axis
= 1))
```

	0	1
0	1	A
1	2	B
2	3	C

In [54]:
```
display(pd.concat([series1,          ,
series2] axis = 0))
```

```
1
2
3
A
B2    C
```

type: object

```
print(pd.concat([series1, series2], ignore_index=True))
```

```
                                                                11
                                                                18
                                                                18
                                                                18
                                                                19
dtype: object                                                   19
```

```
print(pd.concat([series1, series2], ignore_index=False))
```

```
1
2
3
A
B2    C
```

type: object

```
print(pd.concat([series1, series2], keys=['series1', 'series2']))
```

```
0
1
2
0
1
```

In [55]:

```
0
1
2
3
4
5
```

In [56]:

```
0
```

```
1
2
0
1
```

In [57]:

```
series1  0    1
1    2          2
3
series2  0
A          1
B          2
C dtype: object
```

3 .Creating DataFrames from List and Dictionary

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

# Convert to DataFrame
df = pd.DataFrame(data, columns=['Numbers']) print(df)
```

3.1 From List

In [58]:

```
   Numbers 0
1
1        2
2        3
3        4
4        5
```

In [59]:

```
import pandas as pd nme = ["aparna", "pankaj",
"sudhir", "Geeku"] deg = ["MBA", "BCA",
"M.Tech", "MBA"] scr = [90, 40, 80, 98] dict = {'name':
nme, 'degree': deg, 'score': scr} df =
pd.DataFrame(dict) print(df)
```

```
name  degree  score 0
aparna     MBA     90
        1       pankaj     BCA     40
        2       sudhir  M.Tech     80
        3       Geeku     MBA     98 In [60]:
```

```
import pandas as pd data = [['G', 10],
['h', 15], ['i', 20]]  # Create the pandas Dataframe df
= pd.DataFrame(data, columns = ['Name', 'Age'])  #
print dataframe.
  print(df)
```

```
Name  Age 0    G
10
1    h   15
2    i   20
```

```
df=pd.DataFrame({'a':[4,5,6],'b':[7,8,9],'c':[10,11,12]},index=[1,2,3]) print(df)
```

3.2 From Dictionary

In [61]:

```
   a  b   c 1
4  7  10
2  5  8  11
3  6  9  12
```

In [62]:
```
df=pd.DataFrame({'state':['AP','AP','AP','TS','TS','TS'],'year':[2000,2001,2002,2000,2001,2002],'pop':[1
print(df)
```

```
state  year  pop 0
AP  2000  1.5
       1     AP  2001  1.7
       2     AP  2002  3.6
       3     TS  2000  2.4
       4     TS  2001  2.9
       5     TS  2002  3.2 In [63]:
```

```
df=pd.DataFrame({'a':[4,5,6],'b':[7,8,9]},index=pd.MultiIndex.from_tuples([('d',1),('d',2),('e',2)]
,nam print(df)
```

```
a  b n v
d 1  4  7
2  5  8 e
2  6  9
```

In [64]:
```
df=pd.DataFrame({'ap':{'a':0.0,'c':3.0,'d':6.0},'ts':{'a':1.0,'c':4.0,'d':7.0},'tn':{'a':2.0,'c':5.0,
'd' df.reindex(['a','b','c','d'])
```

Out[64]:

	ap	ts	tn
a	0.0	1.0	2.0
b	NaN	NaN	NaN
c	3.0	4.0	5.0
d	6.0	7.0	8.0

4.Import various file formats to pandas DataFrames and preform the following

4.1 Importing file

In [65]:
```
import pandas as pd
data=pd.read_csv('train.csv')
data
```

Out[65]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S

1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

```
data.head(5)
```

4.2 display top and bottom five rows

In [66]:

Out[66]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [67]: `data.tail(5)`

Out[67]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00	C148	C

890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75	NaN	Q

4.3 Get shape,data type,null values,index and column details

```
In [68]: data.shape
Out[68]: (891, 12)
In [69]: data.dtypes
Out[69]: PassengerId      int64
         Survived         int64
         Pclass           int64
         Name            object
         Sex             object
         Age            float64
         SibSp            int64
         Parch            int64
         Ticket          object
         Fare           float64
         Cabin           object
         Embarked        object
         dtype: object
In [70]: data.isnull().sum()
Out[70]: PassengerId      0
         Survived         0
         Pclass           0
         Name             0
         Sex              0
         Age            177
         SibSp            0
         Parch            0
         Ticket           0
         Fare             0
         Cabin          687
         Embarked         2
         dtype: int64
In [71]: data.columns

Out[71]: Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
                'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],       dtype='object')

In [72]: data.index

Out[72]: RangeIndex(start=0, stop=891, step=1)
```

4.4

Select/Delete the records rows/columns based on conditions

```
In [ ]:

In [73]: data.drop([0,3])
```

Out[73]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	male	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

889 rows × 12 columns

In [74]:
```
data.drop(data[data['Fare']>4.3].index)
```

	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked

Out[74]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch				
179	180	0	3	Leonard, Mr. Lionel	male	36.0	0	0	LINE	0.0000	NaN	S
263	264	0	1	Harrison, Mr. William	male	40.0	0	0	112059	0.0000	B94	S
271	272	1	3	Tornquist, Mr. William Henry	male	25.0	0	0	LINE	0.0000	NaN	S
277	278	0	2	Parkes, Mr. Francis "Frank"	male	NaN	0	0	239853	0.0000	NaN	S
302	303	0	3	Johnson, Mr. William Cahoone Jr	male	19.0	0	0	LINE	0.0000	NaN	S
378	379	0	3	Betros, Mr. Tannous	male	20.0	0	0	2648	4.0125	NaN	C
413	414	0	2	Cunningham, Mr. Alfred Fleming	male	NaN	0	0	239853	0.0000	NaN	S
466	467	0	2	Campbell, Mr. William	male	NaN	0	0	239853	0.0000	NaN	S
481	482	0	2	Frost, Mr. Anthony Wood "Archie"	male	NaN	0	0	239854	0.0000	NaN	S
597	598	0	3	Johnson, Mr. Alfred	male	49.0	0	0	LINE	0.0000	NaN	S
633	634	0	1	Parr, Mr. William Henry Marsh	male	NaN	0	0	112052	0.0000	NaN	S
674	675	0	2	Watson, Mr. Ennis Hastings	male	NaN	0	0	239856	0.0000	NaN	S
732	733	0	2	Knight, Mr. Robert J	male	NaN	0	0	239855	0.0000	NaN	S

Out[75]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
806	807	0	1	Andrews, Mr. Thomas Jr	male	39.0	0	0	112050	0.0000	A36	S
815	816	0	1	Fry, Mr. Richard	male	NaN	0	0	112058	0.0000	B102	S
822	823	0	1	Reuchlin, Jonkheer. John George	male	38.0	0	0	19972	0.0000	NaN	S

In [ ]:

In [ ]:

```
data
```

4.5 Sorting and Ranking operations in DataFrame

In [75]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...

Out[76]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch				
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

75

In [76]:
```
data.sort_index(ascending=False)
```

Out[77]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C

Out[78]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S

891 rows × 12 columns

In [ ]:

In [77]:
```
data.sort_values(by=['Sex','Name']).head(6)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
279	280	1	3	Abbott, Mrs. Stanton (Rosa Hunt)	female	35.0	1	1	C.A. 2673	20.2500	NaN	S
874	875	1	2	Abelson, Mrs. Samuel (Hannah Wizosky)	female	28.0	1	0	P/PP 3381	24.0000	NaN	C
40	41	0	3	Ahlin, Mrs. Johan (Johanna Persdotter Larsson)	female	40.0	1	0	7546	9.4750	NaN	S
855	856	1	3	Aks, Mrs. Sam (Leah Rosen)	female	18.0	0	1	392091	9.3500	NaN	S
730	731	1	1	Allen, Miss. Elisabeth Walton	female	29.0	0	0	24160	211.3375	B5	S
297	298	0	1	Allison, Miss. Helen Loraine	female	2.0	1	2	113781	151.5500	C22 C26	S

In [78]:
```
data.rank().head(10)
```

Out[78]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1.0	275.0	646.0	109.0	603.0	218.0	713.0	339.5	672.0	77.0	NaN	567.5
1	2.0	720.5	108.5	191.0	157.5	532.0	713.0	339.5	780.0	789.0	111.0	84.5
2	3.0	720.5	646.0	354.0	157.5	310.5	304.5	339.5	875.0	232.5	NaN	567.5

Out[79]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch				
3	4.0	720.5	108.5	273.0	157.5	488.5	713.0	339.5	67.5	748.0	70.5	567.5
4	5.0	275.0	646.0	16.0	603.0	488.5	304.5	339.5	609.0	264.0	NaN	567.5
9	10.0	720.5	308.5	577.0	157.5	74.5	713.0	339.5	185.5	658.5	NaN	84.5

```
data.rank().head(2)u
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1.0	275.0	646.0	109.0	603.0	218.0	713.0	339.5	672.0	77.0	NaN	567.5
1	2.0	720.5	108.5	191.0	157.5	532.0	713.0	339.5	780.0	789.0	111.0	84.5
5	6.0	275.0	646.0	555.0	603.0	NaN	304.5	339.5	370.0	293.0	NaN	207.0
6	7.0	275.0	108.5	516.0	603.0	668.5	304.5	339.5	124.0	734.5	176.0	567.5
7	8.0	275.0	646.0	625.0	603.0	19.5	853.5	737.5	513.5	532.5	NaN	567.5
8	9.0	720.5	646.0	413.0	157.5	328.5	304.5	836.5	459.0	366.0	NaN	567.5

In [79]:

Out[79]:

In [80]:
```
data.rank(ascending=False).head(5)
```

Out[80]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	891.0	617.0	246.0	783.0	289.0	497.0	179.0	552.5	220.0	815.0	NaN	322.5
1	890.0	171.5	783.5	701.0	734.5	183.0	179.0	552.5	112.0	103.0	94.0	805.5
2	889.0	171.5	246.0	538.0	734.5	404.5	587.5	552.5	17.0	659.5	NaN	322.5
3	888.0	171.5	783.5	619.0	734.5	226.5	179.0	552.5	824.5	144.0	134.5	322.5
4	887.0	617.0	246.0	876.0	289.0	226.5	587.5	552.5	283.0	628.0	NaN	322.5

3) Data cleaning and preparation

Import any csv file to pandas data frame and perform the following

a)Handle missing data by detecting,dropping and replacing/filling missing

```
import pandas as pd import
numpy as np
```

values

In [75]:

The Titanic dataset is a famous dataset used in machine learning for binary classification tasks, where the goal is to predict whether a passenger survived or not based on various features. This dataset comes from the tragic sinking of the RMS Titanic in 1912, where many lives were lost.

Dataset Description:

The Titanic dataset typically contains the following columns: PassengerId: Unique identifier for each passenger.

Survived: Binary outcome (0 = No, 1 = Yes), indicating if the passenger survived.

Pclass: Passenger class (1 = First, 2 = Second, 3 = Third).

Name: The full name of the passenger.

Sex: Gender of the passenger (male/female).

Age: Age of the passenger.

SibSp: Number of siblings.

Parch: Number of parents or children aboard the Titanic.

Ticket: The ticket number.

Fare: The fare paid for the ticket.

Cabin: The cabin number (if available).

In [76]:
```
# Load the CSV file into a Pandas
DataFrame df = pd.read_csv('train.csv') df
```

Out[76]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S

1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	male	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

In [77]:
```
# Display the first few rows of the DataFrame to understand the data
print("Original DataFrame:") print(df.head())
```

```
Original DataFrame:
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1                            Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
                             2 Heikkinen, Miss. Laina  female  26.0      0
3                            Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                            Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
         0      0         A/5 21171   7.2500   NaN        S
         1      0          PC 17599  71.2833   C85        C
         2      0  STON/O2. 3101282   7.9250   NaN        S
         3      0            113803  53.1000  C123        S
         4      0            373450   8.0500   NaN        S  In [78]:
```

```
# 1. Detect missing data missing_data
= df.isnull() print("\nMissing
Data:") print(missing_data.head(10))
```

```
Missing Data:
   PassengerId  Survived  Pclass   Name    Sex    Age  SibSp  Parch  Ticket
\ 0        False     False   False  False  False  False  False  False   False
1       False     False   False  False  False  False  False  False   False
2       False     False   False  False  False  False  False  False   False
3       False     False   False  False  False  False  False  False   False
4       False     False   False  False  False  False  False  False   False
5       False     False   False  False  False   True  False  False   False
6       False     False   False  False  False  False  False  False   False
7       False     False   False  False  False  False  False  False   False
8       False     False   False  False  False  False  False  False   False
9       False     False   False  False  False  False  False  False   False

    Fare  Cabin  Embarked
0  False   True     False
1  False  False     False
2  False   True     False
3  False  False     False
4  False   True     False
5  False   True     False
6  False  False     False
7  False   True     False
8  False   True     False
9  False   True     False
```

In [79]:
```
# No of null values
n=df.isnull().sum()
n
```

Out[79]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
```

```
Ticket          0
Fare            0
Cabin         687
Embarked        2 dtype:
int64
```

```
In [80]: # 2. Drop rows with missing values df_dropna = df.dropna()
         print("\nDataFrame after dropping rows with missing values:")
         print(df_dropna.head(10))
```

```
DataFrame after dropping rows with missing values:
    PassengerId  Survived  Pclass  \ 1
2         1       1
3             4         1       1
6             7         0       1
10           11         1       3
11           12         1       1
21           22         1       2
23           24         1       1
27           28         0       1
52           53         1       1
54           55         0       1

                                                 Name     Sex   Age  SibSp
\ 1   Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
3        Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
6                             McCarthy, Mr. Timothy J    male  54.0      0
10                          Sandstrom, Miss. Marguerite Rut  female   4.0      1
11                          Bonnell, Miss. Elizabeth  female  58.0      0
21                             Beesley, Mr. Lawrence    male  34.0      0
23                       Sloper, Mr. William Thompson    male  28.0      0
27                     Fortune, Mr. Charles Alexander    male  19.0      3
52           Harper, Mrs. Henry Sleeper (Myna Haxtun)  female  49.0      1
54                     Ostby, Mr. Engelhart Cornelius    male  65.0      0

    Parch    Ticket      Fare        Cabin Embarked
1       0  PC 17599   71.2833          C85        C
3       0    113803   53.1000         C123        S
6       0     17463   51.8625          E46        S
10      1   PP 9549   16.7000           G6        S
11      0    113783   26.5500         C103        S
21      0    248698   13.0000          D56        S
23      0    113788   35.5000           A6        S
27      2     19950  263.0000  C23 C25 C27        S
52      0  PC 17572   76.7292          D33        C
54      1    113509   61.9792          B30        C
```

```
In [81]: # 3. Fill missing values with a specific value (e.g., mean, median, or custom value)
         # Let's fill missing values in the 'Age' column with the mean value of that
         column mean_chas = df['Age'].mean() df_fillna = df.fillna({'Age': mean_chas})
         print("\nDataFrame after filling missing values:") print(df_fillna.head(10))
```

```
DataFrame after filling missing values:
PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3
5            6         0       3
6            7         0       1
7            8         0       3
8            9         1       3
9            10         1       2

                                                Name     Sex        Age  \
0                                Braund, Mr. Owen Harris    male  22.000000
```

```
1                                Cumings, Mrs. John Bradley (Florence Briggs Th...  female
                                 38.000000
2                                Heikkinen, Miss. Laina  female  26.000000
3                                Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.000000
4                                Allen, Mr. William Henry    male  35.000000
5                                Moran, Mr. James    male  29.699118
6                                McCarthy, Mr. Timothy J    male  54.000000
7                                Palsson, Master. Gosta Leonard    male   2.000000
8                                Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)  female
                                 27.000000
9                                Nasser, Mrs. Nicholas (Adele Achem)  female  14.000000

   SibSp  Parch            Ticket     Fare Cabin Embarked
0      1      0         A/5 21171   7.2500   NaN        S
1      1      0          PC 17599  71.2833   C85        C
2      0      0  STON/O2. 3101282   7.9250   NaN        S
3      1      0            113803  53.1000  C123        S
4      0      0            373450   8.0500   NaN        S
5      0      0            330877   8.4583   NaN        Q
6      0      0             17463  51.8625   E46        S
7      3      1            349909  21.0750   NaN        S
8      0      2            347742  11.1333   NaN        S
9      1      0            237736  30.0708   NaN        C  In [82]:
```

```
# 4. Replace missing values conditionally
# For example, replace missing values in 'City' with 'Unknown'
df_replace = df.fillna({'LSTAT': 'Unknown'}) print("\nDataFrame
after replacing missing values:") print(df_replace.head())
```

```
DataFrame after replacing missing values:
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1                            Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                            Heikkinen, Miss. Laina  female  26.0      0
3                            Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                            Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S  In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
# Load the CSV file into a Pandas DataFrame
# Replace 'data.csv' with the actual file path if needed
df=pd.read_csv('test.csv')
```

b)transform data using apply() and map() method

```
In [83]:
```

In [84]:
```
# Display the first few rows of the DataFrame to understand the data
print("Original DataFrame:") print(df.head())
```

```
Original DataFrame:
   PassengerId  Pclass                                          Name     Sex
\ 0          892       3                              Kelly, Mr. James    male
1          893       3              Wilkes, Mrs. James (Ellen Needs)  female
2          894       2                     Myles, Mr. Thomas Francis    male
3          895       3                              Wirz, Mr. Albert    male
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)  female

    Age  SibSp  Parch   Ticket     Fare Cabin Embarked
        0     34.5      0      0   330911   7.8292   NaN        Q
        1     47.0      1      0   363272   7.0000   NaN        S
        2     62.0      0      0   240276   9.6875   NaN        Q
        3     27.0      0      0   315154   8.6625   NaN        S
        4     22.0      1      1  3101298  12.2875   NaN        S  In [85]:
```

```
# Assume 'Age' is a column that we want to transform

# 1. Transform using apply() method
# Let's square the values in the 'Age' column df['squared_age'] =
df['Age'].apply(lambda x: x ** 2) df
```

Out[85]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	squared
0	892	3	Kelly, Mr.	male	34.5	0	0	330911	7.8292	NaN	Q	11
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S	22
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q	
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S	7
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S	4
...	...	...	...	...	...	...	...	...	...	...	...	
413	1305	3	Spector, Mr. Woolf	male	NaN	0	0	A.5. 3236	8.0500	NaN	S	
414	1306	1	Oliva y Ocana, Dona. Fermina	female	39.0	0	0	PC 17758	108.9000	C105	C	15
415	1307	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.2500	NaN	S	14
416	1308	3	Ware, Mr. Frederick	male	NaN	0	0	359309	8.0500	NaN	S	
417	1309	3	Peter, Master. Michael J	male	NaN	1	1	2668	22.3583	NaN	C	

384

418 rows × 12 columns

In [86]:
```
# 2. Transform using map() method
# Let's map a new column 'Price_category' based on the 'Price' values
Age_category_map = {0: 'Low', 1: 'Medium', 2: 'High'} df['Age_category']
= df['Age'].map(Age_category_map)
```

In [87]:
```
# Display the transformed DataFrame print("\nDataFrame
after transformation:") print(df.head())
```

```
DataFrame after transformation:
   PassengerId  Pclass                                          Name     Sex
\ 0          892       3                              Kelly, Mr. James    male
1          893       3              Wilkes, Mrs. James (Ellen Needs)  female
2          894       2                     Myles, Mr. Thomas Francis    male
3          895       3                              Wirz, Mr. Albert    male
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)  female

    Age  SibSp  Parch   Ticket     Fare Cabin Embarked  squared_age
\ 0  34.5      0      0   330911   7.8292   NaN        Q      1190.25
1  47.0      1      0   363272   7.0000   NaN        S      2209.00
2  62.0      0      0   240276   9.6875   NaN        Q      3844.00
3  27.0      0      0   315154   8.6625   NaN        S       729.00    4 22.0      1      1  3101298
   12.2875   NaN        S       484.00

  Age_category
               0          NaN
```

```
1          NaN
2          NaN
3          NaN
4          NaN  In [ ]:
```

In [ ]:

In [ ]:

c)Detect and filter outliers

#Dataset Description:, The Abalone dataset contains 8 attributes (or features) and a target variable (rings). Below is a breakdown of the features: Sex: Categorical variable (M = Male, F = Female). Length: Continuous variable (in mm) – Longest shell measurement. Diameter: Continuous variable (in mm) – Diameter perpendicular to length. Height: Continuous variable (in mm) – Height of the abalone with its meat inside. Whole weight: Continuous variable (in grams) – Weight of the whole abalone. Shucked weight: Continuous variable (in grams) – Weight of the meat only. Shell weight: Continuous variable (in grams) – Weight of the dried shell.

In [88]:
```
pip install ucimlrepo
```

```
Requirement already satisfied: ucimlrepo in
c:\users\sivav\appdata\local\programs\python\python312\lib\si te-packages (0.0.7) Requirement
already satisfied: pandas>=1.0.0 in c:\users\sivav\appdata\local\programs\python\python312\li
b\site-packages (from ucimlrepo) (2.2.2) Requirement already satisfied: certifi>=2020.12.5 in
c:\users\sivav\appdata\local\programs\python\python3
12\lib\site-packages (from ucimlrepo) (2024.6.2) Requirement already satisfied: numpy>=1.26.0
in c:\users\sivav\appdata\local\programs\python\python312\li b\site-packages (from
pandas>=1.0.0>ucimlrepo) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in
c:\users\sivav\appdata\local\programs\python\pyt hon312\lib\site-packages (from
pandas>=1.0.0>ucimlrepo) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
c:\users\sivav\appdata\local\programs\python\python312\lib
\site-packages (from pandas>=1.0.0->ucimlrepo) (2024.1) Requirement already satisfied:
tzdata>=2022.7 in c:\users\sivav\appdata\local\programs\python\python312\l ib\site-packages
(from pandas>=1.0.0>ucimlrepo) (2024.1)
Requirement already satisfied: six>=1.5 in
c:\users\sivav\appdata\local\programs\python\python312\lib\sit e-packages (from pythondateutil>=2.8.2-
>pandas>=1.0.0->ucimlrepo) (1.16.0) Note: you may need to restart the kernel to use updated packages.
WARNING: Ignoring invalid distribution ~ip
(C:\Users\sivav\AppData\Local\Programs\Python\Python312\Lib\si te-packages)
WARNING: Ignoring invalid distribution ~ip
(C:\Users\sivav\AppData\Local\Programs\Python\Python312\Lib\si te-packages)
WARNING: Ignoring invalid distribution ~ip
```

```
from ucimlrepo import fetch_ucirepo

# fetch dataset  abalone =
fetch_ucirepo(id=1)
# data (as pandas dataframes)
X = abalone.data.features y =
abalone.data.targets

# metadata
print(abalone.metadata)


# variable information
print(abalone.variables)
```

```
(C:\Users\sivav\AppData\Local\Programs\Python\Python312\Lib\si te-packages)
```

In [89]:

```
{'uci_id': 1, 'name': 'Abalone', 'repository_url': 'https://archive.ics.uci.edu/dataset/1/abalone',
'data _url': 'https://archive.ics.uci.edu/static/public/1/data.csv', 'abstract': 'Predict the age of
 abalone fr om physical measurements', 'area': 'Biology', 'tasks': ['Classification', 'Regression'],
'characteristic s': ['Tabular'], 'num_instances': 4177, 'num_features': 8, 'feature_types':
['Categorical', 'Integer', 'R eal'], 'demographics': [], 'target_col': ['Rings'], 'index_col': None,
'has_missing_values': 'no', 'missi ng_values_symbol': None, 'year_of_dataset_creation': 1994,
'last_updated': 'Mon Aug 28 2023', 'dataset_do i': '10.24432/C55C7W', 'creators': ['Warwick Nash',
'Tracy Sellers', 'Simon Talbot', 'Andrew Cawthorn', 'Wes Ford'], 'intro_paper': None,
'additional_info': {'summary': 'Predicting the age of abalone from phys ical measurements.  The age of
abalone is determined by cutting the shell through the cone, staining it, and counting the number of
rings through a microscope -- a boring and time-consuming task.  Other measure ments, which are easier
to obtain, are used to predict the age.  Further information, such as weather pat terns and location
(hence food availability) may be required to solve the problem.\r\n\r\nFrom the origin al data examples
with missing values were removed (the majority having the predicted value missing), and the ranges of
the continuous values have been scaled for use with an ANN (by dividing by 200).', 'purpos e': None,
'funded_by': None, 'instances_represent': None, 'recommended_data_splits': None,
'sensitive_dat a': None, 'preprocessing_description': None, 'variable_info': 'Given is the attribute
name, attribute typ e, the measurement unit and a brief description.  The number of rings is the value
to predict: either as a continuous value or as a classification problem.\r\n\r\nName / Data Type /
Measurement Unit / Descripti on\r\n-----------------------------\r\nSex / nominal / -- / M, F, and I
(infant)\r\nLength / continuous / mm / Longest shell measurement\r\nDiameter\t/ continuous / mm /
perpendicular to length\r\nHeight / conti nuous / mm / with meat in shell\r\nWhole weight / continuous
/ grams / whole abalone\r\nShucked weight / continuous\t / grams / weight of meat\r\nViscera weight /
continuous / grams / gut weight (after bleedin
g)\r\nShell weight / continuous / grams / after being dried\r\nRings / integer / -- / +1.5 gives the
 age in years\r\n\r\nThe readme file contains attribute statistics.', 'citation': None}}           name
 role         type demographic  \ 0             Sex  Feature  Categorical        None
1          Length  Feature   Continuous        None
2          Diameter  Feature   Continuous        None
3          Height  Feature   Continuous        None
4          Whole_weight  Feature   Continuous        None
5          Shucked_weight  Feature   Continuous        None
6          Viscera_weight  Feature   Continuous        None
7          Shell_weight  Feature   Continuous        None   8           Rings   Target      Integer
           None

                  description  units missing_values
0               M, F, and I (infant)   None             no
1               Longest shell measurement     mm             no
2               perpendicular to length     mm             no
3               with meat in shell     mm             no
4               whole abalone  grams             no
5               weight of meat  grams             no
6               gut weight (after bleeding)  grams             no
7               after being dried  grams             no
8               +1.5 gives the age in years   None             no
```

```
In [90]: df=X df.head()
```

Out[90]:

	Sex	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_weight
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055

```
In [91]: column_name = 'Whole_weight'
```

```
In [92]: z_scores = np.abs((df[column_name] - df[column_name].mean()) / df[column_name].std())
         z_scores.head(10)
```

```
Out[92]: 0    0.641821
         1    1.230130
```

```
2    0.309432
3    0.637743
4    1.271933
5    0.973191
6    0.104493
7    0.123865
8    0.650998
9    0.134093
Name: Whole_weight, dtype: float64
```

```
In [93]: # Define a threshold for outliers (e.g., z-score greater than
         3) z_score_threshold = 3

# Filter the DataFrame to keep rows without outliers filtered_df =
df[z_scores <= z_score_threshold]
```

```
In [94]: # Display the DataFrame after filtering outliers
print("\nDataFrame after filtering outliers:")
print(filtered_df.head())
```

```
DataFrame after filtering outliers:
  Sex  Length  Diameter  Height  Whole_weight  Shucked_weight  Viscera_weight  \
0   M   0.455     0.365   0.095        0.5140          0.2245          0.1010
1   M   0.350     0.265   0.090        0.2255          0.0995          0.0485
2   F   0.530     0.420   0.135        0.6770          0.2565          0.1415
3   M   0.440     0.365   0.125        0.5160          0.2155          0.1140
4   I   0.330     0.255   0.080        0.2050          0.0895          0.0395

   Shell_weight
0         0.150
1         0.070
2         0.210
3         0.155
4         0.055  In [95]:
```

```
# Display the first few rows of the DataFrame to understand the
data print("Original DataFrame:") print(df.head())
```

```
Original DataFrame:
  Sex  Length  Diameter  Height  Whole_weight  Shucked_weight  Viscera_weight  \
0   M   0.455     0.365   0.095        0.5140          0.2245          0.1010
1   M   0.350     0.265   0.090        0.2255          0.0995          0.0485
2   F   0.530     0.420   0.135        0.6770          0.2565          0.1415
3   M   0.440     0.365   0.125        0.5160          0.2155          0.1140
4   I   0.330     0.255   0.080        0.2050          0.0895          0.0395

   Shell_weight
0         0.150
1         0.070
2         0.210
3         0.155
4         0.055  In [96]:
```

In [98]:
```
# Select the column to analyze for outliers (replace 'Value' with the actual column name)
column_name = 'total'
```

In [99]:
```
# Define a threshold for outliers (e.g., z-score greater than 3) z_score_threshold
= 3

# Filter the DataFrame to keep rows without outliers filtered_df
= df[z_scores <= z_score_threshold]

# Display the DataFrame after filtering outliers print("\nDataFrame
after filtering outliers:") print(filtered_df.head())
```

```
DataFrame after filtering outliers:
  Sex  Length  Diameter  Height  Whole_weight  Shucked_weight  Viscera_weight  \
0   M   0.455     0.365   0.095        0.5140          0.2245          0.1010
```

```
1   M   0.350     0.265   0.090        0.2255          0.0995          0.0485
2   F   0.530     0.420   0.135        0.6770          0.2565          0.1415
3   M   0.440     0.365   0.125        0.5160          0.2155          0.1140   4   I   0.330
    0.255   0.080        0.2050          0.0895          0.0395

   Shell_weight
0         0.150
1         0.070
2         0.210
3         0.155
4         0.055
```

d) perform vectorized string operations on pandas series

Dataset Description: The IMDb dataset consists of the following key components: Reviews: The dataset contains 50,000 movie reviews in text format. Each review reflects the opinions of moviegoers, ranging from very positive to very negative. Sentiment Labels: Each review is labeled as either: Positive (represented as 1) Negative (represented as 0) The dataset is evenly balanced with 25,000 positive reviews and 25,000 negative reviews. Training and Testing Sets: Training Set: 25,000 reviews for training models. Testing Set: 25,000 reviews for testing the model's performance. Both sets are balanced with equal numbers of positive and negative reviews. Review Length: The reviews vary in length, from a few sentences to multiple paragraphs, offering a range of complexity for text processing.

```
In [100… # Load the CSV file into a Pandas DataFrame df
         = pd.read_csv('IMDB Dataset.csv') df
```

Out[100…

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
...	...	...
49995	I thought this movie did a down right good job...	positive
49996	Bad plot, bad dialogue, bad acting, idiotic di...	negative
49997	I am a Catholic taught in parochial elementary...	negative
49998	I'm going to have to disagree with the previou...	negative
49999	No one expects the Star Trek movies to be high...	negative

50000 rows × 2 columns

```
In [101… # Assuming 'Name' is the column containing strings

         # Convert all names to uppercase df['Name_uppercase']=
         df['sentiment'].str.upper() df
```

Out[101…

	review	sentiment	Name_uppercase
0	One of the other reviewers has mentioned that ...	positive	POSITIVE
1	A wonderful little production. The...	positive	POSITIVE

2	I thought this was a wonderful way to spend ti...	positive	POSITIVE
3	Basically there's a family where a little boy ...	negative	NEGATIVE
4	Petter Mattei's "Love in the Time of Money" is...	positive	POSITIVE
...	...	...	...
49995	I thought this movie did a down right good job...	positive	POSITIVE
49996	Bad plot, bad dialogue, bad acting, idiotic di...	negative	NEGATIVE
49997	I am a Catholic taught in parochial elementary...	negative	NEGATIVE
49998	I'm going to have to disagree with the previou...	negative	NEGATIVE
49999	No one expects the Star Trek movies to be high...	negative	NEGATIVE

50000 rows × 3 columns

```
In [102… # Calculate the length of each name
df['Name_length'] = df['sentiment'].str.len() df
```

Out[102…

	review	sentiment	Name_uppercase	Name_length
0	One of the other reviewers has mentioned	positive	POSITIVE	8 that ...
1	A wonderful little production. The...	positive	POSITIVE	8
2	I thought this was a wonderful way to spend ti...	positive	POSITIVE	8
3	Basically there's a family where a little boy ...	negative	NEGATIVE	8
4	Petter Mattei's "Love in the Time of Money" is...	positive	POSITIVE	8
...	...	...	...	...
49995	I thought this movie did a down right good job...	positive	POSITIVE	8
49996	Bad plot, bad dialogue, bad acting, idiotic di...	negative	NEGATIVE	8
49997	I am a Catholic taught in parochial elementary...	negative	NEGATIVE	8
49998	I'm going to have to disagree with the previou...	negative	NEGATIVE	8
49999	No one expects the Star Trek movies to be high...	negative	NEGATIVE	8

50000 rows × 4 columns

```
In [103… # Split the names based on a delimiter (e.g., space) and create a new column for the first part of the
          na df['First_name'] = df['sentiment'].str.split(' ').str[0] df
```

Out[103…

	review	sentiment	Name_uppercase	Name_length	First_name

0	One of the other reviewers has mentioned that ...	positive	POSITIVE	8	positive
1	A wonderful little production. The...	positive	POSITIVE	8	positive
2	I thought this was a wonderful way to spend ti...	positive	POSITIVE	8	positive
3	Basically there's a family where a little boy ...	negative	NEGATIVE	8	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive	POSITIVE	8	positive
...	...	...	...	...	...
49995	I thought this movie did a down right good job...	positive	POSITIVE	8	positive
49996	Bad plot, bad dialogue, bad acting, idiotic di...	negative	NEGATIVE	8	negative
49997	I am a Catholic taught in parochial elementary...	negative	NEGATIVE	8	negative
49998	I'm going to have to disagree with the previou...	negative	NEGATIVE	8	negative
49999	No one expects the Star Trek movies to be high...	negative	NEGATIVE	8	negative

50000 rows × 5 columns

4) Data Wrangling

1.concate/join/merge/reshape data frames

CONCATE

Used to concatenate two or more DataFrame objects. By setting axis=0 it concatenates vertically (rows), and by setting axis=1 it concatenates horizontally (columns).

In [3]:
```
import pandas as pd df1 = pd.DataFrame({'A': [1, 2],# Column 'A' with
values 'A0', 'A1'
 'B': [3, 4]})# Column 'B' with values 'B0', 'B1'
 # Create the second DataFrame (df2) with columns 'A' and 'B' and two rows df2
= pd.DataFrame({'A':[ 5, 6],
 'B': [7,8]})
# Concatenate df1 and df2 vertically (axis=0) to stack rows
# This combines the two DataFrames by adding the rows of df2 below the rows of df1
result = pd.concat([df1, df2], axis=0)
```

In [4]: df1

Out[4]:

	A	B
0	1	3
1	2	4

In [5]: df2

Out[5]:

	A	B
0	5	7
1	6	8

In [6]: result

Out[6]:

	A	B
0	1	3
1	2	4
0	5	7
1	6	8

MERGE

Used to merge two data frames based on a key column, similar to SQL joins. Options include how='inner', how='outer', how='left', and how='right' for different types of joins.

In [8]:
```
import pandas as pd #
Create DataFrame
1
df1 = pd.DataFrame({'Rollno': ['1a', '2a', '3a'], 'value1': [1, 2, 3]})
# Create DataFrame 2 df2 = pd.DataFrame({'Rollno': ['2a', '3a', '4a'],
'value2': [4, 5, 6]})
# Merge DataFrames on 'key' column using inner join
result = pd.merge(df1, df2, on='Rollno', how='inner')
```

In [9]: df1

Out[9]:

	Rollno	value1

0	1a	1
1	2a	2
2	3a	3

In [10]: df2

Out[10]:

	Rollno	value2
0	2a	4
1	3a	5
2	4a	6

In [11]: result

Out[11]:

	Rollno	value1	value2

```
import pandas as pd #
Create DataFrame
1
df1 = pd.DataFrame({'Rollno1': ['1a', '2a', '3a'], 'value1': [1, 2, 3]})
# Create DataFrame 2 df2 = pd.DataFrame({'Rollno2': ['2a', '3a', '4a'],
'value2': [4, 5, 6]})
# Merge DataFrames on 'key' column using inner join #
Merge DataFrames on specified keys using inner join
result = pd.merge(df1, df2, left_on='Rollno1', right_on='Rollno2', how='inner')
```

0	2a	2	4
1	3a	3	5

In [13]:

In [14]: df1

Out[14]:

	Rollno1	value1
0	1a	1
1	2a	2
2	3a	3

In [15]: df2

Out[15]:

	Rollno2	value2
0	2a	4
1	3a	5
2	4a	6

In [16]: result Out[16]:

	Rollno1	value1	Rollno2	value2

Rollno2 value2

```
import pandas as pd
# Create DataFrame
1
df1 = pd.DataFrame({'Rollno1': ['1a', '2a', '3a'], 'value1': [1, 2, 3]})
# Create DataFrame 2 df2 = pd.DataFrame({'Rollno2': ['2a', '3a', '4a'],
'value2': [4, 5, 6]})
# Merge DataFrames on 'key' column using inner join #
Merge DataFrames on specified keys using inner join
result = pd.merge(df1, df2, left_on='Rollno1', right_on='Rollno2', how='inner')
```

0	2a	2	2a	4
1	3a	3	3a	5

In [19]:

```
# Reshape the result using pivot
reshaped_result = result.pivot(index='Rollno1', columns='Rollno2',
values=['value1','value2']) reshaped_result
```

Out[19]:

	value1		**value2**	
Rollno2	**2a**	**3a**	**2a**	**3a**
Rollno1				
2a	2.0	NaN	4.0	NaN
3a	NaN	3.0	NaN	5.0

JOIN

A join is a way to combine data from two or more tables (or DataFrames) based on a common column, known as the join key.

In [20]:

```
import pandas as pd #
Create DataFrame
1
df1 = pd.DataFrame({"1A": [1, 2, 3], "1B":
[4,5,6]}, index=["K0", "K1", "K2"]) # Create
DataFrame 2 df2 = pd.DataFrame({"1C":
[7,8,9], "1D":
[10,11,12]}, index=["K0", "K2", "K3"]) # Print
DataFrame 1 print(df1)
# Print DataFrame 2
print(df2)
# Join DataFrames 1 and 2 on index
(default) df3 = df1.join(df2) print(df3)
```

```
1A  1B K0   1
4
K1   2   5
K2   3   6
1C  1D K0
7  10
K2   8  11
K3   9  12
    1A  1B   1C    1D K0
1   4  7.0  10.0
K1   2   5  NaN   NaN
K2   3   6  8.0  11.0
```

INNER JOIN:

Returns rows with matching keys in both DataFrames.

In [21]:
```
#inner join
df4 = df1.join(df2, how='inner') print(df4)
```

```
1A  1B  1C  1D K0   1
4   7  10
K2   3   6   8  11
```

FULL OUTER JOIN:

Returns all rows from both DataFrames.

In [22]:
```
# full outer join      df5 =
f1.join(df2, how='outer')
print(df5)
```

```
1A   1B   1C    1D K0
1.0  4.0  7.0  10.0
K1  2.0  5.0  NaN   NaN
K2  3.0  6.0  8.0  11.0
K3  NaN  NaN  9.0  12.0
```

LEFT OUTER JOIN:

Returns all rows from the left DataFrame and matching rows from the right DataFrame.

In [23]:
```
#left outer join
df6 = df1.join(df2, how='left') print(df6)
```

```
1A  1B   1C    1D K0   1
4  7.0  10.0
K1   2   5  NaN   NaN
K2   3   6  8.0  11.0
```

RIGHT OUTER JOIN

Returns all rows from the right DataFrame and matching rows from the left DataFrame.

In [24]:
```
#right outer join
df7 = df1.join(df2, how='right') print(df7)
```

```
1A   1B  1C  1D K0  1.0
4.0   7  10
K2  3.0  6.0   8  11
K3  NaN  NaN   9  12
```

RESHAPE

Reshaping functions like pivot and melt are used to transform the layout of data frames.

In [25]:
```
import pandas as pd # Create Series 1 s1 = pd.Series([0,
1, 2, 3], index=['a', 'b', 'c', 'd'])
# Create Series 2 s2 = pd.Series([4, 5, 6],
index=['c', 'd', 'e']) # Concatenate Series into
DataFrame df = pd.concat([s1, s2], keys=['one',
'two']) print(df)
```

```
one  a    0
b    1
c    2
d    3 two
c    4
d    5
e    6
dtype:
int64
```

In [ ]:
```
print(df.unstack())
```

```
a    b    c    d    e one  0.0
1.0  2.0  3.0  NaN two  NaN  NaN
4.0  5.0  6.0
```

In [ ]:
```
#reshaping import pandas as pd import numpy as np
data=pd.DataFrame(np.arange(6).reshape((2,3)),index=pd.Index(['apple','cherry'],name='fruit'),columns=pd
data
```

Out[ ]:

color	red	green	blue
fruit			
apple	0	1	2
cherry	3	4	5

In [ ]:
```
result=data.stack()
result
```

Out[ ]:

		0
fruit	**color**	
apple	**red**	0
	green	1
	blue	2
cherry	**red**	3
	green	4
	blue	5

dtype: int64

In [ ]:
```
result.unstack(0)
```

Out[ ]:

color	fruit	apple	cherry
red	0	3	
green	1	4	
blue	2	5	

In [ ]:
```
result.unstack('fruit')
```

Out[ ]:

	fruit	apple	cherry
color			
red	0	3	
green	1	4	
blue	2	5	

2.Read dataframe to create a pivot table

Pivot tables help summarize and analyze large datasets by:

1. Grouping data by specific columns
2. Aggregating values using functions like sum, mean, count
3. Creating customized views of data

In [ ]:
```
import pandas as pd #
Sample DataFrame data
= {
 'A': ['foo', 'foo', 'foo', 'bar', 'bar'],
'B': ['one', 'one', 'two', 'two', 'one'],
 'C': [1, 2, 3, 4, 5]
}
df = pd.DataFrame(data)

# Create a pivot table
pivot_table = pd.pivot_table(df, values='C', index='A', columns='B',
aggfunc='sum') pivot_table
```

Out[ ]:

B	one	two
A		
bar	5	4
foo	3	3

3.Read dataframe to create a cross table

A cross table (or contingency table) displays the relationship between two categorical variables.

In [ ]:
```
import pandas as pd #
Sample DataFrame data
= {
 'Category': ['A', 'B', 'A', 'B', 'A'],
'Status': ['Yes', 'No', 'Yes', 'Yes', 'No']
}
df = pd.DataFrame(data) #
Create a cross table
cross_table = pd.crosstab(index=df['Category'], columns=df['Status']) cross_table
```

Out[ ]:

Category	Status	No	Yes

A	1	2
B	1	1

5) Plotting and Visualization

Data visualization using Matplotlib begins with importing a sample dataset into Pandas. Matplotlib is a versatile library that enables the creation of a wide variety of plots to represent data visually.

Line Plot: A line plot is used to represent data points connected by lines, which is useful for visualizing trends over a continuous interval. You can create a line plot using `plot()` function from Matplotlib, typically showing relationships over time.

Bar Plot: Bar plots are used to display categorical data with rectangular bars. The length of each bar is proportional to the value it represents. The `bar()` function is used to generate a bar plot, which is ideal for comparing quantities among different groups.

Histogram: A histogram visualizes the distribution of a dataset by grouping data into bins and plotting the frequency of data points in each bin. This helps in understanding the distribution of continuous data, and is created using `hist()`.

Density Plot: A density plot is a smoothed version of a histogram that shows the probability density of the variable. It gives insights into the underlying distribution. Matplotlib can achieve this with `kdeplot()` from the Seaborn library, which integrates with Matplotlib.

Scatter Plot: A scatter plot is used to observe the relationship between two continuous variables. Each point represents an observation in the dataset, and the `scatter()` function is used to create this plot, making it useful for identifying correlations or clusters.

These plots help to analyze the underlying patterns in the data, providing insights for decision-making and further analysis.

5)Data Visualization

1.Data Visualization on any Simple dataset using matplotlib for following.

```
pip install plot
```

a. Line Plot

In [15]:

```
Collecting plot
  Downloading plot-0.6.5-py2.py3-none-any.whl.metadata (1.5 kB) Requirement already satisfied:
matplotlib in c:\users\sivav\appdata\local\programs\python\python312\lib\s ite-packages (from
plot) (3.9.1)
Collecting typing (from plot)
  Downloading typing-3.7.4.3.tar.gz (78 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done' Requirement already satisfied:
numpy in c:\users\sivav\appdata\local\programs\python\python312\lib\site-p ackages (from plot)
(1.26.4) Requirement already satisfied: scipy in
c:\users\sivav\appdata\local\programs\python\python312\lib\site-p ackages (from plot) (1.14.0)
Requirement already satisfied: pyyaml in
c:\users\sivav\appdata\local\programs\python\python312\lib\sitepackages (from plot) (6.0.1)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\sivav\appdata\local\programs\python\python312
\lib\site-packages (from matplotlib->plot) (1.2.1) Requirement
already satisfied: cycler>=0.10 in
c:\users\sivav\appdata\local\programs\python\python312\lib
\site-packages (from matplotlib->plot) (0.12.1) Requirement
already satisfied: fonttools>=4.22.0 in
c:\users\sivav\appdata\local\programs\python\python31
2\lib\site-packages (from matplotlib->plot) (4.53.1) Requirement
already satisfied: kiwisolver>=1.3.1 in
c:\users\sivav\appdata\local\programs\python\python31
2\lib\site-packages (from matplotlib->plot) (1.4.5) Requirement
already satisfied: packaging>=20.0 in
c:\users\sivav\appdata\local\programs\python\python312
\lib\site-packages (from matplotlib->plot) (23.2) Requirement already satisfied: pillow>=8 in
c:\users\sivav\appdata\local\programs\python\python312\lib\si te-packages (from matplotlib->plot)
(10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\sivav\appdata\local\programs\python\python312
\lib\site-packages (from matplotlib->plot) (3.1.2) Requirement already satisfied: python-
dateutil>=2.7 in c:\users\sivav\appdata\local\programs\python\pytho n312\lib\site-packages (from
matplotlib->plot) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in
c:\users\sivav\appdata\local\programs\python\python312\lib\sit e-packages (from python-
dateutil>=2.7>matplotlib->plot) (1.16.0)
Downloading plot-0.6.5-py2.py3-none-any.whl (135 kB)
Building wheels for collected packages: typing
  Building wheel for typing (setup.py): started
  Building wheel for typing (setup.py): finished with status 'done'
  Created wheel for typing: filename=typing-3.7.4.3-py3-none-any.whl size=26327
sha256=2770d47646d777286a d0ce92d352ceda0f8baa5f1fd737d79a7651a5e9df9443   Stored
in directory:
c:\users\sivav\appdata\local\pip\cache\wheels\12\98\52\2bffe242a9a487f00886e43b8ed
8dac46456702e11a0d6abef
Successfully built typing
Installing collected packages: typing, plot
Successfully installed plot-0.6.5 typing-3.7.4.3 Note: you may need
to restart the kernel to use updated packages.
WARNING: Ignoring invalid distribution ~ip
(C:\Users\sivav\AppData\Local\Programs\Python\Python312\Lib\si te-packages)
WARNING: Ignoring invalid distribution ~ip
(C:\Users\sivav\AppData\Local\Programs\Python\Python312\Lib\si te-packages)
```

```
WARNING: Ignoring invalid distribution ~ip
(C:\Users\sivav\AppData\Local\Programs\Python\Python312\Lib\si te-packages)
```

In [1]:
```
import pandas as pd import matplotlib.pyplot
as plt df=pd.read_csv('exp5.csv') df
```

Out[1]:

Number

	Index	Highest Charting Position	Number of Times Charted	Week of Highest Charting	Song Name	Streams	Artist	Artist Followers	Song ID
0	1	1	8	2021-07-23- - 202107-30	Beggin'	4,86,33,449	Måneskin	3377762	3Wrjm47oTz2sjIgck11l5e
1	2	2	3	2021-07-23- - 20210730	STAY (with Justin Bieber)	4,72,48,719	The Kid LAROI	2230022	5HCyWIXZPP0y6Gqq8TgA20
2	3	1	11	2021-06-25- - 20210702	good 4 u	4,01,62,559	Olivia Rodrigo	6266514	4ZtFanR9U6ndgddUvNcjcG
3	4	3	5	2021-07-02- - 20210709	Bad Habits	3,77,99,456	Ed Sheeran	83293380	6PQ88X9TkUIAUIZJHW2upE
4	5	5	1	2021-07-23- - 20210730	INDUSTRY BABY (feat. Jack Harlow)	3,39,48,454	Lil Nas X	5473565	27NovPIUIRrOZoCHxABJwK
...	...	...	...	...	...	...	...	...	...
1551	1552	195	1	2019-12-27- - 20200103	New Rules	46,30,675	Dua Lipa	27167675	2ekn2ttSfGqwhhate0LSR0
1552	1553	196	1	2019-12-27- - 20200103	Cheirosa Ao Vivo	46,23,030	Jorge & Mateus	15019109	2PWjKmjyTZeDpmOUa3a5da
1553	1554	197	1	2019-12- - 202001-03	Havana (feat. Young Thug)	46,20,876	Camila Cabello	22698747	1rfofaqEpACxVEHIZBJe6W

1554	1555	198	1	2019-12-27--20200103	Surtada - Remix Brega Funk	46,07,385	Dadá Boladão, Tati Zaqui, OIK	208630		5F8ffc8KWKNawllr5WsW0r
1555	1556	199	1	2019-12-27--20200103	Lover (Remix) [feat. Shawn Mendes]	45,95,450	Taylor Swift	42227614		3i9UVldZOE0aD0JnyfAZZ0

1556 rows × 23 columns

A line plot is a type of graph that displays data points connected by straight lines to show trends over time or ordered categories.

In [2]:
```
s=df.head(5)
s
```

Out[2]:

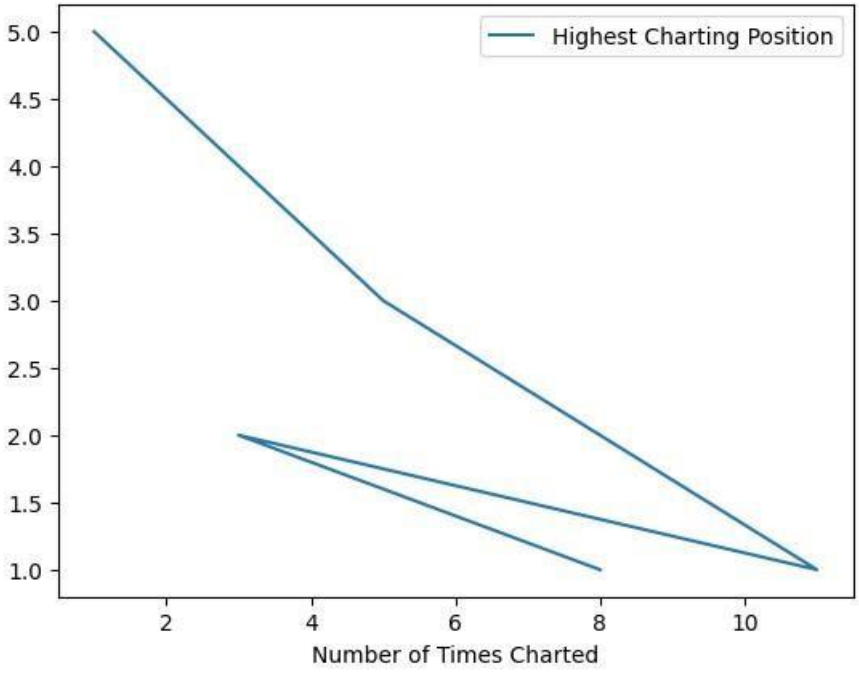
Number

Index	Highest Charting Position	Number of Times Charted	Week of Highest Charting	Song Name	Streams	Artist	Artist Followers	Song ID	
0	1	1	8	2021-07-23--202107-30	Beggin'	4,86,33,449	Måneskin	3377762	3Wrjm47oTz2sJlgck11l5e	
1	2	2	3	2021-07-23--20210730	STAY (with Justin Bieber)	4,72,48,719	The Kid LAROI	2230022	5HCyWlXZPP0y6Gqq8TgA20	[
2	3	1	11	2021-06-25--202107-02	good 4 u	4,01,62,559	Rodrigo Olivia	6266514	4ZtFanR9U6ndgddUvNcjcG	
3	4	3	5	2021-07-02--20210709	Bad Habits	3,77,99,456	Ed Sheeran	83293380	6PQ88X9TkUIAUIZJHW2upE	[
4	5	5	1	2021-07-23--20210730	INDUSTRY BABY (feat. Jack Harlow)	3,39,48,454	Lil Nas X	5473565	27NovPIUIRrOZoCHxABJwK	

'au
h 'p
h
'p 5 rows × 23 columns

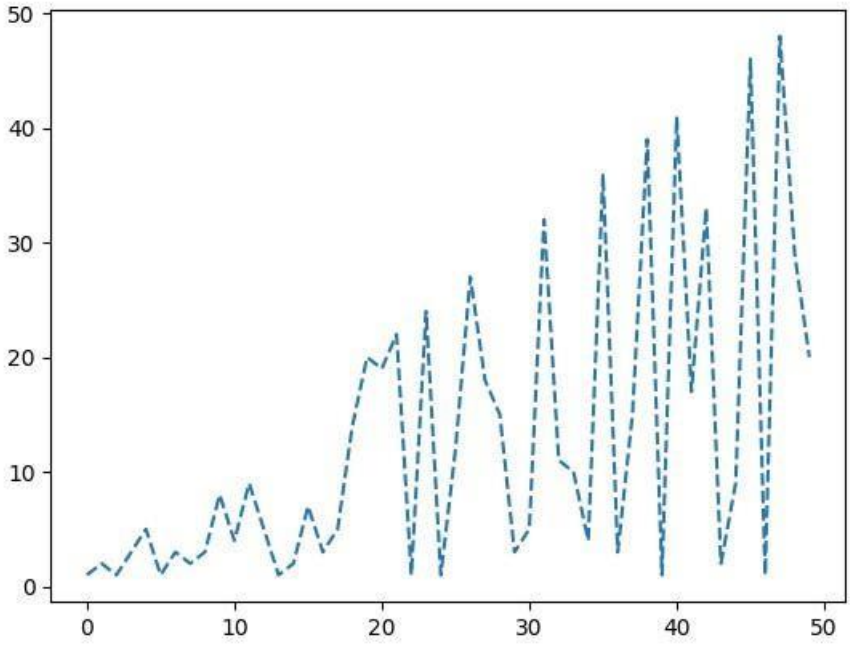
In [3]:
```
s.plot.line(x='Number of Times Charted',y='Highest Charting Position')
```

Out[3]: <Axes: xlabel='Number of Times Charted'>



In [4]:
```
plt.plot(df['Highest Charting Position'].iloc[:50],ls="-")
```

Out[4]: [<matplotlib.lines.Line2D at 0x221ab618b90>]

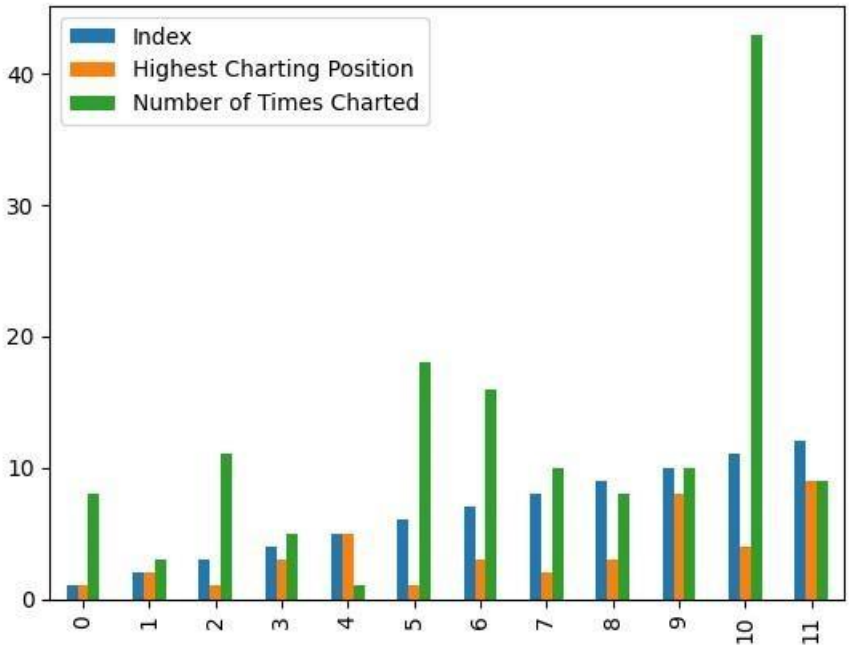


b. Bar Plot

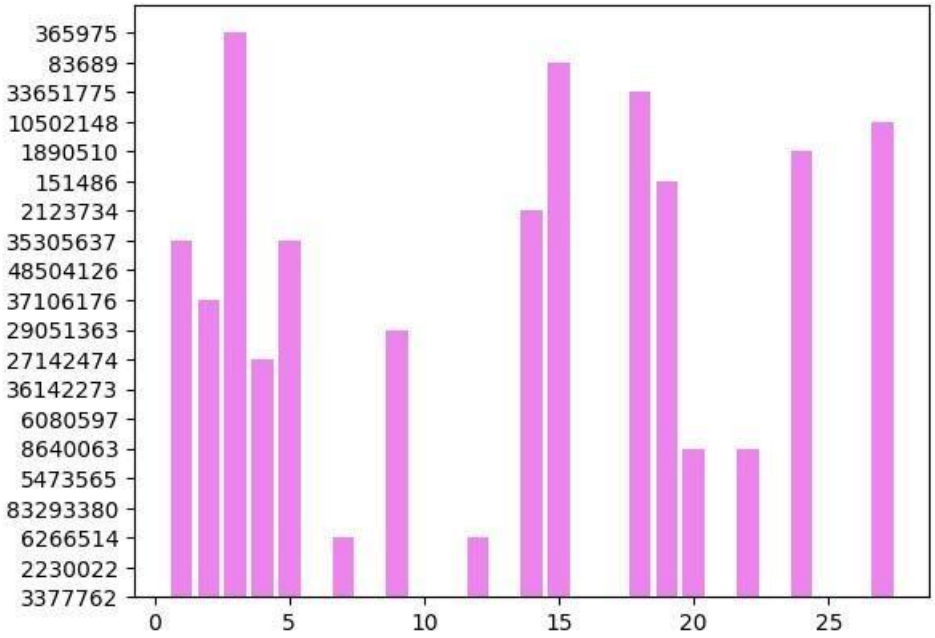
Barplot: A bar plot displays categorical data with rectangular barsrepresenting the frequency or value of each category.

```
In [5]: df1=pd.read_csv('exp5.csv')
        df2=df1.head(12) df2.plot.bar()
```

```
Out[5]: <Axes: >
```




```
In [6]: plt.bar(df1['Highest Charting Position'].iloc[:30],df1['Artist Followers'].iloc[:30],color='violet')
        plt.show()
```



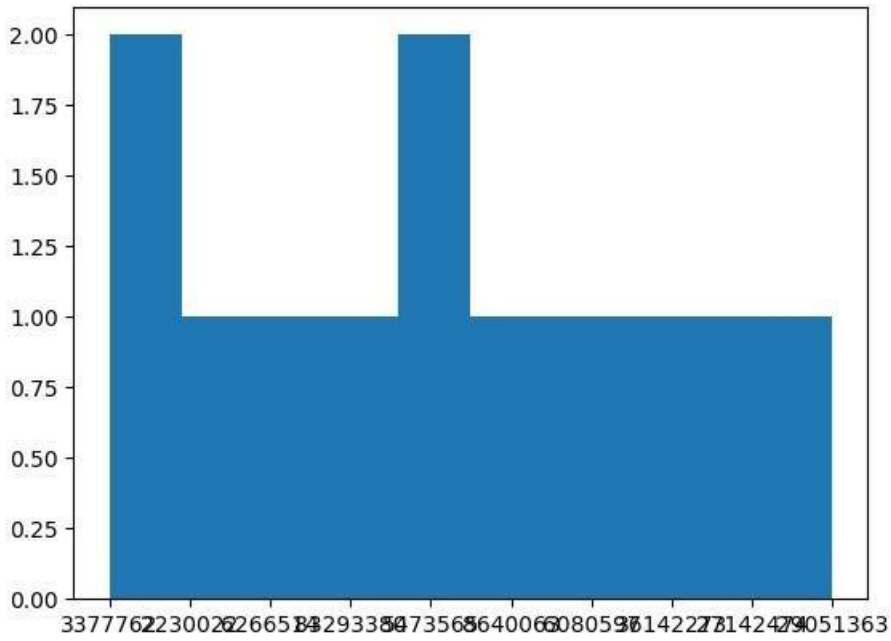
c.Histogram

A histogram displays the distribution of numerical data using bars to represent frequency within intervals.

```
In [9]: plt.hist(df2['Artist Followers'])
```

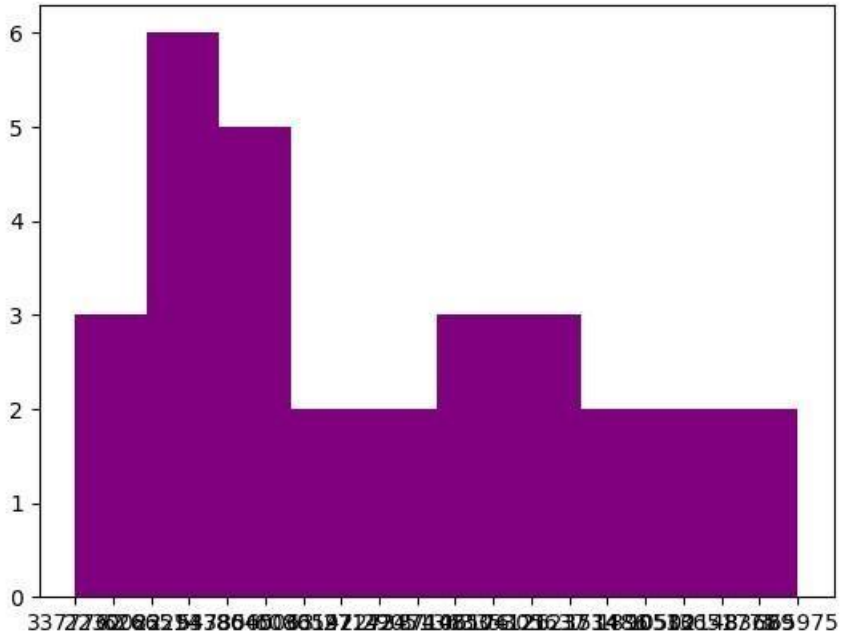
```
Out[9]: (array([2., 1., 1., 1., 2., 1., 1., 1., 1., 1.]),  array([0. ,
         0.9, 1.8, 2.7, 3.6, 4.5, 5.4, 6.3, 7.2, 8.1, 9. ]),
```

<BarContainer object of 10 artists>)



In [10]:
```
plt.hist(df1['Artist Followers'].iloc[:30],color='purple')
```

Out[10]: (array([3., 6., 5., 2., 2., 3., 3., 2., 2., 2.]), array([0. , 1.9, 3.8, 5.7, 7.6, 9.5, 11.4, 13.3, 15.2, 17.1, 19.]), <BarContainer object of 10 artists>)

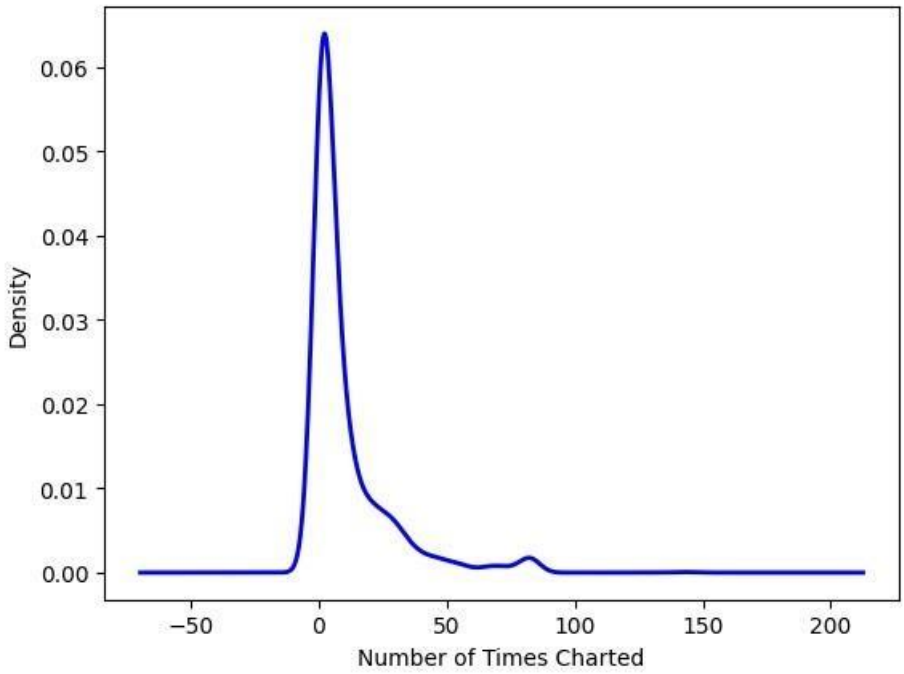


d.Density Plot

A density plot shows the distribution of a continuous variable by estimating its probability density.

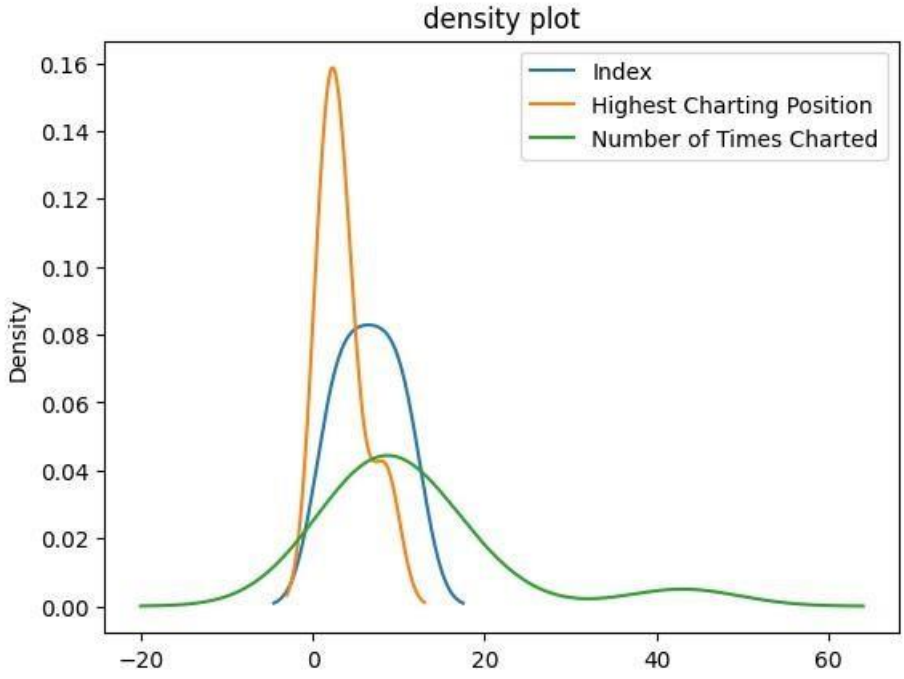
In [17]:
```
df1['Number of Times Charted'].plot.density(color='blue', linewidth=2)
plt.xlabel('Number of Times Charted')
```

Out[17]: Text(0.5, 0, 'Number of Times Charted')



In [18]:
```
df2.plot.density() plt.title('density
plot')
```

Out[18]: Text(0.5, 1.0, 'density plot')

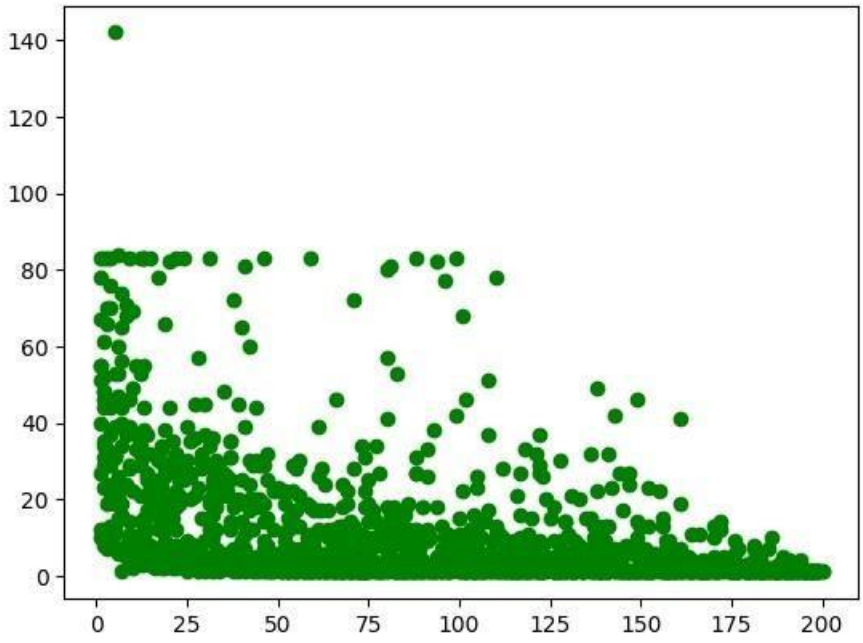


e.Scatter Plot

A scatter plot visualizes the relationship between two numerical variables using points on a Cartesian plane.

In [22]:
```
plt.scatter(df1['Highest Charting Position'].iloc[50:],df1['Number of Times
Charted'].iloc[50:],color='g
```

Out[22]: <matplotlib.collections.PathCollection at 0x221c925b140>



In [ ]: