Numpy Arrays and Vectorized Computation

NumPy is a powerful library in Python used for numerical computations. It's like the backbone of scientific computing in Python, enabling efficient operations on large arrays and matrices. It also comes with a suite of mathematical functions to perform complex calculations. It's widely used in data analysis, machine learning, and even in fields like physics and engineering

Numpy Arrays from Python DataStructures,Intrinsic Numpy Objects and Random Functions

1. 1 Arrays from python datastructures

In [2]:
```
# converting list to numpy array
import numpy as np
a=[1,2,3,4,5,6]
b=np.array(a)
print(a)
```

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

In [30]:
```
# converting two 1D arrays into one 2D array
import numpy as np
x=[1,2,7,3]
y=[3,4,6,5]
z=np.array((x,y))
print(z)
```

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

In [5]:
```
# list to tuple
import numpy as np
a=(1,2,3,4,5,1)
c=np.array(a)
print(c)
```

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

In [33]:
```
#converting list to set
a=[1,2,3,4,5,5]
c=set(a)
np.array(c)
```

Out[33]:
```
array({1, 2, 3, 4, 5}, dtype=object)
```

In [34]:
```
# converting dictonary to list
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)
```

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

1.2 Intirinsic Numpy Objects

In [6]:
```
# creating ndarray using arange function
a=np.array(np.arange(9))
print(a)
```

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

In [7]:
```
# generates list of specified zeros
a=np.zeros(3)
print(a)
```

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

In [8]:
```
# generates 2D array of zeros
b=np.zeros([3,3])
print(b)
```

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

In [12]:
```
# # generates list of specified ones
a=np.ones(4)
print(a)
```

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

In [3]:
```
# generates 2D array of ones
b=np.ones(3)
print(b)
```

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

In [11]:
```
# generates 2D array of having ones in the diagonal
a=np.eye(3)
print(a)
```

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

In [7]:
```
# shifting diagonal ones one step right
c=np.eye(3,k=1)
print(c)
```

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

In [15]:
```
# works same as eye() method
a=np.identity(3)
print(a)
```

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

In [8]:
```
# fills with specified number by specified dimentions
d=np.full([2,2],7)
print(d)
```

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

In [14]:
```
# generates zeros of specified dimentions
a=np.empty((2,3))
print(a)
```

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

In [18]:
```
# generates with the list of items and
np.diag([1,2,3,4])
```

Out[18]:
```
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

In [13]:
```
# crates a meshgrid for gives list of items
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

In [14]:
```
# gives a random number below the number specified
#from numpy import random
import numpy.random
x = random.randint(2)
print(x)
```

```
0
```

In [16]:
```
# generates 2D array with true and false
a=random.choice(['a','b'],size=[2,3])
print(a)
```

```
[['a' 'a' 'a']
 ['b' 'a' 'a']]
```

In [27]:
```
x = np.random.rand(3) + np.random.rand(1)*1j
print (x)
print(x.real)
print(x.imag)
```

```
[0.19029124+0.75740658j 0.34153475+0.75740658j 0.4581672 +0.75740658j]
[0.19029124 0.34153475 0.4581672 ]
[0.75740658 0.75740658 0.75740658]
```

In [23]:
```
# gives a complex number based on specified size
x = np.random.rand(1,5) + random.rand(1,5)*1j
print (x)
```

```
[[0.56912285+0.99074578j 0.97494973+0.74973799j 0.63415417+0.29802275j
  0.98001741+0.99542674j 0.69150049+0.12513674j]]
```

In [24]:
```
#generates complex numbne
np.random.random(size=(2,2))+1j*np.random.random(size=(2,2))
```

Out[24]:
```
array([[0.14736012+0.57136733j, 0.97241982+0.15471679j],
       [0.42027952+0.52003045j, 0.56276305+0.61909801j]])
```

In [25]:
```
# gives numbers is below the specified number in random order
np.random.permutation(5)
```

Out[25]:
```
array([3, 0, 4, 1, 2])
```

In [26]:
```
a=np.array(5)
b=np.random.choice(a,size=5,p=[0.1,0.2,0.3,0.2,0.2])
print(b)
```

```
[4 4 2 2 0]
```

In [43]:
```
# returns a number between the specified range
np.random.randint(1,5)
```

Out[43]:
```
4
```

In [31]:
```
a=np.random.randn(2,5)
print(a)
```

```
[[-0.40183995 -2.22976328 -0.42769238 -1.57968881  0.38336317]
 [ 0.42777609  0.56719638  0.02836425 -1.72142945 -0.34330569]]
```

In [32]:
```
a=np.array(['apple','bananaa','cherry'])
b=np.random.choice(a)
print(b)
```

```
apple
```

In [30]:
```
np.random.shuffle(a)
print(a)
```

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

2.Manipulation Of Numpy Arrays

2.1 Indexing

In [15]:
```python
# accessing values from 1D array
a=np.arange(19)
print(a[9])
```

```
9
```

In [14]:
```python
# accessing values from 2D index
x = np.array([[1, 2], [3, 4], [5, 6]])
print(x[0,1])
```

```
2
```

In [36]:
```python
arr= np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
arr[0,0,2]
```

Out[36]:
```
3
```

In [46]:
```python
# making copy of array
old_values = arr[0].copy()
arr[0] = 42
print(arr)
```

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

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

In [47]:
```python
# performing addtion by accessing values using indexes
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[2] + arr[3])
```

```
7
```

In [50]:
```python
#accessing vlaues form 2D array
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print( arr[0, 1])
```

```
2
```

In [51]:
```python
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print( arr[1, 4])
```

```
10
```

In [52]:
```python
#accessing vlaues form 2D array
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0, 1, 2])
```

```
6
```

In [53]:
```python
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print( arr[1, -1])
```

```
10
```

2.2 Slicing

In [54]:
```python
# accessing data from between purticular range
import numpy as np
arr=np.array([5,6,7,8,9])
print(arr[1:3])
```

```
[6 7]
```

In [55]:
```
# form index 1 to end
import numpy as np
arr=np.array([5,6,7,3,6,8,9])
print(arr[1:])
```

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

In [56]:
```
# form starting to index 3
arr=np.array([5,6,7,8,9])
print(arr[:3])
```

```
[5 6 7]
```

In [39]:
```
# accessing though negetive index
arr=np.array([5,6,7,8,9])
print(arr[-3:-1])
```

```
[7 8]
```

In [49]:
```
# start : stop : step
arr=np.array([5,6,7,8,4,5,6,7,9])
print(arr[1:5:2])
```

```
[6 8]
```

In [50]:
```
arr=np.array([5,6,7,8,4,5,6,7,9])
print(arr[-1:-5:-1])
```

```
[9 7 6 5]
```

In [52]:
```
# accessign data by combining slicing and range functions
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
```

```
[7 8 9]
```

In [1]:
```
# accessign data by combining slicing and range functions
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:2, 2])
```

```
[3 8]
```

In [44]:
```
# accessing from more than one index using slicing
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:3, 1:4])
```

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

In [9]:
```
# accessing data from string
b = "DSP Lab"
print(b[2:5])
```

```
P L
```

In [10]:
```
b = "DSP Lab"
print(b[:5])
```

```
DSP L
```

In [11]:
```
b = "DSP Lab"
print(b[2:])
```

```
P Lab
```

2.3 Re-Shaping

In [46]:
```
# gives dimentions
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)
```

```
(2, 4)
```

In [60]:
```
# chage the dimentions
import numpy as np
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 [61]:
```
# converting 1D to 2D
import numpy as np
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 [4]:
```
import numpy as np
a=np.arange(8)
print(a.reshape(4,2))
```

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

In [5]:
```
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

In [75]:
```
# concatenating arrays
a1=np.arange(6).reshape(3,2)
a2=np.arange(6).reshape(3,2)
print(a1)
print(a2)
print(np.concatenate((a1,a2)))
print(np.concatenate((a1,a2),axis=1))
```

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

In [73]:
```
# joining using stack function
print(np.stack((a1,a2)))
```

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

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

In [50]:
```python
# Join two 2-D arrays along rows (axis=1)
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
arr = np.concatenate((arr1, arr2), axis=1)
print(arr)
```

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

In [8]:
```python
# NumPy provides a helper function: hstack() to stack along rows
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.concatenate((arr1, arr2))
print(arr)
```

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

In [68]:
```python
# NumPy provides a helper function: vstack()  to stack along columns
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.vstack((arr1, arr2))
print(arr)
```

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

In [72]:
```python
# NumPy provides a helper function: dstack() to stack along height, which is the same as depth.
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.dstack((arr1, arr2))
print(arr)
```

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

2.5 Splitting

In [79]:
```python
import numpy as np
a = np.arange(9)
print(a)
```

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

In [80]:
```python
# splitting one array into specified arrays
b = np.split(a,3)
print(b)
```

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

In [10]:
```python
# accessing data based on the row number
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15], [16, 17, 18]])
newarr = np.array_split(arr, 3, axis=0)
print(newarr)
```

```
[array([[1, 2, 3],
       [4, 5, 6]]), array([[ 7,  8,  9],
       [10, 11, 12]]), array([[13, 14, 15],
       [16, 17, 18]])]
```

In [11]:
```python
# accessing data based on the row number
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15], [16, 17, 18]])
newarr = np.array_split(arr, 1, axis=1)
print(newarr)
```

```
[array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12],
       [13, 14, 15],
       [16, 17, 18]])]
```

In [65]:
```python
#Use the hsplit() method to split the 2-D array into three 2-D arrays along rows.
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15], [16, 17, 18]])
newarr = np.hsplit(arr, 3)
print(newarr)
```

```
[array([[ 1],
       [ 4],
       [ 7],
       [10],
       [13],
       [16]]), array([[ 2],
       [ 5],
       [ 8],
       [11],
       [14],
       [17]]), array([[ 3],
       [ 6],
       [ 9],
       [12],
       [15],
       [18]])]
```

3.Computation On Numpy Arrays Using Universal Functions

3.1 Statistical functions

In [81]:
```
arr = np.array([11,22,33,44,55,66,77,88,99])
print(np.min(arr))
```

```
11
```

In [13]:
```
# minimum and maximum
print(np.amin(arr), np.amax(arr))
```

```
5 9
```

In [14]:
```
# range of weight i.e. max weight-min weight
print(np.ptp(arr))
```

```
4
```

In [15]:
```
# mean
print(np.mean(weight))
```

```
55.0
```

In [16]:
```
# median
print(np.median(weight))
```

```
55.0
```

In [17]:
```
# standard deviation
print(np.std(weight))
```

```
28.401877872187722
```

In [18]:
```
# variance
print(np.var(weight))
```

```
806.6666666666666
```

In [19]:
```
# average
print(np.average(weight))
```

```
55.0
```

3.2 Bit-twiddling functions

In [83]:
```
even = np.array([0, 2, 4, 6, 8, 16, 32])
odd = np.array([1, 3, 5, 7, 9, 17, 33])
```

In [21]:
```
# bitwise_and
print(np.bitwise_and(even, odd))
```

```
[ 0  2  4  6  8 16 32]
```

In [23]:
```
# bitwise_or
print(np.bitwise_or(even, odd))
```

```
[ 1  3  5  7  9 17 33]
```

In [24]:
```
# bitwise_xor
print(np.bitwise_xor(even, odd))
```

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

In [84]:
```
# invert or not
print(np.invert(odd))
```

```
[ -2  -4  -6  -8 -10 -18 -34]
```

In [26]:
```
# left_shift
print(np.left_shift(even, 1))
```

```
[ 0  4  8 12 16 32 64]
```

In [27]:
```
# right_shift
print(np.right_shift(even, 1))
```

```
[ 0  1  2  3  4  8 16]
```

3.3 Unary Universal Functions

In [86]:
```
arr = np.arange(10)
print(arr)
```

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

In [29]:
```
# square root of given list of elements
np.sqrt(arr)
```

Out[29]:
```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])
```

In [87]:
```
# give exponential of all elements in the input array
np.exp(arr)
```

Out[87]:
```
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 [31]:
```
# min value
np.min(arr)
```

Out[31]:
```
0
```

In [32]:
```
# max element
np.max(arr)
```

Out[32]:
```
9
```

In [33]:
```
# average of all elements
np.average(arr)
```

Out[33]:
```
4.5
```

In [38]:
```
# absolute values of all elements
print(np.abs(arr))
```

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

In [14]:
```
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]
```

3.3 Binary Universal Functions

In [15]:
```
x = np.random.randn(8)
y = np.random.randn(8)
```

In [ ]:
```
# gives random specified number of values
```

In [16]:
```
print(x)
```

```
[ 1.9029113   0.37516745 -1.30605534  0.40233125 -0.52921987  0.50879897
 -0.14657609  0.76597139]
```

In [21]:
```
print(y)
```

```
[-0.08739821  0.55924299 -0.60581813  1.59797719  0.12302027 -0.37407141
 -0.84599114  0.47792473]
```

In [22]:
```
np.maximum(x, y)
```

Out[22]:
```
array([-0.08739821,  0.55924299,  1.08833746,  1.59797719,  0.16382473,
       -0.36369696, -0.84599114,  0.50381589])
```

In [25]:
```
arr = np.random.randn(7) * 5
remainder, whole_part = np.modf(arr)
print(remainder)
```

```
[ 0.54703048  0.09623633 -0.31652868 -0.09286155  0.88671909  0.16826515
  0.47786293]
```

In [26]:
```
print(whole_part)
```

```
[ 4.  3. -0. -1.  0.  9.  9.]
```

In [13]:
```
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 [14]:
```
np.subtract(a,b)
```

Out[14]:
```
array([[-10,  -9,  -8],
       [ -7,  -6,  -5],
       [ -4,  -3,  -2]])
```

In [17]:
```
a=np.array([1,2,3])
b=np.array([4,5,6])
```

In [18]:
```
np.multiply(a,b)
```

Out[18]:
```
array([ 4, 10, 18])
```

In [4]:
```
import numpy as np
a = np.array([10,100,1000])
np.power(a,2)
```

Out[4]:
```
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

In [49]:
```
a = np.array([[3,7,5],[8,4,3],[2,4,9]])
a
```

Out[49]:
```
array([[3, 7, 5],
       [8, 4, 3],
       [2, 4, 9]])
```

In [50]:
```
# gives sum of all elements
a.sum()
```

Out[50]:
```
45
```

In [19]:
```
# gives percentile of a list for a given level
a = np.array([[30,40,70],[80,20,10],[50,90,60]])
np.percentile(a,90)
```

Out[19]: 82.0

In [20]:
```python
# gives mean
a.mean()
```

Out[20]: 50.0

In [71]:
```python
# mean based on axis
arr.mean(axis=1)
```

Out[71]: array([1., 4., 7.])

In [64]:
```python
# gives median
np.median(arr)
```

Out[64]: 4.0

In [19]:
```python
# standerd deviation
np.std(arr)
```

Out[19]: 0.8542443496205637

In [66]:
```python
# varience
np.var(arr)
```

Out[66]: 6.666666666666667

In [60]:
```python
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 [61]:
```python
print(arr.cumprod(axis=1))
```

```
[[  0   0   0]
 [  3  12  60]
 [  6  42 336]]
```

4.2 Comparison Operations

In [ ]:
```python
# It results either true or flase based on the specified condition
```

In [73]:
```python
a=np.array([[1,2],[3,4]])
b=np.array([[1,2],[3,4]])
print(np.array_equal(a,b))
```

```
True
```

In [79]:
```python
a=np.array([1,15,6,8])
b=np.array([11,12,6,4])
```

In [80]:
```python
print(np.greater(a,b))
```

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

In [89]:
```python
print(np.greater_equal(a,b))
```

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

In [82]:
```python
print(np.less(a[0],b[2]))
```

```
True
```

In [83]:
```python
print(np.less(a,b))
```

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

In [84]:
```python
print(np.less_equal(a,b))
```

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

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

5.1 Sorting

In [22]:
```
import numpy as np
a = np.array([[3,7],[9,1]])
print(a)
```

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

In [23]:
```
# gives sorted list
np.sort(a)
```

Out[23]:
```
array([[3, 7],
       [1, 9]])
```

In [26]:
```
# sort based on the axis
```

In [25]:
```
np.sort(a,axis=0)
```

Out[25]:
```
array([[3, 1],
       [9, 7]])
```

In [93]:
```
np.sort(a,axis=1)
```

Out[93]:
```
array([[3, 7],
       [1, 9]])
```

In [27]:
```
a.sort(1)
print(a)
```

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

5.2 Unique Operation

In [102…
```
# returns unique elements
names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])
print(np.unique(names))
```

```
['Bob' 'Joe' 'Will']
```

In [104…
```
# Contrast np.unique with the pure Python alternative:
sorted(set(names))
```

Out[104…
```
['Bob', 'Joe', 'Will']
```

In [28]:
```
# returns unique elements
ints = np.array([3, 3, 3, 2, 2, 1, 1, 4, 4])
print(np.unique(ints))
```

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

5.3 Set Operations

In [ ]:
```
# set will not allows duplicate elements
```

In [29]:
```
# returns unique elements
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 [6]:
```
# returns union of two sets
arr1=np.array([1,2,3,4])
arr2=np.array([3,4,5,6])
```

In [9]:
```
# perform union on arr1 and arr2
print(np.union1d(arr1,arr2))
```

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

In [10]:
```
#perform intersection on two arrays
print(np.intersect1d(arr1,arr2))
```

```
[3 4]
```

In [11]:
```
#find set difference
print(np.setdiff1d(arr1,arr2))
```

```
[1 2]
```

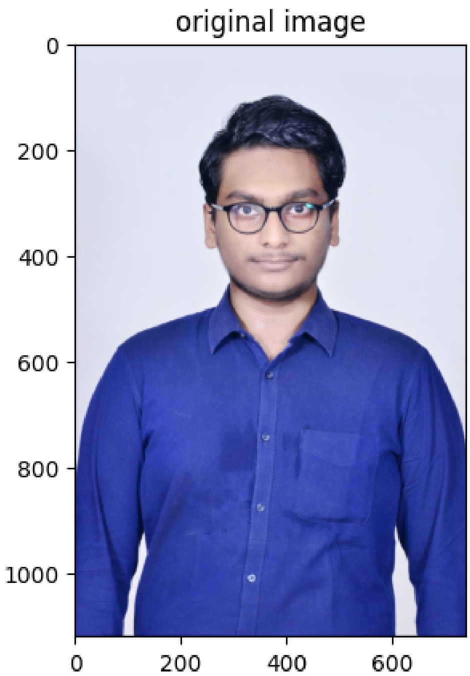
In [8]:
```
#xor between two sets
print(np.setxor1d(arr1,arr2))
```

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

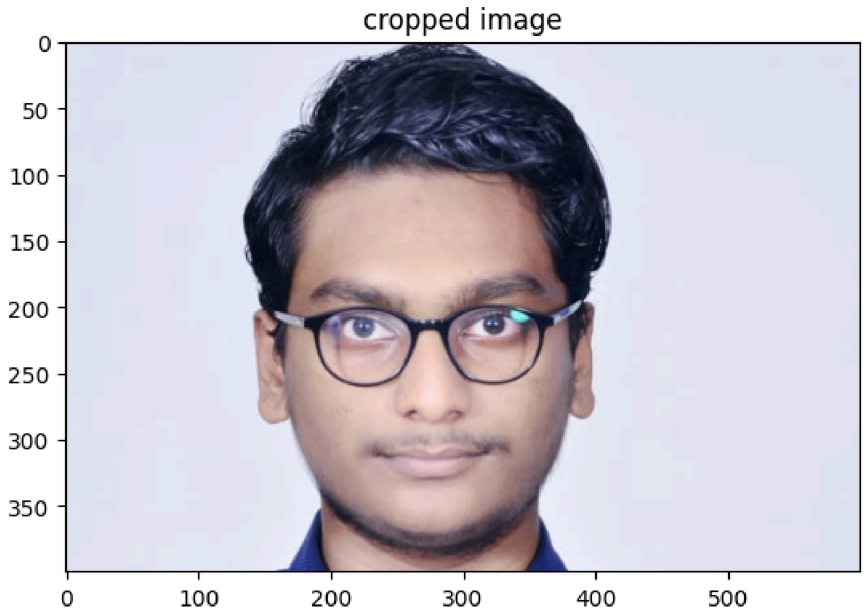
6.Load an image file and do crop and flip operation using Numpy indexing

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
#read image (set image as m)
img = Image.open('imgarr.jpeg')
imgarr=np.array(img)
```

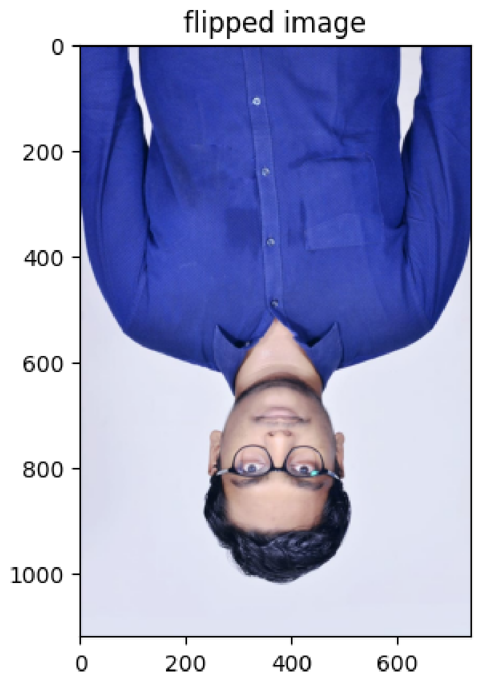
In [2]:
```
#displaying image
plt.imshow(imgarr)
plt.title('original image')
plt.show()
```



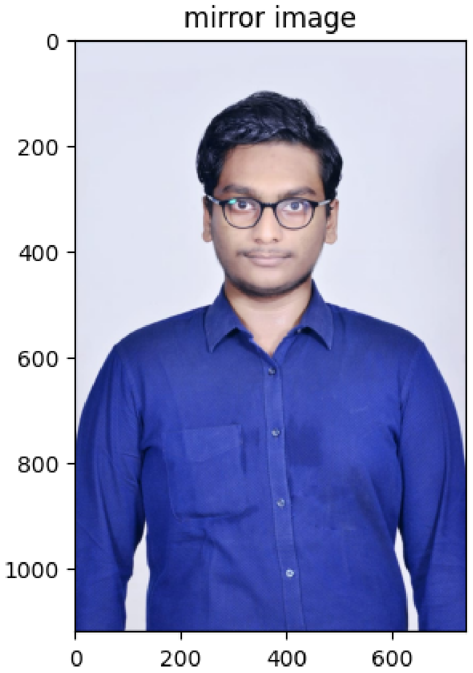
In [4]:
```
# gives croped image
crpimgarr=imgarr[100:500,100:700]
image = Image.fromarray(imgarr)
plt.imshow(crpimgarr)
plt.title('cropped image')
plt.show()
```



In [5]:
```
# flipped by 180 degrees
flipimg=np.flipud(imgarr)
plt.imshow(flipimg)
plt.title('flipped image')
plt.show()
```



In [6]:
```
# mirror image
flipimg=np.fliplr(imgarr)
plt.imshow(flipimg)
plt.title('mirror image')
plt.show()
```



In [ ]:

Data Manipulation with Pandas

Pandas is another essential Python library, specifically designed for data manipulation and analysis. It builds on top of NumPy and provides data structures like DataFrames, which are perfect for handling tabular data akin to databases or spreadsheets. With Pandas, you can perform tasks like data cleaning, merging, reshaping, and aggregating with ease. It’s a game-changer for anyone working in data science or analytics

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

1.1Pandas Series From Python List

In [1]:
```
# import pandas and numpy
import numpy as np
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 [2]:
```
d={'11':[1,2,3],'22':[4,5,6]}
w=pd.Series(d,index=['a','b','c'])
print(w)
```

```
a    NaN
b    NaN
c    NaN
dtype: object
```

In [3]:
```
# create Pandas Series with define indexes
import pandas as pd
ind = [10, 20, 30, 40]
lst = ['G', 'h', 'i', 'j']
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
```

In [4]:
```
# sries from List of Strings:
data = ['apple', 'banana', 'cherry', 'date', 'elderberry']
series = pd.Series(data)
print(series)
```

```
0         apple
1        banana
2        cherry
3          date
4    elderberry
dtype: object
```

In [39]:
```
# Series with Mixed Data Types:
data = [10, 'banana', 30.5, True, None]
series = pd.Series(data,index=[1,2,3,4,5])
print(series)
```

```
1        10
2    banana
3      30.5
4      True
5      None
dtype: object
```

1.2 Pandas Series From Numpy arrays

In [6]:
```
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)
```

```
0    a
1    b
2    c
3    d
4    e
dtype: object
```

In [7]:
```
# 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    d
1004    e
dtype: object
```

In [8]:
```
# Convert NumPy array to Series
numpy_array = np.array([1, 2.8, 3.0, 2, 9, 4.2])
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 [42]:
```
# create a dictionary
dictionary = {'D': 10, 'B': 20, 'C': 30}
# create a series
series = pd.Series(dictionary,index=['a','C','c'])
print(series)
```

```
a     NaN
C    30.0
c     NaN
dtype: float64
```

In [12]:
```
# Series from Dictionary with Mixed Data Types:
data = {'a': 10, 'b': 'banana', 'c': 30.5, 'd': True, 'e': None}
series = pd.Series(data)
print(series)
```

```
a        10
b    banana
c      30.5
d      True
e      None
dtype: object
```

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

2. Data Manipulation with Pandas Series

2.1 Indexing

In [1]:
```
import pandas as pd
import numpy as np
# creating simple array
data = np.array(['a','b','a','n','s','r','z','m'])
ser = pd.Series(data,index=[10,11,12,13,14,15,16,17])
print(ser[16])
```

```
z
```

In [13]:
```
# Indexing with Integer Location (iloc):
data = [10, 20, 30, 40, 50]
index = ['a', 'b', 'c', 'd', 'e']
series = pd.Series(data, index=index)
# Accessing elements by position
print(series.iloc[0])  # First element
print(series.iloc[1:4])  # Second to fourth elements
```

```
10
b    20
c    30
d    40
dtype: int64
```

In [33]:
```
# accessing by specific index
print(sr['Day 1'])
```

```
1/1/2018
```

In [18]:
```
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
```

In [52]:
```
import pandas as pd

data = {'A': [1, 2, 3], 'B': [4, 5, 6]}
df = pd.DataFrame(data, index=['x', 'y', 'z'])
print(df)
# Access row with label 'y' and column 'A'
print(df.loc['y', 'A'])  # Output: 2
print(df.loc['x':'z'])
```

```
   A  B
x  1  4
y  2  5
z  3  6
2
Empty DataFrame
Columns: [A, B]
Index: []
```

In [13]:
```
# Select rows from position 0 to 1 (excluding 2)
print(df.iloc[0:2,0])
```

```
x    1
y    2
Name: A, dtype: int64
```

2.2 Selecting

In [39]:
```
# getting specied values
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
```

In [35]:
```
# accessing value by one index
s['b']
```

Out[35]:
```
1.0
```

In [36]:
```
#accessing value by multiple index
s[['b','a','d']]
```

Out[36]:
```
b    1.0
a    0.0
d    3.0
dtype: float64
```

In [17]:
```
# Selecting a Single Element:
data = [10, 20, 30, 40, 50]
index = ['a', 'b', 'c', 'd', 'e']
series = pd.Series(data, index=index)
# Select the element with label 'b'
print(series.loc['b'])  # Output: 20
```

```
20
```

In [18]:
```
# Selecting a Range of Elements:
s['b':'e']
```

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

In [28]:
```
print(s[[0, 2, 4]])
```

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

2.3 Filtering

In [42]:
```
# getting data by specifing condition
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
```

In [43]:
```
# values less than 2
s[s<2]
```

Out[43]:
```
a    0.0
b    1.0
dtype: float64
```

In [44]:
```
# values grater then 2
s[s>2]
```

Out[44]:
```
d    3.0
e    4.0
dtype: float64
```

In [45]:
```
# not equals to 2
s[s!=2]
```

Out[45]:
```
a    0.0
b    1.0
d    3.0
e    4.0
dtype: float64
```

In [46]:
```
# more then one condition using and
s[(s>2)&(s<5) ]
```

Out[46]:
```
d    3.0
e    4.0
dtype: float64
```

In [33]:
```
s['b':'c']
```

Out[33]:
```
b    5.0
c    2.0
dtype: float64
```

In [47]:
```
# comparision operator
print(s[1:2]==5)
```

```
b    False
dtype: bool
```

In [42]:
```
s[s.isin([2,4])]
```

Out[42]:
```
c    2.0
e    4.0
dtype: float64
```

2.4 Arithmetic Operations

In [49]:
```
import pandas as pd
series1 = pd.Series([1, 2, 3, 4, 5])
series2 = pd.Series([6, 7, 8, 9, 10])
```

In [50]:
```
# addtion
series3 = series1 + series2
print(series3)
```

```
0     7
1     9
2    11
3    13
4    15
dtype: int64
```

In [51]:
```python
# substraction
series3 = series1 - series2
print(series3)
```

```
0   -5
1   -5
2   -5
3   -5
4   -5
dtype: int64
```

In [5]:
```python
# multiplication
series3 = series1 *series2
print(series3)
```

```
0     6
1    14
2    24
3    36
4    50
dtype: int64
```

In [52]:
```python
# division
series3 = series1 /series2
print(series3)
```

```
0    0.166667
1    0.285714
2    0.375000
3    0.444444
4    0.500000
dtype: float64
```

In [53]:
```python
# modulo division
series3 = series1 %series2
print(series3)
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

2.5 Ranking

In [10]:
```python
# ranking based on the condition
# ascending = true
import pandas as pd
s=pd.Series([121,211,153,214,115,116,237,118,219,120])
s.rank(ascending=True)
```

Out[10]:
```
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 [49]:
```python
# ascending= false
s.rank(ascending=False)
```

```
Out[49]: 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 [54]: # usnig min method
s.rank(method='min')
```

```
Out[54]: a    1.0
b    2.0
c    3.0
d    4.0
e    5.0
dtype: float64
```

```
In [55]: # using max method
s.rank(method='max')
```

```
Out[55]: a    1.0
b    2.0
c    3.0
d    4.0
e    5.0
dtype: float64
```

2.6 Sorting

```
In [20]: # Sorting by Index:
# Sort the Series by its index
data = [50, 20, 30, 10, 40]
index = ['e', 'b', 'c', 'a', 'd']
series = pd.Series(data, index=index)
sorted_by_index = series.sort_index()
print(sorted_by_index)
```

```
a    10
b    20
c    30
d    40
e    50
dtype: int64
```

```
In [19]: # creating series
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 [20]: # Sorting in Descending Order:
sr.sort_values(ascending = False)
```

```
Out[20]: 1    16.8000
4    18.1002
0    19.5000
3    20.1240
2    22.7800
dtype: float64
```

```
In [57]: # sort by values
sr.sort_values(ascending = True)
```

Out[57]:
```
1    16.8000
4    18.1002
0    19.5000
3    20.1240
2    22.7800
dtype: float64
```

2.7 checking null values

In [22]:
```
# checking null values
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 [60]:
```
# method isnull
# return true if null
x.isnull()
```

Out[60]:
```
10    False
20    False
30    False
40    False
50    False
60    False
70    False
dtype: bool
```

In [44]:
```
# return true if notnull
x.notnull()
```

Out[44]:
```
california    False
ohio           True
Texas         False
oregon         True
dtype: bool
```

2.8 Concatenation

In [24]:
```
# creating the Series
series1 = pd.Series([1, 2, 3])
series2 = pd.Series(['A', 'B', 'C'])
```

In [65]:
```
# concatenating
display(pd.concat([series1, series2]))
```

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

In [66]:
```
# concatinated by axis =1
display(pd.concat([series1, series2], axis = 1))
```

	0	1
0	1	A
1	2	B
2	3	C

In [67]:
```
# concatinated by axis =1
display(pd.concat([series1, series2], axis = 0))
```

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

3 .Creating DataFrames from List and Dictionary

3.1 From List

In [2]:
```
import pandas as pd
data = [1, 2, 3, 4, 5]
# Convert to DataFrame
df = pd.DataFrame(data)
print(df)
```

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

In [63]:
```
# form more than one list
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 [7]:
```
# Create DataFrame without specifying column names
data = [
    [1, 'A', 24],
    [2, 'B', 27],
    [3, 'C', 22]
]
df = pd.DataFrame(data)
print(df)
# Create DataFrame with column names
df = pd.DataFrame(data, columns=['ID', 'Name', 'Age'],index=[1,2,3])
print(df)
```

```
   0  1   2
0  1  A  24
1  2  B  27
2  3  C  22
   ID Name  Age
1   1    A   24
2   2    B   27
3   3    C   22
```

In [29]:
```
# Creating DataFrame from a List of Dictionaries:
data = [
    {'ID': 1, 'Name': 'Alice', 'Age': 24},
    {'ID': 2, 'Name': 'Bob', 'Age': 27},
    {'ID': 3, 'Name': 'Charlie', 'Age': 22}
]
df = pd.DataFrame(data)
print(df)
```

```
   ID     Name  Age
0   1    Alice   24
1   2      Bob   27
2   3  Charlie   22
```

3.2 From Dictionary

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

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

In [65]:
```
# values as lists
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 [66]:
```
# using tuples
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 [12]:
```
# nested dictonary
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'])
df
```

Out[12]:

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

In [31]:
```
# Creating DataFrame from a Dictionary with Index Specified:
data = {
    'ID': [1, 2, 3],
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [24, 27, 22]
}
index = ['a', 'b', 'c']
df = pd.DataFrame(data, index=index)
print(df)
```

```
   ID     Name  Age
a   1    Alice   24
b   2      Bob   27
c   3  Charlie   22
```

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

4.1 Importing file

In [14]:
```
# file imported
import pandas as pd
data=pd.read_csv('IRIS.csv')
data
```

Out[14]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

4.2 display top and bottom five rows

In [15]:
```
# returns top 5 records
data.head()
```

Out[15]:

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

In [16]:
```
# returns bottom 5 records
data.tail()
```

Out[16]:

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

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

In [4]:
```
# returns number of rows and columns
data.shape
```

Out[4]:
```
(150, 5)
```

In [5]:
```
# returns the data type
data.dtypes
```

Out[5]:
```
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
species          object
dtype: object
```

In [6]:
```
# returns howmany elements are there having null values
data.isnull().sum()
```

Out[6]:
```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

In [18]:
```
# reutrns all columns
data.columns
```

Out[18]:
```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

In [19]:
```
# gives index deatails
data.index
```

Out[19]:
```
RangeIndex(start=0, stop=150, step=1)
```

4.4 Select/Delete the records rows/columns based on conditions

In [33]:
```
data.drop('sepal_length',axis=1)
```

Out[33]:

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

150 rows × 4 columns

In [9]:
```
# accessing data by specifing condition
data.loc[data['sepal_width']>4]
```

Out[9]:

	sepal_length	sepal_width	petal_length	petal_width	species
15	5.7	4.4	1.5	0.4	Iris-setosa
32	5.2	4.1	1.5	0.1	Iris-setosa
33	5.5	4.2	1.4	0.2	Iris-setosa

In [39]:
```
data.loc[(data['sepal_length']>7 )& (data['sepal_length']<7.2)]
```

Out[39]:

	sepal_length	sepal_width	petal_length	petal_width	species
102	7.1	3.0	5.9	2.1	Iris-virginica

In [21]:
```
data.loc[0]
```

Out[21]:
```
sepal_length            5.1
sepal_width             3.5
petal_length            1.4
petal_width             0.2
species         Iris-setosa
Name: 0, dtype: object
```

In [27]:
```
data.drop(data[data['sepal_length']>4.3].index)
```

Out[27]:

	sepal_length	sepal_width	petal_length	petal_width	species
13	4.3	3.0	1.1	0.1	Iris-setosa

In [12]:
```
data.loc[6,'petal_length']
```

Out[12]:
```
1.4
```

In [31]:
```
# accessing data using range
data.loc[11:15][['sepal_length','sepal_width']]
```

Out[31]:

	sepal_length	sepal_width
11	4.8	3.4
12	4.8	3.0
13	4.3	3.0
14	5.8	4.0
15	5.7	4.4

4.5 Sorting and Ranking operations in DataFrame

In [14]:
```
data
```

Out[14]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [41]:

```
# appling sorting funciton
data.sort_index(ascending=False)
```

Out[41]:

	sepal_length	sepal_width	petal_length	petal_width	species
149	5.9	3.0	5.1	1.8	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
145	6.7	3.0	5.2	2.3	Iris-virginica
...	...	...	...	...	...
4	5.0	3.6	1.4	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
0	5.1	3.5	1.4	0.2	Iris-setosa

150 rows × 5 columns

In [17]:

```
data.sort_values(by=['petal_width','petal_length']).head(6)
```

Out[17]:

	sepal_length	sepal_width	petal_length	petal_width	species
13	4.3	3.0	1.1	0.1	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa
32	5.2	4.1	1.5	0.1	Iris-setosa
34	4.9	3.1	1.5	0.1	Iris-setosa
37	4.9	3.1	1.5	0.1	Iris-setosa

In [18]:

```
# appling rank for top 10
data.rank().head(10)
```

Out[18]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	37.0	129.5	17.5	20.5	25.5
1	19.5	70.5	17.5	20.5	25.5
2	10.5	102.0	8.0	20.5	25.5
3	7.5	89.5	30.5	20.5	25.5
4	27.5	134.0	17.5	20.5	25.5
5	49.5	145.5	46.5	45.0	25.5
6	7.5	120.5	17.5	38.0	25.5
7	27.5	120.5	30.5	20.5	25.5
8	3.0	52.5	17.5	20.5	25.5
9	19.5	89.5	30.5	3.5	25.5

In [19]:

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

Out[19]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	37.0	129.5	17.5	20.5	25.5
1	19.5	70.5	17.5	20.5	25.5

In [20]:

```
data.rank(ascending=False).head(5)
```

Out[20]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	114.0	21.5	133.5	130.5	125.5
1	131.5	80.5	133.5	130.5	125.5
2	140.5	49.0	143.0	130.5	125.5
3	143.5	61.5	120.5	130.5	125.5
4	123.5	17.0	133.5	130.5	125.5

In [21]:

```
data['sepal_length'].rank().head(5)
```

Out[21]:

```
0    37.0
1    19.5
2    10.5
3     7.5
4    27.5
Name: sepal_length, dtype: float64
```

4.6 Statistical Operations

In [48]:

```
# appling statistical values
data=pd.read_csv('diabetes.csv')
data
```

Out[48]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [49]:

```
# mean
data.mean()
```

Out[49]:

```
Pregnancies                   3.845052
Glucose                     120.894531
BloodPressure                69.105469
SkinThickness                20.536458
Insulin                      79.799479
BMI                          31.992578
DiabetesPedigreeFunction      0.471876
Age                          33.240885
Outcome                       0.348958
dtype: float64
```

In [51]:

```
# mean of species columns
data.mean()['BMI']
```

Out[51]:

```
31.992578124999998
```

In [27]:

```
# mode
data.mode()
```

Out[27]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	1.0	99	70.0	0.0	0.0	32.0	0.254	22.0	0.0
1	NaN	100	NaN	NaN	NaN	NaN	0.258	NaN	NaN

In [28]:

```
# median
data.median()
```

Out[28]:

```
Pregnancies                   3.0000
Glucose                     117.0000
BloodPressure                72.0000
SkinThickness                23.0000
Insulin                      30.5000
BMI                          32.0000
DiabetesPedigreeFunction      0.3725
Age                          29.0000
Outcome                       0.0000
dtype: float64
```

In [29]:

```
# standerd deviation
data.std()
```

Out[29]:
```
Pregnancies                   3.369578
Glucose                      31.972618
BloodPressure                19.355807
SkinThickness                15.952218
Insulin                     115.244002
BMI                           7.884160
DiabetesPedigreeFunction      0.331329
Age                          11.760232
Outcome                       0.476951
dtype: float64
```

In [30]:
```
# varience
data.var()
```

Out[30]:
```
Pregnancies                    11.354056
Glucose                      1022.248314
BloodPressure                 374.647271
SkinThickness                 254.473245
Insulin                     13281.180078
BMI                            62.159984
DiabetesPedigreeFunction        0.109779
Age                           138.303046
Outcome                         0.227483
dtype: float64
```

In [33]:
```
#  varience of BMI
data.var()['BMI']
```

Out[33]:
```
62.15998395738266
```

In [34]:
```
# menan of specified columns
data.mean()[['BMI','Insulin','Glucose']]
```

Out[34]:
```
BMI         31.992578
Insulin     79.799479
Glucose    120.894531
dtype: float64
```

4.7 count and Uniqueness of given Categorical values

In [52]:
```
len(data)
```

Out[52]:
```
768
```

In [46]:
```
# gives the count of each and every column
data.count()
```

Out[46]:
```
Pregnancies                 768
Glucose                     768
BloodPressure               768
SkinThickness               768
Insulin                     768
BMI                         768
DiabetesPedigreeFunction    768
Age                         768
Outcome                     768
dtype: int64
```

In [47]:
```
# returns unique values
data.value_counts()
```

Out[47]:
```
Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI   DiabetesPedigreeFunction  Age  Outco
me
0            57       60             0              0        21.7  0.735                     67   0
1
             67       76             0              0        45.3  0.194                     46   0
1
5            103      108            37             0        39.2  0.305                     65   0
1
             104      74             0              0        28.8  0.153                     48   0
1
             105      72             29             325      36.9  0.159                     28   0
1

..
2            84       50             23             76       30.4  0.968                     21   0
1
             85       65             0              0        39.6  0.930                     27   0
1
             87       0              23             0        28.9  0.773                     25   0
1
                      58             16             52       32.7  0.166                     25   0
1
17           163      72             41             114      40.9  0.817                     47   1
1
Name: count, Length: 768, dtype: int64
```

In [37]:
```
data.value_counts(data['Insulin'])
```

Out[37]:
```
Insulin
0      374
105     11
140      9
130      9
120      8
      ...
193      1
191      1
188      1
184      1
846      1
Name: count, Length: 186, dtype: int64
```

In [48]:
```
# returns unique values in a specified column
data['Glucose'].unique()
```

Out[48]:
```
array([148,  85, 183,  89, 137, 116,  78, 115, 197, 125, 110, 168, 139,
       189, 166, 100, 118, 107, 103, 126,  99, 196, 119, 143, 147,  97,
       145, 117, 109, 158,  88,  92, 122, 138, 102,  90, 111, 180, 133,
       106, 171, 159, 146,  71, 105, 101, 176, 150,  73, 187,  84,  44,
       141, 114,  95, 129,  79,   0,  62, 131, 112, 113,  74,  83, 136,
        80, 123,  81, 134, 142, 144,  93, 163, 151,  96, 155,  76, 160,
       124, 162, 132, 120, 173, 170, 128, 108, 154,  57, 156, 153, 188,
       152, 104,  87,  75, 179, 130, 194, 181, 135, 184, 140, 177, 164,
        91, 165,  86, 193, 191, 161, 167,  77, 182, 157, 178,  61,  98,
       127,  82,  72, 172,  94, 175, 195,  68, 186, 198, 121,  67, 174,
       199,  56, 169, 149,  65, 190], dtype=int64)
```

4.8 Rename Single/Multiple columns

In [45]:
```
# accessing top 5 records using range function
data.rename(columns={'Glucose':'gluco'}).head(5)
```

Out[45]:

	Pregnancies	gluco	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [50]:
```
# renaming colums names
data.rename(columns={"Pregnancies":'preg',"SkinThickness":'skin',"DiabetesPedigreeFunction":'diabetes'})
```

Out[50]:

	preg	Glucose	BloodPressure	skin	Insulin	BMI	diabetes	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

Data Cleaning and Preparation

1.Import any CSV file to pandas Frame and perform the following:

the employees.csv dataset contains the data related to the employees details.

It has attributes like first name, Gender, Start Date, Last Login Time ,Salary Bonus %, Senior Management, Team.

In [ ]:
```
### First Name: Employee’s first name (e.g., John).
### Gender: Employee’s gender (e.g., Male, Female).
### Start Date: Date employee joined (e.g., 2022-03-15).
### Last Login Time: Last system login time (e.g., 14:35:22).
###Salary: Annual salary (e.g., 50,000).
Bonus %: Percentage of salary as bonus (e.g., 10%).
Senior Management: Whether in senior management (True/False).
Team: Employee’s department (e.g., IT, Finance).
```

In [4]:
```
import numpy as np
import pandas as pd
df=pd.read_csv('employees.csv')
df
```

Out[4]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services
...	...	...	...	...	...	...	...	...
995	Henry	Female	11/23/2014	6:09 AM	132483	16.655	False	Distribution
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	False	Finance
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	False	Product
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	True	Sales

1000 rows × 8 columns

In [28]:
```
df.shape# discribe no of rows and columns in the dataset
```

Out[28]: (1000, 8)

1.1 Handle missing data by detecting, dropping, and replacing/filling missing values.

missing Data can occur when no information is provided for one or more items or for a whole unit. Missing Data is a very big problem in a real-life scenarios. Missing Data can also refer to as NA(Not Available) values in pandas. In DataFrame sometimes many datasets simply arrive with missing data,

either because it exists and was not collected or it never existed.

In [29]:
```
df.isnull()# returnd the whether it value is null or not
```

Out[29]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	True
2	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...
995	False	False	False	False	False	False	False	False
996	False	False	False	False	False	False	False	False
997	False	False	False	False	False	False	False	False
998	False	False	False	False	False	False	False	False
999	False	False	False	False	False	False	False	False

1000 rows × 8 columns

In [30]:
```
# returnd the sum of null values.
df.isnull().sum()
```

Out[30]:
```
First Name           67
Gender                0
Start Date            0
Last Login Time       0
Salary                0
Bonus %               0
Senior Management    67
Team                 43
dtype: int64
```

In [31]:
```
# returs null values in a particular colunm
df[df.isnull()]
```

Out[31]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...
995	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
996	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
997	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
998	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
999	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

1000 rows × 8 columns

In [32]:
```
# droping null values
df.dropna()
```

Out[32]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services
5	Dennis	Male	4/18/1987	1:35 AM	115163	10.125	False	Legal
...	...	...	...	...	...	...	...	...
995	Henry	Female	11/23/2014	6:09 AM	132483	16.655	False	Distribution
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	False	Finance
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	False	Product
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	True	Sales

899 rows × 8 columns

In [33]:

```
#filling missing values with zeros
df.fillna(0)
df
```

Out[33]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services
...	...	...	...	...	...	...	...	...
995	Henry	Female	11/23/2014	6:09 AM	132483	16.655	False	Distribution
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	False	Finance
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	False	Product
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	True	Sales

1000 rows × 8 columns

In [35]:

```
# Drop rows with any missing values
df_dropped_rows = df.dropna()
df_dropped_rows
```

Out[35]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	Client Services
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services
...	...	...	...	...	...	...	...	...
995	Henry	Female	11/23/2014	6:09 AM	132483	16.655	False	Distribution
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	False	Finance
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	False	Product
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	True	Sales

933 rows × 8 columns

In [39]:

```
# Fill missing values with a specific value, e.g., 'Unknown' for categorical data
df['Gender'].fillna('Unknown')
df
```

Out[39]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	Client Services
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services
...	...	...	...	...	...	...	...	...
995	Henry	Female	11/23/2014	6:09 AM	132483	16.655	False	Distribution
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	False	Finance
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	False	Product
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	True	Sales

1000 rows × 8 columns

1.2 Transform data using apply() and map() method

apply() is used to apply a function along an axis of the DataFrame or on values of Series. applymap() is used to apply a function to a DataFrame elementwise.

map() is used to substitute each value in a Series with another value.

In [40]:

```
# 1. Transform using apply() method
# Let's square the values in the 'Price' column
df['new_Salary'] = df['Salary'].apply(lambda x: x//2)
df
```

Out[40]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team	new_Salary
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing	48654
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	Client Services	30966
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance	65295
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance	69352
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services	50502
...	...	...	...	...	...	...	...	...	...
995	Henry	Female	11/23/2014	6:09 AM	132483	16.655	False	Distribution	66241
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	False	Finance	21196
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	False	Product	48457
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development	30250
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	True	Sales	64974

1000 rows × 9 columns

In [41]:

```
new_gender = {'Male':1,'Female':0}
df['gender_new']=df['Gender'].map(new_gender)
df
```

Out[41]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team	new_Salary	gender_new
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing	48654	1
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	Client Services	30966	1
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance	65295	0
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance	69352	1
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services	50502	1
...	...	...	...	...	...	...	...	...	...	...
995	Henry	Female	11/23/2014	6:09 AM	132483	16.655	False	Distribution	66241	0
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	False	Finance	21196	1
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	False	Product	48457	1
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development	30250	1
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	True	Sales	64974	1

1000 rows × 10 columns

1.3 Detect and filter outliers

An outlier is a point or set of data points that lie away from the rest of the data values of the dataset.

That is, it is a data point(s) that appear away from the overall distribution of data values in a dataset.

In [42]:
```
import pandas as pd
import numpy as np
# Calculate the z-scores for the selected column
z_scores = np.abs((df['Salary'] - df['Salary'].mean()) / df['Salary'].std())
# Define a threshold for outliers (e.g., z-score greater than 3)
z_score_threshold = 1.5
# Filter the DataFrame to keep rows without outliers
filtered_df = df[z_scores >= z_score_threshold]
filtered_df
```

Out[42]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team	new_Salary	gender_new
25	NaN	Male	10/8/2012	1:12 AM	37076	18.576	NaN	Client Services	18538	1
26	Craig	Male	2/27/2000	7:45 AM	37598	7.757	True	Marketing	18799	1
36	Rachel	Female	2/16/2009	8:47 PM	142032	12.599	False	Business Development	71016	0
38	Stephanie	Female	9/13/1986	1:52 AM	36844	5.574	True	Business Development	18422	0
44	Cynthia	Female	11/16/1988	6:54 PM	145146	7.482	True	Product	72573	0
...	...	...	...	...	...	...	...	...	...	...
964	Bruce	Male	5/7/1980	8:00 PM	35802	12.391	True	Sales	17901	1
979	Ernest	Male	7/20/2013	6:41 AM	142935	13.198	True	Product	71467	1
981	James	Male	1/15/1993	5:19 PM	148985	19.280	False	Legal	74492	1
983	John	Male	12/23/1982	10:35 PM	146907	11.738	False	Engineering	73453	1
989	Justin	Female	2/10/1991	4:58 PM	38344	3.794	False	Legal	19172	0

133 rows × 10 columns

In [43]:
```
# Select the column to analyze for outliers (replace 'Value' with the actual column name)
column_name = 'Salary'

# Calculate the z-scores for the selected column
z_scores = np.abs((df[column_name] - df[column_name].mean()) / df[column_name].std())
print(z_scores)
# Define a threshold for outliers (e.g., z-score greater than 3)
z_score_threshold = 2300

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

```
0      0.201855
1      0.872599
2      1.212738
3      1.459217
4      0.314115
         ...
995    1.270235
996    1.466123
997    0.189888
998    0.916124
999    1.193269
Name: Salary, Length: 1000, dtype: float64
```

Out[43]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team	new_Salary	gender_new
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing	48654	1
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	Client Services	30966	1
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance	65295	0
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance	69352	1
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services	50502	1
...	...	...	...	...	...	...	...	...	...	...
995	Henry	Female	11/23/2014	6:09 AM	132483	16.655	False	Distribution	66241	0
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	False	Finance	21196	1
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	False	Product	48457	1
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development	30250	1
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	True	Sales	64974	1

1000 rows × 10 columns

1.4 Perform vectorized string operations on pandas series

strength of Python is its relative ease in handling and manipulating string data. Pandas builds on this and provides

a comprehensive set of vectorized string operations that are an important part of the type of munging required when

working with (read: cleaning up) real-world data. In this chapter, we'll walk through some of the Pandas string operations,

and then take a look at using them to partially clean up a very messy dataset of recipes collected from the internet.

In [5]:
```
#Convert all names to uppercase
df['First Name_upper'] = df['First Name'].str.lower()
df
```

Out[5]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team	First Name_upper
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing	douglas
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN	thomas
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance	maria
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance	jerry
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services	larry
...	...	...	...	...	...	...	...	...	...
995	Henry	Female	11/23/2014	6:09 AM	132483	16.655	False	Distribution	henry
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	False	Finance	phillip
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	False	Product	russell
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development	larry
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	True	Sales	albert

1000 rows × 9 columns

In [45]:

```
# Calculate the length of each name
df['screen_name_length'] = df['First Name'].str.len()
df
```

Out[45]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team	new_Salary	gender_new	N
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing	48654	1	
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	Client Services	30966	1	
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance	65295	0	
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance	69352	1	
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services	50502	1	
...	...	...	...	...	...	...	...	...	...	...	
995	Henry	Female	11/23/2014	6:09 AM	132483	16.655	False	Distribution	66241	0	
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	False	Finance	21196	1	
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	False	Product	48457	1	
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development	30250	1	
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	True	Sales	64974	1	

1000 rows × 12 columns

In [48]:

```
# Split the names based on a delimiter (e.g., space) and create a new column for the first part of the n
df['name'] = df['First Name'].str.split(' ').str[0]
df
```

Out[48]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team	new_Salary	gender_new	N
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing	48654	1	
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	Client Services	30966	1	
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance	65295	0	
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance	69352	1	
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services	50502	1	
...	...	...	...	...	...	...	...	...	...	...	
995	Henry	Female	11/23/2014	6:09 AM	132483	16.655	False	Distribution	66241	0	
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	False	Finance	21196	1	
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	False	Product	48457	1	
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development	30250	1	
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	True	Sales	64974	1	

1000 rows × 13 columns

In [52]:

```
#converts the first name to uppercase
df['First Name_upper'] = df['First Name'].str.upper()
df
```

Out[52]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team	new_Salary	gender_new	N
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing	48654	1	
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	Client Services	30966	1	
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance	65295	0	
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance	69352	1	
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services	50502	1	
...	...	...	...	...	...	...	...	...	...	...	
995	Henry	Female	11/23/2014	6:09 AM	132483	16.655	False	Distribution	66241	0	
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	False	Finance	21196	1	
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	False	Product	48457	1	
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development	30250	1	
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	True	Sales	64974	1	

1000 rows × 13 columns

In [61]:

```
#returns boolean values if team column contains record 'finance'
print(df['Team'].str.contains('Finance'))
```

```
0      False
1      False
2       True
3       True
4      False
       ...  
995    False
996     True
997    False
998    False
999    False
Name: Team, Length: 1000, dtype: bool
```

In [6]:

```
print(df['First Name'].str.contains('a'))
```

```
0       True
1       True
2       True
3      False
4       True
       ...  
995    False
996    False
997    False
998     True
999    False
Name: First Name, Length: 1000, dtype: object
```

Data Wrangling

1. Concatenate / Join / Merge/ Reshape DataFrames.

2. Read DataFrame to create a pivot table.

3. Read DataFrame to create a cross table.

the employees.csv dataset contains the data related to the employees details.

It has attributes like first name, Gender, Start Date, Last Login Time ,Salary Bonus %, Senior Management, Team.

First Name: Employee's first name (e.g., John).

Gender: Employee's gender (e.g., Male, Female).

Start Date: Date employee joined (e.g., 2022-03-15).

Last Login Time: Last system login time (e.g., 14:35:22).

Salary: Annual salary (e.g., 50,000).

Bonus %: Percentage of salary as bonus (e.g., 10%).

Senior Management: Whether in senior management (True/False).

Team: Employee's department (e.g., IT, Finance).

In [7]:
```
import pandas as pd
# Load the CSV file into a DataFrame
df = pd.read_csv('employees.csv')
df
```

Out[7]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services
...	...	...	...	...	...	...	...	...
995	Henry	NaN	11/23/2014	6:09 AM	132483	16.655	False	Distribution
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	False	Finance
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	False	Product
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	True	Sales

1000 rows × 8 columns

1 Concatenate / Join / Merge/ Reshape DataFrames.

In [9]:
```
# Display the first few rows to understand the structur
print(df.head())
```

```
  First Name  Gender Start Date Last Login Time  Salary  Bonus %  \
0    Douglas    Male   8/6/1993        12:42 PM   97308    6.945
1     Thomas    Male  3/31/1996         6:53 AM   61933    4.170
2      Maria  Female  4/23/1993        11:17 AM  130590   11.858
3      Jerry    Male   3/4/2005         1:00 PM  138705    9.340
4      Larry    Male  1/24/1998         4:47 PM  101004    1.389

  Senior Management             Team
0              True        Marketing
1              True              NaN
2             False          Finance
3              True          Finance
4              True  Client Services
```

In [10]:
```
# Example DataFrames for Concatenation
df1 = df[['Gender', 'Team']].head(5)  # First 5 rows with 'Year' and 'JAN'
df2 = df[['Gender', 'First Name']].head(5)  # First 5 rows with 'Year' and 'FEB'
```

In [11]:
```
# Concatenate df1 and df2 along columns
concatenated_columns_df = pd.concat([df1, df2], axis=1)
print("\nConcatenated DataFrame (columns):\n", concatenated_columns_df)
```

```
Concatenated DataFrame (columns):
    Gender             Team  Gender First Name
0    Male        Marketing    Male    Douglas
1    Male              NaN    Male     Thomas
2  Female          Finance  Female      Maria
3    Male          Finance    Male      Jerry
4    Male  Client Services    Male      Larry
```

In [12]:
```
# Concatenate df1 and df2 along rows
concatenated_rows_df = pd.concat([df1, df2], axis=0)
print("\nConcatenated DataFrame (rows):\n", concatenated_rows_df)
```

```
Concatenated DataFrame (rows):
    Gender             Team First Name
0    Male        Marketing        NaN
1    Male              NaN        NaN
2  Female          Finance        NaN
3    Male          Finance        NaN
4    Male  Client Services        NaN
0    Male              NaN    Douglas
1    Male              NaN     Thomas
2  Female              NaN      Maria
3    Male              NaN      Jerry
4    Male              NaN      Larry
```

In [14]:
```
# Example DataFrames for Merging
df3 = df[['Gender', 'Team']].head(5)  # First 5 rows with 'Year' and 'MAR'
df4 = df[['Gender', 'First Name']].head(5)  # First 5 rows with 'Year' and 'APR'
```

In [16]:
```
# Merge df3 and df4 on 'Year'
merged_df = pd.merge(df3, df4, on='Gender', how='inner')
print("\nMerged DataFrame on 'Year':\n", merged_df)
```

```
Merged DataFrame on 'Year':
     Gender            Team First Name
0      Male       Marketing    Douglas
1      Male       Marketing     Thomas
2      Male       Marketing      Jerry
3      Male       Marketing      Larry
4      Male             NaN    Douglas
5      Male             NaN     Thomas
6      Male             NaN      Jerry
7      Male             NaN      Larry
8    Female         Finance      Maria
9      Male         Finance    Douglas
10     Male         Finance     Thomas
11     Male         Finance      Jerry
12     Male         Finance      Larry
13     Male  Client Services    Douglas
14     Male  Client Services     Thomas
15     Male  Client Services      Jerry
16     Male  Client Services      Larry
```

In [19]:
```
# Example DataFrames for Joining
df5 = df[['Gender', 'First Name']].head(5).set_index('Gender')  # First 5 rows with 'Year' as index
df6 = df[['Gender', 'Team']].head(5).set_index('Gender')  # First 5 rows with 'Year' as index
```

In [20]:
```
# Join df5 and df6 based on the 'Year' index
joined_df = df5.join(df6)
print("\nJoined DataFrame (based on index 'Year'):\n", joined_df)
```

```
Joined DataFrame (based on index 'Year'):
        First Name             Team
Gender
Male      Douglas        Marketing
Male      Douglas              NaN
Male      Douglas          Finance
Male      Douglas  Client Services
Male       Thomas        Marketing
Male       Thomas              NaN
Male       Thomas          Finance
Male       Thomas  Client Services
Female      Maria          Finance
Male        Jerry        Marketing
Male        Jerry              NaN
Male        Jerry          Finance
Male        Jerry  Client Services
Male        Larry        Marketing
Male        Larry              NaN
Male        Larry          Finance
Male        Larry  Client Services
```

In [32]:
```
# Transpose the entire dataset
transposed_df = df.T

print("Transposed Dataset:")
print(transposed_df.head())
```

```
Transposed Dataset:
                        0          1          2          3          4    \
First Name        Douglas     Thomas      Maria      Jerry      Larry
Gender               Male       Male     Female       Male       Male
Start Date       8/6/1993  3/31/1996  4/23/1993   3/4/2005  1/24/1998
Last Login Time  12:42 PM    6:53 AM   11:17 AM    1:00 PM    4:47 PM
Salary              97308      61933     130590     138705     101004

                        5          6          7          8          9    ...  \
First Name         Dennis       Ruby        NaN     Angela    Frances  ...
Gender               Male     Female     Female     Female     Female  ...
Start Date      4/18/1987  8/17/1987  7/20/2015 11/22/2005   8/8/2002  ...
Last Login Time   1:35 AM    4:20 PM   10:43 AM    6:29 AM    6:51 AM  ...
Salary             115163      65476      45906      95570     139852  ...

                      990        991        992        993        994  \
First Name          Robin       Rose    Anthony       Tina     George
Gender             Female     Female       Male     Female       Male
Start Date      7/24/1987  8/25/2002 10/16/2011  5/15/1997  6/21/2013
Last Login Time   1:35 PM    5:12 AM    8:35 AM    3:53 PM    5:47 PM
Salary             100765     134505     112769      56450      98874

                       995        996        997        998        999
First Name           Henry    Phillip    Russell      Larry     Albert
Gender                 NaN       Male       Male       Male       Male
Start Date      11/23/2014  1/31/1984  5/20/2013  4/20/2013  5/15/2012
Last Login Time    6:09 AM    6:30 AM   12:39 PM    4:45 PM    6:24 PM
Salary              132483      42392      96914      60500     129949

[5 rows x 1000 columns]
```

2 Read DataFrame to create a pivot table.

In [25]:

```python
# Pivot table with average salary by Team and Gender
pivot_table = pd.pivot_table(df,
                             values='Salary',
                             index='Team',
                             columns='Gender',
                             aggfunc='mean')

print("Pivot Table (Average Salary):")
print(pivot_table)
```

```
Pivot Table (Average Salary):
Gender                      Female          Male
Team
Business Development  92669.060000  89071.750000
Client Services       86430.083333  93141.833333
Distribution          81328.162162  93861.800000
Engineering           90311.045455  98408.250000
Finance               92203.454545  95507.731707
Human Resources       93581.837838  91368.733333
Legal                 90790.382353  84254.657143
Marketing             95074.250000  86082.365854
Product               86182.644444  88957.825000
Sales                 89814.564103  93196.666667
```

In [27]:

```python
# Pivot table showing total salary and average bonus % by Gender and Senior Management
pivot_table = pd.pivot_table(df,
                             values=['Salary', 'Bonus %'],
                             index='Gender',
                             columns='Senior Management',
                             aggfunc={'Salary': 'sum', 'Bonus %': 'mean'})

print("Pivot Table (Total Salary & Avg Bonus %):")
print(pivot_table)
```

```
Pivot Table (Total Salary & Avg Bonus %):
                     Bonus %              Salary
Senior Management      False      True     False      True
Gender
Female             10.099785  9.906090  17926660  17915633
Male               10.982788  9.774127  17600164  18241691
```

In [28]:
```python
# Pivot table showing the count of employees in each Team by Gender
pivot_table_count = pd.pivot_table(df,
                                   values='First Name',
                                   index='Team',
                                   columns='Gender',
                                   aggfunc='count')

print("\nPivot Table (Employee Count by Team and Gender):")
print(pivot_table_count)
```

```
Pivot Table (Employee Count by Team and Gender):
Gender                Female  Male
Team
Business Development      49    39
Client Services           47    38
Distribution              31    29
Engineering               43    36
Finance                   42    38
Human Resources           33    43
Legal                     33    34
Marketing                 36    38
Product                   44    39
Sales                     35    37
```

read dataframe to create cross table

3 Read DataFrame to create a cross table.

In [26]:
```python
# Cross table of Team vs Gender
crosstab = pd.crosstab(df['Team'], df['Gender'])

print("\nCross Table (Team vs Gender):")
print(crosstab)
```

```
Cross Table (Team vs Gender):
Gender                Female  Male
Team
Business Development      50    40
Client Services           48    42
Distribution              37    35
Engineering               44    40
Finance                   44    41
Human Resources           37    45
Legal                     34    35
Marketing                 40    41
Product                   45    40
Sales                     39    39
```

In [29]:
```python
# Cross table of Senior Management status vs Gender
crosstab_sm_gender = pd.crosstab(df['Senior Management'], df['Gender'])

print("\nCross Table (Senior Management vs Gender):")
print(crosstab_sm_gender)
```

```
Cross Table (Senior Management vs Gender):
Gender             Female  Male
Senior Management
False                 200   198
True                  200   197
```

In [30]:
```python
# Extract the year from Start Date
df['Start Year'] = pd.to_datetime(df['Start Date']).dt.year

# Cross table of Start Year vs Team
crosstab_start_team = pd.crosstab(df['Start Year'], df['Team'])

print("\nCross Table (Start Year vs Team):")
print(crosstab_start_team)
```

Cross Table (Start Year vs Team):

Start Year	Business Development	Client Services	Distribution	Engineering
1980	0	6	4	3
1981	1	5	1	1
1982	3	3	0	3
1983	2	1	0	1
1984	4	4	4	1
1985	1	5	1	3
1986	2	6	2	2
1987	0	3	1	1
1988	3	0	3	1
1989	1	2	3	1
1990	2	2	3	4
1991	4	1	1	1
1992	3	2	1	3
1993	3	0	1	1
1994	4	3	3	1
1995	2	3	4	1
1996	5	1	2	5
1997	3	2	0	5
1998	0	2	0	3
1999	4	6	6	2
2000	3	1	5	1
2001	2	1	2	3
2002	5	2	3	5
2003	3	4	2	2
2004	3	3	6	0
2005	1	3	5	1
2006	1	1	3	3
2007	2	2	1	5
2008	3	6	3	1
2009	6	6	5	7
2010	4	3	1	4
2011	5	5	2	3
2012	5	4	3	1
2013	8	1	3	3
2014	1	4	5	2
2015	2	1	0	6
2016	0	2	1	2

Start Year	Finance	Human Resources	Legal	Marketing	Product	Sales
1980	3	3	0	3	3	3
1981	2	1	2	4	0	3
1982	3	2	2	4	2	5
1983	1	5	3	3	3	3
1984	2	4	1	6	3	3
1985	3	0	3	1	0	0
1986	5	1	2	1	3	4
1987	1	3	1	2	2	1
1988	1	4	3	1	4	2
1989	5	0	1	2	3	1
1990	1	3	1	1	2	2
1991	4	3	5	1	4	3
1992	3	4	4	4	3	1
1993	1	1	2	2	5	1
1994	2	4	1	2	3	1
1995	6	6	3	5	5	7
1996	4	5	0	2	3	1
1997	2	2	4	4	3	1
1998	3	1	3	0	2	2
1999	5	2	2	2	2	5
2000	1	1	2	2	2	5
2001	2	1	3	4	2	5
2002	1	4	3	5	4	1
2003	2	2	5	1	1	2
2004	5	3	0	6	0	4
2005	5	1	4	1	5	4
2006	4	5	1	1	4	2
2007	5	3	8	2	1	3
2008	0	0	5	2	1	4
2009	2	4	3	5	1	2
2010	2	3	3	2	3	2
2011	6	2	0	0	4	3

2012	1	1	1	4	3	3
2013	1	3	1	3	5	0
2014	4	1	3	3	2	3
2015	4	1	2	4	0	2
2016	0	2	1	3	2	0

Plotting and Visualization

Data visualization on any sample dataset using matplotlib for the following:

a) Line Plot

b) Bar Plot

c) Histogram

d) Density Plot

e) Scatter Plot

In [3]:
```python
import pandas as pd
import matplotlib.pyplot as plt
# Load the CSV file
file_path = "employees.csv"
data= pd.read_csv(file_path)
data
```

Out[3]:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services
...	...	...	...	...	...	...	...	...
995	Henry	Female	11/23/2014	6:09 AM	132483	16.655	False	Distribution
996	Phillip	Male	1/31/1984	6:30 AM	42392	19.675	False	Finance
997	Russell	Male	5/20/2013	12:39 PM	96914	1.421	False	Product
998	Larry	Male	4/20/2013	4:45 PM	60500	11.985	False	Business Development
999	Albert	Male	5/15/2012	6:24 PM	129949	10.169	True	Sales

1000 rows × 8 columns

1. line plot

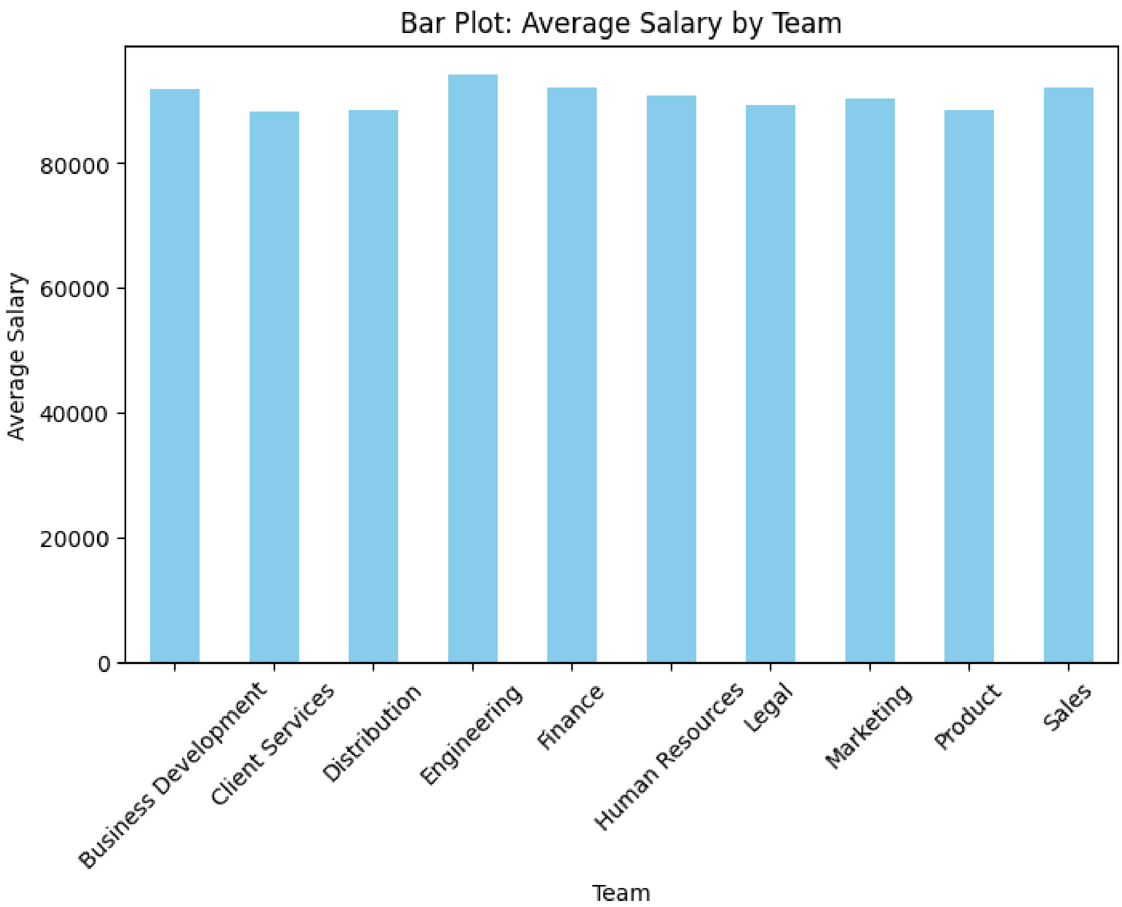
In [4]:
```python
import matplotlib.pyplot as plt

# Example: Assuming 'Year' and 'Salary' are columns in your dataset
plt.figure(figsize=(10, 5))
plt.plot(data['Bonus %'].head(10), data['Salary'].head(10), marker='o')  # Customize with your column na
plt.title('Employee Salary Over Years')
plt.xlabel('Bonus %')
plt.ylabel('Salary')
plt.grid()
plt.show()
```

2.bar plot

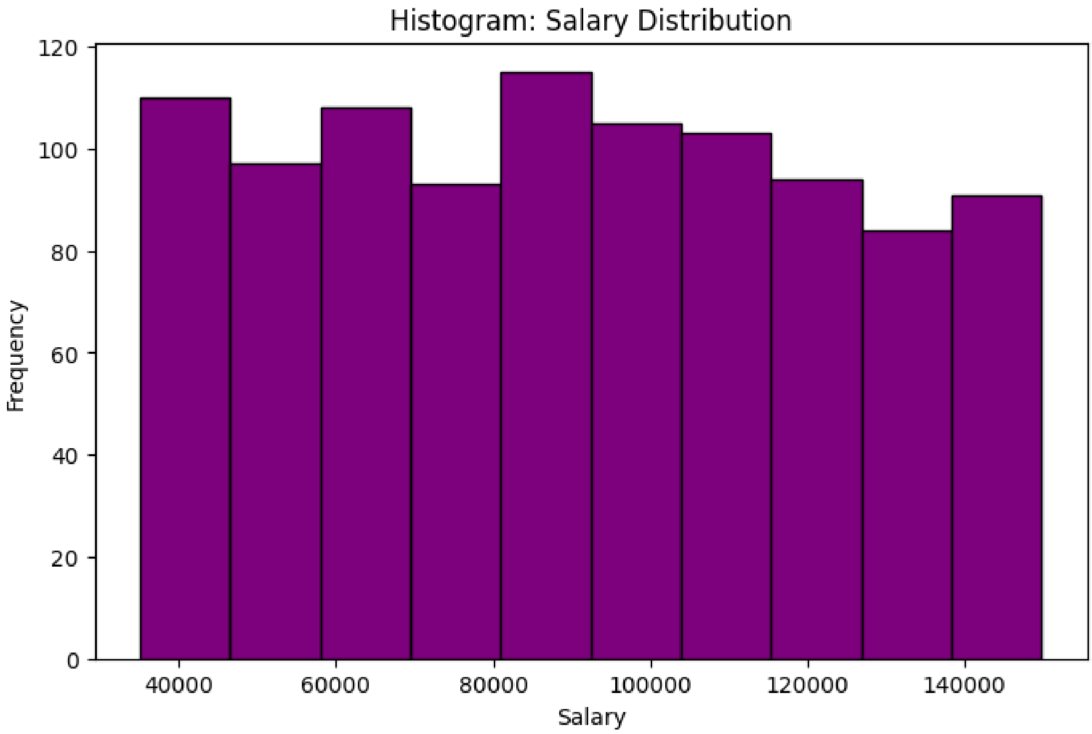
In [5]:
```
# Group by 'Team' and calculate average salary
avg_salary = df.groupby('Team')['Salary'].mean()

# Bar plot: Team vs Average Salary
plt.figure(figsize=(8, 5))
avg_salary.plot(kind='bar', color='skyblue')
plt.title('Bar Plot: Average Salary by Team')
plt.xlabel('Team')
plt.ylabel('Average Salary')
plt.xticks(rotation=45)
plt.show()
```



3. histogram

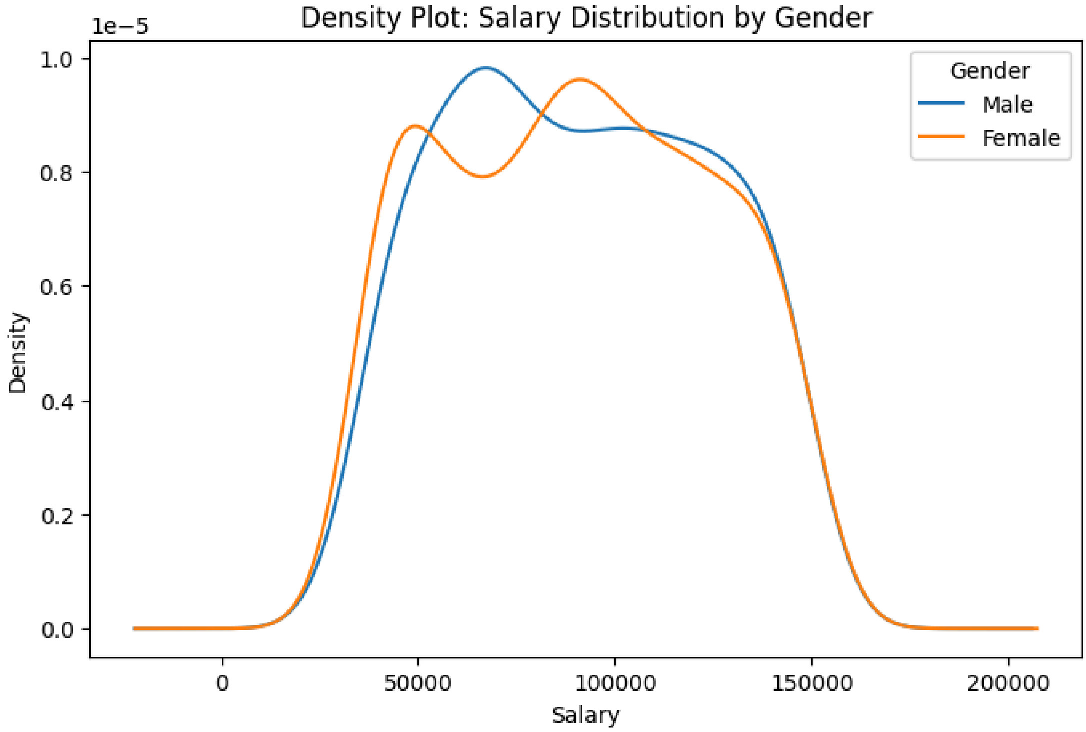
In [8]:
```
# Histogram: Distribution of Salaries
plt.figure(figsize=(8, 5))
plt.hist(df['Salary'], bins=10, color='purple', edgecolor='black')
plt.title('Histogram: Salary Distribution')
plt.xlabel('Salary')
plt.ylabel('Frequency')
plt.show()
```



4. density plot

In [12]:
```
# Density Plot: Salary Distribution by Gender
plt.figure(figsize=(8, 5))
for gender in df['Gender'].unique():
    subset = df[df['Gender'] == gender]
    subset['Salary'].plot(kind='kde', label=gender)

plt.title('Density Plot: Salary Distribution by Gender')
plt.xlabel('Salary')
plt.legend(title='Gender')
plt.show()
```



5. scatter plot

In [10]:
```
# Scatter Plot: Salary vs Bonus %
plt.figure(figsize=(8, 5))
plt.scatter(df['Salary'], df['Bonus %'], color='red', alpha=0.6)
plt.title('Scatter Plot: Salary vs Bonus %')
plt.xlabel('Salary')
plt.ylabel('Bonus %')
plt.show()
```