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

NumPy Arrays and Vectorized Computation

NumPy (Numerical Python) is a fundamental library for numerical computing in Python, providing efficient data structures and operations for large-scale numerical computations. Its core data structure, the ndarray (n-dimensional array), enables seamless storage and manipulation of complex datasets.

Key Features of NumPy:

1. Multi-dimensional arrays: Efficient storage and manipulation of large datasets.
2. Element-wise operations: Fast and efficient computations on entire arrays.
3. Mathematical functions: Comprehensive set of functions for linear algebra, Fourier transforms, random number generation, and more.
4. Integration with other languages: Tools for integrating with C, Fortran, and other languages.
5. Broadcastable operations: Arrays of different shapes can be used together in calculations.

Benefits of NumPy:

1. Performance: Significantly faster than traditional Python lists.
2. Flexibility: Handles data of varying dimensions.
3. Versatility: Essential component of the scientific Python ecosystem.

Applications of NumPy:

1. Scientific computing: Backbone for libraries like SciPy, pandas, and Matplotlib.
2. Data analysis: Efficient data manipulation and analysis.
3. Machine learning: Fast computations for complex algorithms.

1. 1 Arrays from python datastructures

In [1]:
```
# To define a new ndarray, the easiest way is to use the array() function
import numpy as np
a=[1,2,3,4,5]
b=np.array(a)
print(a)
```

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

In [2]:
```
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]:
```
#tuple to arrays conversion
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]:
```
#conversion of arrays to sets
a=[1,2,3,4,5]
c=set(a)
np.array(c)
```

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

In [5]:
```
#dictionary to arrays
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]:
```
# convert list to ndarray
a=np.array(np.arange(9))
print(a)
```

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

In [7]:
```
#zeros function
# The zeros() function, for example, creates a full array of zeros with dimensions def
a=np.zeros(3)
print(a)
```

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

In [8]:
```
b=np.zeros([3,3])
print(b)
```

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

In [9]:
```
# ones function
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]:
```
#eye function
a=np.eye(3)
print(a)
```

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

In [12]:
```
c=np.eye(3,k=1)
print(c)
```

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

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

In [13]:
```
#identity function
a=np.identity(3)
print(a)
```

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

In [14]:
```
# Return a new array of given shape and type, filled with fill_value.
d=np.full((2,2),7)
print(d)
```

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

In [15]:
```
# Return a new array with the same shape and type as a given array.
a=np.empty((2,3))
print(a)
```

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

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]:
```
#meshgrid function
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 [18]:
```
#random functions
from numpy import random

x = random.randint(100)

print(x)
```

```
88
```

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

```
b'd\r\xa32\xdd\xffB'
[['true' 'true' 'true']
 ['false' 'false' 'false']]
```

In [20]:
```
#real numbers using rand functions
x = random.rand(1) + random.rand(1)*1j
print (x)
print(x.real)
print(x.imag)
```

```
[0.16371363+0.74441673j]
[0.16371363]
[0.74441673]
```

In [21]:
```
x = random.rand(1,5) + random.rand(1,5)*1j
print (x)
```

```
[[0.64567564+0.68740431j 0.24762988+0.75057239j 0.7876683 +0.78903212j
  0.74173938+0.5255953j  0.79359481+0.91004664j]]
```

In [22]:
```
np.random.random(size=(2,2))+1j*np.random.random(size=(2,2))
```

Out[22]:
```
array([[0.66048542+0.0872412j , 0.17751447+0.95016574j],
       [0.263532  +0.22998254j, 0.26761083+0.3615973j ]])
```

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

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

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)
```

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

In [25]:
```
np.random.randint(1,5)
```

Out[25]:
```
3
```

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

```
[[ 0.02670893  0.2306991  -0.21613717  0.9290488   2.06553728  0.03327477
   0.20507394 -0.56676126 -0.26141915 -2.09193375]]
```

In [27]:
```
a=np.array(['apple','bananaa','cherry'])
b=np.random.choice(a)
print(b)
```

```
apple
```

In [28]:
```
np.random.shuffle(a)
print(a)
```

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

2.Manipulation Of Numpy Arrays

2.1 Indexing

In [29]:
```
#Integer Indexing
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]:
```
#copy function
old_values = arr3d[0].copy()
arr3d[0] = 42
print(arr3d)
```

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

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

In [33]:
```
#adding the array elements
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[2] + arr[3])
```

```
7
```

In [34]:
```
#accessing the elements in 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 [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])
```

```
6
```

In [37]:
```
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 [38]:
```
#basics os slicing
import numpy as np
arr=np.array([5,6,7,8,9])
print(arr[1:3
         ])
```

```
[6 7]
```

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

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

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

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

In [41]:
```
arr=np.array([5,6,7,8,9])
print(arr[:3
         ])
```

```
[5 6 7]
```

In [42]:
```
arr=np.array([5,6,7,8,9])
print(arr[-3:-1])
```

```
[7 8]
```

In [43]:
```
arr=np.array([5,6,7,8,9])
print(arr[:3
         ])
```

```
[5 6 7]
```

In [44]:
```
arr=np.array([5,6,7,8,9])
print(arr[:3 ])
```

```
[5 6 7]
```

In [45]:
```
arr=np.array([5,6,7,8,9])
print(arr[-3:-1])
```

```
[7 8]
```

In [46]:
```
arr=np.array([5,6,7,8,4,5,6,7,9])
print(arr[1:5:2])
```

```
[6 8]
```

In [47]:
```
arr=np.array([5,6,7,8,4,5,6,7,9])
print(arr[-1:-5:-1])
```

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

In [48]:
```
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 [49]:
```
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 [50]:
```
import numpy as np

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

print(arr[0:2, 1:4])
```

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

In [51]:
```
b = "Hello, World!"
print(b[2:5])
```

```
llo
```

In [52]:
```
b = "Hello, World!"
print(b[:5])
```

```
Hello
```

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

```
llo, World!
```

2.3 Re-Shaping

In [54]:
```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

print(arr.shape)
```

```
(2, 4)
```

In [55]:
```
import numpy as np

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]:
```
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 [57]:
```
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 [58]:
```
import numpy as np
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

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

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)
```

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

In [68]:
```
a = np.arange(12).reshape(4,3)
b=np.hsplit(a,3)
print(b)
```

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

In [69]:
```
b=np.vsplit(a,2)
print(b)
```

```
[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

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

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

```
[-1.41960791  2.26827088  0.31990201  0.56310948 -0.7423566  -2.07586439
 -0.81959863 -0.45593829]
```

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

```
[ 0.83258733  0.49336393  0.6746867  -0.76489589 -2.49153176  1.52499494
 -0.46890112 -0.62381465]
```

In [80]:
```
np.maximum(x, y)
```

Out[80]:
```
array([ 0.83258733,  2.26827088,  0.6746867 ,  0.56310948, -0.7423566 ,
        1.52499494, -0.46890112, -0.45593829])
```

In [81]:
```
arr = np.random.randn(7) * 5
remainder, whole_part = np.modf(arr)
print(remainder)
```

```
[ 0.8455496   0.03161555 -0.00360215 -0.12739546 -0.06538095 -0.0745466
 -0.31413463]
```

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

```
[ 3.  1. -2. -3. -2. -3. -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

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]:
```
0.2320345695540107
```

In [93]:
```
arr.mean(axis=1)
```

Out[93]:
```
array([ 0.47676636,  0.24977455,  0.58722875, -0.01072523, -0.14287158])
```

In [94]:
```
np.median(arr)
```

Out[94]:
```
0.5022052756149222
```

In [95]:
```
np.std(arr)
```

Out[95]: 1.0200536228259969

In [96]:
```
np.var(arr)
```

Out[96]:
```
1.0405093934404412
```

In [97]:
```
arr.sum(axis=0)
```

Out[97]:
```
array([-0.01976238,  5.46903725,  0.71695611, -1.52553959])
```

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

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))
```

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

In [106]:
```
print(np.less(a[0],b[2]))
```

```
True
```

In [107]:
```
print(np.less(a,b))
```

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

In [108]:
```
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 [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)
```

```
[[-0.30776874  0.11904534  0.47765866]
 [-0.56167285  0.34041277 -0.04536726]
 [ 1.88977446  1.10093798 -0.25129716]
 [-0.17736889 -0.12061495 -1.14960814]
 [ 0.15460593  1.45196918  0.88199296]]
```

In [114]:
```
arr.sort(1)
print(arr)
```

```
[[-0.30776874  0.11904534  0.47765866]
 [-0.56167285 -0.04536726  0.34041277]
 [-0.25129716  1.10093798  1.88977446]
 [-1.14960814 -0.17736889 -0.12061495]
 [ 0.15460593  0.88199296  1.45196918]]
```

5.2 Unique Operation

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

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

Type *Markdown* and LaTeX: α^2

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
img = Image.open('th.jpeg')
imgarr=np.array(img)
```

In [5]:
```
#displaying image
plt.imshow(imgarr)
plt.title('original image')
plt.show()
```

In [9]:
```
# gives croped image
crpimgarr=imgarr[100:300,100:500]
image = Image.fromarray(imgarr)
plt.imshow(crpimgarr)
plt.title('cropped image')
plt.show()
```

In [7]:
```
# flipped by 180 degrees
flipimg=np.flipud(imgarr)
plt.imshow(flipimg)
plt.title('flipped image')
plt.show()
```

In [8]:
```
# mirror image
flipimg=np.fliplr(imgarr)
plt.imshow(flipimg)
plt.title('mirror image')
plt.show()
```

In [ ]:

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

That's a great summary of pandas! Here's a reformatted version with added headings and bullet points for clarity:

Pandas Overview

Pandas is a powerful, open-source library for data manipulation and analysis in Python, simplifying work with structured data.

Primary Data Structures

- Series: One-dimensional labeled array
- DataFrame: Two-dimensional, size-mutable, potentially heterogeneous tabular data structure (similar to a spreadsheet or SQL table)

Key Strengths

- Intuitive syntax for data operations (filtering, grouping, aggregating)
- Robust support for handling missing data and time series data
- Seamless time-based indexing and operations
- Integration with other libraries (NumPy, Matplotlib, SciPy)

Key Features

- Data cleaning and transformation
- Data analysis and visualization
- Handling missing data
- Time series data support
- Data merging and joining
- Grouping and aggregating data
- Data filtering and sorting

Use Cases

- Exploratory data analysis
- Data cleaning and preprocessing
- Data visualization
- Machine learning data preparation
- Data science research

Benefits

- User-friendly syntax
- Efficient data manipulation
- Flexible data structures
- Extensive documentation
- Large community support

Importance in Data Science

Pandas is a cornerstone of the data science stack in Python, widely used by data scientists, analysts, and researchers worldwide.

Would you like more information on:

1. Pandas data structures (Series, DataFrame)?
2. Data operations (filtering, grouping, aggregating)?
3. Handling missing data and time series data?
4. Integration with other libraries?

1.1 Pandas Series From Python List

In [1]:
```
#using lists
import pandas as pd
import numpy as np
data=[4,7,-5,3]
a=pd.Series(data)
print(a)
```

```
0    4
1    7
2   -5
3    3
dtype: int64
```

In [6]:
```
# 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 [8]:
```
#using multiple lists
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
```

1.2 Pandas Series From Numpy arrays

In [4]:
```
#pandas using numpy arrays
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 [5]:
```
# 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    d
1004    e
dtype: object
```

In [9]:
```
#pandas series using arrays
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 [3]:
```
#
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    30
dtype: int64
```

In [22]:
```
# 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    50
dtype: int64
```

In [2]:
```
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     NaN
A    50.0
dtype: float64
```

2. Data Manipulation with Pandas Series

2.1 Indexing

In [46]:
```
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])
```

```
n
```

In [2]:
```
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 [3]:
```
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
```

2.2 Selecting

In [24]:
```
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 [19]:
```
s['b']
```

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

In [26]:
```
s[['b','a','d']]
```

Out[26]:
```
b    1.0
a    0.0
d    3.0
dtype: float64
```

In [27]:
```
s['b':'e']
```

Out[27]:
```
b    1.0
c    2.0
d    3.0
e    4.0
```

```
dtype: float64
```

In [20]:
```
s[1]
```

Out[20]:
```
1.0
```

In [21]:
```
s[2:4]
```

Out[21]:
```
c    2.0
d    3.0
dtype: float64
```

In [23]:
```
s[[1,3]]
```

Out[23]:
```
b    1.0
d    3.0
dtype: float64
```

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

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

2.3 Filtering

In [4]:
```
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 [32]:
```
s[s<2]
```

Out[32]:
```
a    0.0
dtype: float64
```

In [36]:
```
s[s>2]
```

Out[36]:
```
b    5.0
d    3.0
e    4.0
dtype: float64
```

In [35]:
```
s[s!=2]
```

Out[35]:
```
a    0.0
b    5.0
d    3.0
e    4.0
dtype: float64
```

In [38]:
```
s[(s>2)&(s<5) ]
```

Out[38]:
```
d    3.0
e    4.0
dtype: float64
```

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

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

In [7]:
```
print(s[1:2]==5)
```

```
b    True
dtype: bool
```

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

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

2.4 Arithmetic Operations

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

In [3]:
```
series3 = series1 + series2
print(series3)
```

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

In [4]:
```
series3 = series1 - series2
print(series3)
```

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

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

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

In [6]:
```
series3 = series1 /series2
print(series3)
```

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

In [9]:
```
series3 = series1 %series2
print(series3)
```

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

2.5 Ranking

In [10]:
```
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]:
```
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 [11]:
```
s.rank(method='min')
```

Out[11]:
```
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 [12]:
```
s.rank(method='max')
```

Out[12]:
```
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 [50]:
```
s.rank(method='first')
```

Out[50]:
```
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 [52]:
```
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 [8]:
```
sr.sort_values(ascending = False)
```

Out[8]:
```
2    22.7800
3    20.1240
0    19.5000
4    18.1002
1    16.8000
dtype: float64
```

In [53]:
```
sr.sort_values(ascending = True)
```

Out[53]:
```
1    16.8000
4    18.1002
0    19.5000
3    20.1240
2    22.7800
dtype: float64
```

In [55]:
```
sr.sort_index()
```

Out[55]:
```
0    19.5000
1    16.8000
2    22.7800
3    20.1240
4    18.1002
dtype: float64
```

In [58]:
```
print(sr.sort_values(kind))
```

```
1    16.8000
4    18.1002
0    19.5000
3    20.1240
2    22.7800
dtype: float64
```

2.7 checking null values

In [40]:
```
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 [42]:
```
x.isnull()
```

Out[42]:
```
california     True
ohio          False
Texas          True
oregon        False
dtype: bool
```

In [44]:
```
x.notnull()
```

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

2.8 Concatenation

In [19]:
```
# 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]:
```
display(pd.concat([series1, series2],
                  axis = 1))
```

	0	1
0	1	A
1	2	B
2	3	C

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

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

In [21]:
```
print(pd.concat([series1, series2], ignore_index=True))
```

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

In [22]:
```
print(pd.concat([series1, series2], ignore_index=False))
```

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

In [69]:
```
print(pd.concat([series1, series2], keys=['series1', 'series2']))
```

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

3 .Creating DataFrames from List and Dictionary

3.1 From List

In [16]:
```
data = [1, 2, 3, 4, 5]

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

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

In [70]:
```
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 [38]:
```
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
```

3.2 From Dictionary

In [39]:
```
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 [13]:
```
df=pd.DataFrame({'state':['AP','AP','AP','TS','TS','TS'],'year':[2000,2001,2002,2000,2001,2002],'pop':[1.5,1.7,3.6,
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 [14]:
```
df=pd.DataFrame({'a':[4,5,6],'b':[7,8,9]},index=pd.MultiIndex.from_tuples([('d',1),('d',2),('e',2)] ,names=['n','v'
print(df)
```

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

In [71]:
```
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':8.0}})
df.reindex(['a','b','c','d'])
```

Out[71]:

	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 [29]:
```
import pandas as pd
data=pd.read_csv('IRIS.csv')
data
```

Out[29]:

	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 [75]:
```
data.head(5)
```

Out[75]:

	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 [76]:
```
data.tail(5)
```

Out[76]:

	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 [77]:
```
data.shape
```

Out[77]:
```
(150, 5)
```

In [78]:
```
data.dtypes
```

Out[78]:
```
 sepal_length    float64
sepal_width      float64
petal_length     float64
petal_width      float64
species           object
dtype: object
```

In [80]:
```
data.isnull().sum()
```

Out[80]:
```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

In [81]:
```
data.columns
```

Out[81]:
```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

In [82]:
```
data.index
```

Out[82]:
```
RangeIndex(start=0, stop=150, step=1)
```

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

In [106]:
```
data.loc[data['sepal_width']>4]
```

Out[106]:

	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 [101]:
```
data.drop([0,3])
```

Out[101]:

	sepal_length	sepal_width	petal_length	petal_width	species
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	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

148 rows × 5 columns

In [24]:
```
data.drop(data[data['sepal_length']>4.3].index)
```

Out[24]:

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

In [107]:
```
data.loc[6,'petal_length']
```

Out[107]:
```
1.4
```

In [108]:
```
data.loc[11:15][['sepal_length','sepal_width']]
```

Out[108]:

	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 [83]:
```
data
```

Out[83]:

	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 [112]:
```
data.sort_index(ascending=False)
```

Out[112]:

	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 [114]:
```
data.sort_values(['petal_width']).head(6)
```

Out[114]:

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

In [116]:
```
data.sort_values(by=['petal_width','petal_length']).head(6)
```

Out[116]:

	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 [109]:
```
data.rank().head(10)
```

Out[109]:

	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 [25]:
```
data.rank().head(2)
```

Out[25]:

	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 [117]:
```
data.rank(ascending=False).head(5)
```

Out[117]:

	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 [119]:
```
data['sepal_length'].rank().head(5)
```

Out[119]:
```
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 [32]:
```
data=pd.read_csv('diabetes.csv')
data
```

Out[32]:

	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 [129]:
```
data.mean()
```

Out[129]:
```
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 [131]:
```
data.mean()[['Insulin','BMI']]
```

Out[131]:
```
Insulin    79.799479
BMI        31.992578
dtype: float64
```

In [132]:
```
data.mode()
```

Out[132]:

	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 [133]:
```
data.median()
```

Out[133]:
```
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 [134]:
```
data.std()
```

Out[134]:
```
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 [135]:
```
data.var()
```

Out[135]:
```
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 [136]:
```
data.sum()
```

Out[136]:
```
Pregnancies                 2953.000
Glucose                    92847.000
BloodPressure              53073.000
SkinThickness              15772.000
Insulin                    61286.000
BMI                        24570.300
DiabetesPedigreeFunction     362.401
Age                        25529.000
Outcome                      268.000
dtype: float64
```

In [137]:
```
data.corr()
```

Out[137]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Ou
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.2
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.4
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.0
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.0
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.1
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.2
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.1
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.2
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.0

In [140]:
```
data.var()['BMI']
```

Out[140]:
```
62.15998395738257
```

In [141]:
```
data.mean()[['BMI','Insulin','Glucose']]
```

Out[141]:
```
BMI          31.992578
Insulin      79.799479
Glucose     120.894531
dtype: float64
```

4.7 count and Uniqueness of given Categorical values

In [142]:
```
data.count()
```

Out[142]:
```
Pregnancies                 768
Glucose                     768
BloodPressure               768
SkinThickness               768
Insulin                     768
BMI                         768
DiabetesPedigreeFunction    768
Age                         768
Outcome                     768
dtype: int64
```

In [143]:
```
data.value_counts()
```

Out[143]:
```
Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI   DiabetesPedigreeFunction  Age  Outcome
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
Length: 768, dtype: int64
```

In [144]:
```
data.value_counts(data['Insulin'])
```

Out[144]:
```
Insulin
0      374
105     11
140      9
130      9
120      8
       ...
193      1
191      1
188      1
184      1
846      1
Length: 186, dtype: int64
```

In [147]:
```
data['Glucose'].unique()
```

Out[147]:
```
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)
```

In [31]:
```
categorical_columns = ['species']
for column in categorical_columns:
    unique_values = data[column].unique()
    unique_counts = data[column].value_counts()

    print(f"Column: {column}")
    print(f"Unique Values: {unique_values}")
    print(f"Value Counts:\n{unique_counts}\n")
```

```
Column: species
Unique Values: ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
Value Counts:
Iris-setosa         50
Iris-versicolor     50
Iris-virginica      50
Name: species, dtype: int64
```

In [153]:
```
print(df.unique())
```

```
[  6.    148.     72.    ...   1.057   0.766   0.171]
```

In [ ]:
```
## 4.8 Rename Single/Multiple columns
```

In [154]:
```
data
```

Out[154]:

	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 [159]:
```
data.rename(columns={'Glucose':'gluco'}).head(5)
```

Out[159]:

	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 [37]:
```
data.rename(columns={"Pregnancies":'preg',"SkinThickness":'skin',"DiabetesPedigreeFunction":'diabetes'})
```

Out[37]:

	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

3. Data cleaning and preparation

Data Cleaning and Preparation

Data cleaning and preparation are essential steps in the data analysis process, ensuring data quality and accuracy.

Data Cleaning

1. Handling missing values: replacing, imputing, or removing
2. Data normalization: scaling, standardization
3. Data transformation: aggregating, grouping, pivoting
4. Removing duplicates, outliers, and errors
5. Data quality checks: validity, consistency, completeness

Data Preparation

1. Data merging and joining: combining datasets
2. Data splitting: dividing datasets for training/testing
3. Feature engineering: creating new variables
4. Feature selection: selecting relevant variables
5. Data transformation: converting data types

Tools and Techniques

1. Pandas (Python)
2. NumPy (Python)
3. R (dplyr, tidyr)
4. SQL (querying, joining)
5. Data visualization (Matplotlib, Seaborn)

Best Practices

1. Document data sources and processing steps
2. Use version control for data and code
3. Test and validate data quality
4. Use automated scripts for reproducibility
5. Collaborate with domain experts

Common Data Quality Issues

1. Missing or duplicate data
2. Inconsistent formatting
3. Inaccurate or outdated data
4. Noisy or erroneous data
5. Data bias or skewness

Data Preparation Checklist

1. Handle missing values
2. Remove duplicates and outliers
3. Normalize and transform data
4. Merge and join datasets
5. Select relevant features
6. Split data for training/testing
7. Validate data quality

By following these guidelines, you'll ensure your data is accurate, complete, and ready for analysis.

Would you like more information on:

1. Handling missing values?
2. Data normalization techniques?
3. Feature engineering strategies?
4. Data visualization best practices?

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

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

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
# Load the CSV file into a Pandas DataFrame
df = pd.read_csv('cropproduction.csv')
df
```

Out[2]:

	State	District	Crop	Year	Season	Area	Area Units	Production	Production Units	Yield
0	Andaman and Nicobar Islands	NICOBARS	Arecanut	2001-02	Kharif	1254.0	Hectare	2061.0	Tonnes	1.643541
1	Andaman and Nicobar Islands	NICOBARS	Arecanut	2002-03	Whole Year	1258.0	Hectare	2083.0	Tonnes	1.655803
2	Andaman and Nicobar Islands	NICOBARS	Arecanut	2003-04	Whole Year	1261.0	Hectare	1525.0	Tonnes	1.209358
3	Andaman and Nicobar Islands	NORTH AND MIDDLE ANDAMAN	Arecanut	2001-02	Kharif	3100.0	Hectare	5239.0	Tonnes	1.690000
4	Andaman and Nicobar Islands	SOUTH ANDAMANS	Arecanut	2002-03	Whole Year	3105.0	Hectare	5267.0	Tonnes	1.696296
...	...	...	...	...	...	...	...	...	...	...
345402	Manipur	IMPHAL WEST	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345403	Manipur	SENAPATI	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345404	Manipur	TAMENGLONG	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345405	Manipur	THOUBAL	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345406	Manipur	UKHRUL	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN

345407 rows × 10 columns

In [3]:
```
# Display the first few rows of the DataFrame to understand the data
print("Original DataFrame:")
print(df.head())
```

```
Original DataFrame:
                          State                 District      Crop     Year  \
0  Andaman and Nicobar Islands                 NICOBARS  Arecanut  2001-02
1  Andaman and Nicobar Islands                 NICOBARS  Arecanut  2002-03
2  Andaman and Nicobar Islands                 NICOBARS  Arecanut  2003-04
3  Andaman and Nicobar Islands  NORTH AND MIDDLE ANDAMAN  Arecanut  2001-02
4  Andaman and Nicobar Islands            SOUTH ANDAMANS  Arecanut  2002-03

       Season    Area Area Units  Production Production Units     Yield
0      Kharif  1254.0     Hectare      2061.0           Tonnes  1.643541
1  Whole Year  1258.0     Hectare      2083.0           Tonnes  1.655803
2  Whole Year  1261.0     Hectare      1525.0           Tonnes  1.209358
3      Kharif  3100.0     Hectare      5239.0           Tonnes  1.690000
4  Whole Year  3105.0     Hectare      5267.0           Tonnes  1.696296
```

In [4]:
```
#Last rows of the data set
print(df.tail())
```

```
          State     District  Crop     Year  Season  Area Area Units  \
345402  Manipur  IMPHAL WEST   NaN  2019-20    Rabi   NaN    Hectare
345403  Manipur     SENAPATI   NaN  2019-20    Rabi   NaN    Hectare
345404  Manipur   TAMENGLONG   NaN  2019-20    Rabi   NaN    Hectare
345405  Manipur      THOUBAL   NaN  2019-20    Rabi   NaN    Hectare
345406  Manipur       UKHRUL   NaN  2019-20    Rabi   NaN    Hectare

        Production  Production Units  Yield
345402         NaN           Tonnes    NaN
345403         NaN           Tonnes    NaN
345404         NaN           Tonnes    NaN
345405         NaN           Tonnes    NaN
345406         NaN           Tonnes    NaN
```

In [5]:
```
# 1. Detect missing data
missing_data = df.isnull()
print("\nMissing Data:")
print(missing_data.head(10))
```

```
Missing Data:
   State  District   Crop   Year  Season   Area  Area Units  Production  \
0  False     False  False  False   False  False       False       False
1  False     False  False  False   False  False       False       False
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
5  False     False  False  False   False  False       False       False
6  False     False  False  False   False  False       False       False
7  False     False  False  False   False  False       False       False
8  False     False  False  False   False  False       False       False
9  False     False  False  False   False  False       False       False

   Production  Units  Yield
0             False  False
1             False  False
2             False  False
3             False  False
4             False  False
5             False  False
6             False  False
7             False  False
8             False  False
9             False  False
```

In [6]:
```
# No of null values
n=df.isnull().sum()
n
```

Out[6]:
```
State                 0
District              0
Crop                 32
Year                  0
Season                1
Area                 33
Area Units            0
Production         4993
Production Units      0
Yield                33
dtype: int64
```

In [7]:
```
# 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:
                          State                  District      Crop     Year  \
0  Andaman and Nicobar Islands                  NICOBARS  Arecanut  2001-02
1  Andaman and Nicobar Islands                  NICOBARS  Arecanut  2002-03
2  Andaman and Nicobar Islands                  NICOBARS  Arecanut  2003-04
3  Andaman and Nicobar Islands  NORTH AND MIDDLE ANDAMAN  Arecanut  2001-02
4  Andaman and Nicobar Islands            SOUTH ANDAMANS  Arecanut  2002-03
5  Andaman and Nicobar Islands            SOUTH ANDAMANS  Arecanut  2003-04
6  Andaman and Nicobar Islands                  NICOBARS    Banana  2002-03
7  Andaman and Nicobar Islands                  NICOBARS    Banana  2003-04
8  Andaman and Nicobar Islands            SOUTH ANDAMANS    Banana  2002-03
9  Andaman and Nicobar Islands            SOUTH ANDAMANS    Banana  2003-04

        Season    Area Area Units  Production Production Units     Yield
0       Kharif  1254.0    Hectare      2061.0           Tonnes  1.643541
1  Whole Year   1258.0    Hectare      2083.0           Tonnes  1.655803
2  Whole Year   1261.0    Hectare      1525.0           Tonnes  1.209358
3       Kharif  3100.0    Hectare      5239.0           Tonnes  1.690000
4  Whole Year   3105.0    Hectare      5267.0           Tonnes  1.696296
5  Whole Year   3118.0    Hectare      5182.0           Tonnes  1.661963
6  Whole Year    213.0    Hectare      1278.0           Tonnes  6.000000
7  Whole Year    266.0    Hectare      1763.0           Tonnes  6.627820
8  Whole Year   1524.0    Hectare     10882.0           Tonnes  7.140420
9  Whole Year   1530.0    Hectare     11558.0           Tonnes  7.554248
```

In [8]:
```
# 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['Area'].mean()
df_fillna = df.fillna({'Area': mean_chas})
print("\nDataFrame after filling missing values:")
print(df_fillna.head(10))
```

```
DataFrame after filling missing values:
                          State                  District      Crop     Year  \
0  Andaman and Nicobar Islands                  NICOBARS  Arecanut  2001-02
1  Andaman and Nicobar Islands                  NICOBARS  Arecanut  2002-03
2  Andaman and Nicobar Islands                  NICOBARS  Arecanut  2003-04
3  Andaman and Nicobar Islands  NORTH AND MIDDLE ANDAMAN  Arecanut  2001-02
4  Andaman and Nicobar Islands            SOUTH ANDAMANS  Arecanut  2002-03
5  Andaman and Nicobar Islands            SOUTH ANDAMANS  Arecanut  2003-04
6  Andaman and Nicobar Islands                  NICOBARS    Banana  2002-03
7  Andaman and Nicobar Islands                  NICOBARS    Banana  2003-04
8  Andaman and Nicobar Islands            SOUTH ANDAMANS    Banana  2002-03
9  Andaman and Nicobar Islands            SOUTH ANDAMANS    Banana  2003-04

        Season    Area Area Units  Production Production Units     Yield
0       Kharif  1254.0    Hectare      2061.0           Tonnes  1.643541
1  Whole Year   1258.0    Hectare      2083.0           Tonnes  1.655803
2  Whole Year   1261.0    Hectare      1525.0           Tonnes  1.209358
3       Kharif  3100.0    Hectare      5239.0           Tonnes  1.690000
4  Whole Year   3105.0    Hectare      5267.0           Tonnes  1.696296
5  Whole Year   3118.0    Hectare      5182.0           Tonnes  1.661963
6  Whole Year    213.0    Hectare      1278.0           Tonnes  6.000000
7  Whole Year    266.0    Hectare      1763.0           Tonnes  6.627820
8  Whole Year   1524.0    Hectare     10882.0           Tonnes  7.140420
9  Whole Year   1530.0    Hectare     11558.0           Tonnes  7.554248
```

In [9]:
```
# 4. Replace missing values conditionally
# For example, replace missing values in 'City' with 'Unknown'
df_replace = df.fillna({'Production': 'Unknown'})
print("\nDataFrame after replacing missing values:")
print(df_replace.head())
```

```
DataFrame after replacing missing values:
                          State                  District      Crop     Year  \
0  Andaman and Nicobar Islands                  NICOBARS  Arecanut  2001-02
1  Andaman and Nicobar Islands                  NICOBARS  Arecanut  2002-03
2  Andaman and Nicobar Islands                  NICOBARS  Arecanut  2003-04
3  Andaman and Nicobar Islands  NORTH AND MIDDLE ANDAMAN  Arecanut  2001-02
4  Andaman and Nicobar Islands            SOUTH ANDAMANS  Arecanut  2002-03

       Season    Area Area Units Production Production Units     Yield
0      Kharif  1254.0    Hectare     2061.0            Tonnes  1.643541
1  Whole Year  1258.0    Hectare     2083.0            Tonnes  1.655803
2  Whole Year  1261.0    Hectare     1525.0            Tonnes  1.209358
3      Kharif  3100.0    Hectare     5239.0            Tonnes  1.690000
4  Whole Year  3105.0    Hectare     5267.0            Tonnes  1.696296
```

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

In [10]:
```
import pandas as pd
# Load the CSV file into a Pandas DataFrame
# Replace 'data.csv' with the actual file path if needed
df=pd.read_csv('lungcancer.csv')
df
```

Out[10]:

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZ
0	M	69	1	2	2	1	1	2	1	
1	M	74	2	1	1	1	2	2	2	
2	F	59	1	1	1	2	1	2	1	
3	M	63	2	2	2	1	1	1	1	
4	F	63	1	2	1	1	1	1	1	
...	...	...	...	...	...	...	...	...	...	...
304	F	56	1	1	1	2	2	2	1	
305	M	70	2	1	1	1	1	2	2	
306	M	58	2	1	1	1	1	1	2	
307	M	67	2	1	2	1	1	2	2	
308	M	62	1	1	1	2	1	2	2	

309 rows × 16 columns

In [11]:
```
# Display the first few rows of the DataFrame to understand the data
print("Original DataFrame:")
print(df.head())
```

```
Original DataFrame:
  GENDER  AGE  SMOKING  YELLOW_FINGERS  ANXIETY  PEER_PRESSURE  \
0      M   69        1               2        2              1
1      M   74        2               1        1              1
2      F   59        1               1        1              2
3      M   63        2               2        2              1
4      F   63        1               2        1              1

   CHRONIC DISEASE  FATIGUE   ALLERGY   WHEEZING  ALCOHOL CONSUMING  COUGHING  \
0                1         2         1         2                  2         2
1                2         2         2         1                  1         1
2                1         2         1         2                  1         2
3                1         1         1         1                  2         1
4                1         1         1         2                  1         2

   SHORTNESS OF BREATH  SWALLOWING DIFFICULTY  CHEST PAIN LUNG_CANCER
0                    2                       2           2         YES
1                    2                       2           2         YES
2                    2                       1           2          NO
3                    1                       2           2          NO
4                    2                       1           1          NO
```

In [12]:
```
# Assume 'Price' is a column that we want to transform

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

Out[12]:

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZ
0	4761	69	1	2	2	1	1	2	1	
1	5476	74	2	1	1	1	2	2	2	
2	3481	59	1	1	1	2	1	2	1	
3	3969	63	2	2	2	1	1	1	1	
4	3969	63	1	2	1	1	1	1	1	
...	...	...	...	...	...	...	...	...	...	
304	3136	56	1	1	1	2	2	2	1	
305	4900	70	2	1	1	1	1	2	2	
306	3364	58	2	1	1	1	1	1	2	
307	4489	67	2	1	2	1	1	2	2	
308	3844	62	1	1	1	2	1	2	2	

309 rows × 16 columns

In [13]:
```
# 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['GENDER'] = df['AGE'].map(Age_category_map)
```

In [14]:
```
# Display the transformed DataFrame
print("\nDataFrame after transformation:")
print(df.head())
```

```
DataFrame after transformation:
  GENDER  AGE  SMOKING  YELLOW_FINGERS  ANXIETY  PEER_PRESSURE  \
0    NaN   69        1               2        2              1
1    NaN   74        2               1        1              1
2    NaN   59        1               1        1              2
3    NaN   63        2               2        2              1
4    NaN   63        1               2        1              1

   CHRONIC DISEASE  FATIGUE   ALLERGY   WHEEZING  ALCOHOL CONSUMING  COUGHING  \
0                1         2         1         2                  2         2
1                2         2         2         1                  1         1
2                1         2         1         2                  1         2
3                1         1         1         1                  2         1
4                1         1         1         2                  1         2

   SHORTNESS OF BREATH  SWALLOWING DIFFICULTY  CHEST PAIN LUNG_CANCER
0                    2                      2           2         YES
1                    2                      2           2         YES
2                    2                      1           2          NO
3                    1                      2           2          NO
4                    2                      1           1          NO
```

c) Detect and filter outliers

In [15]:
```
# Load the CSV file into a Pandas DataFrame
# Replace 'data.csv' with the actual file path if needed
df = pd.read_csv('Indianagriculture.csv')
df
```

Out[15]:

	Dist Code	Year	State Code	State Name	Dist Name	RICE AREA (1000 ha)	RICE PRODUCTION (1000 tons)	RICE YIELD (Kg per ha)	WHEAT AREA (1000 ha)	WHEAT PRODUCTION (1000 tons)	...	SUGARCA YIELD (per
0	1	1966	14	Chhattisgarh	Durg	548.00	185.00	337.59	44.00	20.00	...	1777
1	1	1967	14	Chhattisgarh	Durg	547.00	409.00	747.71	50.00	26.00	...	1500
2	1	1968	14	Chhattisgarh	Durg	556.30	468.00	841.27	53.70	30.00	...	1000
3	1	1969	14	Chhattisgarh	Durg	563.40	400.80	711.40	49.40	26.50	...	1900
4	1	1970	14	Chhattisgarh	Durg	571.60	473.60	828.55	44.20	29.00	...	2000
...	...	...	...	...	...	...	...	...	...	...	...	
16141	917	2013	15	Jharkhand	Singhbhum	267.06	579.70	2170.67	1.53	1.85	...	0
16142	917	2014	15	Jharkhand	Singhbhum	256.33	586.63	2288.57	5.36	6.65	...	0
16143	917	2015	15	Jharkhand	Singhbhum	263.21	264.71	1005.70	1.99	1.82	...	0
16144	917	2016	15	Jharkhand	Singhbhum	224.05	319.01	1423.84	0.38	0.83	...	0
16145	917	2017	15	Jharkhand	Singhbhum	386.91	669.97	1731.62	0.00	0.00	...	0

16146 rows × 80 columns

In [16]:
```
# Display the first few rows of the DataFrame to understand the data
print("Original DataFrame:")
print(df.head())
```

```
Original DataFrame:
   Dist Code  Year  State Code    State Name Dist Name  RICE AREA (1000 ha)  \
0          1  1966          14  Chhattisgarh      Durg                548.0
1          1  1967          14  Chhattisgarh      Durg                547.0
2          1  1968          14  Chhattisgarh      Durg                556.3
3          1  1969          14  Chhattisgarh      Durg                563.4
4          1  1970          14  Chhattisgarh      Durg                571.6

   RICE PRODUCTION (1000 tons)  RICE YIELD (Kg per ha)  WHEAT AREA (1000 ha)  \
0                        185.0                  337.59                  44.0
1                        409.0                  747.71                  50.0
2                        468.0                  841.27                  53.7
3                        400.8                  711.40                  49.4
4                        473.6                  828.55                  44.2

   WHEAT PRODUCTION (1000 tons)  ...  SUGARCANE YIELD (Kg per ha)  \
0                          20.0  ...                      1777.78
1                          26.0  ...                      1500.00
2                          30.0  ...                      1000.00
3                          26.5  ...                      1900.00
4                          29.0  ...                      2000.00

   COTTON AREA (1000 ha)  COTTON PRODUCTION (1000 tons)  \
0                    0.0                            0.0
1                    0.0                            0.0
2                    0.0                            0.0
3                    0.0                            0.0
4                    0.0                            0.0

   COTTON YIELD (Kg per ha)  FRUITS AREA (1000 ha)  VEGETABLES AREA (1000 ha)  \
0                       0.0                   5.95                       6.64
1                       0.0                   5.77                       7.24
2                       0.0                   5.41                       7.40
3                       0.0                   5.52                       7.16
4                       0.0                   5.45                       7.19

   FRUITS AND VEGETABLES AREA (1000 ha)  POTATOES AREA (1000 ha)  \
0                                 12.59                     0.01
1                                 13.02                     0.01
2                                 12.81                     0.10
3                                 12.69                     0.01
4                                 12.64                     0.02

   ONION AREA (1000 ha)  FODDER AREA (1000 ha)
0                  0.60                  0.47
1                  0.56                  1.23
2                  0.58                  1.02
3                  0.56                  0.84
4                  0.52                  0.42

[5 rows x 80 columns]
```

In [17]:
```
# Select the column to analyze for outliers (replace 'Value' with the actual column name)
column_name = 'Year'
```

In [18]:
```
# Calculate the z-scores for the selected column
import numpy as np
z_scores = np.abs((df[column_name] - df[column_name].mean()) / df[column_name].std())
z_scores.head(10)
```

Out[18]:
```
0    1.698523
1    1.631906
2    1.565289
3    1.498672
4    1.432055
5    1.365438
6    1.298821
7    1.232204
8    1.165587
9    1.098970
Name: Year, dtype: float64
```

d) perform vectorized string operations on pandas series

In [22]:
```
# Load the CSV file into a Pandas DataFrame
df = pd.read_csv('cropproduction.csv')
df
```

Out[22]:

	State	District	Crop	Year	Season	Area	Area Units	Production	Production Units	Yield
0	Andaman and Nicobar Islands	NICOBARS	Arecanut	2001-02	Kharif	1254.0	Hectare	2061.0	Tonnes	1.643541
1	Andaman and Nicobar Islands	NICOBARS	Arecanut	2002-03	Whole Year	1258.0	Hectare	2083.0	Tonnes	1.655803
2	Andaman and Nicobar Islands	NICOBARS	Arecanut	2003-04	Whole Year	1261.0	Hectare	1525.0	Tonnes	1.209358
3	Andaman and Nicobar Islands	NORTH AND MIDDLE ANDAMAN	Arecanut	2001-02	Kharif	3100.0	Hectare	5239.0	Tonnes	1.690000
4	Andaman and Nicobar Islands	SOUTH ANDAMANS	Arecanut	2002-03	Whole Year	3105.0	Hectare	5267.0	Tonnes	1.696296
...	...	...	...	...	...	...	...	...	...	...
345402	Manipur	IMPHAL WEST	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345403	Manipur	SENAPATI	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345404	Manipur	TAMENGLONG	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345405	Manipur	THOUBAL	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345406	Manipur	UKHRUL	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN

345407 rows × 10 columns

In [23]:
```
# Assuming 'Name' is the column containing strings

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

Out[23]:

	State	District	Crop	Year	Season	Area	Area Units	Production	Production Units	Yield
0	NICOBARS	NICOBARS	Arecanut	2001-02	Kharif	1254.0	Hectare	2061.0	Tonnes	1.643541
1	NICOBARS	NICOBARS	Arecanut	2002-03	Whole Year	1258.0	Hectare	2083.0	Tonnes	1.655803
2	NICOBARS	NICOBARS	Arecanut	2003-04	Whole Year	1261.0	Hectare	1525.0	Tonnes	1.209358
3	NORTH AND MIDDLE ANDAMAN	NORTH AND MIDDLE ANDAMAN	Arecanut	2001-02	Kharif	3100.0	Hectare	5239.0	Tonnes	1.690000
4	SOUTH ANDAMANS	SOUTH ANDAMANS	Arecanut	2002-03	Whole Year	3105.0	Hectare	5267.0	Tonnes	1.696296
...	...	...	...	...	...	...	...	...	...	...
345402	IMPHAL WEST	IMPHAL WEST	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345403	SENAPATI	SENAPATI	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345404	TAMENGLONG	TAMENGLONG	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345405	THOUBAL	THOUBAL	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345406	UKHRUL	UKHRUL	NaN	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN

345407 rows × 10 columns

In [24]:
```
# Split the names based on a delimiter (e.g., space) and create a new column for the first part of
df['Crop'] = df['Season'].str.split(' ').str[0]
df
```

Out[24]:

	State	District	Crop	Year	Season	Area	Area Units	Production	Production Units	Yield
0	NICOBARS	NICOBARS	Kharif	2001-02	Kharif	1254.0	Hectare	2061.0	Tonnes	1.643541
1	NICOBARS	NICOBARS	Whole	2002-03	Whole Year	1258.0	Hectare	2083.0	Tonnes	1.655803
2	NICOBARS	NICOBARS	Whole	2003-04	Whole Year	1261.0	Hectare	1525.0	Tonnes	1.209358
3	NORTH AND MIDDLE ANDAMAN	NORTH AND MIDDLE ANDAMAN	Kharif	2001-02	Kharif	3100.0	Hectare	5239.0	Tonnes	1.690000
4	SOUTH ANDAMANS	SOUTH ANDAMANS	Whole	2002-03	Whole Year	3105.0	Hectare	5267.0	Tonnes	1.696296
...	...	...	...	...	...	...	...	...	...	...
345402	IMPHAL WEST	IMPHAL WEST	Rabi	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345403	SENAPATI	SENAPATI	Rabi	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345404	TAMENGLONG	TAMENGLONG	Rabi	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345405	THOUBAL	THOUBAL	Rabi	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN
345406	UKHRUL	UKHRUL	Rabi	2019-20	Rabi	NaN	Hectare	NaN	Tonnes	NaN

345407 rows × 10 columns

In [26]:
```
print(df.head())
```

```
                      State                 District    Crop     Year  \
0                  NICOBARS                  NICOBARS  Kharif  2001-02
1                  NICOBARS                  NICOBARS   Whole  2002-03
2                  NICOBARS                  NICOBARS   Whole  2003-04
3  NORTH AND MIDDLE ANDAMAN  NORTH AND MIDDLE ANDAMAN  Kharif  2001-02
4             SOUTH ANDAMANS             SOUTH ANDAMANS   Whole  2002-03

        Season    Area Area Units  Production Production Units     Yield
0       Kharif  1254.0    Hectare      2061.0           Tonnes  1.643541
1  Whole Year  1258.0    Hectare      2083.0           Tonnes  1.655803
2  Whole Year  1261.0    Hectare      1525.0           Tonnes  1.209358
3       Kharif  3100.0    Hectare      5239.0           Tonnes  1.690000
4  Whole Year  3105.0    Hectare      5267.0           Tonnes  1.696296
```

In [ ]:

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 [1]:
```
import pandas as pd
df1 = pd.DataFrame({'A': ['A0', 'A1'],# Column 'A' with values 'A0', 'A1'
 'B': ['B0', 'B1']})# Column 'B' with values 'B0', 'B1'
 # Create the second DataFrame (df2) with columns 'A' and 'B' and two rows
df2 = pd.DataFrame({'A': ['A2', 'A3'],
 'B': ['B2', 'B3']})
# 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 [2]:
```
df1
```

Out[2]:

	A	B
0	A0	B0
1	A1	B1

In [3]:
```
df2
```

Out[3]:

	A	B
0	A2	B2
1	A3	B3

In [4]:
```
result
```

Out[4]:

	A	B
0	A0	B0
1	A1	B1
0	A2	B2
1	A3	B3

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

In [6]:
```
df1
```

Out[6]:

	key	value1
0	A	1
1	B	2
2	C	3

In [7]:
```
df2
```

Out[7]:

	key	value2
0	B	4
1	C	5
2	D	6

In [8]:
```
result
```

Out[8]:

	key	value1	value2
0	B	2	4
1	C	3	5

In [9]:
```
import pandas as pd
# Create DataFrame 1
df1 = pd.DataFrame({'key1': ['A', 'B', 'C'], 'value1': [1, 2, 3]})
# Create DataFrame 2
df2 = pd.DataFrame({'key2': ['B', 'C', 'D'], 'value2': [4, 5, 6]})
# Merge DataFrames on specified keys using inner join
result = pd.merge(df1, df2, left_on='key1', right_on='key2', how='inner')
```

In [10]:
```
df1
```

Out[10]:

	key1	value1
0	A	1
1	B	2
2	C	3

In [11]:
```
df2
```

Out[11]:

	key2	value2
0	B	4
1	C	5
2	D	6

In [12]:
```
result
```

Out[12]:

	key1	value1	key2	value2
0	B	2	B	4
1	C	3	C	5

In [13]:
```
import pandas as pd
# Original data
data1 = {'key1': ['A', 'B', 'C'], 'value1': [1, 2, 3]}
data2 = {'key2': ['B', 'C', 'D'], 'value2': [4, 5, 6]}
df1  =  pd.DataFrame(data1)
df2  =  pd.DataFrame(data2)
# Merge the two DataFrames
result = pd.merge(df1, df2, left_on='key1', right_on='key2', how='inner')
# Reshape the result using pivot
reshaped_result = result.pivot(index='key1', columns='key2', values=['value1','value2'])
reshaped_result
```

Out[13]:

	value1		value2	
key2	B	C	B	C
key1				
B	2.0	NaN	4.0	NaN
C	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.

Type *Markdown* and LaTeX: α^2

In [14]:
```
import pandas as pd
# Create DataFrame 1
df1 = pd.DataFrame({"A": ["A0", "A1", "A2"], "B": ["B0", "B1", "B2"]},
index=["K0", "K1", "K2"])
# Create DataFrame 2
df2 = pd.DataFrame({"C": ["C0", "C2", "C3"], "D": ["D0", "D2", "D3"]},
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)
```

```
     A   B
K0  A0  B0
K1  A1  B1
K2  A2  B2
     C   D
K0  C0  D0
K2  C2  D2
K3  C3  D3
     A   B    C    D
K0  A0  B0   C0   D0
K1  A1  B1  NaN  NaN
K2  A2  B2   C2   D2
```

INNER JOIN:

Returns rows with matching keys in both DataFrames.

In [15]:
```
#inner join
df4 = df1.join(df2, how='inner')
print(df4)
```

```
     A   B   C   D
K0  A0  B0  C0  D0
```

K2 A2 B2 C2 D2

FULL OUTER JOIN:

Returns all rows from both DataFrames.

In [16]:
```
# full outer join
df5 = df1.join(df2, how='outer')
print(df5)
```

```
      A    B    C    D
K0   A0   B0   C0   D0
K1   A1   B1  NaN  NaN
K2   A2   B2   C2   D2
K3  NaN  NaN   C3   D3
```

LEFT OUTER JOIN:

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

In [17]:
```
#Left outer join
df6 = df1.join(df2, how='left')
print(df6)
```

```
     A   B    C    D
K0  A0  B0   C0   D0
K1  A1  B1  NaN  NaN
K2  A2  B2   C2   D2
```

RIGHT OUTER JOIN

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

In [18]:
```
#right outer join
df7 = df1.join(df2, how='right')
print(df7)
```

```
      A    B   C   D
K0   A0   B0  C0  D0
K2   A2   B2  C2  D2
K3  NaN  NaN  C3  D3
```

RESHAPE

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

In [19]:
```
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 [20]:
```
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 [21]:
```
#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'),colu
data
```

Out[21]:

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

In [22]:
```
result=data.stack()
result
```

Out[22]:
```
fruit   color
apple   red      0
        green    1
        blue     2
cherry  red      3
        green    4
        blue     5
dtype: int32
```

In [23]:
```
result.unstack(0)
```

Out[23]:

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

In [24]:
```
result.unstack('fruit')
```

Out[24]:

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 [25]:
```
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[25]:

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 [26]:
```
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[26]:

Status	No	Yes
Category		
A	1	2
B	1	1

5. Data visualization on any sampledataset using matplotlib

Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data. Here are some key aspects of data visualization:

Importance of Data Visualization Simplifies Complex Data: Visualizations make it easier to understand complex datasets and convey information quickly. Identifies Patterns and Trends: Visual representation helps in spotting trends and patterns that might be missed in raw data. Improves Decision-Making: Effective visualizations support informed decision-making by presenting data in a clear and concise manner. Enhances Communication: Visualizations can convey messages more effectively than tables or text, making it easier to share insights with others.

In [7]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Read the CSV file containing crop production data into a DataFrame
df = pd.read_csv('climate change impact.csv')

# Display the first few rows of the DataFrame
a=df.head()
print(a)
```

```
   Year Country         Region  Crop_Type  Average_Temperature_C  \
0  2001   India     West Bengal       Corn                   1.55
1  2024   China           North       Corn                   3.23
2  2001  France  Ile-de-France      Wheat                  21.11
3  2001  Canada       Prairies     Coffee                  27.85
4  1998   India     Tamil Nadu  Sugarcane                   2.19

   Total_Precipitation_mm  CO2_Emissions_MT  Crop_Yield_MT_per_HA  \
0                  447.06             15.22                 1.737
1                 2913.57             29.82                 1.737
2                 1301.74             25.75                 1.719
3                 1154.36             13.91                 3.890
4                 1627.48             11.81                 1.080

   Extreme_Weather_Events  Irrigation_Access_%  Pesticide_Use_KG_per_HA  \
0                       8                14.54                    10.08
1                       8                11.05                    33.06
2                       5                84.42                    27.41
3                       5                94.06                    14.38
4                       9                95.75                    44.35

   Fertilizer_Use_KG_per_HA  Soil_Health_Index Adaptation_Strategies  \
0                     14.78              83.25      Water Management
1                     23.25              54.02         Crop Rotation
2                     65.53              67.78      Water Management
3                     87.58              91.39         No Adaptation
4                     88.08              49.61         Crop Rotation

   Economic_Impact_Million_USD
0                       808.13
1                       616.22
2                       796.96
3                       790.32
4                       401.72
```

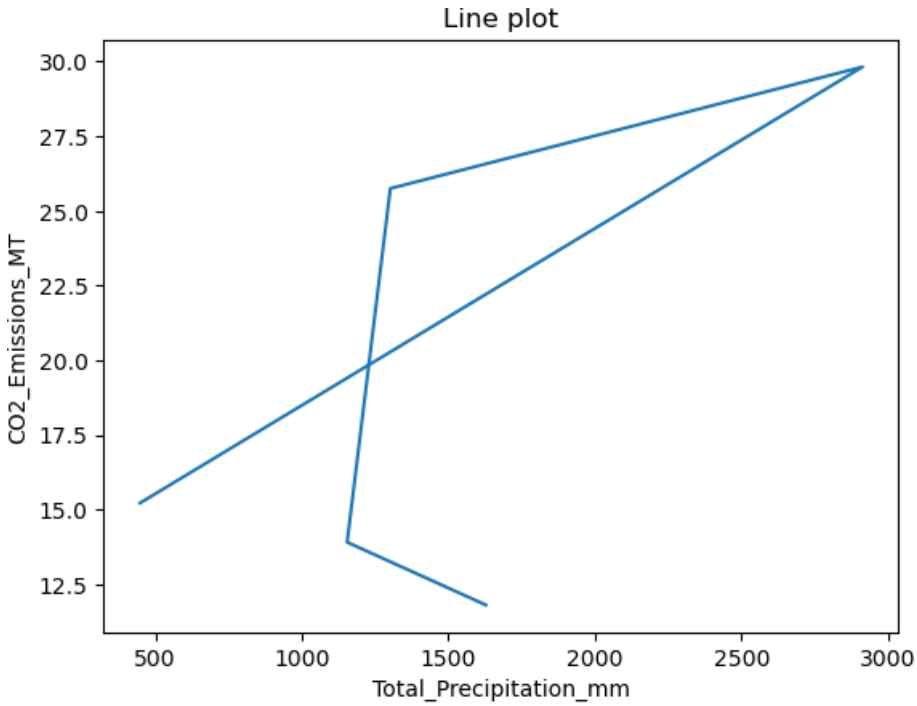
In [8]:
```
print(df.tail())
```

```
      Year    Country              Region Crop_Type   Average_Temperature_C  \
9995  2022     France  Nouvelle-Aquitaine    Cotton                  30.48
9996  1999  Australia          Queensland  Soybeans                   9.53
9997  2000  Argentina           Patagonia    Coffee                  31.92
9998  1996     Brazil           Southeast  Soybeans                  13.95
9999  2015      China               South      Corn                  11.78

      Total_Precipitation_mm   CO2_Emissions_MT  Crop_Yield_MT_per_HA  \
9995                  685.93             17.64                 3.033
9996                 2560.38             10.68                 2.560
9997                  357.76             26.01                 1.161
9998                 1549.52             17.31                 3.348
9999                 1676.25              5.34                 3.710

      Extreme_Weather_Events   Irrigation_Access_%   Pesticide_Use_KG_per_HA  \
9995                       9                 27.56                    41.96
9996                       4                 77.02                     5.45
9997                      10                 78.53                    11.94
9998                       2                 42.65                    44.71
9999                       5                 46.41                    48.28

       Fertilizer_Use_KG_per_HA  Soil_Health_Index  Adaptation_Strategies  \
9995                      10.95              43.41         No Adaptation
9996                      82.32              59.39         No Adaptation
9997                      26.00              41.46      Water Management
9998                      25.07              75.10         Crop Rotation
9999                      98.27              59.38      Water Management

       Economic_Impact_Million_USD
9995                       1483.06
9996                        829.61
9997                        155.99
9998                       1613.90
9999                        453.14
```

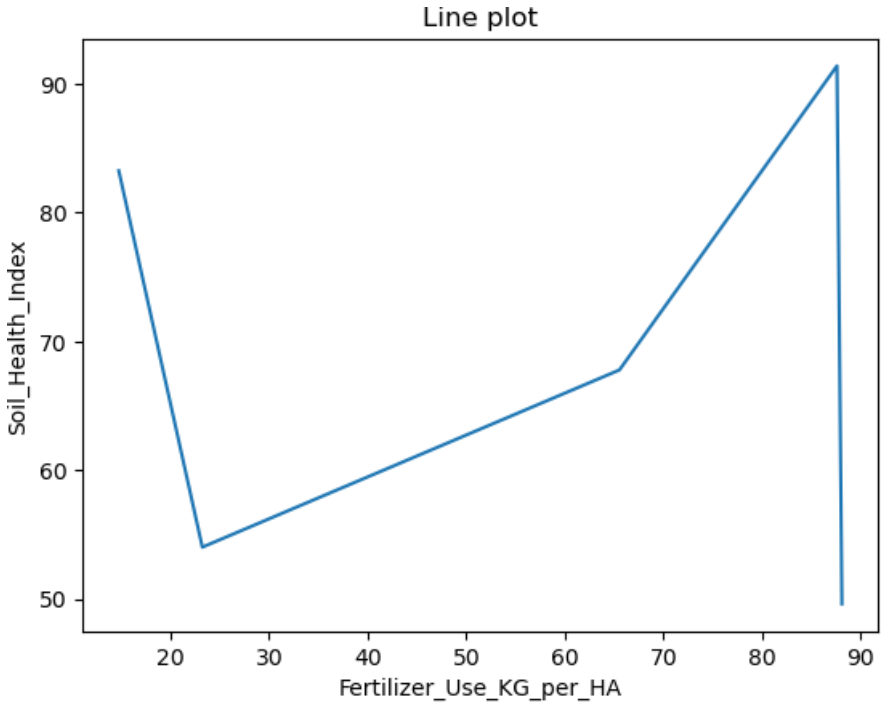
(a) LinePlot

A line plot (or line graph) is a type of data visualization that displays information as a series of data points called 'markers' connected by straight line segments. Line plots are particularly effective for illustrating trends over time, making them a popular choice for time series data or any dataset where the relationship between two variables needs to be analyzed.

In [14]:
```
# Assuming 'df' is your DataFrame and it contains columns 'Age_Group' and 'Family_Background'
plt.plot(a['Total_Precipitation_mm'], a['CO2_Emissions_MT'])
plt.xlabel('Total_Precipitation_mm')
plt.ylabel('CO2_Emissions_MT')
plt.title('Line plot')
plt.show()
```



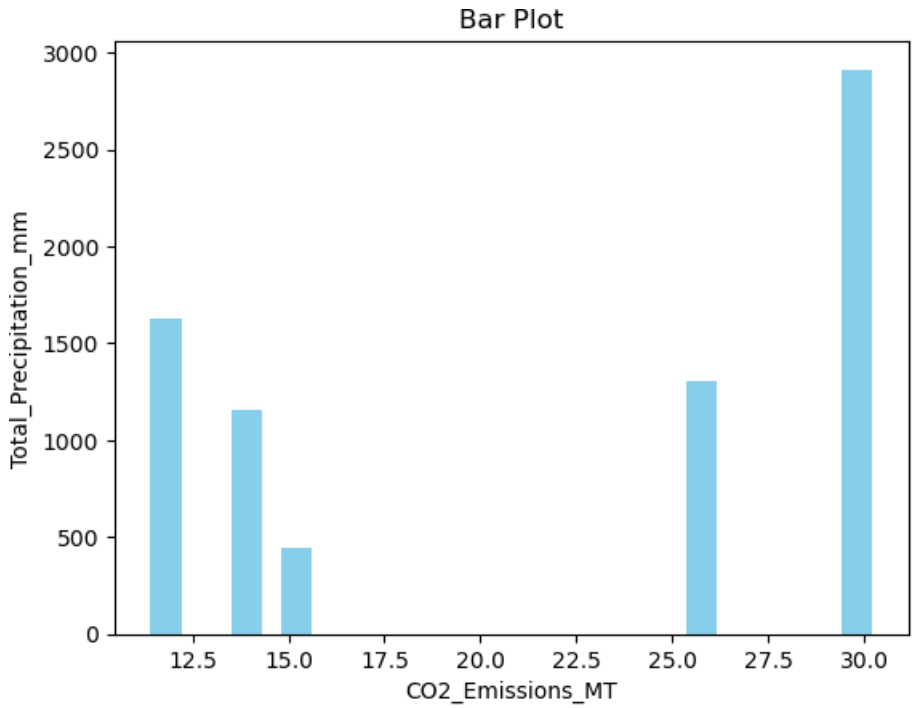
In [16]:
```
plt.plot(a['Fertilizer_Use_KG_per_HA'], a['Soil_Health_Index'])
plt.xlabel('Fertilizer_Use_KG_per_HA')
plt.ylabel('Soil_Health_Index')
plt.title('Line plot')
plt.show()
```



(b) BarPlot

A bar plot (or bar chart) is a type of data visualization that represents categorical data with rectangular bars. The length of each bar is proportional to the value it represents. Bar plots are commonly used to compare different groups or categories.

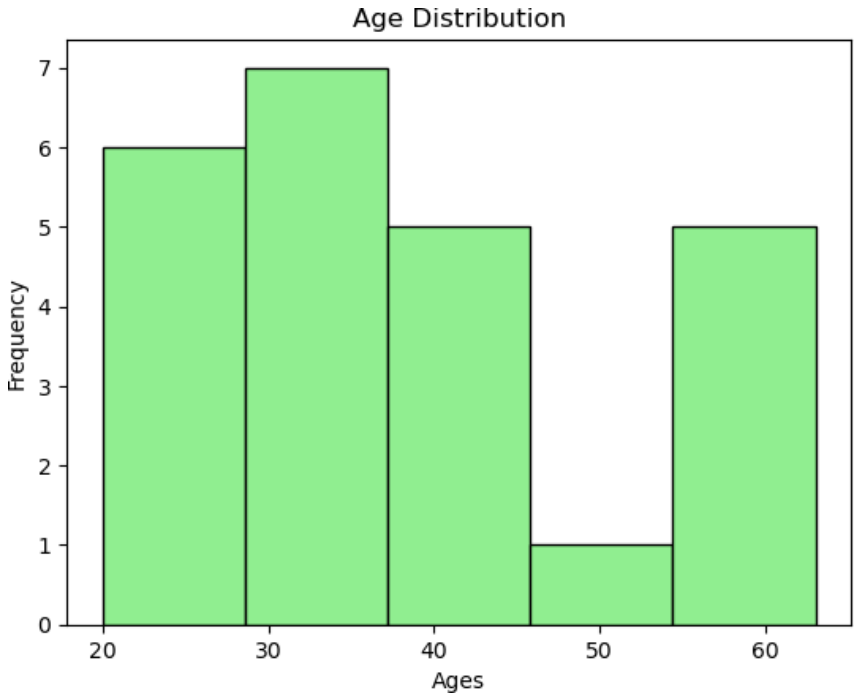
In [23]:
```
# Creating the bar plot
plt.bar(a['CO2_Emissions_MT'], a['Total_Precipitation_mm'] , color='skyblue')

# Adding title and labels
plt.title('Bar Plot')
plt.xlabel('CO2_Emissions_MT')
plt.ylabel('Total_Precipitation_mm')

# Displaying the plot
plt.show()
```



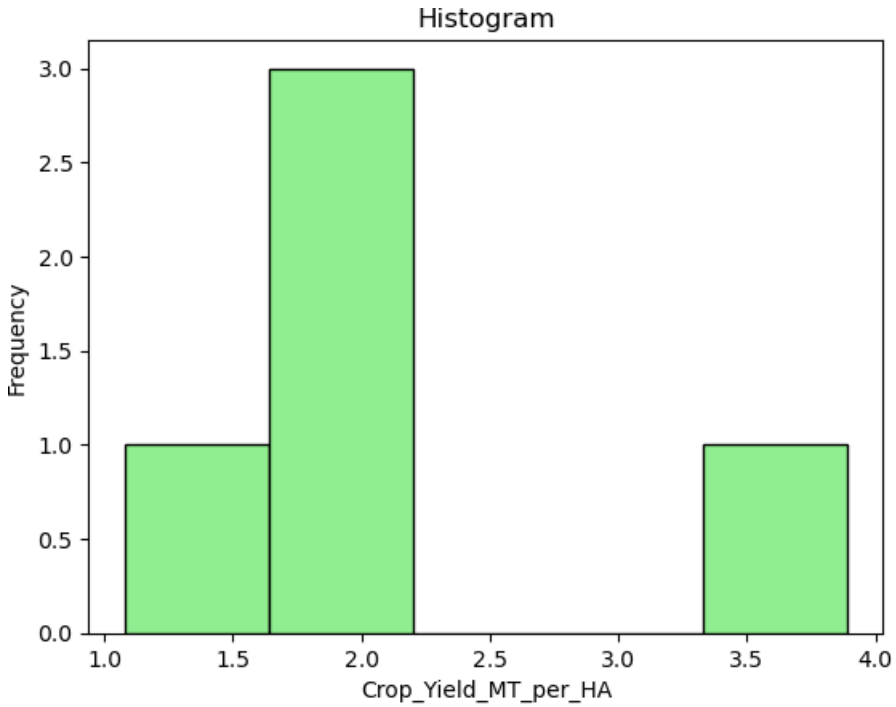
(c) Histogram

A histogram is a graphical representation that organizes a group of data points into user-specified ranges (bins). It shows the frequency of data points falling within each bin, making it easy to see the distribution of the data.

In [32]:
```
import matplotlib.pyplot as plt

# Sample dataset: ages of individuals
ages = [22, 25, 30, 35, 40, 45, 50, 55, 60, 30, 35, 40, 20, 30, 25, 22, 55, 60, 63, 45, 30, 25, 35

# Creating the histogram
plt.hist(ages, bins=5, color='lightgreen', edgecolor='black')

# Adding title and labels
plt.title('Age Distribution')
plt.xlabel('Ages')
plt.ylabel('Frequency')

# Displaying the plot
plt.show()
```



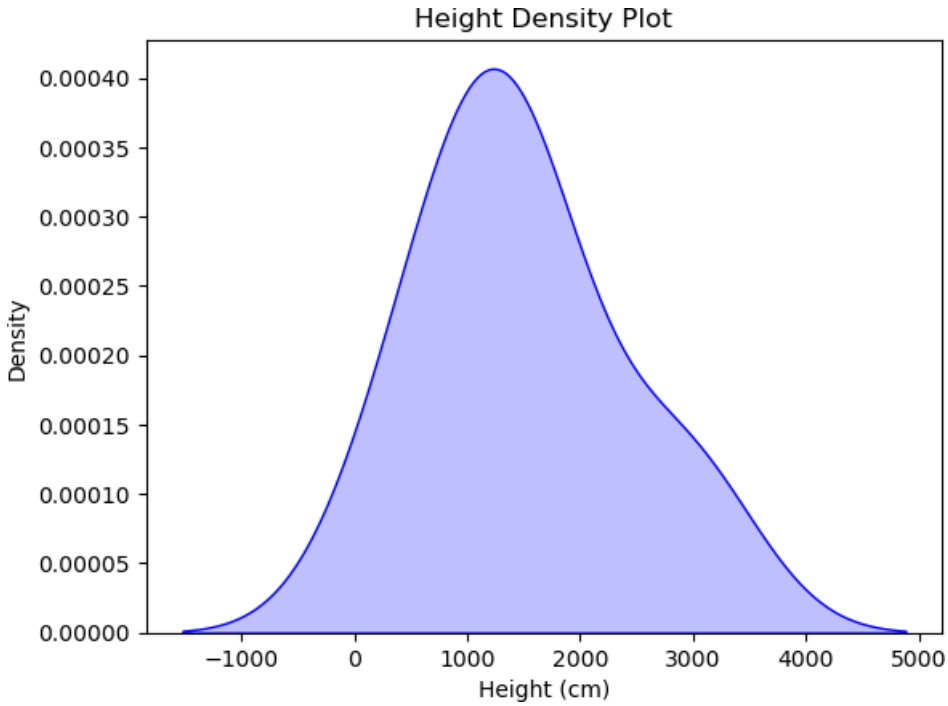
In [29]:
```
# Creating the histogram
plt.hist(a['Crop_Yield_MT_per_HA'], bins=5, color='lightgreen', edgecolor='black')

# Adding title and labels
plt.title('Histogram')
plt.xlabel('Crop_Yield_MT_per_HA')
plt.ylabel('Frequency')

# Displaying the plot
plt.show()
```



(d) DensityPlot

A density plot (or Kernel Density Estimate plot) is a smooth representation of the distribution of a dataset. It provides an estimation of the probability density function of the variable, allowing for a clearer visualization of data distribution compared to histograms.

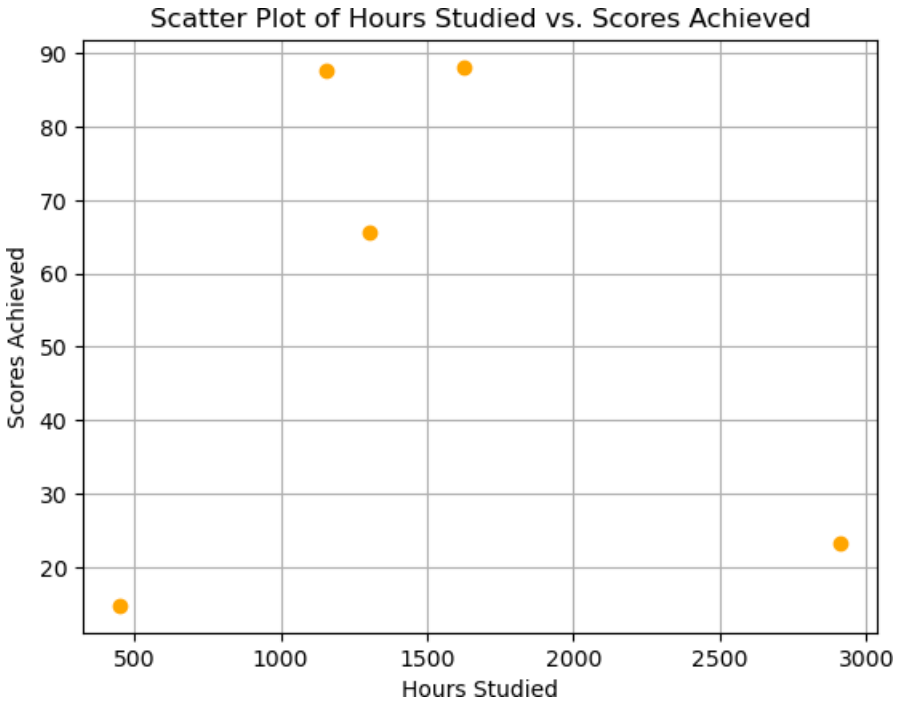
In [33]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Sample dataset: heights of individuals in centimeters
heights = [150, 155, 160, 162, 165, 170, 172, 175, 180, 182, 185, 190, 192, 195, 198, 200]

# Creating the density plot
sns.kdeplot(a['Total_Precipitation_mm'], color='blue', fill=True)

# Adding title and labels
plt.title('Height Density Plot')
plt.xlabel('Height (cm)')
plt.ylabel('Density')

# Displaying the plot
plt.show()
```



(e) ScatterPlot

A scatter plot is a type of plot that uses dots to represent the values obtained for two different variables—one plotted along the x-axis and the other plotted along the y-axis. It is useful for observing relationships and correlations between the two variables.

In [35]:
```
import matplotlib.pyplot as plt

# Sample dataset: hours studied vs. scores achieved
hours_studied = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
scores_achieved = [15, 30, 40, 55, 60, 75, 80, 85, 90, 95]

# Creating the scatter plot
plt.scatter(a['Total_Precipitation_mm'], a['Fertilizer_Use_KG_per_HA'], color='orange', marker='o'

# Adding title and labels
plt.title('Scatter Plot of Hours Studied vs. Scores Achieved')
plt.xlabel('Hours Studied')
plt.ylabel('Scores Achieved')

# Displaying the plot
plt.grid(True)  # Optional: Add grid lines for better readability
plt.show()
```



In [ ]: