NumPy Arrays and Vectorized Computation

1.Numpy module:

NumPy, short for Numerical Python, is a fundamental library for numerical computing in Python. It provides powerful data structures, primarily the ndarray (n-dimensional array), which enables efficient storage and manipulation of large datasets. With its support for multi-dimensional arrays, NumPy allows users to perform complex mathematical operations with ease.

One of the key features of NumPy is its ability to perform element-wise operations on arrays, which is significantly faster than using traditional Python lists. This efficiency stems from its implementation in C, allowing for lower-level optimizations. NumPy also includes a comprehensive set of mathematical functions that can operate on arrays, including linear algebra, Fourier transforms, and random number generation.

In addition to its array capabilities, NumPy provides tools for integrating with other languages, such as C and Fortran, making it a versatile choice for performance-critical applications. It serves as the backbone for many other scientific computing libraries, including SciPy, pandas, and Matplotlib, establishing itself as an essential component of the scientific Python ecosystem.

NumPy’s array operations are broadcastable, meaning that arrays of different shapes can still be used together in calculations, making it easier to handle data of varying dimensions. This flexibility is particularly useful in data analysis and machine learning tasks.

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

```
import numpy as np

# Creating a NumPy array
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(arr)

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


import pandas as pd
import numpy as np

# Creating a dataframe with values ranging from 0 to 98
data = np.arange(99).reshape(11, 9) df =
pd.DataFrame(data) print(df)
     0   1   2   3   4   5   6   7   8
0    0   1   2   3   4   5   6   7   8
1    9  10  11  12  13  14  15  16  17
2    18  19  20  21  22  23  24  25  26
3    27  28  29  30  31  32  33  34  35
4    36  37  38  39  40  41  42  43  44
5    45  46  47  48  49  50  51  52  53
6    54  55  56  57  58  59  60  61  62
7    63  64  65  66  67  68  69  70  71
8    72  73  74  75  76  77  78  79  80
9    81  82  83  84  85  86  87  88  89
10   90  91  92  93  94  95  96  97  98
import pandas as pd
import numpy as np

# Create a DataFrame with NaN values
data = {
    'S': [2223, 3445, np.nan, 3411, 6223, 8334, 2155, np.nan, 3314,
3210]
}
df = pd.DataFrame(data)
```

```
# Sort in ascending order, keeping NaN values in place
sorted_df = df.sort_values(by='S', na_position='last')
print("Sorted (Ascending Order):")
print(sorted_df)

# Rank (sorting in descending order)
ranked_df = df.sort_values(by='S', ascending=False,
na_position='last')
print("\nRanked (Descending Order):")
print(ranked_df)
Sorted          (Ascending
Order):
         S
6  2155.0
0  2223.0
9  3210.0
8  3314.0
3  3411.0
1  3445.0
4  6223.0
5  8334.0
2     NaN
7     NaN
Ranked          (Descending
Order):
         S
5  8334.0
4  6223.0
1  3445.0
3  3411.0
8  3314.0
9  3210.0
0  2223.0
6  2155.0
2     NaN
7     NaN
import pandas as pd
import numpy as np

# Create a DataFrame with NaN values
data = {
    'S': [2223, 3445, np.nan, 3411, 6223, 8334, 2155, np.nan, 3314,
3210]
}
df = pd.DataFrame(data)

# Sort by values in ascending order, keeping NaN values at the end
sorted_by_values_asc = df.sort_values(by='S', na_position='last')
print("Sorted by values (Ascending Order):")
```

```python
print(sorted_by_values_asc)

# Sort by values in descending order, keeping NaN values at the end
sorted_by_values_desc = df.sort_values(by='S', ascending=False,
na_position='last')
print("\nSorted by values (Descending Order):")
print(sorted_by_values_desc)

# Sort by index in ascending order
sorted_by_index_asc = df.sort_index()
print("\nSorted by index (Ascending Order):")
print(sorted_by_index_asc)

# Sort by index in descending order
sorted_by_index_desc = df.sort_index(ascending=False)
print("\nSorted by index (Descending Order):")
print(sorted_by_index_desc)
```

```
Sorted by values (Ascending Order):
        S
6  2155.0
0  2223.0
9  3210.0
8  3314.0
3  3411.0
1  3445.0
4  6223.0
5  8334.0
2     NaN
7     NaN
Sorted by values (Descending Order):
        S
5  8334.0
4  6223.0
1  3445.0
3  3411.0
8  3314.0
9  3210.0
0  2223.0
6  2155.0
2     NaN
7     NaN
Sorted by index (Ascending Order):
        S
0     2223.0
1     3445.0
2     NaN
3     3411.0
```

```
4     6223.0
5     8334.0
6     2155.0
7     NaN
8     3314.0
9     3210.0
Sorted    by
index
(Descending
Order):
        S
9  3210.0
8  3314.0
7     NaN
6  2155.0
5  8334.0
4  6223.0
3  3411.0
2     NaN
1  3445.0
0  2223.0
import pandas as pd


data = {
    'Name':  ['Alice',  'Bob',  'Charlie',  'David',  'Eve',  'Frank',
'Grace'],
    'VerbalScore': [85, 78, 92, 88, 75, 89, 95],
    'QuantitativeScore': [90, 82, 87, 85, 80, 84, 93],
    'Quality': ['Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No']
}

df = pd.DataFrame(data) df['QualityNumeric'] =


df['Quality'].map({'Yes': 1, 'No': 0})


df['TotalScore'] = df['VerbalScore'] + df['QuantitativeScore']
df['Ranking'] = df['TotalScore'].rank(ascending=False, method='min')


df_sorted = df.sort_values(by='Ranking') print(df_sorted)
      Name  VerbalScore  QuantitativeScore Quality  QualityNumeric  \
6    Grace           95                 93      No               0
2  Charlie           92                 87     Yes               1
0    Alice           85                 90     Yes               1
3    David           88                 85     Yes               1
5    Frank           89                 84     Yes               1
```

```
1       Bob           78                      82       No                0
4       Eve           75                      80       No                0
   TotalScore  Ranking
6         188      1.0
2         179      2.0
0         175      3.0
3         173      4.0
5         173      4.0
1         160      6.0
4         155      7.0
```

1. 1 Arrays from python datastructures

```
# 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]
# 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]]

# 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]
#converting list to set
a=[1,2,3,4,5,5]
c=set(a) np.array(c)
array({1, 2, 3, 4, 5}, dtype=object)
# 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

```
# creating ndarray using arange function
a=np.array(np.arange(9)) print(a)
[0 1 2 3 4 5 6 7 8]
# generates list of specified zeros
a=np.zeros(3) print(a)
[0. 0. 0.]
# generates 2D array of zeros
b=np.zeros([3,3]) print(b)

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

# # generates list of specified ones
a=np.ones(4) print(a)
[1. 1. 1. 1.]
# generates 2D array of ones
b=np.ones([3,3]) print(b)

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

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

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

# shifting diagonal ones one step right
c=np.eye(3,k=1) print(c)

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

# works same as eye() method
a=np.identity(3) print(a)

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

# fills with specified number by specified dimentions
d=np.full((2,2),7) print(d)

[[7 7]
 [7 7]]

# generates zeros of specified dimentions
a=np.empty((2,3)) print(a)

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

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

array([[1, 0, 0, 0],
[0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])

# 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

```
# gives a random number below the number specified
from numpy import random x = random.randint(100)
print(x)
67
# generates 2D array with true and false
a=np.random.choice(['true','false'],size=(2,3))
print(a)
[['false' 'true' 'false']
 ['false' 'true' 'true']]
x = np.random.rand(1) + np.random.rand(1)*1j
print (x) print(x.real) print(x.imag)
[0.66070644+0.13190058j]
[0.66070644]
[0.13190058]
# 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]]

#generates complex numbne
np.random.random(size=(2,2))+1j*np.random.random(size=(2,2))

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

# gives numbers is below the specified number in random order
np.random.permutation(5)
array([3, 0, 4, 1, 2])
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]
```

```
# returns a number between the specified range
np.random.randint(1,5)

4

a=np.random.randn(1,10)
print(a)
[[ 0.14633752  0.0881863  -0.24873423  1.26898256  0.66573405
0.88270367

   0.01506427  2.14571715 -0.52908828  0.17795019]]

a=np.array(['apple','bananaa','cherry'])
b=np.random.choice(a) print(b)
cherry
np.random.shuffle(a)
print(a)

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

2.Manipulation Of Numpy Arrays

2.1 Indexing

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

9

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

2

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

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

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

7

#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

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

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

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

[6 7]
```

```
# 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]
# form starting to index 3
arr=np.array([5,6,7,8,9])
print(arr[:3])
[5 6 7]
# accessing though negetive index
arr=np.array([5,6,7,8,9]) print(arr[-3:-
1])
[7 8]
# start : stop : step
arr=np.array([5,6,7,8,4,5,6,7,9])
print(arr[1:5:2])
[6 8]
arr=np.array([5,6,7,8,4,5,6,7,9]) print(arr[-1:-5:-
1])
[9 7 6 5]
# 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]
# 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]
# 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[0:2, 1:4])

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

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

P Lab
```

2.3 Re-Shaping

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

# 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]]]
```

```
import numpy as np
a=np.arange(8)
print(a.reshape(4,2))
```

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

```
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

```
# concatenating arrays
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]]
```

```
# joining using stack function
print(np.stack((a1,a2),axis=1))
```

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

 [[2 3]
[2 3]]
 [[4 5]

  [4 5]]]
```

```
# 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]]
```

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

# 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

```
import numpy as np
a = np.arange(9)
print(a)
[0 1 2 3 4 5 6 7 8]
# 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])]
# 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]])]
```

```
# 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=1)
print(newarr)
```

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

```
#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

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

# minimum and maximum
print(np.amin(arr), np.amax(arr))
5 9
# range of weight i.e. max weight-min weight
print(np.ptp(arr))

4

# mean
print(np.mean(weight))
55.0
# median
print(np.median(weight))
55.0
# standard deviation
print(np.std(weight))
28.401877872187722
# variance
print(np.var(weight))
806.6666666666666
# average
print(np.average(weight))

55.0
```

3.2 Bit-twiddling functions

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

# bitwise_and
print(np.bitwise_and(even, odd))

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

```
# bitwise_or
print(np.bitwise_or(even, odd))
[ 1  3  5  7  9 17 33]
# bitwise_xor
print(np.bitwise_xor(even, odd))
[1 1 1 1 1 1 1]
# invert or not
print(np.invert(even))
[ -1  -3  -5  -7  -9 -17 -33]
# left_shift
print(np.left_shift(even, 1))
[ 0  4  8 12 16 32 64]
# right_shift
print(np.right_shift(even, 1))

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

3.3 Unary Universal Functions

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

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

# square root of given list of elements
np.sqrt(arr)

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

# give exponential of all elements in the input array
np.exp(arr)

array([[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],
       [2.20264658e+04, 5.98741417e+04, 1.62754791e+05],
       [4.42413392e+05, 1.20260428e+06, 3.26901737e+06],
       [8.88611052e+06, 2.41549528e+07, 6.56599691e+07]])

# min value
np.min(arr)

0
```

```
# max element
np.max(arr)

9

# average of all elements
np.average(arr)
4.5
# absolute values of all elements
print(np.abs(arr))
[0 1 2 3 4 5 6 7 8 9]
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

```
x = np.random.randn(8)
y = np.random.randn(8)

# gives random specified number of values
print(x)
[ 1.9029113   0.37516745 -1.30605534  0.40233125 -0.52921987
0.50879897

 -0.14657609  0.76597139]

print(y)
[-0.08739821  0.55924299 -0.60581813  1.59797719  0.12302027 -
0.37407141

 -0.84599114  0.47792473]

np.maximum(x, y)
array([-0.08739821,  0.55924299,  1.08833746,  1.59797719,
0.16382473,

       -0.36369696, -0.84599114,  0.50381589])

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]
```

```
print(whole_part)
[ 4.  3. -0. -1.  0.  9.  9.]
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]]

np.subtract(a,b)

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

np.divide(a,b)

array([[0. , 0.1, 0.2],
       [0.3, 0.4, 0.5],
       [0.6, 0.7, 0.8]])

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

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

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

array([[3, 7, 5],
       [8, 4, 3],
       [2, 4, 9]])

# gives sum of all elements
a.sum()
```

```
45
# 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)
82.0
# gives mean
arr.mean()
0.053930671819051576
# mean based on axis
arr.mean(axis=1)
array([1., 4., 7.])
# gives median
np.median(arr)
4.0
# standerd deviation
np.std(arr)
0.8542443496205637
# varience
np.var(arr)
6.666666666666667
# sum for given axis
arr.sum(axis=0)
array([-3.35777149,  0.03835636,  1.12685565,  0.50732766])
arr = np.array([0, 1, 2, 3, 4, 5, 6, 7])
print(arr.cumsum())
[ 0  1  3  6 10 15 21 28]
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]]

print(arr.cumprod(axis=1))
```

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

4.2 Comparison Operations

```
# It results either true or flase based on the specified condition

a=np.array([[1,2],[3,4]])
b=np.array([[1,2],[3,4]])
print(np.array_equal(a,b))
True
a=np.array([1,15,6,8])
b=np.array([11,12,6,4])
print(np.greater(a,b))
[False  True False  True]
print(np.greater_equal(a,b))
[False  True  True  True]
print(np.less(a[0],b[2]))
True
print(np.less(a,b))
[ True False False False]
print(np.less_equal(a,b))
[ True False  True False]
```

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

5.1 Sorting

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

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

```
# gives sorted list
np.sort(a)

array([[3, 7],
[1, 9]]) # sort based on
the axis
np.sort(a,axis=0)

array([[3, 1],
[9, 7]])
np.sort(a,axis=1)

array([[3, 7],
[1, 9]])
a.sort(1)
print(a)

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

5.2 Unique Operation

```
# returns unique elements
names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])
print(np.unique(names))
['Bob' 'Joe' 'Will']
# Contrast np.unique with the pure Python alternative:
sorted(set(names))
['Bob', 'Joe', 'Will']
# 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

```
# set will not allows duplicate elements

# 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]
# returns union of two sets
arr1=np.array([1,2,3,4])
arr2=np.array([3,4,5,6])

# perform union on arr1 and arr2
print(np.union1d(arr1,arr2))
[1 2 3 4 5 6]
#perform intersection on two arrays
print(np.intersect1d(arr1,arr2))
[3 4]
#find set difference
print(np.setdiff1d(arr1,arr2))
[1 2]
#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

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image #read
image (set image as m) img =
Image.open('hello.png')
imgarr=np.array(img)

#displaying image
plt.imshow(imgarr)
plt.title('original image')
plt.show()
```

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

```python
# flipped by 180 degrees
flipimg=np.flipud(imgarr)
plt.imshow(flipimg)
plt.title('flipped image')
plt.show()
```

Data Manipulation with Pandas

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

Pandas:

Pandas: Powerful Data Analysis and Manipulation

Pandas is a popular open-source library for data manipulation and analysis in Python. It provides data structures and functions to efficiently handle structured data, including tabular data such as spreadsheets and SQL tables.

Key Features:

1. DataFrames: Two-dimensional labeled data structure with columns of potentially different types.
2. Series: One-dimensional labeled array of values.
3. Data Manipulation: Filter, sort, group, merge, and join data.
4. Data Analysis: Perform statistical analysis, data cleaning, and visualization.
5. Input/Output: Read and write data from various formats (CSV, Excel, JSON, SQL).

Advantages:

1. Efficient: Optimized for performance, handling large datasets.
2. Flexible: Handles missing data, data merging, and data reshaping.
3. Intuitive: Simple and consistent API for data manipulation.
4. Integration: Seamlessly integrates with other popular libraries (NumPy, Matplotlib, Scikit-learn).

Common Use Cases:

1. Data Cleaning: Handle missing values, data normalization, and data transformation.
2. Data Analysis: Perform statistical analysis, data visualization, and data mining.
3. Data Science: Build machine learning models, predict outcomes, and evaluate performance.
4. Business Intelligence: Analyze and visualize business data for informed decision-making.

```
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
# 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
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

```
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
# 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
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

```
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
# 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
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

```
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

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

print(sr['Day 1'])
1/1/2018
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

```
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
s['b']
1.0
s[['b','a','d']]
b    1.0
a    0.0
d    3.0
dtype: float64
s['b':'e']
b    1.0
c    2.0
d    3.0
e    4.0
dtype: float64
s[1]
1.0
s[2:4]
c    2.0
d    3.0
dtype: float64
s[[1,3]]
```

```
b     1.0
d     3.0
dtype: float64
print(s[[0, 2, 4]])
a     0.0
c     2.0
e     4.0
dtype: float64
```

2.3 Filtering

```
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
s[s<2]
a     0.0

dtype: float64
s[s>2]
b     5.0
d     3.0
e     4.0
dtype: float64
s[s!=2]
a     0.0
b     5.0
d     3.0
e     4.0
dtype: float64
s[(s>2)&(s<5) ]
d     3.0
e     4.0
dtype: float64
s['b':'c']
```

```
b    5.0
c    2.0
dtype: float64
print(s[1:2]==5)

b    True dtype:
bool
s[s.isin([2,4])]
c    2.0
e    4.0
dtype: float64
```

2.4 Arithmetic Operations

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

series3 = series1 + series2
print(series3)
0      7
1      9
2      11
3      13
4      15
dtype: int64
series3 = series1 - series2
print(series3)
0   -5
1   -5
2   -5
3   -5
4   -5
dtype: int64
series3 = series1 *series2
print(series3)
0      6
1      14
2      24
3      36
4      50
dtype: int64
```

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

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

series3 = series1 %series2
print(series3)
0     1
1     2
2     3
3     4
4     5
dtype: int64
```

2.5 Ranking

```
import pandas as pd
s=pd.Series([121,211,153,214,115,116,237,118,219,120])
s.rank(ascending=True)
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
s.rank(ascending=False)
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
s.rank(method='min')
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
s.rank(method='max')
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
s.rank(method='first')
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

```
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
sr.sort_values(ascending = False)
2    22.7800
3    20.1240
0    19.5000
4    18.1002
1    16.8000
dtype: float64
sr.sort_values(ascending = True)
1    16.8000
4    18.1002
0    19.5000
3    20.1240
2    22.7800
dtype: float64
sr.sort_index()
0    19.5000
1    16.8000
2    22.7800
3    20.1240
4    18.1002
dtype: float64
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

```
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

x.isnull()
california     True
ohio          False
Texas          True
oregon        False
dtype: bool

x.notnull()
california    False
ohio           True
Texas         False
oregon         True
dtype: bool
```

2.8 Concatenation

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

# concatenating
display(pd.concat([series1, series2]))
0    1
1    2
2    3
0    A
1    B
2    C
dtype: object
display(pd.concat([series1, series2],
axis = 1))
   0  1
0  1  A
1  2  B
2  3  C
```

```
display(pd.concat([series1, series2],
axis = 0))
0     1
1     2
2     3
0     A
1     B
2     C
dtype: object
print(pd.concat([series1, series2], ignore_index=True))
0     1
1     2
2     3
3     A
4     B
5     C
dtype: object
print(pd.concat([series1, series2], ignore_index=False))
0     1
1     2
2     3
0     A
1     B
2     C
dtype: object
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

```
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
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
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

```
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
df=pd.DataFrame({'state':['AP','AP','AP','TS','TS','TS'],'year':
[2000,2001,2002,2000,2001,2002],'pop':[1.5,1.7,3.6,2.4,2.9,3.2]})
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
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
d 1   4  7
2  5  8 e
  2  6  9
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'])
    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

```
import pandas as pd
import seaborn as sns

data=sns.get_dataset_names()
data


['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'dowjones',
```

```
 'exercise',
 'flights',
 'fmri',
 'geyser',
 'glue',
 'healthexp',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'seaice',
 'taxis',
 'tips',
 'titanic']
```

4.2 display top and bottom five rows

```
import seaborn as sns
data=sns.load_dataset('exercise')
print(data.head())
   Unnamed: 0  id     diet  pulse    time  kind
0           0   1  low fat     85   1 min  rest
1           1   1  low fat     85  15 min  rest
2           2   1  low fat     88  30 min  rest
3           3   2  low fat     90   1 min  rest
4           4   2  low fat     92  15 min  rest
data.tail(5)
```


```
{"summary":"{\n  \"name\": \"data\",\n  \"rows\": 5,\n  \"fields\": [\
n    {\n      \"column\": \"Unnamed: 0\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 85,\n
\"max\": 89,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          86,\n          89,\n          87\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\ n
},\n    {\n      \"column\": \"id\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 29,\n
\"max\": 30,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          30,\n          29\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"diet\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          \"no fat\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"pulse\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 20,\n        \"min\": 99,\n
\"max\": 150,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          130\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"time\",\n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"15 min\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\":
\"kind\",\n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 1,\n        \"samples\": [\n
\"running\"\n             ],\n             \"semantic_type\":  \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

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

```
data.shape
(90, 6)
data.dtypes
Unnamed: 0        int64
id                int64
diet           category
pulse             int64
time           category
kind           category
dtype: object

data.isnull().sum()
Unnamed: 0    0
id            0
diet          0
pulse         0
time          0
kind          0
dtype: int64

data.columns
Index(['Unnamed: 0', 'id', 'diet', 'pulse', 'time', 'kind'],
dtype='object')

data.index

RangeIndex(start=0, stop=90, step=1)
```

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

```
data.loc[data['pulse']>120]
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 10,\n \"fields\": [\n   {\n   \"column\": \"Unnamed: 0\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 5,\n   \"min\": 70,\n   \"max\": 89,\n   \"num_unique_values\": 10,\n   \"samples\": [\n   86,\n   76,\n   82\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"id\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1,\n   \"min\": 24,\n   \"max\": 30,\n   \"num_unique_values\": 6,\n   \"samples\": [\n   24,\n   26,\n   30\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"diet\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"no fat\",\n   \"low fat\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"pulse\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 8,\n   \"min\": 124,\n   \"max\": 150,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   126,\n   135\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"time\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"30 min\",\n   \"15 min\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"kind\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 1,\n   \"samples\": [\n   \"running\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

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

{"summary":"{\n \"name\": \"data\",\n \"rows\": 88,\n \"fields\": [\n   {\n   \"column\": \"Unnamed: 0\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 25,\n   \"min\": 1,\n   \"max\": 89,\n   \"num_unique_values\": 88,\n   \"samples\": [\n   78,\n   1,\n   28\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"id\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 8,\n   \"min\": 1,\n   \"max\": 30,\n   \"num_unique_values\": 30,\n   \"samples\": [\n   28,\n   16,\n   24\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"diet\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"no fat\",\n   \"low fat\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"pulse\",\n   \"properties\": {\n   \"dtype\": \"number\",\n

\"std\": 14,\n   \"min\": 80,\n   \"max\": 150,\n
\"num_unique_values\": 39,\n   \"samples\": [\n   140,\n
130\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"time\",\n   \"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n   \"samples\": [\n   \"15
min\",\n   \"30 min\"\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"kind\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 3,\n   \"samples\":
[\n   \"rest\",\n   \"walking\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n   ]\n}","type":"dataframe"}

```
data.drop(data[data['pulse']>100].index)
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 63,\n \"fields\":
[\n   {\n   \"column\": \"Unnamed: 0\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 22,\n   \"min\": 0,\n
\"max\": 87,\n   \"num_unique_values\": 63,\n   \"samples\":
[\n   84,\n   69,\n   0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\ n
},\n   {\n   \"column\": \"id\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 7,\n   \"min\": 1,\n
\"max\": 30,\n   \"num_unique_values\": 28,\n   \"samples\":
[\n   10,\n   27,\n   9\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\ n
},\n   {\n   \"column\": \"diet\",\n   \"properties\": {\n
\"dtype\": \"category\",\n   \"num_unique_values\": 2,\n
\"samples\": [\n   \"no fat\",\n   \"low fat\"\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"pulse\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 5,\n   \"min\": 80,\n   \"max\": 100,\n
\"num_unique_values\": 20,\n   \"samples\": [\n   85,\n
86\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"time\",\n   \"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n   \"samples\": [\n   \"1
min\",\n   \"15 min\"\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"kind\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 3,\n   \"samples\":
[\n   \"rest\",\n   \"walking\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

```
data.loc[6,'id']
```

3

```
data.loc[11:15][['id','pulse']]
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 5,\n \"fields\": [\n   {\n   \"column\": \"id\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 4,\n   \"max\": 6,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   4,\n   5,\n   6\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"pulse\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 4,\n   \"min\": 83,\n   \"max\": 92,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   83,\n   91,\n   92\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

4.5 Sorting and Ranking operations in DataFrame

```
data
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 90,\n \"fields\": [\n   {\n   \"column\": \"Unnamed: 0\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 26,\n   \"min\": 0,\n   \"max\": 89,\n   \"num_unique_values\": 90,\n   \"samples\": [\n   40,\n   22,\n   55\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"id\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 8,\n   \"min\": 1,\n   \"max\": 30,\n   \"num_unique_values\": 30,\n   \"samples\": [\n   28,\n   16,\n   24\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"diet\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"no fat\",\n   \"low fat\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"pulse\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 14,\n   \"min\": 80,\n   \"max\": 150,\n   \"num_unique_values\": 39,\n   \"samples\": [\n   140,\n   130\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"time\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"1 min\",\n   \"15 min\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"kind\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"rest\",\n   \"walking\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"data"}

```
data.sort_index(ascending=False)
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 90,\n \"fields\": [\n   {\n   \"column\": \"Unnamed: 0\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 26,\n   \"min\": 0,\n   \"max\": 89,\n   \"num_unique_values\": 90,\n   \"samples\": [\n   49,\n   67,\n   34\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"id\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 8,\n   \"min\": 1,\n   \"max\": 30,\n   \"num_unique_values\": 30,\n   \"samples\": [\n   3,\n   15,\n   7\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"diet\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"low fat\",\n   \"no fat\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"pulse\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 14,\n   \"min\": 80,\n   \"max\": 150,\n   \"num_unique_values\": 39,\n   \"samples\": [\n   88,\n   82\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"time\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"30 min\",\n   \"15 min\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"kind\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"running\",\n   \"walking\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
data.sort_values(['pulse']).head(6)
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 6,\n \"fields\": [\n   {\n   \"column\": \"Unnamed: 0\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 13,\n   \"min\": 9,\n   \"max\": 45,\n   \"num_unique_values\": 6,\n   \"samples\": [\n   9,\n   10,\n   45\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"id\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 4,\n   \"min\": 4,\n   \"max\": 16,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   4,\n   6,\n   16\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"diet\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"no fat\",\n   \"low fat\"\n   ],\n   \"semantic_type\": \"\",\n

\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"pulse\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 1,\n   \"min\": 80,\n   \"max\": 84,\n
\"num_unique_values\": 4,\n   \"samples\": [\n   82,\n
84\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"time\",\n   \"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n   \"samples\": [\n   \"1 min\",\n
\"15 min\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"kind\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   \"walking\",\n   \"rest\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n   ]\n}","type":"dataframe"}

```
data.sort_values(by=['pulse','time']).head(6)
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 6,\n \"fields\": [\
n   {\n   \"column\": \"Unnamed: 0\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 13,\n   \"min\": 9,\n
\"max\": 45,\n   \"num_unique_values\": 6,\n   \"samples\":
[\n   9,\n   10,\n   45\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\ n
},\n   {\n   \"column\": \"id\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 4,\n   \"min\": 4,\n
\"max\": 16,\n   \"num_unique_values\": 3,\n   \"samples\":
[\n   4,\n   6,\n   16\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\ n
},\n   {\n   \"column\": \"diet\",\n   \"properties\": {\n
\"dtype\": \"category\",\n   \"num_unique_values\": 2,\n
\"samples\": [\n   \"no fat\",\n   \"low fat\"\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"pulse\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 1,\n   \"min\": 80,\n   \"max\": 84,\n
\"num_unique_values\": 4,\n   \"samples\": [\n   82,\n
84\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"time\",\n   \"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n   \"samples\": [\n   \"1 min\",\n
\"15 min\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"kind\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 2,\n   \"samples\":
[\n   \"walking\",\n   \"rest\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe"}

```
data.rank().head(10)
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 10,\n \"fields\":
[\n {\n \"column\": \"Unnamed: 0\",\n \"properties\": {\n
\"dtype\": \"number\",\n \"std\": 3.0276503540974917,\n
\"min\": 1.0,\n \"max\": 10.0,\n \"num_unique_values\":
10,\n \"samples\": [\n 9.0,\n 2.0,\n
6.0\n],\n \"semantic_type\": \"\",\n
\"description\": \"\"\n }\n },\n {\n \"column\":
\"id\",\n \"properties\": {\n \"dtype\": \"number\",\n
\"std\": 3.0983866769659336,\n \"min\": 2.0,\n \"max\":
11.0,\n \"num_unique_values\": 4,\n \"samples\": [\n
5.0,\n 11.0,\n 2.0\n],\n \"semantic_type\":
\"\",\n \"description\": \"\"\n }\ n },\n {\n
\"column\": \"diet\",\n \"properties\": {\n \"dtype\":
\"number\",\n \"std\": 0.0,\n \"min\": 23.0,\ n
\"max\": 23.0,\n \"num_unique_values\": 1,\n \"samples\":
[\n 23.0\n],\n \"semantic_type\":
\"\",\n \"description\": \"\"\n }\n },\n {\n
\"column\": \"pulse\",\n \"properties\": {\n \"dtype\":
\"number\",\n \"std\": 16.63196387148019,\n \"min\":
1.0,\n \"max\": 49.0,\n \"num_unique_values\": 8,\n
\"samples\": [\n 16.5\n],\n \"semantic_type\":
\"\",\n \"description\": \"\"\n }\n },\n {\n
\"column\": \"time\",\n \"properties\": {\n \"dtype\":
\"number\",\n \"std\": 26.267851073127396,\n \"min\":
15.5,\n \"max\": 75.5,\n \"num_unique_values\": 3,\n
\"samples\": [\n 15.5\n],\n \"semantic_type\":
\"\",\n \"description\": \"\"\n }\n },\n {\n
\"column\": \"kind\",\n \"properties\": {\n \"dtype\":
\"number\",\n \"std\": 0.0,\n \"min\": 15.5,\n
\"max\": 15.5,\n \"num_unique_values\": 1,\n
\"samples\": [\n 15.5\n],\n \"semantic_type\":
\"\",\n \"description\": \"\"\n }\n }\n]\
n}","type":"dataframe"}

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

{"summary":"{\n \"name\": \"data\",\n \"rows\": 2,\n \"fields\": [\
n {\n \"column\": \"Unnamed: 0\",\n \"properties\": {\n
\"dtype\": \"number\",\n \"std\": 0.7071067811865476,\n
\"min\": 1.0,\n \"max\": 2.0,\n \"num_unique_values\":
2,\n \"samples\": [\n 2.0,\n 1.0\n],\
n \"semantic_type\": \"\",\n \"description\": \"\"\n
}\n },\n {\n \"column\": \"id\",\n \"properties\": {\n
\"dtype\": \"number\",\n \"std\": 0.0,\n \"min\": 2.0,\n
\"max\": 2.0,\n \"num_unique_values\": 1,\n \"samples\":
[\n 2.0\n],\n \"semantic_type\": \"\",\n
\"description\": \"\"\n }\n },\n {\n \"column\":
\"diet\",\n \"properties\": {\n \"dtype\": \"number\",\n
\"std\": 0.0,\n \"min\": 23.0,\n \"max\": 23.0,\n
\"num_unique_values\": 1,\n \"samples\": [\n 23.0\n

],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"pulse\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 9.5,\n   \"max\": 9.5,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   9.5\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"time\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 21.213203435596427,\n   \"min\": 15.5,\n   \"max\": 45.5,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   45.5\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"kind\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 15.5,\n   \"max\": 15.5,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   15.5\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

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

{"summary":"{\n \"name\": \"data\",\n \"rows\": 5,\n \"fields\": [\n   {\n   \"column\": \"Unnamed: 0\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1.5811388300841898,\n   \"min\": 86.0,\n   \"max\": 90.0,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   89.0,\n   86.0,\n   88.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"id\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1.6431676725154984,\n   \"min\": 86.0,\n   \"max\": 89.0,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   86.0,\n   89.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"diet\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 68.0,\n   \"max\": 68.0,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   68.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"pulse\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 7.968688725254614,\n   \"min\": 63.0,\n   \"max\": 81.5,\n   \"num_unique_values\": 4,\n   \"samples\": [\n   74.5\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"time\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 25.099800796022265,\n   \"min\": 15.5,\n   \"max\": 75.5,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   75.5\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"kind\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 75.5,\n   \"max\": 75.5,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   75.5\n   ],

```
\"semantic_type\": \"\",\n          \"description\": \"\"\n        }\ n
}\n  ]\n}","type":"dataframe"}
```


```
data['time'].rank().head(5)
0    15.5
1    45.5
2    75.5
3    15.5
4    45.5
Name: time, dtype: float64
```

4.6 Statistical Operations

```
data=sns.load_dataset('mpg')
data
```


```
{"summary":"{\n  \"name\": \"data\",\n  \"rows\": 398,\n  \"fields\":
[\n    {\n      \"column\": \"mpg\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 7.815984312565782,\n
\"min\": 9.0,\n        \"max\": 46.6,\n        \"num_unique_values\":
129,\n        \"samples\": [\n          17.7,\n          30.5,\n
30.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cylinders\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1,\n        \"min\": 3,\n
\"max\": 8,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          4,\n          5,\n          6\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"displacement\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
104.26983817119591,\n        \"min\": 68.0,\n        \"max\": 455.0,\n
\"num_unique_values\": 82,\n        \"samples\": [\n          122.0,\n
307.0,\n          360.0\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"horsepower\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 38.49115993282849,\n
\"min\": 46.0,\n        \"max\": 230.0,\n
\"num_unique_values\": 93,\n        \"samples\": [\n          92.0,\n
100.0,\n          52.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"weight\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\":  846,\n          \"min\":  1613,\n          \"max\":  5140,\n
\"num_unique_values\": 351,\n        \"samples\": [\n          3730,\n
1995,\n          2215\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"acceleration\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2.757688929812676,\n        \"min\":
8.0,\n        \"max\": 24.8,\n        \"num_unique_values\": 95,\n
\"samples\": [\n          14.7,\n          18.0,\n          14.3\n
```

],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"model_year\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 3,\n   \"min\": 70,\n   \"max\": 82,\n   \"num_unique_values\": 13,\n   \"samples\": [\n   81,\n   79,\n   70\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"origin\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"usa\",\n   \"japan\",\n   \"europe\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"name\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 305,\n   \"samples\": [\n   \"mazda rx-4\",\n   \"ford f108\",\n   \"buick century luxus (sw)\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"data"}

```
data=data.drop(columns=['name','origin'])
data
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 398,\n \"fields\": [\n   {\n   \"column\": \"mpg\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 7.815984312565782,\n   \"min\": 9.0,\n   \"max\": 46.6,\n   \"num_unique_values\": 129,\n   \"samples\": [\n   17.7,\n   30.5,\n   30.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"cylinders\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1,\n   \"min\": 3,\n   \"max\": 8,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   4,\n   5,\n   6\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"displacement\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 104.26983817119591,\n   \"min\": 68.0,\n   \"max\": 455.0,\n   \"num_unique_values\": 82,\n   \"samples\": [\n   122.0,\n   307.0,\n   360.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"horsepower\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 38.49115993282849,\n   \"min\": 46.0,\n   \"max\": 230.0,\n   \"num_unique_values\": 93,\n   \"samples\": [\n   92.0,\n   100.0,\n   52.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"weight\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 846,\n   \"min\": 1613,\n   \"max\": 5140,\n   \"num_unique_values\": 351,\n   \"samples\": [\n   3730,\n   1995,\n   2215\n   ],\n   \"semantic_type\": \"\",\n

```
\"description\": \"\"\n      }\n   },\n    {\n      \"column\":
```

\"acceleration\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 2.757688929812676,\n   \"min\":
8.0,\n   \"max\": 24.8,\n   \"num_unique_values\": 95,\n
\"samples\": [\n   14.7,\n   18.0,\n   14.3\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"model_year\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
3,\n   \"min\": 70,\n   \"max\": 82,\n
\"num_unique_values\": 13,\n   \"samples\": [\n   81,\n
79,\n   70\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"data"}

```
data.mean()
```

```
mpg               23.514573
cylinders          5.454774
displacement     193.425879
horsepower       104.469388
weight          2970.424623
acceleration      15.568090
model_year        76.010050
dtype: float64
```

```
data.mean()[['displacement','horsepower']]
```

```
displacement    193.425879
horsepower      104.469388
dtype: float64
```

```
data.mode()
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 2,\n \"fields\": [\
n   {\n   \"column\": \"mpg\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": null,\n   \"min\":
13.0,\n   \"max\": 13.0,\n   \"num_unique_values\": 1,\n
\"samples\": [\n   13.0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"cylinders\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": null,\n   \"min\": 4.0,\
n   \"max\": 4.0,\n   \"num_unique_values\": 1,\n
\"samples\": [\n   4.0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"displacement\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": null,\n   \"min\":
97.0,\n   \"max\": 97.0,\n   \"num_unique_values\": 1,\n
\"samples\": [\n   97.0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"horsepower\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": null,\n   \"min\":
150.0,\n   \"max\": 150.0,\n   \"num_unique_values\": 1,\n

\"samples\": [\n   150.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\ n
},\n   {\n   \"column\": \"weight\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\": 102,\n
\"min\": 1985,\n   \"max\": 2130,\n   \"num_unique_values\":
2,\n   \"samples\": [\n   2130\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"acceleration\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
null,\n   \"min\": 14.5,\n   \"max\": 14.5,\n
\"num_unique_values\": 1,\n   \"samples\": [\n   14.5\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"model_year\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
null,\n   \"min\": 73.0,\n   \"max\": 73.0,\n
\"num_unique_values\": 1,\n   \"samples\": [\n   73.0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe"}

```
data.median()
mpg               23.0
cylinders          4.0
displacement     148.5
horsepower        93.5
weight          2803.5
acceleration      15.5
model_year        76.0
dtype: float64
```

```
data.std()
mpg               7.815984
cylinders         1.701004
displacement    104.269838
horsepower       38.491160
weight          846.841774
acceleration      2.757689
model_year        3.697627
dtype: float64
```

```
data.var()
mpg                 61.089611
cylinders            2.893415
displacement     10872.199152
horsepower        1481.569393
weight          717140.990526
acceleration         7.604848
model_year          13.672443
dtype: float64
```

```
data.sum()
mpg                9358.8
cylinders          2171.0
displacement      76983.5
horsepower        40952.0
weight          1182229.0
acceleration       6196.1
model_year        30252.0
dtype: float64

data.corr()
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 7,\n \"fields\": [\n   {\n   \"column\": \"mpg\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.8016175656768498,\n   \"min\": -0.8317409332443344,\n   \"max\": 1.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n   1.0,\n   -0.7753962854205539,\n   0.42028891210165054\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\ n   },\n   {\n   \"column\": \"cylinders\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.7947040706832936,\n   \"min\": -0.7753962854205539,\n   \"max\": 1.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n   -0.7753962854205539,\n   1.0,\n   -0.5054194890521758\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"displacement\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.8216456211919416,\n   \"min\": 0.8042028248058979,\n   \"max\": 1.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n   -0.8042028248058979,\n   0.9507213901392415,\n   -0.5436840835009299\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"horsepower\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.8260743651440925,\n   \"min\": 0.7784267838977761,\n   \"max\": 1.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n   -0.7784267838977761,\n   0.8429833569186568,\n   -0.6891955103342376\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"weight\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.7882159155698183,\n   \"min\": -0.8317409332443344,\n   \"max\": 1.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n   -0.8317409332443344,\n   0.8960167954533944,\n   -0.4174573199403932\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"acceleration\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.6365769931677212,\n   \"min\": 0.6891955103342376,\n   \"max\": 1.0,\n

\"num_unique_values\": 7,\n   \"samples\": [\n   0.42028891210165054,\n   -0.5054194890521758,\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"model_year\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.5654915849248219,\n   \"min\": -0.41636147709998894,\n   \"max\": 1.0,\n   \"num_unique_values\": 7,\n   \"samples\": [\n   0.5792671330833092,\n   -0.34874579661359445,\n   0.28813695429949115\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
data.var()['horsepower']
1481.5693929745814
data.mean()[['displacement','horsepower']]
displacement    193.425879
horsepower      104.469388
dtype: float64
```

4.7 count and Uniqueness of given Categorical values

```
data.count()
mpg             398
cylinders       398
displacement    398
horsepower      392
weight          398
acceleration    398
model_year      398
dtype: int64
data.value_counts()
```

```
npg   cylinders  displacement  horsepower  weight  acceleration     1
model_year
                                                                    1
9.0   8          304.0         193.0       4732    18.5          70
                                                                    1
27.0  4          151.0         90.0        2950    17.3          82
                                                                    1
                 140.0         86.0        2790    15.6          82
                                                                    1
                 112.0         88.0        2640    18.6          82

                 101.0         83.0        2202    15.3          76

..
18.5  6          250.0         110.0       3645    16.2          76
```

```
                                        98.0         3525    19.0           77

18.2   8          318.0         135.0        3830    15.2           79

18.1   8          302.0         139.0        3205    11.2           78

46.6   4          86.0          65.0         2110    17.9           80

Name: count, Length: 392, dtype: int64
```

```
1

1

1

1

1
```

```
data.value_counts(data['horsepower'])
```

```
horsepower
150.0    22
90.0     20
88.0     19
110.0    18
100.0    17
..
132.0     1
133.0     1
135.0     1
137.0     1
230.0     1
Name: count, Length: 93, dtype: int64
data['displacement'].unique()
```

```
array([307. , 350. , 318. , 304. , 302. , 429. , 454. , 440. , 455. ,
       390. , 383. , 340. , 400. , 113. , 198. , 199. , 200. ,  97. ,
       110. , 107. , 104. , 121. , 360. , 140. ,  98. , 232. , 225. ,
       250. , 351. , 258. , 122. , 116. ,  79. ,  88. ,  71. ,  72. ,
        91. ,  97.5,  70. , 120. ,  96. , 108. , 155. ,  68. , 114. ,
       156. ,  76. ,  83. ,  90. , 231. , 262. , 134. , 119. , 171. ,
       115. , 101. , 305. ,  85. , 130. , 168. , 111. , 260. , 151. ,
       146. ,  80. ,  78. , 105. , 131. , 163. ,  89. , 267. ,  86. ,
       183. , 141. , 173. , 135. ,  81. , 100. , 145. , 112. , 181. ,
       144. ])
```

```
categorical_columns = ['horsepower'] 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: horsepower

Unique Values: [130. 165. 150. 140. 198. 220. 215. 225. 190. 170. 160.
```

```
95.  97.  85.
  88.  46.  87.  90. 113. 200. 210. 193.  nan 100. 105. 175. 153. 180.
 110.  72.  86.  70.  76.  65.  69.  60.  80.  54. 208. 155. 112.  92.
 145. 137. 158. 167.  94. 107. 230.  49.  75.  91. 122.  67.  83.  78.
  52.  61.  93. 148. 129.  96.  71.  98. 115.  53.  81.  79. 120. 152.
 102. 108.  68.  58. 149.  89.  63.  48.  66. 139. 103. 125. 133. 138.
 135. 142.  77.  62. 132.  84.  64.  74. 116.  82.]
Value Counts:
horsepower
150.0    22
90.0     20
88.0     19
110.0    18
100.0    17
..
61.0      1
93.0      1
148.0     1
152.0     1
82.0      1

Name: count, Length: 93, dtype: int64
```

```
data=data.rename(columns={'displacement':'min_dist'}).head(5)
data
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 5,\n \"fields\": [\n   {\n   \"column\": \"mpg\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1.3038404810405297,\n   \"min\": 15.0,\n   \"max\": 18.0,\n   \"num_unique_values\": 4,\n   \"samples\": [\n   15.0,\n   17.0,\n   18.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"cylinders\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 8,\n   \"max\": 8,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   8\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"min_dist\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 19.879637823662684,\n   \"min\": 302.0,\n   \"max\": 350.0,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   350.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"horsepower\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 13.038404810405298,\n   \"min\": 130.0,\n   \"max\": 165.0,\n   \"num_unique_values\": 4,\n   \"samples\": [\n   165.0\n   ],\n   \"semantic_type\": \"\",\n

\"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"weight\",\n   \"properties\": {\n   \"dtype\": \"number\",\n

\"std\": 110,\n   \"min\": 3433,\n   \"max\": 3693,\n
\"num_unique_values\": 5,\n   \"samples\": [\n   3693\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"acceleration\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0.6519202405202649,\n   \"min\": 10.5,\n   \"max\": 12.0,\n
\"num_unique_values\": 4,\n   \"samples\": [\n   11.5\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"model_year\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0,\n   \"min\": 70,\n   \"max\": 70,\n
\"num_unique_values\": 1,\n   \"samples\": [\n   70\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe","variable_name":"data"}

Data cleaning and preparation

```
import pandas as pd
import seaborn as sns

data=sns.get_dataset_names()
data
```

```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'dowjones',
 'exercise',
 'flights',
 'fmri',
 'geyser',
 'glue',
 'healthexp',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'seaice',
 'taxis',
 'tips',
 'titanic']
```

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

The dataset appears to be a collection of ramen reviews from various brands, with ratings, variety, style, and country information included. The dataset contains ramen reviews with columns for Review #, Brand, Variety, Style, Country, Stars, and Top Ten. The Stars column provides ratings, typically ranging from 0 to 5, which can be analyzed for trends, outliers, and distributions. The Country column indicates the origin of each ramen product, allowing for geographic analysis of ratings. The Style column categorizes ramen into "Cup", "Pack", and "Bowl", useful for comparing ratings by packaging type. The Top Ten column shows whether a ramen was included in a yearly top 10 list, which can be analyzed for brand and country trends. # a)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. Pandas treat None and NaN as essentially interchangeable for indicating missing or null values. To facilitate this convention, there are several useful functions for detecting, removing, and replacing null values in Pandas DataFrame :

isnull() notnull() dropna() fillna() replace() interpolate()

```
import seaborn as sns
df=sns.load_dataset('exercise')
print(df.head())
   Unnamed: 0  id     diet  pulse    time  kind
0           0   1  low fat     85   1 min  rest
1           1   1  low fat     85  15 min  rest
2           2   1  low fat     88  30 min  rest
3           3   2  low fat     90   1 min  rest
4           4   2  low fat     92  15 min  rest
print("Original DataFrame:")
print(df.head())
Original DataFrame:

   Unnamed: 0  id     diet  pulse    time  kind
0           0   1  low fat     85   1 min  rest
1           1   1  low fat     85  15 min  rest
2           2   1  low fat     88  30 min  rest
3           3   2  low fat     90   1 min  rest
4           4   2  low fat     92  15 min  rest
# 1. Detect missing data
missing_data = df.isnull()
print("\nMissing Data:")
print(missing_data.head())
Missing Data:

   Unnamed: 0     id   diet  pulse   time   kind
0       False  False  False  False  False  False
1       False  False  False  False  False  False
2       False  False  False  False  False  False
3       False  False  False  False  False  False
4       False  False  False  False  False  False
# Drop rows with missing values
df_dropna = df.dropna()

# Display the first few rows of the cleaned DataFrame
print("\nDataFrame after dropping rows with missing values:")
print(df_dropna.head())
```

```
DataFrame after dropping rows with missing values:
   Unnamed: 0  id     diet  pulse    time  kind
0           0   1  low fat     85   1 min  rest
1           1   1  low fat     85  15 min  rest
2           2   1  low fat     88  30 min  rest
3           3   2  low fat     90   1 min  rest
4           4   2  low fat     92  15 min  rest
#Handling the missing values
# Drop rows with any missing values
df.dropna(inplace=True) df
    Unnamed: 0  id     diet  pulse    time     kind
0     0   1  low fat     85   1 min     rest
1     1   1  low fat     85  15 min     rest
2     2   1  low fat     88  30 min     rest
3     3   2  low fat     90   1 min     rest
4     4    2   low  fat      92   15  min      rest..
...   ..      ...    ...     ...      ...
85          85  29   no fat    135  15 min  running
86          86  29   no fat    130  30 min  running
87          87  30   no fat     99   1 min  running
88          88  30   no fat    111  15 min  running
89          89  30   no fat    150  30 min  running
[90 rows x 6 columns]
# Count duplicate rows
duplicate_count = df.duplicated().sum()
duplicate_count 0


data = {
    "Row ID": [0, 2575, 2576, 2577, 2578, 2579, 2580],
    "Brand": ["New Touch T's Restaurant", "Vifon", "Wai Wai", "Wei
Lih", "Nissin", "Just Way", "New Touch T's Restaurant"],
    "Variety": ["Tantanmen Cup", "Hu Tiu Nam Vang", "Oriental Style
Instant Noodles", "GGE Ramen Snack Tomato Flavor", "Cup Noodles
Chicken Vegetable", "Noodles Spicy Hot Sesame", "Tantanmen Cup"],
    "Style": ["Japan", "Vietnam", "Thailand", "Taiwan", "USA",
"Taiwan", "Japan"],
    "Country": ["Japan", "Vietnam", "Thailand", "Taiwan", "USA",
"Taiwan", "Japan"],
    "Stars": [3.75, 3.5, 1, 2.75, 2.25, 1, 3.75],
    "Top Ten": [0, 5, 4, 3, 2, 1, 0]
}
```

```
# Create DataFrame df =
pd.DataFrame(data)

# Detect duplicate rows
duplicates = df.duplicated(keep=False)

# Print duplicate rows
print(df[duplicates])

# Drop duplicate rows
cleaned_data = df.drop_duplicates()

# Print cleaned data
print(cleaned_data)


Empty DataFrame
Columns: [Row ID, Brand, Variety, Style, Country, Stars, Top Ten]
Index: []
   Row ID                    Brand                        Variety
Style  \
0    0  New Touch T's Restaurant                  Tantanmen Cup
     Japan
1    2575                    Vifon                 Hu Tiu Nam Vang
Vietnam
2    2576                  Wai Wai  Oriental Style Instant Noodles
Thailand
3    2577                  Wei Lih   GGE Ramen Snack Tomato Flavor
Taiwan
4    2578                   Nissin   Cup Noodles Chicken Vegetable
USA
5    2579                 Just Way        Noodles Spicy Hot Sesame
Taiwan
6    2580  New Touch T's Restaurant                  Tantanmen Cup
Japan

    Country  Stars  Top Ten
0       Japan   3.75        0
1       Vietnam   3.50        5
2       Thailand   1.00        4
3       Taiwan   2.75        3
4       USA   2.25        2
5       Taiwan   1.00        1  6     Japan   3.75        0

df.duplicated().sum()
df

   Row ID                    Brand                        Variety
Style  \
```

```
0    0  New Touch T's Restaurant                   Tantanmen Cup
     Japan
1    2575                    Vifon                 Hu Tiu Nam Vang
Vietnam
2    2576                  Wai Wai  Oriental Style Instant Noodles
Thailand
3    2577                  Wei Lih   GGE Ramen Snack Tomato Flavor
Taiwan
4    2578                   Nissin   Cup Noodles Chicken Vegetable
USA
5    2579                 Just Way        Noodles Spicy Hot Sesame
Taiwan
6    2580  New Touch T's Restaurant                   Tantanmen Cup
Japan

    Country  Stars  Top Ten
0       Japan   3.75        0
1       Vietnam   3.50        5
2       Thailand   1.00        4
3       Taiwan   2.75        3
4       USA   2.25        2
5       Taiwan   1.00        1  6     Japan   3.75        0
```

```python
df=pd.DataFrame(df,columns=['review','Brand','Variety','Style'])
df
```

```
   review                    Brand                         Variety
Style
0     NaN  New Touch T's Restaurant                   Tantanmen Cup
      Japan
1     NaN                    Vifon                 Hu Tiu Nam Vang
Vietnam
2     NaN                  Wai Wai  Oriental Style Instant Noodles
Thailand
3     NaN                  Wei Lih   GGE Ramen Snack Tomato Flavor
Taiwan
4     NaN                   Nissin   Cup Noodles Chicken Vegetable
USA
5     NaN                 Just Way        Noodles Spicy Hot Sesame
Taiwan
6     NaN  New Touch T's Restaurant                   Tantanmen Cup
      Japan df.values

array([[nan, "New Touch T's Restaurant", 'Tantanmen Cup', 'Japan'],
       [nan, 'Vifon', 'Hu Tiu Nam Vang', 'Vietnam'],
       [nan, 'Wai Wai', 'Oriental Style Instant Noodles', 'Thailand'],
       [nan, 'Wei Lih', 'GGE Ramen Snack Tomato Flavor', 'Taiwan'],
       [nan, 'Nissin', 'Cup Noodles Chicken Vegetable', 'USA'],
       [nan, 'Just Way', 'Noodles Spicy Hot Sesame', 'Taiwan'],
```

```
       [nan, "New Touch T's Restaurant", 'Tantanmen Cup', 'Japan']],
dtype=object) df.columns Index(['review', 'Brand', 'Variety',
'Style'], dtype='object')
```

```
df_dropped_rows=df.dropna()
df
```

```
   review                    Brand                         Variety
Style
0     NaN  New Touch T's Restaurant                  Tantanmen Cup
      Japan
1     NaN                    Vifon                Hu Tiu Nam Vang
Vietnam
2     NaN                  Wai Wai  Oriental Style Instant Noodles
Thailand
3     NaN                  Wei Lih   GGE Ramen Snack Tomato Flavor
Taiwan
4     NaN                   Nissin   Cup Noodles Chicken Vegetable
USA
5     NaN                 Just Way        Noodles Spicy Hot Sesame
Taiwan
6     NaN  New Touch T's Restaurant                  Tantanmen Cup
Japan #Fill Missing Values with the Mean of the

Column
```

```
import pandas as pd
df = pd.DataFrame({
    'Brand': ['Nissin', 'Maruchan', 'Nongshim', 'Samyang',
None],'Style': ['Cup', 'Pack', 'Bowl', None, 'Pack'],'Stars': [4.5,
4.0, None, 5.0, None]})
print(df)
```

```
      Brand Style  Stars
0       Nissin   Cup    4.5
1       Maruchan  Pack    4.0
2       Nongshim  Bowl    NaN
3       Samyang  None    5.0
4       None  Pack    NaN
```

```
# Calculate the mean of the 'Stars' column (excluding NaN values)
mean_value = df['Stars'].mean() print(df)
```

```
      Brand Style  Stars
0       Nissin   Cup    4.5
1       Maruchan  Pack    4.0
2       Nongshim  Bowl    NaN
3       Samyang  None    5.0
4       None  Pack    NaN
```

```
# Fill missing values in the 'Stars' column with the calculated mean
df['Stars'].fillna(mean_value, inplace=True)

# Display the DataFrame after filling missing values
print(df)
```

```
      Brand Style  Stars
0       Nissin   Cup    4.5
1       Maruchan  Pack    4.0
2       Nongshim  Bowl    4.5
3       Samyang  None    5.0
4       None  Pack    4.5
```

```
C:\Users\chara\AppData\Local\Temp\ipykernel_10404\638104475.py:2:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  df['Stars'].fillna(mean_value, inplace=True)
```

#b)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.

```
# Create a DataFrame from the provided data
data = {
    'Review #': [2580, 2579, 2578, 2577, 2576, 5, 4, 3, 2, 1],
    'Brand': ['New Touch', 'Just Way', 'Nissin', 'Wei Lih', "Ching's
Secret", 'Vifon', 'Wai Wai', 'Wai Wai', 'Wai Wai', 'Westbrae'],
'Variety': ["T's Restaurant Tantanmen", 'Noodles Spicy Hot
Sesame', 'Cup Noodles Chicken Vegetable', 'GGE Ramen Snack Tomato
```

```
Flavor', 'Singapore Curry', 'Hu Tiu Nam Vang', 'Oriental Style Instant
Noodles', 'Tom Yum Shrimp', 'Tom Yum Chili Flavor', 'Miso Ramen'],
    'Style': ['Cup', 'Pack', 'Cup', 'Pack', 'Pack', 'Bowl', 'Pack',
'Pack', 'Pack', 'Pack'],
    'Country': ['Japan', 'Taiwan', 'USA', 'Taiwan', 'India',
'Vietnam', 'Thailand', 'Thailand', 'Thailand', 'USA'],
    'Stars': [3.75, 1, 2.25, 2.75, 3.75, 3.5, 1, 2, 2, 0.5],
    'Top Ten': [None, None, None, None, None, None, None, None, None,
None]
}
df = pd.DataFrame(data)
df
```

```
   Review #           Brand                        Variety Style
Country  \
0        2580       New Touch        T's Restaurant Tantanmen   Cup
Japan
1        2579        Just Way        Noodles Spicy Hot Sesame  Pack
Taiwan
2        2578          Nissin   Cup Noodles Chicken Vegetable   Cup
         USA
3        2577         Wei Lih   GGE Ramen Snack Tomato Flavor  Pack
Taiwan
4        2576  Ching's Secret                 Singapore Curry  Pack
India
5        5            Vifon                 Hu Tiu Nam Vang  Bowl
Vietnam
6        4          Wai Wai  Oriental Style Instant Noodles  Pack
Thailand
7        3          Wai Wai                  Tom Yum Shrimp  Pack
Thailand
8        2          Wai Wai            Tom Yum Chili Flavor  Pack
Thailand
9        1         Westbrae                      Miso Ramen  Pack
USA

   Stars Top Ten
0   3.75    None
1   1.00    None
2   2.25    None
3   2.75    None
4   3.75    None
5   3.50    None
6   1.00    None
7   2.00    None
8   2.00    None
9   0.50    None
```

```
# Using apply() to classify 'Stars' as 'High' or 'Low'
```

```
df['Rating Category'] = df['Stars'].apply(lambda x: 'High' if x >= 3
else 'Low') df

   Review #           Brand                          Variety Style
Country  \
0        2580       New Touch        T's Restaurant Tantanmen   Cup
Japan
1        2579         Just Way        Noodles Spicy Hot Sesame  Pack
Taiwan
2        2578           Nissin   Cup Noodles Chicken Vegetable   Cup
         USA
3        2577          Wei Lih   GGE Ramen Snack Tomato Flavor  Pack
Taiwan
4        2576  Ching's Secret                 Singapore Curry  Pack
India
5        5            Vifon                Hu Tiu Nam Vang  Bowl
Vietnam
6        4          Wai Wai  Oriental Style Instant Noodles  Pack
Thailand
7        3          Wai Wai                  Tom Yum Shrimp  Pack
Thailand
8        2          Wai Wai            Tom Yum Chili Flavor  Pack
Thailand
9        1         Westbrae                      Miso Ramen  Pack
USA

   Stars Top Ten Rating Category
0   3.75    None            High
1   1.00    None             Low
2   2.25    None             Low
3   2.75    None             Low
4   3.75    None            High
5   3.50    None            High
6   1.00    None             Low
7   2.00    None             Low
8   2.00    None             Low
9   0.50    None             Low

# Using map() to standardize country names
country_map = {
    'USA': 'United States',
    'Japan': 'JP',
    'Taiwan': 'TW',
    'India': 'IN',
    'Vietnam': 'VN',
```

```
    'Thailand': 'TH'
}
df['Country'] = df['Country'].map(country_map)
df
```

```
   Review #           Brand                               Variety Style  \
0         2580         New Touch         T's Restaurant Tantanmen
         Cup
1         2579          Just Way         Noodles Spicy Hot Sesame
         Pack
2         2578             Nissin    Cup Noodles Chicken Vegetable
         Cup
3         2577            Wei Lih    GGE Ramen Snack Tomato Flavor
         Pack
4         2576  Ching's Secret                     Singapore Curry
         Pack
5         5           Vifon                   Hu Tiu Nam Vang   Bowl
6         4         Wai Wai  Oriental Style Instant Noodles  Pack
7         3         Wai Wai                    Tom Yum Shrimp  Pack
8         2         Wai Wai              Tom Yum Chili Flavor  Pack
9         1        Westbrae                        Miso Ramen  Pack
        Country  Stars Top Ten Rating Category
0             JP   3.75    None             High
1             TW   1.00    None              Low
2             United States   2.25    None
             Low
3             TW   2.75    None              Low
4             IN   3.75    None             High
5             VN   3.50    None             High
6             TH   1.00    None              Low
7             TH   2.00    None              Low
8             TH   2.00    None              Low
9             United States   0.50    None
             Low
```

c)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.

Outliers are possible only in continuous values. Thus, the detection and removal of outliers are applicable to regression values only. The outliers in the dataset can be detected by the below methods: Z-score Scatter Plots Interquartile range(IQR)

```
import pandas as pd
import numpy as np
# Select the column to analyze for outliers ('Stars' in this case)
column_name = 'Stars'

# Calculate the Z-scores for the 'Stars' column
z_scores = np.abs((df[column_name] - df[column_name].mean()) /
df[column_name].std())

# Define a threshold for outliers (e.g., Z-score greater than 3)
z_score_threshold = 3

# Filter the DataFrame to keep rows without outliers
filtered_df = df[z_scores <= z_score_threshold]
print(filtered_df.head())
   Review #           Brand                        Variety Style  \
0      2580       New Touch       T's Restaurant Tantanmen   Cup
1      2579        Just Way       Noodles Spicy Hot Sesame  Pack
2      2578          Nissin  Cup Noodles Chicken Vegetable   Cup
3      2577         Wei Lih  GGE Ramen Snack Tomato Flavor  Pack
4      2576  Ching's Secret                Singapore Curry  Pack

        Country  Stars Top Ten Rating Category
0            JP   3.75    None            High
1            TW   1.00    None             Low
2            United States   2.25    None
             Low
3            TW   2.75    None             Low
4            IN   3.75    None            High
```

d) perform vectorized string operations on pandas series

The 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) realworld 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.

```
import pandas as pd
data = {'Brand': ['New Touch','Just Way', 'Wai Wai', 'Nissin']}
df1 = pd.DataFrame(data) df1
       Brand
0     New
      Touch
1     Just
      Way
2     Wai
      Wai
3     Nissin
import pandas as pd
# Convert all names to uppercase
df['Name_uppercase'] = df['Brand'].str.upper()
df
   Review #          Brand                         Variety Style  \
0      2580      New Touch         T's Restaurant Tantanmen   Cup
1      2579       Just Way         Noodles Spicy Hot Sesame  Pack
2      2578         Nissin   Cup Noodles Chicken Vegetable   Cup
```

```
3          2577         Wei Lih   GGE Ramen Snack Tomato Flavor
           Pack
4          2576  Ching's Secret                Singapore Curry
           Pack
5          5           Vifon                 Hu Tiu Nam Vang  Bowl
6          4          Wai Wai  Oriental Style Instant Noodles  Pack
7          3          Wai Wai                  Tom Yum Shrimp  Pack
8          2          Wai Wai             Tom Yum Chili Flavor  Pack
9          1         Westbrae                      Miso Ramen  Pack
         Country  Stars Top Ten Rating Category  Name_uppercase
0             JP   3.75    None             High      NEW TOUCH
1             TW   1.00    None              Low       JUST WAY
2             United States   2.25    None             Low
              NISSIN
3             TW   2.75    None              Low        WEI LIH
4             IN   3.75    None             High  CHING'S SECRET
5             VN   3.50    None             High          VIFON
6             TH   1.00    None              Low        WAI WAI
7             TH   2.00    None              Low        WAI WAI
8             TH   2.00    None              Low        WAI WAI
9             United States   0.50    None             Low
              WESTBRAE
# Split the names based on a delimiter (e.g., space) and create a new
column for the first part of the name
df['First_name'] = df['Variety'].str.split(' ').str[0]
print(df)
   Review #          Brand                        Variety Style  \
0          2580        New Touch         T's Restaurant Tantanmen
           Cup
1          2579         Just Way         Noodles Spicy Hot Sesame
           Pack
2          2578            Nissin    Cup Noodles Chicken Vegetable
           Cup
3          2577           Wei Lih    GGE Ramen Snack Tomato Flavor
           Pack
4          2576  Ching's Secret                  Singapore Curry
           Pack
5          5           Vifon                 Hu Tiu Nam Vang  Bowl
6          4          Wai Wai  Oriental Style Instant Noodles  Pack
7          3          Wai Wai                  Tom Yum Shrimp  Pack
8          2          Wai Wai             Tom Yum Chili Flavor  Pack
9          1         Westbrae                      Miso Ramen  Pack
         Country  Stars Top Ten Rating Category  Name_uppercase
First_nam
0             JP   3.75    None             High      NEW TOUCH
T'
1             TW   1.00    None              Low       JUST WAY
```

```
Noodle
2  United States   2.25    None               Low          NISSIN
Cu
3              TW   2.75    None               Low         WEI LIH
GG
4              IN   3.75    None              High  CHING'S SECRET
Singapor
5              VN   3.50    None              High           VIFON
```

```
Hu
6              TH   1.00    None             Low          WAI WAI
Oriental
7              TH   2.00    None             Low          WAI WAI
Tom
8              TH   2.00    None             Low          WAI WAI
               Tom
9              United States   0.50    None             Low
               WESTBRAE
Miso

# Calculate the length of each name
df['Name_length'] = df['Variety'].str.len()
df

   Review #           Brand                         Variety Style  \
0         2580       New Touch        T's Restaurant Tantanmen   Cup
1         2579        Just Way        Noodles Spicy Hot Sesame  Pack
2         2578          Nissin   Cup Noodles Chicken Vegetable   Cup
3         2577         Wei Lih   GGE Ramen Snack Tomato Flavor  Pack
4         2576  Ching's Secret                 Singapore Curry  Pack
5         5           Vifon                 Hu Tiu Nam Vang  Bowl
6         4         Wai Wai  Oriental Style Instant Noodles  Pack
7         3         Wai Wai                  Tom Yum Shrimp  Pack
8         2         Wai Wai            Tom Yum Chili Flavor  Pack
9         1        Westbrae                      Miso Ramen  Pack

         Country  Stars Top Ten Rating Category  Name_uppercase
First_name  \
0              JP   3.75    None            High      NEW TOUCH
               T's
1              TW   1.00    None             Low       JUST WAY
Noodles
2              United States   2.25    None             Low
               NISSIN
Cup
3              TW   2.75    None             Low        WEI LIH
GGE
4              IN   3.75    None            High  CHING'S SECRET
Singapore
5              VN   3.50    None            High          VIFON
Hu
6              TH   1.00    None             Low        WAI WAI
Oriental
7              TH   2.00    None             Low        WAI WAI
Tom
8              TH   2.00    None             Low        WAI WAI
               Tom
```

```
9             United States   0.50    None             Low
              WESTBRAE
Miso

   Name_length
```

```
0           24
1           24
2           29
3           29
4           15
5           15
6           30
7           14
8           20
9           10
#creating pandas series
data=["tom","CHARANESH",np.nan,"hello@gmail.com","Timber"]
import pandas as pd import numpy as np s=pd.Series(data) s
0                tom
1                CHARANESH
2                NaN
3                hello@gmail.com
4                Timber
dtype: object

#checking whether the string i lower or not
s.str.islower()
0      True
1      False
2      NaN
3      True
4      False
dtype: object
#checking whether the string is upper or not
s.str.isupper()
0      False
1      True
2      NaN
3      False
4      False
dtype: object
#checking whether the string is digit or not
s.str.isdigit()
0      False
1      False
2      NaN
```

```
3    False
4    False
dtype: object
#checking whether the string contains @ or not
s.str.contains('@')
0      False
1      False
2      NaN
3      True
4      False
dtype: object
#replacing @ with $ if present s.str.replace('@','$')
0                tom
1                CHARANESH
2                NaN
3                hello$gmail.com
4                Timber
dtype: object

#checking whether the string starts with H or not
s.str.startswith('H')
0      False
1      False
2      NaN
3      False
4      False
dtype: object
```

Data Wrangling

1.concate/join/merge/reshape data frames

CONCATE

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

Data wrangling, also known as data munging, is the process of transforming and preparing raw data into a clean, organized, and structured format for analysis, visualization, or modeling.

Goals of Data Wrangling:

1. Improve data quality
2. Increase data accuracy
3. Enhance data consistency
4. Reduce data complexity
5. Prepare data for analysis

```python
import pandas as pd

df_sales1 = pd.DataFrame({
    "account": [363000, 383000, 412290, 412290, 412290, 218895, 218895,
218895],
    "name": ["WII LLC", "WILLC", "Jarde-Hilpert", "Jarde-Hilpert",
"Jarde-Hilpert", "Kulas Inc", "Kulas Inc", "Kulas Inc"],
    "order": [10001, 10001, 10005, 10006, 10005, 10000, 10006, 10006],
    "sku": ["B1-20000", "B1-80401", "51-06532", "91-47412", "81-
27722", "51-27722", "81-33067", "91-20000"],
    "quantity": [7, 3, 48, 44, 36, 32, 23, -1],
    "unit": [3369, 35.99, 5582, 78.91, 25.42, 95.65, 22.55, 72.18],
    "price": [235.83, 107.97, 2679.36, 3472.04, 915.12, 3001.12,
518.65, 72.18],
    "ext price": [1582.81, 323.91, 14962.88, 15372.16, 8746.32,
30411.52, 11930.45, -72.18]
})

print(df_sales1)
```

```
   account           name  order       sku  quantity     unit    price

    363000        WII LLC  10001  B1-20000         7  3369.00   235.83

\
0

1   383000          WILLC  10001  B1-80401         3    35.99   107.97

2   412290  Jarde-Hilpert  10005  51-06532        48  5582.00  2679.36

3   412290  Jarde-Hilpert  10006  91-47412        44    78.91  3472.04

4   412290  Jarde-Hilpert  10005  81-27722        36    25.42   915.12

5   218895      Kulas Inc  10000  51-27722        32    95.65  3001.12

6   218895      Kulas Inc  10006  81-33067        23    22.55   518.65

7   218895      Kulas Inc  10006  91-20000        -1    72.18    72.18
   ext price  0
1582.81
1     323.91
2     14962.88
3     15372.16
4     8746.32
5     30411.52
6     11930.45  7     -72.18  import pandas as pd
```

```
df_sales2 = pd.DataFrame({
    "account": [383081, 412291, 412291, 412291, 218896, 218896,
218896, 218896],
    "name": ["isabella", "Olvia", "Olvia", "Olivia", "Sophia",
"Sophis", "Sophis", "Sophia"],
    "order": [10002, 10004, 10004, 10004, 10007, 10007, 10007, 10007],
    "sku": ["C1-20000", "A1-08532", "A1-82801", "A1-00532", "A1-
27722", "C1-33087", "C1-33364", "C1-20000"],
    "quantity": [9, 55, 31, 5, 35, 33, 3, -6],
    "unit": [4389, 67.82, 145.02, 34.55, 67.46, 26.55, 67.30, 67.18],
    "price": [55583, 2379.36, 685.02, 782.95, 6761.12, 705.65, 676.00,
-82.10],
    "ext price": [400247, 131364.8, 4475.62, 3914.75, 23663.92,
23386.3, 2019.00, -535.08]
})

print(df_sales2)

   account      name  order       sku  quantity     unit     price
ext price
0   383081  isabella  10002  C1-20000         9  4389.00  55583.00
400247.00
1   412291     Olvia  10004  A1-08532        55    67.82   2379.36
131364.80
2   412291     Olvia  10004  A1-82801        31   145.02    685.02
4475.62
3   412291    Olivia  10004  A1-00532         5    34.55    782.95
3914.75
4   218896    Sophia  10007  A1-27722        35    67.46   6761.12
23663.92
5   218896    Sophis  10007  C1-33087        33    26.55    705.65
23386.30
6   218896    Sophis  10007  C1-33364         3    67.30    676.00
2019.00
7   218896    Sophia  10007  C1-20000        -6    67.18    -82.10
-535.08
# Concatenate the two DataFrames column-wise
df_concat_columns = pd.concat([df_sales1, df_sales2], axis=1)

# Display the concatenated DataFrame
print(df_concat_columns)

   account           name  order       sku  quantity     unit    price
\
0   363000        WII LLC  10001  B1-20000         7  3369.00   235.83

1   383000          WILLC  10001  B1-80401         3    35.99   107.97

2   412290  Jarde-Hilpert  10005  51-06532        48  5582.00  2679.36
```

```
3   412290  Jarde-Hilpert  10006  91-47412        44    78.91  3472.04

4   412290  Jarde-Hilpert  10005  81-27722        36    25.42   915.12

5   218895      Kulas Inc  10000  51-27722        32    95.65  3001.12

6   218895      Kulas Inc  10006  81-33067        23    22.55   518.65

    7   218895      Kulas Inc  10006  91-20000        -1    72.18

    72.18


   ext price  account      name  order       sku  quantity     unit
price  \
0     1582.81   383081  isabella  10002  C1-20000         9  4389.00
55583.00
1     323.91   412291     Olvia  10004  A1-08532        55    67.82
2379.36
2     14962.88   412291     Olvia  10004  A1-82801        31   145.02
685.02
3     15372.16   412291    Olivia  10004  A1-00532         5    34.55
      782.95
4     8746.32   218896    Sophia  10007  A1-27722        35    67.46
6761.12
5     30411.52   218896    Sophis  10007  C1-33087        33    26.55
705.65
6     11930.45   218896    Sophis  10007  C1-33364         3    67.30
676.00
7     -72.18   218896    Sophia  10007  C1-20000        -6    67.18
-82.10

   ext price  0
400247.00
1    131364.80
2    4475.62
3    3914.75
4    23663.92
5    23386.30
```

```
6    2019.00
7    -535.08
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)
```

df1

{"summary":"{\n \"name\": \"df1\",\n \"rows\": 2,\n \"fields\": [\n   {\n   \"column\": \"A\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"A1\",\n   \"A0\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"B\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"B1\",\n   \"B0\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"df1"} df2

{"summary":"{\n \"name\": \"df2\",\n \"rows\": 2,\n \"fields\": [\n   {\n   \"column\": \"A\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"A3\",\n   \"A2\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"B\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"B3\",\n   \"B2\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"df2"} result

{"summary":"{\n \"name\": \"result\",\n \"rows\": 4,\n \"fields\": [\n   {\n   \"column\": \"A\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 4,\n   \"samples\": [\n   \"A1\",\n   \"A3\",\n   \"A0\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"B\",\n   \"properties\": {\n   \"dtype\": \"string\",\n

\"num_unique_values\": 4,\n   \"samples\": [\n
\"B1\",\n
\"B3\",\n   \"B0\"\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   }\n   ]\
n}","type":"dataframe","variable_name":"result"}

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.

```
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')
df1
```

{"summary":"{\n \"name\": \"df1\",\n \"rows\": 3,\n \"fields\": [\n
{\n   \"column\": \"key\",\n   \"properties\": {\n
\"dtype\": \"string\",\n   \"num_unique_values\": 3,\n
\"samples\": [\n   \"A\",\n   \"B\",\n   \"C\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"value1\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\": 1,\n
\"min\": 1,\n   \"max\": 3,\n   \"num_unique_values\": 3,\n
\"samples\": [\n   1,\n   2,\n   3\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\ n
}\n]\n}","type":"dataframe","variable_name":"df1"} df2

{"summary":"{\n \"name\": \"df2\",\n \"rows\": 3,\n \"fields\": [\n
{\n   \"column\": \"key\",\n   \"properties\": {\n
\"dtype\": \"string\",\n   \"num_unique_values\": 3,\n
\"samples\": [\n   \"B\",\n   \"C\",\n   \"D\"\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"value2\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\": 1,\n
\"min\": 4,\n   \"max\": 6,\n   \"num_unique_values\": 3,\n
\"samples\": [\n   4,\n   5,\n   6\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\ n
}\n]\n}","type":"dataframe","variable_name":"df2"} result

{"summary":"{\n \"name\": \"result\",\n \"rows\": 2,\n \"fields\": [\n   {\n   \"column\": \"key\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"C\",\n   \"B\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"value1\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 2,\n   \"max\": 3,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   3,\n   2\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"value2\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 4,\n   \"max\": 5,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   5,\n   4\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"result"}

```
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') df1
```

{"summary":"{\n \"name\": \"df1\",\n \"rows\": 3,\n \"fields\": [\n   {\n   \"column\": \"key1\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"A\",\n   \"B\",\n   \"C\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"value1\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1,\n   \"min\": 1,\n   \"max\": 3,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   1,\n   2,\n   3\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"df1"} df2

{"summary":"{\n \"name\": \"df2\",\n \"rows\": 3,\n \"fields\": [\n   {\n   \"column\": \"key2\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"B\",\n   \"C\",\n   \"D\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"value2\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1,\n   \"min\": 4,\n   \"max\": 6,\n   \"num_unique_values\": 3,\n

\"samples\": [\n 4,\n 5,\n 6\n],\n \"semantic_type\": \"\",\n \"description\": \"\"\n }\ n }\n]\n}","type":"dataframe","variable_name":"df2"} result

{"summary":"{\n \"name\": \"result\",\n \"rows\": 2,\n \"fields\": [\n {\n \"column\": \"key1\",\n \"properties\": {\n \"dtype\": \"string\",\n \"num_unique_values\": 2,\n \"samples\": [\n \"C\",\n \"B\"\n],\n \"semantic_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"value1\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0,\n \"min\": 2,\n \"max\": 3,\n \"num_unique_values\": 2,\n \"samples\": [\n 3,\n 2\n],\n \"semantic_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"key2\",\n \"properties\": {\n \"dtype\": \"string\",\n \"num_unique_values\": 2,\n \"samples\": [\n \"C\",\n \"B\"\n],\n \"semantic_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"value2\",\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": 0,\n \"min\": 4,\n \"max\": 5,\n \"num_unique_values\": 2,\n \"samples\": [\n 5,\n 4\n],\n \"semantic_type\": \"\",\n \"description\": \"\"\n }\ n }\n]\n}","type":"dataframe","variable_name":"result"}

```
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
```

{"summary":"{\n \"name\": \"reshaped_result\",\n \"rows\": 2,\n \"fields\": [\n {\n \"column\": [\n \"key1\",\n \"\"\n],\n \"properties\": {\n \"dtype\": \"string\",\n \"num_unique_values\": 2,\n \"samples\": [\n \"C\",\n \"B\"\n],\n \"semantic_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": [\n \"value1\",\n \"B\"\n],\n \"properties\": {\n \"dtype\": \"number\",\n \"std\": null,\n \"min\": 2.0,\n

```
\"max\": 2.0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          2.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": [\n
\"value1\",\n        \"C\"\n      ],\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": null,\n        \"min\": 3.0,\
n        \"max\": 3.0,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          3.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": [\n        \"value2\",\n        \"B\"\n      ],\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
null,\n        \"min\": 4.0,\n        \"max\": 4.0,\n
\"num_unique_values\": 1,\n        \"samples\": [\n          4.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": [\n        \"value2\",\n
\"C\"\n      ],\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": 5.0,\n
\"max\": 5.0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          5.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"reshaped_result"}
```


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.

```
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.

```
#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.

```
# 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.

```
#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.

```
#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.

```
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
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
#reshaping import
pandas as pd
```

```
import numpy as np
data=pd.DataFrame(np.arange(6).reshape((2,3)),index=pd.Index(['apple',
'cherry'],name='fruit'),columns=pd.Index(['red','green','blue'],name='
color')) data
```

{"summary":"{\n \"name\": \"data\",\n \"rows\": 2,\n \"fields\": [\n   {\n   \"column\": \"fruit\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"cherry\",\n   \"apple\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"red\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 2,\n   \"min\": 0,\n   \"max\": 3,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   3,\n   0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"green\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 2,\n   \"min\": 1,\n   \"max\": 4,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   4,\n   1\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"blue\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 2,\n   \"min\": 2,\n   \"max\": 5,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   5,\n   2\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"data"}

```
result=data.stack()
result
```

```
fruit   color
apple   red      0
green    1
blue     2 cherry
red      3
green    4
blue     5 dtype:
int64
result.unstack(0)
```

{"summary":"{\n \"name\": \"result\",\n \"rows\": 3,\n \"fields\":
[\n   {\n   \"column\": \"color\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"red\",\n   \"green\",\n   \"blue\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"apple\",\n   \"properties\": {\n   \"dtype\": \"number\",\n

```
\"std\": 1,\n        \"min\": 0,\n        \"max\": 2,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          0,\n
1,\n          2\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cherry\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1,\n        \"min\": 3,\n        \"max\": 5,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          3,\n
4,\n          5\n         ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
result.unstack('fruit')
```


```
{"summary":"{\n  \"name\": \"result\",\n  \"rows\": 3,\n  \"fields\":
[\n    {\n      \"column\": \"color\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"red\",\n          \"green\",\n
\"blue\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"apple\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1,\n        \"min\": 0,\n        \"max\": 2,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          0,\n
1,\n          2\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cherry\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 1,\n        \"min\": 3,\n        \"max\": 5,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          3,\n
4,\n          5\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


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

```
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
```

{"summary":"{\n \"name\": \"pivot_table\",\n \"rows\": 2,\n \"fields\": [\n   {\n   \"column\": \"A\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"foo\",\n   \"bar\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"one\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1,\n   \"min\": 3,\n   \"max\": 5,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   3,\n   5\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"two\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 3,\n   \"max\": 4,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   3,\n   4\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"pivot_table"}

3.Read dataframe to create a cross table

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

```python
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
```

{"summary":"{\n \"name\": \"cross_table\",\n \"rows\": 2,\n \"fields\": [\n   {\n   \"column\": \"Category\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"B\",\n   \"A\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"No\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 1,\n   \"max\": 1,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   1\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Yes\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0,\n   \"min\": 1,\n   \"max\": 2,\n   \"num_unique_values\": 2,\n

```
\"samples\": [\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"cross_table"}
```

Plotting and Visualization

```
import pandas as pd
import seaborn as sns

data=sns.get_dataset_names()
data
```

```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'dowjones',
 'exercise',
 'flights',
 'fmri',
 'geyser',
 'glue',
 'healthexp',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'seaice',
 'taxis',
 'tips',
 'titanic']
```

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

exercise Dataset

Description: Contains information about car crashes.

Variables: total speeding alcohol not_distracted no_previous ins_premium .

Use Cases: Histograms to show distributions, bar charts, and scatter plots to explore the relationship between speeding and alcohol

```
df=sns.load_dataset('car_crashes')
print(df.head())
```

```
  total  speeding  alcohol  not_distracted  no_previous  ins_premium


    18.8     7.332    5.640          18.048       15.040        784.55


\
0

1     18.1     7.421    4.525          16.290       17.014       1053.48

2     18.6     6.510    5.208          15.624       17.856        899.47

3     22.4     4.032    5.824           21.056       21.280        827.34
4   12.0     4.200    3.360          10.920       10.680        878.41
   ins_losses abbrev
0      145.08     AL
1      133.93     AK
2      110.35     AZ
3      142.39     AR
4      165.63     CA
# Display basic information about the dataset
print(df.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51 entries, 0 to 50
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
0     total           51 non-null     float64
1     speeding        51 non-null     float64
2     alcohol         51 non-null     float64
3     not_distracted  51 non-null     float64
4     no_previous     51 non-null     float64
5     ins_premium     51 non-null     float64
6     ins_losses      51 non-null     float64  7   abbrev          51
non-null     object
dtypes: float64(7), object(1)
memory usage: 3.3+ KB
None

# Display summary statistics for numerical columns
print(df.describe())
           total   speeding    alcohol  not_distracted  no_previous  \
```

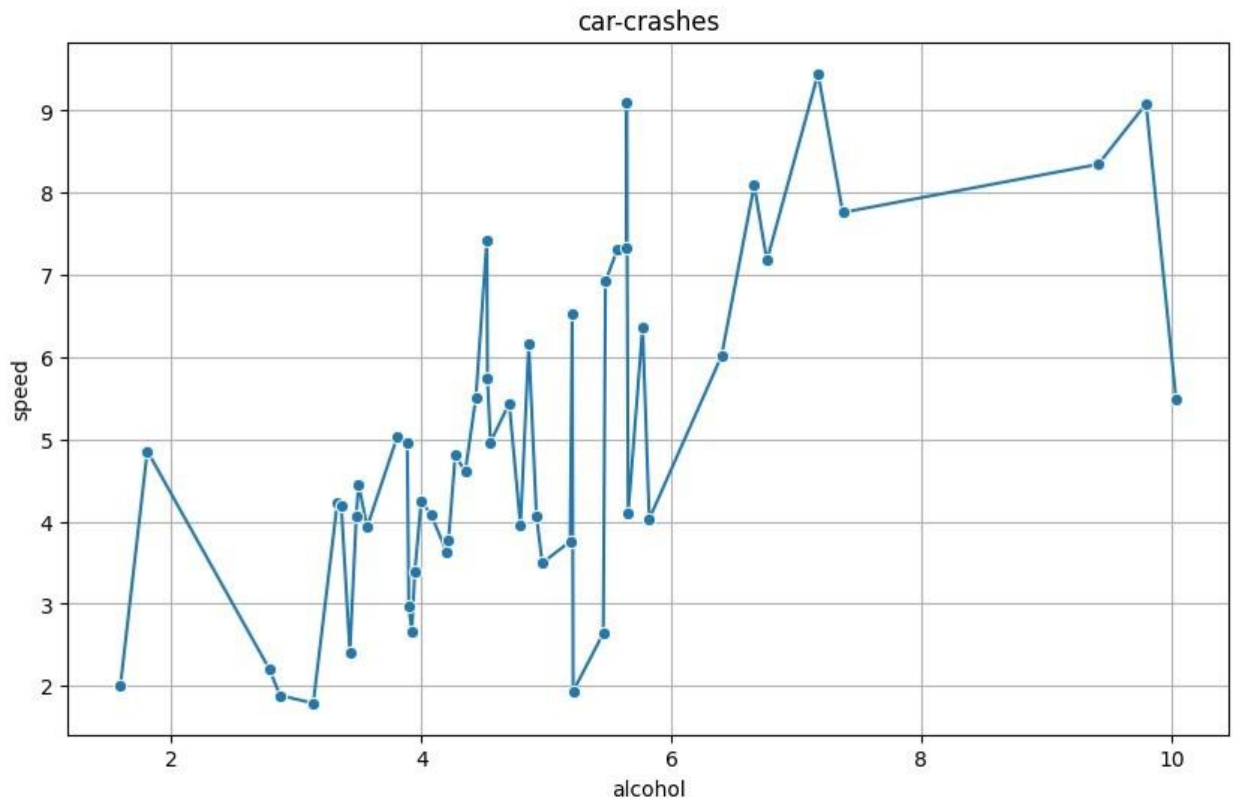
```
count  51.000000  51.000000  51.000000        51.000000    51.000000
mean   15.790196   4.998196   4.886784        13.573176    14.004882
std     4.122002   2.017747   1.729133         4.508977     3.764672
min     5.900000   1.792000   1.593000         1.760000     5.900000
25%    12.750000   3.766500   3.894000        10.478000    11.348000
50%    15.600000   4.608000   4.554000        13.857000    13.775000
75%    18.500000   6.439000   5.604000        16.140000    16.755000
```

```
max    23.900000   9.450000  10.038000       23.661000    21.280000
       ins_premium  ins_losses
count    51.000000   51.000000
mean    886.957647  134.493137
std     178.296285   24.835922
min     641.960000   82.750000
25%     768.430000  114.645000
50%     858.970000  136.050000
75%    1007.945000  151.870000
max    1301.520000  194.780000
```

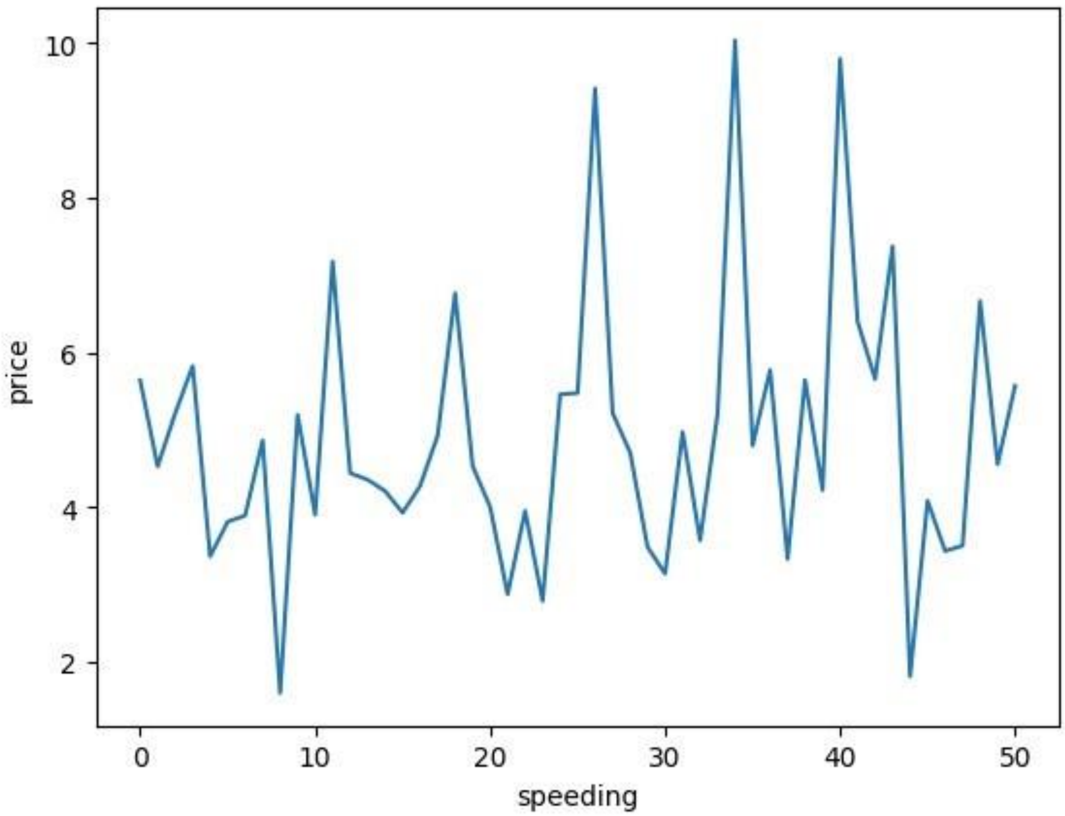
a. Line Plot

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

```
# Line plot:car crashes import
matplotlib.pyplot as plt import
numpy as np import pandas as pd
plt.figure(figsize=(10, 6))
sns.lineplot(x='alcohol', y='speeding', data=df, estimator=np.mean,
marker='o')
plt.title('car-crashes')
plt.xlabel('alcohol')
plt.ylabel('speed')
plt.grid() plt.show()
```




```
#Example 1: alcohol over speeding.
plt.plot(df['alcohol']); plt.xlabel('speeding');
plt.ylabel('price'); plt.show()
```



Line Plot of price according to carat

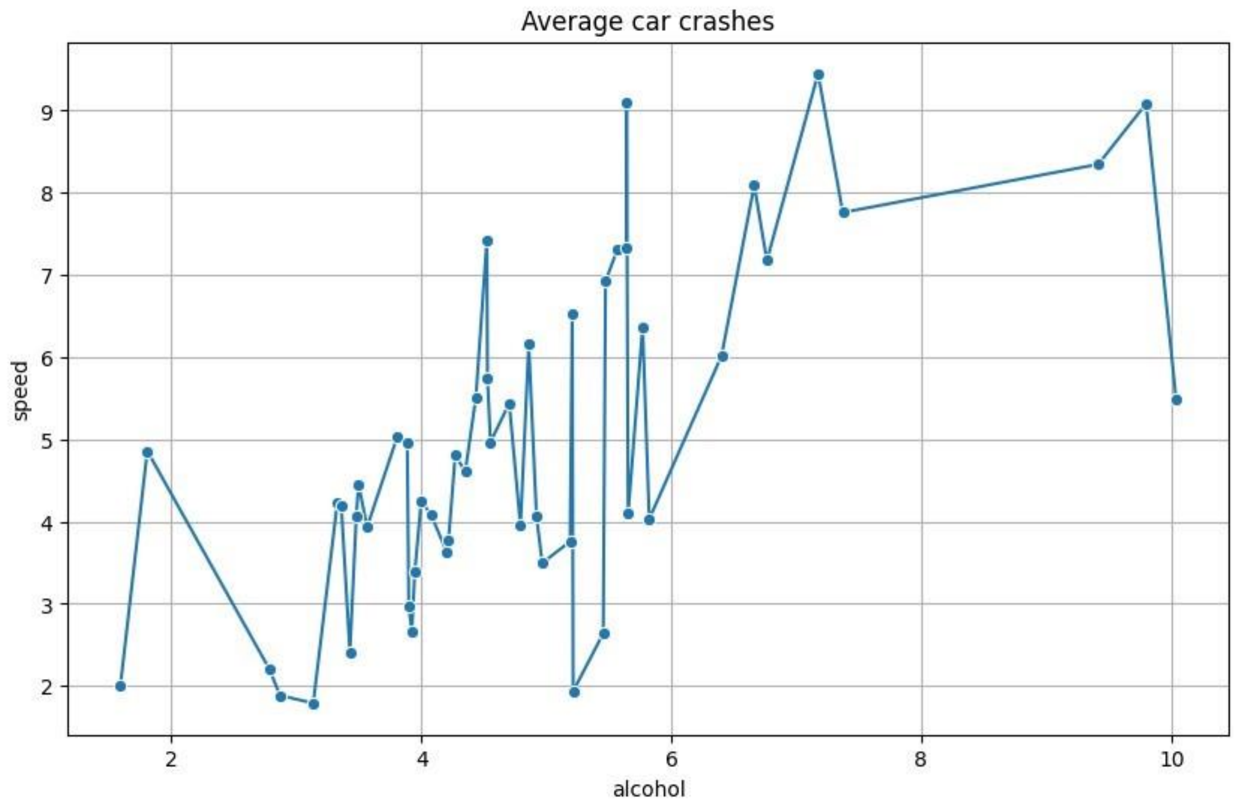
This plot shows the price according to carat.

```
plt.figure(figsize=(10, 6))
sns.lineplot(data=df, x='alcohol', y='speeding', hue='total',
estimator=np.mean, marker='o')
plt.title('price according to carat')
plt.xlabel('carat')
plt.ylabel('price') plt.grid()
plt.show()
```

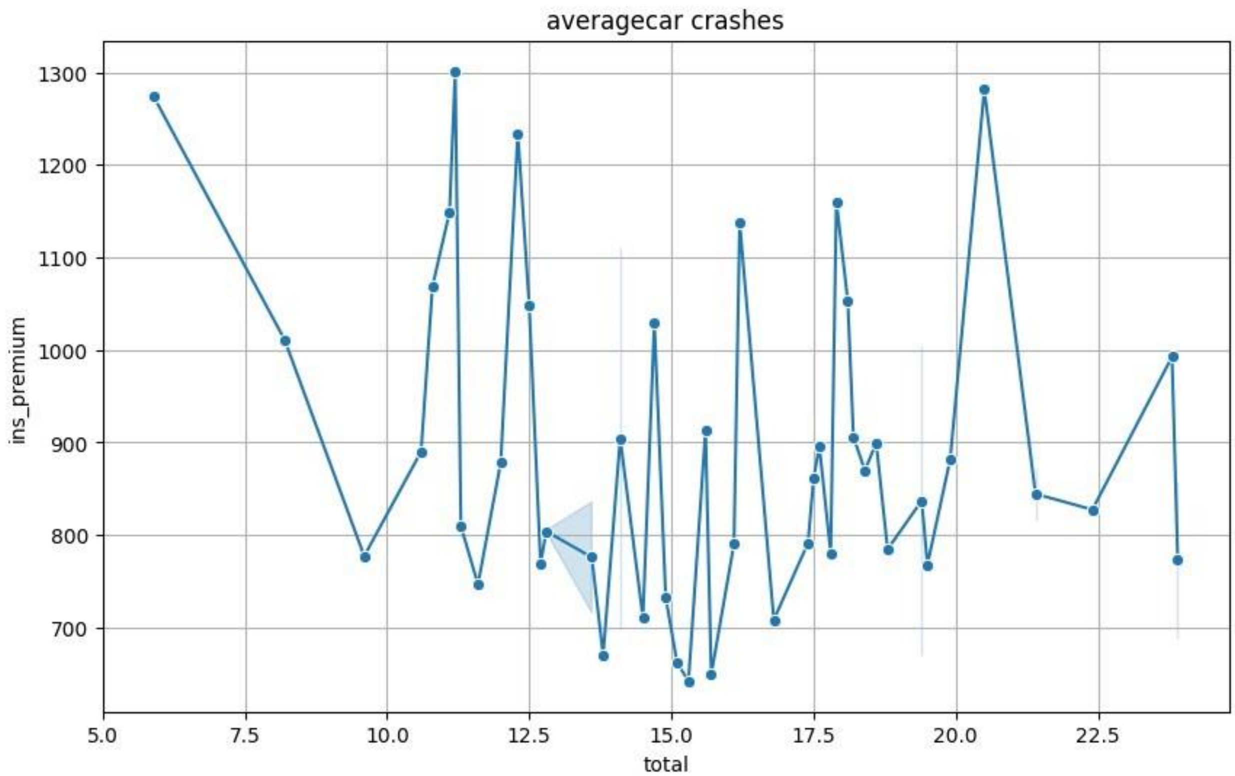
Line Plot of total pricee

This plot shows the average price per carat.

```
plt.figure(figsize=(10, 6))
sns.lineplot(data=df, x='alcohol', y='speeding', estimator=np.mean,
marker='o')
plt.title('Average car crashes')
plt.xlabel('alcohol')
plt.ylabel('speed') plt.grid()
plt.show()
```




```
plt.figure(figsize=(10, 6))
sns.lineplot(data=df, x='total', y='ins_premium', estimator=np.mean,
marker='o')
plt.title('averagecar crashes')
plt.xlabel('total')
plt.ylabel('ins_premium')
plt.grid() plt.show()
```

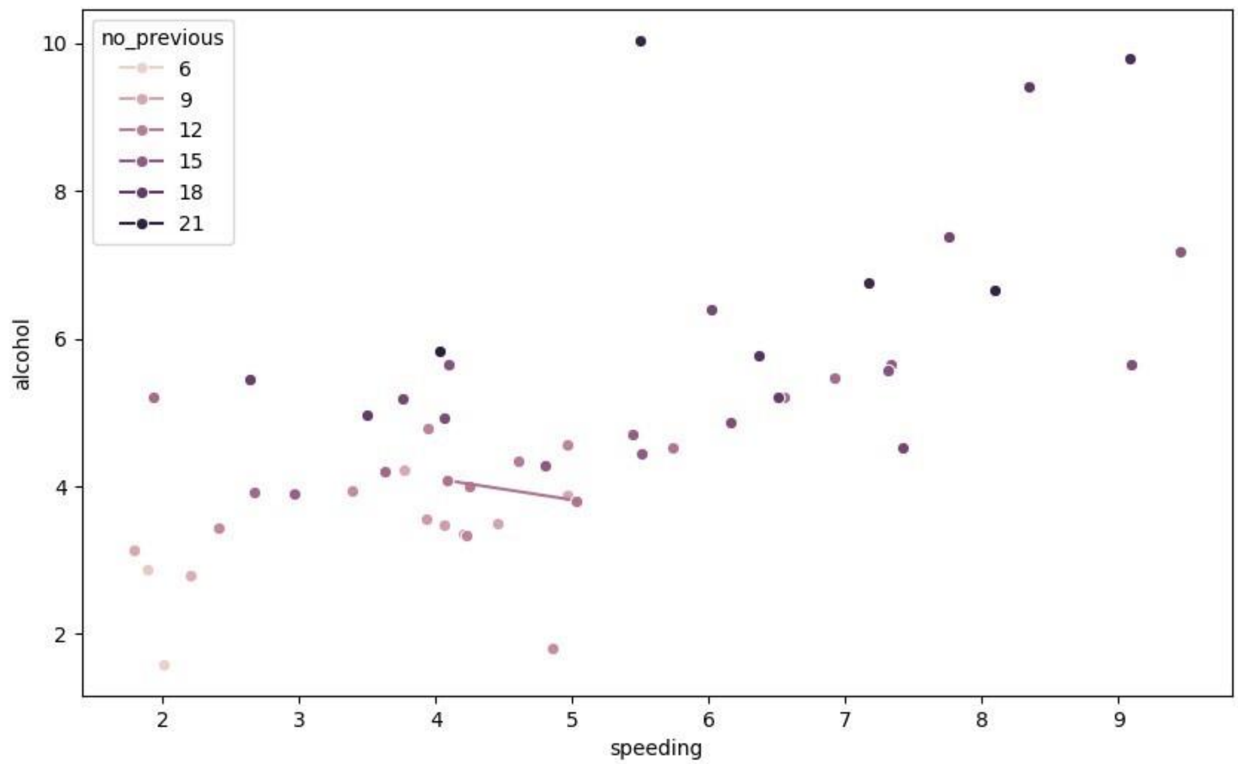


Line Plot of price according to carat

This plot shows the average car crashes, further separated total.

```
plt.figure(figsize=(10, 6))
sns.lineplot(data=df, x='speeding', y='alcohol', hue='no_previous',
marker='o')
```

```
<Axes: xlabel='speeding', ylabel='alcohol'>
```

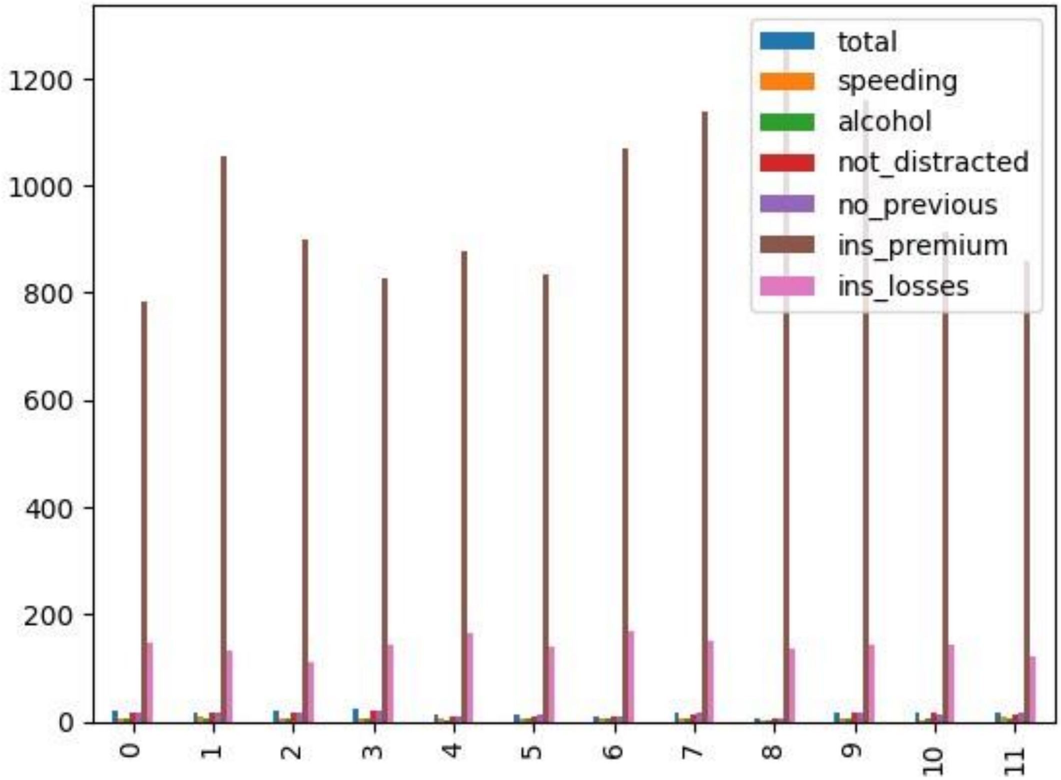


b. Bar Plot

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

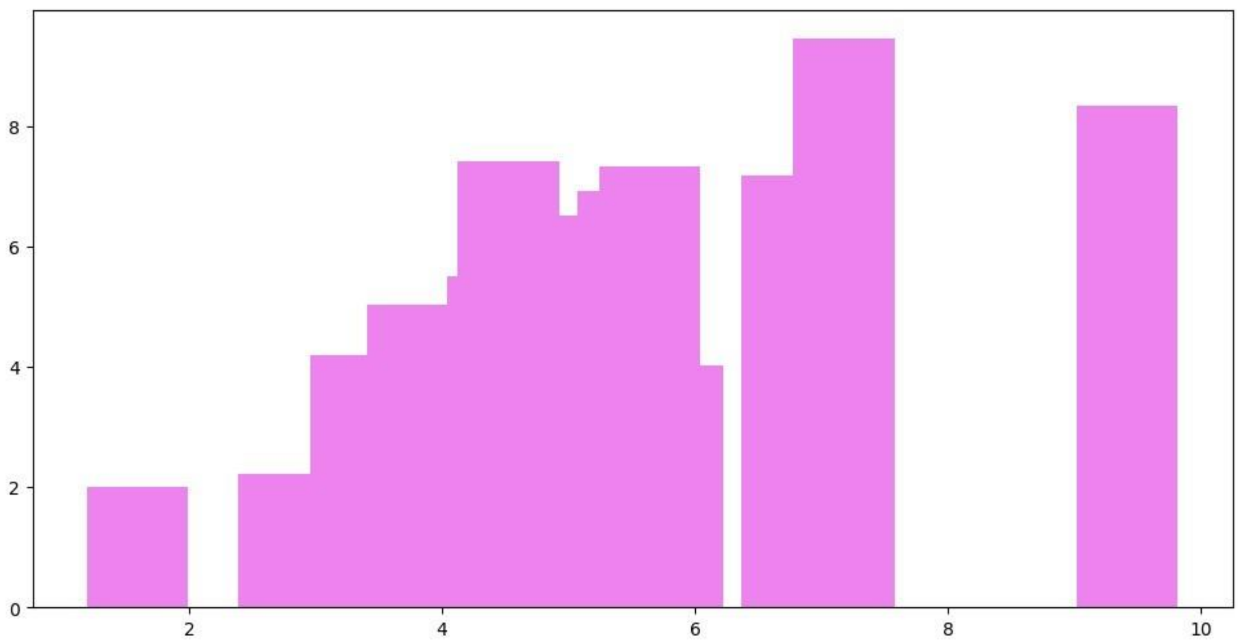
```
df2=df.head(12) df2.plot.bar()

<Axes: >
```




```
plt.figure(figsize=(12, 6))
plt.bar(df['alcohol'].iloc[:30], df['speeding'].iloc[:30],
color='violet')
```

```
<BarContainer object of 30 artists>
```

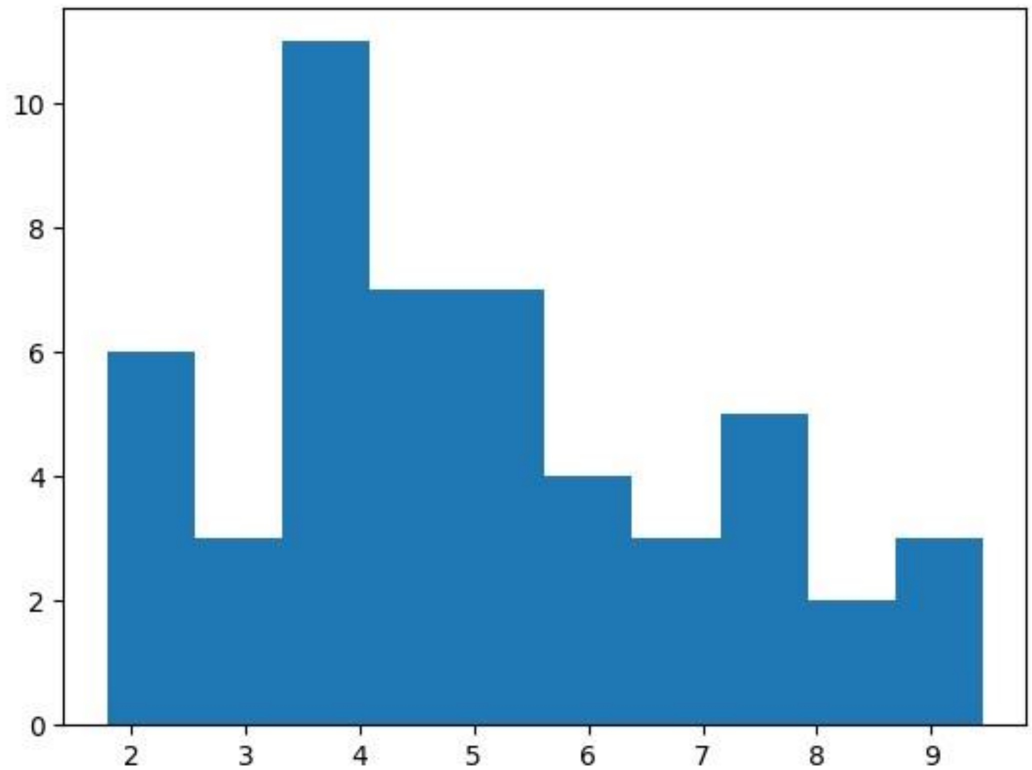


c.Histogram

A histogram is a type of plot that allows you to visualize the distribution of a single variable by dividing the data into bins and counting the number of observations that fall into each bin. It is commonly used to understand the distribution, spread, and skewness of numerical data. In a histogram, the x-axis represents the range of values (divided into intervals or bins), and the yaxis represents the frequency of occurrences within each interval.

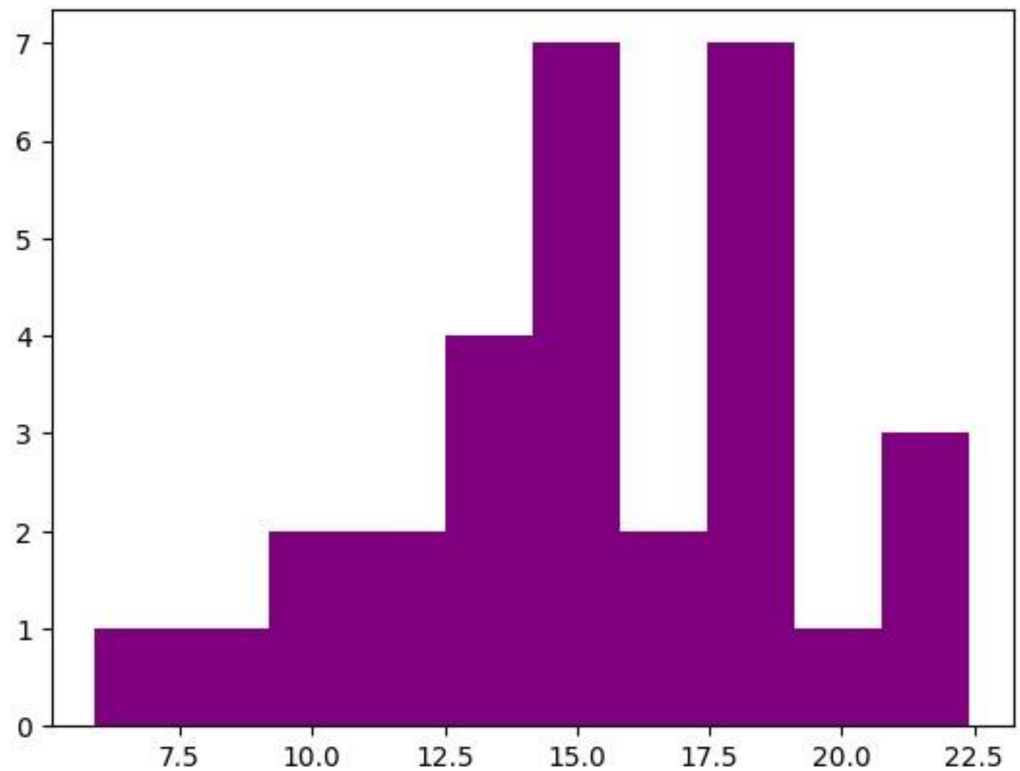
```
plt.hist(df['speeding'])
```

```
(array([ 6.,  3., 11.,  7.,  7.,  4.,  3.,  5.,  2.,  3.]),
 array([1.792 , 2.5578, 3.3236, 4.0894, 4.8552, 5.621 , 6.3868,
7.1526,
        7.9184, 8.6842, 9.45  ]),
 <BarContainer object of 10 artists>)
```




```
plt.hist(df['total'].iloc[:30],color='purple')
```

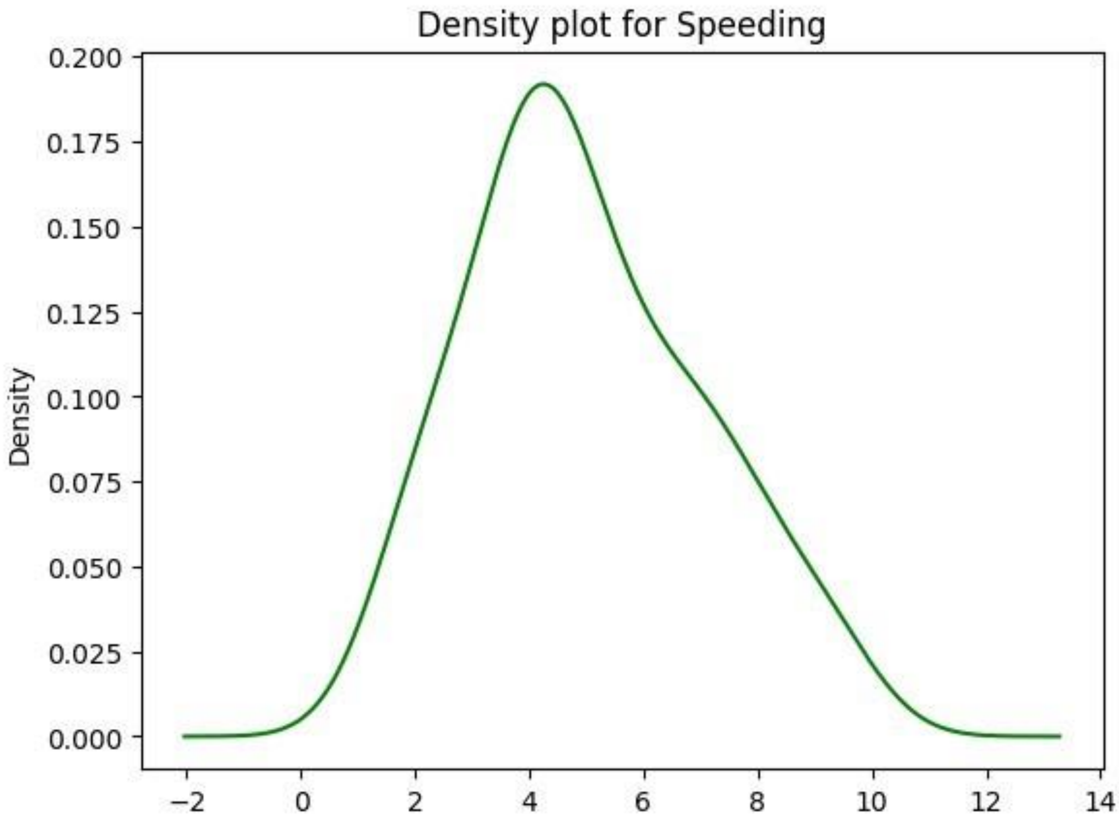
```
(array([1., 1., 2., 2., 4., 7., 2., 7., 1., 3.]),
 array([ 5.9 ,  7.55,  9.2 , 10.85, 12.5 , 14.15, 15.8 , 17.45, 19.1 ,
        20.75, 22.4 ]),
 <BarContainer object of 10 artists>)
```



d.Density Plot

A density plot (or Kernel Density Estimate (KDE) plot) is a data visualization technique used to show the distribution of a continuous variable in a smooth manner. Unlike histograms, which represent the data using bins and bars, density plots use a continuous curve to represent the frequency of values.

```
df.speeding.plot.density(color='green')
plt.title('Density plot for Speeding') plt.show()
```

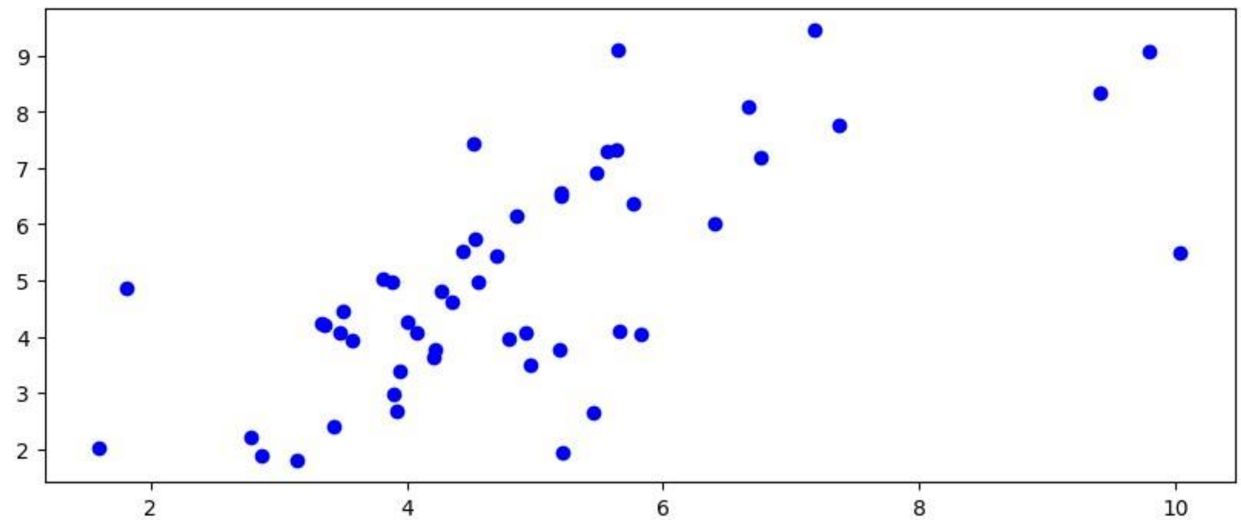


e.Scatter Plot

A scatter plot is a type of plot used to visualize the relationship between two numerical variables. Each point on the scatter plot represents an observation in the dataset, with the xcoordinate corresponding to the value of one variable and the y-coordinate corresponding to the value of another. Scatter plots are often used to identify patterns, trends, correlations, and outliers between two variables.

```
plt.figure(figsize=(10, 4))
plt.scatter(df['alcohol'], df['speeding'], color='blue')

<matplotlib.collections.PathCollection at 0x17237225760>
```




```
plt.scatter(df['carat'].iloc[30:], df['price'].iloc[30:],
color='green')

<matplotlib.collections.PathCollection at 0x2332fdcd190>
```