**15.Grouping and aggregations**

import pandas as pd

# Sample dataset

data = {

    'Department': ['HR', 'IT', 'HR', 'Finance', 'IT', 'Finance', 'HR'],

    'Employee': ['A', 'B', 'C', 'D', 'E', 'F', 'G'],

    'Salary': [50000, 55000, 60000, 65000, 70000, 75000, 80000],

    'Experience': [2, 3, 4, 5, 6, 7, 8]}

df = pd.DataFrame(data)

print("Original DataFrame:\n", df)

# 1. Group by Department and find mean Salary

mean\_salary = df.groupby('Department')['Salary'].mean()

print("\nMean Salary by Department:\n", mean\_salary)

# 2. Group by Department and compute mean and sum of Salary and Experience

aggregated = df.groupby('Department').agg({

    'Salary': ['mean', 'sum'],

    'Experience': ['mean', 'sum']})

print("\nAggregated Salary & Experience by Department:\n", aggregated)

**Output:** Original DataFrame:

   Department Employee  Salary  Experience

0         HR        A   50000           2

1         IT        B   55000           3

2         HR        C   60000           4

3    Finance        D   65000           5

4         IT        E   70000           6

5    Finance        F   75000           7

6         HR        G   80000           8

Mean Salary by Department:

 Department

Finance    70000.000000

HR         63333.333333

IT         62500.000000

Name: Salary, dtype: float64

Aggregated Salary & Experience by Department:

                   Salary         Experience

                    mean     sum       mean sum

Department

Finance     70000.000000  140000   6.000000  12

HR          63333.333333  190000   4.666667  14

IT          62500.000000  125000   4.500000   9

Pivot Table - Average Salary by Department:

                   Salary

Department

Finance     70000.000000

HR          63333.333333

IT          62500.000000

Pivot Table with multiple metrics:

                  mean                      sum

           Experience        Salary Experience  Salary

Department

Finance      6.000000  70000.000000         12  140000

HR           4.666667  63333.333333         14  190000

IT           4.500000  62500.000000          9  125000

**Mathematical operations:**

import pandas as pd

# Sample data

data = {

    'Math': [80, 85, 90, 95, 100],

    'Science': [75, 80, 85, 90, 95]}

df = pd.DataFrame(data)

print("Original DataFrame:\n", df)

# Column-wise operations

print("\nSum:\n", df.sum())

print("Mean:\n", df.mean())

print("Median:\n", df.median())

print("Min:\n", df.min())

print("Max:\n", df.max())

print("Standard Deviation (std):\n", df.std())

print("Variance (var):\n", df.var())

print("Product of elements:\n", df.prod())

print("Cumulative Sum:\n", df.cumsum())

# You can also do these on a Series (like just one column)

math\_scores = df['Math']

print("\nMath Scores:\n", math\_scores)

print("Cumulative Sum of Math Scores:\n", math\_scores.cumsum())

**Output:** Original DataFrame:

    Math  Science

0    80       75

1    85       80

2    90       85

3    95       90

4   100       95

Sum:

 Math       450

Science    425

dtype: int64

Mean:

 Math       90.0

Science    85.0

dtype: float64

Median:

 Math       90.0

Science    85.0

dtype: float64

Min:

 Math       80

Science    75

dtype: int64

Max:

 Math       100

Science     95

dtype: int64

Standard Deviation (std):

 Math       7.905694

Science    7.905694

dtype: float64

Variance (var):

 Math       62.5

Science    62.5

dtype: float64

Product of elements:

 Math       5814000000

Science    4360500000

dtype: int64

Cumulative Sum:

    Math  Science

0    80       75

1   165      155

2   255      240

3   350      330

4   450      425

Math Scores:

 0     80

1     85

2     90

3     95

4    100

Name: Math, dtype: int64

Cumulative Sum of Math Scores:

 0     80

1    165

2    255

3    350

4    450

**Aggregation & Descriptive stats:**

import pandas as pd

# Sample dataset

data = {

    'Department': ['IT', 'HR', 'Finance', 'IT', 'HR', 'Finance', 'HR'],

    'Salary': [50000, 55000, 60000, 65000, 70000, 75000, 80000],

    'Experience': [3, 4, 5, 6, 7, 8, 9]}

df = pd.DataFrame(data)

print("Original DataFrame:\n", df)

# 1. Aggregation: group by Department and calculate sum, mean, min, and max

agg\_results = df.groupby('Department').agg({

    'Salary': ['sum', 'mean', 'min', 'max'],

    'Experience': ['mean', 'std']})

print("\nAggregated stats by Department:\n", agg\_results)

# 2. Descriptive Statistics for the whole DataFrame

desc\_stats = df.describe()

print("\nDescriptive Statistics:\n", desc\_stats)

# 3. Descriptive Statistics for each group

grouped\_desc = df.groupby('Department').describe()

print("\nDescriptive Statistics by Department:\n", grouped\_desc)

**Output:**Original DataFrame:

    Department  Salary  Experience

0         IT   50000           3

1         HR   55000           4

2    Finance   60000           5

3         IT   65000           6

4         HR   70000           7

5    Finance   75000           8

6         HR   80000           9

Aggregated stats by Department:

             Salary                             Experience

               sum          mean    min    max       mean       std

Department

Finance     135000  67500.000000  60000  75000   6.500000  2.121320

HR          205000  68333.333333  55000  80000   6.666667  2.516611

IT          115000  57500.000000  50000  65000   4.500000  2.121320

Descriptive Statistics:

              Salary  Experience

count      7.000000    7.000000

mean   65000.000000    6.000000

std    10801.234497    2.160247

min    50000.000000    3.000000

25%    57500.000000    4.500000

50%    65000.000000    6.000000

75%    72500.000000    7.500000

max    80000.000000    9.000000

Descriptive Statistics by Department:

            Salary                                       ... Experience

            count          mean           std      min  ...        25%  50%   75%  max

Department                                              ...

Finance       2.0  67500.000000  10606.601718  60000.0  ...       5.75  6.5  7.25  8.0

HR            3.0  68333.333333  12583.057392  55000.0  ...       5.50  7.0  8.00  9.0

IT            2.0  57500.000000  10606.601718  50000.0  ...       3.75  4.5  5.25  6.0

**Boolean operations:**

import pandas as pd

# Create a sample Series

s = pd.Series([11, 12, 13, 14, 15, 16, 17])

print("Original Series:\n", s)

# Boolean operation: Check which values are greater than 20

bool\_mask = s > 14

print("\nBoolean mask (s > 14):\n", bool\_mask)

# Use the boolean mask to filter the Series

filtered = s[s > 14]

print("\nFiltered Series (values > 14):\n", filtered)

**Output:** Original Series:

 0    11

1    12

2    13

3    14

4    15

5    16

6    17

dtype: int64

Boolean mask (s > 14):

 0    False

1    False

2    False

3    False

4     True

5     True

6     True

dtype: bool

Filtered Series (values > 14):

 4    15

5    16

6    17

**Heading Missing values:**

import pandas as pd

import numpy as np

# Sample DataFrame with missing values

data = {

    'Name': ['Alice', 'Bob', 'Charlie', 'David'],

    'Age': [20, np.nan, 25, np.nan],

    'Score': [85, 90, np.nan, 95]}

df = pd.DataFrame(data)

print("Original DataFrame:\n", df)

# Detect missing values

print("\nMissing values (isnull):\n", df.isnull())

# Detect non-missing values

print("\nNon-missing values (notnull):\n", df.notnull())

# Drop rows with any missing values

df\_dropped = df.dropna()

print("\nDataFrame after dropping rows with missing values:\n", df\_dropped)

**Output:**

Original DataFrame:

       Name   Age  Score

0    Alice  20.0   85.0

1      Bob   NaN   90.0

2  Charlie  25.0    NaN

3    David   NaN   95.0

Missing values (isnull):

     Name    Age  Score

0  False  False  False

1  False   True  False

2  False  False   True

3  False   True  False

Non-missing values (notnull):

    Name    Age  Score

0  True   True   True

1  True  False   True

2  True   True  False

3  True  False   True

DataFrame after dropping rows with missing values:

     Name   Age  Score

0  Alice  20.0   85.0

**Basic information :**

import pandas as pd

# Create a sample Series

s = pd.Series([10, 15, 20, 25, 30, 35, 40], name='Marks')

print("Original Series:\n", s)

# First few elements

print("\nHead (first 3 elements):\n", s.head(3))

# Last few elements

print("\nTail (last 3 elements):\n", s.tail(3))

# Data type of the elements

print("\nData Type (dtype):", s.dtype)

# Shape (tuple with number of elements,)

print("Shape:", s.shape)

# Size (number of elements)

print("Size:", s.size)

# Index object

print("Index:", s.index)

# Underlying NumPy array of values

print("Values:", s.values)

**Output:** Original Series:

 0     5

1    10

2    15

3    20

4    25

5    30

6    45

Name: Marks, dtype: int64

Head (first 5 elements):

 0     5

1    10

2    15

3    20

4    25

Name: Marks, dtype: int64

Tail (last 5 elements):

 2    15

3    20

4    25

5    30

6    45

Name: Marks, dtype: int64

Data Type (dtype): int64

Shape: (7,)

Size: 7

Index: RangeIndex(start=0, stop=7, step=1)

Values: [ 5 10 15 20 25 30 45]

**Indexing &selection:**

import pandas as pd

# Create a Series with custom labels

s = pd.Series([50, 100, 150, 200, 250], index=['a', 'b', 'c', 'd', 'e'])

print("Original Series:\n", s)

# Indexing using integer index (positional)

print("\nAccess using s[3]:", s[3])  # Third element (150)

# Indexing using .loc[label]

print("Access using s.loc['c']:", s.loc['c'])  # Element with label 'c' (300)

# Indexing using .iloc[i]

print("Access using s.iloc[4]:", s.iloc[4])  # Same as above (200)

# Slicing using label (inclusive of stop)

print("\nSlicing with .loc['b':'d']:\n", s.loc['b':'d'])

# Slicing using position (exclusive of stop)

print("\nSlicing with .iloc[1:4]:\n", s.iloc[1:4])

**Output:** Original Series:

 a     50

b    100

c    150

d    200

e    250

dtype: int64

Access using s[3]: 200

Access using s.loc['c']: 150

Access using s.iloc[4]: 250

Slicing with .loc['b':'d']:

 b    100

c    150

d    200

dtype: int64

Slicing with .iloc[1:4]:

 b    100

c    150

d    200

dtype: int64

**16.Multiple arrays**

import numpy as np

array1 = np.array([1, 2, 3, 4, 5])

array2 = np.array([4, 5, 6, 7, 8])

vertical\_stack = np.vstack((array1, array2))

print("Vertical Stack:\n", vertical\_stack)

horizontal\_stack = np.hstack((array1, array2))

print("Horizontal Stack:\n", horizontal\_stack)

common\_items = np.intersect1d(array1, array2)

print("Common Items:", common\_items)

unique\_items = np.setdiff1d(array1, array2)

print("Unique Items from array1:", unique\_items)

matching\_positions = np.where(array1 == array2)

print("Matching Positions:", matching\_positions)

**Output:**  [[1 2 3 4 5]

 [4 5 6 7 8]]

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

[4 5]

[1 2 3]