# Importing pandas Library

```
import pandas as pd
```

## Data Manipulation and Data Analysis

Data Manipulation

Definition:

Data manipulation is the process of cleaning, transforming, and organizing raw data to make it suitable for analysis.

Key tasks involved:

-Cleaning data: Handling missing values, fixing typos, and removing duplicates.

-Transforming data: Changing the format (e.g., converting dates), normalizing values, or aggregating data.

-Filtering and sorting: Selecting relevant rows or columns based on conditions.

-Merging/joining datasets: Combining data from multiple sources.

-Reshaping: Pivoting or unpivoting data structures.

Data Analysis

Definition:

Data analysis is the process of inspecting, modeling, and interpreting data to discover useful information and support decision-making.

Key tasks involved:

-Descriptive statistics: Mean, median, mode, standard deviation, etc.

-Data visualization: Charts and graphs to explore patterns and trends (using matplotlib, seaborn, or Tableau)

-Exploratory Data Analysis (EDA): Deep dive into data to uncover hidden structures and anomalies.

-Statistical analysis: Hypothesis testing, correlation, regression analysis, etc.

-Predictive modeling: Using machine learning to predict future outcomes.

# Importing Dastsets

```
file_path = "D:/Data Science/Datatsets/sales_data_sample.csv"
sales = pd.read_csv(file_path, encoding='latin1')
```

```
print(sales)
'''  If excel file is present then - sales = pd.read_excel(file_path, 
encoding='latin1') and use .xlsx in file  '''
```

```
      ORDERNUMBER  QUANTITYORDERED  PRICEEACH  ORDERLINENUMBER  
SALES  \
0           10107               30      95.70                2  
2871.00   
1           10121               34      81.35                5  
2765.90   
2           10134               41      94.74                2  
3884.34   
3           10145               45      83.26                6  
3746.70   
4           10159               49     100.00               14  
5205.27   
...           ...              ...        ...              ...      ..
.   
2818        10350               20     100.00               15  
2244.40   
2819        10373               29     100.00                1  
3978.51   
2820        10386               43     100.00                4  
5417.57   
2821        10397               34      62.24                1  
2116.16   
2822        10414               47      65.52                9  
3079.44   

            ORDERDATE    STATUS  QTR_ID  MONTH_ID  YEAR_ID  ...  \
0      2/24/2003 0:00   Shipped       1         2     2003  ...   
1       5/7/2003 0:00   Shipped       2         5     2003  ...   
2       7/1/2003 0:00   Shipped       3         7     2003  ...   
3      8/25/2003 0:00   Shipped       3         8     2003  ...   
4     10/10/2003 0:00   Shipped       4        10     2003  ...   
...               ...       ...     ...       ...      ...  ...   
2818   12/2/2004 0:00   Shipped       4        12     2004  ...   
2819   1/31/2005 0:00   Shipped       1         1     2005  ...   
2820    3/1/2005 0:00  Resolved       1         3     2005  ...   
2821   3/28/2005 0:00   Shipped       1         3     2005  ...   
2822    5/6/2005 0:00   On Hold       2         5     2005  ...   

                       ADDRESSLINE1  ADDRESSLINE2           CITY STATE  
\
0           897 Long Airport Avenue           NaN            NYC    NY   

1                59 rue de l'Abbaye           NaN          Reims   NaN   

2     27 rue du Colonel Pierre Avia           NaN          Paris   NaN   
```

```
3                78934 Hillside Dr.           NaN       Pasadena    CA

4                   7734 Strong St.           NaN  San Francisco    CA

...                             ...           ...            ...   ...

2818             C/ Moralzarzal, 86           NaN         Madrid   NaN

2819                    Torikatu 38           NaN           Oulu   NaN

2820             C/ Moralzarzal, 86           NaN         Madrid   NaN

2821          1 rue Alsace-Lorraine           NaN       Toulouse   NaN

2822             8616 Spinnaker Dr.           NaN         Boston    MA


     POSTALCODE  COUNTRY TERRITORY CONTACTLASTNAME CONTACTFIRSTNAME
DEALSIZE
0         10022      USA       NaN              Yu             Kwai
Small
1         51100   France      EMEA         Henriot             Paul
Small
2         75508   France      EMEA        Da Cunha           Daniel
Medium
3         90003      USA       NaN           Young            Julie
Medium
4           NaN      USA       NaN           Brown            Julie
Medium
...         ...      ...       ...             ...              ...
...
2818      28034    Spain      EMEA          Freyre            Diego
Small
2819      90110  Finland      EMEA       Koskitalo           Pirkko
Medium
2820      28034    Spain      EMEA          Freyre            Diego
Medium
2821      31000   France      EMEA          Roulet          Annette
Small
2822      51003      USA       NaN         Yoshido             Juri
Medium

[2823 rows x 25 columns]

"  If excel file is present then - sales = pd.read_excel(file_path,
encoding='latin1') and use .xlsx in file  "

# series - A Series in pandas is a one-dimensional labeled array that
can hold any data type — integers, strings, floats, Python objects,
etc.
```

```
#Think of it like a column in Excel or a single column from a
DataFrame, but with labels (called the index) for each value.

series=pd.Series(["a","b"],["c","d"])
print(series)
```

```
c    a
d    b
dtype: object
```

```
# Datafrsme
'''A DataFrame is a two-dimensional, tabular data structure in pandas.
Think of it like an Excel spreadsheet or a SQL table — with rows and
columns.
It’s one of the core data structures in pandas, and it's used to store
and manipulate structured data.
'''
Data={"Name":['Raj','Harry','Mark'],
       "City":['pune','Mosco','Delhi'],
       "Age":[10,20,30]}
print(pd.DataFrame(Data))  # To directly print
```

```
    Name   City  Age
0    Raj   pune   10
1  Harry  Mosco   20
2   Mark  Delhi   30
```

```
# Convert to csv and download

df=pd.DataFrame(Data)
df.to_csv("NewFile.csv") # Pass file name we want to the the file

# Automatically saves the file named as NewFile.csv

# Remove index
# df1=pd.DataFrame(Data)
# df1.to_csv("NewFile.csv",index=False)

#for excel use -.to_excel and .xlsx for file

#for json use -.to_json and .json for file
```

## Head and Tail in pandas

```
sales.head(3) # It retruns default first 5 values if n is not passed
```

```
   ORDERNUMBER  QUANTITYORDERED  PRICEEACH  ORDERLINENUMBER
SALES  \
0        10107               30      95.70                2  2871.00
```

```
1        10121               34       81.35                5  2765.90

2        10134               41       94.74                2  3884.34


        ORDERDATE   STATUS  QTR_ID  MONTH_ID  YEAR_ID  ...  \
0  2/24/2003 0:00  Shipped       1         2     2003  ...
1   5/7/2003 0:00  Shipped       2         5     2003  ...
2   7/1/2003 0:00  Shipped       3         7     2003  ...

                    ADDRESSLINE1  ADDRESSLINE2   CITY STATE POSTALCODE
\
0        897 Long Airport Avenue           NaN    NYC    NY      10022

1             59 rue de l'Abbaye           NaN  Reims   NaN      51100

2  27 rue du Colonel Pierre Avia           NaN  Paris   NaN      75508


  COUNTRY TERRITORY CONTACTLASTNAME CONTACTFIRSTNAME DEALSIZE
0     USA       NaN              Yu             Kwai    Small
1  France      EMEA         Henriot             Paul    Small
2  France      EMEA        Da Cunha           Daniel   Medium

[3 rows x 25 columns]
```

```
sales.tail(3)  # It retruns default last 5 values if n is not passed
```

```
      ORDERNUMBER  QUANTITYORDERED  PRICEEACH  ORDERLINENUMBER
SALES  \
2820        10386               43     100.00                4
5417.57
2821        10397               34      62.24                1
2116.16
2822        10414               47      65.52                9
3079.44

           ORDERDATE    STATUS  QTR_ID  MONTH_ID  YEAR_ID  ...  \
2820   3/1/2005 0:00  Resolved       1         3     2005  ...
2821  3/28/2005 0:00   Shipped       1         3     2005  ...
2822   5/6/2005 0:00   On Hold       2         5     2005  ...

               ADDRESSLINE1  ADDRESSLINE2      CITY STATE POSTALCODE
COUNTRY  \
2820     C/ Moralzarzal, 86           NaN    Madrid   NaN      28034
Spain
2821  1 rue Alsace-Lorraine           NaN  Toulouse   NaN      31000
France
2822     8616 Spinnaker Dr.           NaN    Boston    MA      51003
USA
```

```
     TERRITORY CONTACTLASTNAME CONTACTFIRSTNAME DEALSIZE
2820      EMEA          Freyre            Diego   Medium
2821      EMEA          Roulet          Annette    Small
2822       NaN         Yoshido             Juri   Medium

[3 rows x 25 columns]
```

## Find the information about the Dataset-Use .info()

```
''' 1-find number of rows and columns
    2-column Name
    3-Data Type =int64,float64,object
       int64-numerical data.
       float64-numerical data decimals.
       object-categorical data.
    4-Non null counts
    5-Memory usage of Dataframe
    '''
sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ORDERNUMBER       2823 non-null   int64
 1   QUANTITYORDERED   2823 non-null   int64
 2   PRICEEACH         2823 non-null   float64
 3   ORDERLINENUMBER   2823 non-null   int64
 4   SALES             2823 non-null   float64
 5   ORDERDATE         2823 non-null   object
 6   STATUS            2823 non-null   object
 7   QTR_ID            2823 non-null   int64
 8   MONTH_ID          2823 non-null   int64
 9   YEAR_ID           2823 non-null   int64
 10  PRODUCTLINE       2823 non-null   object
 11  MSRP              2823 non-null   int64
 12  PRODUCTCODE       2823 non-null   object
 13  CUSTOMERNAME      2823 non-null   object
 14  PHONE             2823 non-null   object
 15  ADDRESSLINE1      2823 non-null   object
 16  ADDRESSLINE2      302 non-null    object
 17  CITY              2823 non-null   object
 18  STATE             1337 non-null   object
 19  POSTALCODE        2747 non-null   object
 20  COUNTRY           2823 non-null   object
 21  TERRITORY         1749 non-null   object
 22  CONTACTLASTNAME   2823 non-null   object
 23  CONTACTFIRSTNAME  2823 non-null   object
 24  DEALSIZE          2823 non-null   object
dtypes: float64(2), int64(7), object(16)
memory usage: 551.5+ KB
```

```
# Create an dataframe and check info of this dataframe

Data={"Name":['Raj','Harry','Mark'],
       "City":['pune','Mosco','Delhi'],
       "Age":[10,20,30]}
print(pd.DataFrame(Data))
df=pd.DataFrame(Data)
df.info()
```

```
    Name   City  Age
0    Raj   pune   10
1  Harry  Mosco   20
2   Mark  Delhi   30
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Name    3 non-null      object
 1   City    3 non-null      object
 2   Age     3 non-null      int64
dtypes: int64(1), object(2)
memory usage: 204.0+ bytes
```

## Describe method in pandas

```
'''The .describe() method in pandas is used to generate summary
statistics of your DataFrame or Series.

It provides a quick overview of the central tendency, spread, and
shape of a dataset’s numerical columns.'''


Data={"Name":['Raj','Harry','Mark'],
       "City":['pune','Mosco','Delhi'],
       "Age":[10,20,30]}
print(pd.DataFrame(Data))

df.describe()

'''
1]count-Number of columns

2]mean-The arithmetic average of the values in each column
```

```
3]std-std tells us how much the values in the column are spread out or
different from the average(mean)
small std-Values those are very close to the average(mean) ex-
[10,11,12,13]   {data is constient}
large std-Values are far away from Average ex-[10,50,100,200]
{variation is very large}

4]min-min is the Minimum value in the column

5]25%-The 25th percentile (also called the 1st quartile, or Q1) is the
value below which 25% of the data falls.
In your case:The Age values are: [10, 20, 30]
25% of the way through that sorted list (between 10 and 20) = 15.0
calculation-Q1 = 10 + (20 - 10) * 0.25 = 15.0

6]50%-It is also known as median
50% refers to the 50th percentile, also known as the median.
It is the middle value of the sorted data.
It splits the data into two equal halves — 50% of the values are below
it, and 50% are above it.
calculation-20 → This is the **50%** (or median)  {50%-20.0}

7]75%-75% is the 75th percentile, also called the third quartile (Q3).
It means 75% of the data lies below this value, and 25% is above it.
It gives you insight into the upper range of your data.
calculation-Q3 = 20 + (30 - 20) * 0.75 = 25.0


Percentile	Value	Meaning
25%	15.0	Lower quartile (Q1)
50%	20.0	Median (Q2)
75%	25.0	Upper quartile (Q3)

8]max-Display maximum value in column
```

```
    Name   City  Age
0    Raj   pune   10
1  Harry  Mosco   20
2   Mark  Delhi   30
```

```
        Age
count   3.0
mean   20.0
std    10.0
min    10.0
25%    15.0
50%    20.0
75%    25.0
max    30.0
```

## Shape and Column Attributes

```
# The .shape attribute returns the dimensions of a DataFrame or Series
as a tuple: (rows, columns)

Data={"Name":['Raj','Harry','Mark'],
       "City":['pune','Mosco','Delhi'],
       "Age":[10,20,30]}
df=pd.DataFrame(Data)
df.shape
```

```
(3, 3)
```

```
# for large data
sales.shape
```

```
(2823, 25)
```

```
# .columns -returns the name of columns

df.columns
```

```
Index(['Name', 'City', 'Age'], dtype='object')
```

```
# for Large data
sales.columns
```

```
Index(['ORDERNUMBER', 'QUANTITYORDERED', 'PRICEEACH',
'ORDERLINENUMBER',
       'SALES', 'ORDERDATE', 'STATUS', 'QTR_ID', 'MONTH_ID',
'YEAR_ID',
       'PRODUCTLINE', 'MSRP', 'PRODUCTCODE', 'CUSTOMERNAME', 'PHONE',
       'ADDRESSLINE1', 'ADDRESSLINE2', 'CITY', 'STATE', 'POSTALCODE',
       'COUNTRY', 'TERRITORY', 'CONTACTLASTNAME', 'CONTACTFIRSTNAME',
       'DEALSIZE'],
      dtype='object')
```

## Select Filter combine multiple columns

```
''' select specific columns (use square brackets)
returns 1] a series
        2]dataframe multiple columns of data'''

# Selecting one column
column=sales["ORDERNUMBER"]
print(column)

# Accessing multiple columns
columns=sales[["ORDERNUMBER","CITY","PHONE","DEALSIZE","CUSTOMERNAME"]
]
print(columns)
```

```
0       10107
1       10121
2       10134
3       10145
4       10159
        ...  
2818    10350
2819    10373
2820    10386
2821    10397
2822    10414
Name: ORDERNUMBER, Length: 2823, dtype: int64
      ORDERNUMBER           CITY              PHONE DEALSIZE  \
0           10107            NYC         2125557818    Small   
1           10121          Reims         26.47.1555    Small   
2           10134          Paris   +33 1 46 62 7555   Medium   
3           10145       Pasadena         6265557265   Medium   
4           10159  San Francisco         6505551386   Medium   
...           ...            ...                ...      ...   
2818        10350         Madrid     (91) 555 94 44    Small   
2819        10373           Oulu         981-443655   Medium   
2820        10386         Madrid     (91) 555 94 44   Medium   
2821        10397       Toulouse         61.77.6555    Small   
2822        10414         Boston         6175559555   Medium   

                  CUSTOMERNAME  
0            Land of Toys Inc.  
1           Reims Collectables  
2              Lyon Souveniers  
3            Toys4GrownUps.com  
4     Corporate Gift Ideas Co.  
...                        ...  
2818     Euro Shopping Channel  
2819   Oulu Toy Supplies, Inc.  
2820     Euro Shopping Channel  
2821              Alpha Cognac  
2822         Gifts4AllAges.com  

[2823 rows x 5 columns]

''' filtering rows-extract data on specific condition
    *] use boolean indexing
      '''
Data={"Name":['Raj','Harry','Mark'],
       "City":['pune','Mosco','Delhi'],
       "Age":[10,20,30]}

df=pd.DataFrame(Data)
# Based on the single condition
filtered_rows=df[df["Age"]>25]
```

```
print(filtered_rows)

# Based on the Multiple condition here we use and(&) operator we also
use or(||) operator
filtered_rows1=df[(df["Age"]>10)&(df["Age"]<30)]
print("\nAnother output is")
print(filtered_rows1)
```

```
   Name   City  Age
2  Mark  Delhi   30

Another output is
    Name   City  Age
1  Harry  Mosco   20
```

```
# \n is used to get extra space while printing
```

## Adding new column in dataset

```
# Adding new column

Data={"Name":['Raj','Harry','Mark'],
       "City":['pune','Mosco','Delhi'],
       "Age":[10,20,30]}

df=pd.DataFrame(Data)
df["salary"]=[10,20,30]
print(df)
```

```
    Name   City  Age  salary
0    Raj   pune   10      10
1  Harry  Mosco   20      20
2   Mark  Delhi   30      30
```

```
# operations

df["incremented salary"]=df["salary"]*10
print(df)
```

```
    Name   City  Age  salary  incremented salary
0    Raj   pune   10      10                 100
1  Harry  Mosco   20      20                 200
2   Mark  Delhi   30      30                 300
```

## Insert Method

```
# insert method is used to insert values (column) at specific position

# df.insert(location,"column_name",some_data)
```

```
Data={"Name":['Raj','Harry','Mark'],
       "City":['pune','Mosco','Delhi'],
       "Age":[10,20,30]}

df=pd.DataFrame(Data)
df.insert(0,"Ncolumn",[1,2,3])
print(df)
```

```
   Ncolumn   Name   City  Age
0        1    Raj   pune   10
1        2  Harry  Mosco   20
2        3   Mark  Delhi   30
```

## updating

### .loc

```
'''In Pandas, .loc[] is a label-based data selection method.
It is used to access a group of rows and columns by labels or a
boolean array.
df.loc[row_no, column_Name]
updating specific row-column value
'''

Data={"Name":['Raj','Harry','Mark'],
       "City":['pune','Mosco','Delhi'],
       "Age":[10,20,30]}

df=pd.DataFrame(Data)

df.loc[0,"Age"]=[18]
df
```

```
    Name   City  Age
0    Raj   pune   18
1  Harry  Mosco   20
2   Mark  Delhi   30
```

```
# Adding salary column and increasing salary by 5%

Data={"Name":['Raj','Harry','Mark'],
       "City":['pune','Mosco','Delhi'],
       "Age":[10,20,30]}

df=pd.DataFrame(Data)

df["salary"]=[1000,2000,3000]

print(df)
```

```
df["salary"]=df["salary"]*1.05
print(df)
```

```
    Name    City  Age  salary
0    Raj    pune   10    1000
1  Harry  Mosco   20    2000
2   Mark  Delhi   30    3000
    Name    City  Age  salary
0    Raj    pune   10  1050.0
1  Harry  Mosco   20  2100.0
2   Mark  Delhi   30  3150.0
```

## Deleting columns

```
# Remove full column
Data={"Name":['Raj','Harry','Mark'],
       "City":['pune','Mosco','Delhi'],
       "Age":[10,20,30]}

df=pd.DataFrame(Data)

df.drop(columns=["City"],inplace=True)
print(df)
```

```
    Name  Age
0    Raj   10
1  Harry   20
2   Mark   30
```

```
# Remove multiple columns
Data={"Name":['Raj','Harry','Mark'],
       "City":['pune','Mosco','Delhi'],
       "Age":[10,20,30]}

df=pd.DataFrame(Data)

df.drop(columns=["City","Age"],inplace=True)
print(df)
```

```
    Name
0    Raj
1  Harry
2   Mark
```

## Handling Missing Values

```
# NAN-NOT A NUMBER
# None-(for Object Data Type)
# isnull-(Return True-Missing_value or false-Missing_value is not
present

Data={"Name":['Raj','Harry','Mark',None,'Ram','sham',None,'vikas'],
       "City":
['pune','Mosco','Delhi',None,'Nagpur','kanpur','Mumbai',None],
       "Age":[10,20,30,40,None,None,35,55]}

df=pd.DataFrame(Data)
print(df)
```

```
    Name    City   Age
0    Raj    pune  10.0
1  Harry   Mosco  20.0
2   Mark   Delhi  30.0
3   None    None  40.0
4    Ram  Nagpur   NaN
5   sham  kanpur   NaN
6   None  Mumbai  35.0
7  vikas    None  55.0
```

## Remove missing values

```
# Remove unrequired missing values using .dropna()

Data={"Name":['Raj','Harry','Mark',None,'Ram','sham',None,'vikas'],
       "City":
['pune','Mosco','Delhi',None,'Nagpur','kanpur','Mumbai',None],
       "Age":[10,20,30,40,None,None,35,55]}

df=pd.DataFrame(Data)
print(df)

# Removing-

df.dropna(inplace=True)
print("\n Missing Values Table-")
print(df)
```

```
    Name    City   Age
0    Raj    pune  10.0
1  Harry   Mosco  20.0
2   Mark   Delhi  30.0
3   None    None  40.0
4    Ram  Nagpur   NaN
5   sham  kanpur   NaN
6   None  Mumbai  35.0
7  vikas    None  55.0

 Missing Values Table-
```

```
    Name    City   Age
0    Raj    pune  10.0
1  Harry  Mosco  20.0
2   Mark  Delhi  30.0
```

## Fill Missing values

```
# Use fillna to fill the value-df.fillna(value,inplace=True)

Data={"Name":['Raj','Harry','Mark',None,'Ram','sham',None,'vikas'],
       "City":
['pune','Mosco','Delhi',None,'Nagpur','kanpur','Mumbai',None],
       "Age":[10,20,30,40,None,None,35,55]}

df=pd.DataFrame(Data)
print(df)

# filling-

print("\nFilled values with 999-")

df.fillna(999,inplace=True)
print(df)
```

```
    Name     City   Age
0    Raj     pune  10.0
1  Harry    Mosco  20.0
2   Mark    Delhi  30.0
3   None     None  40.0
4    Ram   Nagpur   NaN
5   sham   kanpur   NaN
6   None   Mumbai  35.0
7  vikas     None  55.0

Filled values with 999-
    Name     City    Age
0    Raj     pune   10.0
1  Harry    Mosco   20.0
2   Mark    Delhi   30.0
3    999      999   40.0
4    Ram   Nagpur  999.0
5   sham   kanpur  999.0
6    999   Mumbai   35.0
7  vikas      999   55.0
```

```
# Filled with calculated value

Data={"Name":['Raj','Harry','Mark',None,'Ram','sham',None,'vikas'],
       "City":
['pune','Mosco','Delhi',None,'Nagpur','kanpur','Mumbai',None],
       "Age":[10,20,30,40,None,None,35,55]}

df=pd.DataFrame(Data)
print(df)
mean_age=df["Age"].mean()
df["Age"]=df["Age"].fillna(mean_age)
print("\n Updated Values")
print(df)
```

```
    Name    City   Age
0    Raj    pune  10.0
1  Harry   Mosco  20.0
2   Mark   Delhi  30.0
3   None    None  40.0
4    Ram  Nagpur   NaN
5   sham  kanpur   NaN
6   None  Mumbai  35.0
7  vikas    None  55.0

 Updated Values
    Name    City        Age
0    Raj    pune  10.000000
1  Harry   Mosco  20.000000
2   Mark   Delhi  30.000000
3   None    None  40.000000
4    Ram  Nagpur  31.666667
5   sham  kanpur  31.666667
6   None  Mumbai  35.000000
7  vikas    None  55.000000
```

```
#if we use this type-
Data = {
    "Name": ['Raj','Harry','Mark',None,'Ram','sham',None,'vikas'],
    "City":
['pune','Mosco','Delhi',None,'Nagpur','kanpur','Mumbai',None],
    "Age": [10, 20, 30, 40, None, None, 35, 55]
}

df = pd.DataFrame(Data)
print("Original DataFrame:")
print(df)

  # Calculates the mean of non-null Age values
df["Age"] = df["Age"].fillna(df["Age"].mean())  # Safely fills NaN in
Age

print("\nUpdated Values:")
print(df)
```

```
Original DataFrame:
    Name    City   Age
0    Raj    pune  10.0
1  Harry   Mosco  20.0
2   Mark   Delhi  30.0
3   None    None  40.0
4    Ram  Nagpur   NaN
5   sham  kanpur   NaN
6   None  Mumbai  35.0
7  vikas    None  55.0

Updated Values:
    Name    City        Age
0    Raj    pune  10.000000
1  Harry   Mosco  20.000000
2   Mark   Delhi  30.000000
3   None    None  40.000000
4    Ram  Nagpur  31.666667
5   sham  kanpur  31.666667
6   None  Mumbai  35.000000
7  vikas    None  55.000000
```

```
# Without using inplace
Data = {
    "Name": ['Raj','Harry','Mark',None,'Ram','sham',None,'vikas'],
    "City":
['pune','Mosco','Delhi',None,'Nagpur','kanpur','Mumbai',None],
    "Age": [10, 20, 30, 40, None, None, 35, 55]
}

df = pd.DataFrame(Data)
print("Original DataFrame:")
print(df)

mean_age = df["Age"].mean()  # Calculates the mean of non-null Age
values
df["Age"] = df["Age"].fillna(mean_age)  # Safely fills NaN in Age

print("\nUpdated Values:")
print(df)
```

```
Original DataFrame:
    Name    City   Age
0    Raj    pune  10.0
1  Harry   Mosco  20.0
2   Mark   Delhi  30.0
3   None    None  40.0
4    Ram  Nagpur   NaN
5   sham  kanpur   NaN
6   None  Mumbai  35.0
```

```
7  vikas    None  55.0

Updated Values:
    Name    City        Age
0    Raj    pune  10.000000
1  Harry   Mosco  20.000000
2   Mark   Delhi  30.000000
3   None    None  40.000000
4    Ram  Nagpur  31.666667
5   sham  kanpur  31.666667
6   None  Mumbai  35.000000
7  vikas    None  55.000000
```

## Interpolation

```
'''In Pandas, interpolation is a method used to fill missing values
(NaNs) in a DataFrame or Series by estimating them from existing data
points.

ex-[10,20,NAN,40,50] By series the NAN = 30 Estimated Value

Types-
1]Linear interpolation-
Linear interpolation fills missing values by connecting data points
with straight
lines – it assumes the change between known values is linear (i.e.,
follows a straight line).
ex-df.interpolate(method='linear')
df = pd.DataFrame({"value": [10, 20, np.nan, 40]})

   value
0   10.0
1   20.0
2   30.0   ← Linearly between 20 and 40
3   40.0

2]Polinomial interpolation-
Polynomial interpolation uses a polynomial function (curve) to
estimate missing values.
You can control the degree (or "order") of the curve.
ex-df.interpolate(method='polynomial', order=2)
df = pd.DataFrame({"value": [10, 15, np.nan, 50]})

       value
0  10.000000
1  15.000000
2  27.083333
3  50.000000
```

```
3]Time interpolate-Time interpolation is a method to fill missing
values based on time-based indexes
— typically used in time series data. It assumes that the x-axis is
time, and it interpolates missing values accordingly
ex-df.interpolate(method='time')

Before:
            value
2023-01-01   10.0
2023-01-02    NaN
2023-01-03    NaN
2023-01-04   40.0
2023-01-05   50.0


After (Time Interpolation):
            value
2023-01-01  10.000000
2023-01-02  20.000000
2023-01-03  30.000000
2023-01-04  40.000000
2023-01-05  50.000000
'''


# linear-
Data = {
    "Time":[1,None,3,4,None,6,7,None],
    "Values": [10, 20, 30, 40, None, None, 70, None]
}

df = pd.DataFrame(Data)
print("Before interpolation:")
print(df)

print("\n After interpolation-")
df["Values"]=df["Values"].interpolate(method="linear")
print(df)
```

```
Before interpolation:
   Time  Values
0   1.0    10.0
1   NaN    20.0
2   3.0    30.0
3   4.0    40.0
4   NaN     NaN
5   6.0     NaN
6   7.0    70.0
7   NaN     NaN
```

```
 After interpolation-
   Time  Values
0   1.0    10.0
1   NaN    20.0
2   3.0    30.0
3   4.0    40.0
4   NaN    50.0
5   6.0    60.0
6   7.0    70.0
7   NaN    70.0
```

## Sorting Data

```
# Sorting data in one column
data={"Alphabets":["Apple","Cat","Ball","Zoo","Kite"]}
df=pd.DataFrame(data)
print("Original Dataset")
print(df)
#sort-
print("\n Updated Dataset in Ascending order-")
df.sort_values(by="Alphabets",ascending=True,inplace=True)
print(df)
print("\n Updated Dataset in descending order-")
df.sort_values(by="Alphabets",ascending=False,inplace=True)
print(df)
```

```
Original Dataset
  Alphabets
0     Apple
1       Cat
2      Ball
3       Zoo
4      Kite

 Updated Dataset in Ascending order-
  Alphabets
0     Apple
2      Ball
1       Cat
4      Kite
3       Zoo

 Updated Dataset in descending order-
  Alphabets
3       Zoo
4      Kite
1       Cat
2      Ball
0     Apple
```

```
# Sorting in multiple columns
data={"Alphabets":["Apple","Cat","Ball","Zoo","Kite","Mat"],
     "Numbers":[1,4,6,5,3,2],
     "Age":[10,90,87,63,77,23]}
df=pd.DataFrame(data)
print(df)

# Sorting in multiple columns
df.sort_values(by=["Alphabets","Numbers","Age"],ascending=True,inplace=True)
```

```
  Alphabets  Numbers  Age
0     Apple        1   10
1       Cat        4   90
2      Ball        6   87
3       Zoo        5   63
4      Kite        3   77
5       Mat        2   23
```

## Aggregation

```
# Aggregation means performing a summary operation on data — like calculating the
# sum, mean, count, min, max, etc. — usually on groups or columns of data.

data={"Alphabets":["Apple","Cat","Ball","Zoo","Kite","Mat"],
     "Numbers":[1,4,6,5,3,2],
     "Age":[10,90,87,63,77,23]}
df=pd.DataFrame(data)

print("sum")
print(df.sum())

print("\nmean of Numbers ")
print(df["Numbers"].mean())

print("\nmedian of Numbers")
print(df["Numbers"].median())

print("\n mininum value")
print(df.min())

print("\n maximum value")
print(df.max())
```

```
sum
Alphabets    AppleCatBallZooKiteMat
Numbers                          21
Age                             350
```

```
dtype: object

mean of Numbers
3.5

median of Numbers
3.5

 mininum value
Alphabets    Apple
Numbers          1
Age             10
dtype: object

 maximum value
Alphabets    Zoo
Numbers        6
Age           90
dtype: object
```

## Grouping

```python
data={"Name":["Arun","vrun","karun","Narun","Marun"],
     "Age":[28,34,22,34,28],
      "Salary":[50000,60000,45000,52000,480000]}
df=pd.DataFrame(data)
grouped=df.groupby("Age")["Salary"].sum()
print(grouped)
```

```
Age
22     45000
28    530000
34    112000
Name: Salary, dtype: int64
```

```
'''working-
How it works:
df.groupby("Age"):

Groups the rows by unique Age values: 22, 28, and 34.

["Salary"]:

Focus only on the Salary column within each age group.

.sum():

Adds up the salaries for each age group.

Age   Salaries       Sum
```

```
22   [45000]            45000
28   [50000, 480000] 530000
34   [60000, 52000]  112000

output is-Age
22     45000
28    530000
34    112000
Name: Salary, dtype: int64'''

# Group multiple column
grouped=df.groupby(["Age","Name"])["Salary"].sum()
print(grouped)

Age  Name
22   karun     45000
28   Arun      50000
     Marun    480000
34   Narun     52000
     vrun      60000
Name: Salary, dtype: int64

'''
Working-

groupby(["Age", "Name"])
This tells pandas to:

Group the data first by Age, and then

Within each Age, group by Name.

This creates a MultiIndex group (also called hierarchical indexing).

["Salary"].sum()
Since each (Age, Name) pair is unique in this data, the .sum() just
returns each individual's salary.
But if there were duplicate (Age, Name) pairs, it would sum their
salaries.

Age  Name
22   karun     45000
28   Arun      50000
     Marun    480000
34   Narun     52000
     vrun      60000
Name: Salary, dtype: int64

This is a multi-indexed Series, with:

Level 0: Age
Level 1: Name

Values: Salary sums

When to use this?
Use multi-level grouping when you want to analyze subgroups within
groups.

For example:
→ "What’s the total salary per person grouped by their age?"
→ "Or, within each age group, who earns how much?"
```

## Merging

```
# Joining the Dataframe
df_customer=pd.DataFrame({'customer_id':[1,2,3],
                         'customer_name':['Aman','siddhesh','mukul']})
df_orders=pd.DataFrame({'customer_id':[1,2,4],
                       'oredr_Amount':[150,350,450]})

#type 1 -inner
#Keeps only the common customer_ids (1 and 2)
#Drops unmatched rows (3 and 4)
merge=pd.merge(df_customer,df_orders,on="customer_id",how="inner")
print("\ninner join-")
print(merge)

#type 2 -outer
# Keeps all rows from both DataFrames
#Fills unmatched values with NaN
print("\nouter join-")
merge=pd.merge(df_customer,df_orders,on="customer_id",how="outer")
print(merge)


#type 3 -left
#Keeps all customers, adds order data where available
#Rows in df_customer not found in df_orders → NaN
print("\nLeft join-")
merge=pd.merge(df_customer,df_orders,on="customer_id",how="left")
print(merge)



#type 4 -Right
#Keeps all orders, adds customer data if it exists
#Orders without matching customers → NaN
print("\nRight-")
merge=pd.merge(df_customer,df_orders,on="customer_id",how="right")
```

```
print(merge)

#type 5- cross



df1 = pd.DataFrame({'A': [1, 2]})
df2 = pd.DataFrame({'B': ['x', 'y', 'z']})

result = pd.merge(df1, df2, how='cross')
print(result)

'''A cross merge (also called a cartesian product) combines every row
from
one DataFrame with every row from another — just like a nested loop.
If df1 has m rows and df2 has n rows,
then the result will have m × n rows.

'''


inner join-
   customer_id customer_name  oredr_Amount
0            1          Aman           150
1            2      siddhesh           350

outer join-
   customer_id customer_name  oredr_Amount
0            1          Aman         150.0
1            2      siddhesh         350.0
2            3         mukul           NaN
3            4           NaN         450.0

Left join-
   customer_id customer_name  oredr_Amount
0            1          Aman         150.0
1            2      siddhesh         350.0
2            3         mukul           NaN

Right-
   customer_id customer_name  oredr_Amount
0            1          Aman           150
1            2      siddhesh           350
2            4           NaN           450
   A  B
0  1  x
1  1  y
2  1  z
3  2  x
4  2  y
5  2  z
```

# Concatinate

```
# combines dataframes vertically or horizontally

#region 1
df_region1=pd.DataFrame({"customer_id":[1,2],
                       "name":["Gopal","Raju"]})
#region 2
df_region2=pd.DataFrame({"customer_id":[3,4],
                       "name":["sham","sai"]})
print("vertically-")
df_concat=pd.concat([df_region1,df_region2],ignore_index=True)
#vertically
print(df_concat)

print("\nHorizontally-")
df_concat=pd.concat([df_region1,df_region2],axis=1,ignore_index=True)
#vertically
print(df_concat)
```

```
vertically-
   customer_id   name
0            1  Gopal
1            2   Raju
2            3   sham
3            4    sai

Horizontally-
   0      1  2     3
0  1  Gopal  3  sham
1  2   Raju  4   sai
```