> **Data Science:** is a branch of computer science where we study how to store, use and analyze data for deriving information from it.

Pandas

- Pandas is a Python library used for working with data sets.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.
- Pandas allows us to analyze big data and make conclusions based on statistical theories.
- Pandas can clean messy data sets, and make them readable and relevant.
- Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.
- Pandas is fast and it has high performance & productivity for users.

> **Why Use Pandas?**

- Fast and efficient for manipulating and analyzing data.
- Data from different file objects can be easily loaded.
- Flexible reshaping and pivoting of data sets
- Provides time-series functionality.

> **Uses of Pandas:**

- Data set cleaning, merging, and joining.
- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data.
- Columns can be inserted and deleted from DataFrame and higher dimensional objects.
- Powerful group by functionality for performing split-apply-combine operations on data sets.
- Data Visulaization

Getting Started

> **Installing Pandas**

- We need to install pandas library using the following **pip command:**

```
pip install pandas
```

> **Importing Pandas**

- After installing pandas on the system, we have to import it before any use, using the following statement:

```
import pandas as pd
```

Pandas Data Structures

Pandas provide following 2 data structures for manipulating data:

1. Series
2. DataFrame

> **1. Series:**

- Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.).
- Labels need not be unique but must be a hashable type.
- In the real world, a Pandas Series will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, or an Excel file.
- Pandas Series can be created from lists, dictionaries, and from scalar values, etc.

```
import pandas as pd
import numpy as np

arr = np.array([2,3,4,5,6,7,8])
sr = pd.Series(arr)
print(sr)
```

```
0    2
1    3
2    4
3    5
4    6
5    7
6    8
dtype: int64
```

2. DataFrame:

- Pandas DataFrame is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns).
- A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.
- Applications of DataFrame:
    - Work on Dataset
    - Analysis
    - Dropping
    - Processing
    - Cleaning
    - Join multiple data (CSV, excel format data)
    - Create excel, json, CSV, binary files.
    - Mathematical and Statistical Operations.
    - Use of Group by Function.

Columns: Also called as - Features, variables, field, dimensions.

Rows: Also called as - Records, values, observations, index.

Creating DataFrames:

```
# data = {'a':[1,2,3], 'b':[11,12,13], 'c':[21,22]}
# ValueError: All arrays must be of the same length
# df = pd.DataFrame(data)
# print(df)
```

1. Using a Dictionary with values as lists:

```
data = {'a':[1,2,3], 'b':[11,12,13], 'c':[21,22,23]}
```

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

```
   a   b   c
0  1  11  21
1  2  12  22
2  3  13  23
```

> ### 2. Can fill same value to all rows:

```
d = {'name':'Snehal','age':22,'subjects':['C','C++','HTML','Java','Python']}
df = pd.DataFrame(d)
print(df)
```

```
     name  age subjects
0  Snehal   22        C
1  Snehal   22      C++
2  Snehal   22     HTML
3  Snehal   22     Java
4  Snehal   22   Python
```

> ### 3. From dicstionary of numpy arrays:

```
a = np.array([1,2,3,4])
b = np.array(['A','B','C','D'])
c = np.array(['Kop','San','Sat','Pune'])

d = {'id':a, 'name':b, 'address':c}
df = pd.DataFrame(d)
print(df)
```

```
   id name address
0   1    A     Kop
1   2    B     San
2   3    C     Sat
3   4    D    Pune
```

> ### 4. Create DataFrame from list of lists:

```
lst = [['id','name','address'], [1,2,3,4], ['A','B','C','D'], ['Kop','San','Sat','Pune']]

df = pd.DataFrame(dict(zip(lst[0],lst[1:])))
print(df)
```

```
   id name address
0   1    A     Kop
1   2    B     San
2   3    C     Sat
3   4    D    Pune
```

```
print(type(df))

     <class 'pandas.core.frame.DataFrame'>
```

Importing and Exporting DataFrame

1. CSV File:

```
# Write to CSV file:

df.to_csv('df_csv.csv')
```

```
# Read from CSV file
ddf = pd.read_csv('df_csv.csv')
print(ddf)

        Unnamed: 0  id name address
     0           0   1    A     Kop
     1           1   2    B     San
     2           2   3    C     Sat
     3           3   4    D    Pune
```

2. Excel File:

```
# Write to Excel File
df.to_excel('df_xl.xlsx')
```

```
# Read from Excel File
dex = pd.read_excel('df_xl.xlsx')
print(dex)

        Unnamed: 0  id name address
     0           0   1    A     Kop
     1           1   2    B     San
     2           2   3    C     Sat
     3           3   4    D    Pune
```

3. JSON File:

```
# Write to json file:
df.to_json('df_json.json')
```

```
# Read from json file:
dj = pd.read_json('df_json.json')
print(dj)
```

```
   id name address
0   1    A     Kop
1   2    B     San
2   3    C     Sat
3   4    D    Pune
```

4. HTML File:

```
# Write to HTML File
df.to_html('df_html.html')
```

```
# Read from html file:
dh = pd.read_html('df_html.html')
print(dh)
```

```
[   Unnamed: 0  id name address
0           0   1    A     Kop
1           1   2    B     San
2           2   3    C     Sat
3           3   4    D    Pune]
```

DataFrame Functions

Check size of data frame:

```
df = pd.read_csv('Housing.csv')
print(df)
```

```
        price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
0    13300000  7420         4          2        3      yes        no       no
1    12250000  8960         4          4        4      yes        no       no
2    12250000  9960         3          2        2      yes        no      yes
3    12215000  7500         4          2        2      yes        no      yes
4    11410000  7420         4          1        2      yes       yes      yes
..        ...   ...       ...        ...      ...      ...       ...      ...
540   1820000  3000         2          1        1      yes        no      yes
541   1767150  2400         3          1        1       no        no       no
542   1750000  3620         2          1        1      yes        no       no
543   1750000  2910         3          1        1       no        no       no
544   1750000  3850         3          1        2      yes        no       no

    hotwaterheating airconditioning  parking prefarea furnishingstatus
0                no             yes        2      yes        furnished
1                no             yes        3       no        furnished
```

```
2                no              no        2      yes   semi-furnished
3                no             yes        3      yes        furnished
4                no             yes        2       no        furnished
..              ...             ...      ...      ...              ...
540              no              no        2       no      unfurnished
541              no              no        0       no   semi-furnished
542              no              no        0       no      unfurnished
543              no              no        0       no        furnished
544              no              no        0       no      unfurnished

[545 rows x 13 columns]
7085
RangeIndex(start=0, stop=545, step=1)
```

```
print(df.size)  # --> rows * cols
```

```
7085
```

```
print(df.index) # --> Range of index from Start to End
```

```
RangeIndex(start=0, stop=545, step=1)
```

Get Names of the Columns:

```
print(df.columns)  # --> Names of columns
```

```
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea', 'furnishingstatus'],
      dtype='object')
```

```
print(df.axes)  # --> Range and Names of the columns
```

```
[RangeIndex(start=0, stop=545, step=1), Index(['price', 'area', 'bedrooms', 'bathroom
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea', 'furnishingstatus'],
      dtype='object')]
```

df.info():

- Getting info of overall data frame.

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
```

```
 0   price             545 non-null    int64 
 1   area              545 non-null    int64 
 2   bedrooms          545 non-null    int64 
 3   bathrooms         545 non-null    int64 
 4   stories           545 non-null    int64 
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64 
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
None
```

df.describe():

- Return all the statistical functions values.
- For all the columns with numeric data type, present in the dataframe.
- Does not work for the string(object) data type.

```
print(df.describe())
```

```
              price          area    bedrooms   bathrooms     stories  \
count  5.450000e+02    545.000000  545.000000  545.000000  545.000000   
mean   4.766729e+06   5150.541284    2.965138    1.286239    1.805505   
std    1.870440e+06   2170.141023    0.738064    0.502470    0.867492   
min    1.750000e+06   1650.000000    1.000000    1.000000    1.000000   
25%    3.430000e+06   3600.000000    2.000000    1.000000    1.000000   
50%    4.340000e+06   4600.000000    3.000000    1.000000    2.000000   
75%    5.740000e+06   6360.000000    3.000000    2.000000    2.000000   
max    1.330000e+07  16200.000000    6.000000    4.000000    4.000000   

          parking  
count  545.000000  
mean     0.693578  
std      0.861586  
min      0.000000  
25%      0.000000  
50%      0.000000  
75%      1.000000  
max      3.000000  
```

df.head():

- Return first 5 rows, by default.
- Can also specify number of rows to fetch.

```
print(df.head())
```

```
      price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
0  13300000  7420         4          2        3      yes        no       no
1  12250000  8960         4          4        4      yes        no       no
2  12250000  9960         3          2        2      yes        no      yes
3  12215000  7500         4          2        2      yes        no      yes
4  11410000  7420         4          1        2      yes       yes      yes

  hotwaterheating airconditioning  parking prefarea furnishingstatus
0              no             yes        2      yes        furnished
1              no             yes        3       no        furnished
2              no              no        2      yes   semi-furnished
3              no             yes        3      yes        furnished
4              no             yes        2       no        furnished
```

```
print(df.head(10))
```

```
      price   area  bedrooms  bathrooms  stories mainroad guestroom basement  \
0  13300000   7420         4          2        3      yes        no       no
1  12250000   8960         4          4        4      yes        no       no
2  12250000   9960         3          2        2      yes        no      yes
3  12215000   7500         4          2        2      yes        no      yes
4  11410000   7420         4          1        2      yes       yes      yes
5  10850000   7500         3          3        1      yes        no      yes
6  10150000   8580         4          3        4      yes        no       no
7  10150000  16200         5          3        2      yes        no       no
8   9870000   8100         4          1        2      yes       yes      yes
9   9800000   5750         3          2        4      yes       yes       no

  hotwaterheating airconditioning  parking prefarea furnishingstatus
0              no             yes        2      yes        furnished
1              no             yes        3       no        furnished
2              no              no        2      yes   semi-furnished
3              no             yes        3      yes        furnished
4              no             yes        2       no        furnished
5              no             yes        2      yes   semi-furnished
6              no             yes        2      yes   semi-furnished
7              no              no        0       no      unfurnished
8              no             yes        2      yes        furnished
9              no             yes        1      yes      unfurnished
```

> **df.tail():**

- Return last 5 rows, by default.
- Can also specify number of rows to fetch.

```
print(df.tail())
```

```
       price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
540  1820000  3000         2          1        1      yes        no      yes
541  1767150  2400         3          1        1       no        no       no
542  1750000  3620         2          1        1      yes        no       no
543  1750000  2910         3          1        1       no        no       no
544  1750000  3850         3          1        2      yes        no       no
```

```
    hotwaterheating airconditioning  parking prefarea furnishingstatus
540              no              no        2       no      unfurnished
541              no              no        0       no   semi-furnished
542              no              no        0       no      unfurnished
543              no              no        0       no        furnished
544              no              no        0       no      unfurnished
```

```
print(df.tail(10))
```

```
       price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
535  2100000  3360         2          1        1      yes        no       no
536  1960000  3420         5          1        2       no        no       no
537  1890000  1700         3          1        2      yes        no       no
538  1890000  3649         2          1        1      yes        no       no
539  1855000  2990         2          1        1       no        no       no
540  1820000  3000         2          1        1      yes        no      yes
541  1767150  2400         3          1        1       no        no       no
542  1750000  3620         2          1        1      yes        no       no
543  1750000  2910         3          1        1       no        no       no
544  1750000  3850         3          1        2      yes        no       no

    hotwaterheating airconditioning  parking prefarea furnishingstatus
535              no              no        1       no      unfurnished
536              no              no        0       no      unfurnished
537              no              no        0       no      unfurnished
538              no              no        0       no      unfurnished
539              no              no        1       no      unfurnished
540              no              no        2       no      unfurnished
541              no              no        0       no   semi-furnished
542              no              no        0       no      unfurnished
543              no              no        0       no        furnished
544              no              no        0       no      unfurnished
```

> **isna():**

- Show null values.
- Return DataFrame which contains -
  - True - for NULL values
  - False - for NON-NULL values.

```
d = {'id':[1,2,3], 'name':['A','B','C'], 'age':[21,23,np.nan]}

df = pd.DataFrame(d)
df.isna()
```

	id	name	age
0	False	False	False
1	False	False	False
2	False	False	True

> **Transpose of the DataFrame:**

- Convert the row indices to column names and vice versa.

```
df.iloc[1:5,:5].transpose()
```

	1	2
id	2	3
name	B	C
age	23.0	NaN

```
df.iloc[1:5,:5].T
```

	1	2
id	2	3
name	B	C
age	23.0	NaN

Accessing DataFrame

> **Access by name of the Column:**

- Can access using the following 2 methods:

    1. `df.col_name`

    2. `df['col_name']`

```
df = pd.read_csv('Housing.csv')
```

```
print(df.price)
```

```
0        13300000
1        12250000
2        12250000
3        12215000
4        11410000
           ...
```

```
540     1820000
541     1767150
542     1750000
543     1750000
544     1750000
Name: price, Length: 545, dtype: int64
```

```
# Give column name as index in []
print(df.price[0])
```

```
13300000
```

```
print(df['price'])
```

```
0      13300000
1      12250000
2      12250000
3      12215000
4      11410000
         ...
540     1820000
541     1767150
542     1750000
543     1750000
544     1750000
Name: price, Length: 545, dtype: int64
```

```
# Give name of the column as index in []
print(df['price'][34])
```

```
8120000
```

> ### df.loc:

- It also access the actual values at the index and columns.
- Can get record at an index:

```
df.loc[index]

df.loc[start:end]

df.loc[start:end:step]
```

```
df.loc[10]
```

```
price                 9800000
area                    13200
bedrooms                    3
bathrooms                   1
```

```
stories                     2
mainroad                  yes
guestroom                  no
basement                  yes
hotwaterheating            no
airconditioning           yes
parking                     2
prefarea                  yes
furnishingstatus    furnished
Name: 10, dtype: object
```

```
df.loc[10:12]
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotw
10	9800000	13200	3	1	2	yes	no	yes	
11	9681000	6000	4	3	2	yes	yes	yes	
12	9310000	6550	4	2	2	yes	no	no	

```
df.loc[10:22:2]
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotw
10	9800000	13200	3	1	2	yes	no	yes	
12	9310000	6550	4	2	2	yes	no	no	
14	9240000	7800	3	2	2	yes	no	no	
16	9100000	6600	4	2	2	yes	yes	yes	
18	8890000	4600	3	2	2	yes	yes	no	
20	8750000	4320	3	1	2	yes	no	yes	
22	8645000	8050	3	1	1	yes	yes	yes	

- Can access records with a condition in **loc[]**:

```
df.loc[condition]
```

```
df.loc[df['bedrooms']==3]
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	ho
2	12250000	9960	3	2	2	yes	no	yes	
5	10850000	7500	3	3	1	yes	no	yes	
9	9800000	5750	3	2	4	yes	yes	no	
10	9800000	13200	3	1	2	yes	no	yes	
14	9240000	7800	3	2	2	yes	no	no	
...	...	...	...	...	...	...	...	...	
533	2100000	2400	3	1	2	yes	no	no	
537	1890000	1700	3	1	2	yes	no	no	
541	1767150	2400	3	1	1	no	no	no	
543	1750000	2910	3	1	1	no	no	no	
544	1750000	3850	3	1	2	yes	no	no	

300 rows × 13 columns

```
df.loc[df['bedrooms']<3]
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hot
61	7070000	8880	2	1	1	yes	no	no	
66	6930000	13200	2	1	1	yes	no	yes	
73	6685000	6600	2	2	4	yes	no	yes	
91	6419000	6750	2	1	1	yes	yes	yes	
114	6020000	6800	2	1	1	yes	yes	yes	
...	...	...	...	...	...	...	...	...	
535	2100000	3360	2	1	1	yes	no	no	
538	1890000	3649	2	1	1	yes	no	no	
539	1855000	2990	2	1	1	no	no	no	
540	1820000	3000	2	1	1	yes	no	yes	
542	1750000	3620	2	1	1	yes	no	no	

138 rows × 13 columns

Access Specific Value in the Data Frame:

```
df[col_name][index]
```

```
df['price'][23]
```

```
8645000
```

```
df['price'][23:45]
```

```
23    8645000
24    8575000
25    8540000
26    8463000
27    8400000
28    8400000
29    8400000
30    8400000
31    8400000
32    8295000
33    8190000
34    8120000
35    8080940
36    8043000
37    7980000
38    7962500
39    7910000
40    7875000
41    7840000
42    7700000
43    7700000
44    7560000
Name: price, dtype: int64
```

> **Access Data from Mulyiple Columns:**

- Use names of the columns in the form of a list:

```
df[[col_names_list]]
```

df[['price', 'area', 'bathrooms']]

> **iloc:**

- Pass axes numbers for index and columns.

```
df.iloc[1:5,]
# Return all columns of 1 to 4 rows
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwa
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	

```
df.iloc[1:5,:3]
# Return only firsy 3 columns of rows 1:4.
```

	price	area	bedrooms
1	12250000	8960	4
2	12250000	9960	3
3	12215000	7500	4
4	11410000	7420	4