Pandas is well suited for many different kinds of data:

↳ Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spread sheet.

↳ Ordered and unordered (not necessarily fixed-frequency) time series data.

↳ Arbitary matrix data (homogeneous or heterogeneous typed) with row and column labels.

↳ Any other form of observations / statistical data sets. The data actually need not be labelled at all to be placed into a pandas data structure.

The two primary data structures of Pandas are:

1. Series (1 DIMENSIONAL)

2. Data Frame (2-DIMENSIONAL).

These handle the vast majority of typical use cases in finance, statistics, social servic and many areas of engineering.

↳ For R users, DataFrame provides everything that R's data-frame provides and much more.

↳ Pandas is built on top of NumPy and is intended to integrate well within a specified computing environment with many other 3rd party libraries.

Here are just a few things that Pandas does well:

↳ Easy handling of missing data (represent-ed as NaN) in floating point as well as Non-floating Point data.

↳ Size mutability: Columns can be inserted

and deleted from DataFrame and higher dimensional objects.

↳ Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let Series, DataFrame etc automatically align the data of required/our computations.

↳ Powerful, flexible group by functionality to perform split-apply-combine operations on data sets for both aggregating and transforming data.

↳ Make it easy to convert ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects.

↳ Intelligent label-based slicing, fancy indexing, and subsetting of large data sets.

↳ Intuitive merging and joining data sets.

↳ Flexible re-shaping and pivoting of data sets.

↳ Hierarchial labeling of axes (possible to have multiple labels per tick)

↳ Robust IO tools for loading the data from flat files (csv and delimited), Excel files, databases, and saving/loading data from the ultrafast HDF5 format.

↳ Time series - specifically functionality: Data range generation and frequency conversion, moving window statistics, data shifting and logging.

PANDAS SERIES DATA STRUCTURE:

SYNTAX: pd.series(data, index)

index - Unique, Hashable, same length as data.

By default np.arange(n)

I/P - s = pd.series([1, 2, 3, 4]) - # Pd.1d list object to pandas series.

Print(s)

O/P -
```
0  1
1  2
2  3
3  4
dtype: int64
```

I/P - print(s[2])

O/P - 3

I/P - print(s[1:3])

O/P -
```
1  2
2  3
dtype: int64
```

I/P - s = pd.series(['x', 'y', 'z', 'abc'])

Print(s)

Strings are not allowed in Pandas. Int, float, object are allowed

```
O/P -   0    x
        1    y
        2    z
        3    abc
    dtype: object
```

```
I/P - s = pd.series(['Pavani', 'nadella'])
      Print(s)
```

```
O/P -   0    Pavani
        1    nadella
    dtype: object
```

CREATING SERIES FROM NUMPY ndarray:

```
I/P - data = np.array([10, 20, 30, 40, 50])
      s = pd.series(data)
      Print(s)
```

```
O/P -   0    10
        1    20
        2    30
        3    40
        4    50
    dtype: int32
```

dtype: int32

it takes int32 (4bits) to store.

So, 8 bytes (int64) is not needed for storing memory.

I/p - data = np.array([[1,2,3],[4,5,6]])

s = pd.series(data)

Print(s)

O/p - Error - Data should be 1 dimensional

DATA ACCESSING USING INDEX :

I/p - s = pd.series([1, 2, 3, 4, 5])

Print(s[2])

Print(s[1:])

Print(s[[1,4]])

O/p - 3

s[1:] -

1	2
2	3
3	4
4	5

dtype: int64

s[[1,4]] -

1	2
4	5

dtype: int64

List:

lst[1][2]

Array: arr[1, 2]

I/p - print(s[1:4])

O/p - 1 2
2 3
3 4
dtype: int64

I/p - s = pd.series([1, 2, 3, 4, 5], index = ['a', 'b', 'c', 'd', 'e'])

Print(s)

O/p - a 1
b 2
c 3
d 4
e 5
dtype: int64

I/p - print(s['a'])

O/p - 1

I/p - print(s['a':])

O/p - a 1
b 2
c 3
d 4
e 5
dtype: int64

Retrieve Multiple Elements

```
i/p - print(s[['a', 'b', 'e']])

o/p -  a   1
       b   2
       e   5

   dtype: int64
```

PANDAS DATA FRAME:

CREATING DATAFRAME USING DICTIONARY:

```
i/p - data = {'Name': ['Tom', 'Jack', 'steve', 'Ricky'],
              'Age': [28,34,29,42]}

      df = pd.DataFrame(data)

      df

o/p -      Name   Age
       0   Tom    28
       1   Jack   34
       2   steve  29
       3   Ricky  42
```

CREATING INDEXED DATAFRAME:

I/p - data = {'Name': ['Tom', 'Jack', 'steve', 'Ricky'],
'Age': [28, 34, 29, 42],
'Gender': ['M', 'F', 'F', 'M']}

df = pd.DataFrame(data, index=
['I-1', 'I-2', 'I-3', 'I-4'])

Print(df)

O/p -

	Name	Age	Gender
I-1	Tom	28	M
I-2	Jack	34	F
I-3	steve	29	F
I-4	Ricky	42	M

I/p - df = pd.DataFrame({
'col-1': ['Item-1', 'Item-2', 'Item-3', 'Item-
'col-2': ['Gold', 'Bronze', 'Gold', 'silver'],
'col-3': [1, 2, npnan, 4]})

Print(df)

o/p -

	col-1	col-2	col-3
0	Item-1	Gold	1.0
1	Item-2	Bronze	2.0
2	Item-3	Gold	NaN
3	Item-4	Silver	4.0

i/p - df.info() → Short summary of given data type of each column

o/p - <class 'pandas.core.frame.DataFrame>

RangeIndex: 4 entries, 0 to 3

Data columns (total 3 columns):

#	Column	Non-Null Count	Dtype
0	col-1	4 Non-null	object
1	col-2	4 Non-null	object
2	col-3	3 non-null	float64

dtypes: float64(1), object(2)

memory used: 224.0+ bytes

I/p - df.describe() → Numerical column description.

O/p -

	col-3
count	3.000
mean	2.333
std	1.5275
min	1.000
25%	1.5000
50%	2.000
75%	3.000
max	4.000

CREATING DATAFRAME USING TUPLE:

-Tuple shows one row

```
I/p - data = [('1/1/2019', 13, 6, 'Rain'),      - list of tuple.
              ('2/1/2019', 11, 7, 'Fog'),
              ('3/1/2019', 12, 8, 'sunny'),
              ('4/1/2019', 8, 5, 'Snow'),
              ('5/1/2019', 9, 6, 'Rain')]

df = pd.DataFrame(data, Columns=['Day',
                  'Temperature', 'windspeed', 'Event'])
                  # if 's' is not given to column, it shows error.
df
```

O/p -

	Day	Temperature	Wind Speed	Event
0	1/1/2019	13	6	Rain
1	2/1/2019	11	7	Fog
2	3/1/2019	12	8	Sunny
3	4/1/2019	8	5	Snow
4	5/1/2019	9	6	Rain

(173)

DATAFRAME BASIC FUNCTIONALITY:

CREATING DICTIONARY OF SERIES:

```
I/p - dict = {'Name': pd.series(['Tom', 'Jack', 'steve',
              'Ricky', 'Vin', 'James', 'Vin']),
       'Age': pd.series([25,26,25,35,23,33,31]),
       'Rating': pd.series([4.23,4.1,3.4,5,2.9,4.7,3.1])
      }
df = pd.DataFrame(dict)
df
```

O/p -

	Name	Age	Rating
0	Tom	25	4.23
1	Jack	26	4.10
2	Steve	25	3.40
3	Ricky	35	5.00
4	Vin	23	2.90
5	James	33	4.70
6	Vin	31	3.10

I/p - df.columns - Prints columns name in the table.

O/p - Index(['Name', 'Age', 'Rating'], dtype='object')

I/p - df.T - Transpose → Returns transpose of data frame.

O/p -

	0	1	2	3	4	5	6
Name	Tom	Jack	Steve	Ricky	Vin	James	Vin
Age	25	26	25	35	23	33	31
Rating	4.23	4.1	3.4	5	2.9	4.7	3.1

I/p - df.dtypes - dtypes → returns data type of each column

O/p -

Name	object
Age	int64
Rating	float64

dtype: object

I/p - df.shape - shape → returns tuple representing dimensionality.

O/p - (7, 3)

I/p - df.axes - Axes → returns list of row axis labels & colum axis labels

o/p - [RangeIndex(start=0, stop=7, step=1),

(15) Index(['Name', 'Age', 'Rating'],

dtype='object')]

i/p - df.values - values → returns actual data as ndarray

o/p - array([['Tom', 25, 4.23],

['Jack', 26, 4.1],

['Steve', 25, 3.4],

['Ricky', 35, 5],

['Vin', 23, 2.9],

['James', 33, 4.7],

['Vin', 31, 3.1]], dtype=object)

i/p - df.head() - head → by default head returns first 5 rows

o/p -

	Name	Age	Rating
0	Tom	25	4.23
1	Jack	26	4.10
2	Steve	25	3.40
3	Ricky	35	5.00
4	Vin	23	2.90

I/p - df.head(2)

O/p -

	Name	Age	Rating
0	Tom	25	4.23
1	Jack	26	4.10

I/p - df.tail() - tail → by default tail returns last 5 rows

O/p -

	Name	Age	Rating
2	Steve	25	3.4
3	Ricky	35	5.0
4	Vin	23	2.9
5	James	33	4.7
6	Vin	31	3.1

I/p - df.tail(2)

O/p -

	Name	Age	Rating
5	James	33	4.7
6	Vin	31	3.1

I/p - df.sum() - sum → returns the sum of values for requested axis by default axis=0.

O/p - Name Tom Jack Steve Ricky Vin James Vin

(17) Age 198

27.43

Rating

dtype: object

I/p - print(df.sum(1)) # axis = 1 → row wise sum

O/p -

0	29.23
1	30.10
2	28.40
3	40.00
4	25.90
5	37.70
6	34.10

dtype: float64

I/p - print(df.mean())

O/p -

Age	28.285714
Rating	3.918571

dtype: float64

I/p - print(df.std())

O/p -

Age	4.644505
Rating	0.804828

dtype : float64

I/p - Print(df.describe()) - # describe()
- Summarizing the data

O/p -

	Age	Rating
count	7.000	7.000
mean	28.285	3.918
std	4.644	0.804
min	23.000	2.900
25%	25.000	3.250
50%	26.000	4.100
75%	32.000	4.465
max	35.000	5.000

I/p - print(df.describe(include=['object']))
↳ Includes object, number, all

O/p -

	Name
count	7
unique	6
top	Vin
freq	2

I/p - print(df.describe(include=['number']))

O/p -

	Age	Rating
count	7.000	7.000
mean	28.285	3.918
std	4.644	0.804
min	23.000	2.900
25%	25.000	3.250
50%	26.000	4.100
75%	32.000	4.465
max	35.000	5.000

179

I/p - print(df.describe(include = 'all'))

↳ Don't pass 'all' as a list.

	Name	Age	Rating
count	7	7.000	7.000
unique	6	NaN	NaN
top	Vin	NaN	NaN
freq	2	NaN	NaN
mean	NaN	28.285	3.918
std	NaN	4.644	0.804
min	NaN	23.000	2.900
25%	NaN	25.000	3.250
50%	NaN	26.000	4.100
75%	NaN	32.000	4.465
max	NaN	35.000	5.000