

Contents

1. Pandas.....	2
2. Main usage	2
3. Creator.....	2
4. Data Analysis	2
5. Key Features	3
6. Open source.....	3
7. Environment setup/pandas installation.....	3
8. pip command in python	3
9. Check installed python library.....	4
10. Pandas Data Structures.....	4
10.1. Series.....	5
10.2. DataFrame.....	6
10.3. Panel.....	7
11. Importing pandas	9
12. Aliasing or renaming pandas.....	9
12.1. Why aliasing Boss?.....	9
12.2. Famous Alias name to pandas	9
12.3. Specialty of alias name as pd	10

1. PANDAS - INTRODUCTION

1. Pandas

- ✓ Pandas is an open-source Python Library.
- ✓ Pandas have powerful data structures.

2. Main usage

- ✓ Main usage of pandas is analysing the data.
- ✓ By using pandas we can do below things easily,
 - Data loading
 - Data preparation
 - Data manipulation
 - Data analysis & etc

3. Creator

- ✓ Pandas created by Wes McKinney in the year of 2008

4. Data Analysis

- ✓ It is a process of cleaning, transforming, and modeling data to discover useful information for business decision-making.
- ✓ The purpose of Data Analysis is to extract useful information from data and taking the decision based upon the data analysis.
- ✓ Data Analysis became very easy after pandas was introduced to this world.

5. Key Features

- ✓ It's very easy to load different file formats of data.
- ✓ Handling missing data.
- ✓ Reshaping the data.
- ✓ Grouping the data
- ✓ Indexing and label-based slicing.
- ✓ Easily insert and delete columns from data
- ✓ Operations like aggregation and transformations.
- ✓ High performance merging and joining of data and many of others boss.

6. Open source

- ✓ Pandas is an open source means it's free.

7. Environment setup/pandas installation

- ✓ Open command prompt and execute below command

```
pip install pandas
```

8. pip command in python

- ✓ pip stands for **python installer package**
- ✓ Pip is a package management system.
- ✓ It is used to install and manage software packages.
 - pip install package_name
 - pip install pandas

9. Check installed python library.

- ✓ We can check installed python library by using below command.

```
pip show pandas
```

10. Pandas Data Structures

- ✓ One of the important features in pandas is data structures.
- ✓ There are mainly 3 types of data structures in pandas

Data Structure	Dimentionality	
1. Series	1D	Column
2. DataFrame	2D	Rows & Columns
3. Panel	3D	Group of Dataframes

10.1. Series

- ✓ A Series is similar to a single column of data.

Series

apples	
0	3
1	2
2	0
3	1

Series

oranges	
0	0
1	3
2	7
3	2

10.2. DataFrame

- ✓ A DataFrame is similar to a table which contains rows and columns of data.

DataFrame

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

Organization Salesperson Name Sales

0	Google	Sam	200
1	Google	Charlie	120
2	Salesforce	Ralph	125
3	Salesforce	Emily	250
4	Adobe	Rosalynn	150
5	Adobe	Chelsea	500

10.3. Panel

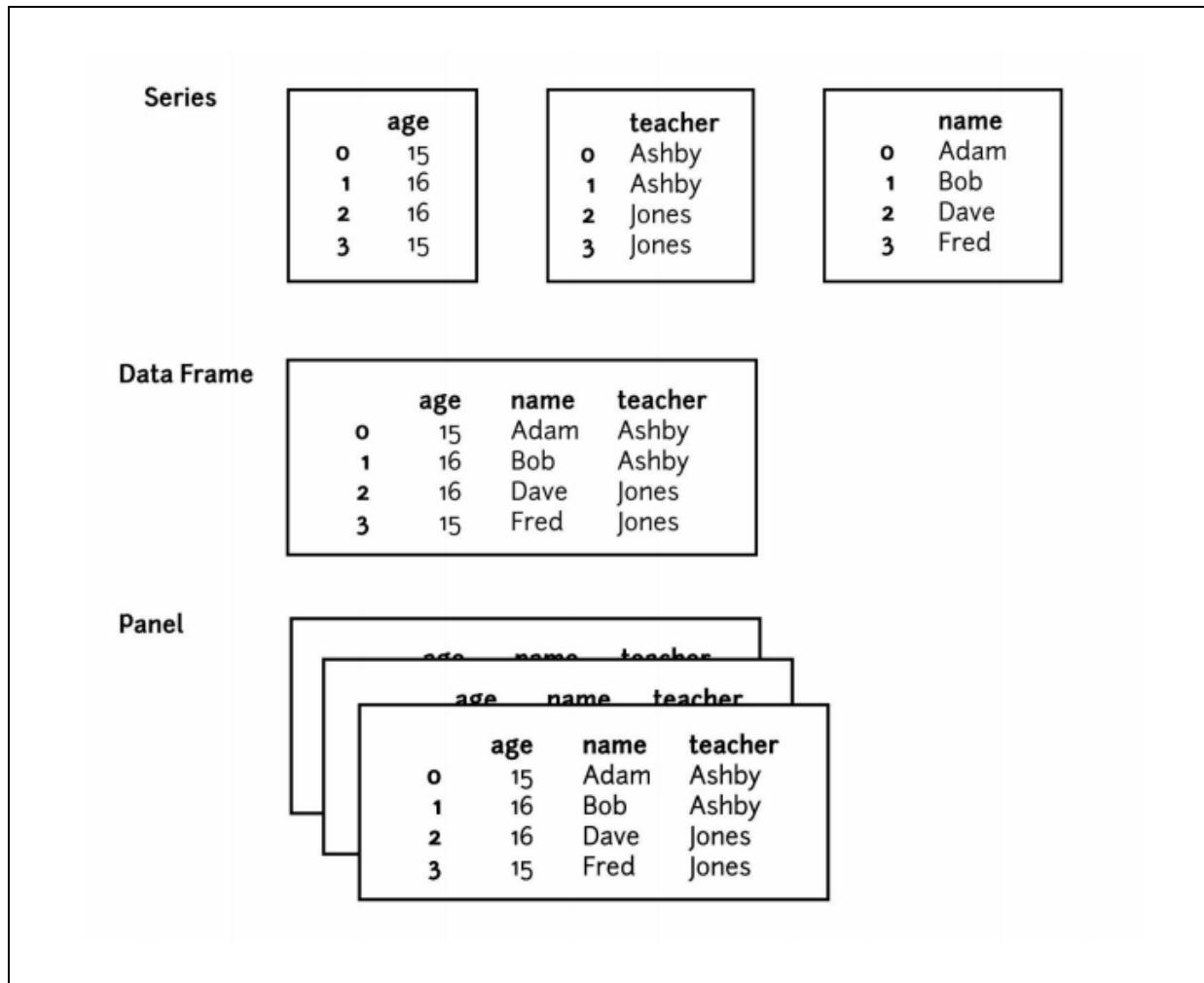
- ✓ A Panel is a group of DataFrames of data.

The diagram illustrates a Panel structure by showing three nested DataFrames stacked vertically. Each DataFrame has columns labeled 'age', 'name', and 'teacher'. The innermost DataFrame contains 4 rows, indexed 0 through 3. The middle DataFrame contains 3 rows, indexed 0 through 2. The outermost DataFrame contains 2 rows, indexed 0 and 1. The 'age' column values are 15, 16, 16, and 15 respectively for each row index. The 'name' column values are Adam, Bob, Dave, and Fred. The 'teacher' column values are Ashby, Ashby, Jones, and Jones.

	age	name	teacher
0	15	Adam	Ashby
1	16	Bob	Ashby
2	16	Dave	Jones
3	15	Fred	Jones

Mostly used

- ✓ The most widely used data structures are the Series and the DataFrame.
- ✓ Series and DataFrames deal with array data and tabular data respectively



Relationship...

- ✓ Series having single column
- ✓ DataFrame can have multiple Series
- ✓ Panel has multiple DataFrames

11. Importing pandas

- ✓ We can import pandas package by using `import` keyword

<p>Program Name</p> <p>demo1.py</p>	<pre>import pandas print("pandas imported successfully")</pre>
Output	pandas imported successfully

12. Aliasing or renaming pandas

- ✓ We can alias or rename pandas name.
- ✓ As per python syntax we need to use `as` keyword to alias pandas name

12.1. Why aliasing Boss?

- ✓ Let me give one best example.
- ✓ My name is: **Kasagani N Daniel**, you can call by Daniel instead of full name for simplicity

12.2. Famous Alias name to pandas

- ✓ We can give alias name to pandas.
- ✓ Note this name can be any name but the famous alias name is **pd**

<p>Program Name</p> <p>aliasing pandas name as pd demo2.py</p> <pre>import pandas as pd print("pandas imported successfully") print("pandas renamed to pd") print("Now onwards we can use pd instead of pandas name")</pre> <p>Output</p> <p>pandas imported successfully pandas renamed to pd Now onwards we can use pd instead of pandas name</p>
--

12.3. Specialty of alias name as pd

- ✓ Here in our example we have given alias name as pd.
- ✓ We can provide any name as alias name.
- ✓ Real time developers habituated to give alias name as pd.
- ✓ So, its **universal** alias name

Contents

1. Series.....	2
2. Creating Series.....	2
2.1. Empty Series object.....	3
2.2. Creating Series by using list	4
2.3. Create a Series from array	8
2.4. Creating Series by column from DataFrame.....	12
3. Index in Series	13
3.1. What is index?.....	13
3.2. Index default value.....	13
4. Accessing values in Series	18

2. PANDAS – SERIES

1. Series

- ✓ The Pandas Series is a one-dimensional labeled array.
- ✓ Series can store same and different types of the data.
- ✓ Series stores data in sequential order.
- ✓ Series is like a, one column information.

Series is a pre-defined class

- ✓ Technically speaking **Series** is a pre-defined class in pandas library.

2. Creating Series

- ✓ We can create Series in different ways,
 - Empty series
 - By using list
 - By using an array
 - By accessing single column from DataFrame.

2.1. Empty Series object

- ✓ We need to create object to Series pre-defined class.

Program Name creating empty Series object
demo1.py

```
import pandas as pd

s = pd.Series()
print(s)
print(type(s))
```

Output
Series([], dtype: float64)
<class 'pandas.core.series.Series'>

2.2. Creating Series by using list

- ✓ We can create Series by using list.

Program Name creating Series object using list
demo2.py

```
import pandas as pd

m = [56, 45, 35, 41, 44, 60]
s = pd.Series(m)
print(s)
```

Output

```
0    56
1    45
2    35
3    41
4    44
5    60
dtype: int64
```

Program Name creating Series object using list, assigning a name
demo3.py

```
import pandas as pd

m = [56, 45, 35, 41, 44, 60]
s = pd.Series(m, name = "marks")
print(s)
```

Output

```
0    56
1    45
2    35
3    41
4    44
5    60
Name: marks, dtype: int64
```

Program Name creating Series object using list, assigning a name
demo4.py

```
import pandas as pd

n = ["Prasad", "Daniel", "Samuel", "Jeswanth"]
s = pd.Series(n, name = "students")
print(s)
```

Output

```
0      Prasad
1      Daniel
2      Samuel
3      Jeswanth
Name: students, dtype: object
```

Program Name creating Series object by using range and list.
demo5.py

```
import pandas as pd

r = range(100)
a = list(r)
s = pd.Series(a)

print(s)
```

Output

```
0      0
1      1
2      2
3      3
4      4
       ..
95     95
96     96
97     97
98     98
99     99
Length: 100, dtype: int64
```

2.3. Create a Series from array

- ✓ We can pass array as an argument to the Series.
- ✓ By default, index is assigned to every element.

Program creating ndarray
Name demo6.py

```
import pandas as pd
import numpy as np

values = [10, 20, 30, 40]
data = np.array(values)

print(data)
print(type(data))
```

Output

```
[10 20 30 40]
<class 'numpy.ndarray'>
```

Program Name creating ndarray and passing argument to the Series
demo7.py

```
import pandas as pd
import numpy as np

values = [10, 20, 30, 40]
data = np.array(values)

s = pd.Series(data)
print(s)
```

Output

```
0    10
1    20
2    30
3    40
dtype: int32
```

Program Name Creating Series using ndarray
demo8.py

```
import pandas as pd
import numpy as np

values = ['a', 'b', 'c', 'd']
data = np.array(values)

s = pd.Series(data)
print(s)
```

Output

```
0    a
1    b
2    c
3    d
dtype: object
```

Program Name Creating Series using ndarray
demo9.py

```
import pandas as pd
import numpy as np

values = ['Vinay', 'Daniel', 'Veeru', 'Arjun']
data = np.array(values)
s = pd.Series(data)
print(s)
```

Output

```
0      Vinay
1      Daniel
2      Veeru
3      Arjun
dtype: object
```

2.4. Creating Series by column from DataFrame

- ✓ If we select single column from DataFrame then it returns Series object.
- ✓ This point we will learn during DataFrame chapter, thanks for understanding.

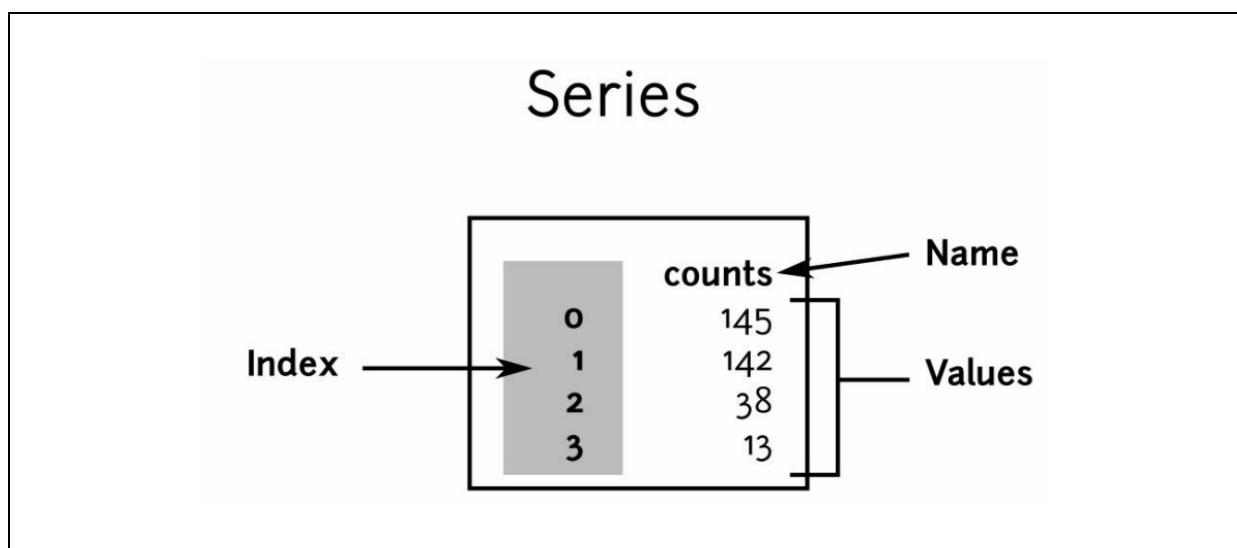
3. Index in Series

3.1. What is index?

- ✓ Index means, the position of value where it stores.
- ✓ The index is a core feature in pandas.
- ✓ By default, index is assigned to every value.
- ✓ From the output, the left most column is the index column.
- ✓ The generic name for an index is an axis.

3.2. Index default value

- ✓ The default values for an index are integers.
- ✓ The index starts from 0, 1, 2, 3, etc.
- ✓ Based on requirement we can customise this index
- ✓ These are called as axis labels



Program Name Creating Series
demo10.py

```
import pandas as pd

v = [145, 142, 38, 13]
s = pd.Series(v)
print(s)
```

Output

```
0    145
1    142
2     38
3     13
dtype: int64
```

Program Name Creating Series object and giving name
demo11.py

```
import pandas as pd

v = [145, 142, 38, 13]
s = pd.Series(v, name = 'counts')
print(s)
```

Output

```
0    145
1    142
2     38
3     13
Name: counts, dtype: int64
```

Program Name Creating Series object and giving name and index
demo12.py

```
import pandas as pd

v = [145, 142, 38, 13]
i = [10, 20, 30, 40]

s = pd.Series(v, name = 'counts', index = i )
print(s)
```

Output

```
10    145
20    142
30     38
40     13
Name: counts, dtype: int64
```

Program Name Creating Series object and giving name and index
demo13.py

```
import pandas as pd

prices = [1000, 2000, 3000, 4000]
products = ["Nokia", "Samsung", "Oppo", "iPhone 6"]

s = pd.Series(prices, name = 'mobiles', index = products )
print(s)
```

Output

```
Nokia      1000
Samsung    2000
Oppo       3000
iPhone 6   4000
Name: mobiles, dtype: int64
```

4. Accessing values in Series

- ✓ We can access series values by using index

Program Name Creating Series and accessing values
demo14.py

```
import pandas as pd

v = [56, 45, 35, 41, 44, 60]
s = pd.Series(v, name = "marks")

print(s)
print()
print(s[0])
print(s[1])
```

Output

```
0    56
1    45
2    35
3    41
4    44
5    60
Name: marks, dtype: int64

56
45
```

Program Name Creating Series and accessing values
demo15.py

```
import pandas as pd

prices = [1000, 2000, 3000, 4000]
products = ["Nokia", "Samsung", "Oppo", "iPhone 6"]

s = pd.Series(prices, name = 'mobiles', index = products )

print(s)
print()
print(s["Nokia"])
print(s["Samsung"])
```

Output

```
Nokia      1000
Samsung    2000
Oppo       3000
iPhone 6   4000
Name: mobiles, dtype: int64

1000
2000
```

Contents

1. NaN	2
2. None, NaN, nan and null.....	2
3. While loading any file.....	3

3. PANDAS – NaN Value

1. NaN Value

- ✓ The full form of NaN is **Not a Number**
- ✓ The purpose of NaN is, to represent the missing values in data.
- ✓ The data type of NaN is **float**.
- ✓ During data analysis we need to handle these NaN values.
- ✓ We will learn **more** about NaN in **12th** chapter which is handling missing values chapter.

2. None, NaN, nan and null

- ✓ None, NaN, nan, and null are synonyms.
- ✓ These all are referring to empty or missing data found in a Series, DataFrame.

Program Name Creating Series with NaN values
demo1.py

```
import pandas as pd
import numpy as np

marks = [36, 70, np.nan, 60]
s = pd.Series(marks)
print(s)
```

Output

```
0    36.0
1    70.0
2    NaN
3    60.0
dtype: float64
```

Program Name Creating Series with NaN values
demo2.py

```
import pandas as pd
import numpy as np

names = ["Daniel", "Ranjan", "Swathi", np.nan]
s = pd.Series(names)
print(s)
```

Output

```
0      Daniel
1      Ranjan
2      Swathi
3      NaN
dtype: object
```

3. While loading any file

- ✓ While loading any csv/excel/json file, if columns having missing values, then pandas consider those values as **NaN**.
- ✓ We will learn same point during loading files.

Contents

1. Series Attributes.....	2
1.1. values	2
1.2. index.....	4
1.3. dtypes.....	7
1.4. size	10

4. PANDAS – SERIES - ATTRIBUTES

1. Series Attributes

- ✓ Series is a predefined class.
- ✓ Series having different attributes.
- ✓ These attributes return information about the object.

1.1. values

- ✓ values is predefined attribute in Series class.
- ✓ We can access values attribute by using series object.
- ✓ This attribute returns the group of values as an array.

Program Name Accessing values attribute.
demo1.py

```
import pandas as pd

marks = [56, 45, 35, 41, 44, 60]
s = pd.Series(marks)

print(s)
print(s.values)
```

Output

```
0    56
1    45
2    35
3    41
4    44
5    60
dtype: int64
[56 45 35 41 44 60]
```

Program Name Accessing values attribute.
demo2.py

```
import pandas as pd

names = ["Bharath", "Daniel", "Nireekshan"]
s = pd.Series(names)

print(s)
print(s.values)
```

Output

```
0      Bharath
1      Daniel
2    Nireekshan
dtype: object
['Bharath' 'Daniel' 'Nireekshan']
```

1.2. index

- ✓ index is predefined attribute in Series class.
- ✓ We can access index attribute by using series object.
- ✓ This attribute returns the index range like, RangeIndex(start=0, stop=6, step=1)

Program Name Accessing index attribute.
demo3.py

```
import pandas as pd

marks = [56, 45, 35, 41, 44, 60]
s = pd.Series(marks)

print(s)
print(s.index)
```

Output

```
0    56
1    45
2    35
3    41
4    44
5    60
dtype: int64
RangeIndex(start=0, stop=6, step=1)
```

Program Name Accessing index attribute.
demo4.py

```
import pandas as pd

marks = [56, 45, 35, 41, 44, 60]
i = [11, 12, 13, 14, 15, 16]
s = pd.Series(marks, index = i)

print(s)
print(s.index)
```

Output

```
11    56
12    45
13    35
14    41
15    44
16    60
dtype: int64
Int64Index([11, 12, 13, 14, 15, 16], dtype='int64')
```

Program Name Accessing index attribute.
demo5.py

```
import pandas as pd

marks = [56, 45, 35, 41, 44, 60]
i = ['a', 'b', 'c', 'd', 'e', 'f']
s = pd.Series(marks, index = i)

print(s)
print(s.index)
```

Output

```
a    56
b    45
c    35
d    41
e    44
f    60
dtype: int64
Index(['a', 'b', 'c', 'd', 'e', 'f'], dtype='object')
```

1.3. dtypes

- ✓ dtypes is predefined attribute in Series class.
- ✓ We can access dtypes attribute by using series object.
- ✓ This attribute returns data type of series column.

Program Name Accessing dtypes attribute.
demo6.py

```
import pandas as pd

marks = [56, 45, 35, 41, 44, 60]
s = pd.Series(marks)

print(s)
print(s.dtype)
```

Output

```
0    56
1    45
2    35
3    41
4    44
5    60
dtype: int64
int64
```

Data Science – Pandas – Series - Attributes

Program Name Accessing dtypes attribute.
demo7.py

```
import pandas as pd

salaries = [1000.23, 1100.45, 8889.7, 999.87]
s = pd.Series(salaries)

print(s)
print(s.dtypes)
```

Output

```
0    1000.23
1    1100.45
2    8889.70
3    999.87
dtype: float64
float64
```

Program Name Accessing dtypes attribute.
demo8.py

```
import pandas as pd

names = ["Daniel", "Abhinav", "Dinesh", "Akshitha"]
s = pd.Series(names)

print(s)
print(s.dtypes)
```

Output

```
0      Daniel
1      Abhinav
2      Dinesh
3      Akshitha
dtype: object
object
```

1.4. size

- ✓ size is predefined attribute in Series class.
- ✓ We can access size attribute by using series object.
- ✓ This attribute returns number of values in series.

Program Name Accessing size attribute.
demo9.py

```
import pandas as pd

marks = [56, 45, 35, 41, 44, 60]
s = pd.Series(marks)

print(s)
print(s.size)
```

Output

```
0    56
1    45
2    35
3    41
4    44
5    60
dtype: int64
6
```

Contents

1. Series Methods.....	2
1.1. head()	2
1.2. tail().....	4
1.3. sum()	5
1.4. count().....	7
1.5. mean()	8
1.6. describe()	9
1.7. unique().....	11
1.8. nunique().....	12

5. PANDAS – SERIES – METHODS

1. Series Methods

- ✓ Series is a predefined class.
- ✓ Series class having different methods
- ✓ These methods perform operations on Series of values.

1.1. head()

- ✓ head() is predefined method in Series class.
- ✓ We can access head() method by using series object.
- ✓ This method returns first five values from the series

Program Name Accessing first five values from series
demo1.py

```
import pandas as pd

marks = [56, 45, 35, 41, 60, 57, 56, 56]
s = pd.Series(marks)

print(s)
print(s.head())
```

Output

Data Science – Pandas - Series- Methods

```
0    56
1    45
2    35
3    41
4    60
5    57
6    56
7    56
dtype: int64
0    56
1    45
2    35
3    41
4    60
dtype: int64
```

1.2. tail()

- ✓ tail() is predefined method in Series class.
- ✓ We can access tail() method by using series object.
- ✓ This method returns last five values from the series

Program Name Accessing last five values from series
demo2.py

```
import pandas as pd

marks = [56, 45, 35, 41, 60, 57, 56, 56]
s = pd.Series(marks)

print(s)
print(s.tail())
```

Output

```
0    56
1    45
2    35
3    41
4    60
5    57
6    56
7    56
dtype: int64
3    41
4    60
5    57
6    56
7    56
dtype: int64
```

1.3. sum()

- ✓ sum() is predefined method in Series class.
- ✓ We can access sum() method by using series object.
- ✓ This method returns the sum of all values.

Program Name Get the sum of series of values.
demo3.py

```
import pandas as pd

sales = [56, 45, 35, 41, 44, 60]
s = pd.Series(sales)

print(s)
print(s.sum())
```

Output

```
0      56
1      45
2      35
3      41
4      44
5      60
dtype: int64
281
```

Program Name Get the sum of series of values.
demo4.py

```
import pandas as pd
import numpy as np

marks = [56, 45, 35, 41, np.nan, 60, np.nan]
s = pd.Series(marks)

print(s)
print(s.sum())
```

Output

```
0    56.0
1    45.0
2    35.0
3    41.0
4    NaN
5    60.0
6    NaN
dtype: float64
237.0
```

1.4. count()

- ✓ count() is predefined method in Series class.
- ✓ We can access count() method by using series object.
- ✓ This method returns the number of non-NAN/null values.

Program Name Get the number of non-NaN values
demo5.py

```
import pandas as pd
import numpy as np

marks = [56, 45, 35, 41, np.nan, 60, np.nan]
s = pd.Series(marks)

print(s)
print(s.count())
```

Output

```
0    56.0
1    45.0
2    35.0
3    41.0
4    NaN
5    60.0
6    NaN
dtype: float64
5
```

1.5. mean()

- ✓ mean() is predefined method in Series class.
- ✓ We can access mean() method by using series object.
- ✓ This method returns the mean of series of values.

Program Name Get the mean of series values
demo6.py

```
import pandas as pd

sales = [10, 20, 30, 40, 50]
s = pd.Series(sales)

print(s)
print(s.mean())
```

Output

```
0    10
1    20
2    30
3    40
4    50
dtype: int64
30.0
```

1.6. describe()

- ✓ describe() is predefined method in Series class.
- ✓ We can access describe() method by using series object.
- ✓ This method returns the below values,
 - count
 - mean
 - std
 - min
 - 25%
 - 50%
 - 75%
 - max

Program Name describe() method
demo7.py

```
import pandas as pd

sales = [56, 45, 35, 41, 60]
s = pd.Series(sales)

print(s)
print(s.describe())
```

Output

```
0      56
1      45
2      35
3      41
4      60
dtype: int64
count    5.000000
mean     47.400000
std      10.406729
min     35.000000
25%     41.000000
50%     45.000000
75%     56.000000
max     60.000000
dtype: float64
```

1.7. unique()

- ✓ unique() is predefined method in Series class.
- ✓ We can access unique() method by using series object.
- ✓ This method returns unique values from the series.

Program Name Get unique values from the series
demo8.py

```
import pandas as pd

marks = [56, 45, 35, 41, 60, 57, 56, 56]
s = pd.Series(marks)

print(s)
print(s.unique())
```

Output

```
0    56
1    45
2    35
3    41
4    60
5    57
6    56
7    56
dtype: int64
[56 45 35 41 60 57]
```

1.8. nunique()

- ✓ nunique() is predefined method in Series class.
- ✓ We can access nunique() method by using series object.
- ✓ This method returns number of unique values from the series.

Program Name Get the number of unique values from the series
demo9.py

```
import pandas as pd

marks = [56, 45, 35, 41, 60, 57, 56, 56]
s = pd.Series(marks)

print(s)
print(s.nunique())
```

Output

```
0    56
1    45
2    35
3    41
4    60
5    57
6    56
7    56
dtype: int64
6
```

Contents

1. DataFrame	2
1.1. DataFrame is a pre-defined class	2
2. Create DataFrame	4
2.1. Create an Empty DataFrame	5
2.2. Create a DataFrame by using list	6
2.3. Creating a DataFrame by using list of lists	9
2.4. Creating a DataFrame by using dictionary	12
2.5. Creating DataFrame by loading the files.....	16

6. PANDAS – DATAFRAME – INTRODUCTION

1. DataFrame

- ✓ A Data frame is a two-dimensional data structure.
- ✓ Data frame is just like a table.
- ✓ Data frame contains rows and columns.

1.1. DataFrame is a pre-defined class

- ✓ DataFrame is a pre-defined class in pandas library.

Example

Emp_No	Name	Salary
101	Ranjan	10000
102	Akshay	20000
103	Daniel	30000
104	Veeru	40000

Columns and Rows are

✓ Columns are

- First column name is : Emp_No
- Second column name is : Name
- Third column name is : Salary

✓ Rows are

- First row data is : 101 Abhi 10000
- Second row data is : 102 Akshay 20000
- Third row data is : 103 Daniel 10000
- Forth row data is : 104 Veeru 10000

2. Create DataFrame

- ✓ DataFrame is a predefined class in pandas.
- ✓ We can create DataFrame in different ways like below,
 - Empty DataFrame
 - By using single list
 - By using nested list
 - By using dictionary
 - with another DataFrame
 - Loading files(real time approach)

Generally

- ✓ In real time when we load existing file then it returns DataFrame

2.1. Create an Empty DataFrame

- ✓ We can create an empty DataFrame

Program Name Creating empty DataFrame
demo1.py

```
import pandas as pd

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

Output

```
Empty DataFrame
Columns: []
Index: []
```

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

2.2. Create a DataFrame by using list

- ✓ We can create DataFrame by using single list.
 - If we are using single list then it's a single column DataFrame
 - If we are using list of lists(nested lists) then it's multiple columns DataFrame

Program Name Creating DataFrame by using single list
demo2.py

```
import pandas as pd

a = [10, 20, 30, 40, 50, 60, 70, 80, 90]
df = pd.DataFrame(a)

print(df)
print(type(df))
```

Output

```
      0
0  10
1  20
2  30
3  40
4  50
5  60
6  70
7  80
8  90
<class 'pandas.core.frame.DataFrame'>
```

Note

- ✓ From the output, DataFrame created with single column.
- ✓ Here column name is Zero, we can customise this as well.

Note on index

- ✓ If no index is passed, then by default, index will be range(n), where n is the array length

Program Name Creating DataFrame by using single list
demo3.py

```
import pandas as pd

names = ["Ranjan", "Sagar", "Daniel", "Prasad", "Kumari",
         "Pravallika", "Arjun", "Akshay"]
df = pd.DataFrame(names)

print(df)
```

Output

```
          0
0      Ranjan
1      Sagar
2     Daniel
3     Prasad
4     Kumari
5  Pravallika
6     Arjun
7     Akshay
```

Program Name Creating single column DataFrame and checking length
demo4.py

```
import pandas as pd

names = ["Ranjan", "Sagar", "Daniel", "Prasad", "Kumari",
"Pravallika", "Arjun", "Akshay"]
df = pd.DataFrame(names)

print(df)
print()
print("The length is:", len(df))
```

Output

```
          0
0      Ranjan
1      Sagar
2     Daniel
3     Prasad
4     Kumari
5  Pravallika
6     Arjun
7     Akshay

The length is: 8
```

2.3. Creating a DataFrame by using list of lists

- ✓ We can create DataFrame with list of lists (nested list).
- ✓ If we are using list of lists then it create a DataFrame with multiple columns.

Program Name Creating DataFrame by using list of lists
demo5.py

```
import pandas as pd

details = [
    ["Ranjan", 11],
    ["Sagar", 12],
    ["Daniel", 13],
    ["Prasad", 14],
    ["Kumari", 15],
    ["Pravallika", 16],
    ["Arjun", 17],
    ["Akshay", 18]
]

df = pd.DataFrame(details)

print(df)
```

Output

	0	1
0	Ranjan	11
1	Sagar	12
2	Daniel	13
3	Prasad	14
4	Kumari	15
5	Pravallika	16
6	Arjun	17
7	Akshay	18

Note

- ✓ From the output, DataFrame created with two columns.
- ✓ Here column names are 0 and 1 and we can customise this as well.

2.3.1. Giving column names to DataFrame

- ✓ We can give column names to DataFrame.

Program Name Creating DataFrame and giving names to columns
demo6.py

```
import pandas as pd

details = [
    ["Sagar", 20, 10000],
    ["Daniel", 16, 20000],
    ["Veeru", 24, 30000],
    ["Raju", 25, 40000],
    ["Kiran", 26, 50000],
    ["Kedar", 27, 60000],
    ["Reena", 28, 70000],
    ["Karthik", 29, 80000],
    ["Satish", 30, 90000]
]

cols = ["Name", "Age", "Salary"]

df = pd.DataFrame(details, columns = cols)

print(df)
```

Output

	Name	Age	Salary
0	Sagar	20	10000
1	Daniel	16	20000
2	Veeru	24	30000
3	Raju	25	40000
4	Kiran	26	50000
5	Kedar	27	60000
6	Reena	28	70000
7	Karthik	29	80000
8	Satish	30	90000

2.4. Creating a DataFrame by using dictionary

- ✓ We can create DataFrame by using dictionary
- ✓ If we are using list of lists then it create a DataFrame with multiple columns.

Program Name Creating DataFrame by using dictionary
demo8.py

```
import pandas as pd

details = {
    "Name": ["Daniel", "Abhi", "Veeru", "Raju", "Kiran",
    "Kedar", "Reena", "Karthik", "Satish"],

    "Age": [20, 21, 23, 24, 25, 26, 27, 28, 29]
}

df = pd.DataFrame(details)

print(df)
```

Output

	Name	Age
0	Daniel	20
1	Abhi	21
2	Veeru	23
3	Raju	24
4	Kiran	25
5	Kedar	26
6	Reena	27
7	Karthik	28
8	Satish	29

Note

- ✓ In above example Name and age considered as column names

2.4.1. We can customize the index values

- ✓ By default index value start from 0
- ✓ We can customise the index values in DataFrame.
- ✓ If index is passed, then the length of the index should equal to the length of the DataFrame.

Program Name Creating DataFrame and giving index
demo9.py

```
import pandas as pd

details = [
    ["Sagar", 20, 10000],
    ["Daniel", 16, 20000],
    ["Veeru", 24, 30000],
    ["Raju", 25, 40000],
    ["Kiran", 26, 50000],
    ["Kedar", 27, 60000],
    ["Reena", 28, 70000],
    ["Karthik", 29, 80000],
    ["Satish", 30, 90000]
]

c = ["Name", "Age", "Salary"]
i = [11, 22, 33, 44, 55, 66, 77, 88, 99]

df = pd.DataFrame(details, columns = c, index = i)

print(df)
```

Output

	Name	Age	Salary
11	Sagar	20	10000
22	Daniel	16	20000
33	Veeru	24	30000
44	Raju	25	40000
55	Kiran	26	50000
66	Kedar	27	60000
77	Reena	28	70000
88	Karthik	29	80000
99	Satish	30	90000

Program Name Creating DataFrame and giving index
demo10.py

```
import pandas as pd

details = [
    ["Sagar", 20, 10000],
    ["Daniel", 16, 20000],
    ["Veeru", 24, 30000],
    ["Raju", 25, 40000],
    ["Kiran", 26, 50000],
    ["Kedar", 27, 60000],
    ["Reena", 28, 70000],
    ["Karthik", 29, 80000],
    ["Satish", 30, 90000]
]

c = ["Name", "Age", "Salary"]
i = ["Row1", "Row2", "Row4", "Row5", "Row6", "Row7", "Row8",
"Row9", "Row10"]

df = pd.DataFrame(details, columns = c, index = i)

print(df)
```

Output

	Name	Age	Salary
Row1	Sagar	20	10000
Row2	Daniel	16	20000
Row4	Veeru	24	30000
Row5	Raju	25	40000
Row6	Kiran	26	50000
Row7	Kedar	27	60000
Row8	Reena	28	70000
Row9	Karthik	29	80000
Row10	Satish	30	90000

2.5. Creating DataFrame by loading the files.

- ✓ We can create DataFrame by loading files like csv, json etc.
- ✓ This we will learn more on 8th chapter.

Contents

1. Loading files	2
2. Loading files	2
2.1. csv file.....	3
2.2. json file.....	7
2.3. excel file	8
2.4. TSV file.....	10
2.5. Table from webpage	11

7. PANDAS – LOADING DIFFERENT FILES

1. Loading files

- ✓ We can load different files in pandas.
- ✓ Whenever we are loading the files then pandas return a **DataFrame**.
- ✓ Once after DataFrame is returned then based on requirement we can perform several operations.

2. Loading files

- ✓ Csv file
- ✓ Json file
- ✓ Excel file
- ✓ Tsv file
- ✓ Table from webpage

2.1. csv file

- ✓ Csv file stands for comma separated file.
- ✓ A csv file contains data like rows and columns.
- ✓ Whenever we are loading csv file then pandas returns DataFrame.
- ✓ We can load csv file by using **read_csv("file name with path")** function

Syntax

```
pd.read_csv("file name with path")
```

Program Loading csv file

Name demo1.py

Input file sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df)
```

Output

	Order ID	Customer Name	Product	Quantity
0	166837	Veeru	34in Ultrawide Monitor	2
1	166838	Tarun	Samsung m10	3
2	166839	Kedar	20in Monitor	1
3	166840	Lavanya	iPhone 11	3
4	166841	Venu	Macbook Pro Laptop	2
..
595	167403	Balaji	Macbook Pro Laptop	1
596	167404	Lavanya	ThinkPad Laptop	1
597	167405	Venu	Flatscreen TV	1
598	167406	Siddhu	Samsung m20	2
599	167407	Tarun	LG Washing Machine	1

[600 rows x 4 columns]

Note

- ✓ Above program, sales1.csv file should exist in current directory otherwise we will get an error like **FileNotFoundException**
- ✓ Let's load file from the folder in this case we need to provide file name with folder.

Program Name Loading csv file
Input file demo2.py
If file name not exist then we will get an error

```
import pandas as pd

df = pd.read_csv("sales123.csv")
print(df)
```

Output

FileNotFoundException: No such file or directory: 'sales123.csv'

Program Name Loading csv file from **files** folder
Input file demo3.py
files\sales1.csv

```
import pandas as pd

df = pd.read_csv('files\sales1.csv')
print(df)
```

Output

	Order ID	Customer Name	Product	Quantity
0	166837	Veeru	34in Ultrawide Monitor	2
1	166838	Tarun	Samsung m10	3
2	166839	Kedar	20in Monitor	1
3	166840	Lavanya	iPhone 11	3
4	166841	Venu	Macbook Pro Laptop	2
..
595	167403	Balaji	Macbook Pro Laptop	1
596	167404	Lavanya	ThinkPad Laptop	1
597	167405	Venu	Flatscreen TV	1
598	167406	Siddhu	Samsung m20	2
599	167407	Tarun	LG Washing Machine	1

[600 rows x 4 columns]

Program Name Loading csv file from D drive
Input file demo4.py
D:\sales1.csv

```
import pandas as pd

df = pd.read_csv('D:\\sales1.csv')
print(df)
```

Output

	Order ID	Customer Name	Product	Quantity
0	166837	Veeru	34in Ultrawide Monitor	2
1	166838	Tarun	Samsung m10	3
2	166839	Kedar	20in Monitor	1
3	166840	Lavanya	iPhone 11	3
4	166841	Venu	Macbook Pro Laptop	2
..
595	167403	Balaji	Macbook Pro Laptop	1
596	167404	Lavanya	ThinkPad Laptop	1
597	167405	Venu	Flatscreen TV	1
598	167406	Siddhu	Samsung m20	2
599	167407	Tarun	LG Washing Machine	1

[600 rows x 4 columns]

2.2. json file

- ✓ JSON stands for **JavaScript Object Notation**
- ✓ In json file data is like key-value pairs and arrays.
- ✓ It is commonly used for transmitting data in web applications
- ✓ Whenever we are loading json file then pandas returns DataFrame.
- ✓ We can load csv file by using **read_json("file name with path")** function

Syntax

```
pd.read_json("files name with path")
```

Program Name Loading json file
Input file demo5.py
sales1.json

```
import pandas as pd

df = pd.read_json("sales1.json")
print(df)
```

Output

```
      order_id    cust_name        product  quantity
0   16278939       Lavanya     ThinkPad Laptop      2
1   16278966       Kedar      Flatscreen TV      1
2   16278993  Jaya Chandra    Macbook Pro Laptop      2
3   16279020  Mallikarjun      iPhone 11      3
4   16279047       Shahid      LG Washing Machine      1
..       ...
495  16292304       Sagar      iPhone 7      1
496  16292331  Chaithanya      Samsung m20      1
497  16292358       Siddhu      LG Dryer      2
498  16292385       Siddhu  AA Batteries (4-pack)      1
499  16292412       Sagar      iPhone 11      2

[500 rows x 4 columns]
```

2.3. excel file

- ✓ Excel is a spreadsheet from Microsoft.
- ✓ It is using mainly to store business applications data
- ✓ Whenever we are loading excel file then pandas returns DataFrame.
- ✓ We can load csv file by using **read_excel("file name with path")** function

Run below command

```
pip install xlrd
```

Syntax

```
pd.read_excel("files name with path")
```

Program Loading excel file
Name demo6.py
Input file sales1.xlsx

```
import pandas as pd

df = pd.read_excel("sales1.xlsx")
print(df)
```

Output

```
   Order ID Custer Name          Product  Quantity
0      166837    Veeru  34in Ultrawide Monitor       2
1      166838    Tarun        Samsung m10       3
2      166839    Kedar     20in Monitor       1
3      166840   Lavanya       iPhone 11       3
4      166841    Venu     Macbook Pro Laptop       2
..      ...
595     167403   Balaji     Macbook Pro Laptop       1
596     167404   Lavanya     ThinkPad Laptop       1
597     167405    Venu      Flatscreen TV       1
598     167406   Siddhu        Samsung m20       2
599     167407   Tarun      LG Washing Machine       1

[600 rows x 4 columns]
```

2.4. TSV file

- ✓ TSV stands for **Tab Separated File**
- ✓ Whenever we are loading tsv file then pandas returns DataFrame.
- ✓ We can load csv file by using **read_table("file name with path")** function

Syntax

```
pd.read_table(path and file name)
```

Program Name Loading tsv file

Input file demo7.py

sales1.tsv

```
import pandas as pd
```

```
df = pd.read_table("sales1.tsv")
print(df)
```

Output

	Order ID	Custer Name	Product	Quantity
0	166837	Veeru	34in Ultrawide Monitor	2
1	166838	Tarun	Samsung m10	3
2	166839	Kedar	20in Monitor	1
3	166840	Lavanya	iPhone 11	3
4	166841	Venu	Macbook Pro Laptop	2
..
595	167403	Balaji	Macbook Pro Laptop	1
596	167404	Lavanya	ThinkPad Laptop	1
597	167405	Venu	Flatscreen TV	1
598	167406	Siddhu	Samsung m20	2
599	167407	Tarun	LG Washing Machine	1

[600 rows x 4 columns]

2.5. Table from webpage

- ✓ We can load table from webpage.
- ✓ Whenever we are loading table from webpage then it pandas returns DataFrame.
- ✓ We can load csv file by using `read_html("url")` function

Syntax

```
pd.read_html("url")
```

Program Loading table from website

Name demo8.py

Input Loading from web page

```
import pandas as pd
```

```
url = 'https://en.wikipedia.org/wiki/The_World%27s_Billionaires'  
df_list = pd.read_html(url)
```

```
print(df_list[2])
```

Output

No.	Name	Net worth (USD)	Age	Nationality	Source(s) of wealth
0 1	Elon Musk	\$219 billion	50	United States	Tesla, SpaceX
1 2	Jeff Bezos	\$177 billion	58	United States	Amazon
2 3	Bernard Arnault & family	\$158 billion	73	France	LVMH
3 4	Bill Gates	\$129 billion	66	United States	Microsoft
4 5	Warren Buffett	\$118 billion	91	United States	Berkshire Hathaway
5 6	Larry Page	\$111 billion	49	United States	Alphabet Inc.
6 7	Sergey Brin	\$107 billion	48	United States	Alphabet Inc.
7 8	Larry Ellison	\$106 billion	77	United States	Oracle Corporation
8 9	Steve Ballmer	\$91.4 billion	66	United States	Microsoft
9 10	Mukesh Ambani	\$90.7 billion	64	India	Reliance Industries

Contents

1. DataFrame Attributes	2
1.1. Length of DataFrame	2
1.2. columns.....	3
1.3. shape.....	4
1.4. shape[0]	5
1.5. shape[1]	6
1.6. size	7
1.7. dtypes.....	9
1.8. empty	10
1.9. index.....	12
1.10. values	13
1.11. T.....	14

8. PANDAS – DATAFRAME – ATTRIBUTES

1. DataFrame Attributes

- ✓ DataFrame is a predefined class.
- ✓ DataFrame having different attributes.
- ✓ These attributes return information about the DataFrame object.

1.1. Length of DataFrame

- ✓ We can check length of DataFrame by using len(p) function
- ✓ This function returns the total number of rows from the DataFrame

Program Checking total number of rows/length from the DataFrame

Name demo1.py

Input file sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print("Total number of rows in DataFrame:", len(df))
```

Output

Total number of rows in DataFrame: 600

1.2. columns

- ✓ columns is predefined attribute in DataFrame class.
- ✓ We can access columns attribute by using DataFrame object.
- ✓ This attribute returns all column names from the DataFrame

Program Accessing columns attribute from DataFrame.

Name demo2.py

Input file sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df.columns)
```

Output

```
Index(['Order ID', 'Customer Name', 'Product', 'Quantity'],
      dtype='object')
```

1.3. shape

- ✓ shape is predefined attribute in DataFrame class.
- ✓ We can access shape attribute by using DataFrame object.
- ✓ This attribute returns the total number of rows and columns in tuple format.
 - From the tuple, first value represents total number of rows
 - From the tuple, second value represents total number of columns

Program Name Accessing shape attribute from DataFrame.

Input file demo3.py

sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df.shape)
```

Output

(600, 4)

1.4. shape[0]

- ✓ shape is predefined attribute in DataFrame class.
- ✓ We can access shape attribute by using DataFrame object.
- ✓ This attribute returns the total number of rows and columns in tuple
- ✓ Shape[0] returns total number for rows from the DataFrame

Program Name Accessing shape attribute and checking total number of rows

Input file demo4.py
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df.shape[0])
```

Output

600

1.5. shape[1]

- ✓ shape is predefined attribute in DataFrame class.
- ✓ We can access shape attribute by using DataFrame object.
- ✓ This attribute returns the total number of rows and columns in tuple
- ✓ Shape[1] returns total number for columns from the DataFrame

Program Name Accessing shape attribute and checking total number of columns

Input file demo5.py
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df.shape[1])
```

Output

4

1.6. size

- ✓ size is predefined attribute in DataFrame class.
- ✓ We can access size attribute by using DataFrame object.
- ✓ This attribute returns the total number of elements/values in DataFrame

Program Accessing total number of elements/values from DataFrame.

Name demo6.py

Input file sales1.csv

```
import pandas as pd  
  
df = pd.read_csv("sales1.csv")  
print(df.size)
```

Output

```
2400
```

Make a note:

- ✓ size = Number of rows X Number of columns
- ✓ size = Row_count X Column_count

Program Name Accessing total number of elements/values from DataFrame.
Input file demo7.py
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print("Number of elements in DataFrame:", df.size)
print("Number of elements in DataFrame:",
      df.shape[0]*df.shape[1])
```

Output

```
Number of elements in DataFrame: 2400
Number of elements in DataFrame: 2400
```

1.7. dtypes

- ✓ dtypes is predefined attribute in DataFrame class.
- ✓ We can access dtypes attribute by using DataFrame object.
- ✓ This attribute returns the datatype of each column.

Program Checking all columns datatype from DataFrame

Name demo8.py

Input file sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df.dtypes)
```

Output

```
Order ID          int64
Customer Name    object
Product          object
Quantity         int64
dtype: object
```

1.8. empty

- ✓ empty is predefined attribute in DataFrame class.
- ✓ We can access empty attribute by using DataFrame object.
- ✓ This attribute check DataFrame is empty or not,
 - If DataFrame is empty then it returns **True** other **False**.

Program Checking DataFrame empty or not

Name demo9.py

Input file sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df.empty)
```

Output

False

Program Name Checking DataFrame empty or not

Name demo10.py

Input file sales1.csv

```
import pandas as pd

df = pd.DataFrame()

print(df)
print()
print(df.empty)
```

Output

```
Empty DataFrame
Columns: []
Index: []

True
```

1.9. index

- ✓ index is predefined attribute in DataFrame class.
- ✓ We can access index attribute by using DataFrame object.
- ✓ This attribute return index start and end value from the DataFrame.

Program Name Accessing index attribute from DataFrame

Input file demo11.py

sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df.index)
```

Output

```
RangelIndex(start=0, stop=600, step=1)
```

1.10. values

- ✓ values is predefined attribute in DataFrame class.
- ✓ We can access values attribute by using DataFrame object.
- ✓ This attribute return values of DataFrame,
 - Each row values in one array from starting to last row.

Program Accessing values attribute from DataFrame

Name demo12.py

Input file sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df.values)
```

Output

```
[[166837 'Veeru' '34in Ultrawide Monitor' 2]
 [166838 'Tarun' 'Samsung m10' 3]
 [166839 'Kedar' '20in Monitor' 1]
 ...
 [167405 'Venu' 'Flatscreen TV' 1]
 [167406 'Siddhu' 'Samsung m20' 2]
 [167407 'Tarun' 'LG Washing Machine' 1]]
```

1.11. T

- ✓ T is predefined attribute in DataFrame class.
- ✓ We can access T attribute by using DataFrame object.
- ✓ T means its transpose, it returns **rows as columns** and **columns as rows**

Program Name Converting rows as column and columns as rows from DataFrame demo13.py

```
import pandas as pd

details = [["Sagar", 20, 10000], ["Daniel", 16, 20000], ["Veeru", 24, 30000], ["Raju", 25, 40000], ["Kiran", 26, 50000], ["Kedar", 27, 60000], ["Reena", 28, 70000]]

df = pd.DataFrame(details, columns = ["Name", "Age", "Salary"])

print(df)
print()
print(df.T)
```

Output

```
Name    Age   Salary
0   Sagar   20   10000
1   Daniel  16   20000
2   Veeru   24   30000
3   Raju    25   40000
4   Kiran   26   50000
5   Kedar   27   60000
6   Reena   28   70000

          0        1        2        3        4        5        6
Name     Sagar   Daniel  Veeru   Raju   Kiran   Kedar   Reena
Age      20       16      24      25      26      27      28
Salary   10000   20000   30000   40000   50000   60000   70000
```

Contents

1. DataFrame Methods	2
1.1. head()	2
1.2. tail()	3
1.3. info().....	4
1.4. count().....	6
1.5. describe()	8
1.6. nunique().....	9
2. Accessing single column From DataFrame	10
3. Rearranging columns in DataFrame	14

9. PANDAS – DATAFRAME – METHODS

1. DataFrame Methods

- ✓ DataFrame is a predefined class.
- ✓ DataFrame having different methods.
- ✓ These methods perform an operation on DataFrame and returns result

1.1. head()

- ✓ head() is predefined method in DataFrame class.
- ✓ We can access head() method by using DataFrame object only.
- ✓ This method returns first five rows from the DataFrame

Program Accessing first five rows from DataFrame

Name demo1.py

Input file sales1.csv

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")
df2 = df1.head()

print(df2)
```

Output

	Order ID	Customer Name	Product	Quantity
0	166837	Veeru	34in Ultrawide Monitor	2
1	166838	Tarun	Samsung m10	3
2	166839	Kedar	20in Monitor	1
3	166840	Lavanya	iPhone 11	3
4	166841	Venu	Macbook Pro Laptop	2

1.2. tail()

- ✓ tail() is predefined method in DataFrame class.
- ✓ We can access tail() method by using DataFrame object only.
- ✓ This method returns last five rows from the DataFrame

Program Accessing last five rows from DataFrame

Name demo2.py

Input file sales1.csv

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")
df2 = df1.tail()

print(df2)
```

Output

	Order ID	Customer Name	Product	Quantity
595	167403	Balaji	Macbook Pro Laptop	1
596	167404	Lavanya	ThinkPad Laptop	1
597	167405	Venu	Flatscreen TV	1
598	167406	Siddhu	Samsung m20	2
599	167407	Tarun	LG Washing Machine	1

1.3. info()

- ✓ info() is predefined method in DataFrame class.
- ✓ We can access info() method by using DataFrame object only.
- ✓ This method returns below information about DataFrame,
 - Type of object
 - Range of object
 - Number of columns
 - Number of rows
 - The data type of each column
 - Number of data types
 - Total memory usage

Program Name Accessing info() method from DataFrame
Input file demo3.py
sales1.csv

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")
df1.info()
```

Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 600 entries, 0 to 599
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Order ID        600 non-null    int64  
 1   Customer Name  600 non-null    object  
 2   Product         600 non-null    object  
 3   Quantity        600 non-null    int64  
dtypes: int64(2), object(2)
memory usage: 18.9+ KB
```

Program Name Accessing info() method from DataFrame
Input file demo4.py
sales1_with_nan.csv

```
import pandas as pd

df1 = pd.read_csv("sales1_with_nan.csv")
df1.info()
```

Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 600 entries, 0 to 599
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype  
---  --  
 0   Order ID        600 non-null    int64  
 1   Customer Name  599 non-null    object  
 2   Product         600 non-null    object  
 3   Quantity        598 non-null    float64 
dtypes: float64(1), int64(1), object(2)
memory usage: 18.9+ KB
```

1.4. count()

- ✓ count() is predefined method in DataFrame class.
- ✓ We can access count() method by using DataFrame object only.
- ✓ This method returns number of non-null values from each column.

Program Accessing count() method from DataFrame

Name demo5.py

Input file sales1.csv

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")
c = df1.count()
print(c)
```

Output

```
Order ID      600
Customer Name 600
Product        600
Quantity       600
dtype: int64
```

Program Name Accessing count() method from DataFrame
Input file demo6.py
sales1_with_nan.csv

```
import pandas as pd

df1 = pd.read_csv("sales1_with_nan.csv")
c = df1.count()

print(c)
```

Output

```
Order ID      600
Customer Name 599
Product       600
Quantity      598
dtype: int64
```

1.5. describe()

- ✓ `describe()` is predefined method in `DataFrame` class.
- ✓ We can access `describe()` method by using `DataFrame` object.
- ✓ This method returns the below values,
 - `count`
 - `mean`
 - `std`
 - `min`
 - `25%`
 - `50%`
 - `75%`
 - `max`

Program Name Accessing `describe()` method from `DataFrame`

Input file `demo7.py`

Input file `sales1_with_nan.csv`

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")
dsc = df1.describe()

print(dsc)
```

Output

	Order ID	Quantity
count	600.000000	600.000000
mean	167122.751667	1.481667
std	164.948568	0.683454
min	166837.000000	1.000000
25%	166980.750000	1.000000
50%	167120.500000	1.000000
75%	167266.250000	2.000000
max	167409.000000	3.000000

1.6. nunique()

- ✓ nunique() is predefined method in DataFrame class.
- ✓ We can access nunique() method by using DataFrame object only.
- ✓ This method returns number of unique values from the DataFrame.

Program Name Get the number of nunique values from the DataFrame
demo8.py

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")
nu = df1.nunique()

print(nu)
```

Output

```
Order ID      573
Customer Name  23
Product        21
Quantity        3
dtype: int64
```

2. Accessing single column From DataFrame

- ✓ We can access columns from the DataFrame,
 - If we access single column then it returns the Series
 - If we access two column then it returns the DataFrame with two columns

Program Name Accessing single column from the DataFrame
Input file demo9.py
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")

print(df.Product)
```

Output

```
0      34in Ultrawide Monitor
1                      Samsung m10
2                  20in Monitor
3                  iPhone 11
4      Macbook Pro Laptop
...
595      Macbook Pro Laptop
596      ThinkPad Laptop
597      Flatscreen TV
598      Samsung m20
599      LG Washing Machine
Name: Product, Length: 600, dtype: object
```

Program Name Accessing single column from the DataFrame
Input file demo10.py
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")

print(df["Product"])
```

Output

```
0      34in Ultrawide Monitor
1                  Samsung m10
2                  20in Monitor
3                  iPhone 11
4      Macbook Pro Laptop
        ...
595     Macbook Pro Laptop
596     ThinkPad Laptop
597     Flatscreen TV
598     Samsung m20
599     LG Washing Machine
Name: Product, Length: 600, dtype: object
```

Program Name Accessing two column from the DataFrame
Input file demo11.py
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")

print(df[["Customer Name", "Product"]])
```

Output

```
      Customer Name          Product
0        Veeru    34in Ultrawide Monitor
1        Tarun            Samsung m10
2        Kedar        20in Monitor
3       Lavanya           iPhone 11
4        Venu      Macbook Pro Laptop
..         ...
595      Balaji      Macbook Pro Laptop
596      Lavanya      ThinkPad Laptop
597      Venu        Flatscreen TV
598      Siddhu        Samsung m20
599      Tarun        LG Washing Machine

[600 rows x 2 columns]
```

Program Name Accessing single column from the DataFrame, applying sum
Input file demo12.py
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
total = df['Quantity'].sum()

print(total)
```

Output

889

3. Rearranging columns in DataFrame

- ✓ We can rearrange columns in DataFrame
- ✓ We can customise the order of columns in DataFrame

Program Name Creating DataFrame by loading csv file
demo13.py
Input file sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")

print(df)
```

Output

```
      Order ID Customer Name          Product  Quantity
0      166837     Veeru  34in Ultrawide Monitor        2
1      166838     Tarun       Samsung m10        3
2      166839     Kedar      20in Monitor        1
3      166840    Lavanya      iPhone 11        3
4      166841     Venu     Macbook Pro Laptop        2
..      ...
595     167403    Balaji     Macbook Pro Laptop        1
596     167404    Lavanya     ThinkPad Laptop        1
597     167405     Venu      Flatscreen TV        1
598     167406    Siddhu      Samsung m20        2
599     167407     Tarun      LG Washing Machine        1

[600 rows x 4 columns]
```

Program Name Rearranging columns in DataFrame
Input file demo14.py
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
df = df[["Product", "Customer Name", "Quantity", "Order ID"]]

print(df)
```

Output

	Product	Customer Name	Quantity	Order ID
0	34in Ultrawide Monitor	Veeru	2	166837
1	Samsung m10	Tarun	3	166838
2	20in Monitor	Kedar	1	166839
3	iPhone 11	Lavanya	3	166840
4	Macbook Pro Laptop	Venu	2	166841
..
595	Macbook Pro Laptop	Balaji	1	167403
596	ThinkPad Laptop	Lavanya	1	167404
597	Flatscreen TV	Venu	1	167405
598	Samsung m20	Siddhu	2	167406
599	LG Washing Machine	Tarun	1	167407

[600 rows x 4 columns]

10. Pandas – DataFrame – Rename column & index

Contents

1. rename(p) method - Changing single column name	2
2. rename(p) method - Changing multiple column names	5
3. columns attribute - Changing multiple column names.....	8
4. rename(p) method - Changing index in DataFrame.....	10
5. index attribute - Changing multiple column names.....	12
6. Converting column names from lower case to upper case	15

10. Pandas – DataFrame – Rename column & index

- ✓ Sometimes we need to process column names and index.
- ✓ Based on requirement we can rename/change/modify/update column names and index.

1. rename(p) method - Changing single column name

- ✓ rename(p) is a predefined method in DataFrame
- ✓ We can access rename() method by using DataFrame object.
- ✓ By using this method we can change single column name in DataFrame.

Program Name Creating DataFrame and checking column names
Input file demo1.py
sales3.csv

```
import pandas as pd

df = pd.read_csv("sales3.csv")
print(df.head())
print()
print(df.columns)
```

Output

```
    ord id cust name  cust id          prod name  prod cost
0  192837     Veeru       3           LG Mobile   65999
1  192838   Neelima      19      Apple iPad 10.2-inch   63000
2  192839    Balaji      12  34in Ultrawide Monitor   75999
3  192840    Shahid      20        iPhone 11   60000
4  192841     Vinay      10  Bose SoundSport Headphones   69999
Index(['ord id', 'cust name', 'cust id', 'prod name', 'prod cost'], dtype='object')
```

Note

- ✓ We need to use python dictionary to specify old column as key and new column as value.
- ✓ Whenever changing column then key should be match with column name in DataFrame otherwise changes will not reflects.

Program Name Renaming single column in DataFrame
Input file demo2.py
sales3.csv

```
import pandas as pd

df1 = pd.read_csv("sales3.csv")
d = {
    "ord id": "Order Id"
}

df2 = df1.rename(columns = d)

print(df1.head())
print()
print(df2.head())
```

Output

	ord id	cust name	cust id	prod name	prod cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

	Order Id	cust name	cust id	prod name	prod cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

Note

- ✓ Whenever changing column then key should be match with column name in DataFrame otherwise changes will not reflects.

Program Name Renaming single column in DataFrame
Input file demo3.py
sales3.csv

```
import pandas as pd

df1 = pd.read_csv("sales3.csv")

d = {
    "ord id": "Order Id"
}

df2 = df1.rename(columns = d)

print(df1.head())
print()
print(df2.head())
```

Output

```
      ord id cust name  cust id          prod name  prod cost
0  192837     Veeru        3           LG Mobile   65999
1  192838   Neelima       19      Apple iPad 10.2-inch  63000
2  192839    Balaji       12  34in Ultrawide Monitor  75999
3  192840    Shahid       20        iPhone 11  60000
4  192841     Vinay       10  Bose SoundSport Headphones  69999

      ord id cust name  cust id          prod name  prod cost
0  192837     Veeru        3           LG Mobile   65999
1  192838   Neelima       19      Apple iPad 10.2-inch  63000
2  192839    Balaji       12  34in Ultrawide Monitor  75999
3  192840    Shahid       20        iPhone 11  60000
4  192841     Vinay       10  Bose SoundSport Headphones  69999
```

2. rename(p) method - Changing multiple column names

- ✓ rename(p) is a predefined method in DataFrame
- ✓ We can access rename() method by using DataFrame object.
- ✓ We can even change multiple column names by using rename(p) method.
- ✓ We need to use python dictionary to specify old column as key and new column as value.

Program Renaming multiple column in DataFrame

Name demo4.py

Input file sales3.csv

```
import pandas as pd

df1 = pd.read_csv("sales3.csv")

d = {
    'ord id': 'Order Id',
    'cust name': 'Customer Name',
    'cust id': 'Customer Id',
    'prod name': 'Product Name',
    'prod cost': 'Product Cost'
}

df2 = df1.rename(columns = d)

print(df1.head())
print()
print(df2.head())
```

Output

Data Science – Pandas – DataFrame – Rename column, index

```
    ord id cust name  cust id                  prod name  prod cost
0  192837     Veeru       3                 LG Mobile   65999
1  192838   Neelima      19      Apple iPad 10.2-inch   63000
2  192839    Balaji      12  34in Ultrawide Monitor   75999
3  192840    Shahid      20            iPhone 11   60000
4  192841    Vinay       10  Bose SoundSport Headphones   69999

Order Id Customer Name  Customer Id                  Product Name  Product Cost
0    192837     Veeru       3                 LG Mobile   65999
1    192838   Neelima      19      Apple iPad 10.2-inch   63000
2    192839    Balaji      12  34in Ultrawide Monitor   75999
3    192840    Shahid      20            iPhone 11   60000
4    192841    Vinay       10  Bose SoundSport Headphones   69999
```

Program Name Renaming multiple column in DataFrame
Input file demo5.py
sales3.csv

```
import pandas as pd

df1 = pd.read_csv("sales3.csv")

d = {
    "ord id": "order_id",
    "cust name": "customer_name",
    "cust id": "customer_id",
    "prod name": "product_name",
    "prod cost": "product_cost"
}

df2 = df1.rename(columns = d)

print(df1.head())
print()
print(df2.head())
```

Output

	ord id	cust name	cust id	prod name	prod cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

	order_id	customer_name	customer_id	product_name	product_cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

3. columns attribute - Changing multiple column names

- ✓ columns is predefined attribute in DataFrame class.
- ✓ We can access columns attribute by using DataFrame object.
- ✓ By using columns attribute we can even change multiple column names.

Program Name Renaming multiple column in DataFrame

Input file demo6.py

sales3.csv

```
import pandas as pd

df1 = pd.read_csv("sales3.csv")

print(df1.head())

df1.columns = [
    "order_id",
    "customer_name",
    "customer_id",
    "product_name",
    "product_cost"
]

print()
print(df1.head())
```

Output

	ord id	cust name	cust id	prod name	prod cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999
	order_id	customer_name	customer_id	product_name	product_cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

Note

- ✓ While using columns attributes to change column names then Number of DataFrame column names should be match with existing columns otherwise we will get an error.

Program Renaming multiple column in DataFrame
Name demo7.py
Input file sales3.csv

```
import pandas as pd

df1 = pd.read_csv("sales3.csv")

print(df1.head())

df1.columns = [
    "order_id",
    "customer_name",
    "customer_id",
    "product_name",
    "product_cost",
    "total"
]

print()
print(df1.head())
```

Output

ValueError: Length mismatch: Expected axis has 5 elements, new values have 6 elements

4. rename(p) method - Changing index in DataFrame

- ✓ rename(p) is a predefined method in DataFrame
- ✓ We can access rename() method by using DataFrame object.
- ✓ Based on requirement we can change the index in DataFrame.
- ✓ By using rename(p) we can change the DataFrame index

Program Name Creating a sample Dataframe
demo8.py

```
import pandas as pd

d = {
    "order_id": [11, 21, 31],
    "customer_name": ["Prasad", "Daniel", "Jeswanth"],
    "product": ["iPhone", "hTC", "macbook"]
}

df1 = pd.DataFrame(d)
print(df1)
```

Output

	order_id	customer_name	product
0	11	Prasad	iPhone 11
1	21	Daniel	hTC
2	31	Jeswanth	macbook

Program Name Changing index in DataFrame
demo9.py

```
import pandas as pd

d = {
    "order_id": [11, 21, 31],
    "customer_name": ["Prasad", "Daniel", "Jeswanth"],
    "product": ["iPhone", "hTC", "macbook"]
}

i = {0: 77, 1: 88, 2: 99}

df1 = pd.DataFrame(d)
df2 = df1.rename(index = i)

print(df1)
print()
print(df2)
```

Output

```
   order_id  customer_name   product
0         11           Prasad    iPhone
1         21           Daniel      hTC
2         31        Jeswanth   macbook

   order_id  customer_name   product
77        11           Prasad    iPhone
88        21           Daniel      hTC
99        31        Jeswanth   macbook
```

5. index attribute - Changing multiple column names

- ✓ index is predefined attribute in DataFrame class.
- ✓ We can access index attribute by using DataFrame object.
- ✓ By using index attribute we can even change index in DataFrame

Program Name Changing index in DataFrame, using axis parameter
demo10.py

```
import pandas as pd

d = {
    "order_id": [11, 21, 31],
    "customer_name": ["Prasad", "Daniel", "Jeswanth"],
    "product": ["iPhone", "hTC", "macbook"]
}

df1 = pd.DataFrame(d)
print(df1)

df1.index = [77, 88, 99]

print()
print(df1)
```

Output

```
      order_id customer_name   product
0          11        Prasad     iPhone
1          21        Daniel       hTC
2          31      Jeswanth    macbook

      order_id customer_name   product
77         11        Prasad     iPhone
88         21        Daniel       hTC
99         31      Jeswanth    macbook
```

Program Name Changing index in DataFrame, using index attribute.
Input file demo11.py
sales31.csv

```
import pandas as pd

df1 = pd.read_csv("sales31.csv")
print(df1)

df1.index = range(10, 20)

print()
print(df1)
```

Output

	ord id	cust name	cust id	prod name	prod cost
0	192837	Veetu	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999
5	192842	Vinay	10	20in Monitor	65999
6	192843	Balaji	12	iPhone 8	55999
7	192844	Nireekshan	1	27in 4K Gaming Monitor	55999
8	192845	Venu	23	iPhone 9	55999
9	192846	Veetu	3	20in Monitor	63000
	ord id	cust name	cust id	prod name	prod cost
10	192837	Veetu	3	LG Mobile	65999
11	192838	Neelima	19	Apple iPad 10.2-inch	63000
12	192839	Balaji	12	34in Ultrawide Monitor	75999
13	192840	Shahid	20	iPhone 11	60000
14	192841	Vinay	10	Bose SoundSport Headphones	69999
15	192842	Vinay	10	20in Monitor	65999
16	192843	Balaji	12	iPhone 8	55999
17	192844	Nireekshan	1	27in 4K Gaming Monitor	55999
18	192845	Venu	23	iPhone 9	55999
19	192846	Veetu	3	20in Monitor	63000

Program Name Changing columns and index in a Dataframe
demo12.py

```
import pandas as pd

d = {
    "Ord Id": [11, 21, 31],
    "Customer Name": ["Prasad", "Daniel", "Jeswanth"],
    "Product": ["iPhone", "hTC", "macbook"]
}

df1 = pd.DataFrame(d)
print(df1)

df1.index = [333, 444, 555]
df1.columns = ["order_id", "customer_name", "product"]

print()
print(df1)
```

Output

```
   Ord Id Customer Name  Product
0        11        Prasad   iPhone
1        21        Daniel      hTC
2        31     Jeswanth  macbook

   order_id customer_name  product
333          11        Prasad   iPhone
444          21        Daniel      hTC
555          31     Jeswanth  macbook
```

6. Converting column names from lower case to upper case

- ✓ Based on requirement we can convert DataFrame column names from lower case to upper case.

Program Name demo13.py
Input file sales3.csv

```
import pandas as pd

df1 = pd.read_csv("sales3.csv")

print(df1.head())

df1.columns = df1.columns.str.upper()

print()
print(df1.head())
```

Output

```
    ord id cust name  cust id                  prod name  prod cost
0  192837     Veeru        3                 LG Mobile   65999
1  192838   Neelima       19      Apple iPad 10.2-inch   63000
2  192839    Balaji       12  34in Ultrawide Monitor   75999
3  192840    Shahid       20            iPhone 11   60000
4  192841     Vinay       10  Bose SoundSport Headphones   69999

    ORD ID CUST NAME  CUST ID                  PROD NAME  PROD COST
0  192837     Veeru        3                 LG Mobile   65999
1  192838   Neelima       19      Apple iPad 10.2-inch   63000
2  192839    Balaji       12  34in Ultrawide Monitor   75999
3  192840    Shahid       20            iPhone 11   60000
4  192841     Vinay       10  Bose SoundSport Headphones   69999
```

11. PANDAS – INPLACE PARAMETER

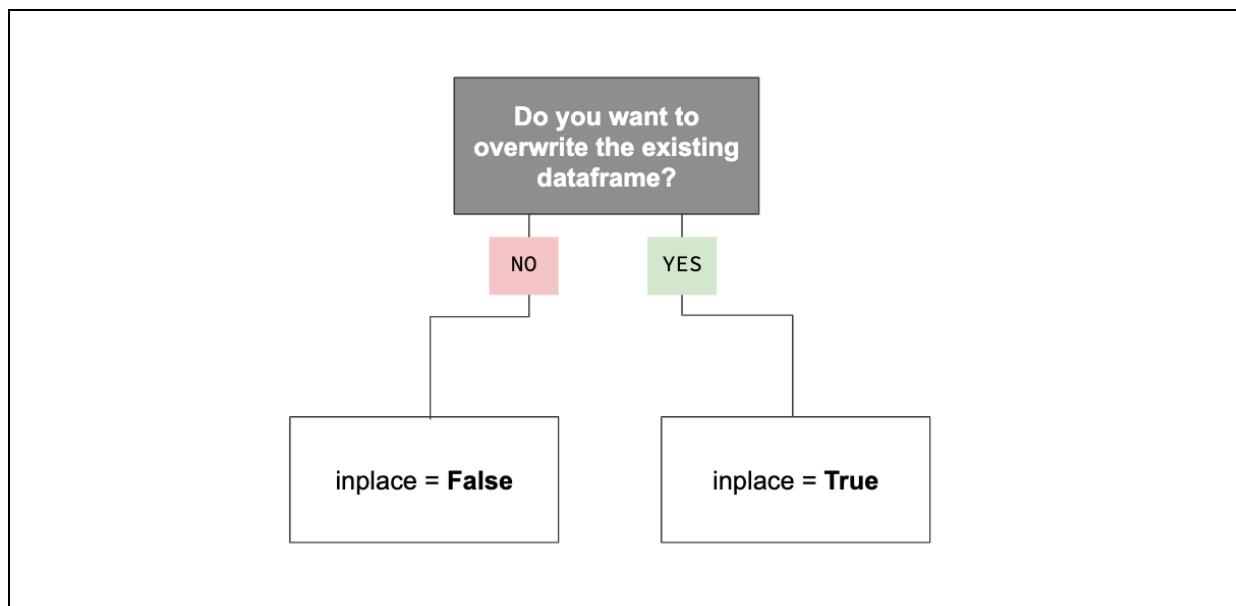
Contents

1. The inplace parameter	2
2. rename(p) method – without inplace parameter.....	3
3. rename(p) method – with inplace parameter.....	4

11. PANDAS – INPLACE PARAMETER

1. The inplace parameter

- ✓ inplace is a parameter for DataFrame methods like,
 - rename(p, inplace)
 - drop(p, inplace),
 - sort_values(p, inplace),
 - set_index(p, inplace) etc
- ✓ This inplace parameter accepts two values,
 - inplace = **False** means it cannot overwrite the existing dataframe
 - inplace = **True** means it can overwrite the existing dataframe.



Simple words

- ✓ In-place means that you should update the original object instead of creating new object

2. rename(p) method – without inplace parameter

- ✓ rename(p) is a predefined method in DataFrame
- ✓ We can access rename() method by using DataFrame object.
- ✓ We can even change multiple column names by using rename(p) method.

Program rename(p) method - without inplace parameter

Name demo1.py

Input file sales3.csv

```
import pandas as pd

df1 = pd.read_csv("sales3.csv")

d = {"ord id": "order_id", "cust name": "customer_name", "cust id": "customer_id", "prod name": "product_name", "prod cost": "product_cost"}

print(df1.head())
df1.rename(columns = d)
print()
print(df1.head())
```

Output

	ord id	cust name	cust id	prod name	prod cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

	ord id	cust name	cust id	prod name	prod cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

3. rename(p) method – with inplace parameter

- ✓ rename(p, inplace = **True**) is a predefined method in DataFrame.
- ✓ We can provide inplace parameter to rename(p, inplace = **True**) method
- ✓ If we provide inplace parameter then directly changes will be apply on original DataFrame.
- ✓ Whenever if we apply inplace = **True** then, it should update the original DataFrame instead of creating new DataFrame.

Program rename(p) method - with inplace parameter

Name demo2.py

Input file sales3.csv

```
import pandas as pd

df1 = pd.read_csv("sales3.csv")

d = {"ord id": "order_id", "cust name": "customer_name", "cust id": "customer_id", "prod name": "product_name", "prod cost": "product_cost"}

print(df1.head())
df1.rename(columns = d, inplace = True)
print()
print(df1.head())
```

Output

	ord id	cust name	cust id	prod name	prod cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999
	order_id	customer_name	customer_id	product_name	product_cost
0	192837	Veeru	3	LG Mobile	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000
2	192839	Balaji	12	34in Ultrawide Monitor	75999
3	192840	Shahid	20	iPhone 11	60000
4	192841	Vinay	10	Bose SoundSport Headphones	69999

13. PANDAS – iLOC AND LOC

Contents

1. Selecting single column.....	2
2. Selecting multiple columns	4
3. Selecting specific column values.....	6
4. iloc and loc indexers.....	9
5. iloc[] indexer.....	10
5.1. iloc[] syntax	10
5.2. Creating a DataFrame	11
5.3. Selecting single column.....	12
5.4. Selecting multiple rows and columns	19
6. loc[] indexer.....	24
6.1. loc[] - Selecting rows by label/index	25
6.2. Creating a dataframe	25
6.3. Boolean / Logical indexing	35

13. PANDAS – iLOC AND LOC

1. Selecting single column

- ✓ Based on requirement we can select columns from the DataFrame.
 - If we select a single column then it returns the Series

Program Name demo1.py
Input file sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
print(df.Product)
```

Output

```
0      34in Ultrawide Monitor
1                  Samsung m10
2                  20in Monitor
3                  iPhone 11
4      Macbook Pro Laptop
        ...
595     Macbook Pro Laptop
596     ThinkPad Laptop
597     Flatscreen TV
598     Samsung m20
599     LG Washing Machine
Name: Product, Length: 600, dtype: object
```

Program Name Selecting single column from the DataFrame
Input file demo2.py
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
s = df["Product"]
print(s)
```

Output

```
0      34in Ultrawide Monitor
1                  Samsung m10
2                  20in Monitor
3                  iPhone 11
4      Macbook Pro Laptop
       ...
595     Macbook Pro Laptop
596     ThinkPad Laptop
597     Flatscreen TV
598     Samsung m20
599     LG Washing Machine
Name: Product, Length: 600, dtype: object
```

2. Selecting multiple columns

- ✓ Based on requirement we can select columns from the DataFrame.
 - If we select more than one column then it returns the DataFrame.

Program Selecting multiple columns from the DataFrame
Name demo3.py
Input file sales1.csv

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")
cols = ["Customer Name", "Product"]
df2 = df1[cols]

print(df2)
```

Output

	Customer Name	Product
0	Veeru	34in Ultrawide Monitor
1	Tarun	Samsung m10
2	Kedar	20in Monitor
3	Lavanya	iPhone 11
4	Venu	Macbook Pro Laptop
..
595	Balaji	Macbook Pro Laptop
596	Lavanya	ThinkPad Laptop
597	Venu	Flatscreen TV
598	Siddhu	Samsung m20
599	Tarun	LG Washing Machine
[600 rows x 2 columns]		

Program Name Selecting single column from the DataFrame, apply total
Input file demo4.py
sales1.csv

```
import pandas as pd

df = pd.read_csv("sales1.csv")
total = df['Quantity'].sum()
print(total)
```

Output

889

3. Selecting specific column values

- ✓ Based on requirement we can select specific column values from the DataFrame
 - By using Boolean condition we can select the data from DataFrame.

Program Name Creating a DataFrame by loading csv file
demo5.py
Input file sales1.csv

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")

print(df1)
```

Output

	Order ID	Customer Name	Product	Quantity
0	166837	Veeru	34in Ultrawide Monitor	2
1	166838	Tarun	Samsung m10	3
2	166839	Kedar	20in Monitor	1
3	166840	Lavanya	iPhone 11	3
4	166841	Venu	Macbook Pro Laptop	2
..
595	167403	Balaji	Macbook Pro Laptop	1
596	167404	Lavanya	ThinkPad Laptop	1
597	167405	Venu	Flatscreen TV	1
598	167406	Siddhu	Samsung m20	2
599	167407	Tarun	LG Washing Machine	1

[600 rows x 4 columns]

Program Name Select specific customer from existing DataFrame
Input file demo6.py
sales1.csv

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")

cust_name = df1["Customer Name"] == "Veeru"

print(df1[cust_name])
```

Output

	Order ID	Customer Name	Product	Quantity
0	166837	Veeru	34in Ultrawide Monitor	2
25	166861	Veeru	Wired Headphones	1
34	166870	Veeru	Wired Headphones	2
61	166897	Veeru	USB-C Charging Cable	2
65	166901	Veeru	Samsung m10	2
77	166913	Veeru	iPhone 9	1
85	166920	Veeru	iPhone 9	2
88	166923	Veeru	27in FHD Monitor	1
113	166948	Veeru	Samsung m10	1
117	166952	Veeru	LG Washing Machine	2
129	166964	Veeru	Samsung m10	1
130	166965	Veeru	Bose SoundSport Headphones	1
134	166968	Veeru	iPhone 7	2
190	167019	Veeru	Macbook Pro Laptop	2
283	167105	Veeru	LG Washing Machine	1
315	167135	Veeru	LG Washing Machine	1
322	167142	Veeru	iPhone 7s	1
326	167146	Veeru	Samsung m20	2

Program Name Select specific customer from existing DataFrame
Input file demo7.py
sales1.csv

```
import pandas as pd

df1 = pd.read_csv("sales1.csv")

prod_name = df1["Product"] == "Macbook Pro Laptop"

print(df1[prod_name])
```

Output

	Order ID	Customer Name	Product	Quantity
4	166841	Venu	Macbook Pro Laptop	2
11	166848	Karteek	Macbook Pro Laptop	1
15	166851	Jaya Chandra	Macbook Pro Laptop	1
71	166907	Karteek	Macbook Pro Laptop	1
122	166957	Harsha	Macbook Pro Laptop	2
136	166970	Tarun	Macbook Pro Laptop	2
156	166986	Pradhan	Macbook Pro Laptop	1
184	167013	Sumanth	Macbook Pro Laptop	1
190	167019	Veeru	Macbook Pro Laptop	2
192	167021	Mallikarjun	Macbook Pro Laptop	1
199	167027	Balaji	Macbook Pro Laptop	1
225	167050	Venu	Macbook Pro Laptop	2
242	167067	Madhurima	Macbook Pro Laptop	2
251	167074	Lavanya	Macbook Pro Laptop	1
261	167084	Shahid	Macbook Pro Laptop	1
285	167107	Jaya Chandra	Macbook Pro Laptop	1
302	167123	Karteek	Macbook Pro Laptop	1
310	167130	Partha	Macbook Pro Laptop	2
332	167152	Venki	Macbook Pro Laptop	1
348	167167	Kedar	Macbook Pro Laptop	2
353	167172	Balaji	Macbook Pro Laptop	1
357	167176	Shahid	Macbook Pro Laptop	2

4. iloc and loc indexers

- ✓ We can select columns from the DataFrame by using iloc and loc.
- ✓ **iloc** and **loc** are called as indexers in DataFrame
- ✓ By using these indexers we can get,
 - Rows and columns of DataFrame
 - Slice of DataFrame

5. iloc[] indexer

- ✓ The iloc is used for indexed-based selection method.
- ✓ We have to pass only integer index to select specific row/column.
- ✓ By using this we can get rows or columns at particular positions in the index (so it only takes integers).
- ✓ This does not include the last element in DataFrame

5.1. iloc[] syntax

iloc[] syntax

- ✓ `df.iloc[<row selection>]`

iloc[] syntax

- ✓ `df.iloc[<row selection>, <column selection>]`

Note on syntax

- ✓ So, according to the syntax there are two arguments,
 - Row selection
 - Column selection

5.2. Creating a DataFrame

- ✓ Whenever we load a csv file then pandas returns the DataFrame

Program Name Loading csv file by using pandas
demo8.py
Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
print(df1)
```

Output

```
   Order id Customer name Customer id          Product name  Product cost
0      192837        Partha            8       Apple iPad 10.2-inch     59000
1      192838        Vinay             10      Flatscreen TV     65999
2      192839       Lavanya            16      20in Monitor     75000
3      192840        Mohan            24    Bose SoundSport Headphones     55000
4      192841        Kedar             2       Google Phone     59000
..        ...
895     193732       Balaji            12      Samsung Galaxy S20     60000
896     193733     Neelima            19      27in 4K Gaming Monitor     75999
897     193734       Siddhu            18       Google Phone     69999
898     193735        Vinay            10      20in Monitor     69999
899     193736  Madhurima            7      27in FHD Monitor     55999

[900 rows x 5 columns]
```

5.3. Selecting single column

- ✓ We can select single or multiple rows by using iloc indexer.
 - If we select **one row or column** then it returns a Series.

Program First **row** of the dataframe
Name demo9.py
Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
s = df1.iloc[0]

print(s)
```

Output

Order id	192837
Customer name	Partha
Customer id	8
Product name	Apple iPad 10.2-inch
Product cost	59000
Name:	0, dtype: object

Program Name Second **row** of the dataframe
demo10.py

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
s = df1.iloc[1]

print(s)
```

Output

```
Order id          192838
Customer name    Vinay
Customer id       10
Product name     Flatscreen TV
Product cost      65999
Name: 1, dtype: object
```

Program Last row of the dataframe

Name demo11.py

Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
s = df1.iloc[-1]

print(s)
```

Output

```
Order id                193736
Customer name           Madhurima
Customer id              7
Product name      27in FHD Monitor
Product cost            55999
Name: 899, dtype: object
```

Program First **row** of the dataframe

Name demo12.py

Input file sales2.csv

```
import pandas as pd

df = pd.read_csv('sales2.csv')
s = df.iloc["Order id"]

print(s)
```

Output

TypeError: Cannot index by location index with a non-integer key

Program Name First **column** of the dataframe
Input file demo13.py
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df2 = df1.iloc[:, 0]

print(df2)
```

Output

```
0      192837
1      192838
2      192839
3      192840
4      192841
...
895    193732
896    193733
897    193734
898    193735
899    193736
Name: Order id, Length: 900, dtype: int64
```

Program Name Second **column** of the dataframe
Input file demo14.py
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df2 = df1.iloc[:, 1]

print(df2)
```

Output

```
0      Partha
1      Vinay
2    Lavanya
3      Mohan
4      Kedar
...
895    Balaji
896  Neelima
897    Siddhu
898    Vinay
899  Madhurima
Name: Customer name, Length: 900, dtype: object
```

Program Name Last column of the dataframe
Input file demo15.py
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df2 = df1.iloc[:, -1]

print(df2)
```

Output

```
0      59000
1      65999
2      75000
3      55000
4      59000
...
895    60000
896    75999
897    69999
898    69999
899    55999
Name: Product cost, Length: 900, dtype: int64
```

5.4. Selecting multiple rows and columns

- ✓ We can select single or multiple rows by using iloc indexer.
 - If we select **multiple rows and columns** by using iloc indexer then returns a DataFrame.

Program First five rows from dataframe

Name demo16.py

Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df2 = df1.iloc[0:5]

print(df2)
```

Output

	Order id	Customer name	Customer id	Product name	Product cost
0	192837	Partha	8	Apple iPad 10.2-inch	59000
1	192838	Vinay	10	Flatscreen TV	65999
2	192839	Lavanya	16	20in Monitor	75000
3	192840	Mohan	24	Bose SoundSport Headphones	55000
4	192841	Kedar	2	Google Phone	59000

Program Name First two columns of the dataframe with all rows
Input file demo17.py
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df2 = df1.iloc[:, 0:2]

print(df2)
```

Output

```
      Order id Customer name
0      192837      Partha
1      192838      Vinay
2      192839     Lavanya
3      192840      Mohan
4      192841      Kedar
..        ...
895    193732      Balaji
896    193733     Neelima
897    193734      Siddhu
898    193735      Vinay
899    193736   Madhurima

[900 rows x 2 columns]
```

Program Name First three columns of the dataframe with all rows
Input file demo18.py
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df2 = df1.iloc[:, 0:3]

print(df2)
```

Output

```
   Order id Customer name Customer id
0      192837        Partha             8
1      192838         Vinay            10
2      192839       Lavanya            16
3      192840        Mohan            24
4      192841        Kedar              2
..        ...
895     193732        Balaji            12
896     193733      Neelima            19
897     193734        Siddhu            18
898     193735         Vinay            10
899     193736    Madhurima              7

[900 rows x 3 columns]
```

Program Name first 5 rows and first 2 columns of the DataFrame
Input file demo19.py
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df2 = df1.iloc[0:5, 0:2]

print(df1)
print()
print(df2)
```

Output

```
   Order id Customer name Customer id      Product name  Product cost
0     192837        Partha          8       Apple iPad 10.2-inch    59000
1     192838        Vinay          10      Flatscreen TV    65999
2     192839      Lavanya          16      20in Monitor    75000
3     192840        Mohan          24  Bose SoundSport Headphones    55000
4     192841        Kedar           2      Google Phone    59000
..      ...
895    193732      Balaji          12  Samsung Galaxy S20    60000
896    193733     Neelima          19  27in 4K Gaming Monitor    75999
897    193734      Siddhu          18      Google Phone    69999
898    193735        Vinay          10      20in Monitor    69999
899    193736  Madhurima          7      27in FHD Monitor    55999

[900 rows x 5 columns]
   Order id Customer name
0     192837        Partha
1     192838        Vinay
2     192839      Lavanya
3     192840        Mohan
4     192841        Kedar
```

Program Name first 5 rows and first 3 columns of the DataFrame
Input file demo20.py
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df2 = df1.iloc[0:5, 0:3]

print(df1)
print()
print(df2)
```

Output

```
   Order id Customer name ... Product name Product cost
0    192837      Partha ...     Apple iPad 10.2-inch    59000
1    192838      Vinay ... Flatscreen TV    65999
2    192839     Lavanya ... 20in Monitor    75000
3    192840      Mohan ... Bose SoundSport Headphones    55000
4    192841      Kedar ... Google Phone    59000
..     ...
895   193732     Balaji ... Samsung Galaxy S20    60000
896   193733    Neelima ... 27in 4K Gaming Monitor    75999
897   193734     Siddhu ... Google Phone    69999
898   193735      Vinay ... 20in Monitor    69999
899   193736  Madhurima ... 27in FHD Monitor    55999

[900 rows x 5 columns]

   Order id Customer name Customer id
0    192837      Partha          8
1    192838      Vinay          10
2    192839     Lavanya         16
3    192840      Mohan          24
4    192841      Kedar           2
```

6. loc[] indexer

- ✓ This indexer used to get rows or columns with particular labels from the index.
- ✓ The loc indexer is label based data selection means we have to pass the name of the row or column which we want to select.
- ✓ The loc indexer can also do boolean selection
- ✓ This includes the last element in DataFrame

Usage

- ✓ We can use loc[] indexer for two purposes,
 - Selecting rows by label/index
 - Selecting rows with a boolean/conditional lookup

Syntax

- ✓ `df.loc[<row selection>, <column selection>]`

6.1. loc[] - Selecting rows by label/index

- ✓ We can set index for the DataFrame by using set_index(p) method.
- ✓ The loc[] indexer directly selects based on index values of any rows.
 - For example, if we set index as Product then we can select the specific product directly.

6.2. Creating a dataframe

- ✓ If we load a csv file in pandas then it returns DataFrame

Program Loading csv file

Name demo21.py

Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv("sales2.csv")
print(df1)
```

Output

	Order id	Customer name	Customer id	Product name	Product cost
0	192837	Partha	8	Apple iPad 10.2-inch	59000
1	192838	Vinay	10	Flatscreen TV	65999
2	192839	Lavanya	16	20in Monitor	75000
3	192840	Mohan	24	Bose SoundSport Headphones	55000
4	192841	Kedar	2	Google Phone	59000
..
895	193732	Balaji	12	Samsung Galaxy S20	60000
896	193733	Neelima	19	27in 4K Gaming Monitor	75999
897	193734	Siddhu	18	Google Phone	69999
898	193735	Vinay	10	20in Monitor	69999
899	193736	Madhurima	7	27in FHD Monitor	55999
[900 rows x 5 columns]					

Program Name Setting index as a Product name and selecting DataFrame
Input file demo22.py
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df1.set_index("Product name", inplace = True)

print(df1)
```

Output

Product name	Order id	Customer name	Customer id	Product cost
Apple iPad 10.2-inch	192837	Partha	8	59000
Flatscreen TV	192838	Vinay	10	65999
20in Monitor	192839	Lavanya	16	75000
Bose SoundSport Headphones	192840	Mohan	24	55000
Google Phone	192841	Kedar	2	59000
...
Samsung Galaxy S20	193732	Balaji	12	60000
27in 4K Gaming Monitor	193733	Neelima	19	75999
Google Phone	193734	Siddhu	18	69999
20in Monitor	193735	Vinay	10	69999
27in FHD Monitor	193736	Madhurima	7	55999

[900 rows x 4 columns]

Program Name Setting index as a Product name and selecting Product
demo23.py
Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df1.set_index("Product name", inplace =True)

a = 'iPhone 9'
df2 = df1.loc[a]
print(df2.head(10))
```

Output

Product name	Order id	Customer name	Customer id	Product cost
iPhone 9	192861	Sagar	6	60000
iPhone 9	192867	Shafi	3	65000
iPhone 9	192891	Sagar	6	50000
iPhone 9	192902	Balaji	12	69999
iPhone 9	192914	Jaya Chandra	21	59000
iPhone 9	192921	Venki	15	50000
iPhone 9	192939	Tarun	11	55999
iPhone 9	192969	Siddhu	18	65000
iPhone 9	192984	Neelima	19	69000
iPhone 9	192998	Vinay	10	75999

Program Name Setting index as a Product name and selecting Product
demo24.py
Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df1.set_index("Product name", inplace = True)

a = 'ThinkPad Laptop'
df2 = df1.loc[a]
print(df2.head(10))
```

Output

Product name	Order id	Customer name	Customer id	Product cost
ThinkPad Laptop	192859	Vijay	9	75000
ThinkPad Laptop	192862	Harsha	5	55000
ThinkPad Laptop	192868	Balaji	12	75999
ThinkPad Laptop	192888	Jaya Chandra	21	75000
ThinkPad Laptop	192915	Balaji	12	50000
ThinkPad Laptop	192974	Siddhu	18	69000
ThinkPad Laptop	192983	Sagar	6	63999
ThinkPad Laptop	192989	Venki	15	65000
ThinkPad Laptop	192999	Mallikarjun	13	69999
ThinkPad Laptop	193014	Sumanth	22	55999

Program Name Setting index as a Product name and selecting two products
Input file demo25.py
sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df1.set_index("Product name", inplace = True)

a = ['iPhone 9', 'iPhone 11']
df2 = df1.loc[a]
print(df2)
```

Output

Product name	Order id	Customer name	Customer id	Product cost
iPhone 9	192861	Sagar	6	60000
iPhone 9	192867	Shafi	3	65000
iPhone 9	192891	Sagar	6	50000
iPhone 9	192902	Balaji	12	69999
iPhone 9	192914	Jaya Chandra	21	59000
...
iPhone 11	193615	Venki	15	69999
iPhone 11	193700	Venki	15	63999
iPhone 11	193706	Madhurima	7	60000
iPhone 11	193716	Harsha	5	69000
iPhone 11	193727	Nireekshan	1	59000

[90 rows x 4 columns]

Program Name Setting index as a Product name and selecting two products
demo26.py
Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df1.set_index("Product name", inplace = True)

a = ['ThinkPad Laptop', '27in FHD Monitor']
df2 = df1.loc[a]
print(df2)
```

Output

	Order id	Customer name	Customer id	Product cost
Product name				
ThinkPad Laptop	192859	Vijay	9	75000
ThinkPad Laptop	192862	Harsha	5	55000
ThinkPad Laptop	192868	Balaji	12	75999
ThinkPad Laptop	192888	Jaya Chandra	21	75000
ThinkPad Laptop	192915	Balaji	12	50000
...
27in FHD Monitor	193587	Jaya Chandra	21	59000
27in FHD Monitor	193631	Karteek	4	55000
27in FHD Monitor	193679	Jaya Chandra	21	63000
27in FHD Monitor	193680	Madhurima	7	63000
27in FHD Monitor	193736	Madhurima	7	55999
[71 rows x 4 columns]				

Program

Setting index as a Product name and selecting a couple of product names with Product cost and Customer id

Name demo27.py
Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df1.set_index("Product name", inplace = True)

a = ['iPhone 9', 'ThinkPad Laptop']
b = ['Product cost', 'Customer id']

df2 = df1.loc[a, b]
print(df2)
```

Output

Product name	Product cost	Customer id
iPhone 9	60000	6
iPhone 9	65000	3
iPhone 9	50000	6
iPhone 9	69999	12
iPhone 9	59000	21
...
ThinkPad Laptop	63000	17
ThinkPad Laptop	50000	18
ThinkPad Laptop	63000	13
ThinkPad Laptop	65999	15
ThinkPad Laptop	59000	17
[82 rows x 2 columns]		

Program

Setting index as a Product name and selecting a couple of product names with Product cost and Customer name

Name demo28.py
Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df1.set_index("Product name", inplace = True)

a = ['iPhone 9', 'ThinkPad Laptop']
b = ['Product cost', 'Customer name']

df2 = df1.loc[a, b]

print(df2)
```

Output

	Product name	cost	Customer name
Product name			
iPhone 9	60000	Sagar	
iPhone 9	65000	Shafi	
iPhone 9	50000	Sagar	
iPhone 9	69999	Balaji	
iPhone 9	59000	Jaya Chandra	
...	
ThinkPad Laptop	63000	Pradhan	
ThinkPad Laptop	50000	Siddhu	
ThinkPad Laptop	63000	Mallikarjun	
ThinkPad Laptop	65999	Venki	
ThinkPad Laptop	59000	Pradhan	
[82 rows x 2 columns]			

Program

Setting index as a Product name and selecting a couple of product names with all columns between Order id and Product cost

Name demo29.py
Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df1.set_index("Product name", inplace = True)

a = ['iPhone 8', 'Google Phone']
df2 = df1.loc[a, 'Order id' : 'Product cost']

print(df2)
```

Output

Product name	Order id	Customer name	Customer id	Product cost
iPhone 8	192865	Siddhu	18	59000
iPhone 8	192874	Sumanth	22	69999
iPhone 8	192878	Jaya Chandra	21	69000
iPhone 8	192881	Lavanya	16	65000
iPhone 8	192897	Chaithanya	14	69999
...
Google Phone	193666	Karteek	4	50000
Google Phone	193672	Karteek	4	75000
Google Phone	193725	Mallikarjun	13	75999
Google Phone	193729	Chaithanya	14	65000
Google Phone	193734	Siddhu	18	69999

[93 rows x 4 columns]

Program

Setting index as a Product name and selecting a couple of product names with all columns between Order id and Customer id

Name demo30.py
Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')
df1.set_index("Product name", inplace = True)

a = ['iPhone 8', 'Google Phone']
df2 = df1.loc[a, 'Order id' : 'Customer id']

print(df2)
```

Output

Product name	Order id	Customer name	Customer id
iPhone 8	192865	Siddhu	18
iPhone 8	192874	Sumanth	22
iPhone 8	192878	Jaya Chandra	21
iPhone 8	192881	Lavanya	16
iPhone 8	192897	Chaithanya	14
...
Google Phone	193666	Karteek	4
Google Phone	193672	Karteek	4
Google Phone	193725	Mallikarjun	13
Google Phone	193729	Chaithanya	14
Google Phone	193734	Siddhu	18
[93 rows x 3 columns]			

6.3. Boolean / Logical indexing

- ✓ Based on conditions we can get rows and columns from Dataframe
- ✓ This is the most commonly used data analysis
- ✓ In most use cases, you will make selections based on the values of different columns in your data set.

Program

Select rows with specific product with all columns between Order id and Product cost

Name demo31.py
Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')

a = df1['Product name'] == 'LG Washing Machine'

df2 = df1.loc[a]

print(df2.head())
```

Output

	Order id	Customer name	Customer id	Product name	Product cost
5	192842	Karteek	4	LG Washing Machine	75000
70	192907	Balaji	12	LG Washing Machine	65999
111	192948	Sagar	6	LG Washing Machine	63000
117	192954	Sagar	6	LG Washing Machine	61000
173	193010	Madhurima	7	LG Washing Machine	65000

Program

Select rows with specific product with all columns between Order id and Product cost

Name demo32.py
Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')

a = df1['Product name'] == 'LG Washing Machine'

df2 = df1.loc[a, 'Order id' : 'Product cost']

print(df2.head())
```

Output

	Order id	Customer name	Customer id	Product name	Product cost
5	192842	Karteek	4	LG Washing Machine	75000
70	192907	Balaji	12	LG Washing Machine	65999
111	192948	Sagar	6	LG Washing Machine	63000
117	192954	Sagar	6	LG Washing Machine	61000
173	193010	Madhurima	7	LG Washing Machine	65000

Program

Select rows with specific Customer with Product name and Product cost columns

Name demo33.py
Input file sales2.csv

```
import pandas as pd

df1 = pd.read_csv('sales2.csv')

a = df1['Customer name'] == 'Sagar'

df2 = df1.loc[a, 'Product name' : 'Product cost']

print(df2.head())
```

Output

	Product name	Product cost
24	iPhone 9	60000
50	Google Phone	60000
54	iPhone 9	50000
111	LG Washing Machine	63000
117	LG Washing Machine	61000

14. PANDAS – DataFrame – Filtering

Contents

1. Filtering	2
1.1. Filtering Examples	2
1.2. We can filter data by using	3
1.3. Creating a DataFrame	3
2. By using relational operators	4
3. Filtering by using loc and iloc	9
3.1. Rows position and column name.....	11
3.2. Selecting multiple values of a column	14
3.3. isin() method.....	15
3.4. unique() method	17
4. Select Non-Missing Data from DataFrame	19

14. PANDAS – DataFrame - Filtering

1. Filtering

- ✓ Filtering the data in DataFrame is very common requirement.
- ✓ It is very important step in Data Analysis project
- ✓ Based on condition we can filter the data from DataFrame

1.1. Filtering Examples

- ✓ Banking
 - Select all the active customers whose accounts were opened after 1st January 2020
 - Get the details of all the customers who made more than 300 transactions in the last 6 months
- ✓ Organization
 - Fetch information of employees who spent more than 3 years in the organization and received highest rating in the past 2 years
- ✓ Telecom
 - Analyse complaints data and identify customers who filed more than 5 complaints in the last 1 year
- ✓ General
 - Extract details of metro cities where per capita income is greater than 40K dollars
- ✓ Many more...

1.2. We can filter data by using

- ✓ By using relational operators.
 - Single condition
 - Multiple condition
- ✓ By using loc & iloc indexers

1.3. Creating a DataFrame

- ✓ We can create DataFrame by loading csv file.

Program Name Loading csv file
Input file demo1.py
sales4.csv

```
import pandas as pd

df = pd.read_csv("sales4.csv")

print(df)
```

Output

```
      Order_Id Customer_Name Customer_Id          Product_Name Product_Cost
0        1023       Venki           15    27in FHD Monitor     59000
1        1024   Chaithanya          14        iPhone 11     69000
2        1025      Shahid           20  Bose SoundSport Headphones     65999
3        1026      Veeru            3      Apple iPad 10.2-inch     63999
4        1027       Venu           23        Google Phone     63999
...
...       ...
299995    301018      Karteek            4      Apple iPad 10.2-inch     51999
299996    301019      Veeru             3      Macbook Pro Laptop     51999
299997    301020      Harsha            5      LG Washing Machine     65000
299998    301021  Nireekshan           1        LG Mobile     60000
299999    301022      Pradhan           17    34in Ultrawide Monitor     55000

[300000 rows x 5 columns]
```

2. By using relational operators

- ✓ By using relational operators we can apply the filter on dataframe.

Program Name Filtering DataFrame by using relational operator: Single condition
demo2.py
Input file sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

con1 = df1['Product_Cost'] > 65000
df2 = df1[con1]

print(df2)
```

Output

```
      Order_Id Customer_Name Customer_Id          Product_Name Product_Cost
1        1024    Chaithanya       14             iPhone 11     69000
2        1025      Shahid        20  Bose SoundSport Headphones     65999
11       1034       Tarun        11      Samsung Galaxy S20     69999
13       1036    Chaithanya       14      LG ThinQ Refrigerator     69999
17       1040    Sumanth         22             iPhone 8     65999
...
...
...
299982    301005       Tarun        11      Samsung Galaxy S20     69000
299983    301006      Shafi        25  27in 4K Gaming Monitor     69000
299985    301008      Venki        15   Apple Airpods Headphones     65999
299988    301011      Venki        15        Google Phone     65999
299990    301013     Balaji        12             iPhone 11     65999

[112280 rows x 5 columns]
```

Program Name Filtering DataFrame by using relational operator: Single condition
demo3.py
Input file sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

con1 = df1['Product_Cost'] > 70000
df2 = df1[con1]

print(df2)
```

Output

```
   Order_Id Customer_Name Customer_Id      Product_Name Product_Cost
25        1048       Partha          8  Samsung Galaxy S9 Plus     75000
28        1051      Lavanya         16    LG Washing Machine     75000
42        1065   Madhurima          7      20in Monitor     75999
45        1068       Vinay          10  Samsung Galaxy S9 Plus     75999
53        1076      Tarun          11    LG Washing Machine     75999
...
...           ...
299955    300978      Siddhu         18  34in Ultrawide Monitor     75000
299963    300986      Shahid         20  34in Ultrawide Monitor     75999
299964    300987      Harsha          5   Macbook Pro Laptop     75000
299965    300988   Madhurima          7   Flatscreen TV     75000
299973    300996      Vinay          10  Samsung Galaxy S9 Plus     75999

[37401 rows x 5 columns]
```

Program

Filtering DataFrame by using relational operator: Multiple conditions

Name demo4.py
Input file sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

con1 = df1['Product_Cost'] > 50000
con2 = df1['Product_Cost'] < 60000

df2 = df1[con1 & con2]

print(df2)
```

Output

```
   Order_Id Customer_Name Customer_Id      Product_Name Product_Cost
0       1023        Venki          15    27in FHD Monitor     59000
8       1031        Kedar           2    20in Monitor     55999
12      1035        Kedar           2      Google Phone     55999
14      1037    Nireekshan          1   Apple Airpods Headphones     55999
22      1045       Lavanya          16      iPhone 7s     51999
...
299993    301016       Siddhu          18      iPhone 9     59000
299994    301017       Kedar           2  ThinkPad Laptop     55999
299995    301018      Karteek          4  Apple iPad 10.2-inch     51999
299996    301019       Veeru           3  Macbook Pro Laptop     51999
299999    301022      Pradhan          17  34in Ultrawide Monitor     55000

[74794 rows x 5 columns]
```

Program

Filtering DataFrame by using relational operator: Multiple conditions

Name demo5.py
Input file sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

con1 = df1.Product_Name == "iPhone 11"
con2 = df1.Customer_Name == "Nireekshan"

df2 = df1[con1 & con2]

print(df2)
```

Output

```
      Order_Id Customer_Name Customer_Id Product_Name Product_Cost
821       1844   Nireekshan          1     iPhone 11      63999
1086      2109   Nireekshan          1     iPhone 11      65000
1529      2552   Nireekshan          1     iPhone 11      55000
1539      2562   Nireekshan          1     iPhone 11      61000
1676      2699   Nireekshan          1     iPhone 11      65000
...
...        ...
296768    297791   Nireekshan          1     iPhone 11      55000
297335    298358   Nireekshan          1     iPhone 11      63999
297717    298740   Nireekshan          1     iPhone 11      50000
297766    298789   Nireekshan          1     iPhone 11      55999
298524    299547   Nireekshan          1     iPhone 11      65999

[581 rows x 5 columns]
```

Program

Filtering DataFrame by using relational operator: Multiple conditions

Name demo6.py
Input file sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

con1 = df1.Product_Name == "iPhone 11"
con2 = df1.Customer_Name == "Shahid"

df2 = df1[con1 & con2]

print(df2)
```

Output

```
      Order_Id Customer_Name Customer_Id Product_Name Product_Cost
26        1049     Shahid         20    iPhone 11      65999
783       1806     Shahid         20    iPhone 11      69000
1260      2283     Shahid         20    iPhone 11      69000
1834      2857     Shahid         20    iPhone 11      65999
1848      2871     Shahid         20    iPhone 11      69999
...
298667    299690     Shahid         20    iPhone 11      65999
298969    299992     Shahid         20    iPhone 11      69000
299206    300229     Shahid         20    iPhone 11      65000
299691    300714     Shahid         20    iPhone 11      63999
299950    300973     Shahid         20    iPhone 11      75000

[594 rows x 5 columns]
```

3. Filtering by using loc and iloc

- ✓ We can filter the dataframe by using loc and iloc indexers as well

Program Name Filtering DataFrame by using loc indexer
Input file demo7.py
sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

con1 = df1.Product_Name == "iPhone 11"
con2 = df1.Customer_Name == "Shahid"

df2 = df1.loc[con1 & con2]

print(df2)
```

Output

```
      Order_Id Customer_Name Customer_Id Product_Name Product_Cost
26        1049     Shahid          20   iPhone 11       65999
783       1806     Shahid          20   iPhone 11       69000
1260      2283     Shahid          20   iPhone 11       69000
1834      2857     Shahid          20   iPhone 11       65999
1848      2871     Shahid          20   iPhone 11       69999
...
...
...
298667    299690     Shahid          20   iPhone 11       65999
298969    299992     Shahid          20   iPhone 11       69000
299206    300229     Shahid          20   iPhone 11       65000
299691    300714     Shahid          20   iPhone 11       63999
299950    300973     Shahid          20   iPhone 11       75000

[594 rows x 5 columns]
```

Program Name Filtering DataFrame by using iloc indexer
Input file demo8.py
sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

df2 = df1.iloc[:5, ]

print(df2)
```

Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
0	1023	Venki	15	27in FHD Monitor	59000
1	1024	Chaitanya	14	iPhone 11	69000
2	1025	Shahid	20	Bose SoundSport Headphones	65999
3	1026	Veeru	3	Apple iPad 10.2-inch	63999
4	1027	Venu	23	Google Phone	63999

3.1. Rows position and column name

- ✓ We can even select the dataframe by providing rows position and column name

Program Filtering DataFrame by using loc indexer
Name demo9.py
Input file sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

rows = df1.index[0:]
cols = ["Product_Name", "Customer_Id"]

df2 = df1.loc[rows, cols]

print(df2)
```

Output

```
          Product_Name Customer_Id
0      27in FHD Monitor        15
1          iPhone 11         14
2    Bose SoundSport Headphones     20
3      Apple iPad 10.2-inch        3
4          Google Phone        23
...
299995      Apple iPad 10.2-inch        4
299996      Macbook Pro Laptop        3
299997      LG Washing Machine        5
299998          LG Mobile         1
299999  34in Ultrawide Monitor       17

[300000 rows x 2 columns]
```

Program Name Filtering DataFrame by using loc indexer
Input file demo10.py
sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

rows = df1.index[0:4]
cols = ["Product_Name", "Customer_Id"]

df2 = df1.loc[rows, cols]

print(df2)
```

Output

	Product_Name	Customer_Id
0	27in FHD Monitor	15
1	iPhone 11	14
2	Bose SoundSport Headphones	20
3	Apple iPad 10.2-inch	3

Program Name Filtering DataFrame by using loc indexer
Input file demo11.py
sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

rows = df1.index[5:]
cols = ["Product_Name", "Customer_Id"]

df2 = df1.loc[rows, cols]

print(df2)
```

Output

```
          Product_Name Customer_Id
5      Samsung Galaxy S9 Plus        6
6                  iPhone 11       23
7      27in FHD Monitor        25
8      20in Monitor            2
9      LG Washing Machine       19
...
299995    Apple iPad 10.2-inch        4
299996      Macbook Pro Laptop        3
299997      LG Washing Machine        5
299998          LG Mobile            1
299999  34in Ultrawide Monitor       17

[299995 rows x 2 columns]
```

3.2. Selecting multiple values of a column

- ✓ We can filter dataframe by providing multiple values of a column

Program Name Filtering DataFrame by using loc indexer
demo12.py
Input file sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

a = df1.Product_Name == "LG Washing Machine"
b = df1.Customer_Id == 1
c = a | b

df2 = df1.loc[c]

print(df2)
```

Output

```
      Order_Id Customer_Name Customer_Id          Product_Name  Product_Cost
9           1032       Neelima        19     LG Washing Machine      63000
14          1037    Nireekshan        1   Apple Airpods Headphones      55999
24           1047        Shafi        25     LG Washing Machine      63999
28           1051      Lavanya        16     LG Washing Machine      75000
39           1062       Tarun        11     LG Washing Machine      61000
...
...
...
299936      300959      Partha         8     LG Washing Machine      65999
299937      300960    Nireekshan        1  27in FHD Monitor      60000
299970      300993    Nireekshan        1      iPhone 8      60000
299997      301020      Harsha        5     LG Washing Machine      65000
299998      301021    Nireekshan        1      LG Mobile      60000
[26278 rows x 5 columns]
```

3.3. isin() method

- ✓ isin() is predefined method in Series class.
- ✓ We should access this method by using Series object.
- ✓ By using this method we can select data from DataFrame

Program Name Filtering DataFrame by using loc isin() method

Input file demo13.py

sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

a = ["Macbook Pro Laptop"]

b = df1.Product_Name.isin(a)

df2 = df1[b]

print(df2)
```

Output

```
   Order_Id Customer_Name  Customer_Id  Product_Name  Product_Cost
23        1046    Madhurima            7  Macbook Pro Laptop      50000
49        1072     Balaji             12  Macbook Pro Laptop      61000
56        1079     Tarun              11  Macbook Pro Laptop      75000
60        1083     Tarun              11  Macbook Pro Laptop     65999
85        1108     Vijay              9  Macbook Pro Laptop      59000
...
...          ...
299906     300929     Partha             8  Macbook Pro Laptop      51999
299907     300930    Neelima            19  Macbook Pro Laptop      63000
299964     300987     Harsha              5  Macbook Pro Laptop      75000
299974     300997  Chaithanya            14  Macbook Pro Laptop     65999
299996     301019     Veeru              3  Macbook Pro Laptop      51999
[15144 rows x 5 columns]
```

Data Science – Pandas – DataFrame - Filtering

Program Name Filtering DataFrame by using loc isin() method
demo14.py
Input file sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")

a = ["34in Ultrawide Monitor", "Macbook Pro Laptop"]

b = df1.Product_Name.isin(a)

df2 = df1[b]

print(df2)
```

Output

```
   Order_Id Customer_Name Customer_Id      Product_Name Product_Cost
23        1046    Madhurima            7     Macbook Pro Laptop      50000
49        1072     Balaji             12     Macbook Pro Laptop      61000
56        1079     Tarun              11     Macbook Pro Laptop      75000
60        1083     Tarun              11     Macbook Pro Laptop      65999
71        1094    Shahid             20  34in Ultrawide Monitor      61000
...
...          ...
299964     300987     Harsha             5     Macbook Pro Laptop      75000
299969     300992     Tarun             11  34in Ultrawide Monitor      65999
299974     300997  Chaithanya            14     Macbook Pro Laptop      65999
299996     301019     Veeru              3     Macbook Pro Laptop      51999
299999     301022    Pradhan             17  34in Ultrawide Monitor      55000

[30299 rows x 5 columns]
```

3.4. unique() function

- ✓ unique() is predefined function in pandas.
- ✓ We should access this function by using pandas module.
- ✓ This function returns the unique values from the column.

Program Selecting unique column values

Name demo15.py

Input file sales4.csv

```
import pandas as pd

df = pd.read_csv("sales4.csv")

a = pd.unique(df.Product_Name)

print(a)
print(len(a))
```

Output

```
['27in FHD Monitor' 'iPhone 11' 'Bose SoundSport Headphones'
 'Apple iPad 10.2-inch' 'Google Phone' 'Samsung Galaxy S9 Plus'
 '20in Monitor' 'LG Washing Machine' 'iPhone 7s' 'Samsung Galaxy S20'
 'LG ThinQ Refrigerator' 'Apple Airpods Headphones' 'iPhone 8'
 'Macbook Pro Laptop' 'LG Mobile' 'ThinkPad Laptop' 'Flatscreen TV'
 '34in Ultrawide Monitor' 'iPhone 9' '27in 4K Gaming Monitor']
20
```

Program Name Selecting unique column values
Input file demo16.py
sales4.csv

```
import pandas as pd

df = pd.read_csv("sales4.csv")

a = pd.unique(df.Customer_Name)

print(a)
print(len(a))
```

Output

```
['Venki' 'Chaithanya' 'Shahid' 'Veeru' 'Venu' 'Daniel' 'Shafi' 'Kedar'
 'Neelima' 'Vijay' 'Tarun' 'Nireekshan' 'Karteek' 'Sumanth' 'Mallikarjun'
 'Vinay' 'Lavanya' 'Madhurima' 'Partha' 'Siddhu' 'Jaya Chandra' 'Balaji'
 'Pradhan' 'Harsha' 'Mohan']
25
```

4. Select Non-Missing Data from DataFrame

notnull() method

- ✓ notnull() is predefined method in Series class.
- ✓ We should access this method by using Series object
- ✓ By using this function we can select the DataFrame which having non NaN values

Program Name Creating a DataFrame
demo17.py

```
import pandas as pd
import numpy as np

data = [
    ['Shahid', 21, 40000],
    ['Nireekshan', 22, 20000],
    ['Veeru', 45, 90000],
    ['Sumanth', 20, 95000],
    [np.nan, 2, 99000],
    ['Prasad', 1, 41000]
]

c = ['Name', 'Age', 'Salary']

df1 = pd.DataFrame(data, columns = c)

print(df1)
```

Output

	Name	Age	Salary
0	Shahid	21	40000
1	Nireekshan	22	20000
2	Veeru	45	90000
3	Sumanth	20	95000
4	NaN	2	99000
5	Prasad	1	41000

Program notnull() method
Name demo18.py

```
import pandas as pd
import numpy as np

data = [
    ['Shahid', 21, 40000],
    ['Nireekshan', 22, 20000],
    ['Veeru', 45, 90000],
    ['Sumanth', 20, 95000],
    [np.nan, 2, 99000],
    ['Prasad', 1, 41000]
]

c = ['Name', 'Age', 'Salary']

df1 = pd.DataFrame(data, columns = c)

d = df1.Name.notnull()

df2 = df1[d]

print(df1)
print()
print(df2)
```

Output

```
Name    Age   Salary
0     Shahid  21    40000
1  Nireekshan  22    20000
2      Veeru  45    90000
3     Sumanth  20    95000
4        NaN   2    99000
5      Prasad   1    41000

Name    Age   Salary
0     Shahid  21    40000
1  Nireekshan  22    20000
2      Veeru  45    90000
3     Sumanth  20    95000
5      Prasad   1    41000
```

15. Pandas – DataFrame – Sorting

Contents

1. Sorting.....	2
1.1. sort_values(by="column name")	3
1.2. sort_index().....	8

15. Pandas – DataFrame – Sorting

1. Sorting

- ✓ DataFrame contains group of rows, column, index and values.
- ✓ Based on requirement we can applying sorting on column name, index etc.

Creating a dataframe:

- ✓ If we load a csv file in pandas then it returns dataframe

Program Loading csv file
Name demo1.py
Input file sales4.csv

```
import pandas as pd

df = pd.read_csv("sales4.csv")
print(df)
```

Output

```
      Order_Id Customer_Name Customer_Id          Product_Name  Product_Cost
0        1023       Venki           15    27in FHD Monitor      59000
1        1024   Chaithanya           14            iPhone 11      69000
2        1025       Shahid           20  Bose SoundSport Headphones      65999
3        1026       Veeru            3        Apple iPad 10.2-inch      63999
4        1027       Venu            23        Google Phone      63999
...
...
...
299995     301018     Karteek           4    Apple iPad 10.2-inch      51999
299996     301019       Veeru            3      Macbook Pro Laptop      51999
299997     301020       Harsha           5      LG Washing Machine      65000
299998     301021   Nireekshan           1        LG Mobile      60000
299999     301022     Pradhan           17  34in Ultrawide Monitor      55000

[300000 rows x 5 columns]
```

1.1. sort_values(by="column name")

- ✓ sort_values(p) is predefined method in DataFrame class.
- ✓ We should access this method by using DataFrame object.
- ✓ This method sort the values based on column which we specified.
 - Number default sorting is ascending order
 - String default sorting is alphabetical order

Program Name Sorting dataframe by using sort_values(p) method
Input file demo2.py
sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")
df2 = df1.sort_values(by = "Product_Cost")

print(df2)
```

Output

```
      Order_Id Customer_Name Customer_Id          Product_Name Product_Cost
52994      54017     Sumanth         22    Samsung Galaxy S9 Plus      50000
122987     124010    Karteek          4   27in FHD Monitor      50000
16661       17684      Venu         23        Google Phone      50000
122986     124009     Harsha          5           iPhone 9      50000
122974     123997    Lavanya         16  34in Ultrawide Monitor      50000
...        ...
273569     274592  Jaya Chandra        21      Macbook Pro Laptop     75999
273565     274588     Partha          8  Apple Airpods Headphones     75999
171470     172493     Veeru          3   27in FHD Monitor     75999
76102       77125     Lavanya         16  Samsung Galaxy S20     75999
76472       77495  Nireekshan          1  Samsung Galaxy S20     75999

[300000 rows x 5 columns]
```

Program Name Sorting dataframe by using sort_values(p) method
Input file demo3.py
 sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")
df2 = df1.sort_values(by = "Customer_Id")

print(df2)
```

Output

```
   Order_Id Customer_Name Customer_Id      Product_Name Product_Cost
160540    161563    Nireekshan          1  27in FHD Monitor     75000
158658    159681    Nireekshan          1    Macbook Pro Laptop    63999
266895    267918    Nireekshan          1      LG Washing Machine    59000
266908    267931    Nireekshan          1        Google Phone     55000
183395    184418    Nireekshan          1   LG ThinQ Refrigerator    69000
...
...
...
...
104589    105612       Shafi           25  LG ThinQ Refrigerator    69999
194864    195887       Shafi           25        iPhone 11     63999
223127    224150       Shafi           25  Samsung Galaxy S20     75999
249626    250649       Shafi           25    Macbook Pro Laptop    61000
109653    110676       Shafi           25  Samsung Galaxy S9 Plus    65999

[300000 rows x 5 columns]
```

Program Name Sorting dataframe by using sort_values(p) method

Input file demo4.py

sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")
df2 = df1.sort_values(by = "Customer_Id", ascending = False)

print(df2)
```

Output

```
   Order_Id Customer_Name Customer_Id      Product_Name Product_Cost
225837    226860        Shafi          25  Apple iPad 10.2-inch     69999
58299     59322        Shafi          25      ThinkPad Laptop     63999
122755    123778        Shafi          25  Samsung Galaxy S20     61000
220596    221619        Shafi          25  Bose SoundSport Headphones     69999
9252      10275        Shafi          25      Flatscreen TV     60000
...
191789    192812  Nireekshan           1      Flatscreen TV     69999
146543    147566  Nireekshan           1        iPhone 7s     60000
283770    284793  Nireekshan           1  Apple Airpods Headphones     69000
79317     80340  Nireekshan           1      20in Monitor     69000
78470     79493  Nireekshan           1        iPhone 7s     61000

[300000 rows x 5 columns]
```

Program Name Sorting dataframe by using sort_values(p) method
Input file demo5.py
 sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")
df2 = df1.sort_values(by = "Customer_Id", ascending = 0)

print(df2)
```

Output

```
   Order_Id Customer_Name Customer_Id      Product_Name Product_Cost
225837    226860        Shafi          25  Apple iPad 10.2-inch     69999
58299     59322        shafi          25       ThinkPad Laptop     63999
122755    123778        Shafi          25  Samsung Galaxy S20     61000
220596    221619        Shafi          25  Bose SoundSport Headphones     69999
9252      10275        Shafi          25      Flatscreen TV     60000
...
191789    192812  Nireekshan           1      Flatscreen TV     69999
146543    147566  Nireekshan           1         iPhone 7s     60000
283770    284793  Nireekshan           1  Apple Airpods Headphones     69000
79317     80340  Nireekshan           1      20in Monitor     69000
78470      79493  Nireekshan           1         iPhone 7s     61000

[300000 rows x 5 columns]
```

Program Name Sorting dataframe by using sort_values() method
Input file demo6.py
 sales4.csv

```
import pandas as pd

df1 = pd.read_csv("sales4.csv")
df2 = df1.sort_values(by = "Customer_Name")

print(df2)
```

Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
136196	137219	Balaji	12	LG Mobile	60000
128754	129777	Balaji	12	Samsung Galaxy S20	61000
292113	293136	Balaji	12	LG Mobile	65999
128753	129776	Balaji	12	LG ThinQ Refrigerator	65999
73777	74800	Balaji	12	Apple iPad 10.2-inch	51999
...
280203	281226	Vinay	10	iPhone 8	65999
155180	156203	Vinay	10	Samsung Galaxy S9 Plus	65000
82486	83509	Vinay	10	LG ThinQ Refrigerator	59000
26986	28009	Vinay	10	iPhone 11	59000
186628	187651	Vinay	10	LG ThinQ Refrigerator	51999

[300000 rows x 5 columns]

1.2. sort_index()

- ✓ sort_index() is predefined method in DataFrame class.
- ✓ We should access this method by using DataFrame object.
- ✓ This method sort the indexes in DataFrame

Program Name Sorting dataframe by using sort_index()
demo7.py

```
import pandas as pd

d = {
    'Order id': [11, 21, 31],
    'Customer name': ['Kedar', 'Nireekshan', 'Daniel'],
    'Product': ['iPhone 11','hTC', 'macbook']
}

i = [555, 444, 333]

df1 = pd.DataFrame(d, index = i)
df2 = df1.sort_index()

print(df1)
print()
print(df2)
```

Output

	Order	id	Customer name	Product
555		11	Kedar	iPhone 11
444		21	Nireekshan	hTC
333		31	Daniel	macbook

	Order	id	Customer name	Product
333		31	Daniel	macbook
444		21	Nireekshan	hTC
555		11	Kedar	iPhone 11

16. Pandas – DataFrame – Groupby

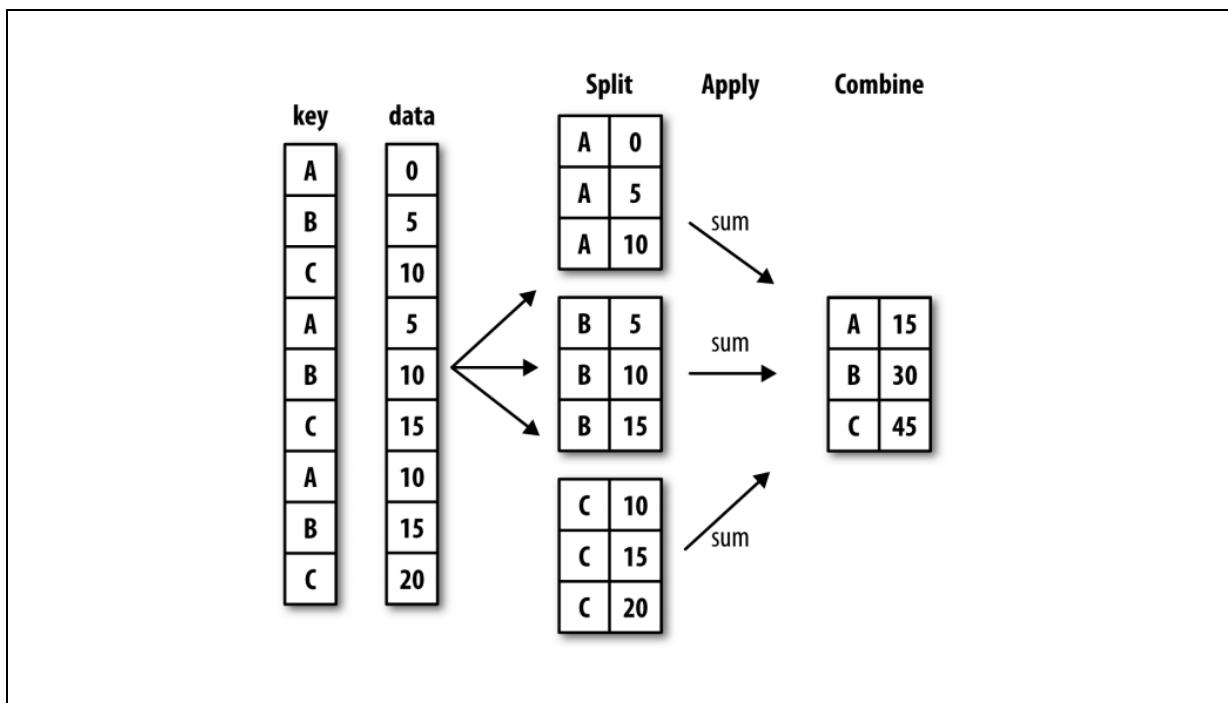
Contents

1. Groupby Introduction	2
2. groupby(p)	3

16. Pandas – DataFrame - Groupby

1. Groupby Introduction

- ✓ Groupby is very common and important operations in Data analysis,
- ✓ There are mainly 3 steps in Groupby operation,
 - Splitting the data into groups based on some criteria.
 - Applying operations to each group independently.
 - Combining the results.



- ✓ For example,
 - If we apply groupby on Product_Name then it groups the data into based on name of the product.
- ✓ The groupby(p) method returns a GroupBy object.

2. groupby(p)

- ✓ groupby(p) is predefined method in DataFrame.
- ✓ We should access this method by using DataFrame object.
- ✓ This method returns GroupBy object.

Program Name Creating products DataFrame
demo1.py

```
import pandas as pd

d = {
    "Product": ["Samsung", "Nokia", "Samsung", "Motorola",
    "Nokia", "Samsung", "Samsung"],

    "Orders": [2, 4, 3, 4, 6, 7, 3]
}

df1 = pd.DataFrame(d)

print(df1)
```

Output

	Product	Orders
0	Samsung	2
1	Nokia	4
2	Samsung	3
3	Motorola	4
4	Nokia	6
5	Samsung	7
6	Samsung	3

Program Name

Creating products DataFrame, applying groupby
demo2.py

```
import pandas as pd

d = {
    "Product": ["Samsung", "Nokia", "Samsung", "Motorola",
    "Nokia", "Samsung", "Samsung"],

    "Orders": [2, 4, 3, 4, 6, 7, 3]
}

df1 = pd.DataFrame(d)

grouped = df1.groupby(["Product"])
result = grouped.sum()

print(df1)
print()
print(result)
```

Output

```
      Product  Orders
0     Samsung      2
1       Nokia      4
2     Samsung      3
3  Motorola      4
4       Nokia      6
5     Samsung      7
6     Samsung      3

      Orders
Product
Motorola      4
Nokia        10
Samsung      15
```

Program Name Get the number of product on dates
Input file demo3.py
sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")
grouped = df1.groupby(["Mail_Id"])
result = grouped.size()

print(result)
```

Output

```
Mail_Id
daniel@gmail.com      3
kedar@gmail.com       3
nirekshan@gmail.com   4
partha@gmail.com      1
prasad@gmail.com      3
shahid@gmail.com      1
dtype: int64
```

Program Name Get the number of product on count
Input file demo4.py
sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")
grouped = df1.groupby(["Product_Name"])
result = grouped.size()

print(result)
```

Output

```
Product_Name
Kindle Paper White      3
LG Washing Machine     1
Samsung                 3
Sofa set                1
hTC mobile               2
iPad                     1
iPhone 8                  3
iPhone 9                  1
dtype: int64
```

Program Name How many products sold out on each month?
Input file demo5.py
sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")

cols = ['Date', 'Product_Name']
grouped = df1.groupby(cols)['Date']
result = grouped.count()

print(result)
```

Output

```
Date      Product_Name
09/01/2019  iPad          1
09/02/2019  LG Washing Machine  1
09/06/2019  Kindle Paper White  2
              Sofa set        1
              iPhone 8       1
10/31/2019  Kindle Paper White  1
11/02/2019  Samsung        2
              hTC mobile     1
11/06/2019  Samsung        1
              iPhone 8       2
              iPhone 9       1
11/10/2019  hTC mobile     1
Name: Date, dtype: int64
```

Program Name Product wise each month sales
Input file demo6.py
sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")

cols = ['Product_Name', 'Date']
grouped = df1.groupby(cols)['Date']
result = grouped.count()

print(result)
```

Output

```
Product_Name      Date
Kindle Paper White 09/06/2019    2
                  10/31/2019    1
LG Washing Machine 09/02/2019    1
Samsung           11/02/2019    2
                  11/06/2019    1
Sofa set          09/06/2019    1
hTC mobile         11/02/2019    1
                  11/10/2019    1
iPad              09/01/2019    1
iPhone 8           09/06/2019    1
                  11/06/2019    2
iPhone 9            11/06/2019    1
Name: Date, dtype: int64
```

Program Name Customer wise each month sales
Input file demo7.py
sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")

cols = ['Mail_Id', 'Date']
grouped = df1.groupby(cols)['Mail_Id']
result = grouped.count()

print(result)
```

Output

```
Mail_Id          Date
daniel@gmail.com 09/06/2019    2
                  10/31/2019   1
kedar@gmail.com  09/06/2019    1
                  11/06/2019   2
nirekshan@gmail.com 09/06/2019  1
                      11/02/2019  2
                      11/06/2019  1
partha@gmail.com  11/06/2019  1
prasad@gmail.com  09/02/2019  1
                      11/02/2019  1
                      11/10/2019  1
shahid@gmail.com  09/01/2019  1
Name: Mail_Id, dtype: int64
```

Program Name Customer wise product details
Input file demo8.py
sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")

col = ['Mail_Id', 'Product_Name']
grouped = df1.groupby(col)['Product_Cost']
result = grouped.sum()

print(result)
```

Output

```
Mail_Id          Product_Name
Daniel@gmail.com Kindle Paper White    20000
                  Sofa set            50000
kedar@gmail.com   iPhone 8           60000
nirekshan@gmail.com Kindle Paper White    20000
                      Samsung          30000
partha@gmail.com  iPhone 9           30000
prasad@gmail.com  LG Washing Machine  25000
                      hTC mobile        30000
shahid@gmail.com iPad                70000
Name: Product_Cost, dtype: int64
```

Program Name Customer wise product details
Input file demo9.py
sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")

col = ['Mail_Id', 'Product_Name']
grouped = df1.groupby(col, as_index = False)['Product_Cost']
result = grouped.sum()

print(result)
```

Output

	Mail_Id	Product_Name	Product_Cost
0	Daniel@gmail.com	Kindle Paper White	20000
1	Daniel@gmail.com	Sofa set	50000
2	kedar@gmail.com	iPhone 8	60000
3	nirekshan@gmail.com	Kindle Paper White	20000
4	nirekshan@gmail.com	Samsung	30000
5	partha@gmail.com	iPhone 9	30000
6	prasad@gmail.com	LG Washing Machine	25000
7	prasad@gmail.com	hTC mobile	30000
8	shahid@gmail.com	iPad	70000

Program Customer wise product sales

Name demo10.py

Input file sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")
col = ['Mail_Id']
grouped = df1.groupby(col)['Product_Cost']
result = grouped.sum()

print(result)
```

Output

```
Mail_Id
Daniel@gmail.com      70000
kedar@gmail.com       60000
nirekshan@gmail.com   50000
partha@gmail.com      30000
prasad@gmail.com      55000
shahid@gmail.com      70000
Name: Product_Cost, dtype: int64
```

Program Name Product wise whole sales

Name demo11.py

Input file sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")
col = ['Product_Name']
grouped = df1.groupby(col)['Product_Cost']
result = grouped.sum()

print(result)
```

Output

```
Product_Name
Kindle Paper White      40000
LG Washing Machine     25000
Samsung                 30000
Sofa set                50000
hTC mobile               30000
iPad                      70000
iPhone 8                  60000
iPhone 9                  30000
Name: Product_Cost, dtype: int64
```

Program Name Day wise product sales

Name demo12.py

Input file sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")
col = ['Date']
grouped = df1.groupby(col)['Product_Cost']
result = grouped.sum()

print(result)
```

Output

```
Date
09/01/2019      70000
09/02/2019      25000
09/06/2019     100000
10/31/2019      10000
11/02/2019      35000
11/06/2019      80000
11/10/2019      15000
Name: Product_Cost, dtype: int64
```

Program Describe method

Name demo13.py

Input file sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")
df2 = df1['Product_Cost'].describe()

print(df2)
```

Output

```
count      15.000000
mean      22333.333333
std       16888.993316
min      10000.000000
25%      10000.000000
50%      20000.000000
75%      22500.000000
max      70000.000000
Name: Product_Cost, dtype: float64
```

Program Name Applying a single function to columns in groups
Input file demo14.py
sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")

d = {
    'Product_Cost' : sum
}

cols = ['Date', 'Product_Name']
grouped = df1.groupby(cols)
result = grouped.agg(d)

print(result)
```

Output

Date	Product_Name	Product_Cost
09/01/2019	iPad	70000
09/02/2019	LG Washing Machine	25000
09/06/2019	Kindle Paper White	30000
	Sofa set	50000
	iPhone 8	20000
10/31/2019	Kindle Paper White	10000
11/02/2019	Samsung	20000
	hTC mobile	15000
11/06/2019	Samsung	10000
	iPhone 8	40000
	iPhone 9	30000
11/10/2019	hTC mobile	15000

Program Name Applying a single function to columns in groups
Input file demo15.py
sales5.py

```
import pandas as pd

df1 = pd.read_csv("sales5.csv")

d = {
    'Product_Cost': sum,
    'Product_Name': "count",
}

cols = ['Date', 'Product_Name']

grouped = df1.groupby(cols)
result = grouped.agg(d)

print(result)
```

Output

Date	Product_Name	Product_Cost	Product_Name
09/01/2019	iPad	70000	1
09/02/2019	LG Washing Machine	25000	1
09/06/2019	Kindle Paper White	30000	2
	Sofa set	50000	1
	iPhone 8	20000	1
10/31/2019	Kindle Paper White	10000	1
11/02/2019	Samsung	20000	2
	hTC mobile	15000	1
11/06/2019	Samsung	10000	1
	iPhone 8	40000	2
	iPhone 9	30000	1
11/10/2019	hTC mobile	15000	1

Program Name Applying a single function to columns in groups
Input file demo16.py
sales5.py

```
import pandas as pd

df1 = pd.read_csv('sales5.csv')

d = {
    'Product_Cost': [min, max, sum]
}

col = ['Product_Cost']
grouped = df1.groupby(col)
result = grouped.agg(d)

print(result)
```

Output

	Product_Cost	min	max	sum
Product_Cost				
10000		10000	10000	50000
15000		15000	15000	30000
20000		20000	20000	80000
25000		25000	25000	25000
30000		30000	30000	30000
50000		50000	50000	50000
70000		70000	70000	70000

17. Pandas – DataFrame - Merging or Joining

Contents

1. Introduction.....	2
2. merge(p1, p2, p3, p4) function	2
3. Types of joins.....	2
3.1. Inner join	3
3.2. Left join	8
3.3. Right join	11
3.4. Outer Merge / Full outer join.....	14
4. Other type of joins	17
4.1. One to one	18
4.2. Many to one.....	20
4.3. Many to many	22

17. Pandas – DataFrame - Merging or Joining

1. Introduction

- ✓ By using pandas we can perform join operations as well.
- ✓ This is same as join operations in database.
- ✓ Here we are performing join in between the DataFrames
- ✓ Joining is the process of bringing two DataFrames into one DataFrame based on common attributes in columns

2. merge(p1, p2, p3, p4) function

- ✓ merge(p1, p2, p3, p4) is a predefined function in pandas.
 - pd.merge(df1, df2, on = "column", how = "type of join")
- ✓ We should access this function by using pandas library
- ✓ By using this function we can perform join operations over DataFrames

'how' Argument

- ✓ The how argument helps to specify the type of join.

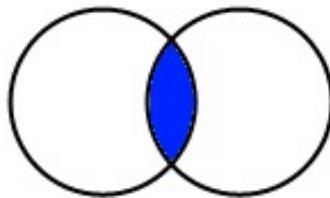
3. Types of joins

- ✓ Inner Merge / Inner join
- ✓ Left Merge / Left outer join
- ✓ Right Merge / Right outer join
- ✓ Outer Merge / Full outer join

3.1. Inner join

- ✓ We can perform inner join on DataFrames.
- ✓ This inner join is a kind of intersection means, it keeps the common data.

INNER JOIN



`merge(df1, df2, on = "column", how = "type")`

- ✓ `merge(df1, df2, on = "column", how = "type")` is predefined function in pandas.
- ✓ To perform inner join, we need to pass how value as "inner" as per the syntax

Syntax

```
pd.merge(df1, df2, on = "column", how = "inner")
```

Program Name Creating two DataFrames
demo1.py

```
import pandas as pd

d1 = {
    "Id": [1, 2, 3, 4, 5, 6],
    "Name": ["Pradhan", "Venu", "Madhurima", "Nireekshan",
    "Shafi", "Veelu"],
    "Subject": ["English", "Java", "Html", "Python", "C", "dot
    net"]
}

d2 = {
    "Id": [11, 12, 13, 14, 15, 16],
    "Name": ["Srinu", "Sumanth", "Neelima", "Daniel", "Arjun",
    "Veelu"],
    "Subject": ["Java", "Html", "Cpp", "Python", "C", "dot net"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

print(df1)
print()
print(df2)
```

Output

```
      Id      Name  Subject
0    1    Pradhan  English
1    2      Venu      Java
2    3  Madhurima     Html
3    4  Nireekshan   Python
4    5      Shafi        C
5    6      Veeru  dot net

      Id      Name  Subject
0   11      Srinu      Java
1   12  Sumanth     Html
2   13  Neelima      Cpp
3   14    Daniel   Python
4   15    Arjun        C
5   16      Veeru  dot net
```

Program Inner Join
Name demo2.py

```
import pandas as pd

d1 = {
    "Id": [1, 2, 3, 4, 5, 6],
    "Name": ["Pradhan", "Venu", "Madhurima", "Nireekshan",
    "Shafi", "Veeru"],
    "Subject": ["English", "Java", "Html", "Python", "C", "dot
    net"]
}

d2 = {
    "Id": [11, 12, 13, 14, 15, 16],
    "Name": ["Srinu", "Sumanth", "Neelima", "Daniel", "Arjun",
    "Veeru"],
    "Subject": ["Java", "Html", "Cpp", "Python", "C", "dot net"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

inn_join = pd.merge(df1, df2, on = "Subject", how = "inner")

print(df1)
print()
print(df2)
print()
print(inn_join)
```

Output

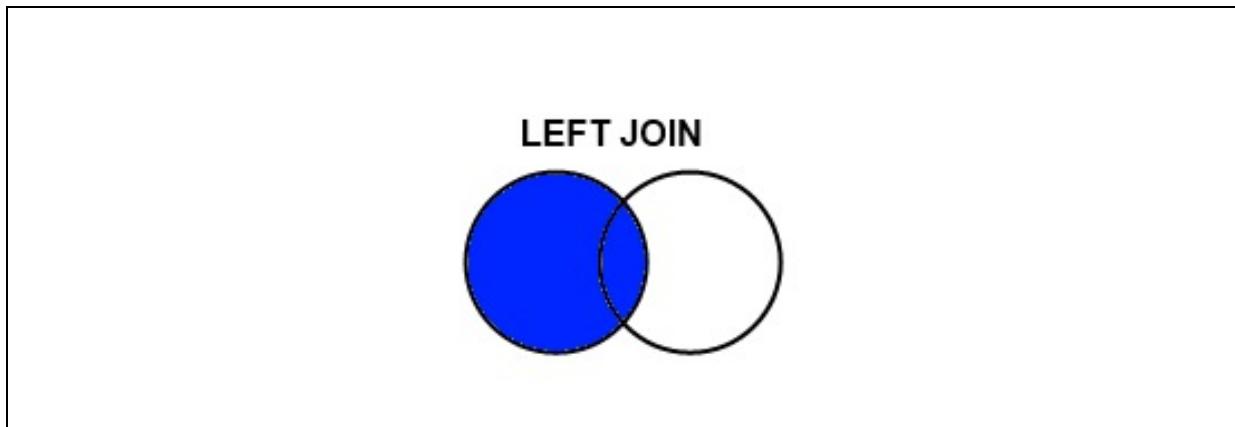
	Id	Name	Subject
0	1	Pradhan	English
1	2	Venu	Java
2	3	Madhurima	Html
3	4	Nireekshan	Python
4	5	Shafi	C
5	6	Veeru	dot net

	Id	Name	Subject
0	11	Srinu	Java
1	12	Sumanth	Html
2	13	Neelima	Cpp
3	14	Daniel	Python
4	15	Arjun	C
5	16	Veeru	dot net

	Id_x	Name_x	Subject	Id_y	Name_y
0	2	Venu	Java	11	Srinu
1	3	Madhurima	Html	12	Sumanth
2	4	Nireekshan	Python	14	Daniel
3	5	Shafi	C	15	Arjun
4	6	Veeru	dot net	16	Veeru

3.2. Left join

- ✓ Keep every row in the left dataframe.
- ✓ Where there are missing values of the "on" variable in the right dataframe filled with NaN values in the result.



Program Name Left Join
demo3.py

```
import pandas as pd

d1 = {
    "Id": [1, 2, 3, 4, 5, 6],
    "Name": ["Pradhan", "Venu", "Madhurima", "Nireekshan",
              "Shafi", "Veetu"],
    "Subject": ["English", "Java", "Html", "Python", "C", "dot
                net"]
}

d2 = {
    "Id": [11, 12, 13, 14, 15, 16],
    "Name": ["Srinu", "Sumanth", "Neelima", "Daniel", "Arjun",
              "Veetu"],
    "Subject": ["Java", "Html", "Cpp", "Python", "C", "dot net"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

left_join = pd.merge(df1, df2, on = "Subject", how = "left")

print(df1)
print()
print(df2)
print()
print(left_join)
```

Output

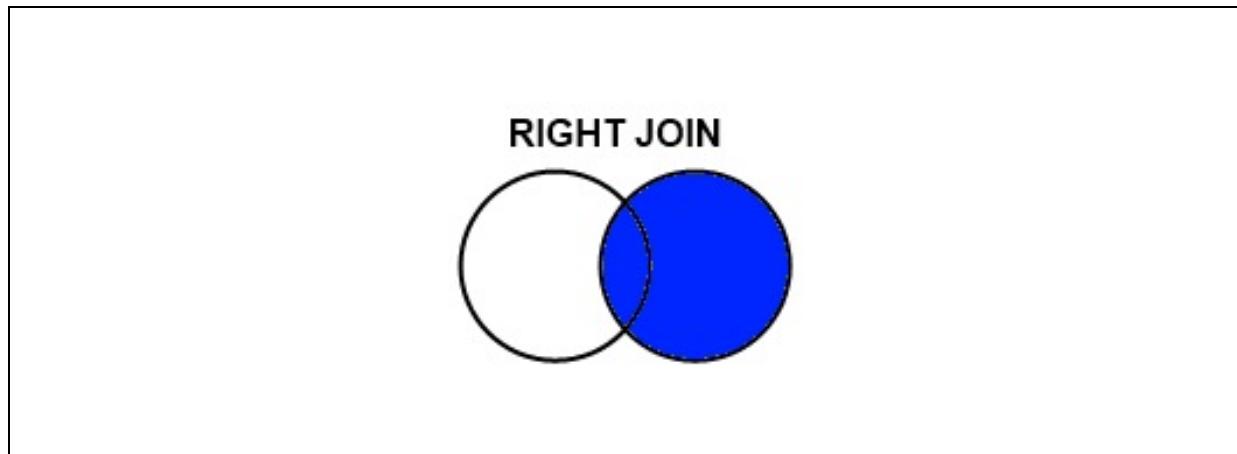
```
      Id      Name  Subject
0    1    Pradhan  English
1    2      Venu     Java
2    3  Madhurima    Html
3    4  Nireekshan  Python
4    5      Shafi       C
5    6      Veeru  dot net

      Id      Name  Subject
0   11      Srinu     Java
1   12  Sumanth    Html
2   13  Neelima     Cpp
3   14    Daniel  Python
4   15      Arjun       C
5   16      Veeru  dot net

  Id_x      Name_x  Subject  Id_y      Name_y
0    1    Pradhan  English  NaN        NaN
1    2      Venu     Java  11.0     Srinu
2    3  Madhurima    Html  12.0    Sumanth
3    4  Nireekshan  Python  14.0    Daniel
4    5      Shafi       C  15.0    Arjun
5    6      Veeru  dot net  16.0    Veeru
```

3.3. Right join

- ✓ Keep every row in the right dataframe.
- ✓ Where there are missing values of the "on" variable in the left column filled with NaN values in the result.



Program Name Right Join
demo4.py

```
import pandas as pd

d1 = {
    "Id": [1, 2, 3, 4, 5, 6],
    "Name": ["Pradhan", "Venu", "Madhurima", "Nireekshan",
              "Shafi", "Veetu"],
    "Subject": ["English", "Java", "Html", "Python", "C", "dot
                net"]
}

d2 = {
    "Id": [11, 12, 13, 14, 15, 16],
    "Name": ["Srinu", "Sumanth", "Neelima", "Daniel", "Arjun",
              "Veetu"],
    "Subject": ["Java", "Html", "Cpp", "Python", "C", "dot net"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

right_join = pd.merge(df1, df2, on = "Subject", how = "right")

print(df1)
print()
print(df2)
print()
print(right_join)
```

Output

```
      Id      Name  Subject
0    1    Pradhan  English
1    2      Venu      Java
2    3  Madhurima     Html
3    4  Nireekshan   Python
4    5      Shafi        C
5    6      Veeru  dot net

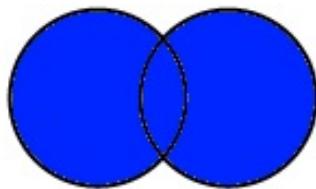
      Id      Name  Subject
0   11    Srinu      Java
1   12  Sumanth     Html
2   13  Neelima      Cpp
3   14  Daniel   Python
4   15    Arjun        C
5   16    Veeru  dot net

  Id_x      Name_x  Subject  Id_y      Name_y
0  2.0        Venu      Java    11    Srinu
1  3.0  Madhurima     Html    12  Sumanth
2  NaN        NaN      Cpp    13  Neelima
3  4.0  Nireekshan   Python    14  Daniel
4  5.0      Shafi        C    15    Arjun
5  6.0      Veeru  dot net    16    Veeru
```

3.4. Outer Merge / Full outer join

- ✓ A full outer join returns all the rows from the left and right DataFrames.
- ✓ Where there is no common data there it will be filled with NaN values.

FULL OUTER JOIN



Program Name Outer Join
demo5.py

```
import pandas as pd

d1 = {
    "Id": [1, 2, 3, 4, 5, 6],
    "Name": ["Pradhan", "Venu", "Madhurima", "Nireekshan",
              "Shafi", "Veetu"],
    "Subject": ["English", "Java", "Html", "Python", "C", "dot
                net"]
}

d2 = {
    "Id": [11, 12, 13, 14, 15, 16],
    "Name": ["Srinu", "Sumanth", "Neelima", "Daniel", "Arjun",
              "Veetu"],
    "Subject": ["Java", "Html", "Cpp", "Python", "C", "dot net"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

outer_join = pd.merge(df1, df2, on = "Subject", how = "outer")

print(df1)
print()
print(df2)
print()
print(outer_join)
```

Output

	Id	Name	Subject
0	1	Pradhan	English
1	2	Venu	Java
2	3	Madhurima	Html
3	4	Nireekshan	Python
4	5	Shafi	C
5	6	Veeru	dot net

	Id	Name	Subject
0	11	Srinu	Java
1	12	Sumanth	Html
2	13	Neelima	Cpp
3	14	Daniel	Python
4	15	Arjun	C
5	16	Veeru	dot net

	Id_x	Name_x	Subject	Id_y	Name_y
0	1.0	Pradhan	English	NaN	NaN
1	2.0	Venu	Java	11.0	Srinu
2	3.0	Madhurima	Html	12.0	Sumanth
3	4.0	Nireekshan	Python	14.0	Daniel
4	5.0	Shafi	C	15.0	Arjun
5	6.0	Veeru	dot net	16.0	Veeru
6	NaN	NaN	Cpp	13.0	Neelima

4. Other type of joins

- ✓ One to one
- ✓ Many to one
- ✓ Many to many

Program Name Creating DataFrames
demo6.py

```
import pandas as pd

d1 = {
    "Employee": ["Nireekshan", "Veeru", "Lavanya", "Pradhan"],
    "Group": ["Development", "Testing", "Testing", "HR"]
}

d2 = {
    "Employee": ["Lavanya", "Nireekshan", "Veeru", "Pradhan"],
    "Hire_date": [2010, 2012, 2014, 2016]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

print(df1)
print()
print(df2)
```

Output

```
      Employee      Group
0  Nireekshan  Development
1      Veeru        Testing
2      Lavanya        Testing
3      Pradhan            HR

      Employee  Hire_date
0      Lavanya      2010
1  Nireekshan      2012
2      Veeru      2014
3      Pradhan      2016
```

4.1. One to one

- ✓ This is very simple join and similar to the column-wise concatenation

Program Name	Creating DataFrames demo7.py
---------------------	---------------------------------

```
import pandas as pd

d1 = {
    "Employee": ["Nireekshan", "Veeru", "Lavanya", "Pradhan"],
    "Group": ["Development", "Testing", "Testing", "HR"]
}

d2 = {
    "Employee": ["Lavanya", "Nireekshan", "Veeru", "Pradhan"],
    "Hire_date": [2010, 2012, 2014, 2016]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

one_one = pd.merge(df1, df2)

print(df1)
print()
print(df2)
print()
print(one_one)
```

Output

```
      Employee      Group
0   Nireekshan  Development
1       Veeru        Testing
2     Lavanya        Testing
3     Pradhan          HR

      Employee  Hire_date
0     Lavanya      2010
1  Nireekshan      2012
2       Veeru      2014
3     Pradhan      2016

      Employee      Group  Hire_date
0   Nireekshan  Development      2012
1       Veeru        Testing      2014
2     Lavanya        Testing      2010
3     Pradhan          HR        2016
```

4.2. Many to one

- ✓ Many-to-one joins are joins in which one of the two key columns contains duplicate entries

Program Name

Many to one

demo8.py

```
import pandas as pd

d1 = {
    "Employee": ["Nireekshan", "Veeru", "Lavanya", "Pradhan"],
    "Group": ["Development", "Testing", "Testing", "HR"]
}

d2 = {
    "Employee": ["Lavanya", "Nireekshan", "Veeru", "Pradhan"],
    "Hire_date": [2010, 2012, 2014, 2016]
}

d3 = {
    "Group": ["Testing", "Development", "HR"],
    "supervisor": ["Shafi", "Daniel", "Neelima"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)
df3 = pd.DataFrame(d3)

one_one = pd.merge(df1, df2)

many_one = pd.merge(one_one, df3)

print(df1)
print()
print(df2)
print()
print(many_one)
```

Output

```
      Employee      Group
0  Nireekshan  Development
1      Veeru        Testing
2    Lavanya        Testing
3    Pradhan            HR

      Employee  Hire_date
0      Lavanya     2010
1  Nireekshan     2012
2      Veeru     2014
3    Pradhan     2016

      Employee      Group  Hire_date supervisor
0  Nireekshan  Development     2012      Daniel
1      Veeru        Testing     2014       Shafi
2    Lavanya        Testing     2010       Shafi
3    Pradhan            HR     2016   Neelima
```

4.3. Many to many

- ✓ If the key column in both the left and right DataFrame contains duplicates, then the result is a many-to-many merge

Program

Many to many

Name

demo9.py

```
import pandas as pd

d1 = {
    "Employee": ["Nireekshan", "Veeru", "Lavanya", "Pradhan"],
    "Group": ["Development", "Testing", "Testing", "HR"]
}

d2 = {
    "Group": ["Testing", "Testing", "Development",
              "Development", "HR", "HR"],
    "Skills": ["Manual", "Automation", "Coding", "Logical",
               "Spreadsheets", "Organization"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

many_many = pd.merge(df1, df2)

print(df1)
print()
print(df2)
print()
print(many_many)
```

Output

```
      Employee      Group
0  Nireekshan  Development
1      Veeru      Testing
2    Lavanya      Testing
3    Pradhan          HR

      Group      Skills
0  Testing      Manual
1  Testing  Automation
2 Development      Coding
3 Development      Logical
4          HR  Spreadsheets
5          HR  Organization

      Employee      Group      Skills
0  Nireekshan  Development      Coding
1  Nireekshan  Development      Logical
2      Veeru      Testing      Manual
3      Veeru      Testing  Automation
4    Lavanya      Testing      Manual
5    Lavanya      Testing  Automation
6    Pradhan          HR  Spreadsheets
7    Pradhan          HR  Organization
```

Based on column wise

- ✓ We can also do merge DataFrames based on columns wise as well

Program Name	Creating DataFrames demo10.py
---------------------	----------------------------------

```
import pandas as pd

d1 = {
    "Id": [1, 2, 3, 4, 5, 6],
    "Name": ["Pradhan", "Venu", "Madhurima", "Nireekshan",
              "Shafi", "Veeru"],
    "Subject": ["English", "Java", "Html", "Python", "C", "dot
                net"]
}

d2 = {
    "Id": [11, 12, 13, 14, 15, 16],
    "Name": ["Srinu", "Sumanth", "Neelima", "Daniel", "Arjun",
              "Veeru"],
    "Subject": ["Java", "Html", "Cpp", "Python", "C", "dot net"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

print(df1)
print()
print(df2)
```

Output

```
      Id      Name  Subject
0    1    Pradhan  English
1    2        Venu     Java
2    3  Madhurima     Html
3    4   Nireekshan  Python
4    5       Shafi        C
5    6       Veeru  dot net

      Id      Name  Subject
0   11      Srinu     Java
1   12  Sumanth     Html
2   13   Neelima      Cpp
3   14     Daniel  Python
4   15     Arjun        C
5   16     Veeru  dot net
```

Program Name Merging two DataFrames based on column
demo11.py

```
import pandas as pd

d1 = {
    "Id": [1, 2, 3, 4, 5, 6],
    "Name": ["Pradhan", "Venu", "Madhurima", "Nireekshan",
    "Shafi", "Veeru"],
    "Subject": ["English", "Java", "Html", "Python", "C", "dot
    net"]
}

d2 = {
    "Id": [11, 12, 13, 14, 15, 16],
    "Name": ["Srinu", "Sumanth", "Neelima", "Daniel", "Arjun",
    "Veeru"],
    "Subject": ["Java", "Html", "Cpp", "Python", "C", "dot net"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

result = pd.merge(df1, df2, on = 'Subject')

print(df1)
print()
print(df2)
print()
print(result)
```

Output

```
      Id      Name  Subject
0    1    Pradhan   English
1    2        Venu      Java
2    3  Madhurima     Html
3    4  Nireekshan   Python
4    5        Shafi       C
5    6        Veeru  dot net

      Id      Name  Subject
0   11      Srinu      Java
1   12  Sumanth     Html
2   13  Neelima      Cpp
3   14    Daniel   Python
4   15    Arjun       C
5   16    Veeru  dot net

  Id_x      Name_x  Subject  Id_y      Name_y
0    2        Venu      Java    11      Srinu
1    3  Madhurima     Html    12  Sumanth
2    4  Nireekshan   Python    14    Daniel
3    5        Shafi       C    15    Arjun
4    6        Veeru  dot net    16    Veeru
```

Program Name Merging two DataFrames with multiple columns
demo12.py

```
import pandas as pd

d1 = {
    "Id": [1, 2, 3, 4, 5, 6],
    "Name": ["Pradhan", "Venu", "Madhurima", "Nireekshan",
    "Shafi", "Veeru"],
    "Subject": ["English", "Java", "Html", "Python", "C", "dot
    net"]
}

d2 = {
    "Id": [11, 12, 13, 14, 15, 16],
    "Name": ["Srinu", "Sumanth", "Neelima", "Daniel", "Arjun",
    "Veeru"],
    "Subject": ["Java", "Html", "Cpp", "Python", "C", "dot net"]
}

df1 = pd.DataFrame(d1)
df2 = pd.DataFrame(d2)

result = pd.merge(df1, df2, on=["Name", "Subject"])

print(df1)
print()
print(df2)
print()
print(result)
```

Output

```
      Id      Name  Subject
0    1    Pradhan   English
1    2      Venu      Java
2    3  Madhurima     Html
3    4  Nireekshan   Python
4    5      Shafi        C
5    6      Veeru  dot net

      Id      Name  Subject
0   11     Srinu      Java
1   12  Sumanth     Html
2   13  Neelima      Cpp
3   14    Daniel   Python
4   15    Arjun        C
5   16      Veeru  dot net

  Id_x      Name  Subject  Id_y
0    6    Veeru  dot net     16
```

18. Pandas – DataFrame – Concat

Contents

1. Introduction	2
2. concat(p)	2

18. Pandas – DataFrame – Concat

1. Introduction

- ✓ Based on requirement we can add(concatenate) one DataFrame with another DataFrame.

2. concat(p)

- ✓ concat(p) is predefined function in pandas package.
- ✓ We should access this function by using pandas library name
- ✓ This function concatenate one DataFrame to another DataFrame

Program Name Creating DataFrames
demo1.py

```
import pandas as pd

d1 = [[11, 22], [33, 44]]
d2 = [[55, 66], [77, 88]]

c1 = ["A", "B"]

df1 = pd.DataFrame(d1, columns = c1)
df2 = pd.DataFrame(d2, columns = c1)

print(df1)
print()
print(df2)
```

Output

	A	B
0	11	22
1	33	44
	A	B
0	55	66
1	77	88

Program Name Concatenating one DataFrame to another DataFrame
demo2.py

```
import pandas as pd

d1 = [[11, 22], [33, 44]]
d2 = [[55, 66], [77, 88]]

c1 = ["A", "B"]

df1 = pd.DataFrame(d1, columns = c1)
df2 = pd.DataFrame(d2, columns = c1)

result = [df1, df2]

df3 = pd.concat(result)

print(df1)
print()
print(df2)
print()
print(df3)
```

Output

```
      A    B
0   11   22
1   33   44

      A    B
0   55   66
1   77   88

      A    B
0   11   22
1   33   44
0   55   66
1   77   88
```

Program Name Concatenating one DataFrame to another DataFrame
demo3.py

```
import pandas as pd

d1 = [[11, 22], [33, 44]]
d2 = [[55, 66], [77, 88]]

c1 = ["A", "B"]

df1 = pd.DataFrame(d1, columns = c1)
df2 = pd.DataFrame(d2, columns = c1)

result = [df1, df2]

df3 = pd.concat(result, ignore_index = True)

print(df1)
print()
print(df2)
print()
print(df3)
```

Output

```
      A    B
0   11   22
1   33   44

      A    B
0   55   66
1   77   88

      A    B
0   11   22
1   33   44
2   55   66
3   77   88
```

Program Name Concatenating one DataFrame to another DataFrame
demo4.py

```
import pandas as pd

d1 = [[11, 22], [33, 44]]
d2 = [[55, 66], [77, 88], [67, 99]]

c1 = ["A", "B"]

df1 = pd.DataFrame(d1, columns = c1)
df2 = pd.DataFrame(d2, columns = c1)

result = [df1, df2]

df3 = pd.concat(result)

print(df1)
print()
print(df2)
print()
print(df3)
```

Output

```
      A    B
0   11   22
1   33   44

      A    B
0   55   66
1   77   88
2   67   99

      A    B
0   11   22
1   33   44
0   55   66
1   77   88
2   67   99
```

Program Name Concatenating one DataFrame to another DataFrame
demo5.py

```
import pandas as pd

d1 = [[11, 22], [33, 44]]
d2 = [[55, 66], [77, 88], [67, 99]]

c1 = ["A", "B"]

df1 = pd.DataFrame(d1, columns = c1)
df2 = pd.DataFrame(d2, columns = c1)

result = [df1, df2]

df3 = pd.concat(result, ignore_index = True)

print(df1)
print()
print(df2)
print()
print(df3)
```

Output

```
      A    B
0   11   22
1   33   44

      A    B
0   55   66
1   77   88
2   67   99

      A    B
0   11   22
1   33   44
2   55   66
3   77   88
4   67   99
```

Program Name Concatenating one DataFrame to another DataFrame
demo6.py

```
import pandas as pd

d1 = [[11, 22], [33, 44]]
d2 = [[55, 66], [77, 88]]

c1 = ["A", "B"]
c2 = ["X", "Y"]

df1 = pd.DataFrame(d1, columns = c1)
df2 = pd.DataFrame(d2, columns = c2)

print(df1)
print()
print(df2)
```

Output

	A	B
0	11	22
1	33	44
	X	Y
0	55	66
1	77	88

Program Name Concatenating one DataFrame to another DataFrame
demo7.py

```
import pandas as pd

d1 = [[11, 22], [33, 44]]
d2 = [[55, 66], [77, 88]]

c1 = ["A", "B"]
c2 = ["X", "Y"]

df1 = pd.DataFrame(d1, columns = c1)
df2 = pd.DataFrame(d2, columns = c2)

result = [df1, df2]

df3 = pd.concat(result, axis = 1)

print(df1)
print()
print(df2)
print()
print(df3)
```

Output

```
A    B
0  11  22
1  33  44

      X    Y
0  55  66
1  77  88

      A    B    X    Y
0  11  22  55  66
1  33  44  77  88
```

Program Name Concatenating one DataFrame to another DataFrame
demo8.py

```
import pandas as pd

d1 = [[11, 22], [33, 44]]
d2 = [[55, 66], [77, 88], [65, 99]]

c1 = ["A", "B"]
c2 = ["X", "Y"]

df1 = pd.DataFrame(d1, columns = c1)
df2 = pd.DataFrame(d2, columns = c2)

print(df1)
print()
print(df2)
```

Output

	A	B
0	11	22
1	33	44

	X	Y
0	55	66
1	77	88
2	65	99

Program Name Concatenating one DataFrame to another DataFrame
demo9.py

```
import pandas as pd

d1 = [[11, 22], [33, 44]]
d2 = [[55, 66], [77, 88], [65, 99]]

c1 = ["A", "B"]
c2 = ["X", "Y"]

df1 = pd.DataFrame(d1, columns = c1)
df2 = pd.DataFrame(d2, columns = c2)

result = [df1, df2]

df3 = pd.concat(result, axis = 1)

print(df1)
print()
print(df2)
print()
print(df3)
```

Output

```
      A    B
0   11   22
1   33   44

      X    Y
0   55   66
1   77   88
2   65   99

      A    B    X    Y
0   11.0  22.0  55   66
1   33.0  44.0  77   88
2     NaN   NaN   65   99
```

19. Pandas – DataFrame - Adding, dropping columns & rows

Contents

1. Adding column to DataFrame	2
2. Dropping columns from DataFrame	6
3. Dropping rows from DataFrame	8

19. Pandas – DataFrame - Adding, dropping columns & rows

1. Adding column to DataFrame

- ✓ Based on requirement we can add column to existing DataFrame

Program Creating DataFrame
Name demo1.py
Input file sales8.csv

```
import pandas as pd

df = pd.read_csv("sales8.csv")

print(df.head(5))
```

Output

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity
0	192837	Veeru	3	LG Mobile	65999	1
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1
3	192840	Shahid	20	iPhone 11	60000	2
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3

Data Science – Pandas DataFrame – Adding, dropping cols/rows

Program Name Adding Status column to DataFrame
Name demo6.py
Input file sales8.csv

```
import pandas as pd

df = pd.read_csv("sales8.csv")

df['Status'] = "Delivered"
print(df.head())
```

Output

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity	Status
0	192837	Veeru	3	LG Mobile	65999	1	Delivered
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2	Delivered
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1	Delivered
3	192840	Shahid	20	iPhone 11	60000	2	Delivered
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3	Delivered

Program Name Adding Total cost column to DataFrame
Name demo2.py
Input file sales8.csv

```
import pandas as pd

df = pd.read_csv("sales8.csv")

df["Total Cost"] = df['Product cost']*df['Quantity']

print(df.head(5))
```

Output

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity	Total Cost
0	192837	Veeru	3	LG Mobile	65999	1	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2	126000
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1	75999
3	192840	Shahid	20	iPhone 11	60000	2	120000
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3	209997

Program Name Adding Total cost column to DataFrame by using apply method
Input file demo3.py
sales8.csv

```
import pandas as pd

df = pd.read_csv('sales8.csv')

def total(df):
    t = df['Product cost'] * df['Quantity']
    return t

df['Total cost'] = df.apply(total, axis = 1)

print(df.head())
```

Output

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity	Total cost
0	192837	Veeru	3	LG Mobile	65999	1	65999
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2	126000
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1	75999
3	192840	Shahid	20	iPhone 11	60000	2	120000
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3	209997

Data Science – Pandas DataFrame – Adding, dropping cols/rows

Program Name demo5.py
Input file sales8.csv

```
import pandas as pd

df = pd.read_csv("sales8.csv")

print(df.head())

new = df['Product cost'] * df['Quantity']
df.insert(5,"Total Cost", new)

print()
print(df.head(5))
```

Output

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity
0	192837	Veeru	3	LG Mobile	65999	1
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1
3	192840	Shahid	20	iPhone 11	60000	2
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3

	Order Id	Customer name	Customer Id	Product name	Product cost	Total Cost	Quantity
0	192837	Veeru	3	LG Mobile	65999	65999	1
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	126000	2
2	192839	Balaji	12	34in Ultrawide Monitor	75999	75999	1
3	192840	Shahid	20	iPhone 11	60000	120000	2
4	192841	Vinay	10	Bose SoundSport Headphones	69999	209997	3

2. Dropping columns from DataFrame

- ✓ Based on requirement we can drop column from existing DataFrame

Program Dropping single columns
Name demo7.py
Input file sales8.csv

```
import pandas as pd

df1 = pd.read_csv("sales8.csv")

df2 = df1.drop(columns = 'Customer name')

print(df1.head())
print()
print(df2.head())
```

Output

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity
0	192837	Veeru	3	LG Mobile	65999	1
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1
3	192840	Shahid	20	iPhone 11	60000	2
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3

	Order Id	Customer Id	Product name	Product cost	Quantity
0	192837	3	LG Mobile	65999	1
1	192838	19	Apple iPad 10.2-inch	63000	2
2	192839	12	34in Ultrawide Monitor	75999	1
3	192840	20	iPhone 11	60000	2
4	192841	10	Bose SoundSport Headphones	69999	3

Data Science – Pandas DataFrame – Adding, dropping cols/rows

Program Dropping multiple columns
Name demo8.py
Input file sales8.csv

```
import pandas as pd

df1 = pd.read_csv("sales8.csv")

df2 = df1.drop(['Customer name', 'Product name'], axis = 1)

print(df1.head())
print()
print(df2.head())
```

Output

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity
0	192837	Veeru	3	LG Mobile	65999	1
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1
3	192840	Shahid	20	iPhone 11	60000	2
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3

	Order Id	Customer Id	Product cost	Quantity
0	192837	3	65999	1
1	192838	19	63000	2
2	192839	12	75999	1
3	192840	20	60000	2
4	192841	10	69999	3

3. Dropping rows from DataFrame

- ✓ We can drop the rows from DataFrame by using drop method

Program Dropping single row
Name demo9.py
Input file sales8.csv

```
import pandas as pd

df1 = pd.read_csv("sales8.csv")

df2 = df1.drop(3, axis = 0)

print(df1.head())
print()
print(df2.head())
```

Output

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity
0	192837	Veeru	3	LG Mobile	65999	1
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1
3	192840	Shahid	20	iPhone 11	60000	2
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity
0	192837	Veeru	3	LG Mobile	65999	1
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3
5	192842	Vinay	10	20in Monitor	65999	2

Data Science – Pandas DataFrame – Adding, dropping cols/rows

Program Name Dropping multiple rows
Input file demo10.py
sales8.csv

```
import pandas as pd

df1 = pd.read_csv("sales8.csv")
df2 = df1.drop([1, 2], axis = 0)

print(df1.head())
print()
print(df2.head())
```

Output

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity
0	192837	Veeru	3	LG Mobile	65999	1
1	192838	Neelima	19	Apple iPad 10.2-inch	63000	2
2	192839	Balaji	12	34in Ultrawide Monitor	75999	1
3	192840	Shahid	20	iPhone 11	60000	2
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3

	Order Id	Customer name	Customer Id	Product name	Product cost	Quantity
0	192837	Veeru	3	LG Mobile	65999	1
3	192840	Shahid	20	iPhone 11	60000	2
4	192841	Vinay	10	Bose SoundSport Headphones	69999	3
5	192842	Vinay	10	20in Monitor	65999	2
6	192843	Balaji	12	iPhone 8	55999	3

20. Pandas – DataFrame - Date and time

Contents

1. Date data type	2
2. Converting object data type into date data type.....	4
2.1. <code>to_datetime(p)</code>	4
2.2. <code>astype (p)</code>	5
3. Format parameter	6
4. NaT values	14
5. Selecting from start to end date values	18
6. Access specific dates like last 20 days or 2 months or 2 years records	19
7. Extract year, month, day from Date column.....	24
8. Encoding Days of the Week	26

20. Pandas – DataFrame - Date and time

1. Date data type

- ✓ Whenever we load csv file, if that file contains any column having date values then by default pandas will consider that column as object
- ✓ We can provide parse_dates = ['name of the column1', 'name of the column2'] then pandas considered those columns as datetime data type

Program Creating DataFrames

Name demo1.py

File name sales7_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv')

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

Output

```
   Order_Id Customer_Name Customer_Id          Product_Name Product_Cost Pur_Date
0      1023        Venki           15       27in FHD Monitor     59000  1/1/2019 0:00
1      1024    Chaithanya          14            iPhone 11     69000  1/1/2019 1:00
2      1025       Shahid           20  Bose SoundSport Headphones     65999  1/1/2019 2:00
3      1026       Veeru            3      Apple iPad 10.2-inch     63999  1/1/2019 3:00
4      1027        Venu           23            Google Phone     63999  1/1/2019 4:00

Order_Id      int64
Customer_Name   object
Customer_Id      int64
Product_Name     object
Product_Cost      int64
Pur_Date         object
dtype: object
```

Program Name Loading csv file by using parse_dates parameter

File name demo2.py

sales7_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv', parse_dates = ['Pur_Date'])

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

Output

```
   Order_Id Customer_Name Customer_Id      Product_Name  Product_Cost      Pur_Date
0      1023        Venki          15    27in FHD Monitor     59000 2019-01-01 00:00:00
1      1024    Chaithanya          14        iPhone 11     69000 2019-01-01 01:00:00
2      1025       Shahid          20  Bose SoundSport Headphones     65999 2019-01-01 02:00:00
3      1026       Veeru           3      Apple iPad 10.2-inch     63999 2019-01-01 03:00:00
4      1027        Venu          23        Google Phone     63999 2019-01-01 04:00:00

Order_Id          int64
Customer_Name    object
Customer_Id       int64
Product_Name     object
Product_Cost      int64
Pur_Date         datetime64[ns]
dtype: object
```

2. Converting object data type into date data type

- ✓ We can convert from object data type into datetime data type explicitly.
- ✓ By using,
 - `to_datetime(p)` function
 - `astype(p)` method in Series

2.1. `to_datetime(p)`

- ✓ `to_datetime(p)` is predefined function in pandas
- ✓ This function we should access by using pandas library.
- ✓ This function convert from object data type into date data type.

Program Name Loading csv file and converting Pur_Date column into Date format
File name demo3.py
sales7_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv')

df['Pur_Date'] = pd.to_datetime(df['Pur_Date'])

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

Output

```
   Order_Id Customer_Name Customer_Id      Product_Name  Product_Cost      Pur_Date
0      1023       Venki          15    27in FHD Monitor     59000 2019-01-01 00:00:00
1      1024  Chaithanya          14           iPhone 11     69000 2019-01-01 01:00:00
2      1025       Shahid          20  Bose SoundSport Headphones     65999 2019-01-01 02:00:00
3      1026       Veeru           3    Apple iPad 10.2-inch     63999 2019-01-01 03:00:00
4      1027        Venu          23      Google Phone     63999 2019-01-01 04:00:00

Order_Id          int64
Customer_Name    object
Customer_Id       int64
Product_Name     object
Product_Cost      int64
Pur_Date   datetime64[ns]
dtype: object
```

2.2. astype (p)

- ✓ astype(p) is predefined method Series class
- ✓ This method we should access by using Series object only
- ✓ This method convert from object data type into datetime data type.

Program Name Loading csv file and converting Pur_Date column into Date format
File name demo4.py
sales7_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv')

df['Pur_Date'] = df['Pur_Date'].astype('datetime64[ns]')

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

Output

```
   Order_Id Customer_Name Customer_Id      Product_Name Product_Cost      Pur_Date
0     1023        Venki          15    27in FHD Monitor     59000 2019-01-01 00:00:00
1     1024  Chaithanya          14           iPhone 11     69000 2019-01-01 01:00:00
2     1025       Shahid          20  Bose SoundSport Headphones     65999 2019-01-01 02:00:00
3     1026       Veeru           3      Apple iPad 10.2-inch     63999 2019-01-01 03:00:00
4     1027        Venu          23            Google Phone     63999 2019-01-01 04:00:00

Order_Id          int64
Customer_Name    object
Customer_Id       int64
Product_Name     object
Product_Cost      int64
Pur_Date    datetime64[ns]
dtype: object
```

3. Format parameter

- ✓ We can represent the date formats in different ways,
 - March 23rd, 2015 as "03-23-15" or "3|23|2015" and etc
- ✓ So, we can use the format parameter to specify the exact format of the string.

Code	Description	Example
%Y	Full year	2001
%m	Month w/ zero padding	04
%d	Day of the month w/ zero padding	09
%I	Hour (12hr clock) w/ zero padding	02
%p	AM or PM	AM
%M	Minute w/ zero padding	05
%S	Second w/ zero padding	09

Program Name Creating a DataFrame
demo5.py

```
import pandas as pd

data = {
    'Product': ['Samsung', 'iPhone', 'Motorola'],
    'Status': ['Success', 'Success', 'Failed'],
    'Cost': [10000, 50000, 15000],
    'PurDate': ['20190902','20190913','20190921'],
}

df = pd.DataFrame(data)

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

Output

```
      Product   Status   Cost   PurDate
0   Samsung  Success  10000  20190902
1     iPhone  Success  50000  20190913
2  Motorola    Failed  15000  20190921

Product      object
Status       object
Cost        int64
PurDate     object
dtype: object
```

Program Name Converting Date with specific format
demo6.py

```
import pandas as pd

data = {
    'Product': ['Samsung', 'iPhone', 'Motorola'],
    'Status': ['Success', 'Success', 'Failed'],
    'Cost': [10000, 50000, 15000],
    'PurDate': ['20190902','20190913','20190921'],
}

df = pd.DataFrame(data)
df['PurDate'] = pd.to_datetime(df['PurDate'])

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

Output

```
      Product    Status    Cost    PurDate
0    Samsung  Success  10000  2019-09-02
1      iPhone  Success  50000  2019-09-13
2  Motorola    Failed  15000  2019-09-21

Product          object
Status           object
Cost            int64
PurDate    datetime64[ns]
dtype: object
```

Program Name Converting Date with specific format
demo7.py

```
import pandas as pd

data = {
    "Product": ["Samsung", "iPhone", "Motorola"],
    "Status": ["Success", "Success", "Failed"],
    "Cost": [10000, 50000, 15000],
    "PurDate": ['02092019','13092019','21092019'],
}

df = pd.DataFrame(data)

df['PurDate'] = pd.to_datetime(df['PurDate'], format = '%d%m%Y')

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

Output

```
   Product    Status    Cost    PurDate
0  Samsung  Success  10000  2019-09-02
1    iPhone  Success  50000  2019-09-13
2  Motorola   Failed  15000  2019-09-21

Product          object
Status           object
Cost            int64
PurDate    datetime64[ns]
dtype: object
```

Program Name Converting Date with specific format
demo8.py

```
import pandas as pd

data = {
    "Product": ["Samsung", "iPhone", "Motorola"],
    "Status": ["Success", "Success", "Failed"],
    "Cost": [10000, 50000, 15000],
    "PurDate": ['02Sep2019','13Sep2019','21Sep2019'],
}

df = pd.DataFrame(data)

df['PurDate'] = pd.to_datetime(df['PurDate'])

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

Output

```
   Product    Status    Cost      PurDate
0  Samsung  Success  10000  2019-09-02
1    iPhone  Success  50000  2019-09-13
2  Motorola    Failed  15000  2019-09-21

Product          object
Status           object
Cost            int64
PurDate    datetime64[ns]
dtype: object
```

Program Name Converting Date with specific format
demo9.py

```
import pandas as pd

data = {
    "Product": ["Samsung", "iPhone", "Motorola"],
    "Status": ["Success", "Success", "Failed"],
    "Cost": [10000, 50000, 15000],
    "PurDate": ['02Sep2019','13Sep2019','21Sep2019'],
}

df = pd.DataFrame(data)

df['PurDate'] = pd.to_datetime(df['PurDate'], format = '%d%b%Y')

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

Output

```
      Product    Status   Cost     PurDate
0    Samsung  Success  10000  2019-09-02
1      iPhone  Success  50000  2019-09-13
2  Motorola    Failed  15000  2019-09-21

Product              object
Status               object
Cost                int64
PurDate        datetime64[ns]
dtype: object
```

Program Name Converting Date with specific format
demo10.py

```
import pandas as pd

data = {
    "Product": ["Samsung", "iPhone", "Motorola"],
    "Status": ["Success", "Success", "Failed"],
    "Cost": [10000, 50000, 15000],
    "PurDate": ['02-Sep-2019','13-Sep-2019','21-Sep-2019'],
}

df = pd.DataFrame(data)

df['PurDate'] = pd.to_datetime(df['PurDate'])

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

Output

```
      Product    Status    Cost    PurDate
0    Samsung  Success  10000  2019-09-02
1     iPhone  Success  50000  2019-09-13
2  Motorola    Failed  15000  2019-09-21

Product           object
Status            object
Cost             int64
PurDate   datetime64[ns]
dtype: object
```

Program Name Converting Date with specific format
demo11.py

```
import pandas as pd

data = {
    "Product": ["Samsung", "iPhone", "Motorola"],
    "Status": ["Success", "Success", "Failed"],
    "Cost": [10000, 50000, 15000],
    "PurDate":
    ['20190902093000','20190913093000','20190921200000'],
}

df = pd.DataFrame(data)

df['PurDate'] = pd.to_datetime(df['PurDate'])

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

Output

```
      Product    Status    Cost        PurDate
0   Samsung  Success  10000  2019-09-02 09:30:00
1     iPhone  Success  50000  2019-09-13 09:30:00
2  Motorola   Failed  15000  2019-09-21 20:00:00

Product          object
Status           object
Cost            int64
PurDate    datetime64[ns]
dtype: object
```

4. NaT values

- ✓ Date column may contains missing values in pandas DataFrame.
- ✓ If Date column contains missing values then while converting into Date data type then we will get an error.
- ✓ By using errors = "coerce" keyword argument we can solve this problem.
- ✓ This argument converts Date column missing values into NaT (Not a Time) values.
 - Coerce errors i.e. convert un parse able date into **NaT** (Not a Time)

Program Name	Creating DataFrame demo12.py
---------------------	---------------------------------

```
import pandas as pd

data = {
    "Product": ["Samsung", "iPhone", "Motorola"],
    "Status": ["Success", "Success", "Failed"],
    "Cost": [10000, 50000, 15000],
    "PurDate": ['02-Sep-2019','Here date is missing','21-Sep-2019']
}

df = pd.DataFrame(data)

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

Output

```
      Product    Status    Cost          PurDate
0   Samsung  Success  10000  02-Sep-2019
1     iPhone  Success  50000  Here date is missing
2  Motorola   Failed  15000  21-Sep-2019

Product      object
Status       object
Cost        int64
PurDate     object
dtype: object
```

Program Name Converting Date with specific format: **Error**
demo13.py

```
import pandas as pd

data = {
    "Product": ["Samsung", "iPhone", "Motorola"],
    "Status": ["Success", "Success", "Failed"],
    "Cost": [10000, 50000, 15000],
    "PurDate": ['02-Sep-2019','Here date is missing','21-Sep-2019']
}

df = pd.DataFrame(data)

df['PurDate'] = pd.to_datetime(df['PurDate'])

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

Output

TypeError: invalid string coercion to datetime for "Here date is missing" at position 1

Program Name

to_datetime(p) function

demo14.py

```
import pandas as pd

data = {
    "Product": ["Samsung", "iPhone", "Motorola"],
    "Status": ["Success", "Success", "Failed"],
    "Cost": [10000, 50000, 15000],
    "PurDate": ['02-Sep-2019','Here date is missing','21-Sep-2019']
}

df = pd.DataFrame(data)

df['PurDate'] = pd.to_datetime(df['PurDate'], errors="coerce")

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

Output

```
   Product    Status    Cost      PurDate
0  Samsung  Success  10000  2019-09-02
1    iPhone  Success  50000        NaT
2  Motorola    Failed  15000  2019-09-21

Product          object
Status           object
Cost            int64
PurDate    datetime64[ns]
dtype: object
```

5. Selecting from start to end date values

- ✓ Based on requirement we can select specific dates, like
 - Start date to end date

Program Selecting Dataframe in between the dates

Name demo15.py

File Name sales7_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv', parse_dates=['Pur_Date'])

start = df['Pur_Date'] > '2019-1-1 01:00:00'
end = df['Pur_Date'] < '2019-1-1 05:00:00'

result = df[start & end]

print(result)
```

Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost	Pur_Date
2	1025	Shahid	20	Bose SoundSport Headphones	65999	2019-01-01 02:00:00
3	1026	Veeru	3	Apple iPad 10.2-inch	63999	2019-01-01 03:00:00
4	1027	Venu	23	Google Phone	63999	2019-01-01 04:00:00

6. Access specific dates like last 20 days or 2 months or 2 years records

- ✓ Based on requirement we can get last 10 days, 20 days, 40 days, 1 month, 2 months, 3 months, 1 year and 2 years date data as well.
- ✓ We can also sort the dataframe by using `sort_values()`

Program Name Creating DataFrame
File name demo16.py
sales7_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv', parse_dates = ['Pur_Date'])

print(df.head())
```

Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost	Pur_Date
0	1023	Venki	15	27in FHD Monitor	59000	2019-01-01 00:00:00
1	1024	Chaitanya	14	iPhone 11	69000	2019-01-01 01:00:00
2	1025	Shahid	20	Bose SoundSport Headphones	65999	2019-01-01 02:00:00
3	1026	Veeru	3	Apple iPad 10.2-inch	63999	2019-01-01 03:00:00
4	1027	Venu	23	Google Phone	63999	2019-01-01 04:00:00

Program Name Accessing last 10 days records
File name demo17.py
sales7_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv', parse_dates = ['Pur_Date'])

new_df = df.set_index("Pur_Date")
days_10 = new_df.last("10D")

print(days_10)
```

Output

Pur_Date	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
2020-02-11 15:00:00	10782	Harsha	5	Google Phone	75000
2020-02-11 16:00:00	10783	Veeru	3	Google Phone	61000
2020-02-11 17:00:00	10784	Vinay	10	34in Ultrawide Monitor	69999
2020-02-11 18:00:00	10785	Jaya Chandra	21	ThinkPad Laptop	50000
2020-02-11 19:00:00	10786	Shahid	20	iPhone 11	51999
...
2020-02-21 10:00:00	11017	Karteek	4	20in Monitor	51999
2020-02-21 11:00:00	11018	Vinay	10	Apple Airpods Headphones	75999
2020-02-21 12:00:00	11019	Neelima	19	LG ThinQ Refrigerator	75999
2020-02-21 13:00:00	11020	Siddhu	18	iPhone 7s	65999
2020-02-21 14:00:00	11021	Tarun	11	34in Ultrawide Monitor	55000

[240 rows x 5 columns]

Program Name Accessing last 40 days records
File name demo18.py
sales7_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv', parse_dates = ['Pur_Date'])

new_df = df.set_index("Pur_Date")
days_40 = new_df.last("40D")

print(days_40)
```

Output

Pur_Date	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
2020-01-12 15:00:00	10062	Madhurima	7	Google Phone	51999
2020-01-12 16:00:00	10063	Pradhan	17	ThinkPad Laptop	63000
2020-01-12 17:00:00	10064	Venu	23	LG ThinQ Refrigerator	69000
2020-01-12 18:00:00	10065	Venki	15	ThinkPad Laptop	60000
2020-01-12 19:00:00	10066	Balaji	12	Flatscreen TV	65000
...
2020-02-21 10:00:00	11017	Karteek	4	20in Monitor	51999
2020-02-21 11:00:00	11018	Vinay	10	Apple Airpods Headphones	75999
2020-02-21 12:00:00	11019	Neelima	19	LG ThinQ Refrigerator	75999
2020-02-21 13:00:00	11020	Siddhu	18	iPhone 7s	65999
2020-02-21 14:00:00	11021	Tarun	11	34in Ultrawide Monitor	55000

[960 rows x 5 columns]

Program Name Accessing last 1 month records
File name demo19.py
sales7_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv', parse_dates = ['Pur_Date'])

new_df = df.set_index("Pur_Date")
month_1 = new_df.last("1M")

print(month_1)
```

Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
Pur_Date					
2020-01-31 15:00:00	10518	Pradhan	17	LG ThinQ Refrigerator	65999
2020-01-31 16:00:00	10519	Shafi	25	Samsung Galaxy S20	63000
2020-01-31 17:00:00	10520	Veeru	3	34in Ultrawide Monitor	59000
2020-01-31 18:00:00	10521	Balaji	12	Macbook Pro Laptop	50000
2020-01-31 19:00:00	10522	Pradhan	17	iPhone 7s	51999
...
2020-02-21 10:00:00	11017	Karteek	4	20in Monitor	51999
2020-02-21 11:00:00	11018	Vinay	10	Apple Airpods Headphones	75999
2020-02-21 12:00:00	11019	Neelima	19	LG ThinQ Refrigerator	75999
2020-02-21 13:00:00	11020	Siddhu	18	iPhone 7s	65999
2020-02-21 14:00:00	11021	Tarun	11	34in Ultrawide Monitor	55000

[504 rows x 5 columns]

Program Name Accessing last 1 year records
File name demo20.py
sales7_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv', parse_dates = ['Pur_Date'])

new_df = df.set_index("Pur_Date")
year_1 = new_df.last("1Y")

print(year_1)
```

Output

Pur_Date	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost
2019-12-31 15:00:00	9774	Veeru	3	27in FHD Monitor	65999
2019-12-31 16:00:00	9775	Shafi	25	27in 4K Gaming Monitor	65999
2019-12-31 17:00:00	9776	Shahid	20	Samsung Galaxy S20	50000
2019-12-31 18:00:00	9777	Venki	15	27in 4K Gaming Monitor	50000
2019-12-31 19:00:00	9778	Sumanth	22	Google Phone	65999
...
2020-02-21 10:00:00	11017	Karteek	4	20in Monitor	51999
2020-02-21 11:00:00	11018	Vinay	10	Apple Airpods Headphones	75999
2020-02-21 12:00:00	11019	Neelima	19	LG ThinQ Refrigerator	75999
2020-02-21 13:00:00	11020	Siddhu	18	iPhone 7s	65999
2020-02-21 14:00:00	11021	Tarun	11	34in Ultrawide Monitor	55000

[1248 rows x 5 columns]

7. Extract year, month, day from Date column

- ✓ Based on requirement we can get year, month, day, hour, minute from Date column.
- ✓ Sometimes it can be useful to break up a column of dates into components.

Program Name demo21.py
File name sales7_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv', parse_dates = ['Pur_Date'])

df['year'] = df['Pur_Date'].dt.year
df['month'] = df['Pur_Date'].dt.month
df['day'] = df['Pur_Date'].dt.day

print(df.head())
```

Output

```
   Order_Id Customer_Name Customer_Id ... year month day
0      1023         Venki        15 ... 2019    1    1
1      1024     Chaithanya        14 ... 2019    1    1
2      1025        Shahid        20 ... 2019    1    1
3      1026        Veeru         3 ... 2019    1    1
4      1027         Venu        23 ... 2019    1    1

[5 rows x 9 columns]
```

Program Breaking up the Date column value into multiple features

Name demo22.py

File name sales7_dates.csv

```
import pandas as pd

df = pd.read_csv('sales7_dates.csv', parse_dates = ['Pur_Date'])

df['year'] = df['Pur_Date'].dt.year
df['month'] = df['Pur_Date'].dt.month
df['day'] = df['Pur_Date'].dt.day
df['hour'] = df['Pur_Date'].dt.hour
df['minute'] = df['Pur_Date'].dt.minute

print(df.head())
```

Output

```
   Order_Id Customer_Name Customer_Id      Product_Name ...  month day hour minute
0     1023        Venki         15    27in FHD Monitor ...
1     1024  Chaithanya         14          iPhone 11 ...
2     1025       Shahid         20  Bose SoundSport Headphones ...
3     1026        Veeru          3    Apple iPad 10.2-inch ...
4     1027        Venu         23        Google Phone ...
[5 rows x 11 columns]
```

8. Encoding Days of the Week

- ✓ We can get the day of the week for each date by using pandas
 - Knowing the days names will helpful to understand the business flow, like we can compare total sales on specific day for the past three years.

Program Name Encoding Days of the Week
demo23.py

```
import pandas as pd

data = {
    'Product': ['Samsung', 'iPhone', 'Motorola'],
    'Status': ['Success', 'Success', 'Failed'],
    'Cost': [10000, 50000, 15000],
    'PurDate': ['2018-01-01','2018-01-02','2018-01-03'],
}

df = pd.DataFrame(data)
df['PurDate'] = pd.to_datetime(df['PurDate'])

print(df.head())
print()
print(df["PurDate"].dt.day_name())
```

Output

```
   Product  Status   Cost      PurDate
0  Samsung  Success  10000  2018-01-01
1    iPhone  Success  50000  2018-01-02
2  Motorola    Failed  15000  2018-01-03

0      Monday
1     Tuesday
2   Wednesday
Name: PurDate, dtype: object
```

Note

- ✓ The day of the week with Monday = 0, Sunday = 6

Program Name Encoding Days of the Week
demo24.py

```
import pandas as pd

data = {
    'Product': ['Samsung', 'iPhone', 'Motorola'],
    'Status': ['Success', 'Success', 'Failed'],
    'Cost': [10000, 50000, 15000],
    'PurDate': ['2018-01-01','2018-01-02','2018-01-03'],
}

df = pd.DataFrame(data)
df['PurDate'] = pd.to_datetime(df['PurDate'])

print(df.head())
print()
print(df['PurDate'].dt.weekday)
```

Output

```
      Product   Status   Cost   PurDate
0    Samsung  Success  10000  2018-01-01
1      iPhone  Success  50000  2018-01-02
2  Motorola    Failed  15000  2018-01-03

0    0
1    1
2    2
Name: PurDate, dtype: int64
```

21. Pandas – DataFrame - Concatenate Multiple csv files

Contents

1. Real time requirement	2
2. Loading csv files from all files	2
3. os module	2
4. listdir(p) function.....	3
5. filter(p1, p2) function	4
6. Concatenating all csv file.....	5

21. Pandas – DataFrame - Concatenate Multiple csv files

1. Real time requirement

- ✓ Generally total data will be stored with multiple files
 - Yearly data: Jan sales, Feb sales,, Dec sales
- ✓ So, we need to concatenate all these monthly sales to bring year sales right

2. Loading csv files from all files

- ✓ The very first step is we need to access all files from specific folder.
- ✓ From that folder we need to capture only csv files.

3. os module

- ✓ os is a predefined module in python.
- ✓ By using this module we can load all files from the folder.

4. **listdir(p)** function

- ✓ `listdir(p)` is a predefined function in `os` module
- ✓ This function we should access with `os` module name.
- ✓ By using this function we can get all file names from folder.
- ✓ This function returns all file names in list.

Program Name Accessing all files from the folder
Input file demo1.py **daniel/jan_sales.csv,....,dec_sales.csv [15 files]**

```
import os

path = "./daniel"

all_files = os.listdir(path)

print(all_files)
```

Output

```
['apr_sales.csv', 'aug_sales.csv', 'dec_sales.csv', 'feb_sales.csv',
'fun.jpg', 'jan_sales.csv', 'jul_sales.csv', 'jun_sales.csv',
'mar_sales.csv', 'may_sales.csv', 'nov_sales.csv', 'oct_sales.csv',
'progress.txt', 'sep_sales.csv', 'status.xlsx']
```

5. filter(p1, p2) function

- ✓ filter(p1, p2) is a predefined function in python
- ✓ We can access this function directly.
- ✓ By using this function we can apply Boolean logic and get results accordingly.

Program Name Accessing only csv files from folder
Input file demo2.py
daniel/jan_sales.csv,....,dec_sales.csv

```
import os

path = "./daniel"

all_files = os.listdir(path)

f = filter(lambda name: name.endswith('.csv'), all_files)
csv_files = list(f)

print(all_files)
print()
print(csv_files)
```

Output

```
['apr_sales.csv', 'aug_sales.csv', 'dec_sales.csv', 'feb_sales.csv',
'fun.jpg', 'jan_sales.csv', 'jul_sales.csv', 'jun_sales.csv',
'mar_sales.csv', 'may_sales.csv', 'nov_sales.csv', 'oct_sales.csv',
'progress.txt', 'sep_sales.csv', 'status.xlsx']
```

```
['apr_sales.csv', 'aug_sales.csv', 'dec_sales.csv', 'feb_sales.csv',
'jan_sales.csv', 'jul_sales.csv', 'jun_sales.csv', 'mar_sales.csv',
'may_sales.csv', 'nov_sales.csv', 'oct_sales.csv', 'sep_sales.csv']
```

6. Concatenating all csv file

- ✓ Once we loaded all csv file then we can concatenate all csv file.
- ✓ Based on requirement by using pandas we can concatenate all csv files into one csv file

Program Name Concatenating all csv files
Input file demo3.py
demo3.py daniel/jan_sales.csv,....,dec_sales.csv [12 files]

```
import os
import glob
import pandas as pd

p = '.\daniel'

files = os.path.join(p, "*.csv")
csv_files = glob.glob(files)

result = (pd.read_csv(every) for every in csv_files)

df = pd.concat(result, ignore_index = True)

print(df)
df.to_csv("year.csv", index = False)
```

Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost	Date
0	1320	Venu	23	LG ThinQ Refrigerator	55000	4/11/2019 9:00
1	1321	Jaya Chandra	21	Bose SoundSport Headphones	63000	4/13/2019 10:00
2	1322	Mallikarjun	13	Apple Airpods Headphones	69999	4/13/2019 11:00
3	1323	Siddhu	18	Samsung Galaxy S9 Plus	55000	4/14/2019 12:00
4	1324	Daniel	6	iPhone 8	55000	4/15/2019 13:00
..
103	1819	Daniel	6	LG Washing Machine	50000	9/3/2019 4:00
104	1820	Neelima	19	20in Monitor	55000	9/3/2019 5:00
105	1821	Karteek	4	20in Monitor	50000	9/3/2019 6:00
106	1822	Jaya Chandra	21	LG ThinQ Refrigerator	75999	9/3/2019 7:00
107	1823	Chaithanya	14	27in FHD Monitor	55000	9/3/2019 8:00

[108 rows x 6 columns]

Program Name Loading year.csv files

Input file demo4.py

year.csv

```
import pandas as pd

df = pd.read_csv("year.csv", parse_dates = ["Date"])

print(df)
```

Output

	Order_Id	Customer_Name	Customer_Id	Product_Name	Product_Cost	Date
0	1320	Venu	23	LG ThinQ Refrigerator	55000	2019-04-11 09:00:00
1	1321	Jaya Chandra	21	Bose SoundSport Headphones	63000	2019-04-13 10:00:00
2	1322	Mallikarjun	13	Apple Airpods Headphones	69999	2019-04-13 11:00:00
3	1323	Siddhu	18	Samsung Galaxy S9 Plus	55000	2019-04-14 12:00:00
4	1324	Daniel	6	iPhone 8	55000	2019-04-15 13:00:00

DATA ANALYSIS - EDA – PROJECT

Contents

1. EDA.....	2
2. Real time data	2
3. Results from EDA.....	3

DATA ANALYSIS - EDA – PROJECT

1. EDA

- ✓ The full form of EDA means Explanatory Data Analysis.
- ✓ It is a process of performing initial investigations on data.
- ✓ By doing this kind of analysis, it helps us to discover patterns, standards.
- ✓ It's clearly explains the summary of statistics and graphical representations

2. Real time data

- ✓ e-Commerce domain
 - The dataset consists of transactional data with customers in different countries who make purchases from an online retail company based in the United Kingdom (UK).
 - Company: UK-based and online retail
 - Products for selling: Mainly all-occasion gifts
 - Customers: Most are wholesalers (local or international)
 - Transactions Period Data : From 1st Dec 2010 to 9th Dec 2011 (One year)



3. Results from EDA

- ✓ **Highest** number of **orders** from specific country
- ✓ **Highest money** spent on purchase specific country
- ✓ **Top 5** countries place the **highest number** of orders
- ✓ **Top 5** countries spent the **money on purchase** products
- ✓ **Highest sales** on specific Month
- ✓ There are **no transactions** on between dates.
- ✓ **Sales increase** days
- ✓ **Sales decrease** days
- ✓ **Highest** number of **orders hours** and etc

Dataset explanation

- ✓ InvoiceNo (invoice_num) : A number assigned to each transaction
- ✓ StockCode (stock_code) : Product code
- ✓ Description (description) : Product name
- ✓ Quantity (quantity) : Number of products purchased for each transaction
- ✓ InvoiceDate (invoice_date) : Timestamp for each transaction
- ✓ UnitPrice (unit_price) : Product price per unit
- ✓ CustomerID (cust_id) : Unique identifier each customer
- ✓ Country (country) : Country name