**CHAPTER-1** Data Handling using Pandas –I

#### Pandas:

* It is a package useful for data analysis and manipulation.
* Pandas provide an easy way to create, manipulate and wrangle the data.
* Pandas provide powerful and easy-to-use data structures, as well as the means to quickly perform operations on these structures.

Data scientists use Pandas for its following advantages:

* Easily handles missing data.
* It uses Series for one-dimensional data structure and DataFrame for multi-dimensional data structure.
* It provides an efficient way to slice the data.
* It provides a flexible way to merge, concatenate or reshape the data.

#### DATA STRUCTURE IN PANDAS

A data structure is a way to arrange the data in such a way that so it can be accessed quickly and we can perform various operation on this data like- retrieval, deletion, modification etc.

Pandas deals with 3 data structure-

1. Series
2. Data Frame
3. Panel

We are having only series and data frame in our syllabus.

**Series-**Series is a one-dimensional array like structure with homogeneous data, which can be used to handle and manipulate data. What makes it special is its index attribute, which has incredible functionality and is heavily mutable.

**Series**

**It has two parts-**

1. **Data part (An array of actual data)**
2. **Associated index with data (associated array of indexes or data labels)**

**e.g.-**

|  |  |
| --- | --- |
| Index | Data |
| 0 | 10 |
| 1 | 15 |
| 2 | 18 |
| 3 | 22 |

* We can say that **Series** is a labeled *one-dimensional array which can hold any type of data.*
* Data of **Series** is *always mutable*, means it can be changed.
* But the size of Data of **Series** is *always immutable*, means it cannot be changed.
* **Series** may be considered as a **Data Structure with two arrays** out which **one array** works as *Index (Labels)* and the **second array** works as *original Data.*
* ***Row Labels*** in Series are called *Index*.

**Syntax to create a Series:**

**<Series Object>=pandas.Series (data, index=idx *(optional))***

* **Where data may be *python sequence (Lists)*, ndarray, scalar value or a python dictionary.**

**How to create Series with nd array**

**Program-**

Output-

1 15

2 18

**print(s)**

3 22

**Data**

**arr=np.array([10,15,18,22])**

**s = pd.Series(arr)**

**Default Index**

**import numpy as np**

0 10

Here we create an array of 4 values.

**import pandas as pd**

**How to create Series with Mutable index**

**Program-**

**import pandas as pd**

**print(s)**

**import numpy as np arr=np.array(['a','b','c','d']) s=pd.Series(arr,**

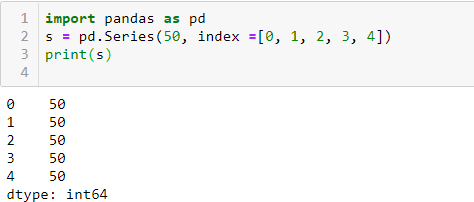
**index=['first','second','third','fourth'])**

**Output-**

first a second b third c fourth d

## Creating a series from Scalar value

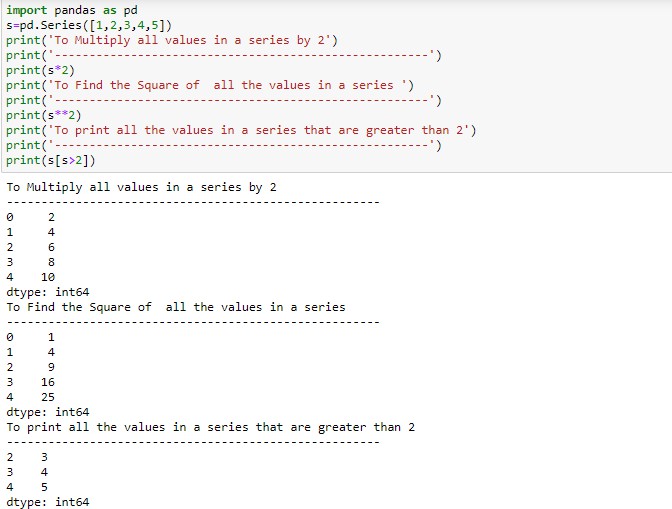
To create a series from scalar value, an index must be provided. The scalar value will be repeated as per the length of index.



**Creating a series from a Dictionary**

****

**Mathematical Operations in Series**



**Print all the values of the Series by multiplying them by 2.**

**Print Square of all the values of the series.**

**Print all the values of the Series that are greater than 2.**

**Example-2**



**While adding two series, if Non-Matching Index is found in either of the**

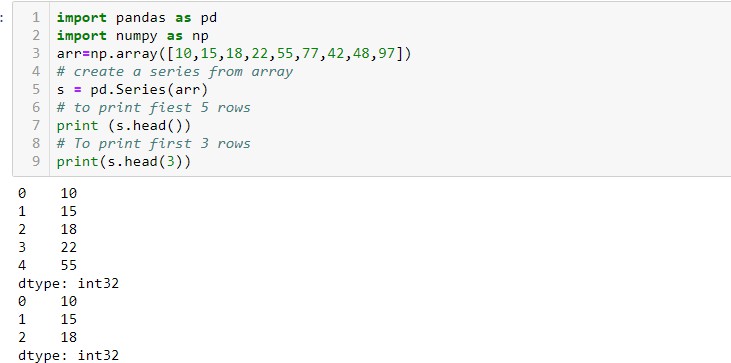
**Series, Then NaN will be printed corresponds to Non-Matching Index.**

**If Non-Matching Index is found in either of the series, then this Non-**

**Matching Index corresponding value of that series will be filled as 0.**

**Head and Tail Functions in Series**

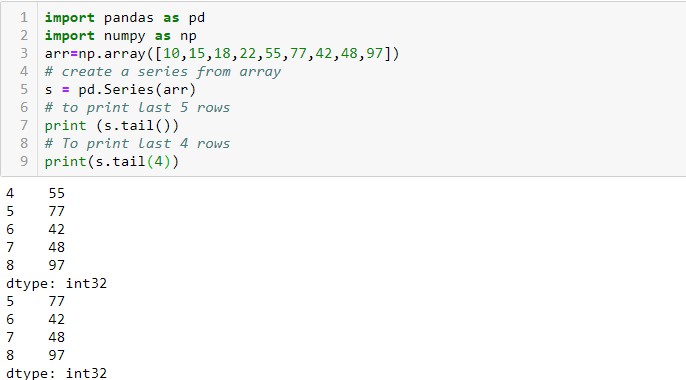
head (): It is used to access the first 5 rows of a series. Note :To access first 3 rows we can call series\_name.head(3)



Result of s.head(3)

Result of s.head()

tail(): It is used to access the last 5 rows of a series. Note :To access last 4 rows we can call series\_name.tail (4)

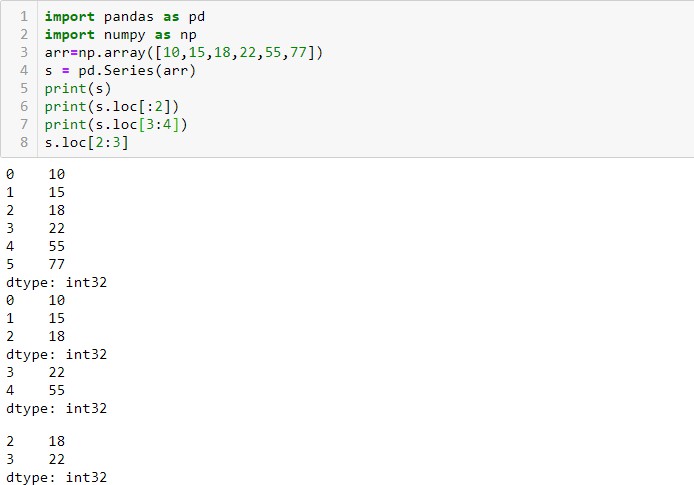


## Selection in Series

Series provides index label loc and ilocand [] to access rows and columns.

1. loc index label :-

Syntax:-series\_name.loc[StartRange: StopRange] Example-

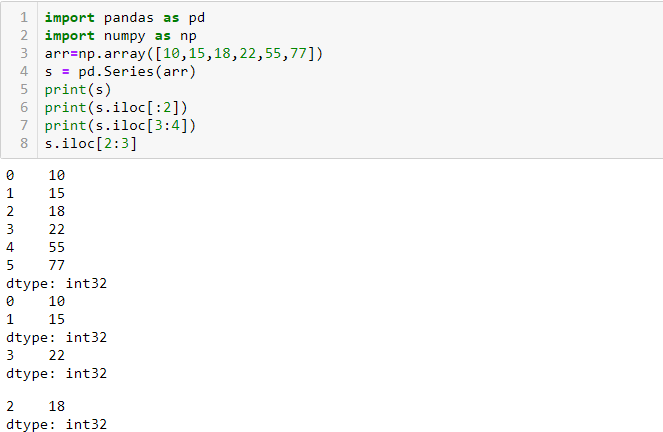


To Print Values from Index 3 to 4

To Print Values from Index 0 to 2

1. Selection Using iloc index label :-

Syntax:-series\_name.iloc[StartRange : StopRange] Example-

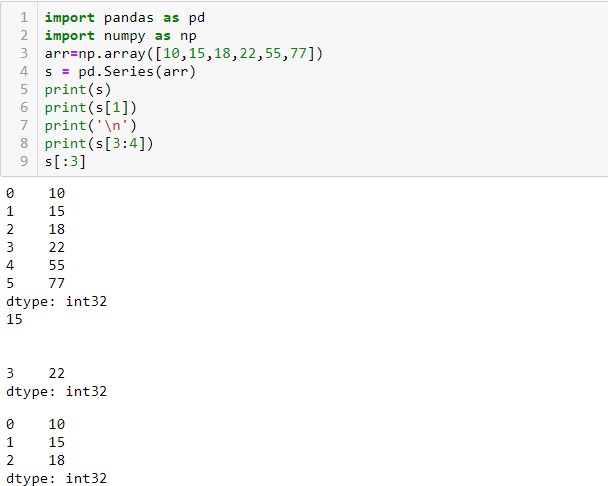


To Print Values from Index 0 to 1.

1. Selection Using [] :

Syntax:-series\_name[StartRange> : StopRange] or series\_name[ index]

Example-

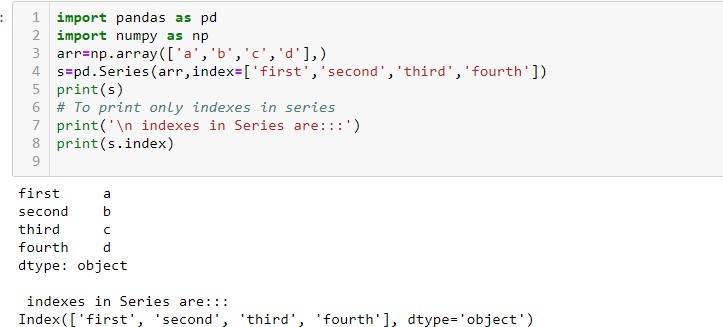


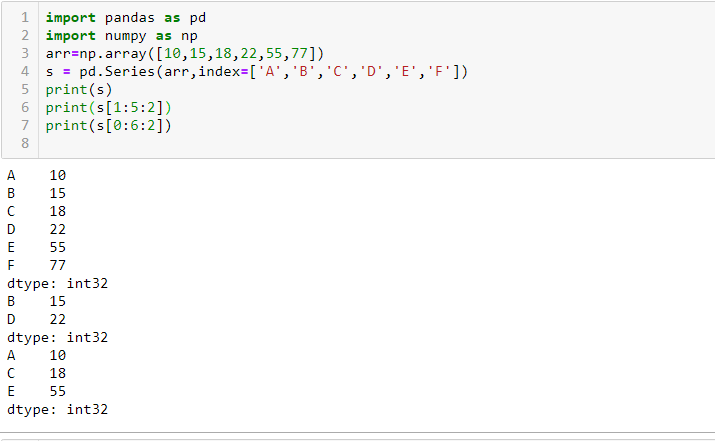
To Print Values at Index 3.

## Indexing in Series

Pandas provide index attribute to get or set the index of entries or values in series.

Example-





## Slicing in Series

Slicing is a way to retrieve subsets of data from a pandas object. A slice object syntax is –

SERIES\_NAME [start:end: step]

The segments start representing the first item, end representing the last item, and step representing the increment between each item that you would like.

Example :-

**DATAFRAME**-It is a two-dimensional object that is useful in representing data in the form of rows and columns. It is similar to a spreadsheet or an SQL table. This is the most commonly used pandas object. Once we store the data into the Dataframe, we can perform various operations that are useful in analyzing and understanding the data.

**DATAFRAME**

#### DATAFRAME STRUCTURE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **COLUMNS** | | **PLAYERNAME** | **IPLTEAM** | **BASEPRICEINCR** |
|  | | | | |
| 0 |  | ROHIT | MI | 13 |
| 1 | VIRAT | RCB | 17 |
| 2 | HARDIK | MI | 14 |

**INDEX DATA**

**PROPERTIES OF DATAFRAME**

1. A Dataframe has axes (indices)-
   * Row index (axis=0)
   * Column index (axes=1)
2. It is similar to a spreadsheet , whose row index is called index and column index is called column name.
3. A Dataframe contains Heterogeneous data.
4. A Dataframe Size is Mutable.
5. A Dataframe Data is Mutable.

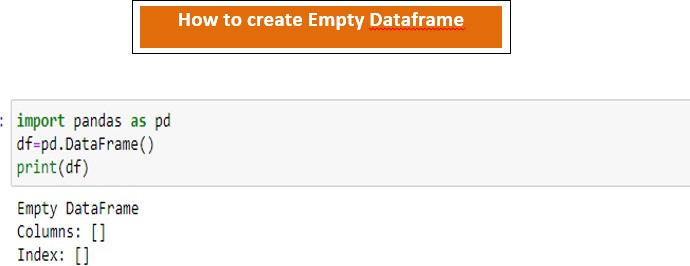
**A data frame can be created using any of the following-**

1. **Series**
2. **Lists**
3. **Dictionary**
4. **A numpy 2D array**

**Output-**

0

1. a
2. b **Default Column Name As 0**
3. c
4. d

****

**How to create Dataframe From Series**

**Program-**

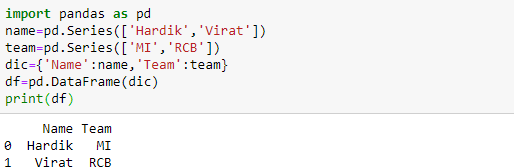
**import pandas as pd**

**s = pd.Series(['a','b','c','d']) df=pd.DataFrame(s)**

**print(df)**

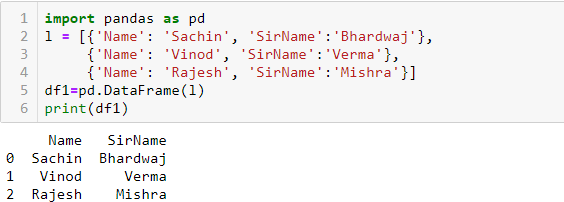
## DataFrame from Dictionary of Series

Example-



## DataFrame from List of Dictionaries

Example-



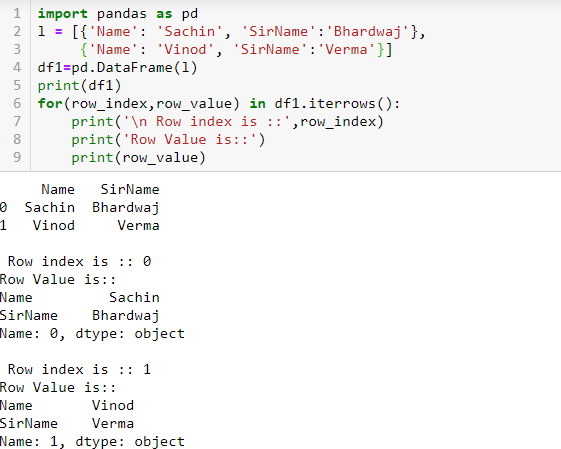
## Iteration on Rows and Columns

If we want to access record or data from a data frame row wise or column wise then iteration is used. Pandas provide 2 functions to perform iterations-

1. iterrows ()
2. iteritems ()

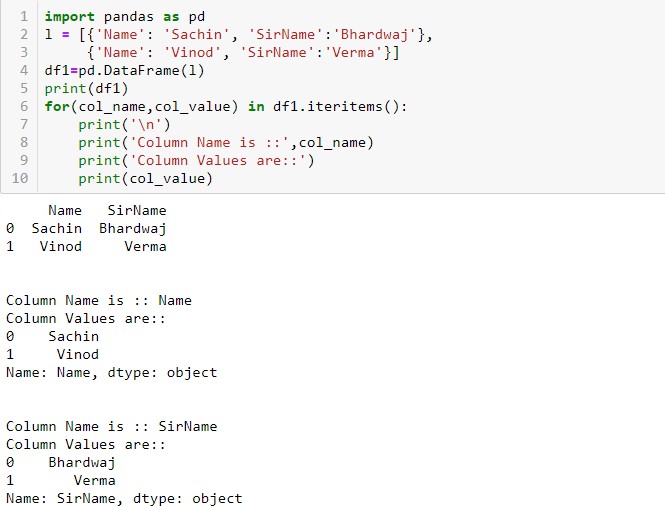
## iterrows()

It is used to access the data row wise. Example-



## iteritems()

It is used to access the data column wise. Example-



# Select operation in data frame

**To access the column data ,we can mention the column name as subscript.**

**e.g. - df[empid] This can also be done by using df.empid.**

**To access multiple columns we can write as df[ [col1, col2,---] ]**

**Example -**

****

**>>df.empid or df[‘empid’]**

**0 101**

**1 102**

**2 103**

**3 104**

**4 105**

**5 106**

**Name: empid, dtype: int64**

**>>df[[‘empid’,’ename’]]**

**5**

**106**

**UmaSelvi**

|  |  |  |
| --- | --- | --- |
|  | **empid** | **ename** |
| **0** | **101** | **Sachin** |
| **1** | **102** | **Vinod** |
| **2** | **103** | **Lakhbir** |
| **3** | **104** | **Anil** |
| **4** | **105** | **Devinder** |

# To Add & Rename a column in data frame

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **import pandas as pd**  **s = pd.Series([10,15,18,22]) df=pd.DataFrame(s)**  **df.columns=[‘List1’] To Rename the default column of Data Frame as List1**  **df[‘List2’]=20 To create a new column List2 with all values as 20** | | | | |
|  |  | | | |
| **df[‘List3’]=df[‘List1’]+df[‘List2’]** | **Output-** | | | |
| **Add Column1 and Column2 and store in** | **List1 List2 List3**  **0 10 20 30** | | | |
| **New column List3** | **1 15 20 35**  **2 18 20 38** | | | |
| **print(df)** | **3** | **22** | **20** | **42** |

**To Delete a Column in data frame**

**We can delete the column from a data frame by using any of the the following –**

1. **del**
2. **pop()**
3. **drop()**

**>>del df[‘List3’]**

**We can simply delete a column by passing**

**column name in subscript with df**

**>>df**

|  |
| --- |
| **Output-** |
| **List1 List2**  **0 10 20**  **1 15 20**  **2 18 20**  **3 22 20** |
| **>>df.pop(‘List2’) we can simply delete a column by passing column name in pop method.**  **>>df** |
| **List1**  **0 10**  **1 15**  **2 18**  **3 22** |

# To Delete a Column Using drop()

**import pandas as pd**

**s= pd.Series([10,20,30,40]) df=pd.DataFrame(s) df.columns=[‘List1’] df[‘List2’]=40**

**df1=df.drop(‘List2’,axis=1) (axis=1) means to delete Data**

**column wise df2=df.drop(index=[2,3],axis=0) (axis=0) means to delete**

**data row wise with given index**

**print(df)**

**print(“ After deletion::”)**

**print(df1)**

**print (“ After row deletion::”)**

**print(df2)**

**Output-**

|  |  |
| --- | --- |
| **List1** | **List2** |
| **0 10** | **40** |
| **1 20** | **40** |
| **2 30** | **40** |
| **3 40** | **40** |

**After deletion:: List1**

**0 10**

**1 20**

**2 30**

**3 40**

**After row deletion::**

**List1**

**0 10**

**1 20**

## Accessing the data frame through loc() and iloc() method or indexing using Labels

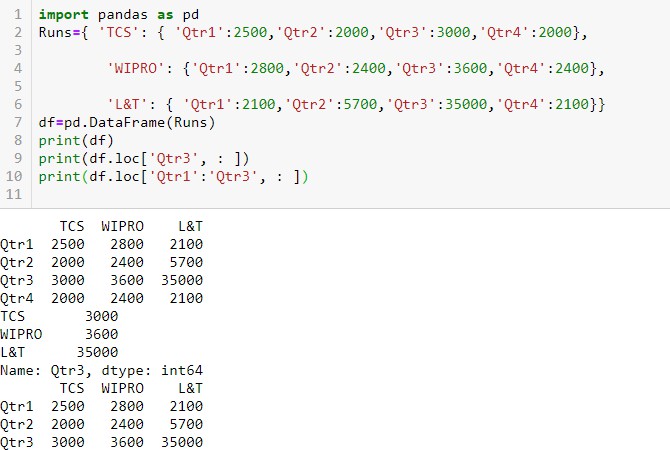
Pandas provide loc() and iloc() methods to access the subset from a data frame using row/column.

## Accessing the data frame through loc()

It is used to access a group of rows and columns. Syntax-

##### Df.loc[StartRow : EndRow, StartColumn : EndColumn]

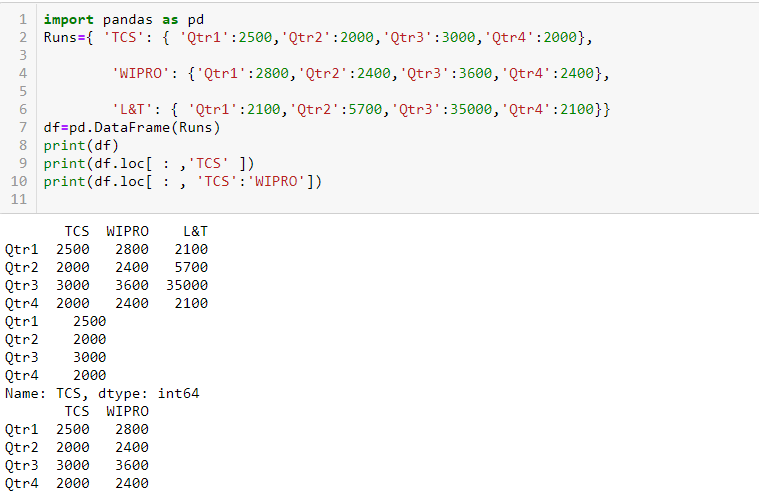
Note -If we pass : in row or column part then pandas provide the entire rows or columns respectively.



To access multiple Rows Qtr1 to Qtr3

To access a single row

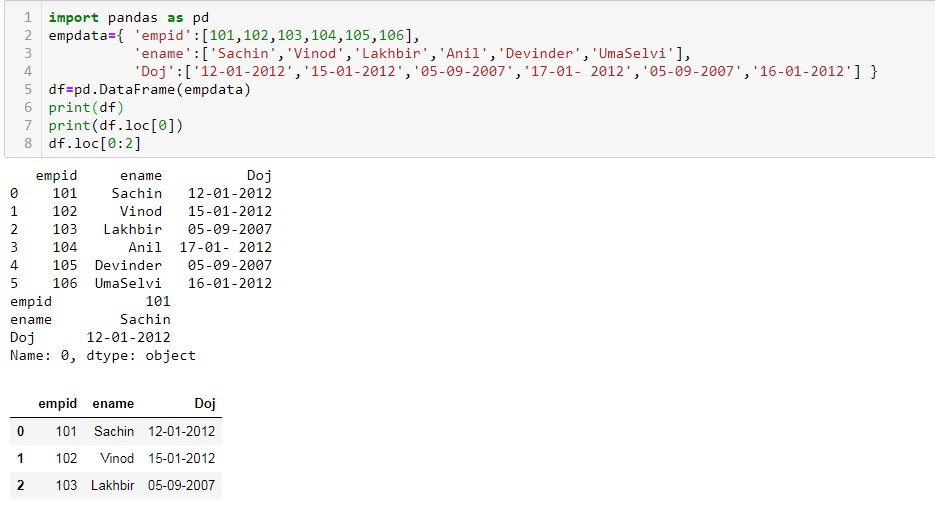
Example 2:-



To access single column

To access Multiple Column namely TCS and WIPRO

Example-3



To access first row

To access first 3 Rows

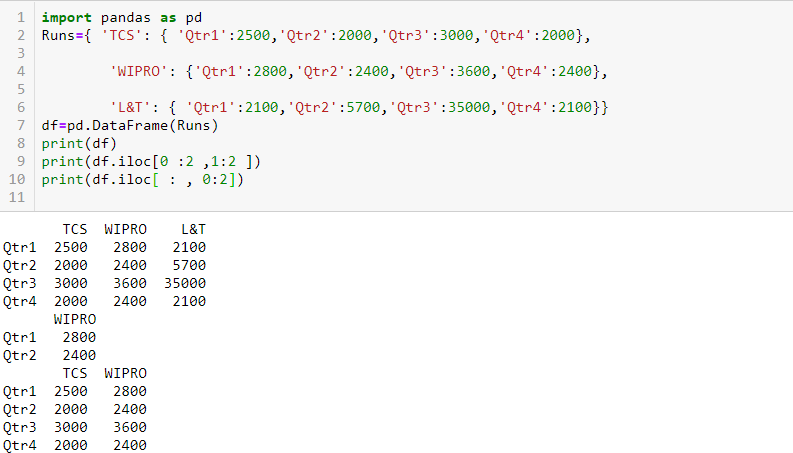
## Accessing the data frame through iloc()

It is used to access a group of rows and columns based on numeric index value.

Syntax-

Df.loc[StartRowindexs : EndRowindex, StartColumnindex : EndColumnindex]

Note -If we pass : in row or column part then pandas provide the entire rows or columns respectively.



To access all Rows and First Two columns Record

To access First two Rows and Second column

**head() and tail() Method s**

The method head() gives the first 5 rows and the method tail() returns the last 5 rows.

**Visit Python4csip.com for more update**

**import pandas as pd**

**empdata={ 'Doj':['12-01-2012','15-01-2012','05-09-2007',**

**'17-01-2012','05-09-2007','16-01-2012'], 'empid':[101,102,103,104,105,106],**

**'ename':['Sachin','Vinod','Lakhbir','Anil','Devinder','UmaSelvi']**

**}**

**df=pd.DataFrame(empdata) print(df)**

**print(df.head())**

**print(df.tail())**

Output-

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **0**  **1**  **2**  **3**  **4**  **5** | **Doj empid 12-01-2012 101**  **15-01-2012 102**  **05-09-2007 103**  **17-01-2012 104**  **05-09-2007 105**  **16-01-2012 106** | | **ename Sachin Vinod Lakhbir**  **Anil**  **Devinder UmaSelvi** | **Data Frame** |
|  | **Doj empid** | | **ename** |  |
| **0** | **12-01-2012** | **101** | **Sachin** |  |
| **1** | **15-01-2012** | **102** | **Vinod** | **head() displays first 5 rows** |
| **2** | **05-09-2007** | **103** | **Lakhbir** |  |
| **3** | **17-01-2012** | **104** | **Anil** |  |
| **4** | **05-09-2007** | **105** | **Devinder** |  |
|  | **Doj empid** | | **ename** |  |
| **1** | **15-01-2012** | **102** | **Vinod** |  |
| **2** | **05-09-2007** | **103** | **Lakhbir** |  |
| **3** | **17-01-2012** | **104** | **Anil** | **tail() display last 5 rows** |
| **4** | **05-09-2007** | **105** | **Devinder** |  |
| **5** | **16-01-2012** | **106** | **UmaSelvi** |  |

**import pandas as pd**

**empdata={ 'Doj':['12-01-2012','15-01-2012','05-09-2007',**

**'17-01-2012','05-09-2007','16-01-2012'], 'empid':[101,102,103,104,105,106],**

**'ename':['Sachin','Vinod','Lakhbir','Anil','Devinder','UmaSelvi']**

**}**

**df=pd.DataFrame(empdata) print(df)**

**print(df.head(2)) print(df.tail(2)) print(df[2:5])**

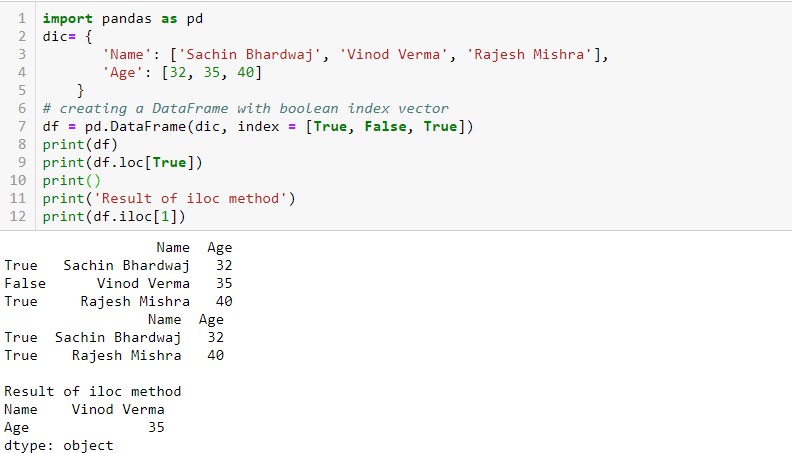
**To display first 2 rows we can use head(2) and to returns last2 rows we can use tail(2) and to return 3rd to 4th row we can write df[2:5].**

Output-

|  |  |  |
| --- | --- | --- |
|  | **Doj empid ename** |  |
| **0** | **12-01-2012 101 Sachin** |  |
| **1** | **15-01-2012 102 Vinod** |  |
| **2** | **05-09-2007 103 Lakhbir** |  |
| **3** | **17-01- 2012 104 Anil** |  |
| **4** | **05-09-2007 105 Devinder** |  |
| **5** | **16-01-2012 106 UmaSelvi** |  |
|  | **Doj empid ename** |  |
| **0** | **12-01-2012 101 Sachin** | **head(2) displays first 2 rows** |
| **1** | **15-01-2012 102 Vinod** |  |
|  | **Doj empid ename** |  |
| **4** | **05-09-2007 105 Devinder** | **tail(2) displays last 2 rows** |
| **5** | **16-01-2012 106 UmaSelvi** |  |
|  | **Doj empid ename** |  |
| **2** | **05-09-2007 103 Lakhbir** |  |
| **3** | **17-01- 2012 104 Anil** | **df[2:5] display 2nd to 4th row** |
| **4** | **05-09-2007 105 Devinder** |  |

## Boolean Indexing in Data Frame

Boolean indexing helps us to select the data from the DataFrames using a boolean vector. We create a DataFrame with a boolean index to use the boolean indexing.



**We can pass only integer value in iloc**

**To Return Data frame where index is True**

### Concat operation in data frame

Pandas provides various facilities for easily combining together **Series, DataFrame.**

pd.concat(objs, axis=0, join='outer', join\_axes=None,ignore\_index=False)

* **objs** − This is a sequence or mapping of Series, DataFrame, or

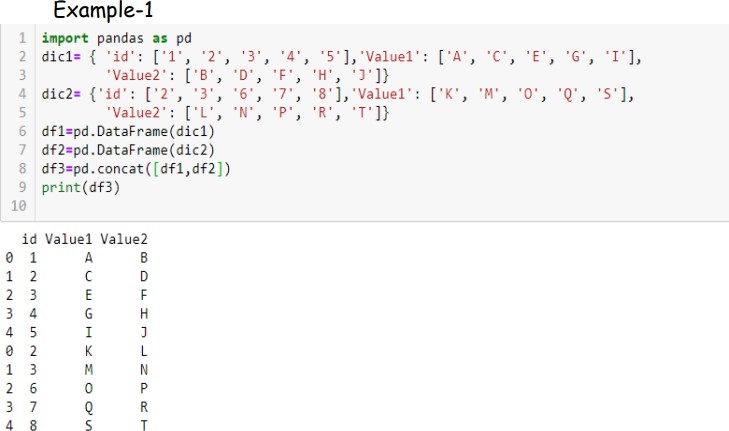
Panel objects.

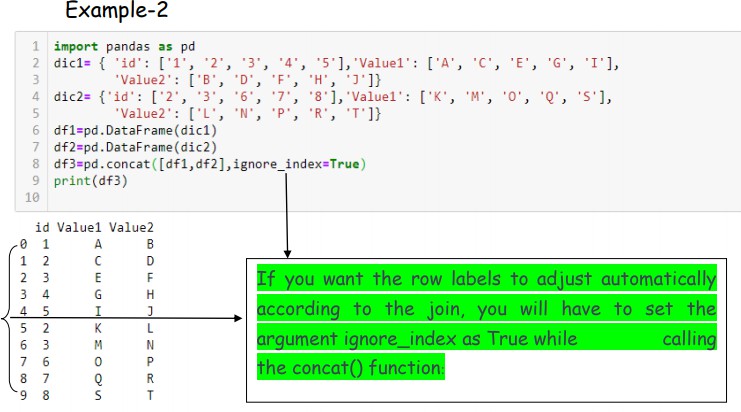
* **axis** − {0, 1, ...}, default 0. This is the axis to concatenate along.
* **join** − {‘inner’, ‘outer’}, default ‘outer’. How to handle indexes on

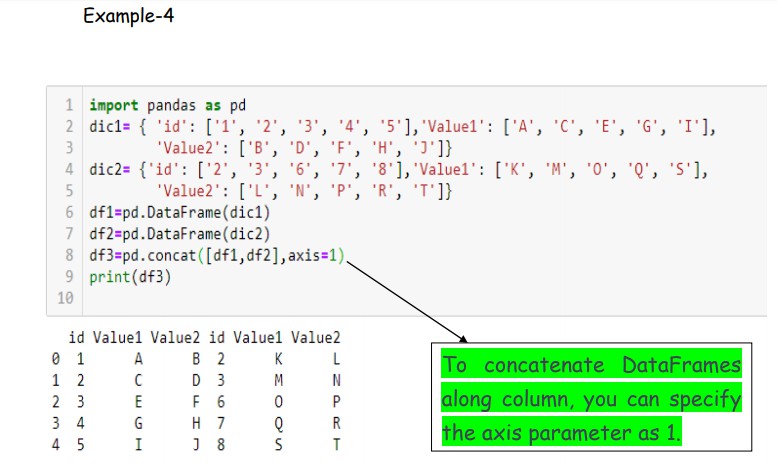
other axis(es). Outer for union and inner for intersection.

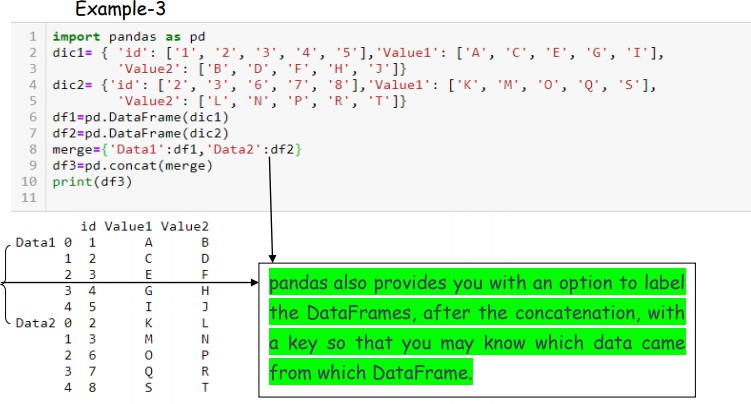
* **ignore\_index** − boolean, default False. If True, do not use the index values on the concatenation axis. The resulting axis will be labeled 0, ..., n - 1.
* **join\_axes** − This is the list of Index objects. Specific indexes to use for the other (n-1) axes instead of performing inner/outer set logic.

The Concat() performs concatenation operations along an axis.





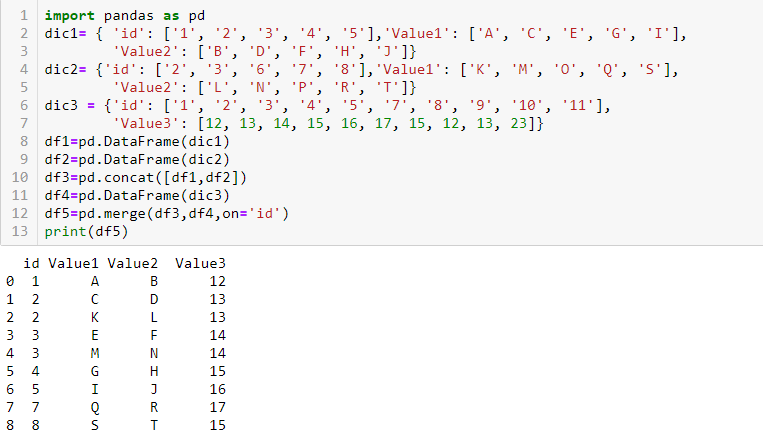




### Merge operation in data frame

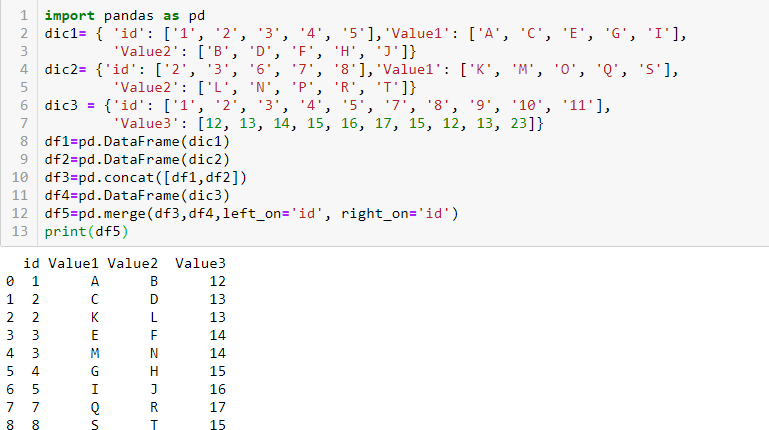
Two DataFrames might hold different kinds of information about the same entity and linked by some common feature/column. To join these DataFrames, pandas provides multiple functions like merge(), join() etc.

Example-1



This will give the common rows between the two data frames for the corresponding column values (‘id’).

Example-2



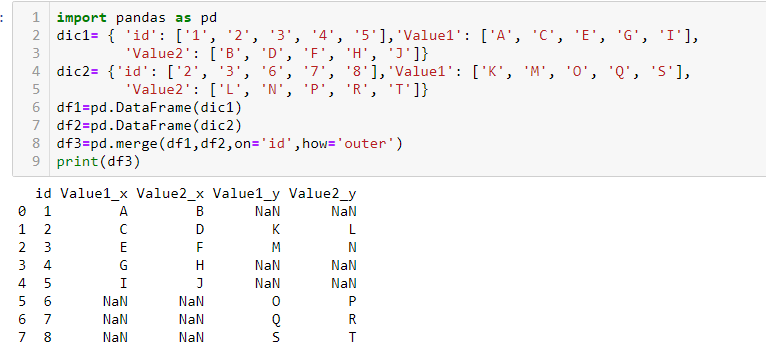
It might happen that the column on which you want to merge the Data Frames have different names (unlike in this case). For such merges, you will have to specify the arguments left\_on as the left DataFrame name and right\_on as the right DataFrame name.

### Join operation in data frame

It is used to merge data frames based on some common column/key.

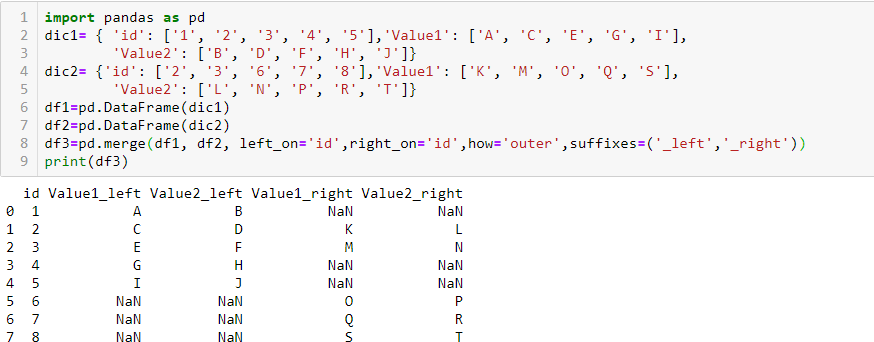
1. **Full Outer Join**:- The full outer join combines the results of both the left and the right outer joins. The joined data frame will contain all records from both the data frames and fill in NaNs for missing matches on either side. You can perform a full outer join by specifying the how argument as outer in merge() function.

Example-



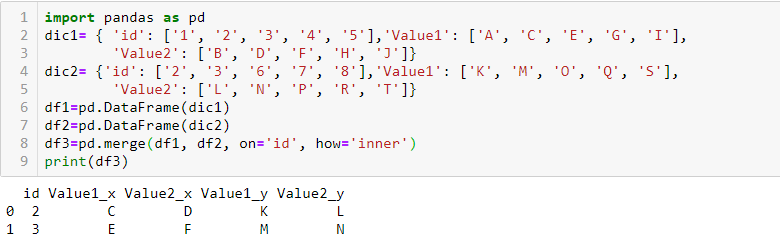
**The resulting DataFrame had all the entries from both the tables with NaN values for missing matches on either side. However, one more thing to notice is the suffix which got appended to the column names to show which column came from which DataFrame. The default suffixes are x and y, however, you can modify them by specifying the suffixes argument in the merge() function.**

Example-2



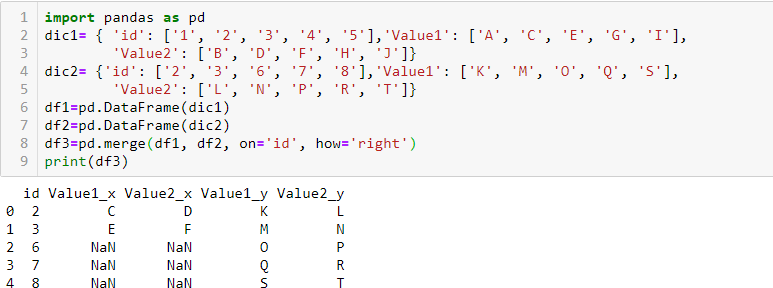
1. **Inner Join** :- The inner join produce only those records that match in both the data frame. You have to pass inner in how argument inside merge() function.

Example-



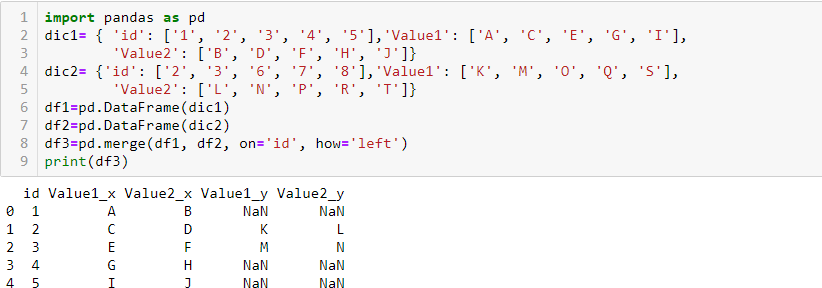
1. **RightJoin** :-The right join produce a complete set of records from data frame B(Right side Data Frame) with the matching records (where available) in data frame A( Left side data frame). If there is no match right side will contain null. You have to pass right in how argument inside merge() function.

Example-



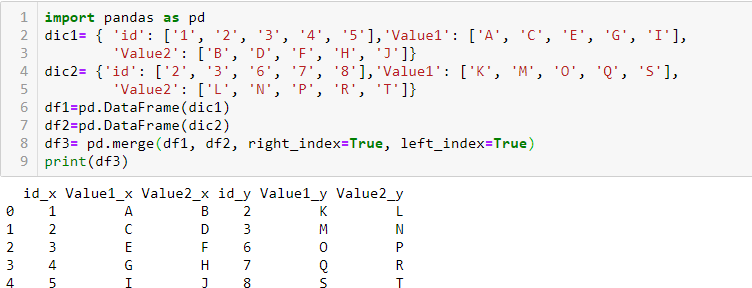
1. **Left Join** :- The left join produce a complete set of records from data frame A(Left side Data Frame) with the matching records (where available) in data frame B( Right side data frame). If there is no match left side will contain null. You have to pass left in how argument inside merge() function.

Example-



1. **Joining on Index** :-Sometimes you have to perform the join on the indexes or the row labels. For that you have to specify right\_index( for the indexes of the right data frame ) and left\_index( for the indexes of left data frame) as True.

Example-

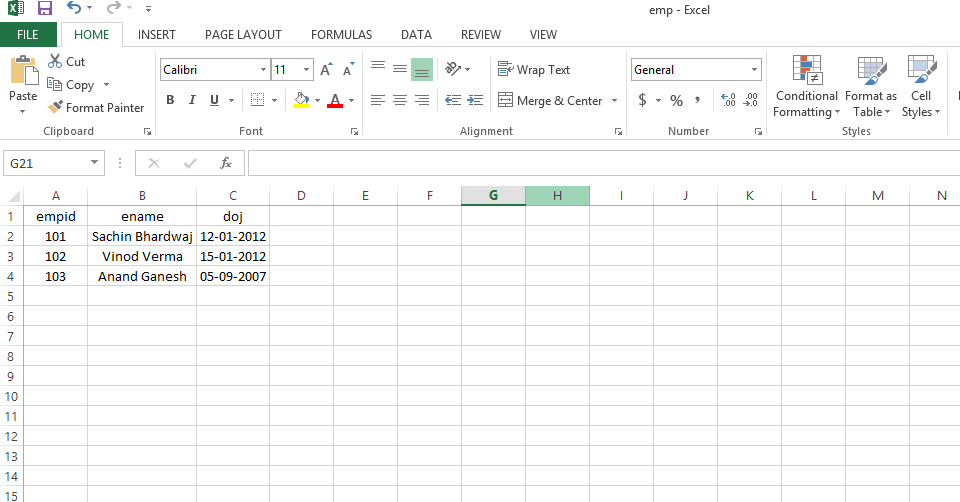


# CSV File

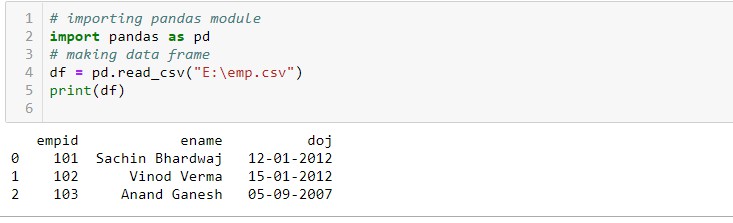
A CSV is a comma separated values file, which allows data to be saved in a tabular format. CSV is a simple file such as a spreadsheet or database. Files in the csv format can be imported and exported from programs that store data in tables, such as Microsoft excel or Open Office.

CSV files data fields are most often separated, or delimited by a comma. Here the data in each row are delimited by comma and individual rows are separated by newline.

To create a csv file, first choose your favorite text editor such as- Notepad and open a new file. Then enter the text data you want the file to contain, separating each value with a comma and each row with a new line. Save the file with the extension.csv. You can open the file using MS Excel or another spread sheet program. It will create the table of similar data.

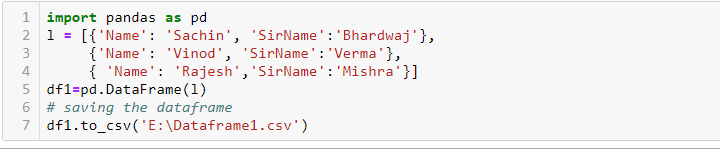


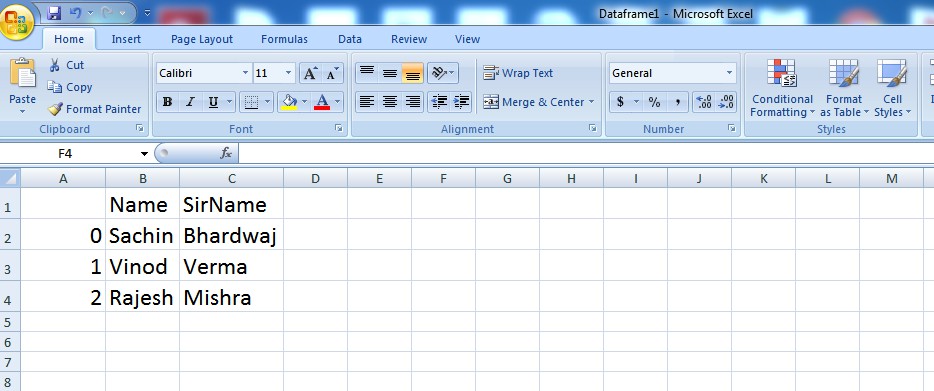
pd.read\_csv() method is used to read a csv file.



# Exporting data from dataframe to CSV File

To export a data frame into a csv file first of all, we create a data frame say df1 and use dataframe.to\_csv(‘ E:\Dataframe1.csv ’ ) method to export data frame df1 into csv file Dataframe1.csv.





And now the content of df1 is exported to csv file Dataframe1.