PANDAS

Pandas stand for Panel Data System

Pandas is an open source library for data analysis, Data manipulation and Data Visualization.

(OR) Pandas provide powerful data structures for data analysis, time series and statistics.

Pandas works on the top numpy and matplotlib.

Features of pandas

1. Handling huge amount data
2. Missing Data
3. Cleaning up data
4. Alignment and indexing
5. Merging and joining
6. Grouping and Visualizing data
7. Time Series Functionality
8. Allows to load data from multiple file formats
9. Input and Output Tools

Pandas library is used by scikit-learn for ML

Applications of Pandas

1. Recommendation Systems
2. Stock Prediction
3. Big Data and Data Science
4. NLP (Natural Language Processing)
5. Statistics and Analytics
6. Neuroscience

Important data structures of Pandas are,

1. Series
2. DataFrame

Q: What is data analysis?

Data analysis is process of collecting, transforming, cleaning and modeling the data with goal of discovering required information.

Data analysis process consists of the following steps.

1. Data Requirement Specifications
2. Data Collection
3. Data Processing
4. Data Cleaning

5. Data Analysis
6. Communication

What is Series?

Pandas series is a one dimensional array object, this object can hold data of any type. It can be integers, floats, string or python objects.

Pandas series represents or equal to a column in any data base (MsExcel, Oracle, MySQL, SQLServer,..)

What is DataFrame?

DataFrame is a two dimensional array object or data structure. Data stored tabular format, which is rows and columns.

The Dataframe consist of 3 components.

1. Data
2. Rows
3. Columns

How to install pandas?

Other than jupyter and googlecolab, it is required to install pandas lib.

```
pip install pandas
```

Pandas Series

Series is single dimension array like object with homogeneous or heterogeneous data.

Series object can be created in different ways.

1. Using array
2. Using Dictionary
3. Using Scalar values
4. Using other iterables

Series is name of the class or type which is used to construct Series object.

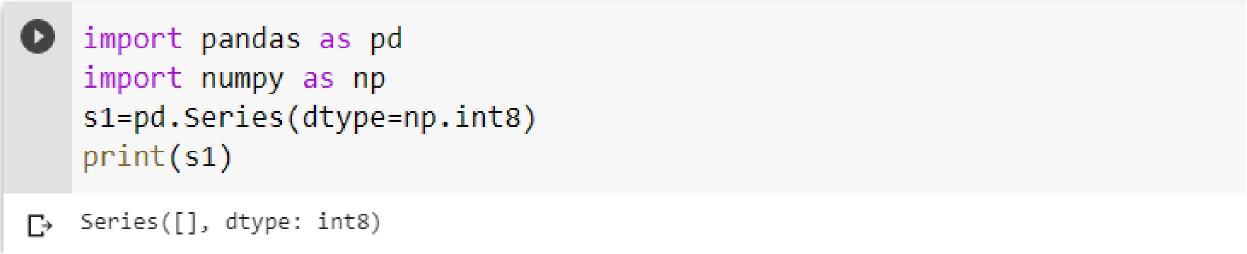
Syntax: Series(data,index,dtype)

Data : the source using which series object is created

Index : index values must hashable and must be unique

dtype: type of the series is defined using dtype.

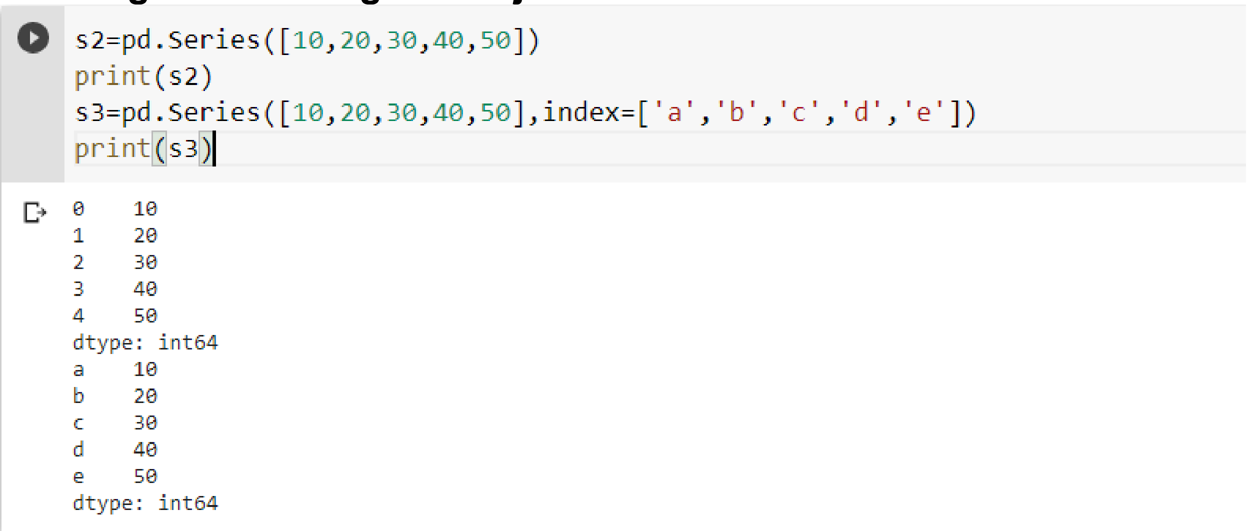
Creating Empty Series



```
import pandas as pd
import numpy as np
s1=pd.Series(dtype=np.int8)
print(s1)
```

```
Series([], dtype: int8)
```

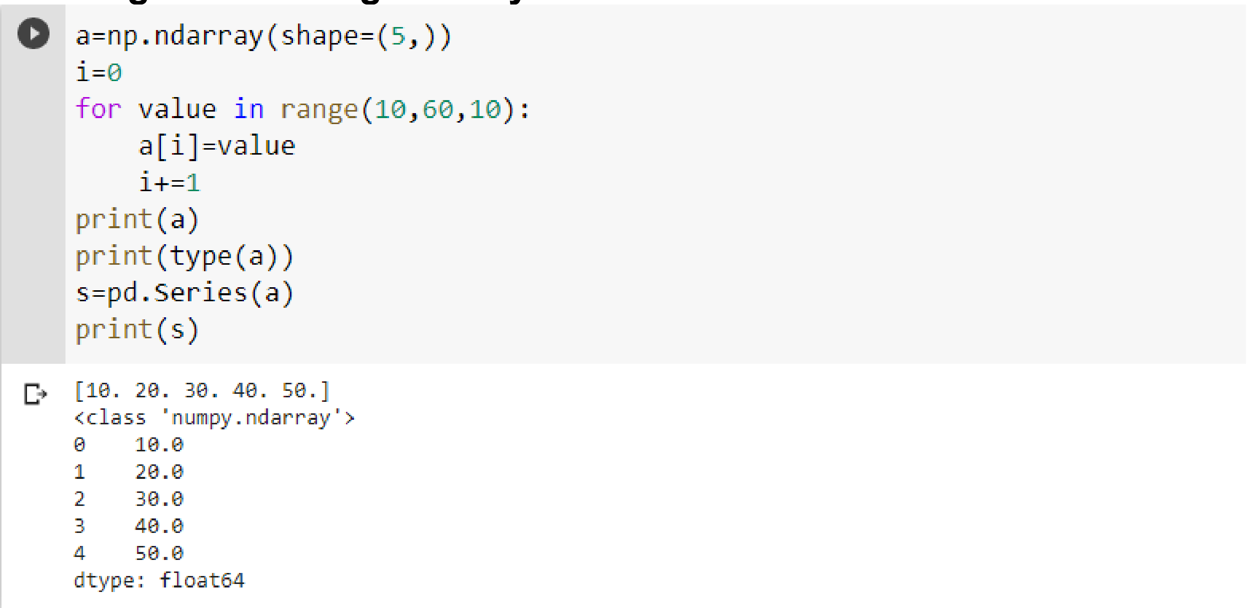
Creating Series using List object



```
s2=pd.Series([10,20,30,40,50])
print(s2)
s3=pd.Series([10,20,30,40,50],index=['a','b','c','d','e'])
print(s3)
```

```
0    10
1    20
2    30
3    40
4    50
dtype: int64
a    10
b    20
c    30
d    40
e    50
dtype: int64
```

Creating Series using ndarray



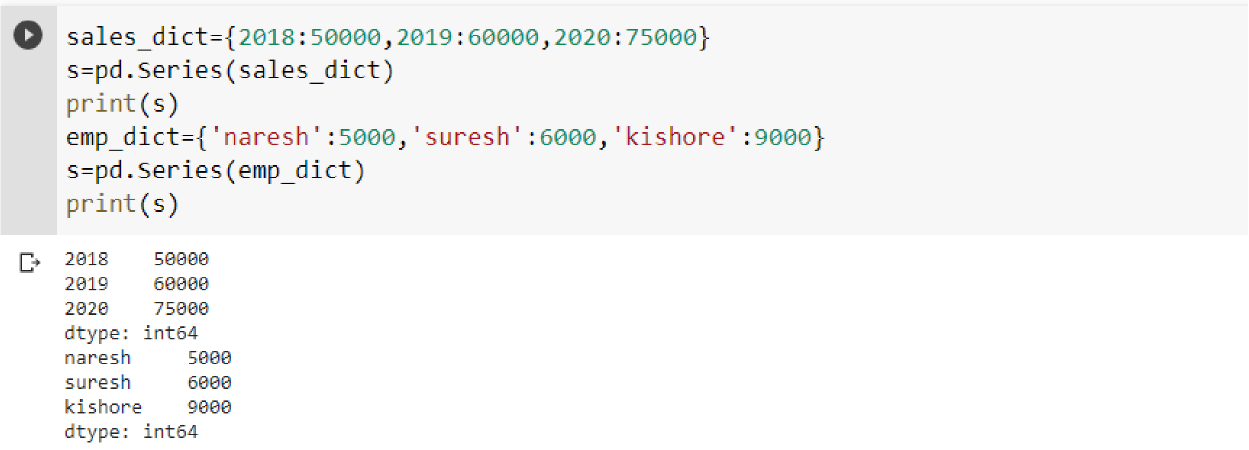
```
a=np.ndarray(shape=(5,))
i=0
for value in range(10,60,10):
    a[i]=value
    i+=1
print(a)
print(type(a))
s=pd.Series(a)
print(s)
```

```
[10. 20. 30. 40. 50.]
<class 'numpy.ndarray'>
0    10.0
1    20.0
2    30.0
3    40.0
4    50.0
dtype: float64
```

Creating Series Using Dictionary

We can create series using dictionary (OR) we can pass the dictionary object to series.

Series object is using dictionary values as data and dictionary keys as index labels.

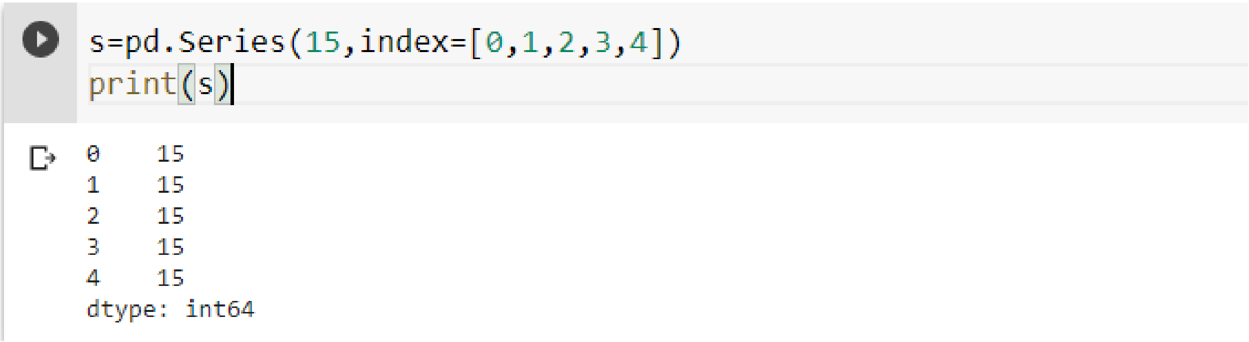


```
sales_dict={2018:50000,2019:60000,2020:75000}
s=pd.Series(sales_dict)
print(s)
emp_dict={'naresh':5000,'suresh':6000,'kishore':9000}
s=pd.Series(emp_dict)
print(s)
```

```
2018    50000
2019    60000
2020    75000
dtype: int64
naresh     5000
suresh     6000
kishore    9000
dtype: int64
```

Creating Series using Scalar values

If the series is created using scalar values we must define index. This index defines the length of series.



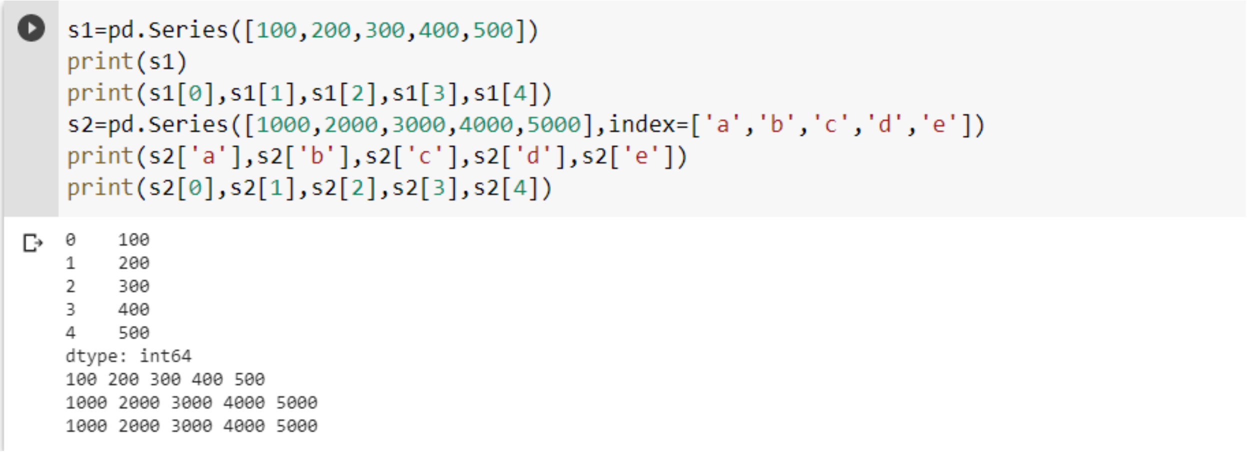
```
s=pd.Series(15,index=[0,1,2,3,4])
print(s)
```

```
0    15
1    15
2    15
3    15
4    15
dtype: int64
```

Accessing Data from Series

Series is index based collection, we can read and manipulate data using index.

This index starts with 0.

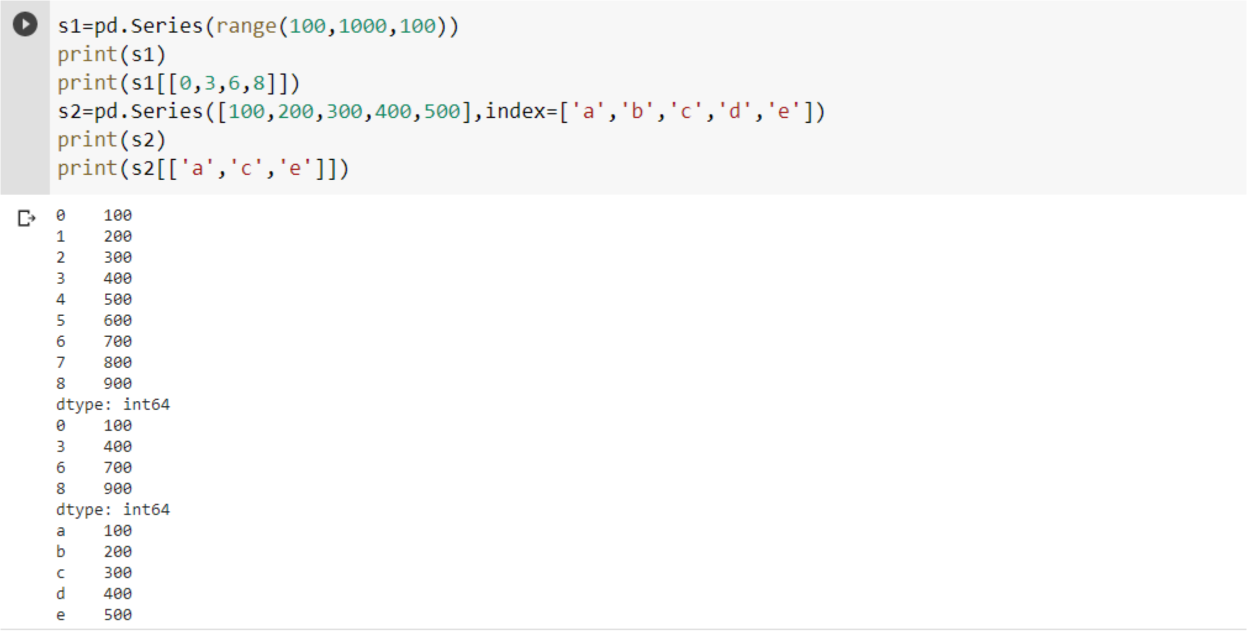


```
s1=pd.Series([100,200,300,400,500])
print(s1)
print(s1[0],s1[1],s1[2],s1[3],s1[4])
s2=pd.Series([1000,2000,3000,4000,5000],index=['a','b','c','d','e'])
print(s2['a'],s2['b'],s2['c'],s2['d'],s2['e'])
print(s2[0],s2[1],s2[2],s2[3],s2[4])
```

```
0    100
1    200
2    300
3    400
4    500
dtype: int64
100 200 300 400 500
1000 2000 3000 4000 5000
1000 2000 3000 4000 5000
```

Reading multiple elements/values from series

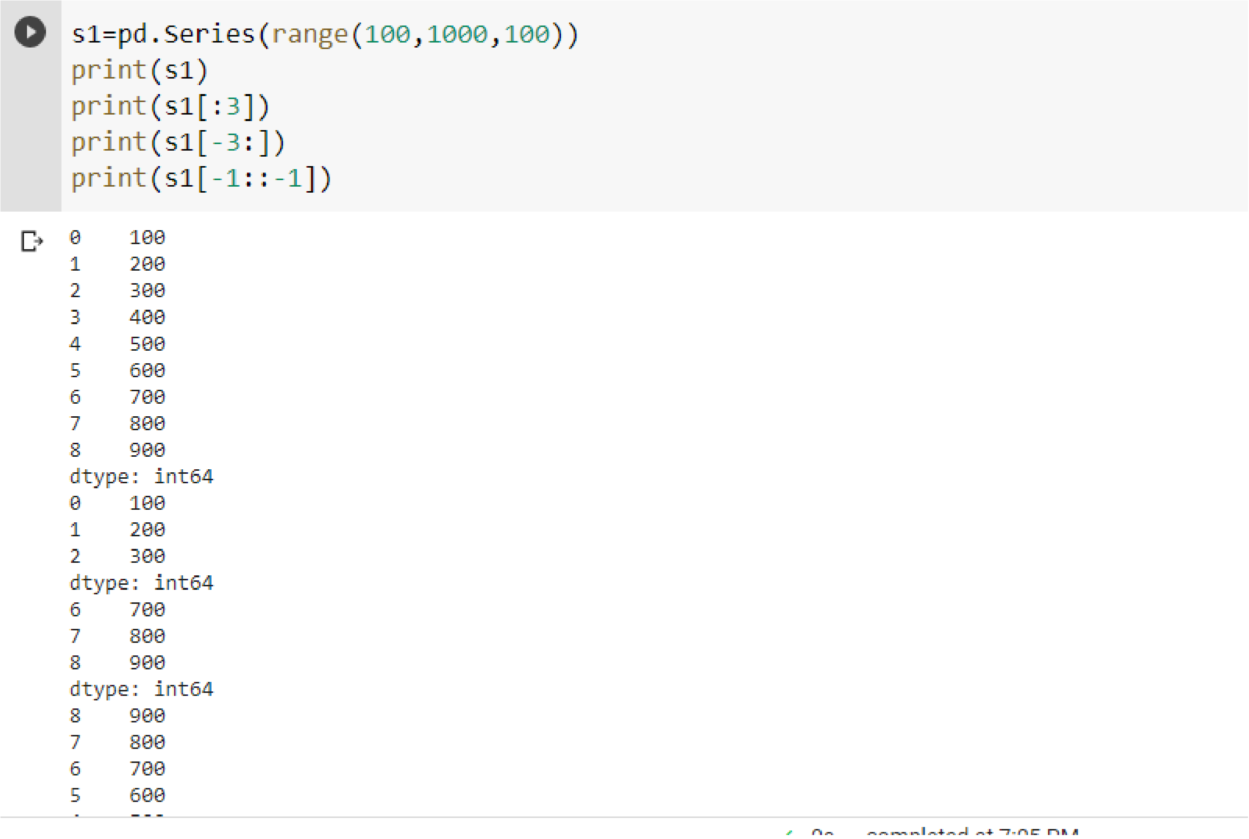
Series allows reading multiple elements by defining index labels within list.



```
s1=pd.Series(range(100,1000,100))
print(s1)
print(s1[[0,3,6,8]])
s2=pd.Series([100,200,300,400,500],index=['a','b','c','d','e'])
print(s2)
print(s2[['a','c','e']])
```

```
0    100
1    200
2    300
3    400
4    500
5    600
6    700
7    800
8    900
dtype: int64
0    100
3    400
6    700
8    900
dtype: int64
a    100
b    200
c    300
d    400
e    500
```

Series allows slicing, to read multiple elements/values.



```
s1=pd.Series(range(100,1000,100))
print(s1)
print(s1[:3])
print(s1[-3:])
print(s1[-1::-1])
```

```
0    100
1    200
2    300
3    400
4    500
5    600
6    700
7    800
8    900
dtype: int64
0    100
1    200
2    300
dtype: int64
6    700
7    800
8    900
dtype: int64
8    900
7    800
6    700
5    600
```

DataFrame

DataFrame is two dimensional array object with heterogeneous data. In DataFrame data is stored in the form of rows and columns.

How to create DataFrame?

DataFrame can be created in different ways.

1. Series
2. Lists
3. Dictionary
4. Numpy array
5. From another dataframe
6. Data can read from files or database

“DataFrame” is type or class name, to create dataframe object

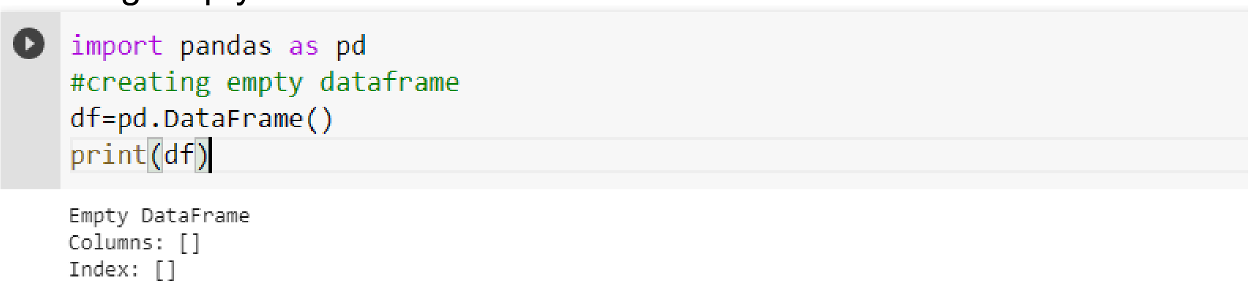
Syntax:

DataFrame(data,index,columns,dtype)

data : data is taken from various sources
Index : row labels
columns : columns labels

dtype: data type of each column

Creating empty dataframe

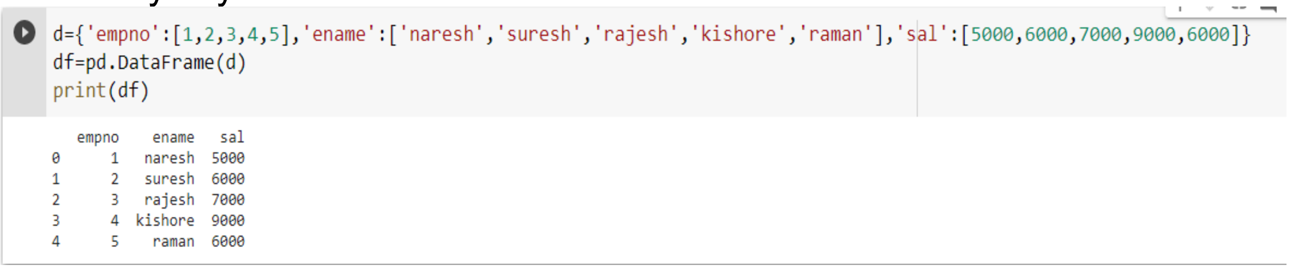


```
import pandas as pd
#creating empty dataframe
df=pd.DataFrame()
print(df)
```

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

Creating DataFrame using dictionary

Dictionary consist of key and values.
Dictionary keys as columns headers and values are columns values

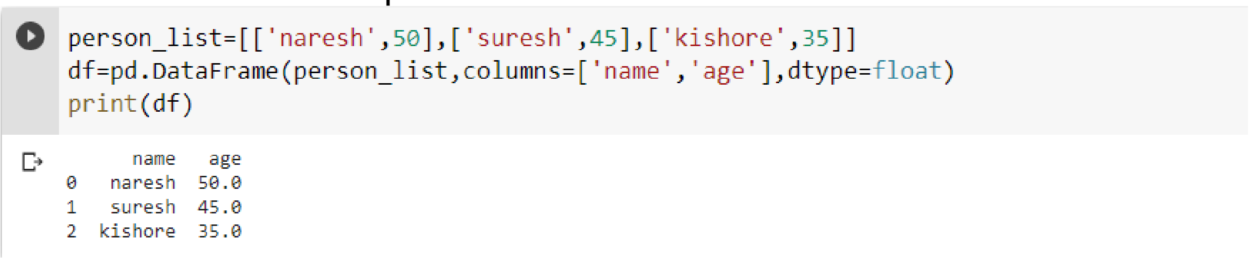


```
d={'empno':[1,2,3,4,5],'ename':['naresh','suresh','rajesh','kishore','raman'],'sal':[5000,6000,7000,9000,6000]}
df=pd.DataFrame(d)
print(df)
```

```
   empno    ename   sal
0      1   naresh  5000
1      2   suresh  6000
2      3   rajesh  7000
3      4  kishore  9000
4      5    raman  6000
```

Create DataFrame using List

A nested list represents the content of dataframe.
Each list within list is represented as row.

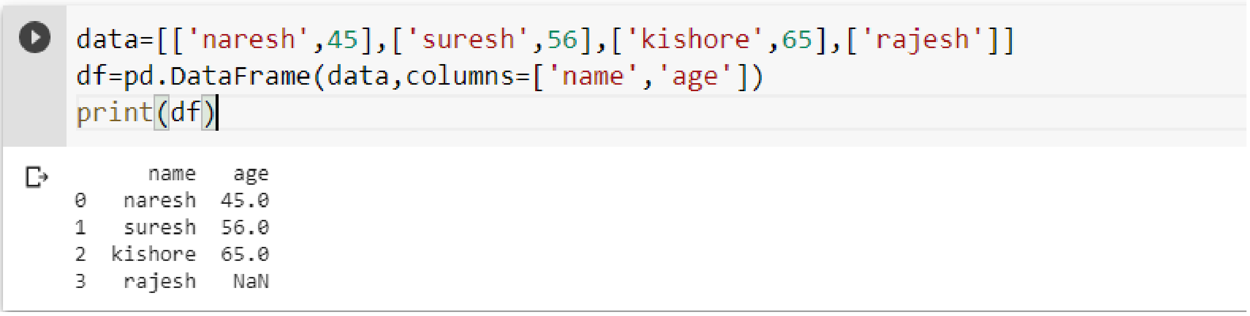


```
person_list=[['naresh',50],['suresh',45],['kishore',35]]
df=pd.DataFrame(person_list,columns=['name','age'],dtype=float)
print(df)
```

```
      name   age
0   naresh  50.0
1   suresh  45.0
2  kishore  35.0
```

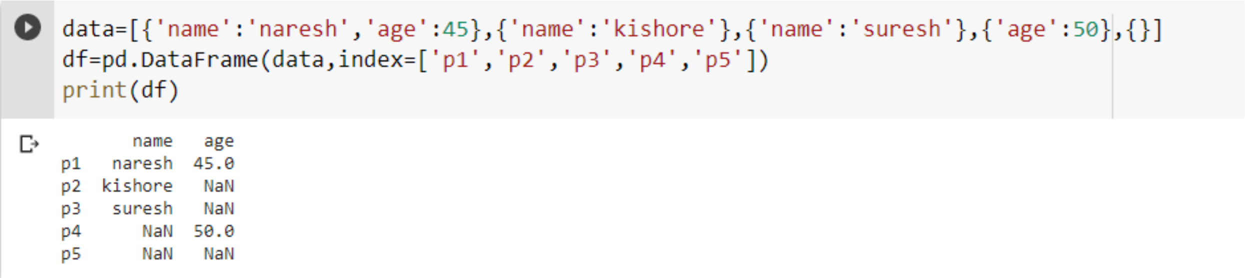
DataFrame created with missing data

Missing data is identified with NaN(Not a Number)



```
data=[['naresh',45],['suresh',56],['kishore',65],['rajesh']]
df=pd.DataFrame(data,columns=['name','age'])
print(df)
```

```
      name   age
0   naresh  45.0
1   suresh  56.0
2  kishore  65.0
3   rajesh   NaN
```



```
data=[{'name':'naresh','age':45},{'name':'kishore'},{'name':'suresh'},{'age':50},{}]
df=pd.DataFrame(data,index=['p1','p2','p3','p4','p5'])
print(df)
```

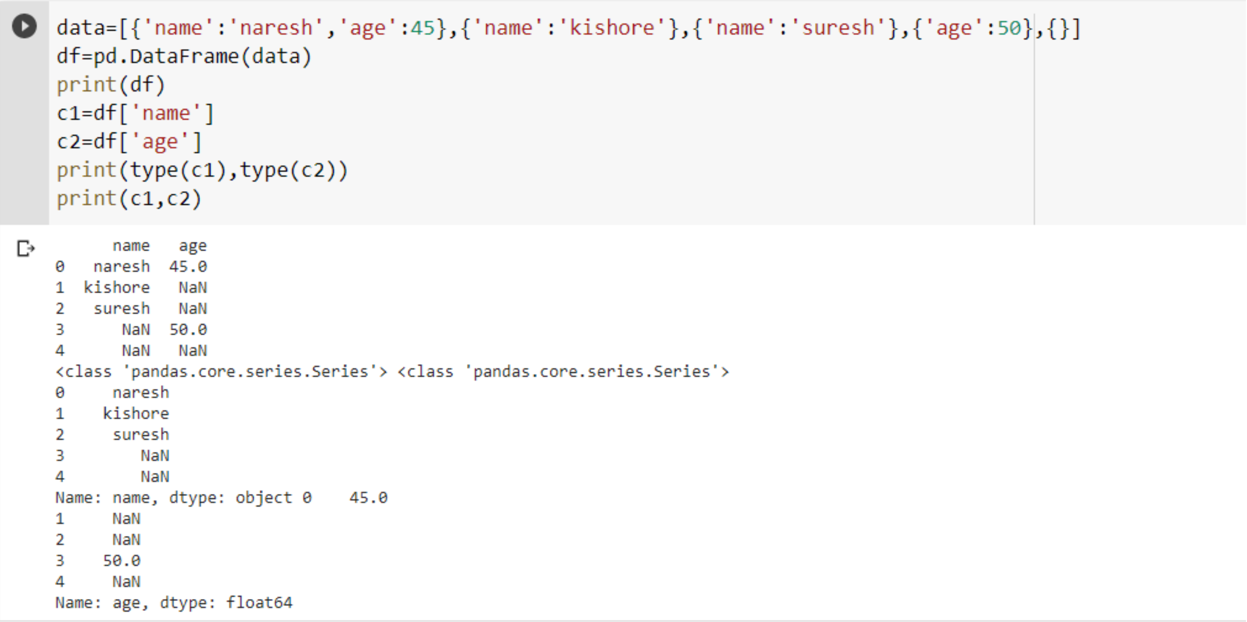
```
      name   age
p1   naresh  45.0
p2  kishore   NaN
p3   suresh   NaN
p4      NaN  50.0
p5      NaN   NaN
```

Selecting Data

1. Row Selection
2. Column Selection

Column Selection

Selecting columns from DataFrame can be done using column header.



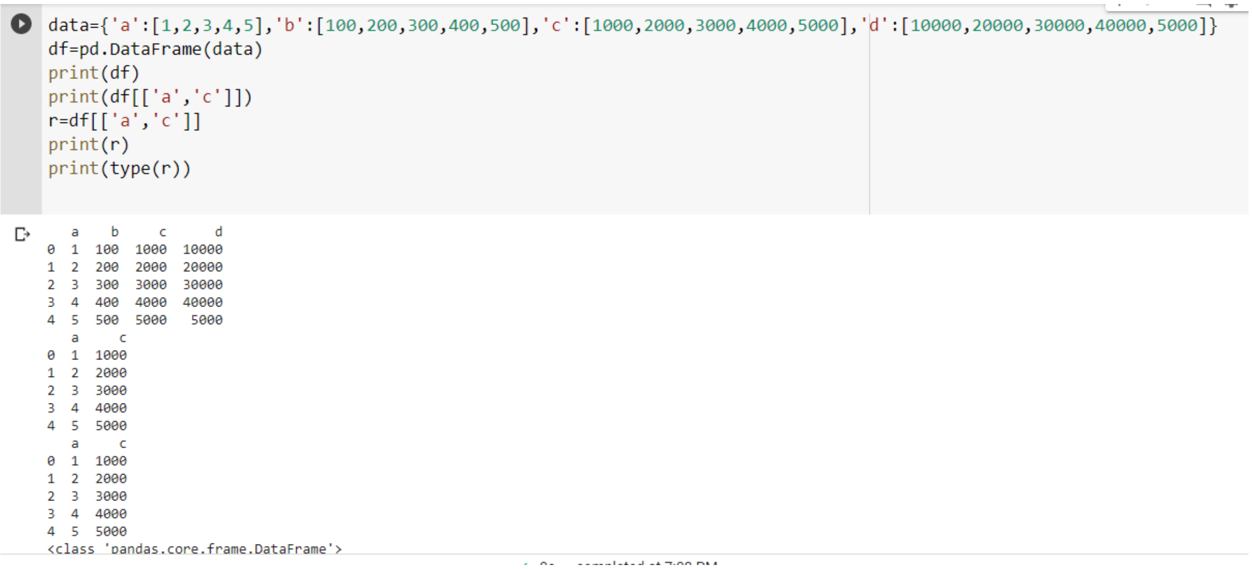
```
data=[{'name':'naresh','age':45},{'name':'kishore'},{'name':'suresh'},{'age':50},{}]
df=pd.DataFrame(data)
print(df)
c1=df['name']
c2=df['age']
print(type(c1),type(c2))
print(c1,c2)
```

```
      name   age
0   naresh  45.0
1  kishore   NaN
2   suresh   NaN
3      NaN  50.0
4      NaN   NaN
<class 'pandas.core.series.Series'> <class 'pandas.core.series.Series'>
0     naresh
1    kishore
2     suresh
3        NaN
4        NaN
Name: name, dtype: object 0    45.0
1     NaN
2     NaN
3    50.0
4     NaN
Name: age, dtype: float64
```

Reading multiple columns from DataFrame

In order to read multiple columns, the column names must be defined as a list. It return multiple columns as a dataframe.

When we single column it read as a series.

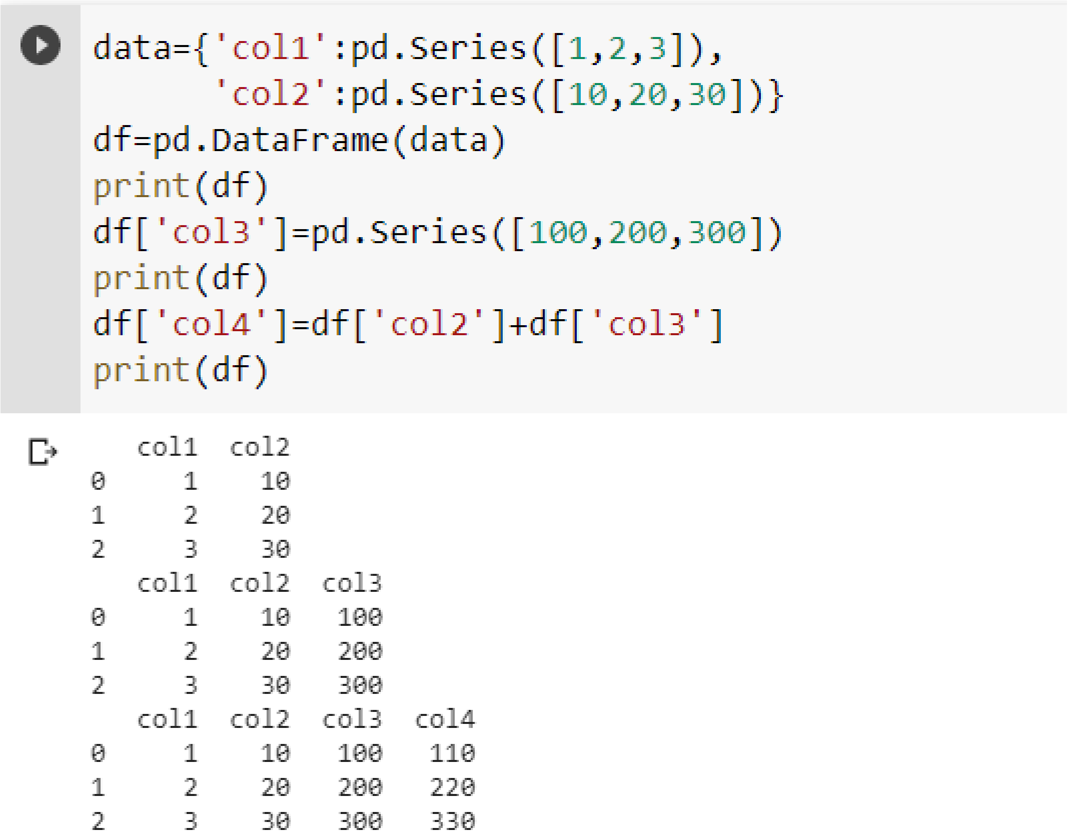


```
data={'a':[1,2,3,4,5],'b':[100,200,300,400,500],'c':[1000,2000,3000,4000,5000],'d':[10000,20000,30000,40000,5000]}
df=pd.DataFrame(data)
print(df)
print(df[['a','c']])
r=df[['a','c']]
print(r)
print(type(r))
```

```
   a    b     c      d
0  1  100  1000  10000
1  2  200  2000  20000
2  3  300  3000  30000
3  4  400  4000  40000
4  5  500  5000   5000
   a     c
0  1  1000
1  2  2000
2  3  3000
3  4  4000
4  5  5000
   a     c
0  1  1000
1  2  2000
2  3  3000
3  4  4000
4  5  5000
<class 'pandas.core.frame.DataFrame'>
```

Column Addition

Adding new column to the existing DataFrame.

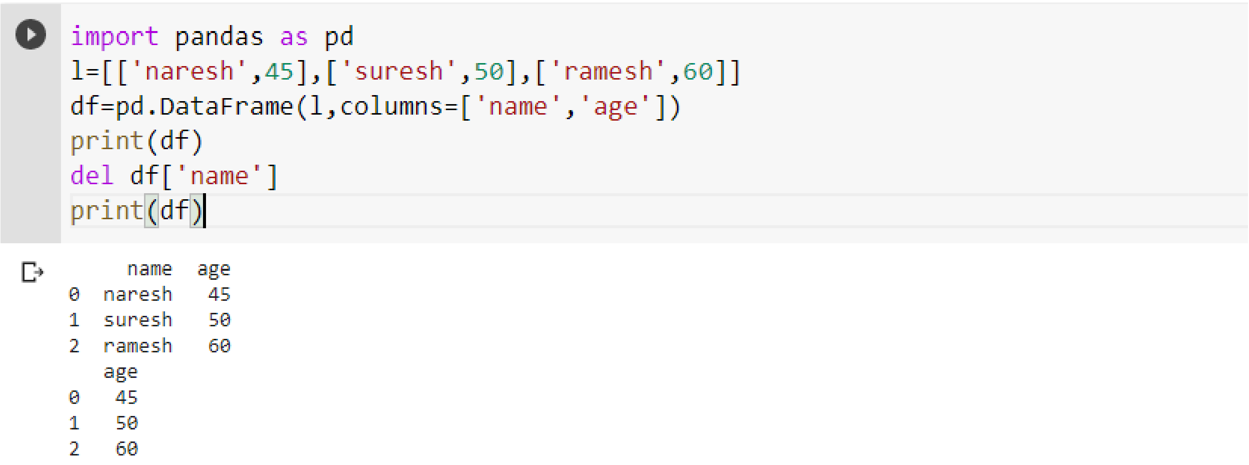


```
data={'col1':pd.Series([1,2,3]),
      'col2':pd.Series([10,20,30])}
df=pd.DataFrame(data)
print(df)
df['col3']=pd.Series([100,200,300])
print(df)
df['col4']=df['col2']+df['col3']
print(df)
```

```
   col1  col2
0     1    10
1     2    20
2     3    30
   col1  col2  col3
0     1    10   100
1     2    20   200
2     3    30   300
   col1  col2  col3  col4
0     1    10   100   110
1     2    20   200   220
2     3    30   300   330
```

Column Deletion

The column deletion is done using del keyword.
It allows deleting one or more than one columns.
The column is deleted with column name or column labels.



```
import pandas as pd
l=[['naresh',45],['suresh',50],['ramesh',60]]
df=pd.DataFrame(l,columns=['name','age'])
print(df)
del df['name']
print(df)
```

```
      name  age
0   naresh   45
1   suresh   50
2   ramesh   60
   age
0   45
1   50
2   60
```

Row Selection, Addition and Deletion

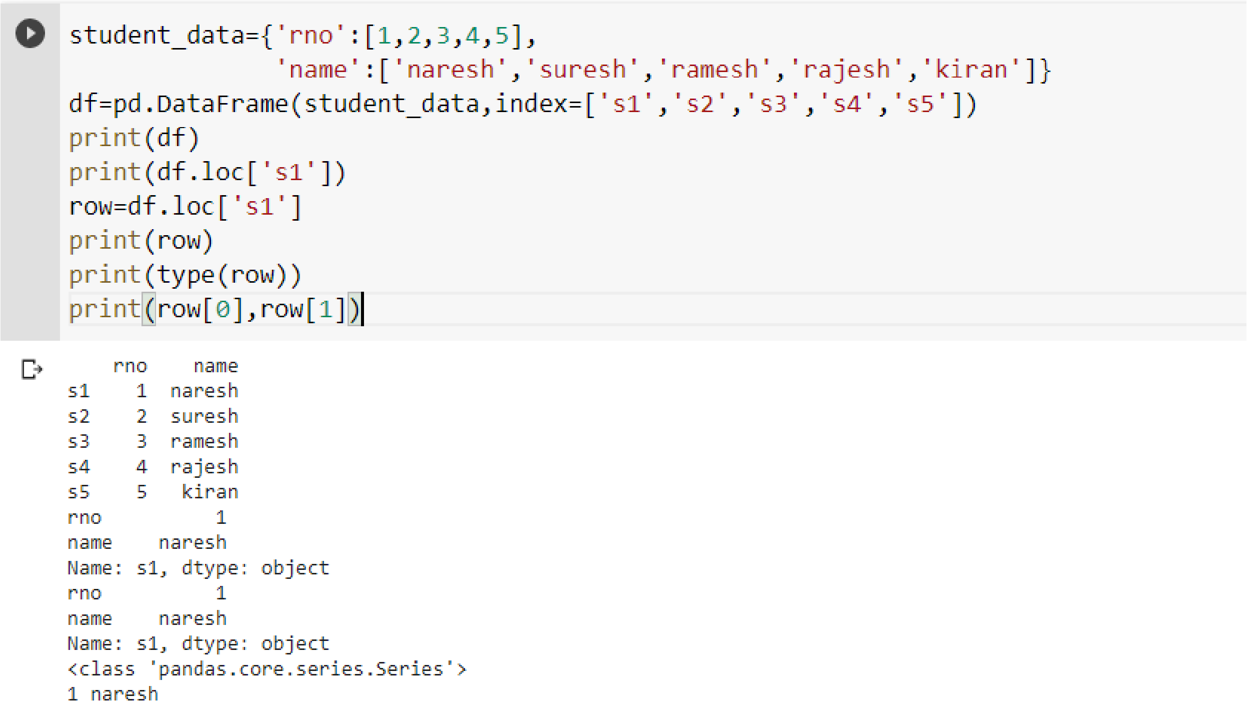
Each row is identified with index or label. We can read rows from dataframe using index or label.

DataFrame provide two methods to perform this operation.

1. loc
2. iloc

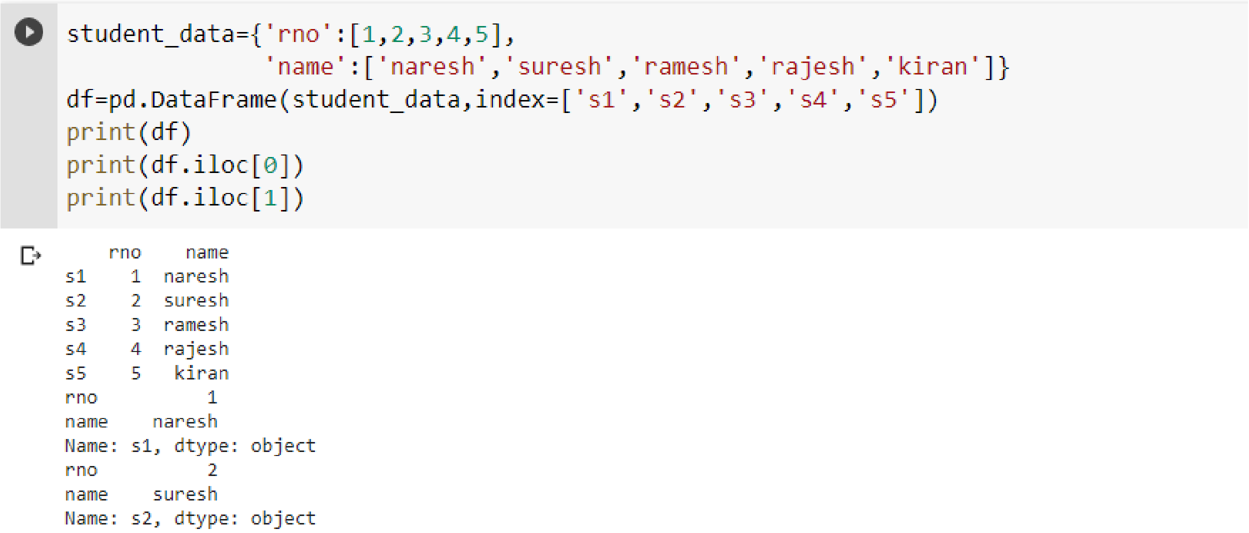
loc() is used to read the rows using label

iloc() is used to read the rows using index



```
student_data={'rno':[1,2,3,4,5],
              'name':['naresh','suresh','ramesh','rajesh','kiran']}
df=pd.DataFrame(student_data,index=['s1','s2','s3','s4','s5'])
print(df)
print(df.loc['s1'])
row=df.loc['s1']
print(row)
print(type(row))
print(row[0],row[1])
```

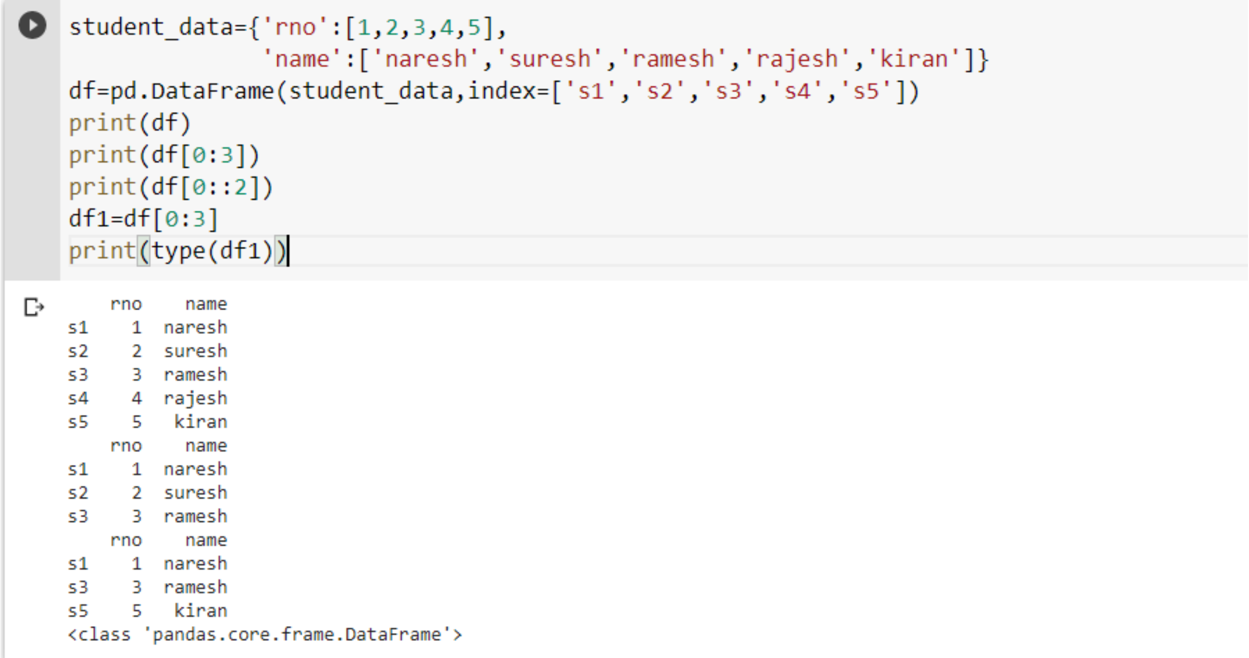
```
    rno    name
s1    1  naresh
s2    2  suresh
s3    3  ramesh
s4    4  rajesh
s5    5   kiran
rno          1
name    naresh
Name: s1, dtype: object
rno          1
name    naresh
Name: s1, dtype: object
<class 'pandas.core.series.Series'>
1 naresh
```



```
student_data={'rno':[1,2,3,4,5],
              'name':['naresh','suresh','ramesh','rajesh','kiran']}
df=pd.DataFrame(student_data,index=['s1','s2','s3','s4','s5'])
print(df)
print(df.iloc[0])
print(df.iloc[1])
```

```
    rno    name
s1    1  naresh
s2    2  suresh
s3    3  ramesh
s4    4  rajesh
s5    5   kiran
rno          1
name    naresh
Name: s1, dtype: object
rno          2
name    suresh
Name: s2, dtype: object
```

Slicing is used to read more than one row

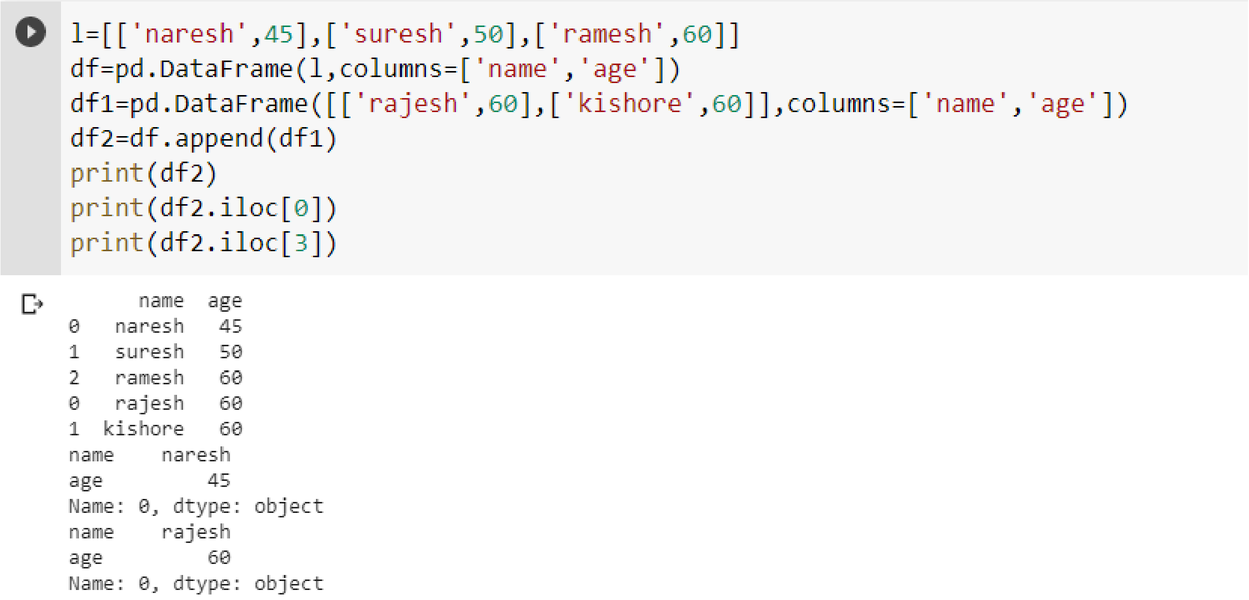


```
student_data={'rno':[1,2,3,4,5],
              'name':['naresh','suresh','ramesh','rajesh','kiran']}
df=pd.DataFrame(student_data,index=['s1','s2','s3','s4','s5'])
print(df)
print(df[0:3])
print(df[0::2])
df1=df[0:3]
print(type(df1))
```

```
    rno    name
s1    1  naresh
s2    2  suresh
s3    3  ramesh
s4    4  rajesh
s5    5   kiran
    rno    name
s1    1  naresh
s2    2  suresh
s3    3  ramesh
    rno    name
s1    1  naresh
s3    3  ramesh
s5    5   kiran
<class 'pandas.core.frame.DataFrame'>
```

Append Row

After creating data frame we can add a new row using append method.
This method will add row at the end of dataframe.
dataframe.append(row)
Row is represented as a dataframe.

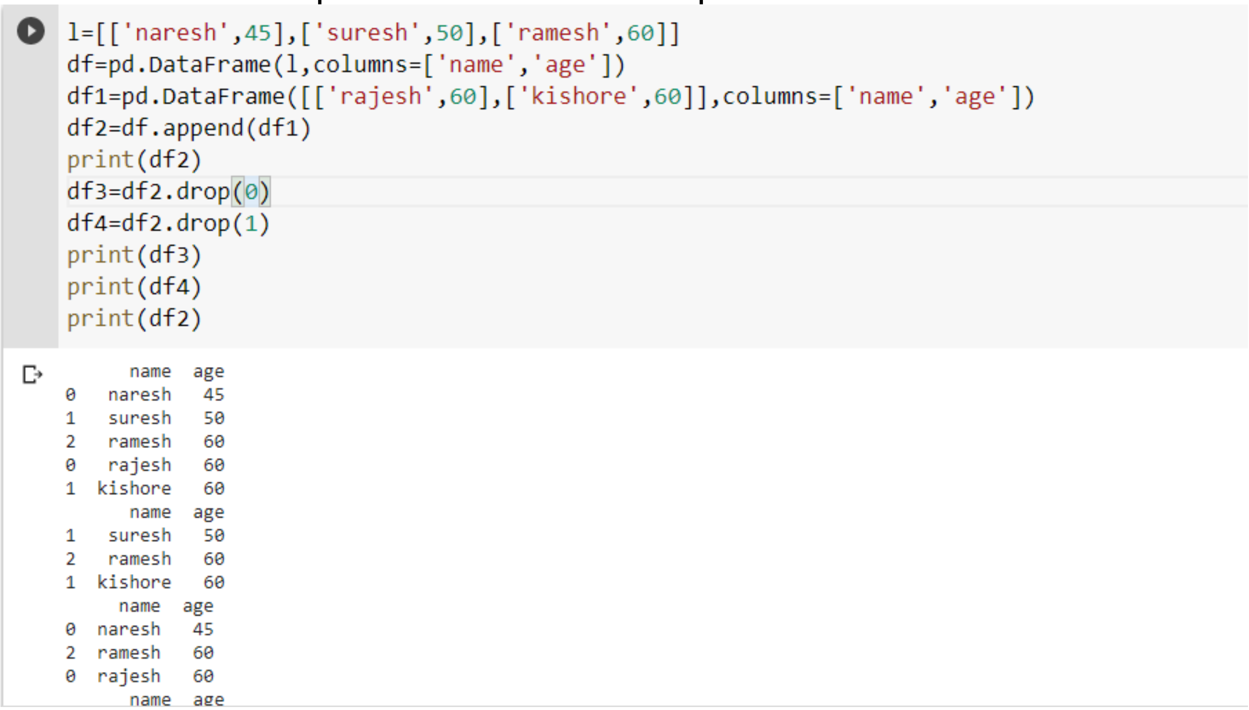


```
l=[['naresh',45],['suresh',50],['ramesh',60]]
df=pd.DataFrame(l,columns=['name','age'])
df1=pd.DataFrame([['rajesh',60],['kishore',60]],columns=['name','age'])
df2=df.append(df1)
print(df2)
print(df2.iloc[0])
print(df2.iloc[3])
```

```
      name  age
0   naresh   45
1   suresh   50
2   ramesh   60
0   rajesh   60
1  kishore   60
name    naresh
age         45
Name: 0, dtype: object
name    rajesh
age         60
Name: 0, dtype: object
```

Deletion of rows

Deletion of rows are done using a method drop().
It delete only one row.
Deleting is done using row labels/index.
It row labels are duplicated it remove multiple rows.



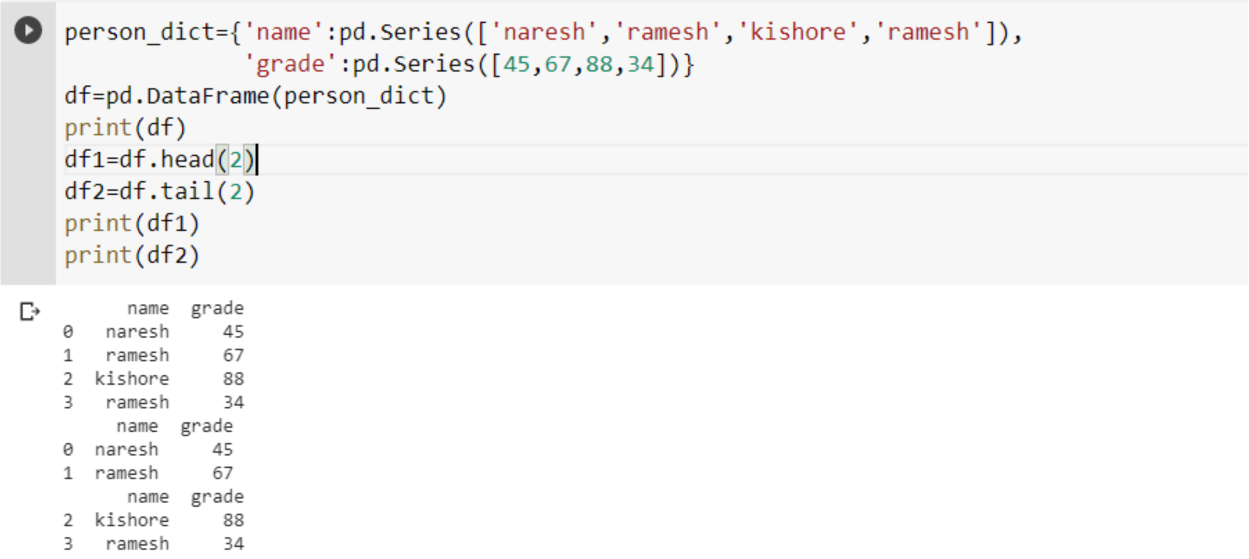
```
l=[['naresh',45],['suresh',50],['ramesh',60]]
df=pd.DataFrame(l,columns=['name','age'])
df1=pd.DataFrame([['rajesh',60],['kishore',60]],columns=['name','age'])
df2=df.append(df1)
print(df2)
df3=df2.drop(0)
df4=df2.drop(1)
print(df3)
print(df4)
print(df2)
```

```
      name  age
0   naresh   45
1   suresh   50
2   ramesh   60
0   rajesh   60
1  kishore   60
      name  age
1   suresh   50
2   ramesh   60
1  kishore   60
     name  age
0  naresh   45
2  ramesh   60
0  rajesh   60
      name  age
```

head and tail methods of DataFrame

head and tail are the methods of DataFrame object.

head() returns first n number of rows
tail() returns last n number of rows

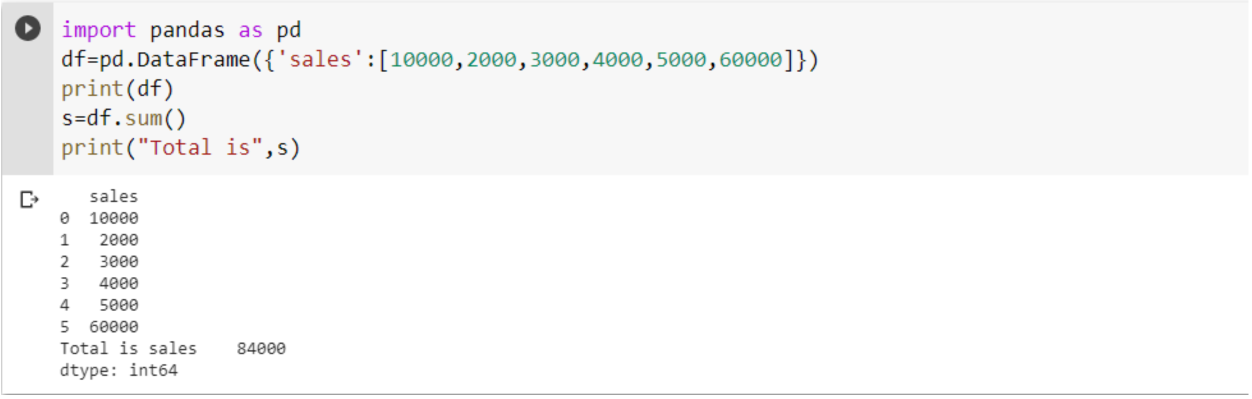


```
person_dict={'name':pd.Series(['naresh','ramesh','kishore','ramesh']),
             'grade':pd.Series([45,67,88,34])}
df=pd.DataFrame(person_dict)
print(df)
df1=df.head(2)
df2=df.tail(2)
print(df1)
print(df2)
```

```
      name  grade
0   naresh     45
1   ramesh     67
2  kishore     88
3   ramesh     34
     name  grade
0  naresh     45
1  ramesh     67
      name  grade
2  kishore     88
3   ramesh     34
```

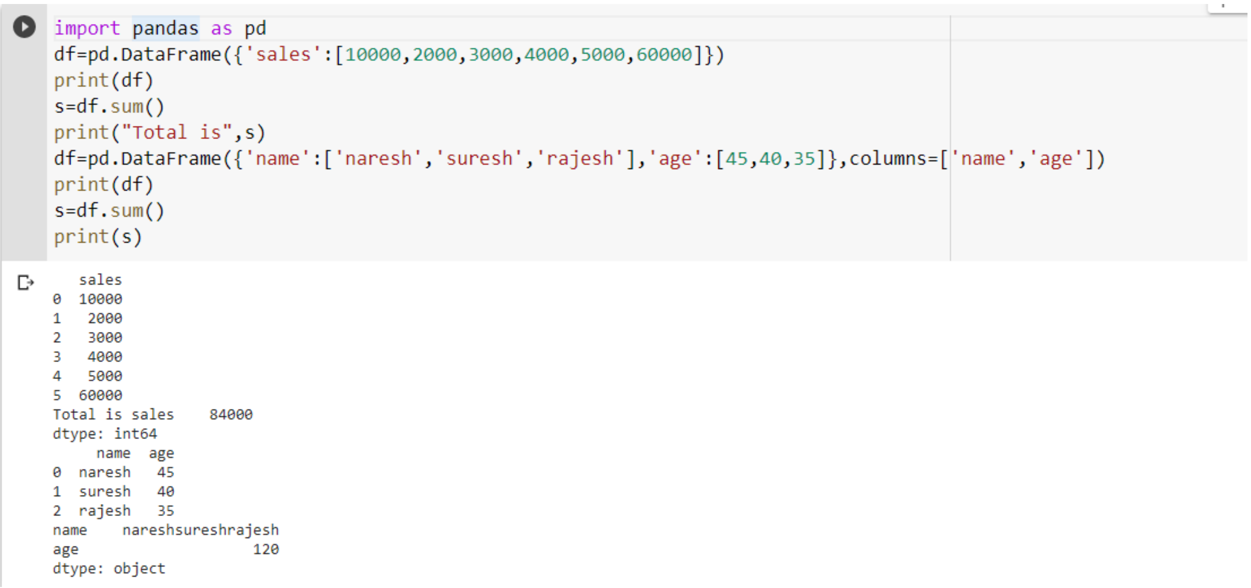
Other Operations of DataFrame

sum(): This function return sum



```
import pandas as pd
df=pd.DataFrame({'sales':[10000,2000,3000,4000,5000,60000]})
print(df)
s=df.sum()
print("Total is",s)
```

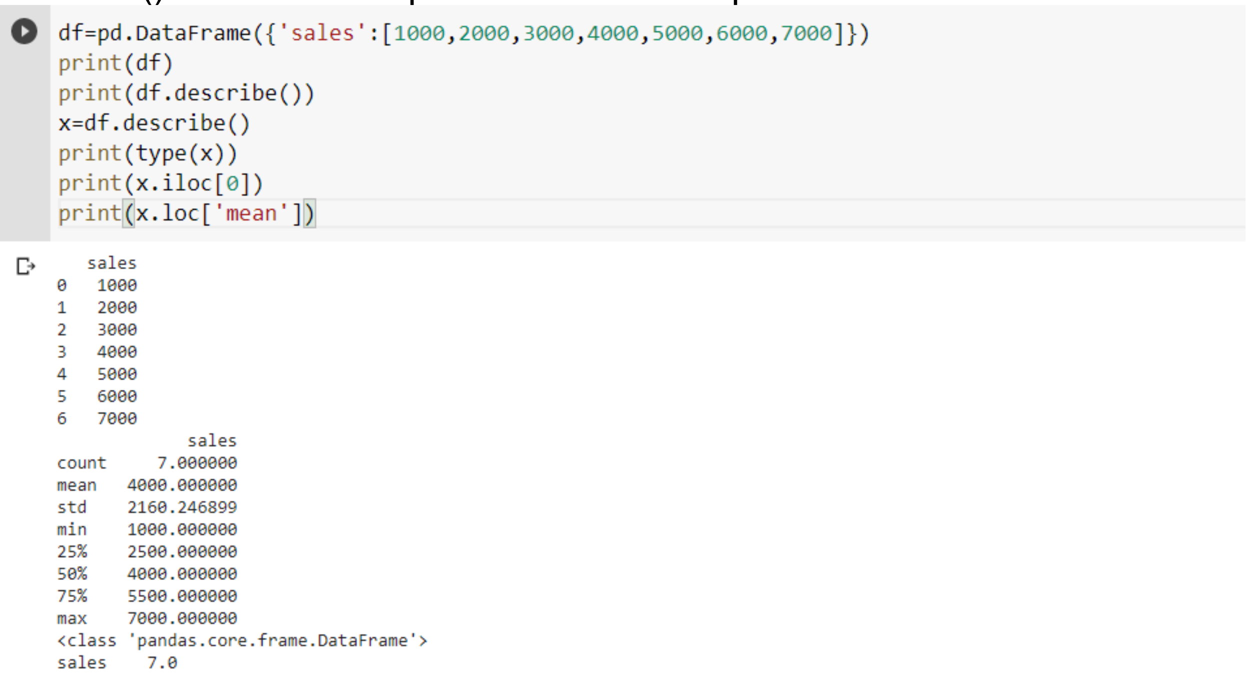
```
   sales
0  10000
1   2000
2   3000
3   4000
4   5000
5  60000
Total is sales    84000
dtype: int64
```



```
import pandas as pd
df=pd.DataFrame({'sales':[10000,2000,3000,4000,5000,60000]})
print(df)
s=df.sum()
print("Total is",s)
df=pd.DataFrame({'name':['naresh','suresh','rajesh'],'age':[45,40,35]},columns=['name','age'])
print(df)
s=df.sum()
print(s)
```

```
   sales
0  10000
1   2000
2   3000
3   4000
4   5000
5  60000
Total is sales    84000
dtype: int64
     name  age
0  naresh   45
1  suresh   40
2  rajesh   35
name    nareshsureshrajesh
age                    120
dtype: object
```

describe(): This function perform statistical operations on dataframe.



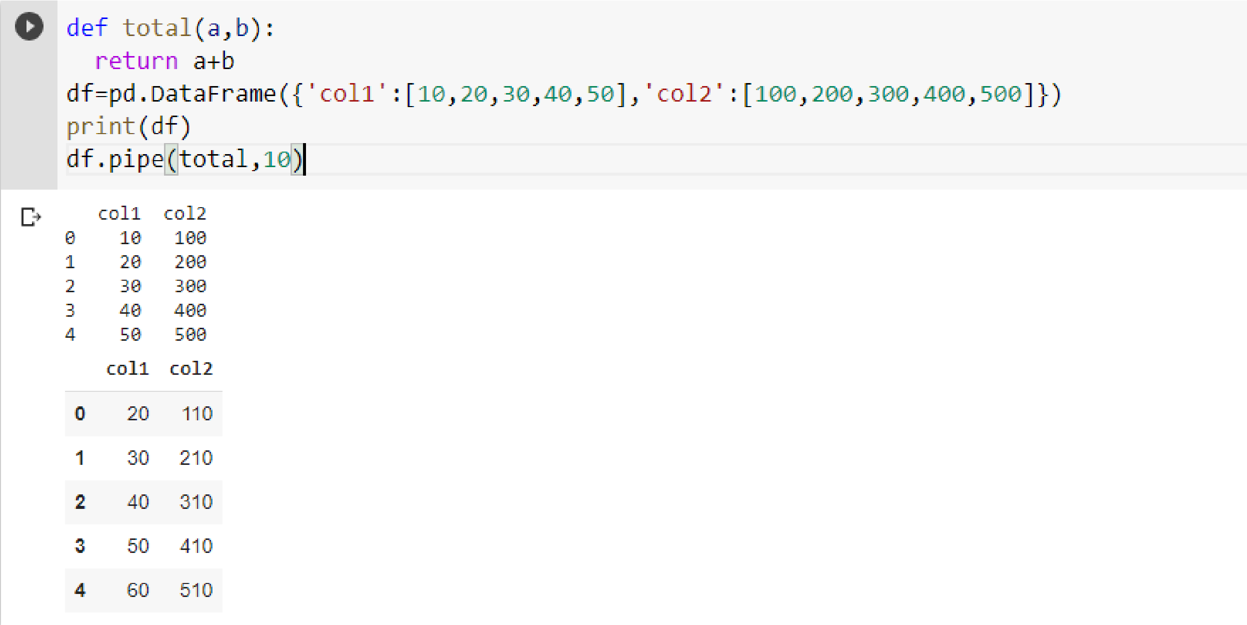
```
df=pd.DataFrame({'sales':[1000,2000,3000,4000,5000,6000,7000]})
print(df)
print(df.describe())
x=df.describe()
print(type(x))
print(x.iloc[0])
print(x.loc['mean'])
```

```
   sales
0   1000
1   2000
2   3000
3   4000
4   5000
5   6000
6   7000
             sales
count     7.000000
mean   4000.000000
std    2160.246899
min    1000.000000
25%    2500.000000
50%    4000.000000
75%    5500.000000
max    7000.000000
<class 'pandas.core.frame.DataFrame'>
sales    7.0
```

Pandas: Function Application

We can apply customized functions from library or userdefined.
This functions are applied based on the application requirement on rows, columns or element wise.

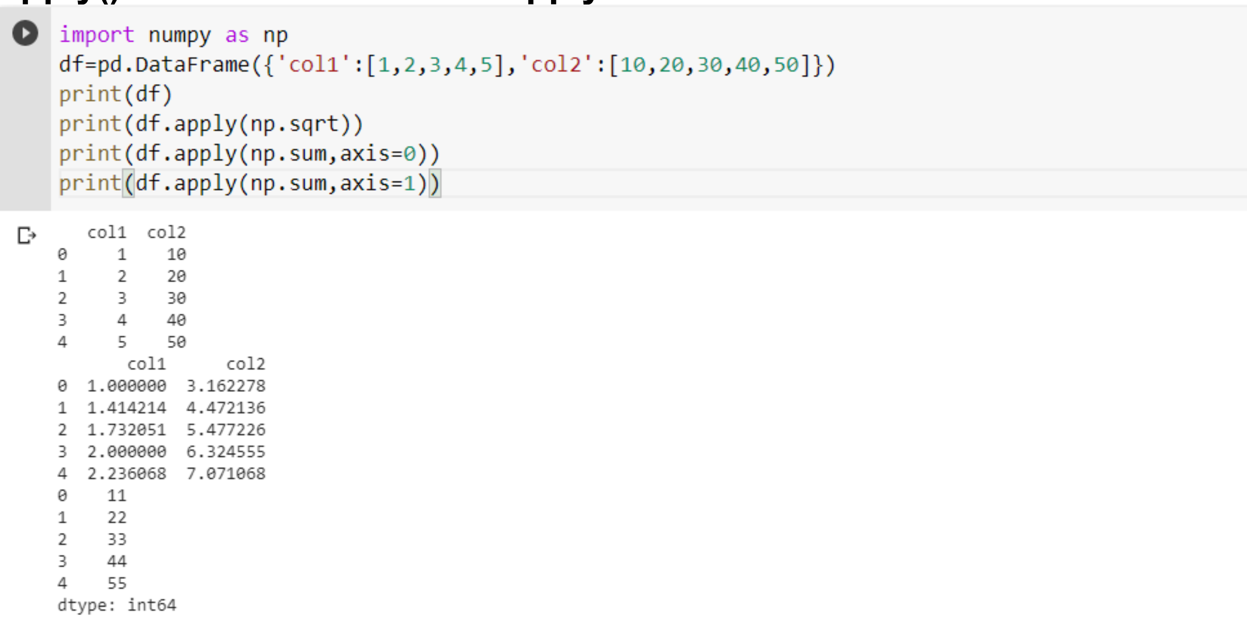
pipe() : table based
apply() : row based or column based
applymap(): element based



```
def total(a,b):
  return a+b
df=pd.DataFrame({'col1':[10,20,30,40,50],'col2':[100,200,300,400,500]})
print(df)
df.pipe(total,10)
```

```
   col1  col2
0    10   100
1    20   200
2    30   300
3    40   400
4    50   500
   col1  col2
0    20   110
1    30   210
2    40   310
3    50   410
4    60   510
```

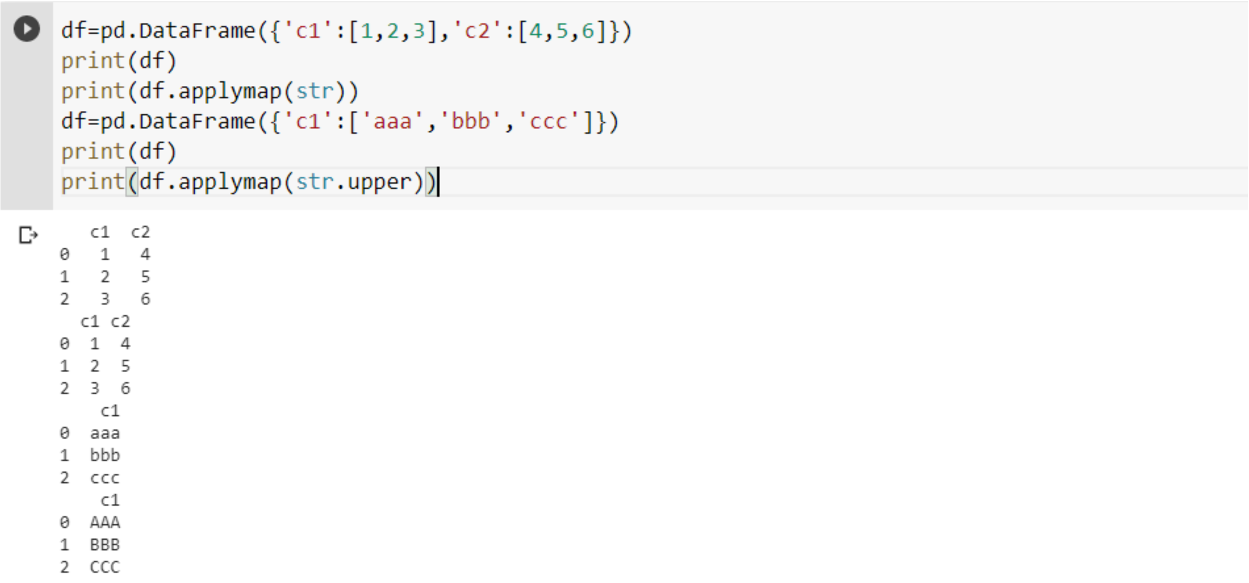
apply() this function is used apply a function to rows or columns.



```
import numpy as np
df=pd.DataFrame({'col1':[1,2,3,4,5],'col2':[10,20,30,40,50]})
print(df)
print(df.apply(np.sqrt))
print(df.apply(np.sum,axis=0))
print(df.apply(np.sum,axis=1))
```

```
   col1  col2
0     1    10
1     2    20
2     3    30
3     4    40
4     5    50
       col1      col2
0  1.000000  3.162278
1  1.414214  4.472136
2  1.732051  5.477226
3  2.000000  6.324555
4  2.236068  7.071068
0    11
1    22
2    33
3    44
4    55
dtype: int64
```

applymap() : This function is used to apply a function to individual element in dataframe.



```
df=pd.DataFrame({'c1':[1,2,3],'c2':[4,5,6]})
print(df)
print(df.applymap(str))
df=pd.DataFrame({'c1':['aaa','bbb','ccc']})
print(df)
print(df.applymap(str.upper))
```

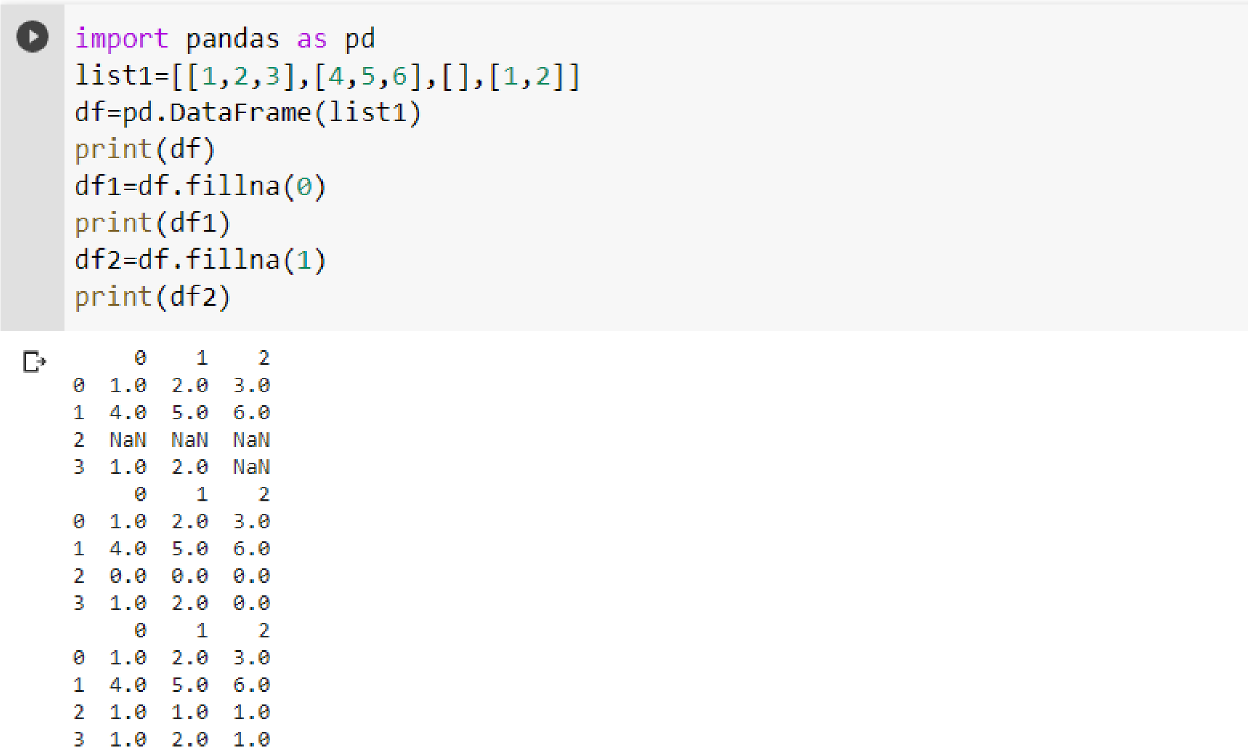
```
   c1  c2
0   1   4
1   2   5
2   3   6
  c1 c2
0  1  4
1  2  5
2  3  6
    c1
0  aaa
1  bbb
2  ccc
    c1
0  AAA
1  BBB
2  CCC
```

Pandas : Missing Data

Pandas library provide different functions for cleaning missing values or data.

NaN which is defined as missing value.

The fillna() function fill object with NaN values

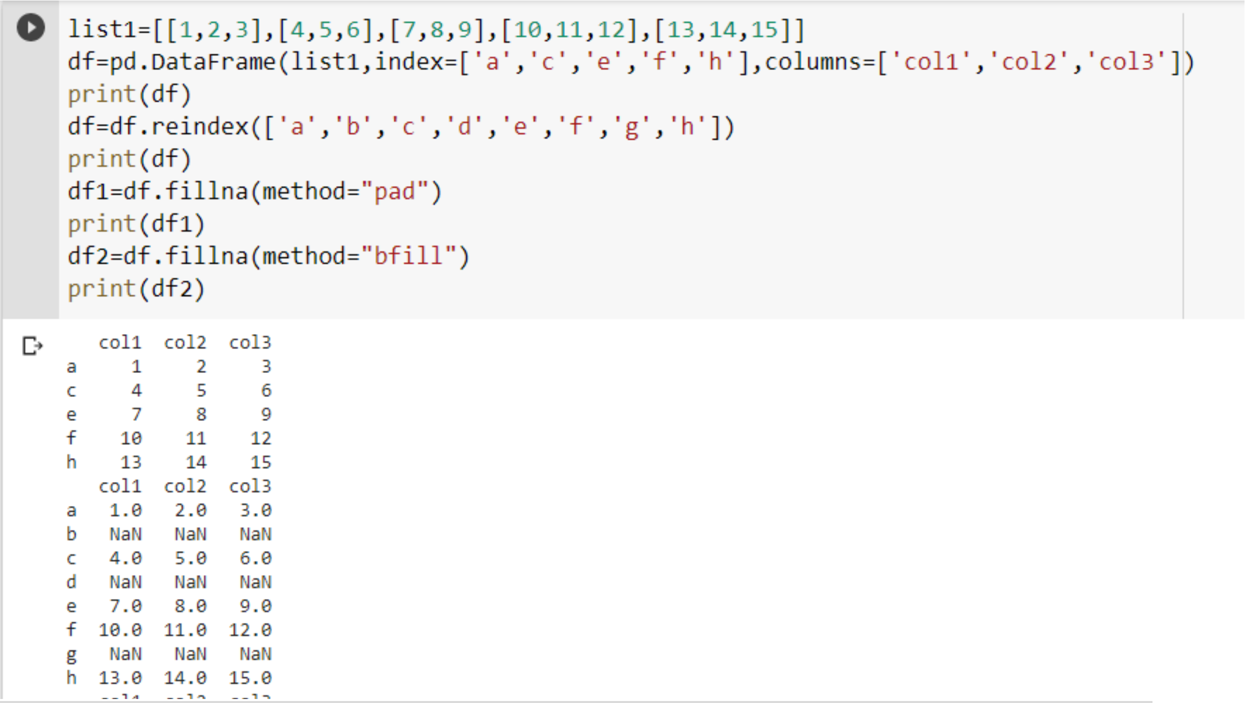


```
import pandas as pd
list1=[[1,2,3],[4,5,6],[],[1,2]]
df=pd.DataFrame(list1)
print(df)
df1=df.fillna(0)
print(df1)
df2=df.fillna(1)
print(df2)
```

```
     0    1    2
0  1.0  2.0  3.0
1  4.0  5.0  6.0
2  NaN  NaN  NaN
3  1.0  2.0  NaN
     0    1    2
0  1.0  2.0  3.0
1  4.0  5.0  6.0
2  0.0  0.0  0.0
3  1.0  2.0  0.0
     0    1    2
0  1.0  2.0  3.0
1  4.0  5.0  6.0
2  1.0  1.0  1.0
3  1.0  2.0  1.0
```

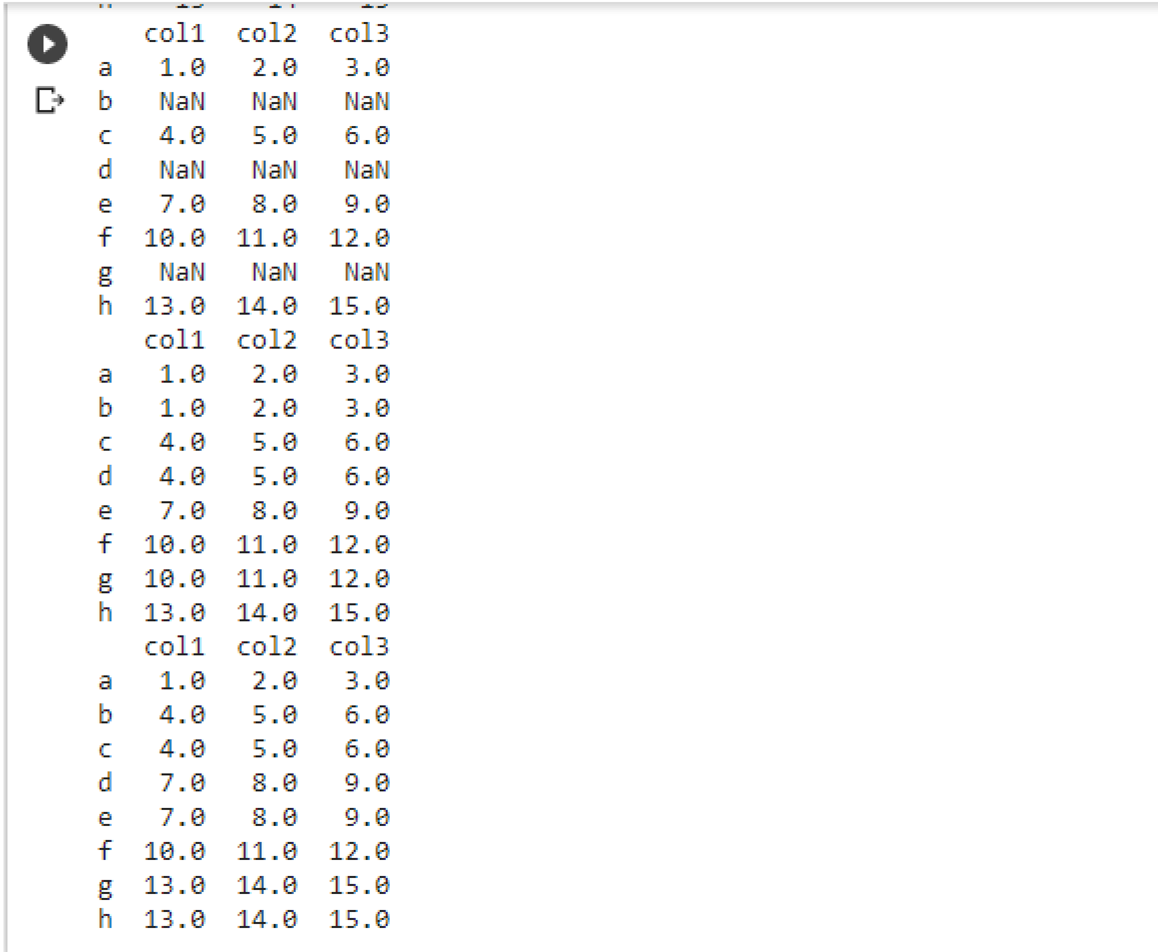
Fill NaN forward and backward

We can fill the values in the different dirctions over the object.

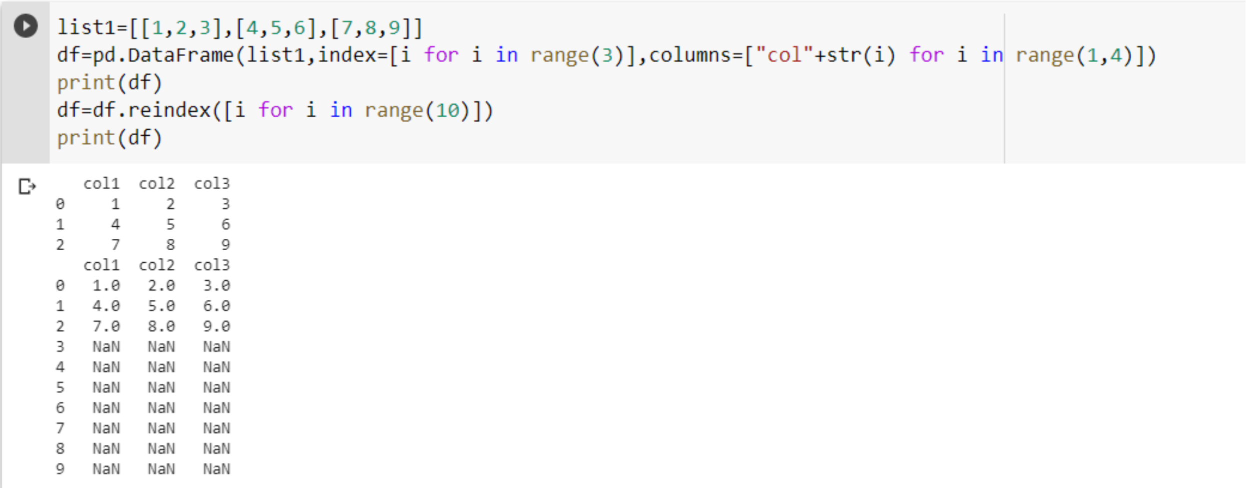


```
list1=[[1,2,3],[4,5,6],[7,8,9],[10,11,12],[13,14,15]]
df=pd.DataFrame(list1,index=['a','c','e','f','h'],columns=['col1','col2','col3'])
print(df)
df=df.reindex(['a','b','c','d','e','f','g','h'])
print(df)
df1=df.fillna(method="pad")
print(df1)
df2=df.fillna(method="bfill")
print(df2)
```

```
   col1  col2  col3
a     1     2     3
c     4     5     6
e     7     8     9
f    10    11    12
h    13    14    15
   col1  col2  col3
a   1.0   2.0   3.0
b   NaN   NaN   NaN
c   4.0   5.0   6.0
d   NaN   NaN   NaN
e   7.0   8.0   9.0
f  10.0  11.0  12.0
g   NaN   NaN   NaN
h  13.0  14.0  15.0
```



```
   col1  col2  col3
a   1.0   2.0   3.0
b   NaN   NaN   NaN
c   4.0   5.0   6.0
d   NaN   NaN   NaN
e   7.0   8.0   9.0
f  10.0  11.0  12.0
g   NaN   NaN   NaN
h  13.0  14.0  15.0
   col1  col2  col3
a   1.0   2.0   3.0
b   1.0   2.0   3.0
c   4.0   5.0   6.0
d   4.0   5.0   6.0
e   7.0   8.0   9.0
f  10.0  11.0  12.0
g  10.0  11.0  12.0
h  13.0  14.0  15.0
   col1  col2  col3
a   1.0   2.0   3.0
b   4.0   5.0   6.0
c   4.0   5.0   6.0
d   7.0   8.0   9.0
e   7.0   8.0   9.0
f  10.0  11.0  12.0
g  13.0  14.0  15.0
h  13.0  14.0  15.0
```

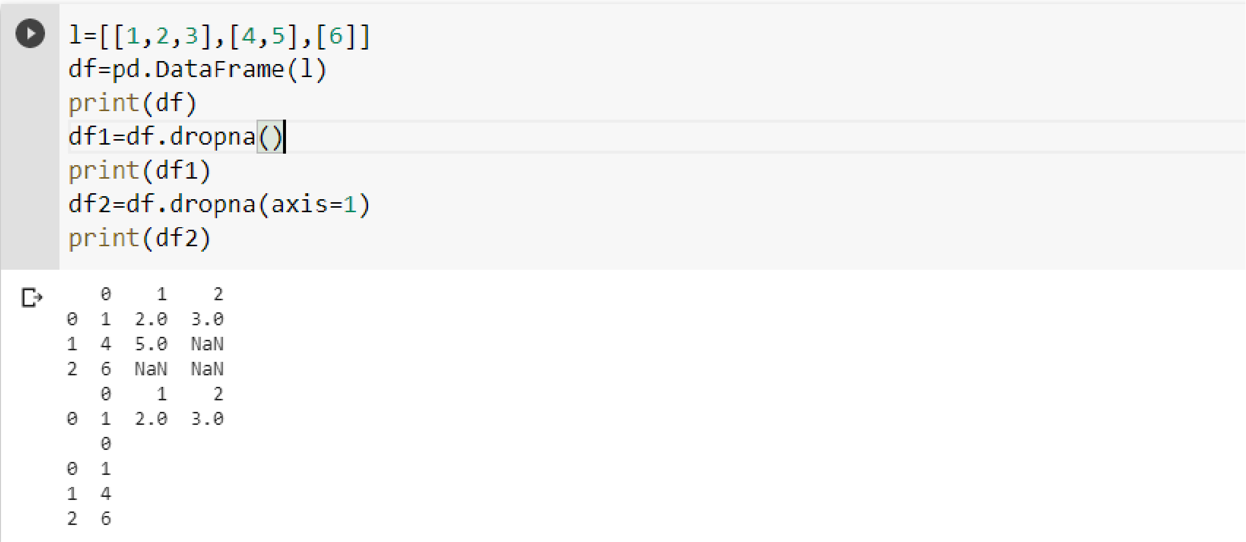


```
list1=[[1,2,3],[4,5,6],[7,8,9]]
df=pd.DataFrame(list1,index=[i for i in range(3)],columns=["col"+str(i) for i in range(1,4)])
print(df)
df=df.reindex([i for i in range(10)])
print(df)
```

```
   col1  col2  col3
0     1     2     3
1     4     5     6
2     7     8     9
   col1  col2  col3
0   1.0   2.0   3.0
1   4.0   5.0   6.0
2   7.0   8.0   9.0
3   NaN   NaN   NaN
4   NaN   NaN   NaN
5   NaN   NaN   NaN
6   NaN   NaN   NaN
7   NaN   NaN   NaN
8   NaN   NaN   NaN
9   NaN   NaN   NaN
```

Drop Missing Values

We can drop or exclude missing elements/values using a predefined method called dropna().
If any row value is NaN the complete row is excluded.

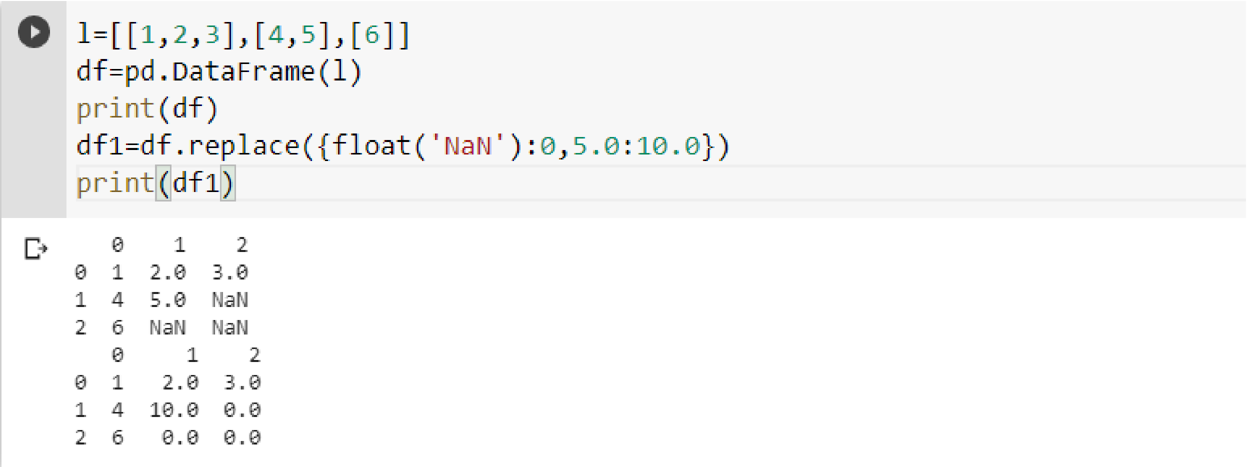


```
l=[[1,2,3],[4,5],[6]]
df=pd.DataFrame(l)
print(df)
df1=df.dropna()
print(df1)
df2=df.dropna(axis=1)
print(df2)
```

```
   0    1    2
0  1  2.0  3.0
1  4  5.0  NaN
2  6  NaN  NaN
   0    1    2
0  1  2.0  3.0
   0
0  1
1  4
2  6
```

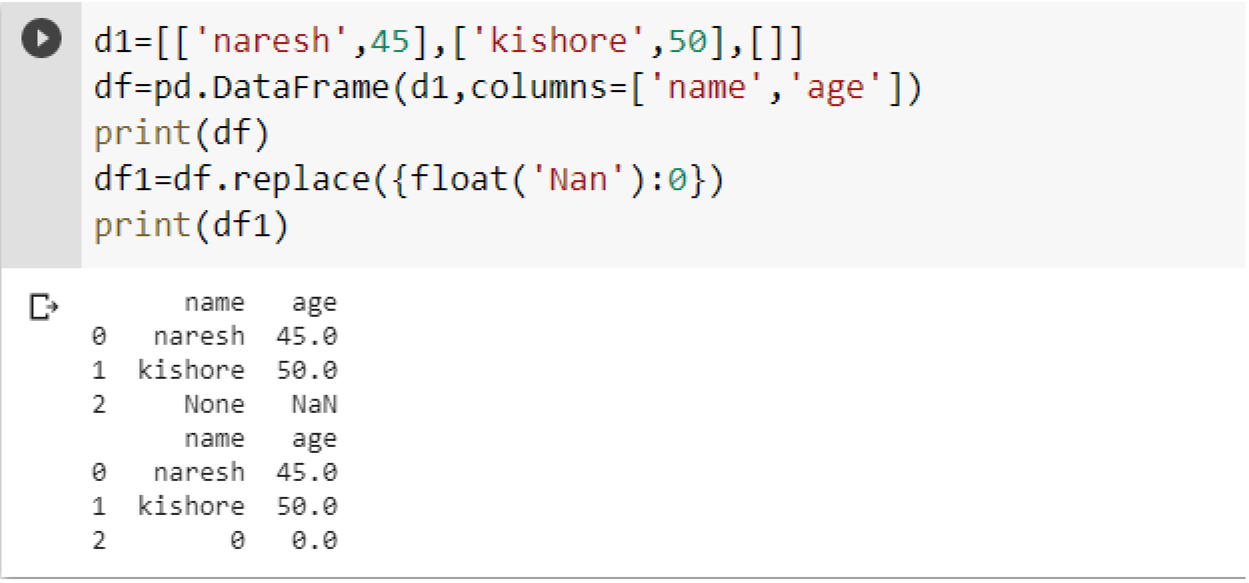
Replacing Missing or Generic values

Dataframe provide replace method, this method is used to replace missing value or generic value with any other value or specific value.



```
l=[[1,2,3],[4,5],[6]]
df=pd.DataFrame(l)
print(df)
df1=df.replace({float('NaN'):0,5.0:10.0})
print(df1)
```

```
   0    1    2
0  1  2.0  3.0
1  4  5.0  NaN
2  6  NaN  NaN
   0     1    2
0  1   2.0  3.0
1  4  10.0  0.0
2  6   0.0  0.0
```



```
d1=[['naresh',45],['kishore',50],[]]
df=pd.DataFrame(d1,columns=['name','age'])
print(df)
df1=df.replace({float('Nan'):0})
print(df1)
```

```
      name   age
0   naresh  45.0
1  kishore  50.0
2     None   NaN
      name   age
0   naresh  45.0
1  kishore  50.0
2        0   0.0
```

Pandas Groupy
Group by is one of the important concept in processing data in data science.
We can create group of categories and apply functions to that categories.
Group by involving the following steps.

1. Splitting
2. Applying
3. Combining

Pandas datasets/dataframe can be split on any axis. There are multiple ways or methods are used to split data.

1. DataFrame.groupby(key)
2. DataFrame.groupby(key,axis=1)

3. DataFrame.groupby(key1,key2)

Grouping date using one or more than one key

In order to group data using one key, we pass one key argument to groupby method. Key is field name or column name. in order to group multiple columns, we need provide more than one column name in groupby.

	empno	ename	job	deptno	salary
0	1	aaa	manager	10	1000
1	2	bbb	clerk	20	2000
2	3	ccc	manager	10	3000
3	4	ddd	hr	10	4000
4	5	eee	clerk	20	5000
5	6	fff	manager	20	6000
6	7	ggg	manager	30	1000
7	8	hhh	clerk	10	3000
8	9	iii	hr	20	4000
9	10	jjjj	hr	30	8000

deptno :10

0	1	aaa	manager	10	1000
2	3	ccc	manager	10	3000
3	4	ddd	hr	10	4000
7	8	hhh	clerk	10	3000

job : manager

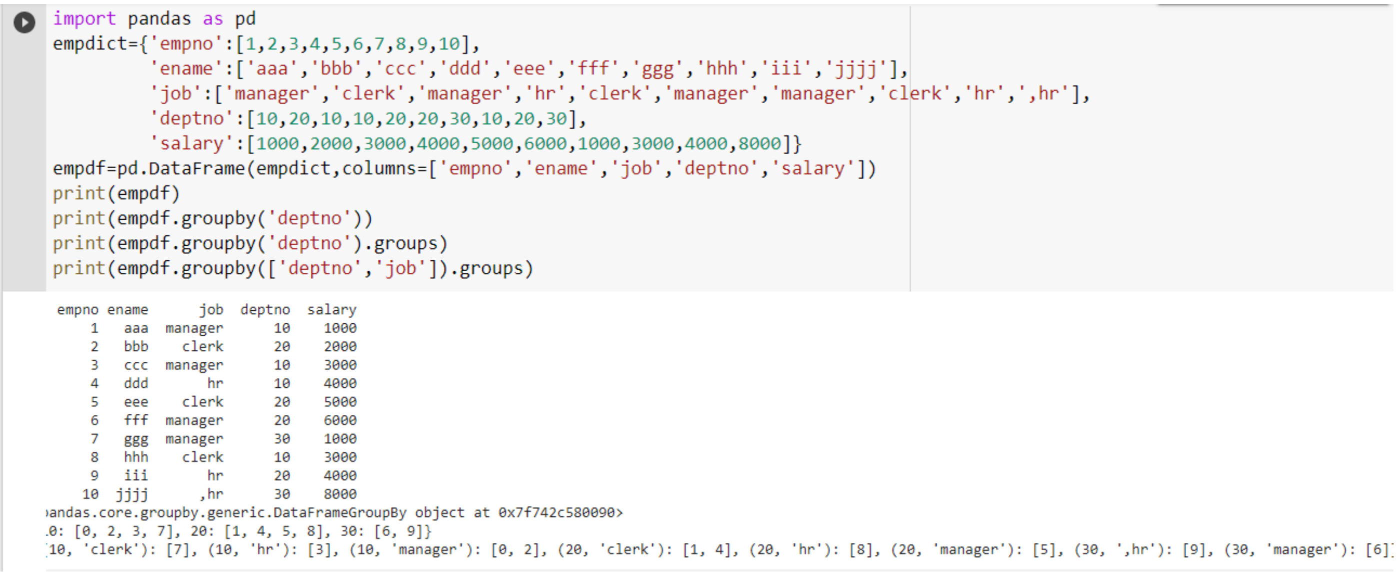
0	1	aaa	manager	10	1000
2	3	ccc	manager	10	3000

job: clerk

7	8	hhh	clerk	10	3000

job: hr

3	4	ddd	hr	10	4000

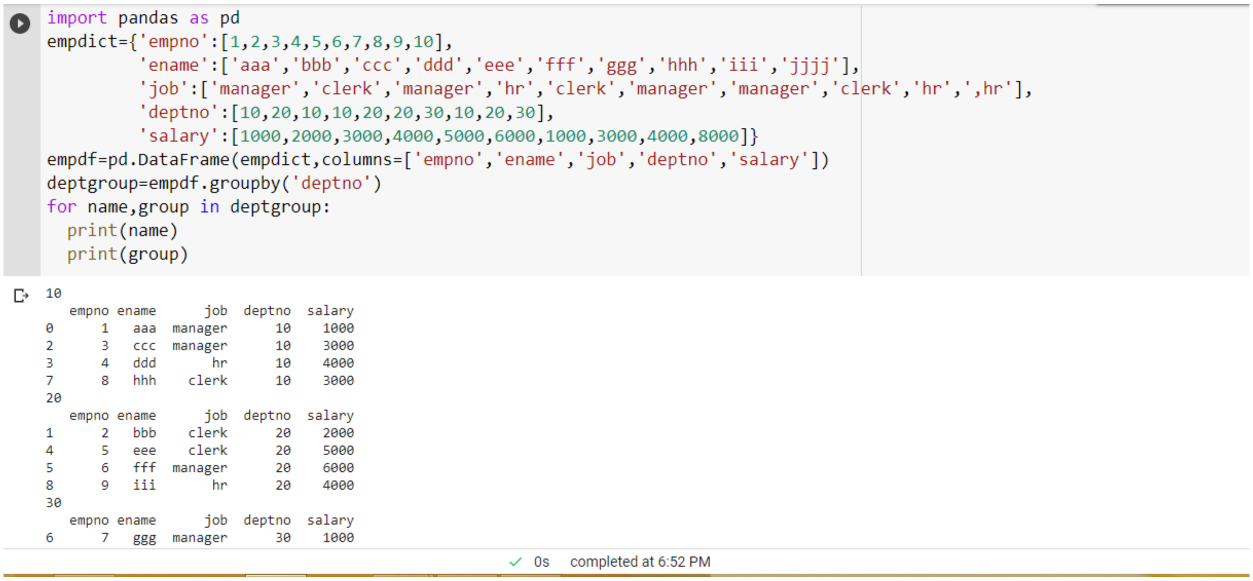


```
import pandas as pd
empdict={'empno':[1,2,3,4,5,6,7,8,9,10],
         'ename':['aaa','bbb','ccc','ddd','eee','fff','ggg','hhh','iii','jjjj'],
         'job':['manager','clerk','manager','hr','clerk','manager','manager','clerk','hr',',hr'],
         'deptno':[10,20,10,10,20,20,30,10,20,30],
         'salary':[1000,2000,3000,4000,5000,6000,1000,3000,4000,8000]}
empdf=pd.DataFrame(empdict,columns=['empno','ename','job','deptno','salary'])
print(empdf)
print(empdf.groupby('deptno'))
print(empdf.groupby('deptno').groups)
print(empdf.groupby(['deptno','job']).groups)
```

```
 empno ename      job  deptno  salary
     1   aaa  manager      10    1000
     2   bbb    clerk      20    2000
     3   ccc  manager      10    3000
     4   ddd       hr      10    4000
     5   eee    clerk      20    5000
     6   fff  manager      20    6000
     7   ggg  manager      30    1000
     8   hhh    clerk      10    3000
     9   iii       hr      20    4000
    10  jjjj      ,hr      30    8000
andas.core.groupby.generic.DataFrameGroupBy object at 0x7f742c580090>
.0: [0, 2, 3, 7], 20: [1, 4, 5, 8], 30: [6, 9]}
10, 'clerk'): [7], (10, 'hr'): [3], (10, 'manager'): [0, 2], (20, 'clerk'): [1, 4], (20, 'hr'): [8], (20, 'manager'): [5], (30, ',hr'): [9], (30, 'manager'): [6]
```

Iterating through groups

We can read the content of group by object. This can be done using for loop.

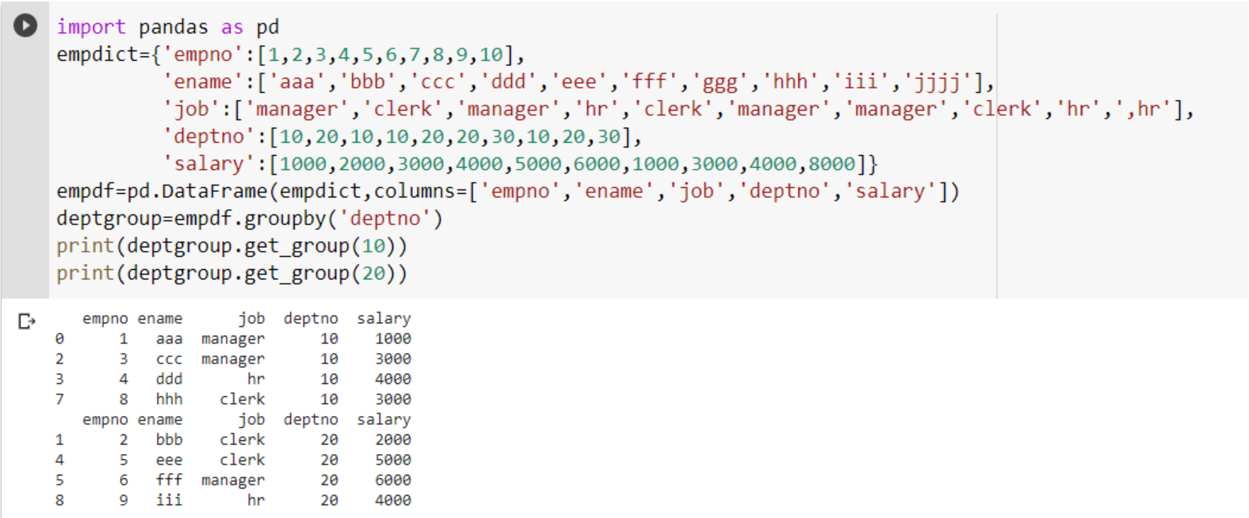


```
import pandas as pd
empdict={'empno':[1,2,3,4,5,6,7,8,9,10],
         'ename':['aaa','bbb','ccc','ddd','eee','fff','ggg','hhh','iii','jjjj'],
         'job':['manager','clerk','manager','hr','clerk','manager','manager','clerk','hr',',hr'],
         'deptno':[10,20,10,10,20,20,30,10,20,30],
         'salary':[1000,2000,3000,4000,5000,6000,1000,3000,4000,8000]}
empdf=pd.DataFrame(empdict,columns=['empno','ename','job','deptno','salary'])
deptgroup=empdf.groupby('deptno')
for name,group in deptgroup:
  print(name)
  print(group)
```

```
10
   empno ename      job  deptno  salary
0      1   aaa  manager      10    1000
2      3   ccc  manager      10    3000
3      4   ddd       hr      10    4000
7      8   hhh    clerk      10    3000
20
   empno ename      job  deptno  salary
1      2   bbb    clerk      20    2000
4      5   eee    clerk      20    5000
5      6   fff  manager      20    6000
8      9   iii       hr      20    4000
30
   empno ename      job  deptno  salary
6      7   ggg  manager      30    1000
```

✓ 0s completed at 6:52 PM

get_group(name) : This method is used to get the selected group from grouped objects (OR) select single group.

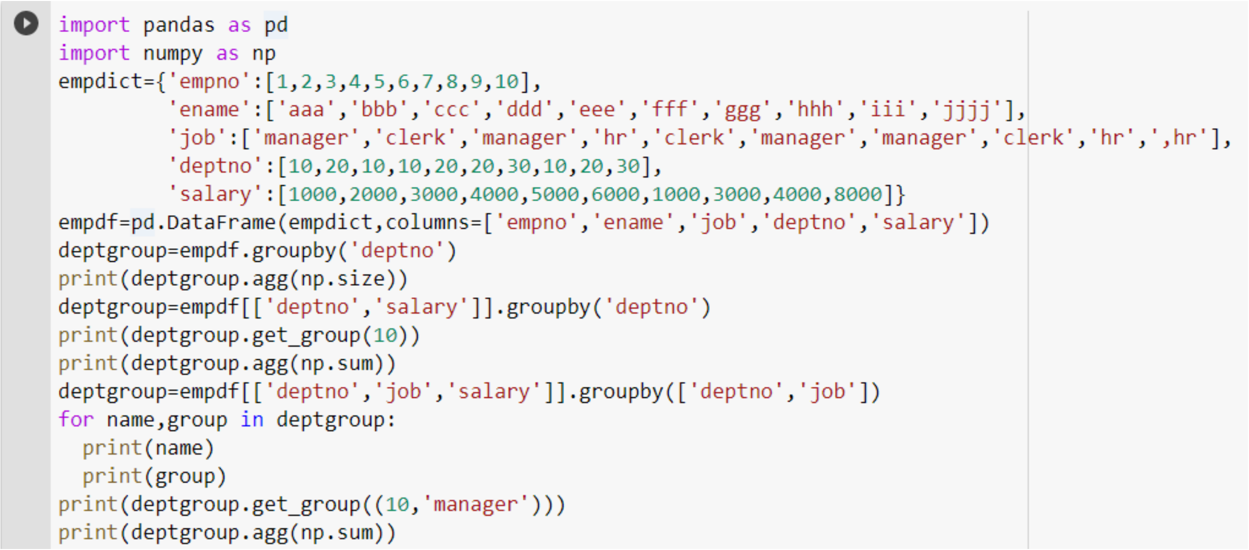


```
import pandas as pd
empdict={'empno':[1,2,3,4,5,6,7,8,9,10],
         'ename':['aaa','bbb','ccc','ddd','eee','fff','ggg','hhh','iii','jjjj'],
         'job':['manager','clerk','manager','hr','clerk','manager','manager','clerk','hr',',hr'],
         'deptno':[10,20,10,10,20,20,30,10,20,30],
         'salary':[1000,2000,3000,4000,5000,6000,1000,3000,4000,8000]}
empdf=pd.DataFrame(empdict,columns=['empno','ename','job','deptno','salary'])
deptgroup=empdf.groupby('deptno')
print(deptgroup.get_group(10))
print(deptgroup.get_group(20))
```

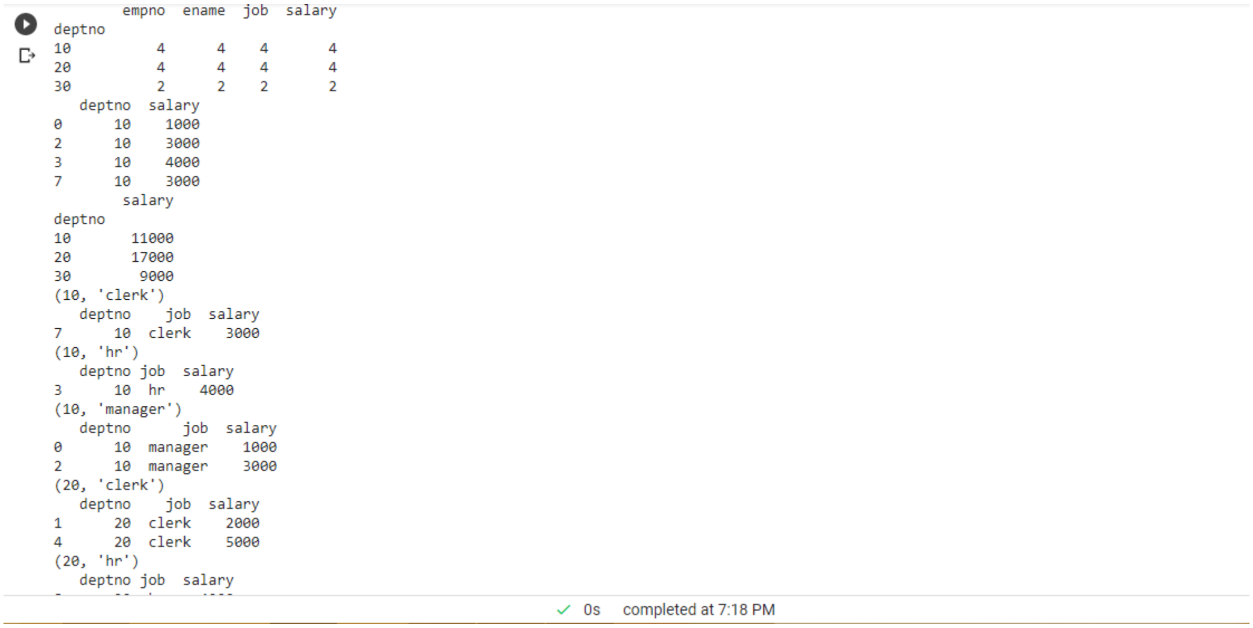
```
   empno ename      job  deptno  salary
0      1   aaa  manager      10    1000
2      3   ccc  manager      10    3000
3      4   ddd       hr      10    4000
7      8   hhh    clerk      10    3000
   empno ename      job  deptno  salary
1      2   bbb    clerk      20    2000
4      5   eee    clerk      20    5000
5      6   fff  manager      20    6000
8      9   iii       hr      20    4000
```

Aggregations

The aggregate function return single aggregated value for each group.
Once the group by object is created we can perform different/several aggregate operations.
The aggregate operations are applied on group using ‘agg’ method



```
import pandas as pd
import numpy as np
empdict={'empno':[1,2,3,4,5,6,7,8,9,10],
         'ename':['aaa','bbb','ccc','ddd','eee','fff','ggg','hhh','iii','jjjj'],
         'job':['manager','clerk','manager','hr','clerk','manager','manager','clerk','hr',',hr'],
         'deptno':[10,20,10,10,20,20,30,10,20,30],
         'salary':[1000,2000,3000,4000,5000,6000,1000,3000,4000,8000]}
empdf=pd.DataFrame(empdict,columns=['empno','ename','job','deptno','salary'])
deptgroup=empdf.groupby('deptno')
print(deptgroup.agg(np.size))
deptgroup=empdf[['deptno','salary']].groupby('deptno')
print(deptgroup.get_group(10))
print(deptgroup.agg(np.sum))
deptgroup=empdf[['deptno','job','salary']].groupby(['deptno','job'])
for name,group in deptgroup:
  print(name)
  print(group)
print(deptgroup.get_group((10,'manager')))
print(deptgroup.agg(np.sum))
```

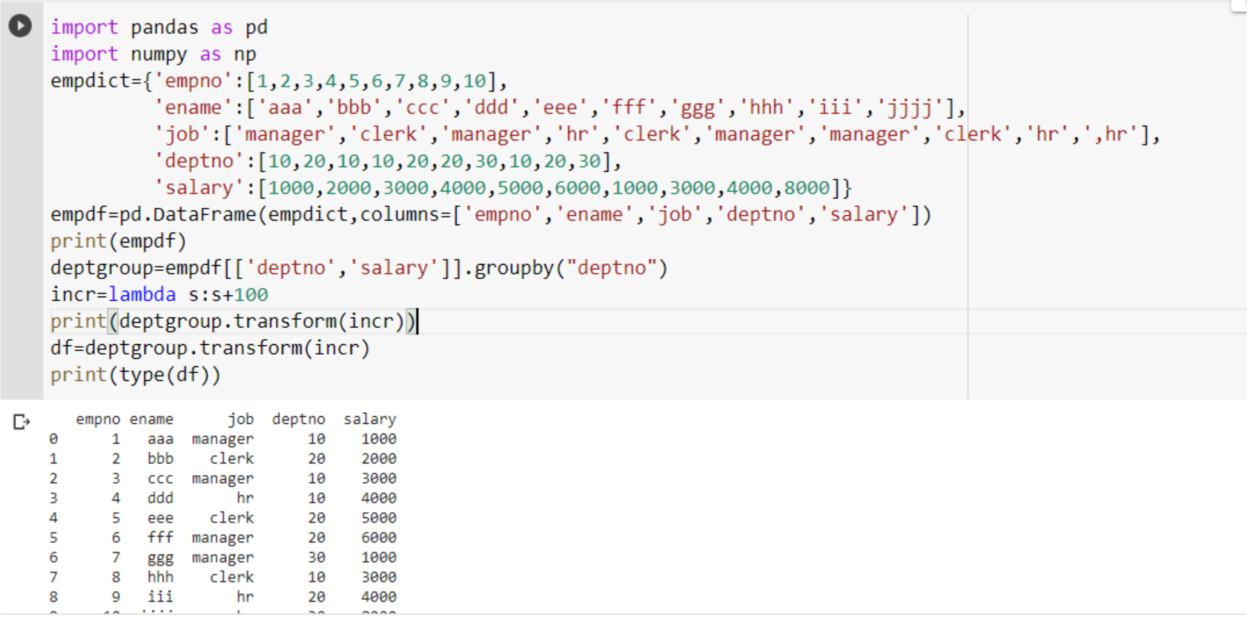


```
        empno  ename  job  salary
deptno
10          4      4    4       4
20          4      4    4       4
30          2      2    2       2
   deptno  salary
0      10    1000
2      10    3000
3      10    4000
7      10    3000
        salary
deptno
10       11000
20       17000
30        9000
(10, 'clerk')
   deptno    job  salary
7      10  clerk    3000
(10, 'hr')
   deptno job  salary
3      10  hr    4000
(10, 'manager')
   deptno      job  salary
0      10  manager    1000
2      10  manager    3000
(20, 'clerk')
   deptno    job  salary
1      20  clerk    2000
4      20  clerk    5000
(20, 'hr')
   deptno job  salary
```

✓ 0s completed at 7:18 PM

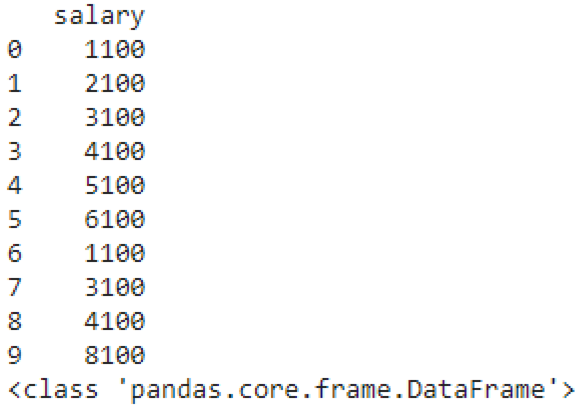
Transformations

Transformation on a group or a column return an object that is indexed the same size of that is being grouped. The transformation should return result the same size of that group.



```
import pandas as pd
import numpy as np
empdict={'empno':[1,2,3,4,5,6,7,8,9,10],
         'ename':['aaa','bbb','ccc','ddd','eee','fff','ggg','hhh','iii','jjjj'],
         'job':['manager','clerk','manager','hr','clerk','manager','manager','clerk','hr',',hr'],
         'deptno':[10,20,10,10,20,20,30,10,20,30],
         'salary':[1000,2000,3000,4000,5000,6000,1000,3000,4000,8000]}
empdf=pd.DataFrame(empdict,columns=['empno','ename','job','deptno','salary'])
print(empdf)
deptgroup=empdf[['deptno','salary']].groupby("deptno")
incr=lambda s:s+100
print(deptgroup.transform(incr))
df=deptgroup.transform(incr)
print(type(df))
```

```
   empno ename      job  deptno  salary
0      1   aaa  manager      10    1000
1      2   bbb    clerk      20    2000
2      3   ccc  manager      10    3000
3      4   ddd       hr      10    4000
4      5   eee    clerk      20    5000
5      6   fff  manager      20    6000
6      7   ggg  manager      30    1000
7      8   hhh    clerk      10    3000
8      9   iii       hr      20    4000
```

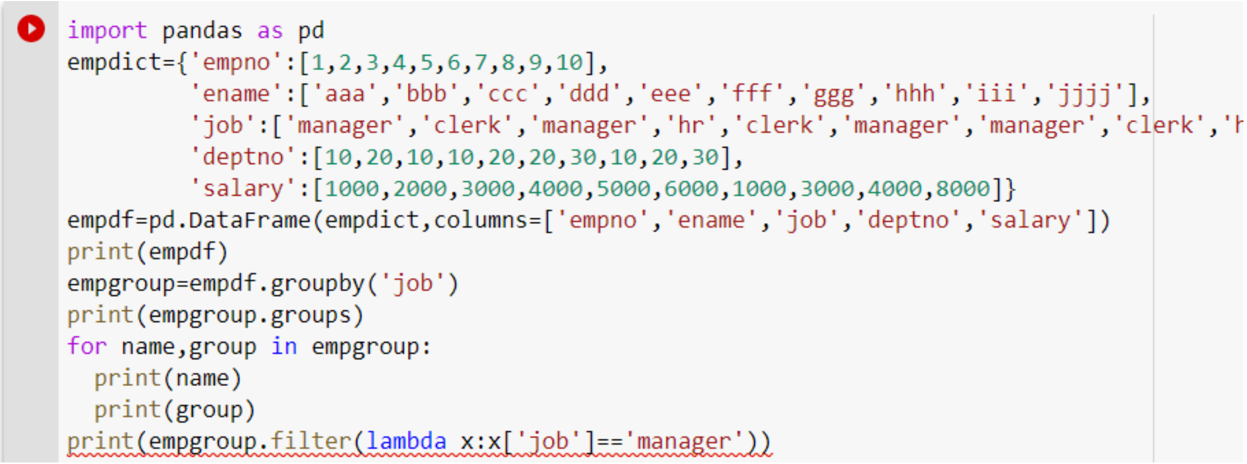


```
   salary
0    1100
1    2100
2    3100
3    4100
4    5100
5    6100
6    1100
7    3100
8    4100
9    8100
<class 'pandas.core.frame.DataFrame'>
```

Filtration

Filtration filters the data of dataset based on condition or test.
filter() function is used to filter data. This function returns subset of data which is again one dataframe.

filter(condition)



```
import pandas as pd
empdict={'empno':[1,2,3,4,5,6,7,8,9,10],
         'ename':['aaa','bbb','ccc','ddd','eee','fff','ggg','hhh','iii','jjjj'],
         'job':['manager','clerk','manager','hr','clerk','manager','manager','clerk','h
         'deptno':[10,20,10,10,20,20,30,10,20,30],
         'salary':[1000,2000,3000,4000,5000,6000,1000,3000,4000,8000]}
empdf=pd.DataFrame(empdict,columns=['empno','ename','job','deptno','salary'])
print(empdf)
empgroup=empdf.groupby('job')
print(empgroup.groups)
for name,group in empgroup:
  print(name)
  print(group)
print(empgroup.filter(lambda x:x['job']=='manager'))
```

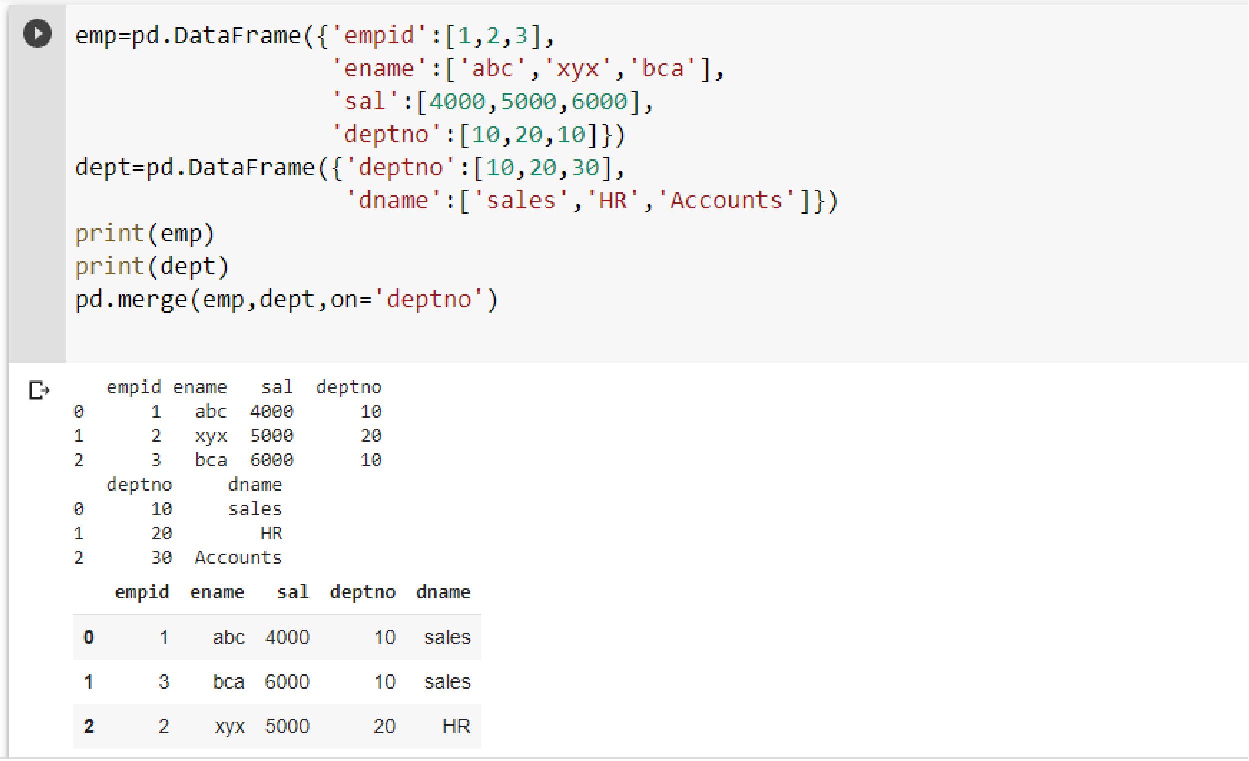
Merging and joining

Merging and joining DataFrame objects.

Pandas provide a single function called merge which is used to perform join operation between dataframe objects.

pandas.merge(left, right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'), copy=True, indicator=False, validate=None)

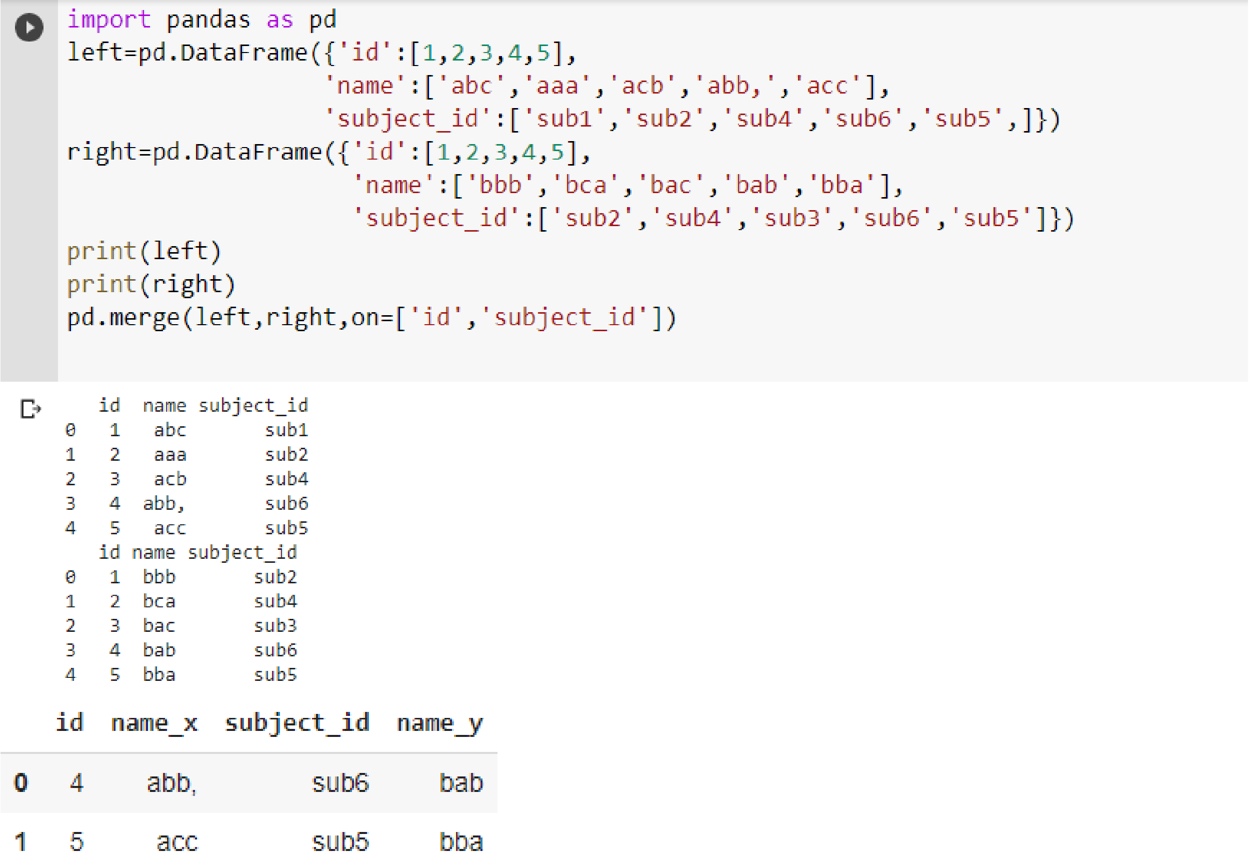
left → A dataframe object
right → A dataframe object
on → column names to join on. The column names must be exists in data frame object of left and right.
how → the value of this can be left,right,inner
sort → sort the result of dataframe



```python
emp=pd.DataFrame({'empid':[1,2,3],
                  'ename':['abc','xyx','bca'],
                  'sal':[4000,5000,6000],
                  'deptno':[10,20,10]})
dept=pd.DataFrame({'deptno':[10,20,30],
                   'dname':['sales','HR','Accounts']})
print(emp)
print(dept)
pd.merge(emp,dept,on='deptno')
```

```
   empid ename   sal  deptno
0      1   abc  4000      10
1      2   xyx  5000      20
2      3   bca  6000      10
   deptno     dname
0      10     sales
1      20        HR
2      30  Accounts
```

```
   empid  ename   sal  deptno  dname
0      1    abc  4000      10  sales
1      3    bca  6000      10  sales
2      2    xyx  5000      20     HR
```



```python
import pandas as pd
left=pd.DataFrame({'id':[1,2,3,4,5],
                   'name':['abc','aaa','acb','abb,','acc'],
                   'subject_id':['sub1','sub2','sub4','sub6','sub5',]})
right=pd.DataFrame({'id':[1,2,3,4,5],
                    'name':['bbb','bca','bac','bab','bba'],
                    'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print(left)
print(right)
pd.merge(left,right,on=['id','subject_id'])
```

```
   id  name subject_id
0   1   abc       sub1
1   2   aaa       sub2
2   3   acb       sub4
3   4  abb,       sub6
4   5   acc       sub5
   id name subject_id
0   1  bbb       sub2
1   2  bca       sub4
2   3  bac       sub3
3   4  bab       sub6
4   5  bba       sub5
```

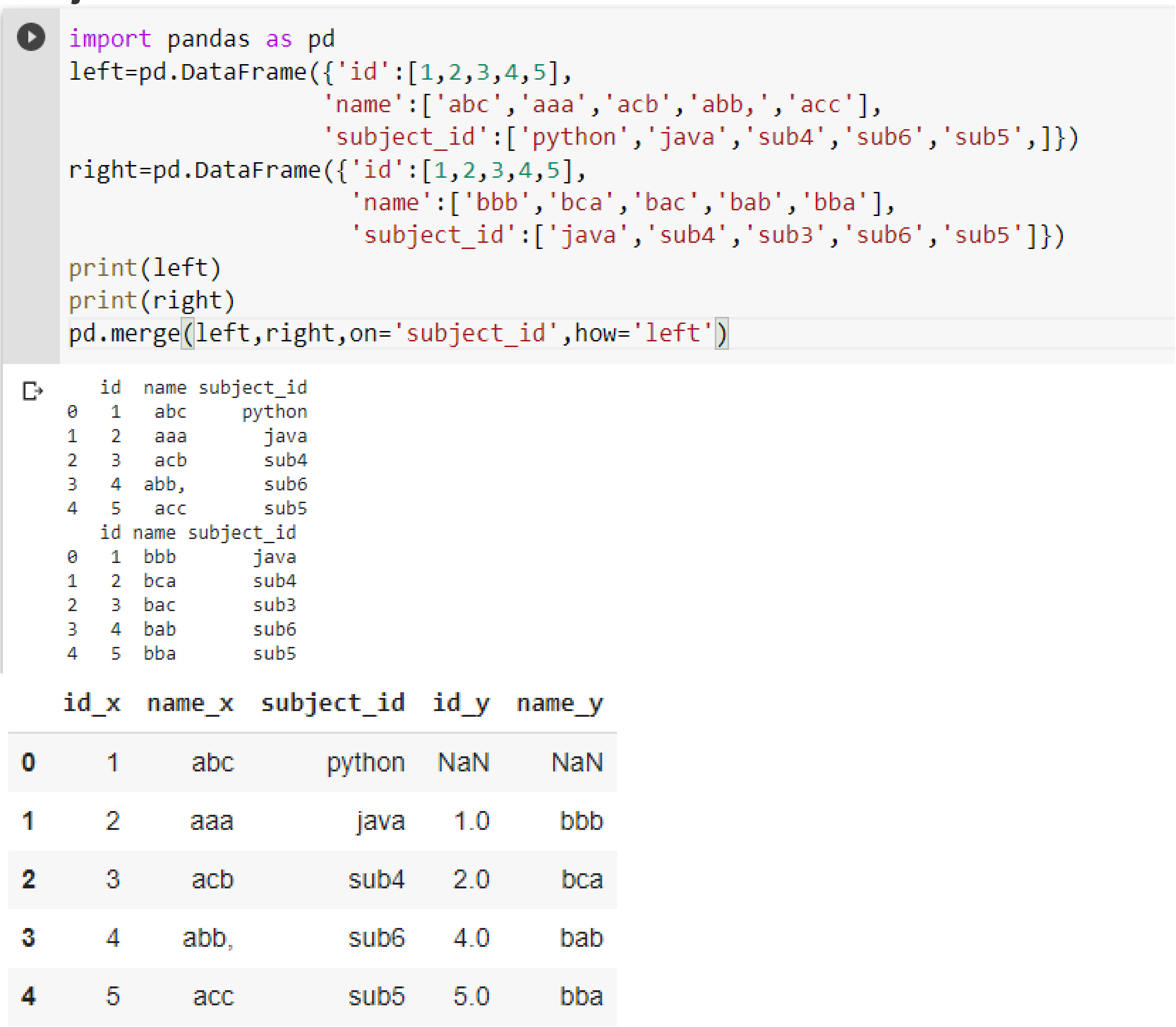
```
   id  name_x  subject_id  name_y
0   4    abb,        sub6     bab
1   5     acc        sub5     bba
```

Merging using “how” argument

The how argument of merge specifies which keys are included in the returning result dataframe/table. If key combination does not exists/appear either of dataframes/tables, the value joined table is NaN.

Merge	SQL Terminology	Description
Left	Left Outer Join	Use keys from left object
Right	Right Outer Join	Use keys from right object
Outer	Full outer join	Use Union of keys
Inner	Inner Join	Use intersection Keys

Left join

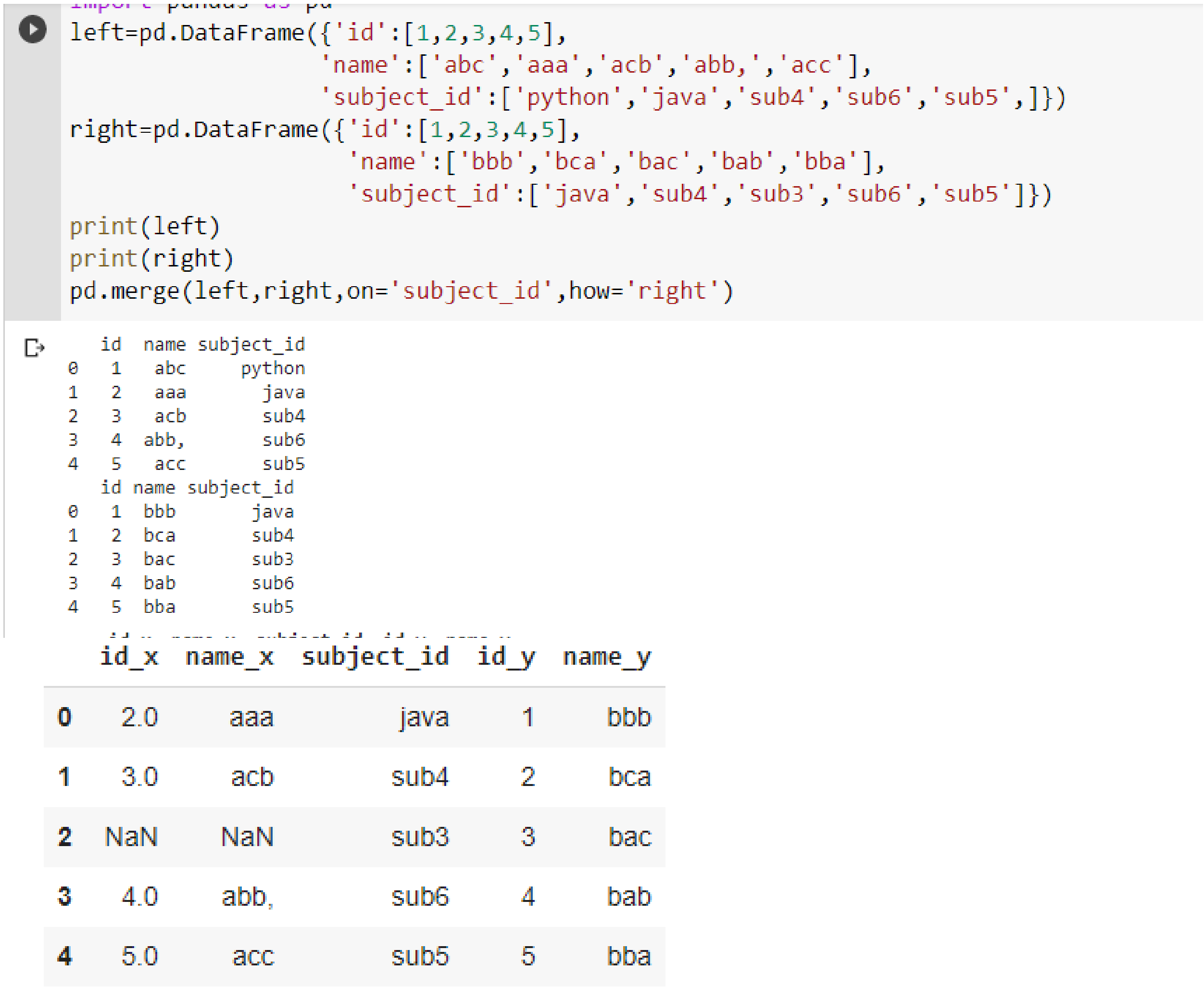


```
import pandas as pd
left=pd.DataFrame({'id':[1,2,3,4,5],
                  'name':['abc','aaa','acb','abb,','acc'],
                  'subject_id':['python','java','sub4','sub6','sub5',]})
right=pd.DataFrame({'id':[1,2,3,4,5],
                    'name':['bbb','bca','bac','bab','bba'],
                    'subject_id':['java','sub4','sub3','sub6','sub5']})
print(left)
print(right)
pd.merge(left,right,on='subject_id',how='left')
```

```
   id  name subject_id
0   1   abc     python
1   2   aaa       java
2   3   acb       sub4
3   4  abb,       sub6
4   5   acc       sub5
   id name subject_id
0   1  bbb       java
1   2  bca       sub4
2   3  bac       sub3
3   4  bab       sub6
4   5  bba       sub5
```

	id_x	name_x	subject_id	id_y	name_y
0	1	abc	python	NaN	NaN
1	2	aaa	java	1.0	bbb
2	3	acb	sub4	2.0	bca
3	4	abb,	sub6	4.0	bab
4	5	acc	sub5	5.0	bba

Right outer join

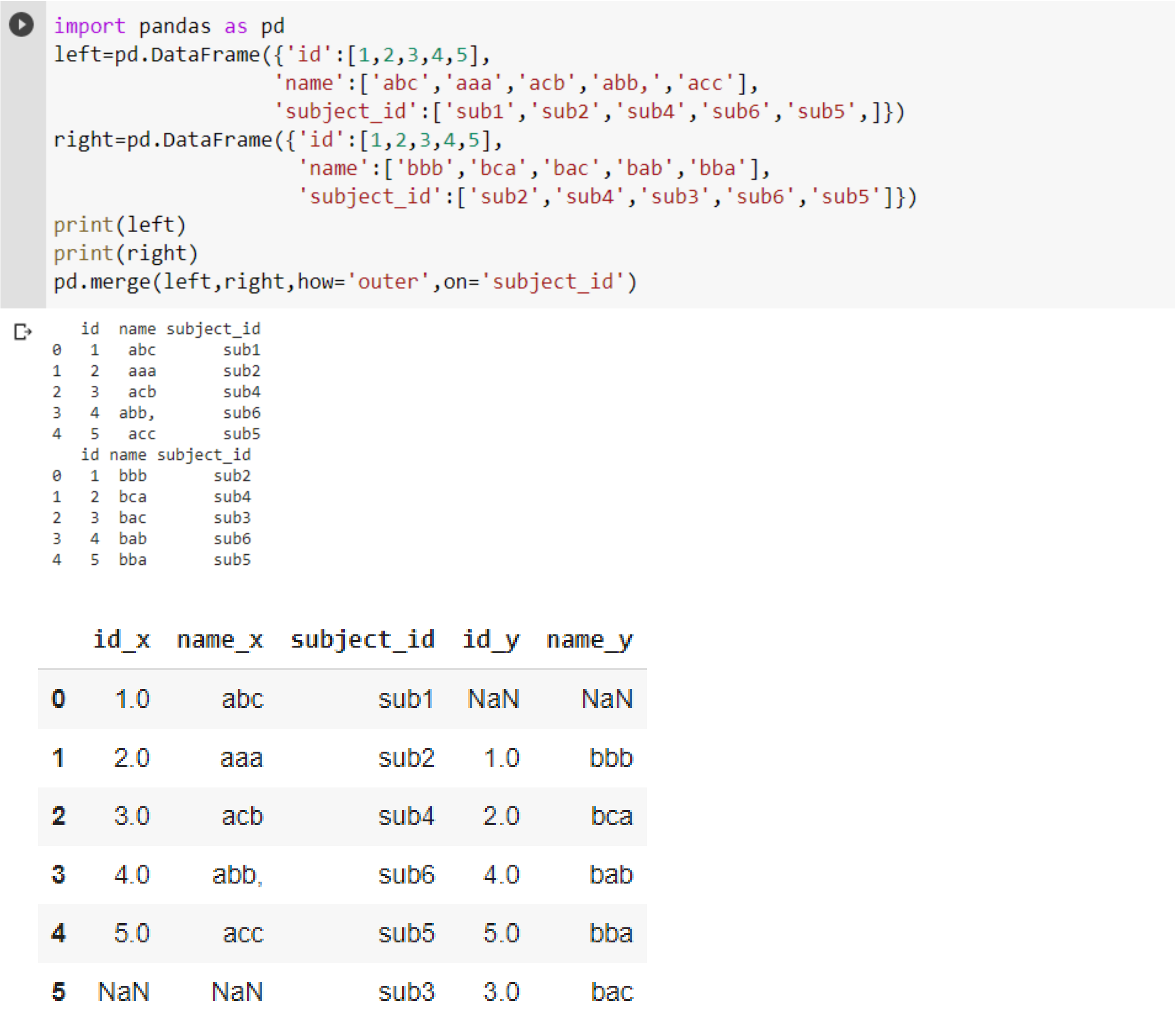


```
import pandas as pd
left=pd.DataFrame({'id':[1,2,3,4,5],
                   'name':['abc','aaa','acb','abb,','acc'],
                   'subject_id':['python','java','sub4','sub6','sub5',]})
right=pd.DataFrame({'id':[1,2,3,4,5],
                     'name':['bbb','bca','bac','bab','bba'],
                     'subject_id':['java','sub4','sub3','sub6','sub5']})
print(left)
print(right)
pd.merge(left,right,on='subject_id',how='right')
```

```
   id  name subject_id
0   1   abc     python
1   2   aaa       java
2   3   acb       sub4
3   4  abb,       sub6
4   5   acc       sub5
   id name subject_id
0   1  bbb       java
1   2  bca       sub4
2   3  bac       sub3
3   4  bab       sub6
4   5  bba       sub5
```

	id_x	name_x	subject_id	id_y	name_y
0	2.0	aaa	java	1	bbb
1	3.0	acb	sub4	2	bca
2	NaN	NaN	sub3	3	bac
3	4.0	abb,	sub6	4	bab
4	5.0	acc	sub5	5	bba

Full outer join



```
import pandas as pd
left=pd.DataFrame({'id':[1,2,3,4,5],
                   'name':['abc','aaa','acb','abb,','acc'],
                   'subject_id':['sub1','sub2','sub4','sub6','sub5',]})
right=pd.DataFrame({'id':[1,2,3,4,5],
                    'name':['bbb','bca','bac','bab','bba'],
                    'subject_id':['sub2','sub4','sub3','sub6','sub5']})
print(left)
print(right)
pd.merge(left,right,how='outer',on='subject_id')
```

```
   id  name subject_id
0   1   abc       sub1
1   2   aaa       sub2
2   3   acb       sub4
3   4  abb,       sub6
4   5   acc       sub5
   id name subject_id
0   1  bbb       sub2
1   2  bca       sub4
2   3  bac       sub3
3   4  bab       sub6
4   5  bba       sub5
```

	id_x	name_x	subject_id	id_y	name_y
0	1.0	abc	sub1	NaN	NaN
1	2.0	aaa	sub2	1.0	bbb
2	3.0	acb	sub4	2.0	bca
3	4.0	abb,	sub6	4.0	bab
4	5.0	acc	sub5	5.0	bba
5	NaN	NaN	sub3	3.0	bac

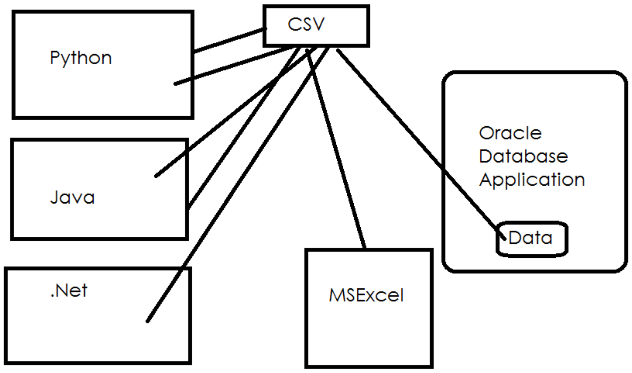
How to load data into dataframe from different source?

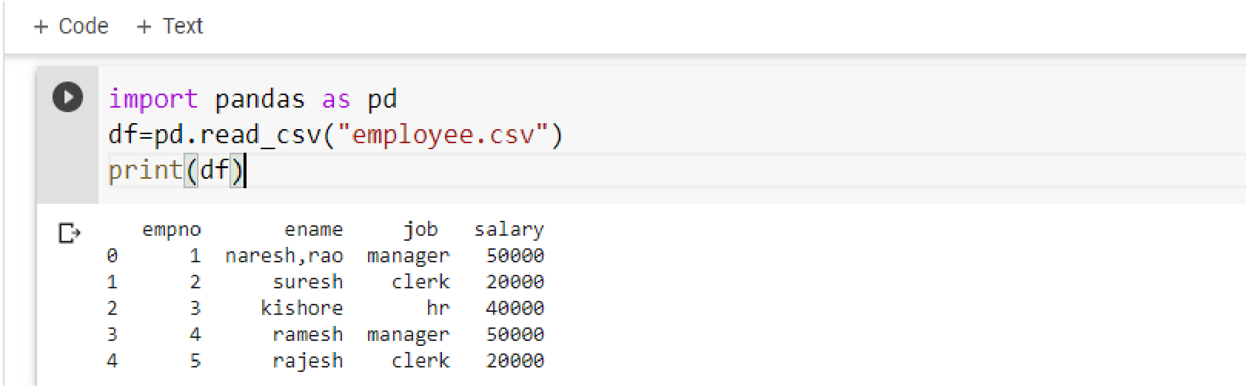
1. **Loading from CSV file**
2. **Loading from excel file**
3. **Loading from database table**
4. **Loading from JSON file**
5. **SAS**
6. **SPSS**
7. **STATA**

CSV File

CSV stands for comma separated values.

CSV is standard or protocol. This standard is used to exchange data between two different applications.
CSV standard import and export format used by database applications and many programming languages.



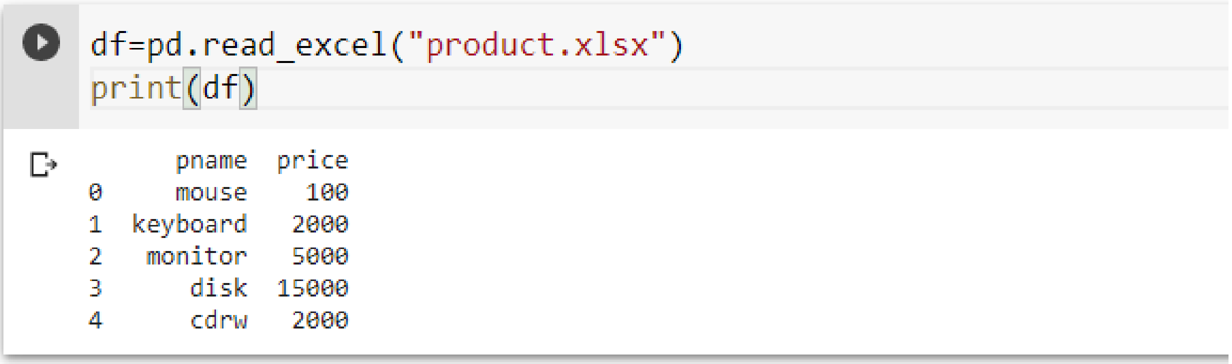


```
import pandas as pd
df=pd.read_csv("employee.csv")
print(df)
```

```
   empno        ename      job  salary
0      1   naresh,rao  manager   50000
1      2       suresh    clerk   20000
2      3      kishore       hr   40000
3      4       ramesh  manager   50000
4      5       rajesh    clerk   20000
```

Reading Excel Sheet

We can read and load the data from excel shell data frame.
This is done using a function pandas.read_excel(path)



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

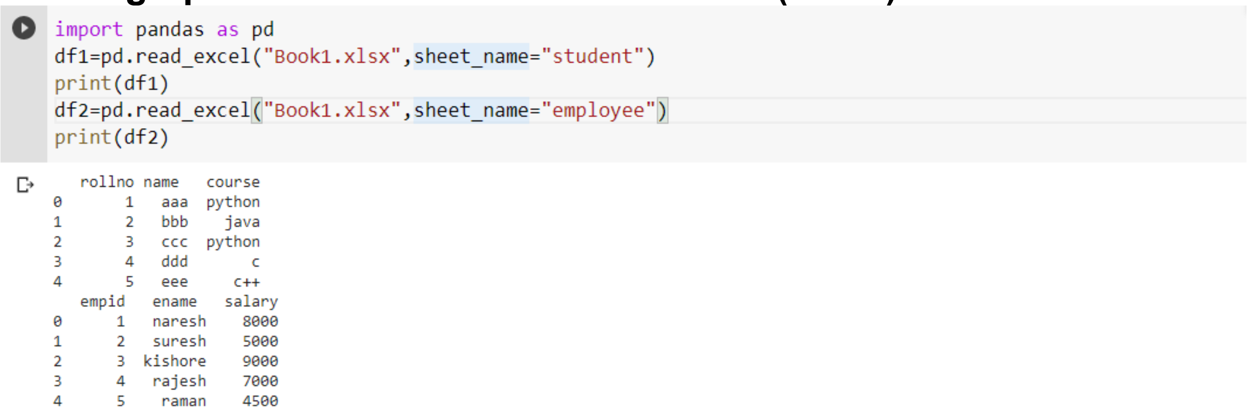
```
      pname  price
0     mouse    100
1  keyboard   2000
2   monitor   5000
3      disk  15000
4      cdrw   2000
```

Reading JSON file

JSON stands Java Script Object Notation. It is standard which is used to exchange data between two different applications. JSON is a text file. It is used in internet applications exchange data between client and server or between the servers.

pandas.read_json()
This is used to read data from json file.

Reading Specific worksheet from workbook(Excel)

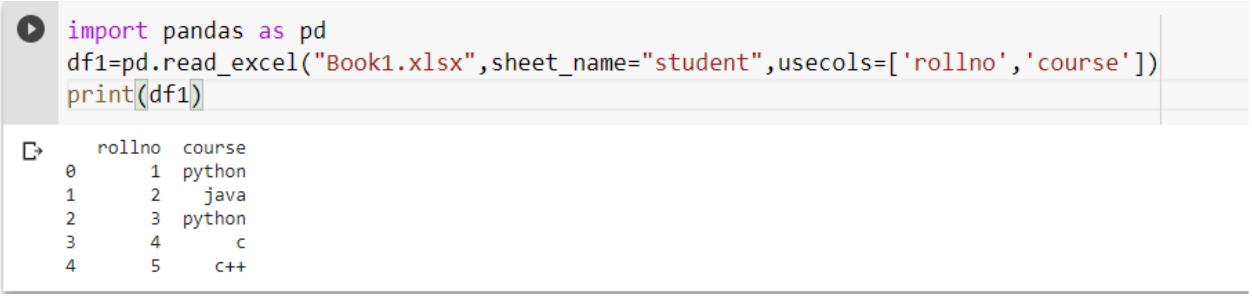


```
import pandas as pd
df1=pd.read_excel("Book1.xlsx",sheet_name="student")
print(df1)
df2=pd.read_excel("Book1.xlsx",sheet_name="employee")
print(df2)
```

```
   rollno name  course
0       1  aaa  python
1       2  bbb    java
2       3  ccc  python
3       4  ddd       c
4       5  eee     c++
   empid    ename  salary
0      1   naresh    8000
1      2   suresh    5000
2      3  kishore    9000
3      4   rajesh    7000
4      5    raman    4500
```

How to read specific Columns from worksheet?

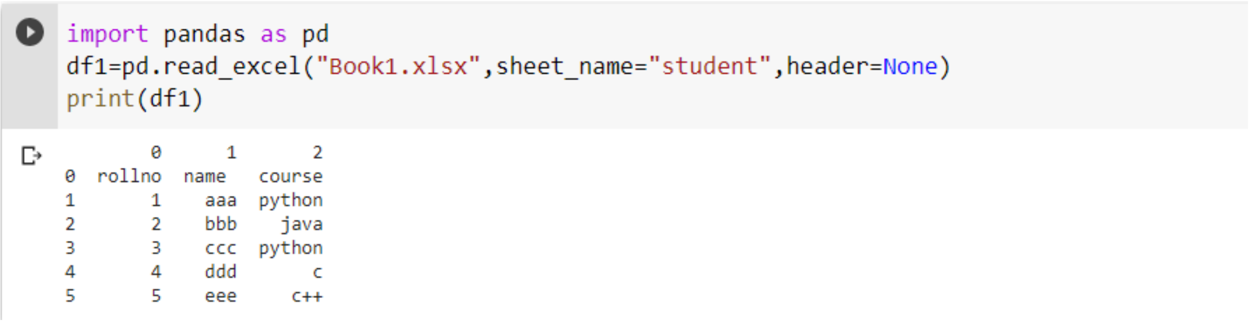
read_excel function provide an optional argument called usecols, by default it read all the columns from worksheet.



```
import pandas as pd
df1=pd.read_excel("Book1.xlsx",sheet_name="student",usecols=['rollno','course'])
print(df1)
```

```
   rollno  course
0       1  python
1       2    java
2       3  python
3       4       c
4       5     c++
```

How to read data from worksheet without header row?

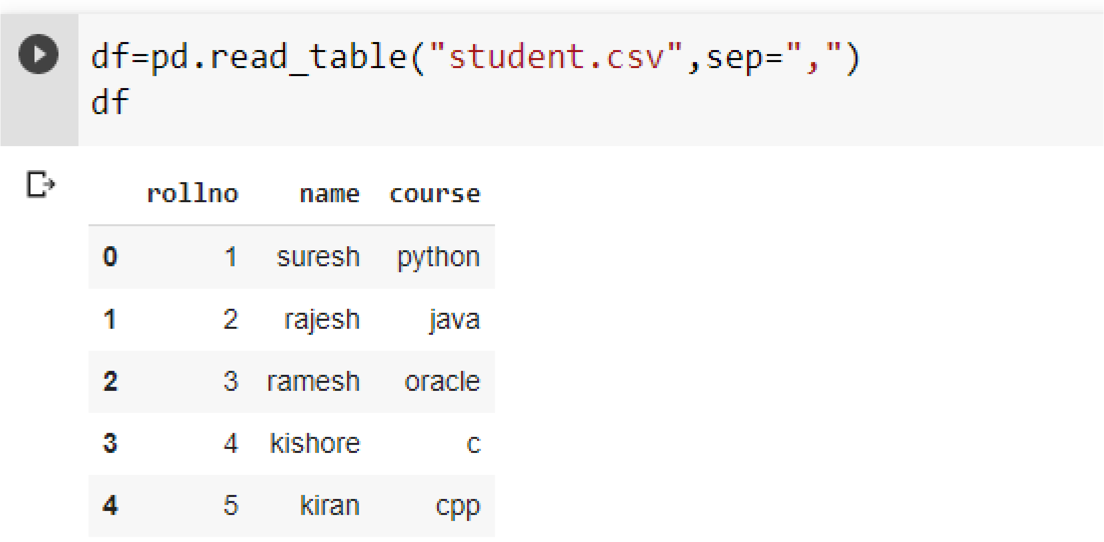


```
import pandas as pd
df1=pd.read_excel("Book1.xlsx",sheet_name="student",header=None)
print(df1)
```

```
        0     1       2
0  rollno  name  course
1       1   aaa  python
2       2   bbb    java
3       3   ccc  python
4       4   ddd       c
5       5   eee     c++
```

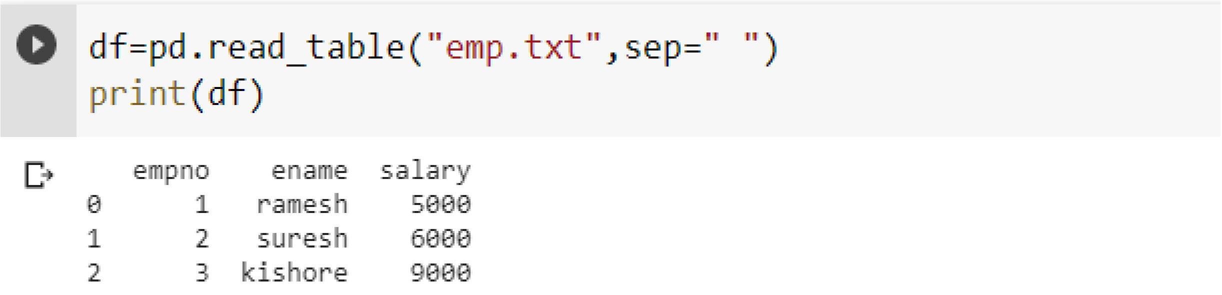
pandas.read_table()

this function is used to read data from text file. It will read text file and return data in table format.



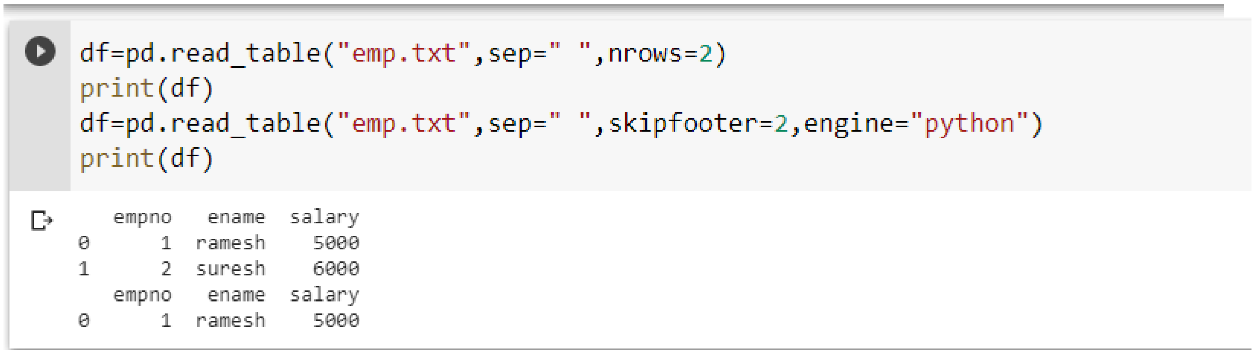
```
df=pd.read_table("student.csv",sep=",")
df
```

```
   rollno     name  course
0       1   suresh  python
1       2   rajesh    java
2       3   ramesh  oracle
3       4  kishore       c
4       5    kiran     cpp
```



```
df=pd.read_table("emp.txt",sep=" ")
print(df)
```

```
   empno    ename  salary
0      1   ramesh    5000
1      2   suresh    6000
2      3  kishore    9000
```



```
df=pd.read_table("emp.txt",sep=" ",nrows=2)
print(df)
df=pd.read_table("emp.txt",sep=" ",skipfooter=2,engine="python")
print(df)
```

```
   empno   ename  salary
0      1  ramesh    5000
1      2  suresh    6000
   empno   ename  salary
0      1  ramesh    5000
```