Pandas

Series

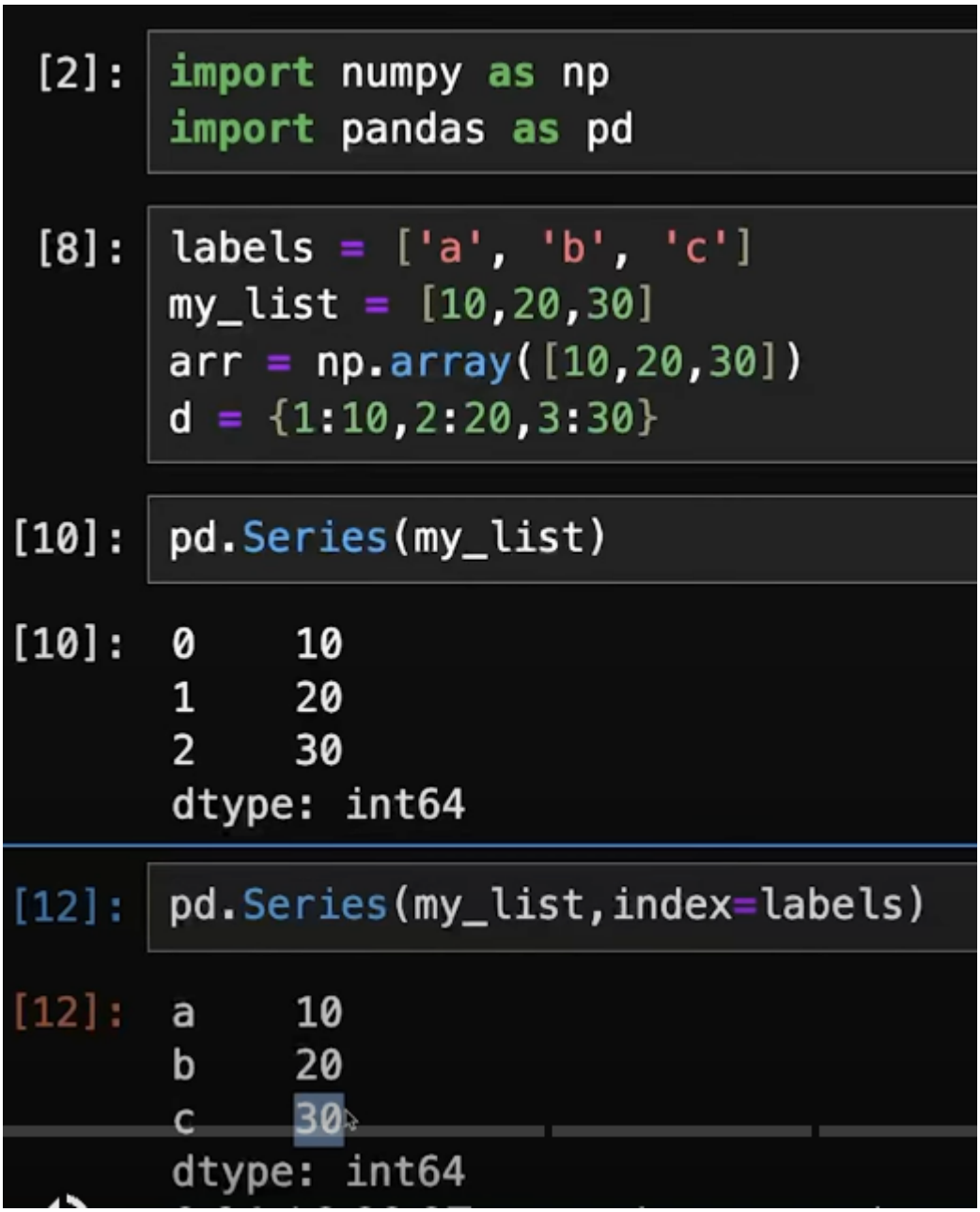
```
import numpy as np ;
import pandas as pd;
```

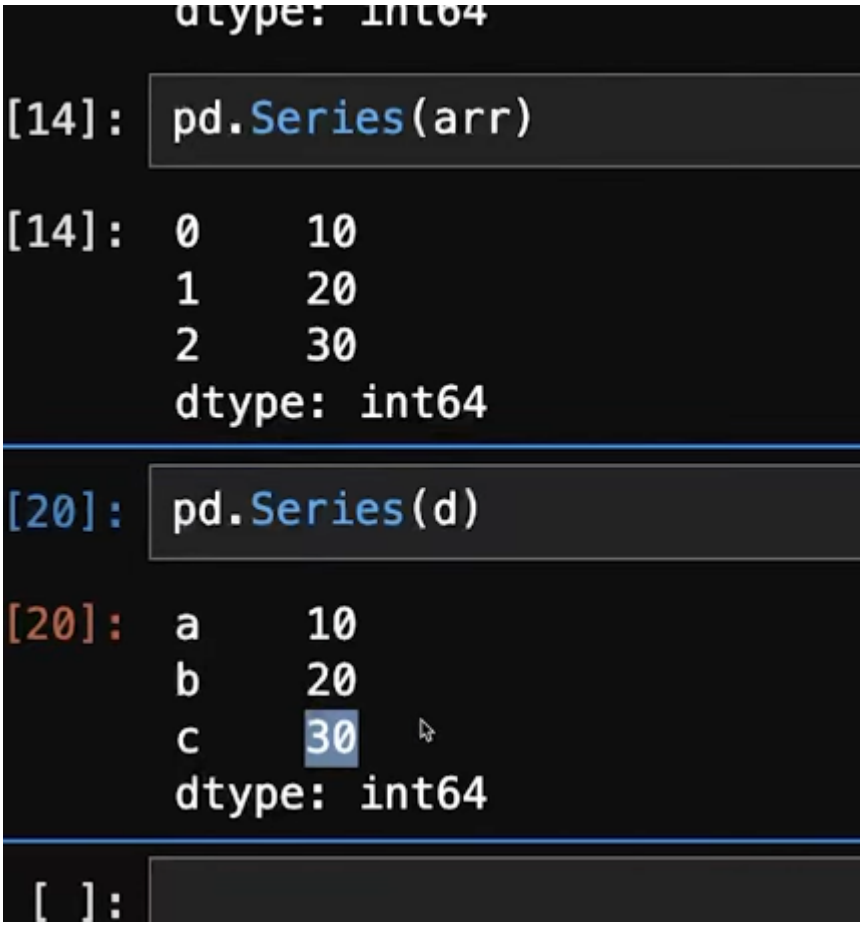
Series এর দুইটা পার্ট থাকে একটা Datapoin আরেকটা IndexPoint আমরা তো Datapoint হিসেবে array , list দিতে পারি । আর যদি আমরা indexpoint না দেই তাহলে হচ্ছে আমাদের indexpoint 0 থেকে start হয়ে যাবে ।

```
labels = ['a','b','c']
my_list = [10,20,30]
arr = np.array(my_list)
d = {1 : 10 , 2: 25 , 3: 30}

# এখন আমরা যদি Series বানাতে চাই আমরা Datapoint হিসেবে"
# array / list provide করব প্রথম parameter হিসেবে ।
#  আর পরে indexpoint হিসেবে 2nd parameter দিব "

pd.Series(my_list, index = labels) ;
```

আর যদি আমরা pd.series() এর মধ্যে dictionary দিয়ে দেই তাহলে আমরা আলাদা করে indexpoint দেওয়ার দরকার হয় না কারণ dictionary এর key টা indexpoint হিসেবে কাজ করে।

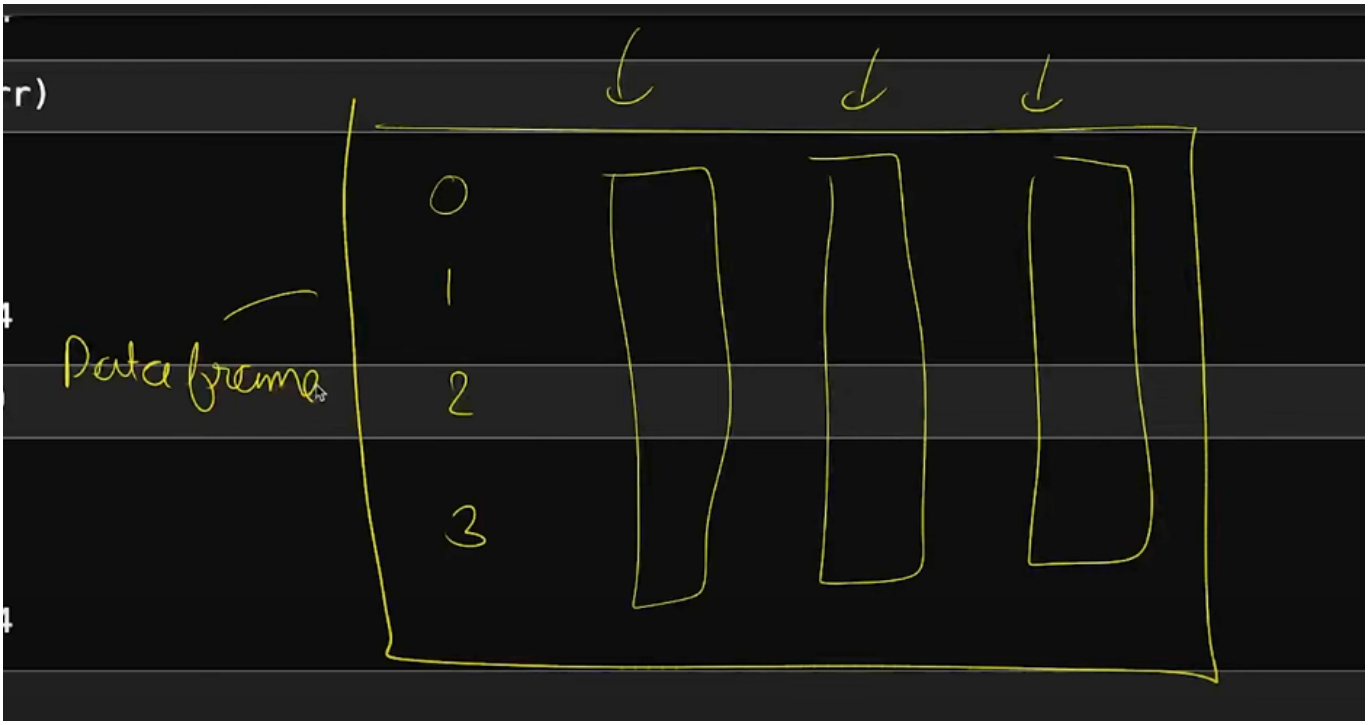
Series এ আমরা one dimetional array দিব কারণ series one dimetional .

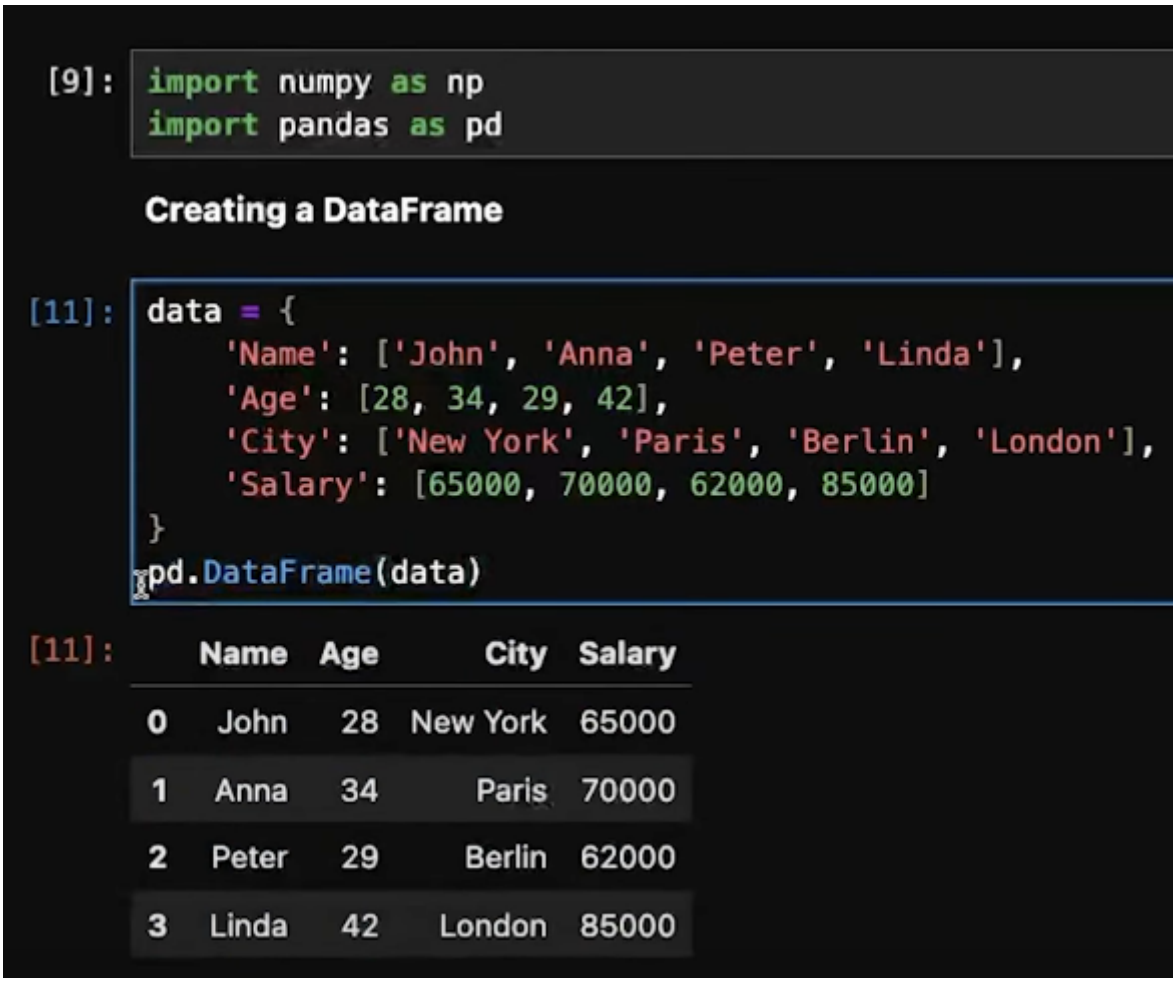

[2]:
import numpy as np
import pandas as pd
[8]:
labels = ['a', 'b', 'c']
my_list = [10,20,30]
arr = np.array([10,20,30])
d = {1:10,2:20,3:30}
[10]:
pd.Series(my_list)
[10]:
0 10
1 20
2 30
dtype: int64
[12]:
pd.Series(my_list,index=labels)
[12]:
a 10
b 20
c 30
dtype: int64



Dataframe

Multiple Series কে একসাথে combine করে দিলে এইটা dataframe এ কনভার্ট হয়ে যায়।





[9]:
```
import numpy as np
import pandas as pd
```

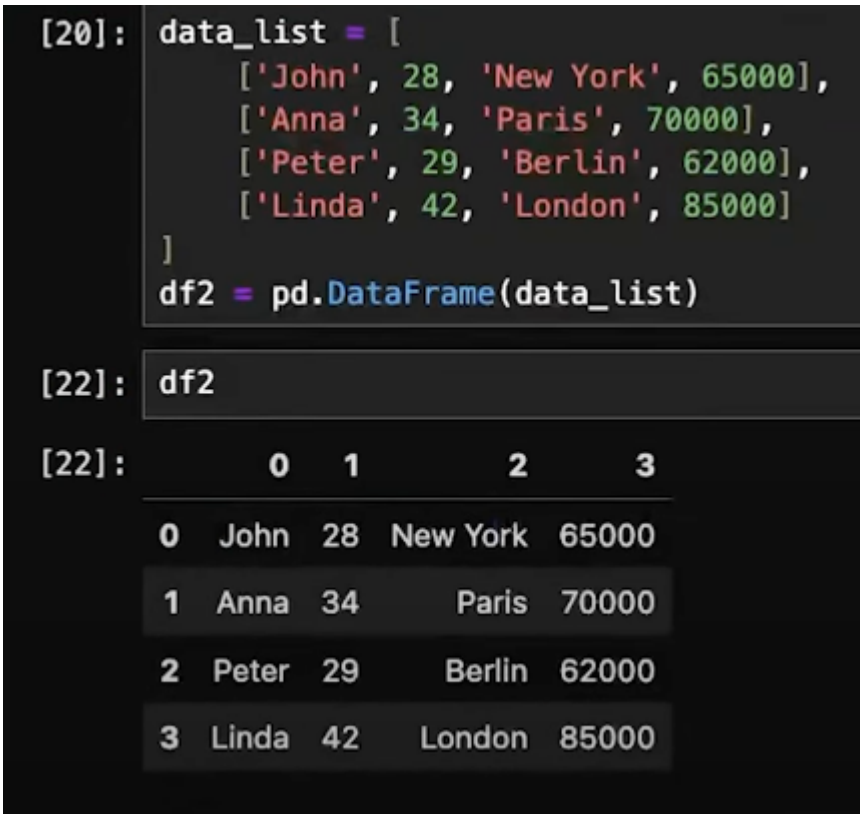
Creating a DataFrame

[11]:
```
data = {
    'Name': ['John', 'Anna', 'Peter', 'Linda'],
    'Age': [28, 34, 29, 42],
    'City': ['New York', 'Paris', 'Berlin', 'London'],
    'Salary': [65000, 70000, 62000, 85000]
}
pd.DataFrame(data)
```

[11]:

	Name	Age	City	Salary
0	John	28	New York	65000
1	Anna	34	Paris	70000
2	Peter	29	Berlin	62000
3	Linda	42	London	85000

Dataframe এ যদি একটা list provide করি তাহলে আমাদের কলাম নাম ডিফল্ট ভাবে ০,১... হয়ে সেট হয়।

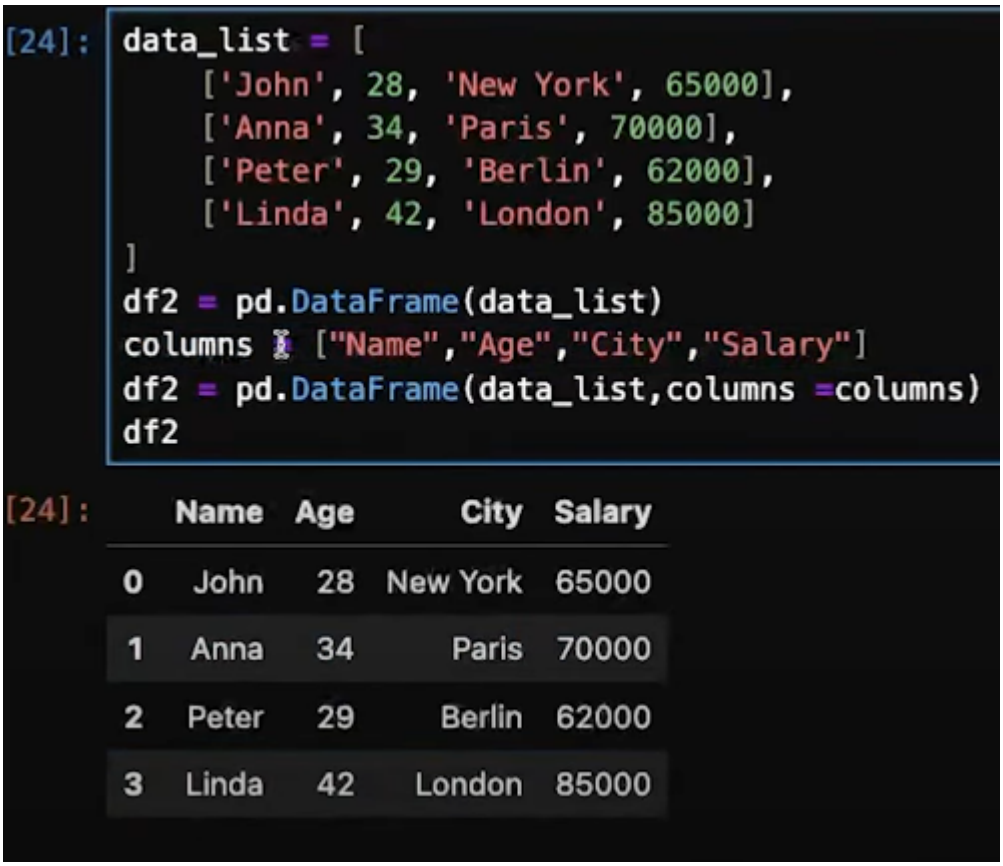


[20]:
```
data_list = [
    ['John', 28, 'New York', 65000],
    ['Anna', 34, 'Paris', 70000],
    ['Peter', 29, 'Berlin', 62000],
    ['Linda', 42, 'London', 85000]
]
df2 = pd.DataFrame(data_list)
```

[22]:
```
df2
```

[22]:

	0	1	2	3
0	John	28	New York	65000
1	Anna	34	Paris	70000
2	Peter	29	Berlin	62000
3	Linda	42	London	85000

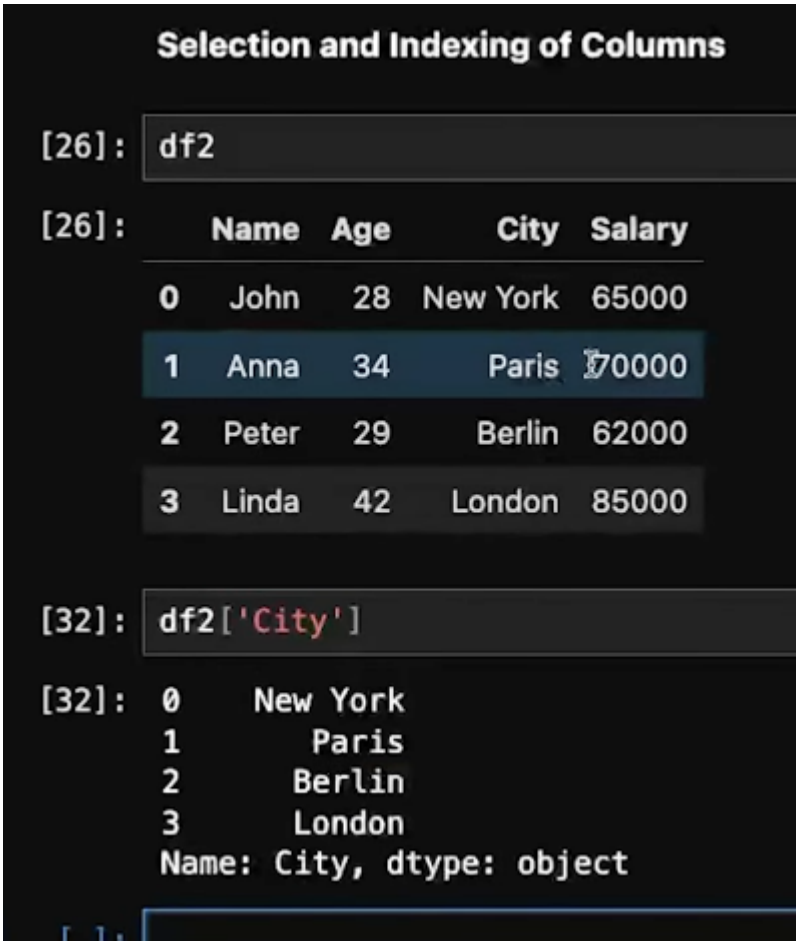


```
[24]: data_list = [
          ['John', 28, 'New York', 65000],
          ['Anna', 34, 'Paris', 70000],
          ['Peter', 29, 'Berlin', 62000],
          ['Linda', 42, 'London', 85000]
      ]
      df2 = pd.DataFrame(data_list)
      columns = ["Name","Age","City","Salary"]
      df2 = pd.DataFrame(data_list,columns =columns)
      df2
```

[24]:

	Name	Age	City	Salary
0	John	28	New York	65000
1	Anna	34	Paris	70000
2	Peter	29	Berlin	62000
3	Linda	42	London	85000

Selecting , Creating & Removing Column



Selection and Indexing of Columns

```
[26]: df2
```

[26]:

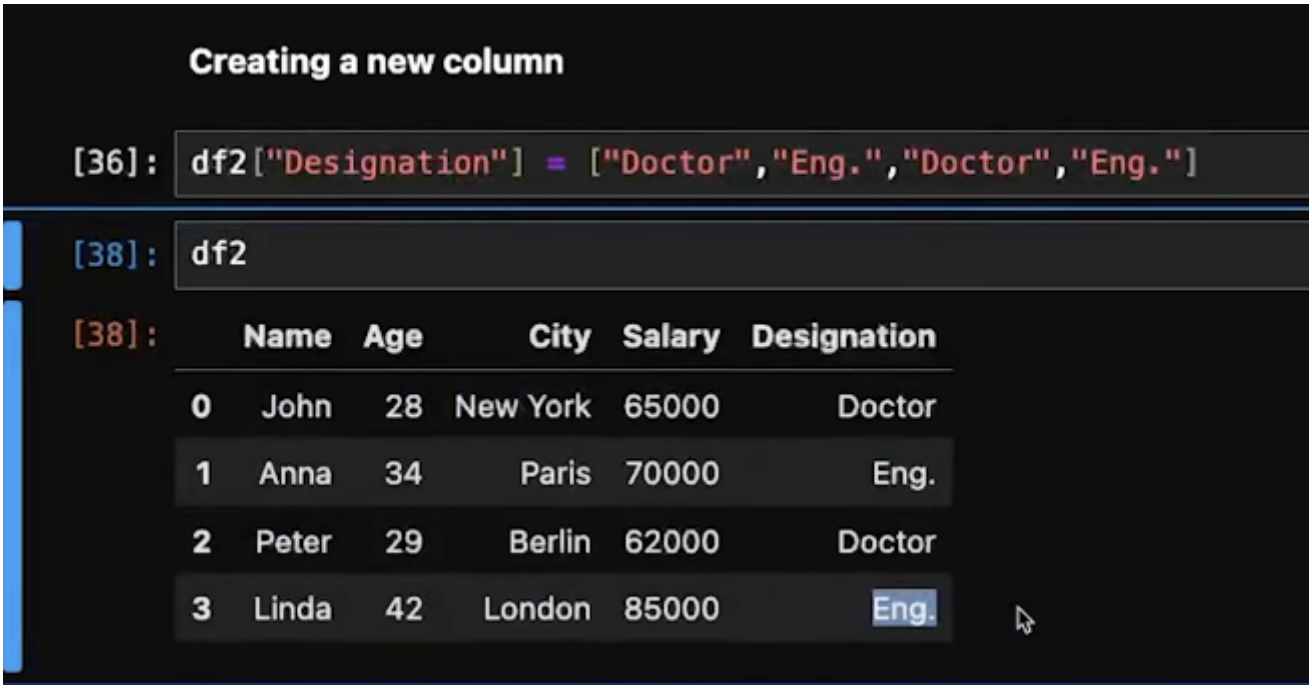
	Name	Age	City	Salary
0	John	28	New York	65000
1	Anna	34	Paris	70000
2	Peter	29	Berlin	62000
3	Linda	42	London	85000

```
[32]: df2['City']
```

```
[32]: 0    New York
      1       Paris
      2      Berlin
      3      London
      Name: City, dtype: object
```

কয়েকটা ইনডেক্স একসাথে দেখতে চাইলে। (fancy index এর মতো)

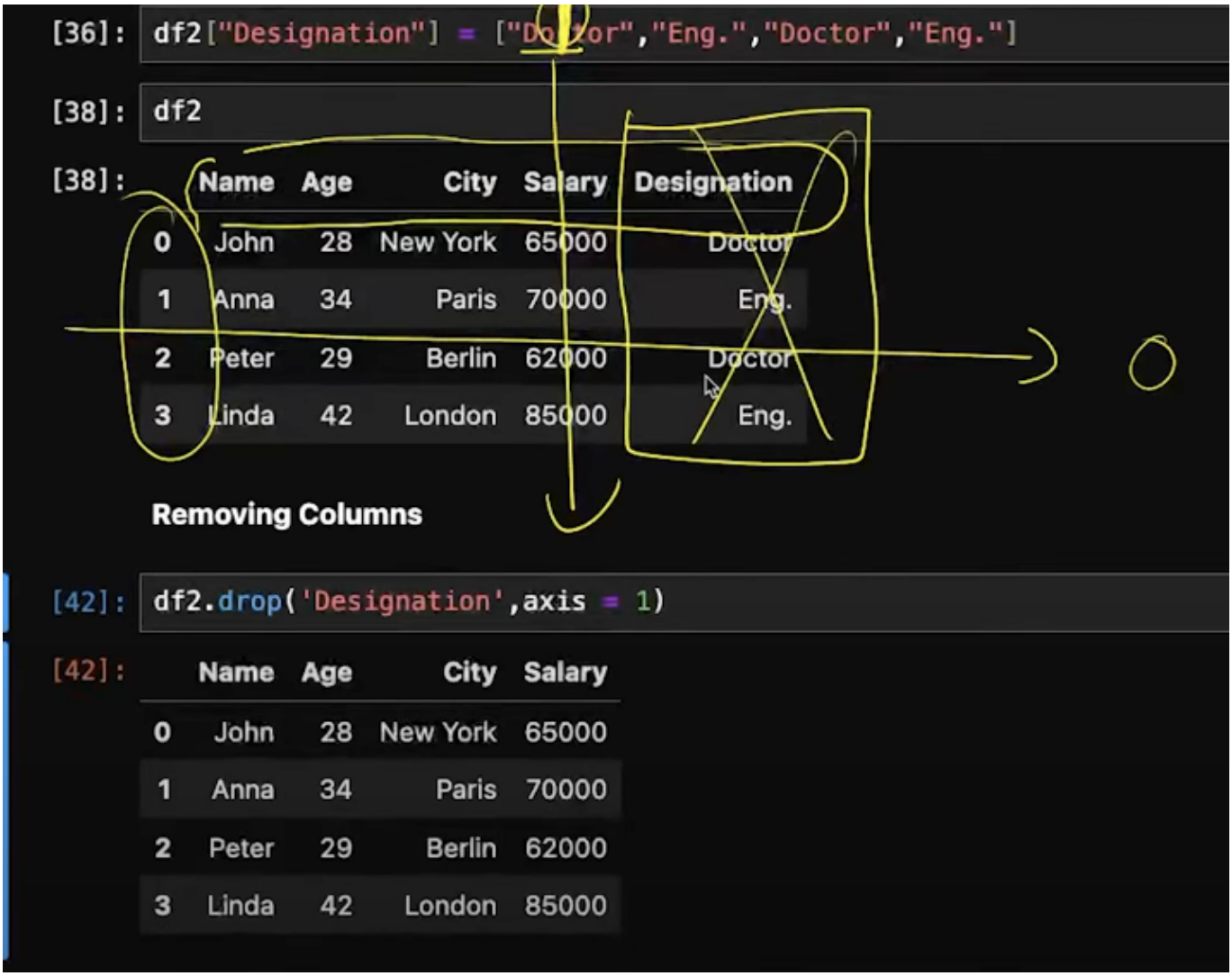
Creating a new Column:



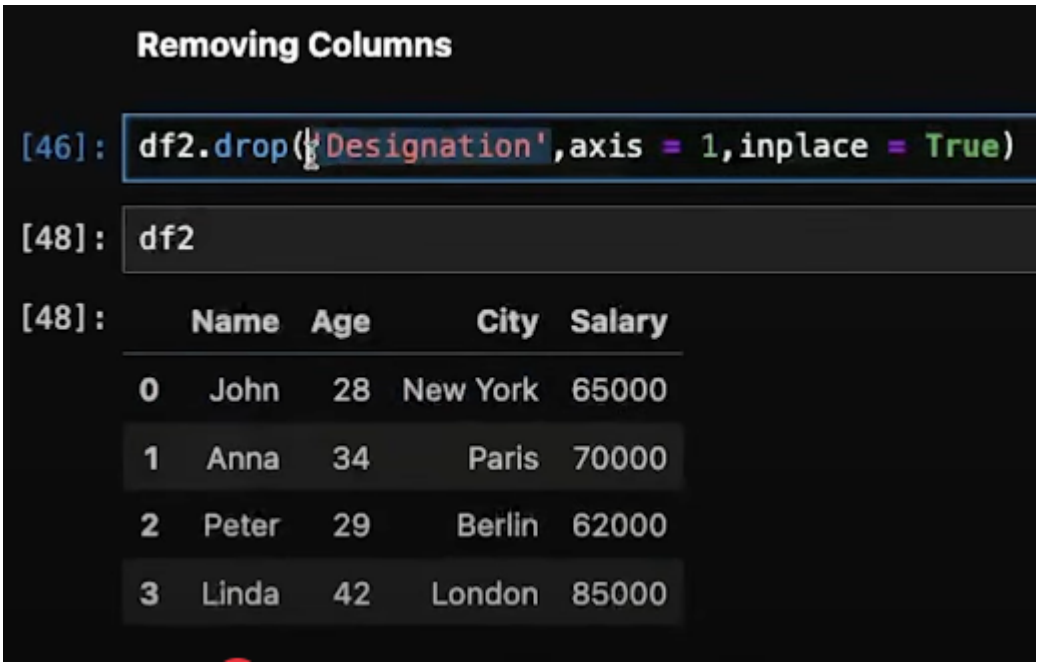
```python
import pandas as pd

data = {
    "Name":['Ram','Shyam','Ghanshyam','Dhanshyam','Aditi','Jagdish','Raj','Simran'],
    "Age":[28,34,22,30,29,40,25,32],
    "Salary":[50000,60000,45000,52000,49000,70000,48000,58000],
    "Performance_Score":[85,90,78,92,88,95,80,89]
}
```

```
        Name  Age  Salary  Performance_Score
0        Ram   28   50000                 85
1      Shyam   34   60000                 90
2  Ghanshyam   22   45000                 78
3  Dhanshyam   30   52000                 92
4      Aditi   29   49000                 88
5    Jagdish   40   70000                 95
6        Raj   25   48000                 80
7     Simran   32   58000                 89
        Name  Age  Salary  Performance_Score   Bonus
0        Ram   28   50000                 85  5000.0
1      Shyam   34   60000                 90  6000.0
2  Ghanshyam   22   45000                 78  4500.0
3  Dhanshyam   30   52000                 92  5200.0
4      Aditi   29   49000                 88  4900.0
5    Jagdish   40   70000                 95  7000.0
6        Raj   25   48000                 80  4800.0
7     Simran   32   58000                 89  5800.0
```

Removing Column

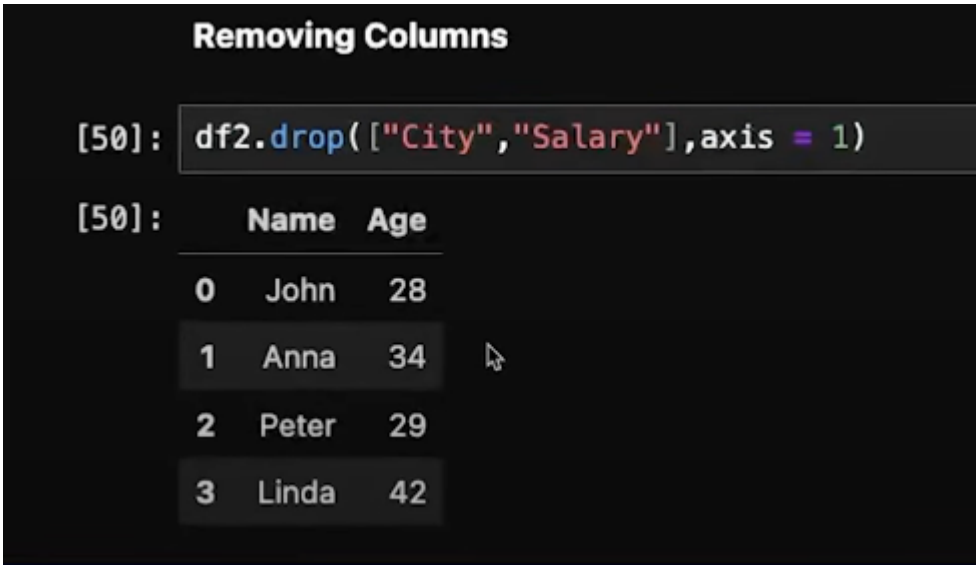


কিন্তু আমরা যদি inplace = True দেই তাহলে তখন এই টা main memory থেকেও delete হয়ে যাবে।



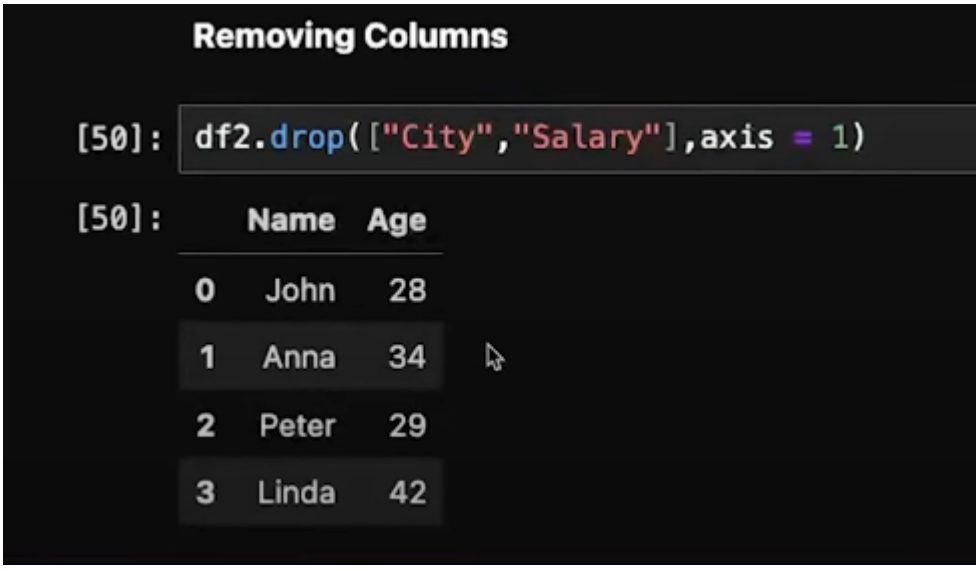
Removing multiple column .

```
# df.drop(columns = ["ColumnName"], inplace=True)
```



Removing multiple column .

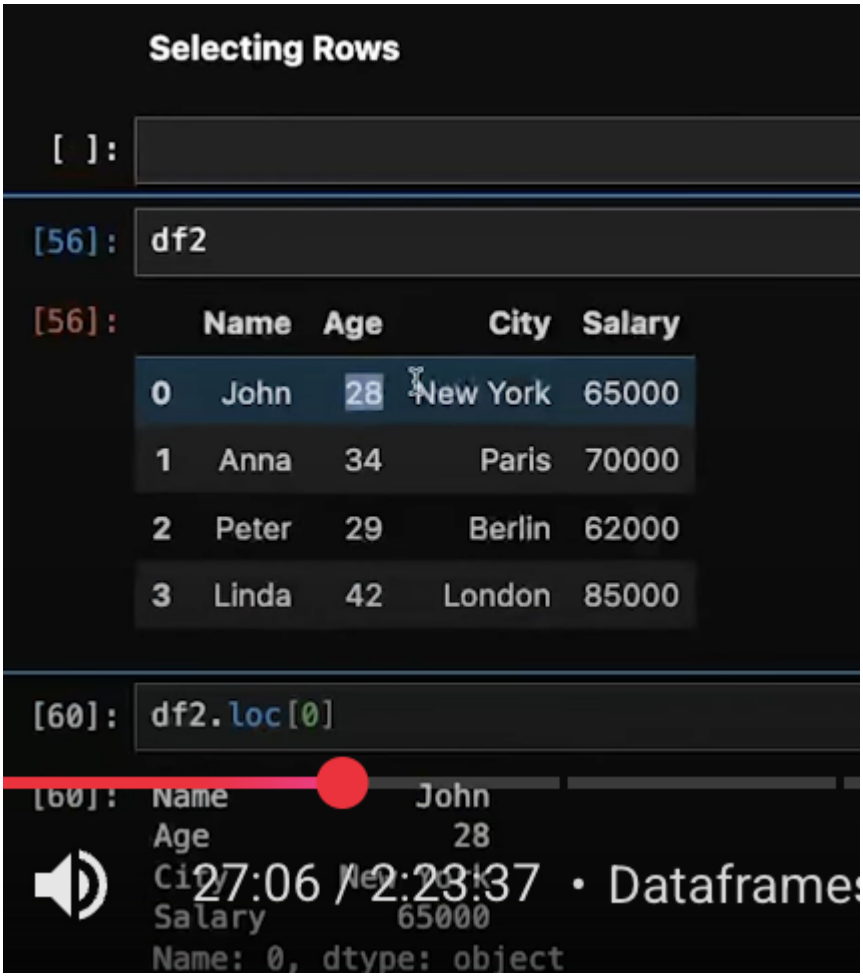

```
# df.drop(columns = ["ColumnName"], inplace=True)
```

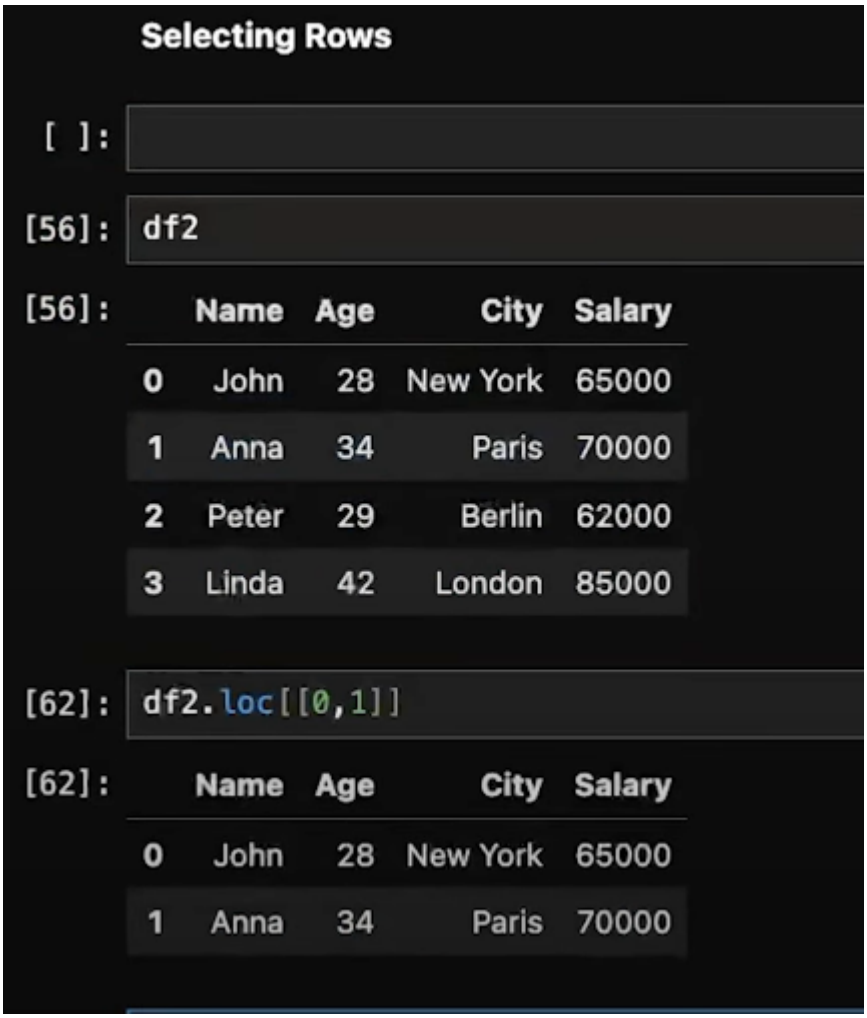
Inplace = True মানে মেইন ডাটাসেট এ ভ্যালু পরিবর্তন যেন হয়ে যায়।

Removing row :

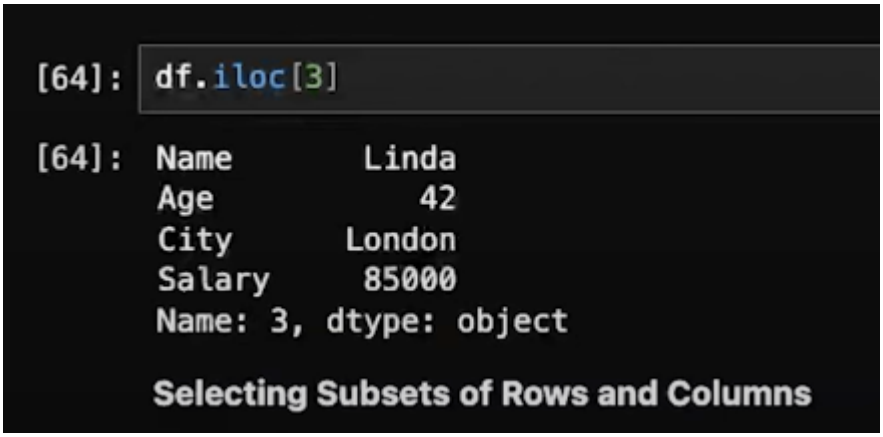
আমরা row select করার সময় আমরা loc[] function use করব।

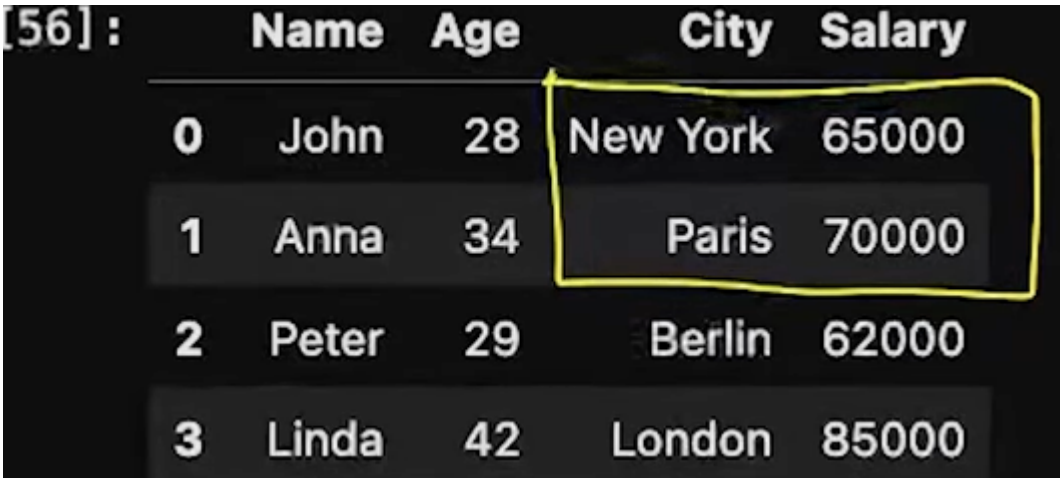


Selecting Multiple Row :



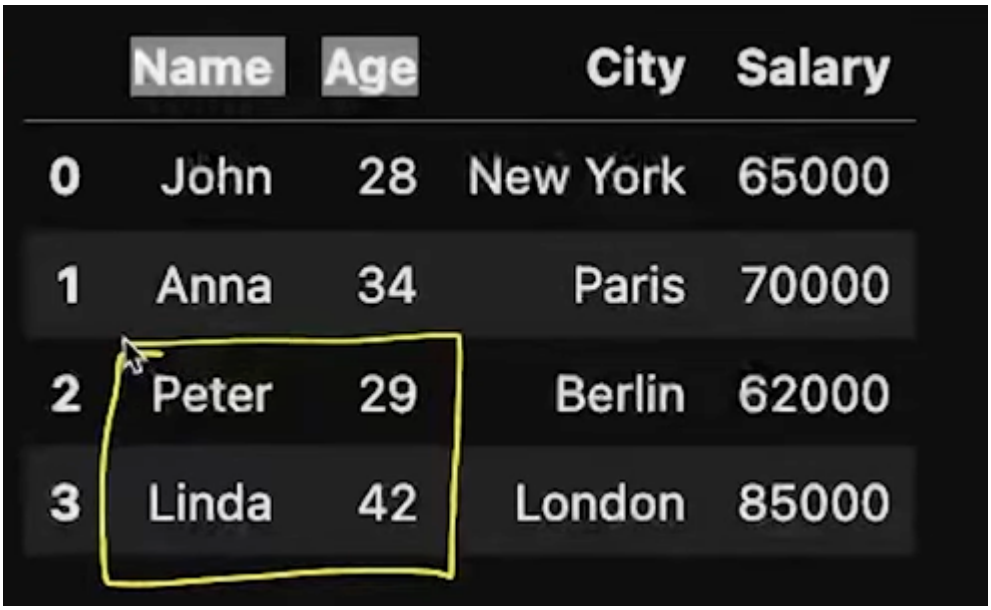
Iloc - index location .





[56]:	Name	Age	City	Salary
0	John	28	New York	65000
1	Anna	34	Paris	70000
2	Peter	29	Berlin	62000
3	Linda	42	London	85000

```
# first of all row select করব তারপর column select করব
df2.loc[[0,1]][['City' , 'Salary']]
```



	Name	Age	City	Salary
0	John	28	New York	65000
1	Anna	34	Paris	70000
2	Peter	29	Berlin	62000
3	Linda	42	London	85000

```
df2.loc[[2,3]][['Name','Age']]
```

Adding new columns specific position Insert method :

```
data = {
    "Name":['Ram','Shyam','Ghanshyam','Dhanshyam','Aditi','Jagdish','Raj','Simran'],
    "Age":[28,34,22,30,29,40,25,32],
    "Salary":[50000,60000,45000,52000,49000,70000,48000,58000],
    "Performance_Score":[85,90,78,92,88,95,80,89]
}
```

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

df["Bonus"] = df['Salary'] * 0.1
print(df)

#using insert()
# df.insert(loc, "Column_Name", some_data)
df.insert(0, "Employee ID", [10,20,30,40,50,60,70,80])
print(df)
```

	Employee ID	Name	Age	Salary	Performance_Score	Bonus
0	10	Ram	28	50000	85	5000.0
1	20	Shyam	34	60000	90	6000.0
2	30	Ghanshyam	22	45000	78	4500.0
3	40	Dhanshyam	30	52000	92	5200.0
4	50	Aditi	29	49000	88	4900.0
5	60	Jagdish	40	70000	95	7000.0
6	70	Raj	25	48000	80	4800.0
7	80	Simran	32	58000	89	5800.0

UPDATE METHOD

```python
data = {
    "Name":['Ram','Shyam','Ghanshyam','Dhanshyam','Aditi','Jagdish','Raj','Simran'],
    "Age":[28,34,22,30,29,40,25,32],
    "Salary":[50000,60000,45000,52000,49000,70000,48000,58000],
    "Performance_Score":[85,90,78,92,88,95,80,89]
}
```

Row_index : মানে আমরা কোন রো টা update করতে চাচ্ছি। df.loc[0,'salary'] = 55000

```python
# .loc[]
# df.loc[row_index, "Column Name"] = new_value
df.loc[0, 'Salary'] = 55000
print(df)
```

Conditional Selection

Conditional Selection

[70]:
```
df2
```

[70]:

	Name	Age	City	Salary
0	John	28	New York	65000
1	Anna	34	Paris	70000
2	Peter	29	Berlin	62000
3	Linda	42	London	85000

[ ]:
```
#I only want to see those people whose age is above 30
```

[72]:
```
df2[df2["Age"] > 30]
```

[72]:

	Name	Age	City	Salary
1	Anna	34	Paris	70000
3	Linda	42	London	85000

]:
```
#I only want poeple whose age is above 30 and their city must be paris
```

6]:
```
df2[(df2["Age"] > 30) & (df2["City"] == 'Paris')]
```

6]:

	Name	Age	City	Salary
1	Anna	34	Paris	70000

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

high_salary = df[df['Salary'] > 50000]
print('Employees with salary > 50000')
print(high_salary)

#filtering rows salary > 50k & age > 30
filtered = df[(df['Age'] > 30) & (df['Salary'] > 50000)]
print(f'Employee list Age > 30 + Salary > 50000')
print(filtered)

#using OR condition
filtered_or = df[(df['Age'] > 35) | (df["Performance_Score"] > 90)]
print('Employees older than 35 OR performance score > 90')
print(filtered_or)
```

```
Employees with salary > 50000
      Name  Age  Salary  Performance_Score
1    Shyam   34   60000                 90
3  Dhanshyam   30   52000                 92
5   Jagdish   40   70000                 95
7    Simran   32   58000                 89
Employee list Age > 30 + Salary > 50000
      Name  Age  Salary  Performance_Score
1    Shyam   34   60000                 90
5  Jagdish   40   70000                 95
7   Simran   32   58000                 89
Employees older than 35 OR performance score > 90
      Name  Age  Salary  Performance_Score
3  Dhanshyam   30   52000                 92
5   Jagdish   40   70000                 95
sagarchouksey@MacBook-Air Pandas %
```

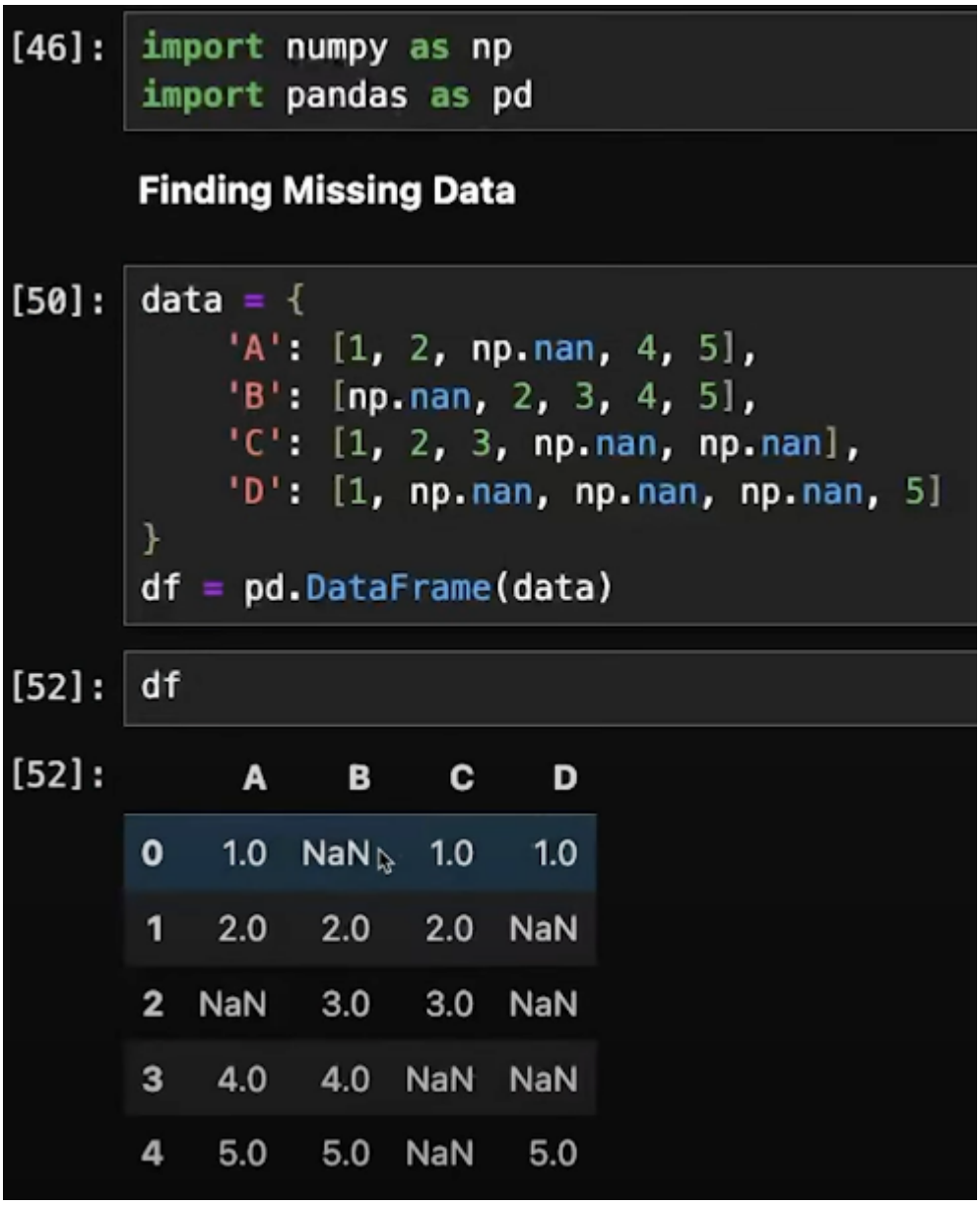
Missing Data

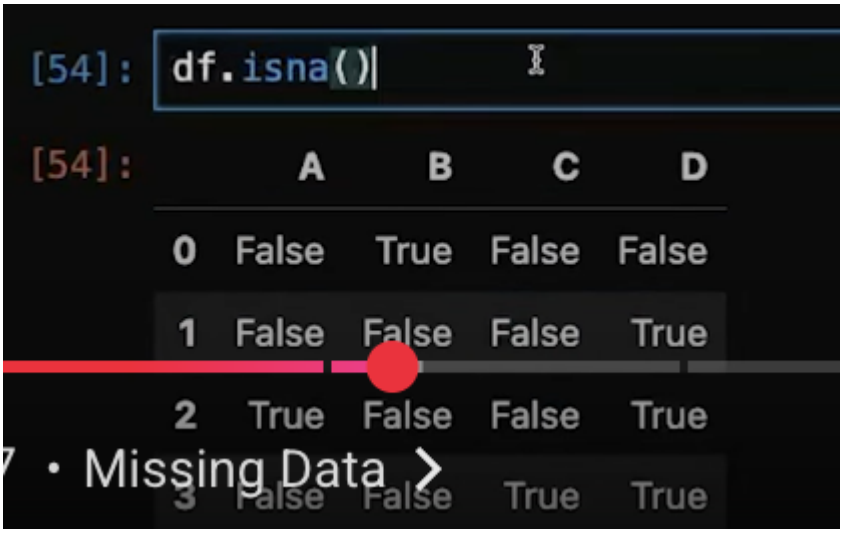
- nan value fill করার জন্য fillna()
- nan value drop করার জন্য dropna()
- drop করার জন্য আমরা যদি check করতে চাই এত টা nan value আছে কি না তাহলে thresh দিব
- nan value check করার জন্য isna()

```
NaN (Not a number)
None(for object data types)

isnull()
True - NaN is missing
False - value is  present
```

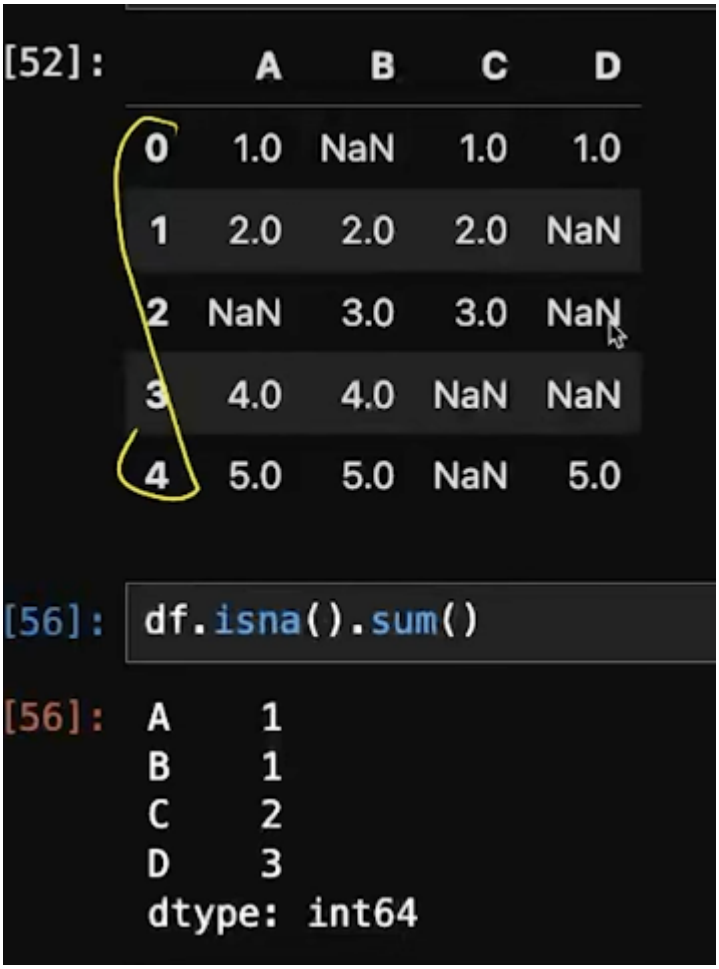
```
data = {
    "Name":['Ram',None,'Ghanshyam','Dhanshyam','Aditi','Jagdish','Raj','Simran'],
    "Age":[28,None,22,30,29,40,25,32],
    "Salary":[50000,None,45000,52000,49000,70000,48000,58000],
    "Performance_Score":[85,None,78,92,88,95,80,89]
}
```

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

print(df.isnull().sum())
```

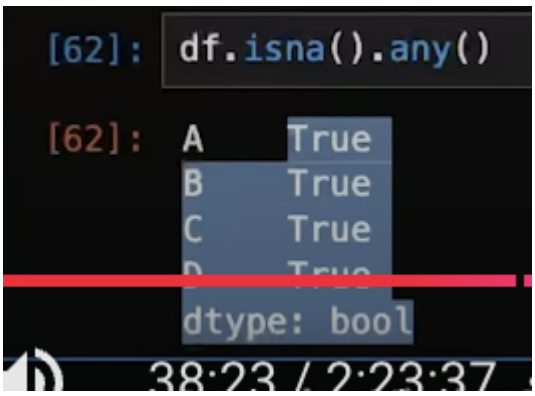

[46]:
import numpy as np
import pandas as pd
Finding Missing Data
[50]:
data = {
'A': [1, 2, np.nan, 4, 5],
'B': [np.nan, 2, 3, 4, 5],
'C': [1, 2, 3, np.nan, np.nan],
'D': [1, np.nan, np.nan, np.nan, 5]
}
df = pd.DataFrame(data)
[52]:
df
[52]:
A B C D
0 1.0 NaN 1.0 1.0
1 2.0 2.0 2.0 NaN
2 NaN 3.0 3.0 NaN
3 4.0 4.0 NaN NaN
4 5.0 5.0 NaN 5.0


[54]:
df.isna()
[54]:
A B C D
0 False True False False
1 False False False True
2 True False False True
Missing Data

.sum() দিয়ে জানতে পারি কোন কলামে সবচেয়ে বেশি nan value আছে।



.isna().any() দিয়ে দেখতে পারি আমাদের কি সব কলামেই nan value আছে নাকি কোন একটা কলাম এ নাই nan value.



dropna() function

শুধু যে row তে NAN ভ্যালু থাকবে না সেই row ছাড়া বাকি সব সব row গায়বে।

```
[74]: df

[74]:
     A  B    C    D
0  1.0  1  1.0  1.0
1  2.0  2  2.0  NaN
2  NaN  3  3.0  NaN
3  4.0  4  NaN  NaN
4  5.0  5  NaN  5.0

[76]: df.dropna()

[76]:
     A  B    C    D
0  1.0  1  1.0  1.0
```

Thresh = 3 দেওয়ার কারণে check করবে একটা row তে কমপক্ষে ৩ টা not null value আছে কি না। যদি থাকে

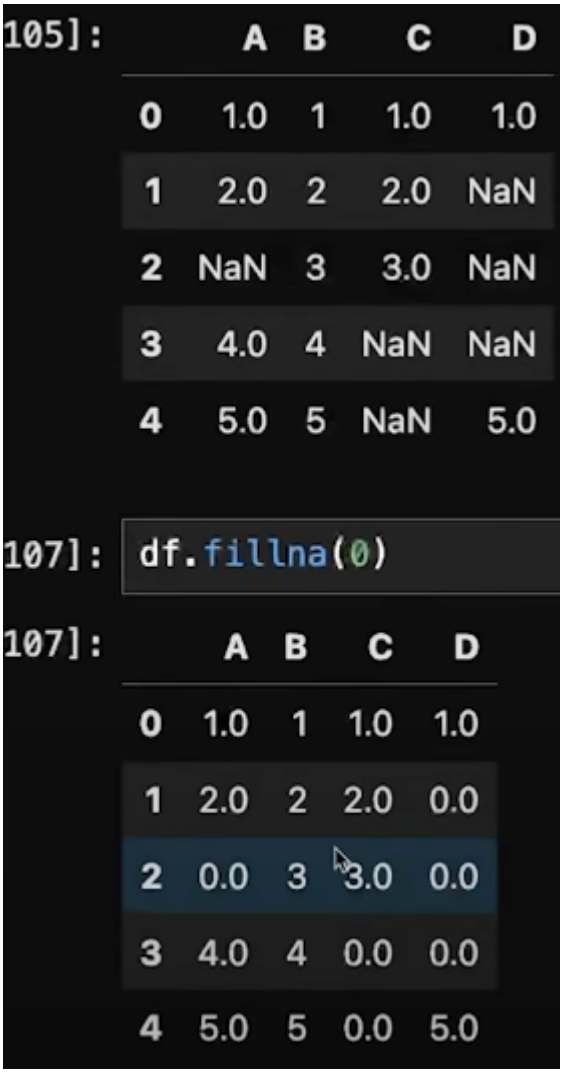
```
[95]: df

[95]:
     A  B    C    D
0  1.0  1  1.0  1.0
1  2.0  2  2.0  NaN
2  NaN  3  3.0  NaN
3  4.0  4  NaN  NaN
4  5.0  5  NaN  5.0

[97]: df.dropna(thresh=4)

[97]:
     A  B    C    D
0  1.0  1  1.0  1.0
```

• Missing Data

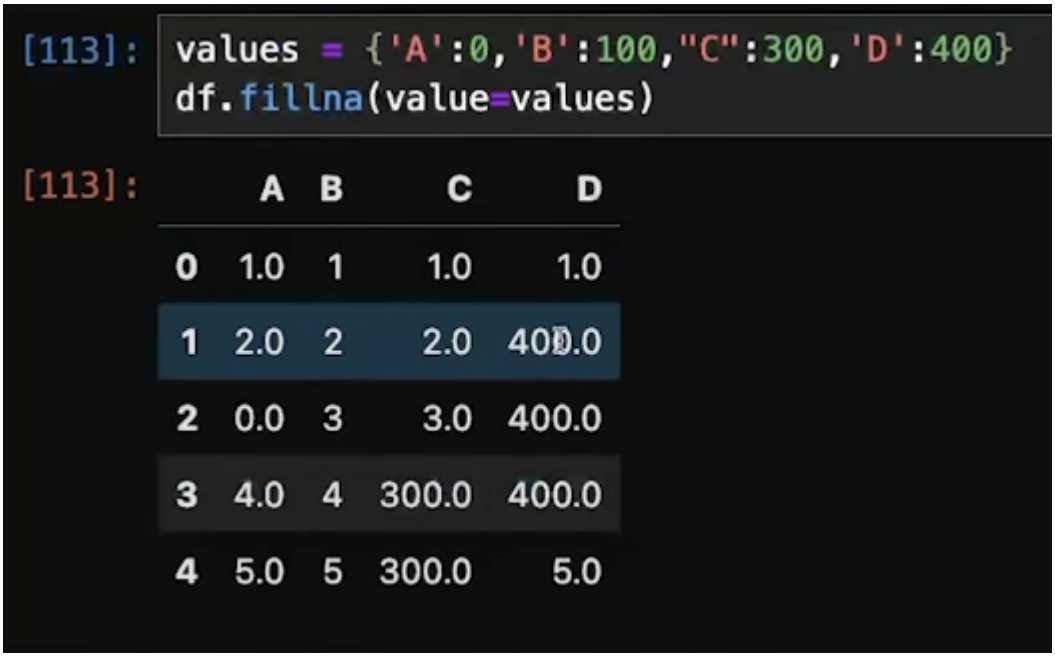
তাহলে সেগুলো শুধু দিন।

Filling The messing Data :

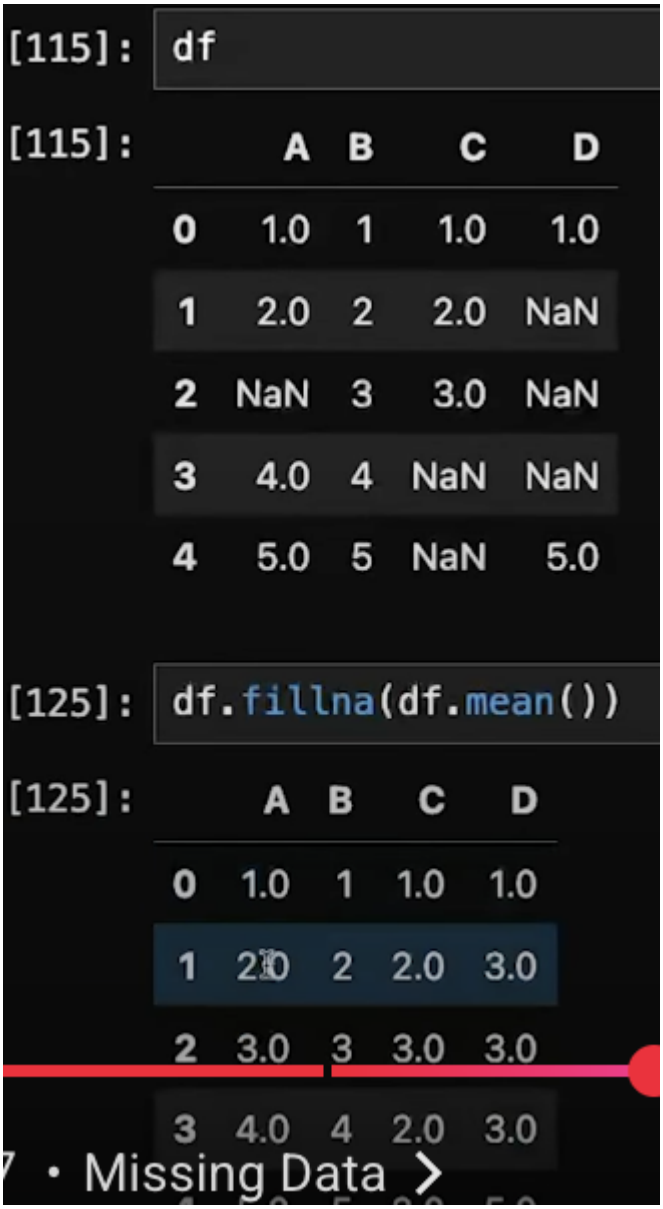
fillna method এর ভিতরে আমরা যে ভ্যালু দিব সে ভ্যালু দিয়ে এখানে যেগুলো nan value আছে সেগুলো 0 দিয়ে কনভার্ট হয়ে যাক।



যে কলামে nan value থাকবে সেগুলো তে values dictionary ভ্যালু অনুযায়ী পরিবর্তন হবে।



যে কলামে NaN আছে সেগুলো তে পুরো টেবিলের mean value দিয়ে ফিল আপ হয়ে যাবে।



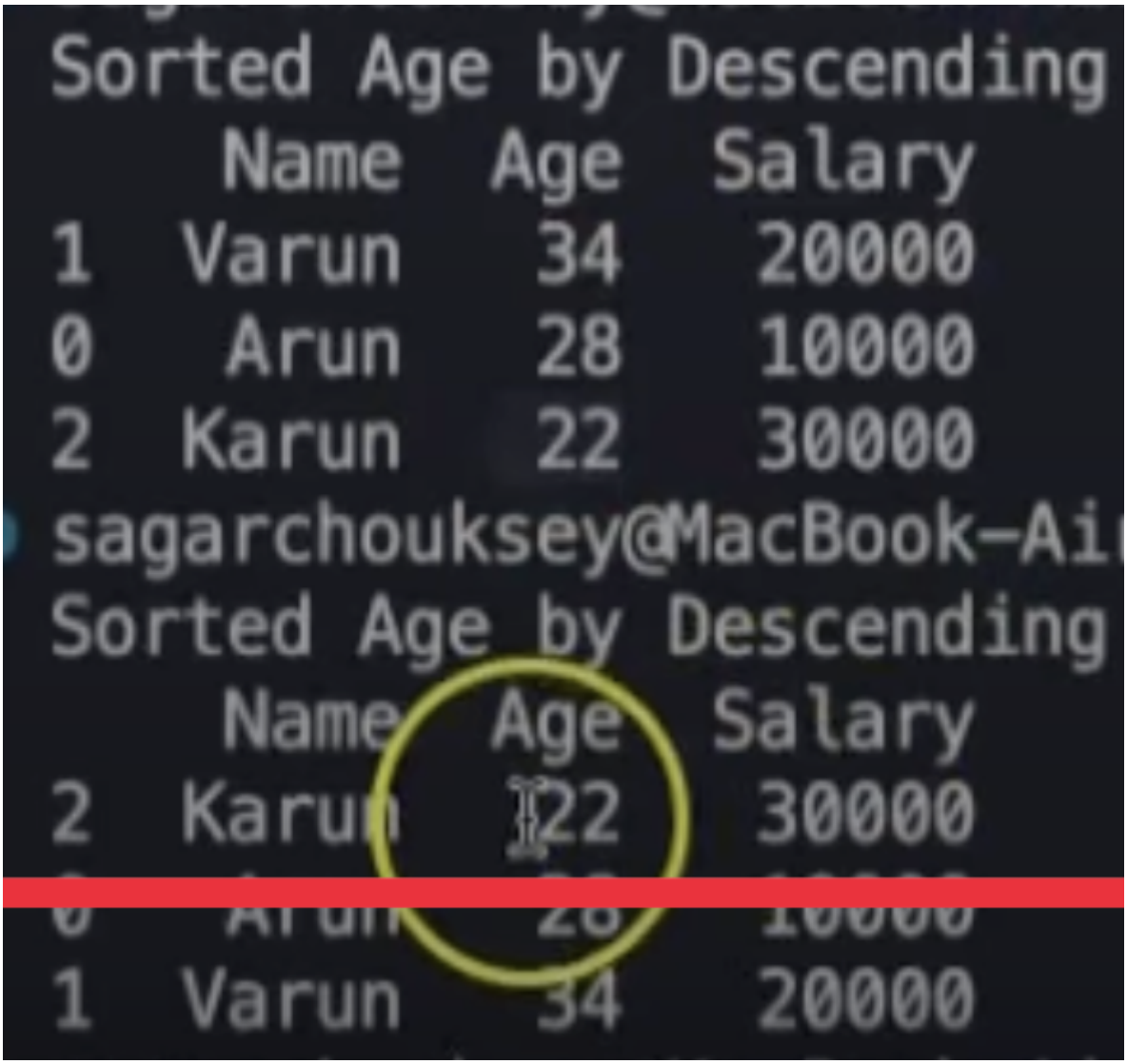
Handling Missing Data : interpolation method()

Sorting Data

True দিয়ে বুজায় ascending order এ সাজবে False দিয়ে বুড়াচ্ছে descending order এ সাজবে।

```
df.sort(by= columnName , ascending = True / False , inplace = True)
```

```python
#sorting data
#SORTING DATA 1 COLUMN sort_values()
#df.sort_values(by="Column Name", True/False, inplace = True)

import pandas as pd

data = {
    "Name":['Arun','Varun','Karun'],
    "Age":[28,34,22],
    "Salary":[10000,20000,30000]
}

df = pd.DataFrame(data)
df.sort_values(by="Age", ascending=True, inplace=True)
print('Sorted Age by Descending')
print(df)
```


Sorted Age by Descending
Name Age Salary
1 Varun 34 20000
0 Arun 28 10000
2 Karun 22 30000
sagarchouksey@MacBook-Ai
Sorted Age by Descending
Name Age Salary
2 Karun 22 30000
1 Varun 34 20000

```python
import pandas as pd

data = {
    "Name":['Arun','Varun','Karun'],
    "Age":[28,34,22],
    "Salary":[10000,20000,30000]
}

df = pd.DataFrame(data)
df.sort_values(by=["Age", "Salary"], ascending=False, inplace=True)
print('Sorted Age by Descending')
print(df)
```

```
Sorted Age by Descending
    Name  Age  Salary
1  Varun   34   20000
0   Arun   28   10000
2  Karun   22   30000
```

```python
df = pd.DataFrame(data)
df.sort_values(by=["Age", "Salary"], ascending=[True, False], inplace=Tr
print('Sorted Age by Descending')
print(df)
```

Merging , Joining And Concatenation 2 dataframe :

[32]:
```
import numpy as np
import pandas as pd
```

Merging 2 dataframes

[34]:
```
employees = pd.DataFrame({
    'employee_id': [1, 2, 3, 4, 5],
    'name': ['John', 'Anna', 'Peter', 'Linda', 'Bob'],
    'department': ['HR', 'IT', 'Finance', 'IT', 'HR']
})

# DataFrame 2: Salary information
salaries = pd.DataFrame({
    'employee_id': [1, 2, 3, 6, 7],
    'salary': [60000, 80000, 65000, 70000, 90000],
    'bonus': [5000, 10000, 7000, 8000, 12000]
})
```

[36]:
```
employees
```

[36]:

	employee_id	name	department
0	1	John	HR
1	2	Anna	IT
2	3	Peter	Finance
3	4	Linda	IT
4	5	Bob	HR

[38]:
```
salaries
```

[38]:

	employee_id	salary	bonus
0	1	60000	5000
1	2	80000	10000
2	3	65000	7000
3	6	70000	8000
4	7	90000	12000

Two dataframe [redacted]

on= ‘employe_id’ মানে তাদের মধ্যে কমন কলাম হলো employee_id এর basis এ টেবিল দুইটা merge হয়েছে। এখানে ভালো করে খেয়াল করলে দেখা যাবে employee_id যেগুলো আছে সেগুলোই শুধু add হয়েছে।

- How = inner মনে employee_id কলামে যেগুলো মিলে সেগুলো ADD করবে। মানে যাদের keys সেম।
- How = outer মানে শুধু employee_id কলামের উপর ভিত্তি করে হবে। আর যেগুলো নাই সেখানে ভ্যালু NAN আসবে। সবগুলো কি নিয়ে নিবে আর যেগুলো মিলবেনা সেগুলো nan value দিয়ে ফিল করে দিবে।
- How = left মানে লেফট employee table এর basis এ add করবে। লেফট সাইডের টা রাখবে
- How = right মানে salary table এর basis এ add করবে।

```
[48]: pd.merge(employees,salaries,on='employee_id',how ='inner')
```

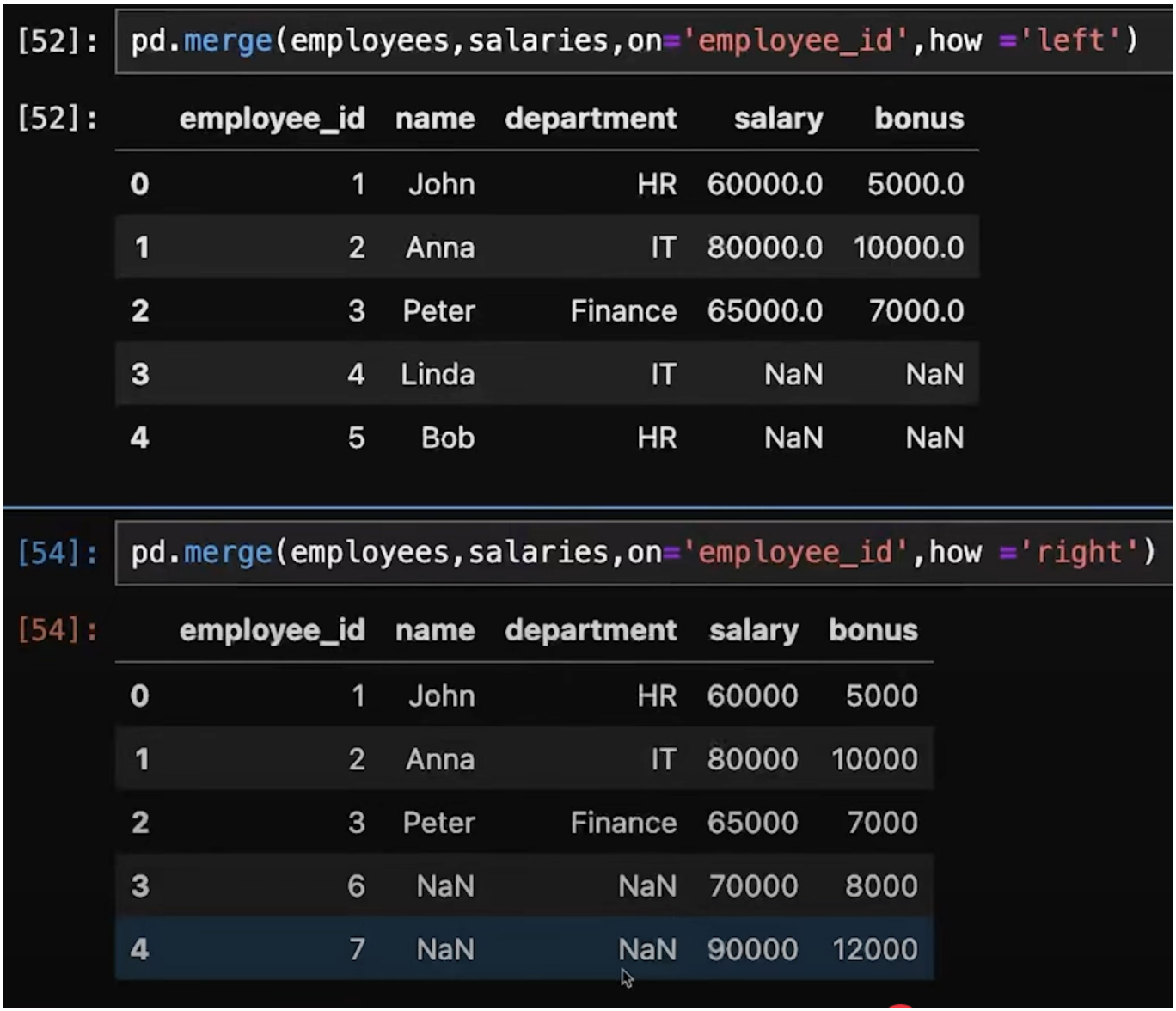
[48]:

	employee_id	name	department	salary	bonus
0	1	John	HR	60000	5000
1	2	Anna	IT	80000	10000
2	3	Peter	Finance	65000	7000

```
[50]: pd.merge(employees,salaries,on='employee_id',how ='outer')
```

[50]:

	employee_id	name	department	salary	bonus
0	1	John	HR	60000.0	5000.0
1	2	Anna	IT	80000.0	10000.0
2	3	Peter	Finance	65000.0	7000.0
3	4	Linda	IT	NaN	NaN
4	5	Bob	HR	NaN	NaN
5	6	NaN	NaN	70000.0	8000.0
6	7	NaN	NaN	90000.0	12000.0



```
[52]: pd.merge(employees,salaries,on='employee_id',how ='left')
```

[52]:

	employee_id	name	department	salary	bonus
0	1	John	HR	60000.0	5000.0
1	2	Anna	IT	80000.0	10000.0
2	3	Peter	Finance	65000.0	7000.0
3	4	Linda	IT	NaN	NaN
4	5	Bob	HR	NaN	NaN

```
[54]: pd.merge(employees,salaries,on='employee_id',how ='right')
```

[54]:

	employee_id	name	department	salary	bonus
0	1	John	HR	60000	5000
1	2	Anna	IT	80000	10000
2	3	Peter	Finance	65000	7000
3	6	NaN	NaN	70000	8000
4	7	NaN	NaN	90000	12000

concatenation

Concatanation হয়তো vertically নয়ত horizontally add করবে।

যদি আমরা পুরো dataframe কে concatenation করতে চাই তাহলে concat function ব্যবহার করব।

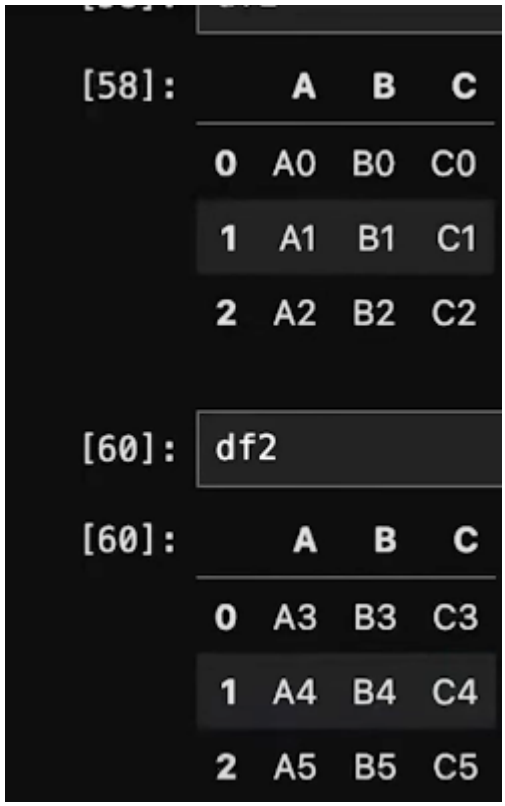
আর যদি আমরা dataframe কে শুধু একটা column এর ভিত্তিতে করতে চাই তাহলে merge করব।

আমরা যখন concat করব আর যখন axis দিব না তখন Default প্রথম dataset পর দ্বিতীয় ডাটাসেট যুক্ত হবে।

আর যখন axis = 1 দিব তখন কলম wise add হবে।

```
vertically (row-wise)
horizontally (column wise)

pd.concate([df1, df2], axis=0, ignore_index=True)

[df1, df2] =
axis = 1

ignore_index = True
```

```
#region1
df_Region1 = pd.DataFrame({
    'CustomerID':[1,2],
    'Name':['Gopal','Raju']
})

#region2
df_Region2 = pd.DataFrame({
    'CustomerID':[3,4],
    'Name':['Shyam','Baburao']
})

#concatenate vertically
df_concat = pd.concat([df_Region1, df_Region2], ignore_index=True)
print(df_concat)
```



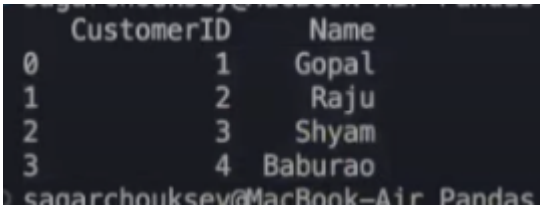
[58]:

	A	B	C
0	A0	B0	C0
1	A1	B1	C1
2	A2	B2	C2

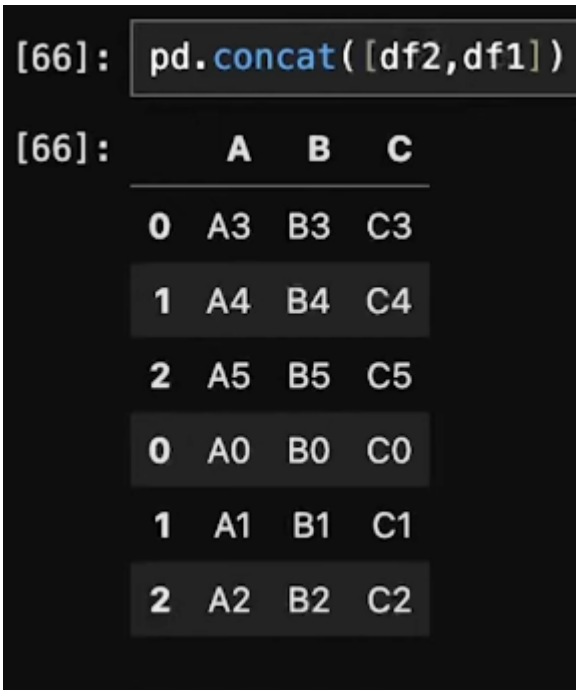
[60]: df2

[60]:

	A	B	C
0	A3	B3	C3
1	A4	B4	C4
2	A5	B5	C5



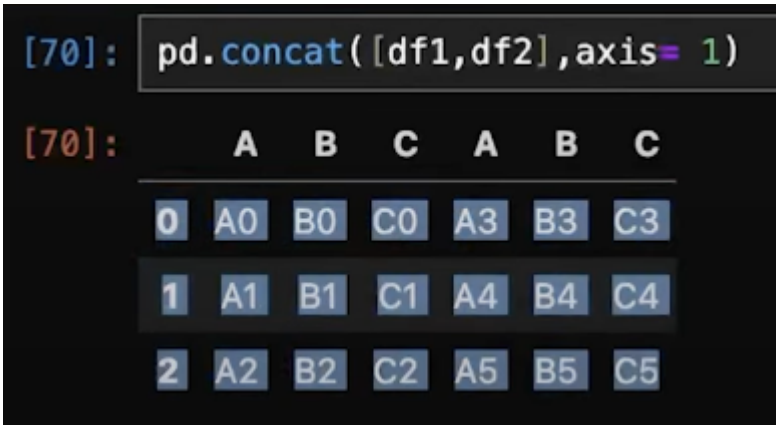
	CustomerID	Name
0	1	Gopal
1	2	Raju
2	3	Shyam
3	4	Baburao



[66]: pd.concat([df2,df1])

[66]:

	A	B	C
0	A3	B3	C3
1	A4	B4	C4
2	A5	B5	C5
0	A0	B0	C0
1	A1	B1	C1
2	A2	B2	C2



[70]: pd.concat([df1,df2],axis= 1)

[70]:

	A	B	C	A	B	C
0	A0	B0	C0	A3	B3	C3
1	A1	B1	C1	A4	B4	C4
2	A2	B2	C2	A5	B5	C5

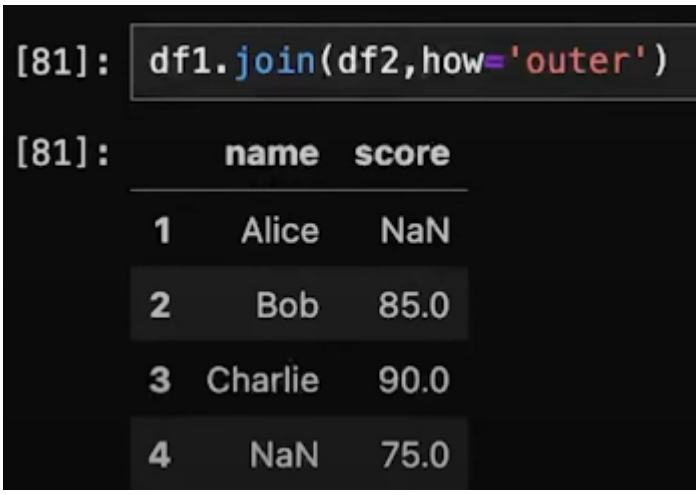
Joining

```
[73]: df1 = pd.DataFrame({
          'name': ['Alice', 'Bob', 'Charlie']
      }, index=[1, 2, 3])

      # Second DataFrame
      df2 = pd.DataFrame({
          'score': [85, 90, 75]
      }, index=[2, 3, 4])
```

```
[75]: df1

[75]:      name
      1    Alice
      2      Bob
      3  Charlie
```

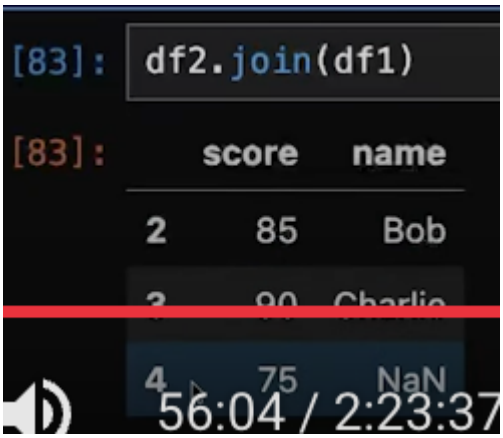
```
[77]: df2

[77]:    score
      2     85
      3     90
      4     75
```

```
[79]: df1.join(df2)

[79]:      name  score
      1    Alice    NaN
      2      Bob   85.0
      3  Charlie   90.0
```



```
[81]: df1.join(df2,how='outer')

[81]:
```

	name	score
1	Alice	NaN
2	Bob	85.0
3	Charlie	90.0
4	NaN	75.0

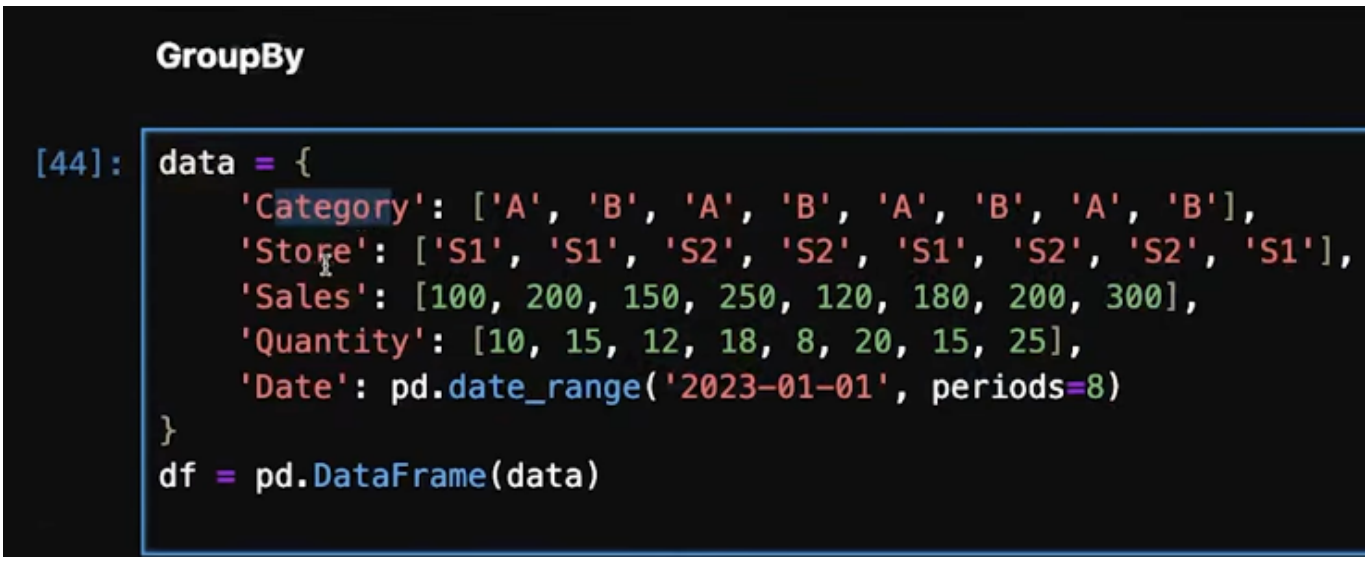


```
[83]: df2.join(df1)

[83]:
```

	score	name
2	85	Bob
3	90	Charlie
4	75	NaN

56:04 / 2:23:37

GroupBy And Aggregation:



GroupBy

```
[44]: data = {
          'Category': ['A', 'B', 'A', 'B', 'A', 'B', 'A', 'B'],
          'Store': ['S1', 'S1', 'S2', 'S2', 'S1', 'S2', 'S2', 'S1'],
          'Sales': [100, 200, 150, 250, 120, 180, 200, 300],
          'Quantity': [10, 15, 12, 18, 8, 20, 15, 25],
          'Date': pd.date_range('2023-01-01', periods=8)
      }
      df = pd.DataFrame(data)
```

```
[46]: df
```

[46]:

	Category	Store	Sales	Quantity	Date
0	A	S1	100	10	2023-01-01
1	B	S1	200	15	2023-01-02
2	A	S2	150	12	2023-01-03
3	B	S2	250	18	2023-01-04
4	A	S1	120	8	2023-01-05
5	B	S2	180	20	2023-01-06
6	A	S2	200	15	2023-01-07
7	B	S1	300	25	2023-01-08

আমরা যখন groupby করি তখন এখানে dataset ক্রিয়েট হয় না। এই টা ক্রিয়েট হয় অবজেক্ট। অবজেক্ট আমরা দেখতে পারব না যদি দেখতে চাই তাহলে আমরা ফর লুপ চালিয়ে দেখতে পারি। এখানে ক্যাটাগরি ভিত্তিতে দুইটা দেখাচ্ছে।

```
[60]: # Group by Category and calculate the sum of Sales
cat = df.groupby('Category')
cat
for i, v in cat:
    print(i)
    print(v)
```

```
A
  Category Store  Sales  Quantity       Date
0        A    S1    100        10 2023-01-01
2        A    S2    150        12 2023-01-03
4        A    S1    120         8 2023-01-05
6        A    S2    200        15 2023-01-07
B
  Category Store  Sales  Quantity       Date
1        B    S1    200        15 2023-01-02
3        B    S2    250        18 2023-01-04
5        B    S2    180        20 2023-01-06
7        B    S1    300        25 2023-01-08
```

```
[64]: # Group by Category and calculate the sum of Sales
      cat = df.groupby('Category')['Sales'].sum()
      cat

[64]: Category
      A    570
      B    930
      Name: Sales, dtype: int64
```

```
[68]: # Group by Store and calculate the sum of Sales
      cat = df.groupby('Store')['Sales'].sum()
      cat

[68]: Store
      S1    720
      S2    780
      Name: Sales, dtype: int64
```

Qualitative : category Quantitive : continuous

```
[70]: # Group by multiple columns
      # Group by Category and Store
      cat = df.groupby(['Category','Store'])['Sales'].sum()
      cat

[70]: Category  Store
      A         S1       220
                S2       350
      B         S1       500
                S2       430
      Name: Sales, dtype: int64
```

Aggregation Funciton :

Aggregation

```
[72]: df['Sales'].mean()
      """mean median min max count std """
```

```
[72]: 187.5
```

```
[82]: df['Sales'].agg(['sum', 'mean', 'min', 'max', 'count', 'std', 'median'])
```

```
[82]: sum       1500.000000
      mean       187.500000
      min        100.000000
      max        300.000000
      count        8.000000
      std         66.062741
      median     190.000000
      Name: Sales, dtype: float64
```

```
[ ]:
```

pivot table

Code

```
    'Rep': ['John', 'Mary', 'Bob', 'Alice', 'John', 'Mary', 'Bob', 'Alice', 'John', 'Mary',
            'Bob', 'Alice', 'John', 'Mary', 'Bob', 'Alice', 'John', 'Mary', 'Bob', 'Alice']
}

df = pd.DataFrame(data)

df['Month'] = df['Date'].dt.month_name()
df['Quarter'] = 'Q' + df['Date'].dt.quarter.astype(str)
df
```

	Date	Product	Region	Sales	Units	Rep	Month	Quarter
0	2023-01-01	A	East	825	26	John	January	Q1
1	2023-01-02	B	West	611	51	Mary	January	Q1
2	2023-01-03	C	North	768	11	Bob	January	Q1
3	2023-01-04	D	South	665	71	Alice	January	Q1
4	2023-01-05	A	East	157	20	John	January	Q1
5	2023-01-06	B	West	436	43	Mary	January	Q1
6	2023-01-07	C	North	227	39	Bob	January	Q1
7	2023-01-08	D	South	783	35	Alice	January	Q1
8	2023-01-09	A	East	352	54	John	January	Q1
9	2023-01-10	B	West	740	31	Mary	January	Q1
10	2023-01-11	C	North	944	31	Bob	January	Q1

```
[37]: pd.pivot_table(df,values = "Sales",index = 'Region',columns="Product")
```

[37]:

Region	A	B	C	D
East	474.2	NaN	NaN	NaN
North	NaN	NaN	617.0	NaN
South	NaN	NaN	NaN	485.0
West	NaN	527.2	NaN	NaN

Region east এর A score এর সব গুলো ভ্যালুর mean যাতে বের হয়।

```
39]: pd.pivot_table(df,values = "Sales",index = 'Region',columns="Product",aggfunc = 'median')
```

39]:

Region	A	B	C	D
East	415.0	NaN	NaN	NaN
North	NaN	NaN	596.0	NaN
South	NaN	NaN	NaN	626.0
West	NaN	535.0	NaN	NaN

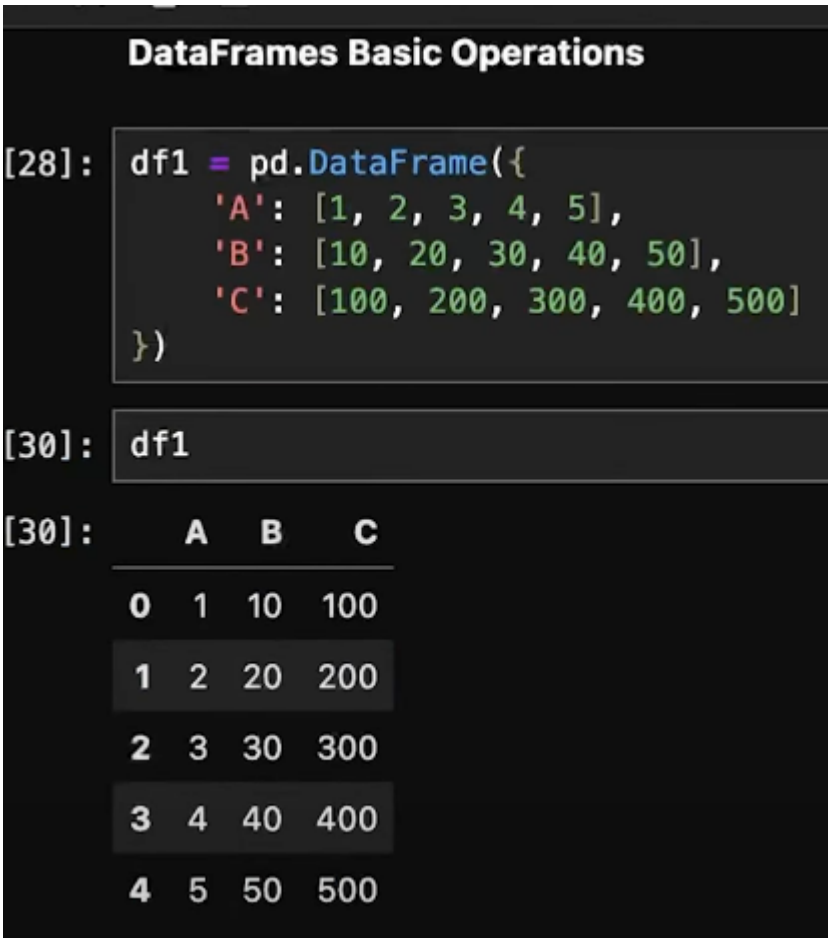
```
[41]: pivot2 = pd.pivot_table(df, values=['', 'Units'], index='Region', columns='Product')
      pivot2
```

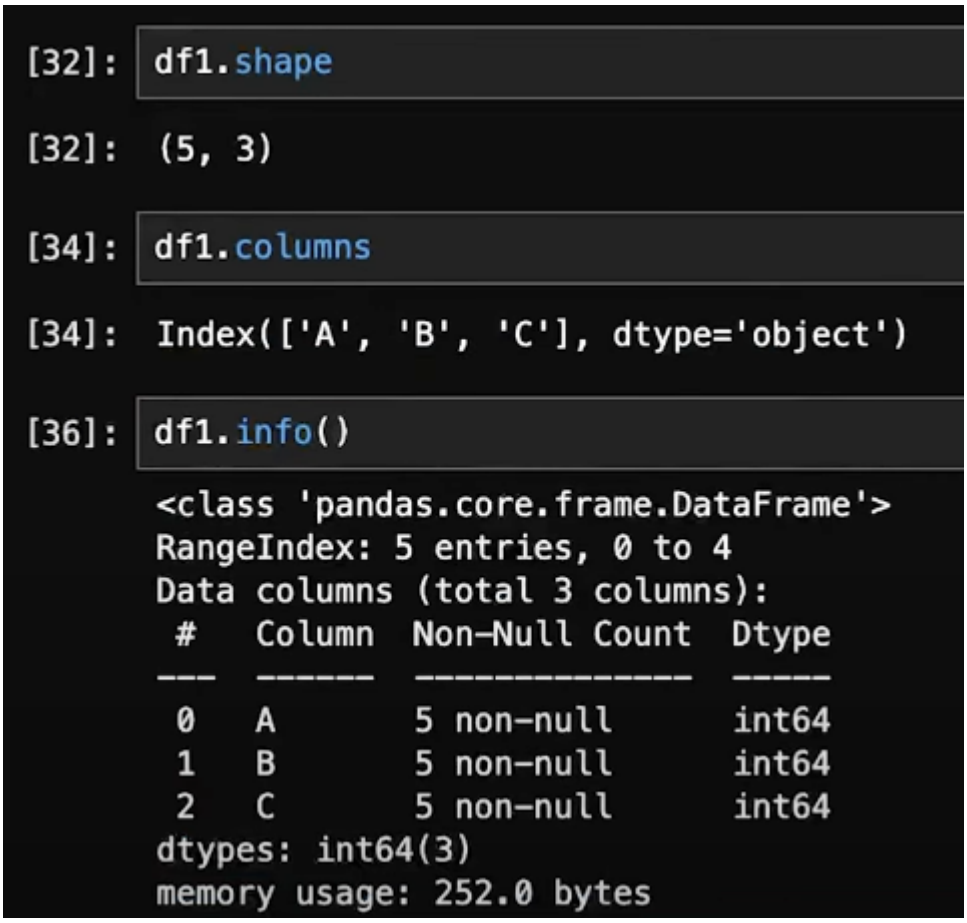
[41]:

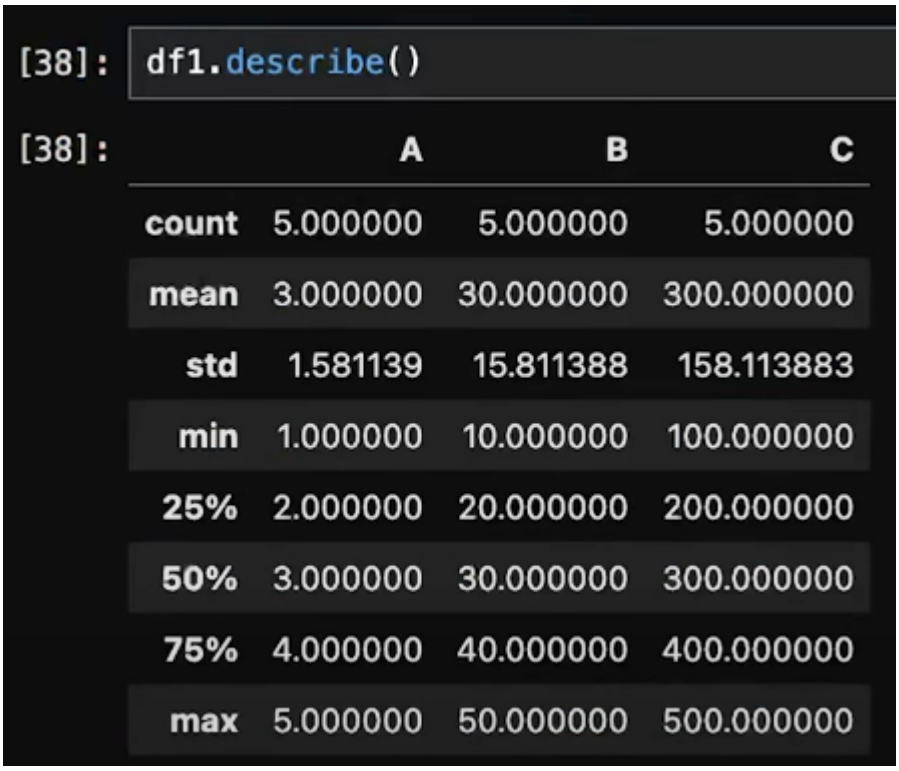
	Sales				Units			
Product	A	B	C	D	A	B	C	D
Region								
East	474.2	NaN	NaN	NaN	38.6	NaN	NaN	NaN
North	NaN	NaN	617.0	NaN	NaN	NaN	42.0	NaN
South	NaN	NaN	NaN	485.0	NaN	NaN	NaN	64.4
West	NaN	527.2	NaN	NaN	NaN	63.6	NaN	NaN

ভ্যালু তে আমরা must be numeric value দিতে হবে।

Basic Operation


DataFrames Basic Operations
[28]:
df1 = pd.DataFrame({
'A': [1, 2, 3, 4, 5],
'B': [10, 20, 30, 40, 50],
'C': [100, 200, 300, 400, 500]
})
[30]:
df1
[30]:
A B C
0 1 10 100
1 2 20 200
2 3 30 300
3 4 40 400
4 5 50 500

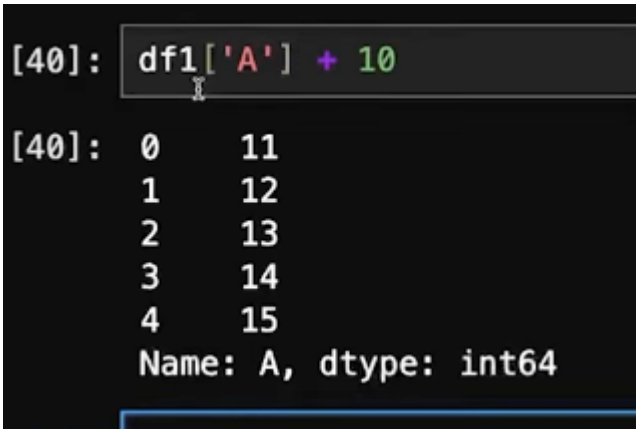

[32]:
df1.shape
[32]:
(5, 3)
[34]:
df1.columns
[34]:
Index(['A', 'B', 'C'], dtype='object')
[36]:
df1.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
Column Non-Null Count Dtype
--- ------ -------------- -----
0 A 5 non-null int64
1 B 5 non-null int64
2 C 5 non-null int64
dtypes: int64(3)
memory usage: 252.0 bytes



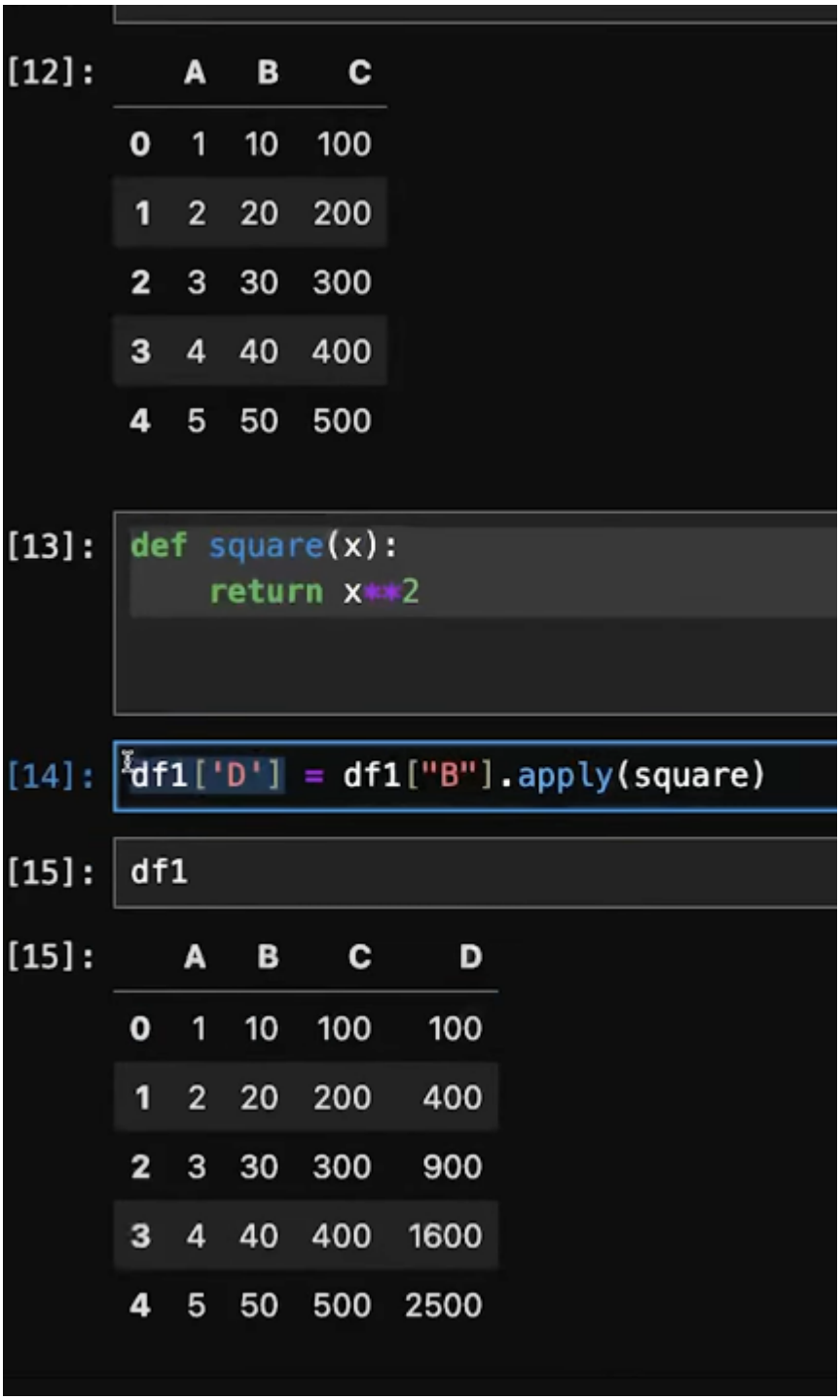
```
[38]: df1.describe()
```

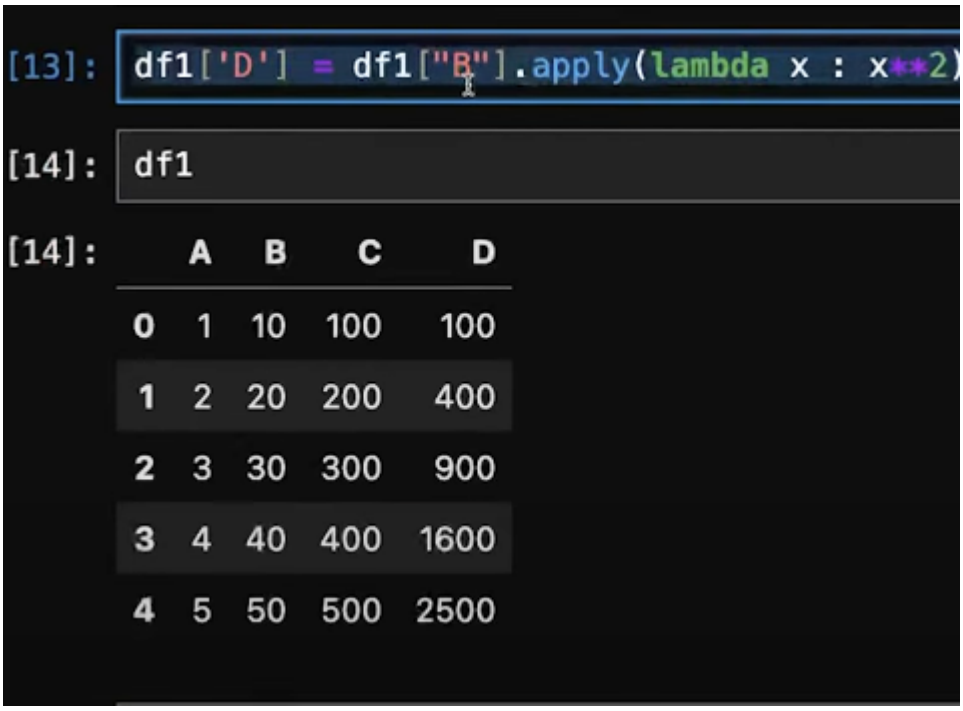
[38]:

	A	B	C
count	5.000000	5.000000	5.000000
mean	3.000000	30.000000	300.000000
std	1.581139	15.811388	158.113883
min	1.000000	10.000000	100.000000
25%	2.000000	20.000000	200.000000
50%	3.000000	30.000000	300.000000
75%	4.000000	40.000000	400.000000
max	5.000000	50.000000	500.000000



```
[40]: df1['A'] + 10

[40]: 0    11
      1    12
      2    13
      3    14
      4    15
      Name: A, dtype: int64
```


[12]:
A B C
0 1 10 100
1 2 20 200
2 3 30 300
3 4 40 400
4 5 50 500
[13]:
def square(x):
return x**2
[14]:
df1['D'] = df1["B"].apply(square)
[15]:
df1
[15]:
A B C D
0 1 10 100 100
1 2 20 200 400
2 3 30 300 900
3 4 40 400 1600
4 5 50 500 2500


[13]: df1['D'] = df1["B"].apply(lambda x : x**2)
[14]: df1
[14]:
A B C D
0 1 10 100 100
1 2 20 200 400
2 3 30 300 900
3 4 40 400 1600
4 5 50 500 2500