Python Pandas - Advanced

Agenda

- Concatenate Series and Dataframe
- Join the Dataframes

Concatenate a Series

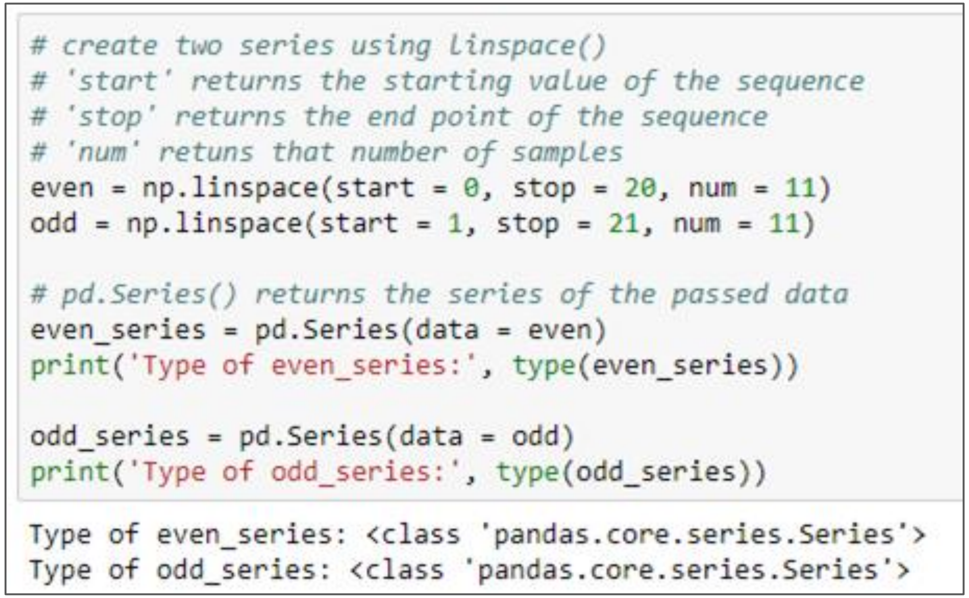
Concatenate a series

- Creates a new series by appending a series with another series
- The concat() and append() methods are used to concatenate a series

- One can concatenate more than two series

Create a series

Create python series as shown below:

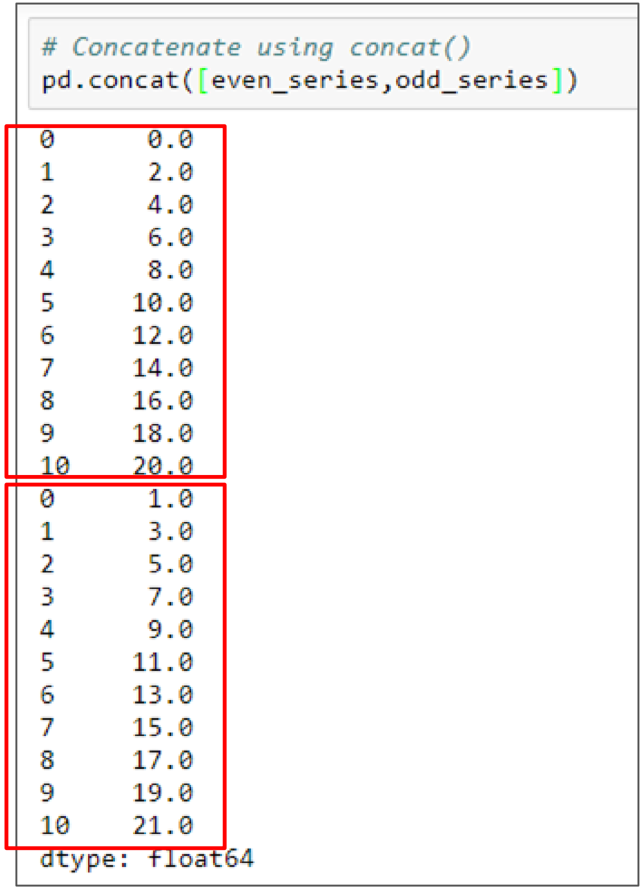


```
# create two series using linspace()
# 'start' returns the starting value of the sequence
# 'stop' returns the end point of the sequence
# 'num' retuns that number of samples
even = np.linspace(start = 0, stop = 20, num = 11)
odd = np.linspace(start = 1, stop = 21, num = 11)

# pd.Series() returns the series of the passed data
even_series = pd.Series(data = even)
print('Type of even_series:', type(even_series))

odd_series = pd.Series(data = odd)
print('Type of odd_series:', type(odd_series))
```

```
Type of even_series: <class 'pandas.core.series.Series'>
Type of odd_series: <class 'pandas.core.series.Series'>
```

Concatenate a series

The concat() method concatenates a series in the order they are passed in the function

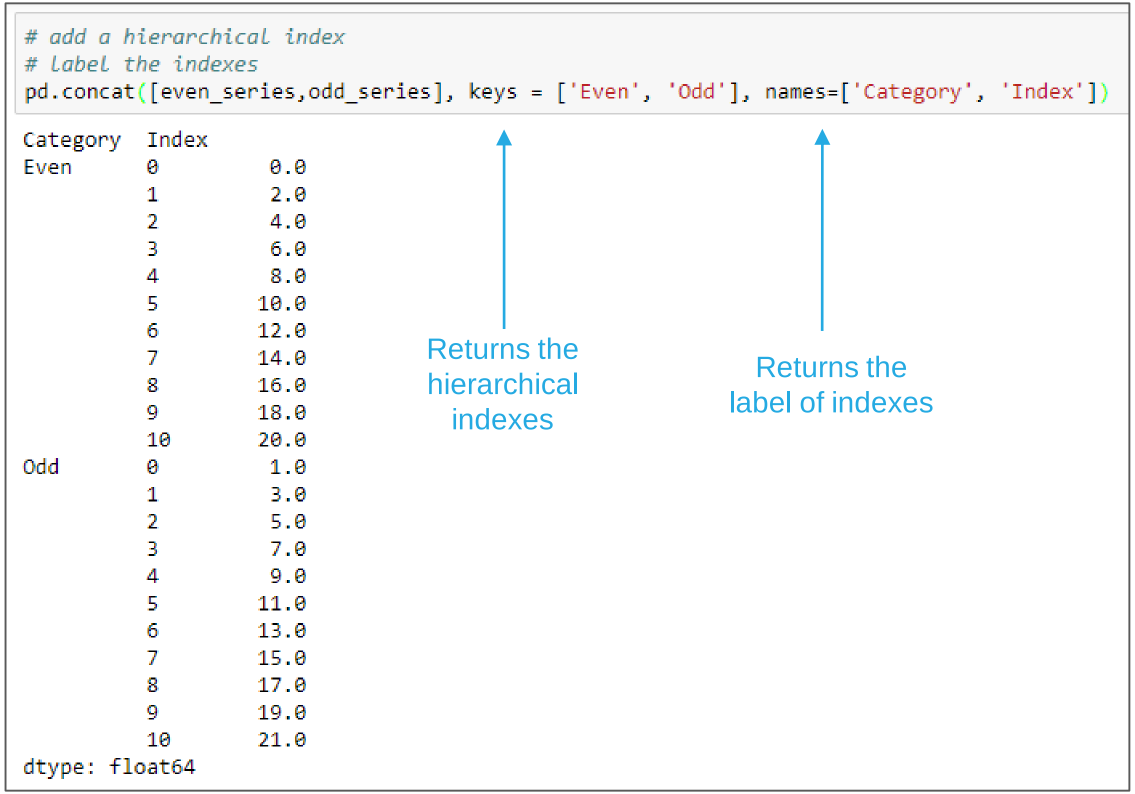


```
# Concatenate using concat()
pd.concat([even_series,odd_series])
```

```
0      0.0
1      2.0
2      4.0
3      6.0
4      8.0
5     10.0
6     12.0
7     14.0
8     16.0
9     18.0
10    20.0
0      1.0
1      3.0
2      5.0
3      7.0
4      9.0
5     11.0
6     13.0
7     15.0
8     17.0
9     19.0
10    21.0
dtype: float64
```

Add hierarchical index and label the index

Add the hierarchical indexes and labels while concatenating two series



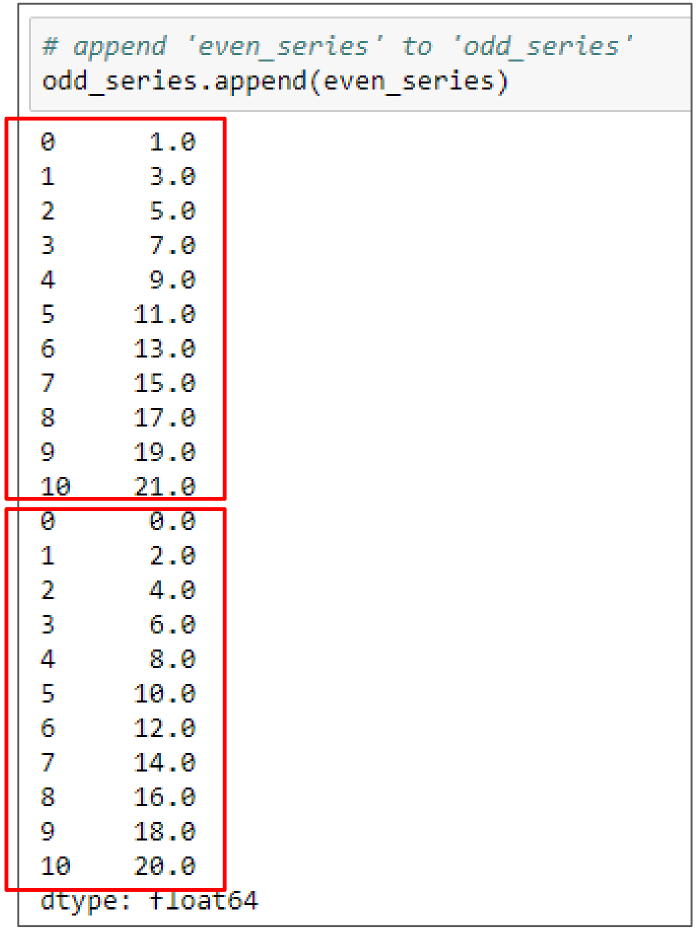
```
# add a hierarchical index
# label the indexes
pd.concat([even_series,odd_series], keys = ['Even', 'Odd'], names=['Category', 'Index'])
```

```
Category  Index
Even      0         0.0
          1         2.0
          2         4.0
          3         6.0
          4         8.0
          5        10.0
          6        12.0
          7        14.0
          8        16.0
          9        18.0
          10       20.0
Odd       0         1.0
          1         3.0
          2         5.0
          3         7.0
          4         9.0
          5        11.0
          6        13.0
          7        15.0
          8        17.0
          9        19.0
          10       21.0
dtype: float64
```

Returns the hierarchical indexes
Returns the label of indexes

Concatenate a series

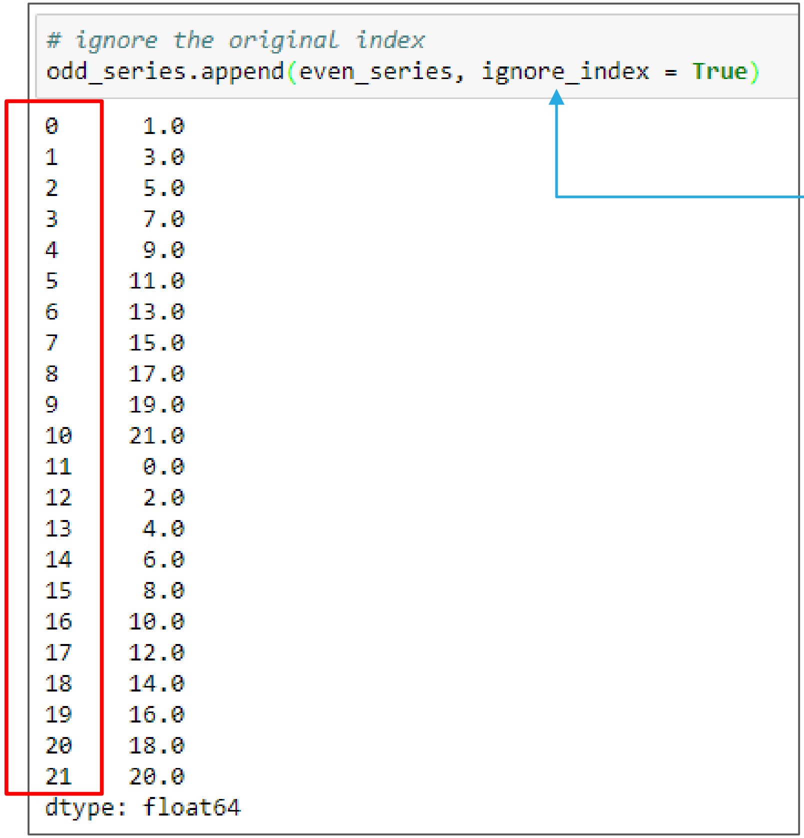
- The append() method is used append a series with another
- Here, we append the 'even_series' to 'odd_series'
- Appended indexes are same as the original series



```
# append 'even_series' to 'odd_series'
odd_series.append(even_series)
```

```
0       1.0
1       3.0
2       5.0
3       7.0
4       9.0
5      11.0
6      13.0
7      15.0
8      17.0
9      19.0
10     21.0
0       0.0
1       2.0
2       4.0
3       6.0
4       8.0
5      10.0
6      12.0
7      14.0
8      16.0
9      18.0
10     20.0
dtype: float64
```

Concatenate a series



```
# ignore the original index
odd_series.append(even_series, ignore_index = True)
```

```
0      1.0
1      3.0
2      5.0
3      7.0
4      9.0
5     11.0
6     13.0
7     15.0
8     17.0
9     19.0
10    21.0
11     0.0
12     2.0
13     4.0
14     6.0
15     8.0
16    10.0
17    12.0
18    14.0
19    16.0
20    18.0
21    20.0
dtype: float64
```

Ignores the index labels of original series

Concatenate the DataFrames

Concatenate the DataFrames

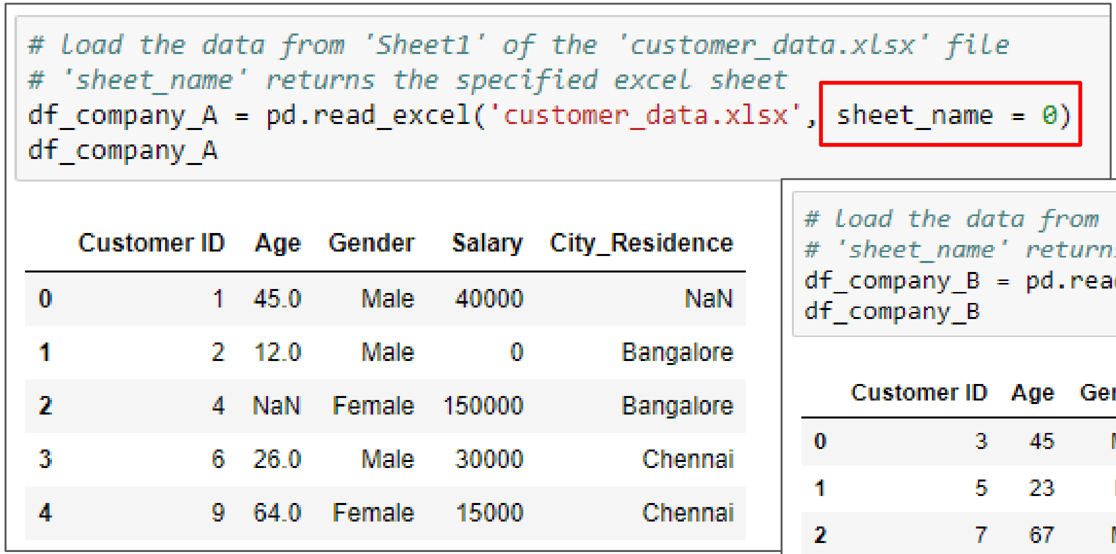
- A Pandas DataFrame is a two dimensional size-mutable, heterogeneous data structure with labeled rows and columns

- DataFrames can be concatenated vertically (column-wise) and horizontally (row-wise)

- The concat() and append() methods are used to concatenate the DataFrames

Read the DataFrames

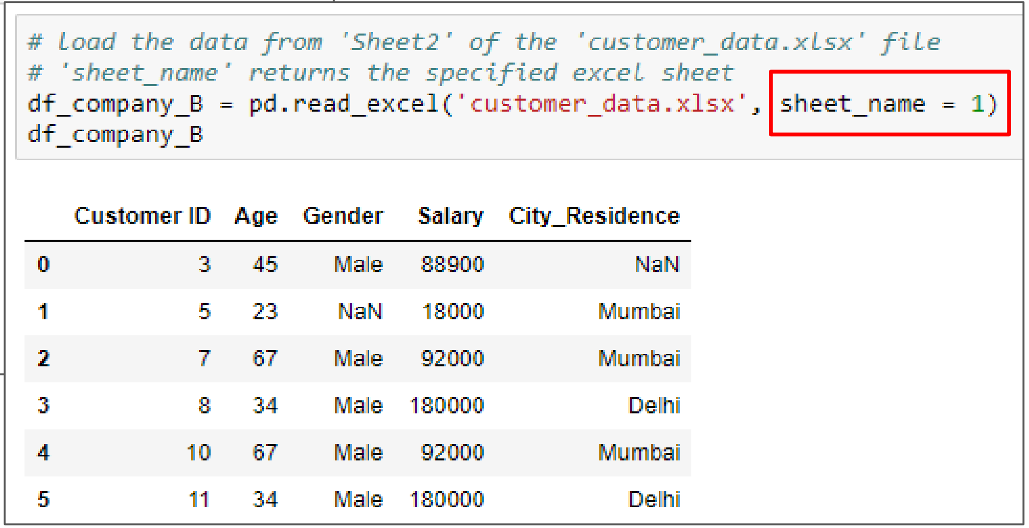
Use these DataFrames for further manipulations:



```
# Load the data from 'Sheet1' of the 'customer_data.xlsx' file
# 'sheet_name' returns the specified excel sheet
df_company_A = pd.read_excel('customer_data.xlsx', sheet_name = 0)
df_company_A
```

	Customer ID	Age	Gender	Salary	City_Residence
0	1	45.0	Male	40000	NaN
1	2	12.0	Male	0	Bangalore
2	4	NaN	Female	150000	Bangalore
3	6	26.0	Male	30000	Chennai
4	9	64.0	Female	15000	Chennai

Customer details of company A



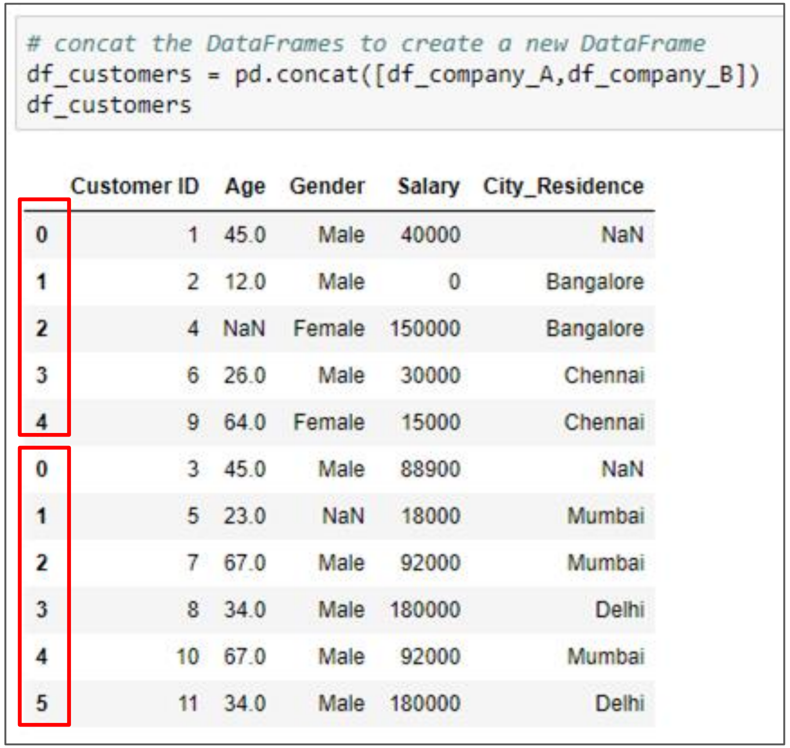
```
# Load the data from 'Sheet2' of the 'customer_data.xlsx' file
# 'sheet_name' returns the specified excel sheet
df_company_B = pd.read_excel('customer_data.xlsx', sheet_name = 1)
df_company_B
```

	Customer ID	Age	Gender	Salary	City_Residence
0	3	45	Male	88900	NaN
1	5	23	NaN	18000	Mumbai
2	7	67	Male	92000	Mumbai
3	8	34	Male	180000	Delhi
4	10	67	Male	92000	Mumbai
5	11	34	Male	180000	Delhi

Customer details of company B

Concatenate the DataFrames

- Two DataFrames are concatenated using concat() method
- By default, the concat() method concatenates along the axis = 0 (vertically)
- The concatenation is in the order they are passed in the function
- The index numbers of the concatenated DataFrame are of the actual DataFrames

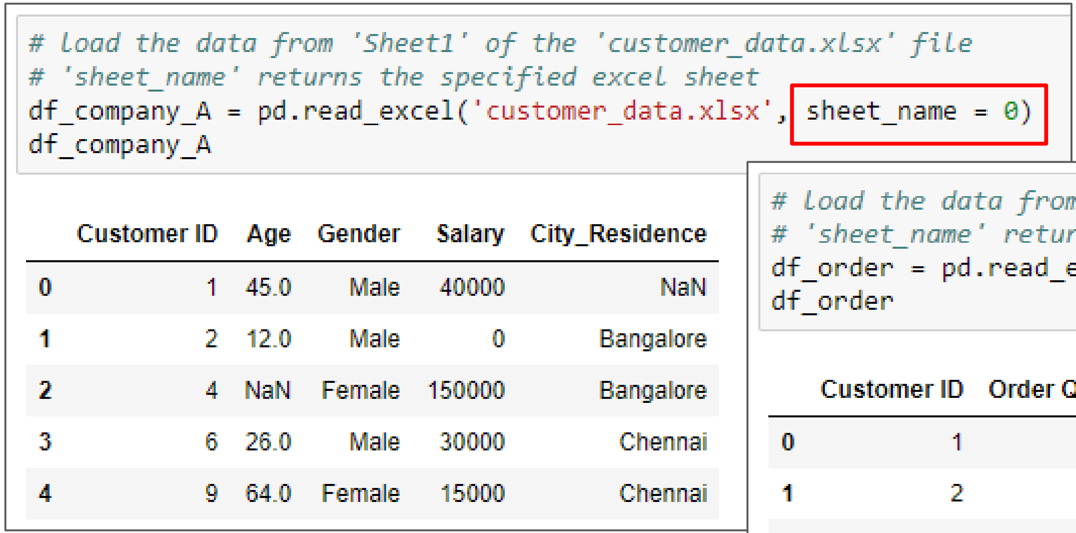


```
# concat the DataFrames to create a new DataFrame
df_customers = pd.concat([df_company_A,df_company_B])
df_customers
```

```
   Customer ID   Age  Gender  Salary City_Residence
0            1  45.0    Male   40000            NaN
1            2  12.0    Male       0      Bangalore
2            4   NaN  Female  150000      Bangalore
3            6  26.0    Male   30000        Chennai
4            9  64.0  Female   15000        Chennai
0            3  45.0    Male   88900            NaN
1            5  23.0     NaN   18000         Mumbai
2            7  67.0    Male   92000         Mumbai
3            8  34.0    Male  180000          Delhi
4           10  67.0    Male   92000         Mumbai
5           11  34.0    Male  180000          Delhi
```

Read the DataFrames

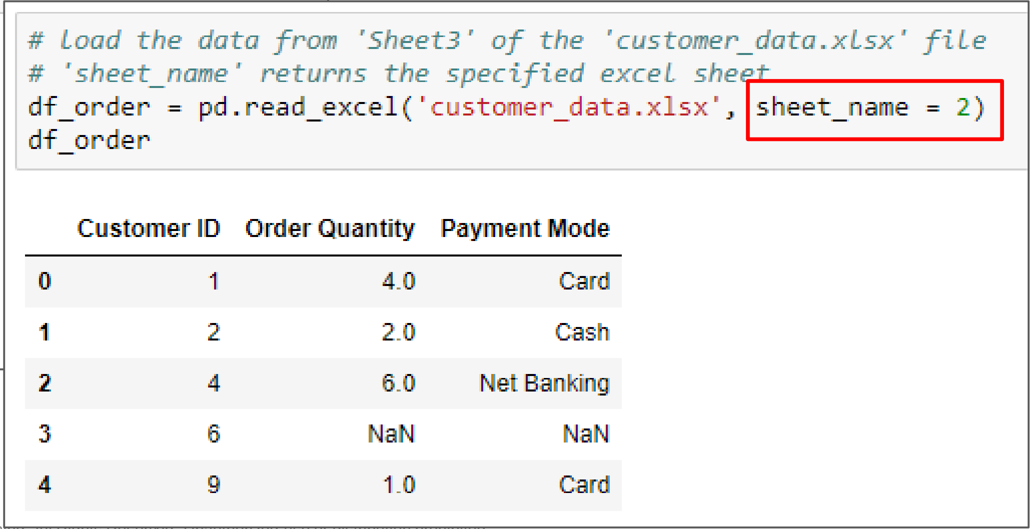
The DataFrames contains order data for customers of company 'A'. Use these DataFrames for further manipulations:



```
# load the data from 'Sheet1' of the 'customer_data.xlsx' file
# 'sheet_name' returns the specified excel sheet
df_company_A = pd.read_excel('customer_data.xlsx', sheet_name = 0)
df_company_A
```

	Customer ID	Age	Gender	Salary	City_Residence
0	1	45.0	Male	40000	NaN
1	2	12.0	Male	0	Bangalore
2	4	NaN	Female	150000	Bangalore
3	6	26.0	Male	30000	Chennai
4	9	64.0	Female	15000	Chennai

Customer details



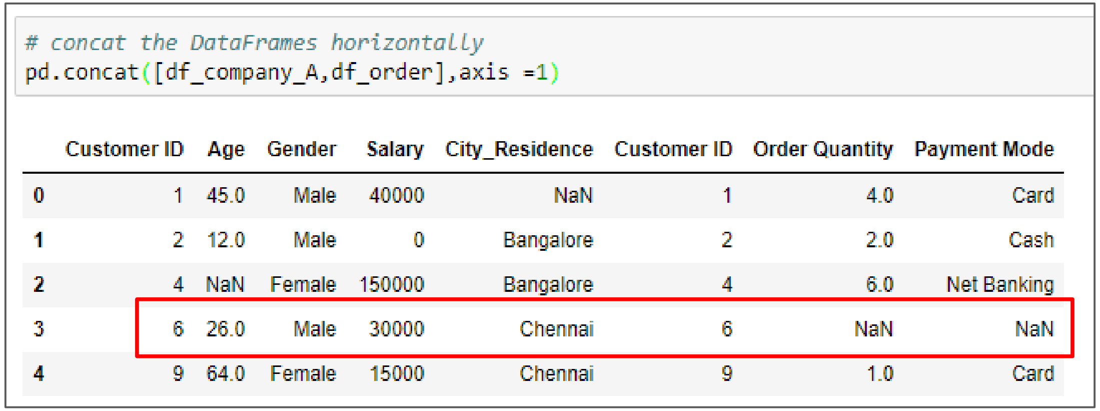
```
# load the data from 'Sheet3' of the 'customer_data.xlsx' file
# 'sheet_name' returns the specified excel sheet
df_order = pd.read_excel('customer_data.xlsx', sheet_name = 2)
df_order
```

	Customer ID	Order Quantity	Payment Mode
0	1	4.0	Card
1	2	2.0	Cash
2	4	6.0	Net Banking
3	6	NaN	NaN
4	9	1.0	Card

Order details

Concatenate the DataFrames

- The parameter, ‘axis = 1’ concatenates the DataFrames horizontally
- The concatenation is in the order they are passed in the function
- As, we do not have the order details for ‘Customer ID = 6’, the NaNs are printed for corresponding columns

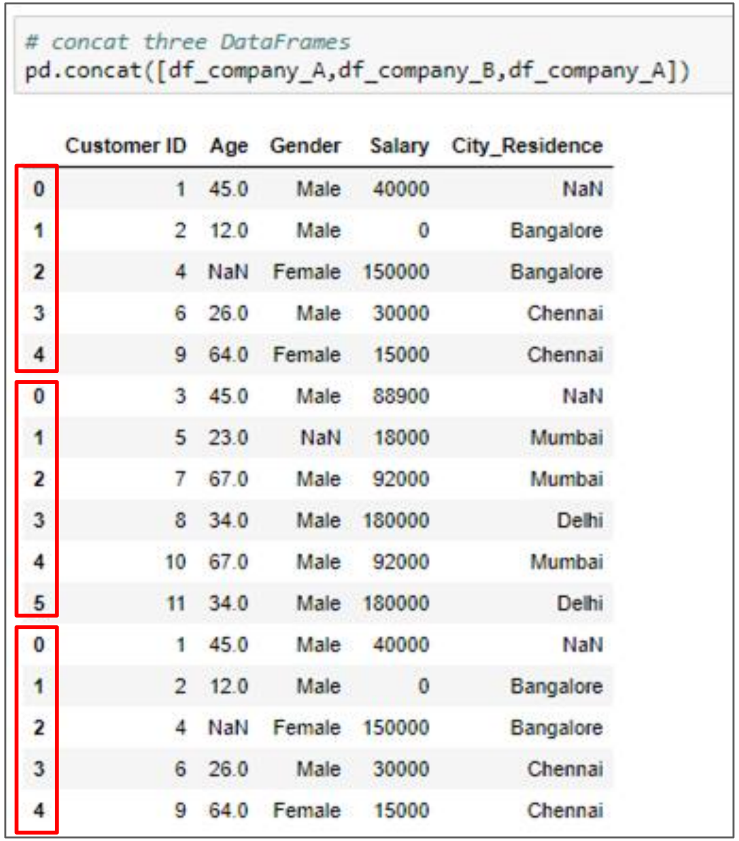


```
# concat the DataFrames horizontally
pd.concat([df_company_A,df_order],axis =1)
```

	Customer ID	Age	Gender	Salary	City_Residence	Customer ID	Order Quantity	Payment Mode
0	1	45.0	Male	40000	NaN	1	4.0	Card
1	2	12.0	Male	0	Bangalore	2	2.0	Cash
2	4	NaN	Female	150000	Bangalore	4	6.0	Net Banking
3	6	26.0	Male	30000	Chennai	6	NaN	NaN
4	9	64.0	Female	15000	Chennai	9	1.0	Card

Concatenate multiple DataFrames

The concat() method can be used to concatenate more than two DataFrames simultaneously

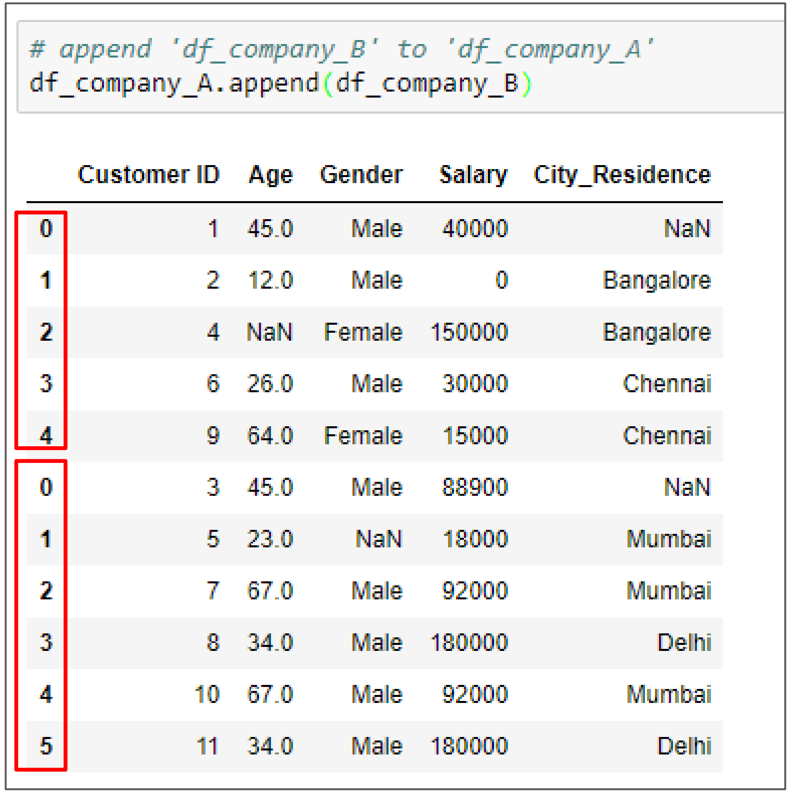


```
# concat three DataFrames
pd.concat([df_company_A,df_company_B,df_company_A])
```

	Customer ID	Age	Gender	Salary	City_Residence
0	1	45.0	Male	40000	NaN
1	2	12.0	Male	0	Bangalore
2	4	NaN	Female	150000	Bangalore
3	6	26.0	Male	30000	Chennai
4	9	64.0	Female	15000	Chennai
0	3	45.0	Male	88900	NaN
1	5	23.0	NaN	18000	Mumbai
2	7	67.0	Male	92000	Mumbai
3	8	34.0	Male	180000	Delhi
4	10	67.0	Male	92000	Mumbai
5	11	34.0	Male	180000	Delhi
0	1	45.0	Male	40000	NaN
1	2	12.0	Male	0	Bangalore
2	4	NaN	Female	150000	Bangalore
3	6	26.0	Male	30000	Chennai
4	9	64.0	Female	15000	Chennai

Concatenate the DataFrames

- The append() method is used append a DataFrame with another
- Here, we append the customers data of company ‘B’ to data of company ‘A’

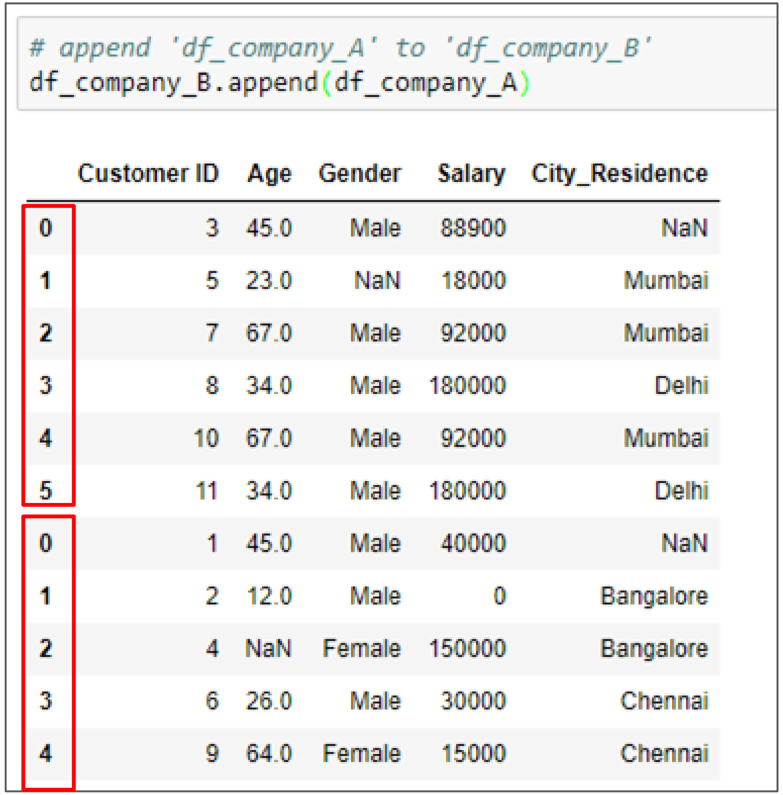


```
# append 'df_company_B' to 'df_company_A'
df_company_A.append(df_company_B)
```

	Customer ID	Age	Gender	Salary	City_Residence
0	1	45.0	Male	40000	NaN
1	2	12.0	Male	0	Bangalore
2	4	NaN	Female	150000	Bangalore
3	6	26.0	Male	30000	Chennai
4	9	64.0	Female	15000	Chennai
0	3	45.0	Male	88900	NaN
1	5	23.0	NaN	18000	Mumbai
2	7	67.0	Male	92000	Mumbai
3	8	34.0	Male	180000	Delhi
4	10	67.0	Male	92000	Mumbai
5	11	34.0	Male	180000	Delhi

Concatenate the DataFrames

Here, we append the customers data of company ‘A’ to data of company ‘B’



```
# append 'df_company_A' to 'df_company_B'
df_company_B.append(df_company_A)
```

	Customer ID	Age	Gender	Salary	City_Residence
0	3	45.0	Male	88900	NaN
1	5	23.0	NaN	18000	Mumbai
2	7	67.0	Male	92000	Mumbai
3	8	34.0	Male	180000	Delhi
4	10	67.0	Male	92000	Mumbai
5	11	34.0	Male	180000	Delhi
0	1	45.0	Male	40000	NaN
1	2	12.0	Male	0	Bangalore
2	4	NaN	Female	150000	Bangalore
3	6	26.0	Male	30000	Chennai
4	9	64.0	Female	15000	Chennai

Append vs. Concat

append()	concat()
returns the error if one tries to concatenate more than two DataFrames simultaneously	concatenates multiple DataFrames simultaneously

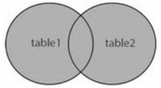
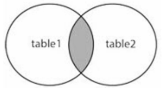
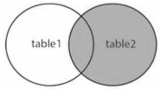
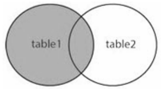
Join the DataFrames

Join the DataFrames

- The join() method join the DataFrames based on index or key column
- Index of the first DataFrame should match to one of the column in the second DataFrame

Types of join

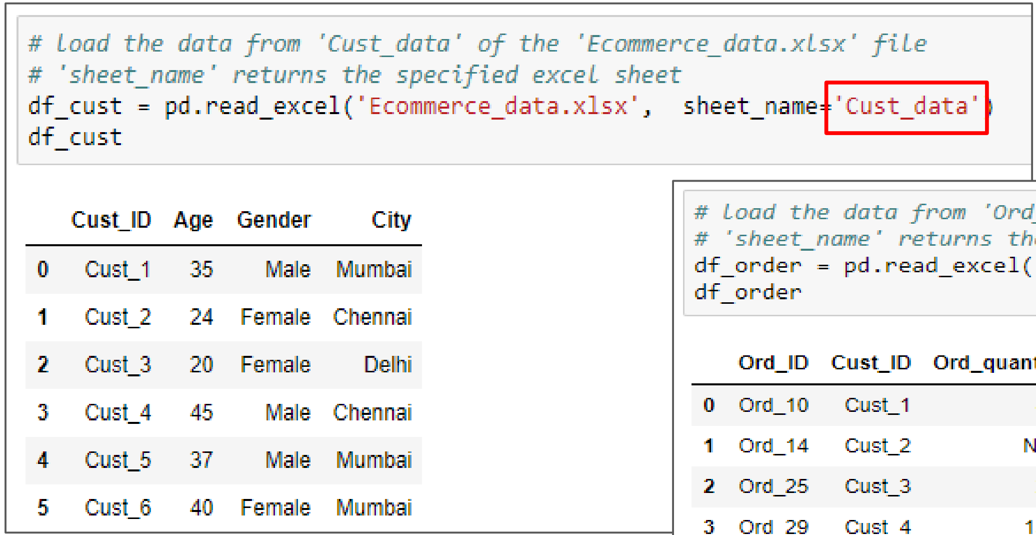
The join types can be specified using the parameter, 'how'

how = 'Type'	Description	
outer	Use union of index (or column) observed in both DataFrames	
inner	Use intersection of index (or column) observed in both DataFrames	
right	Use only the index found in the right DataFrame	
left	Use only the index (or column) found in the left DataFrame	

If the type is not specified, by default it is 'left'

Read the DataFrames

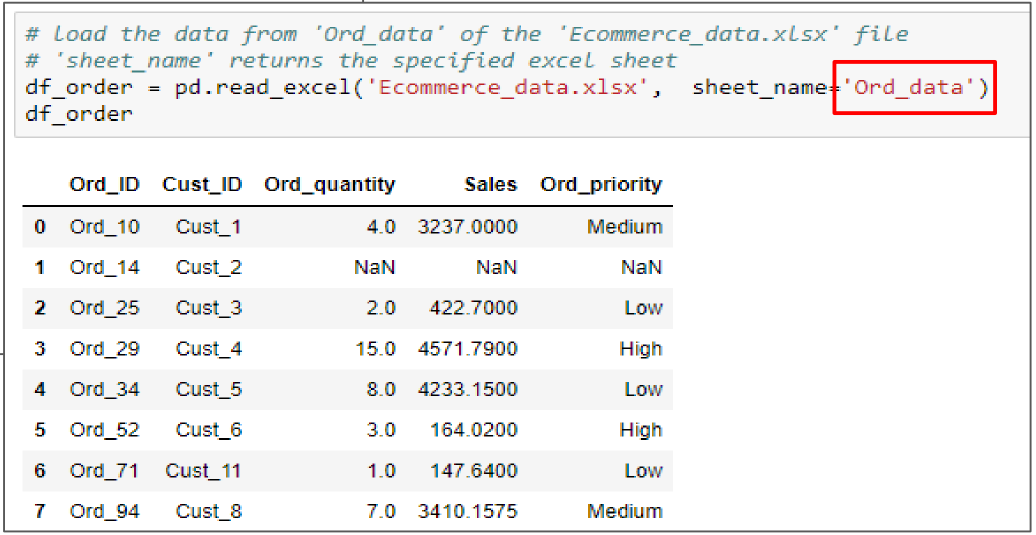
Use the following DataFrames for further manipulations:



```
# load the data from 'Cust_data' of the 'Ecommerce_data.xlsx' file
# 'sheet_name' returns the specified excel sheet
df_cust = pd.read_excel('Ecommerce_data.xlsx',  sheet_name='Cust_data')
df_cust
```

	Cust_ID	Age	Gender	City
0	Cust_1	35	Male	Mumbai
1	Cust_2	24	Female	Chennai
2	Cust_3	20	Female	Delhi
3	Cust_4	45	Male	Chennai
4	Cust_5	37	Male	Mumbai
5	Cust_6	40	Female	Mumbai

Customer details



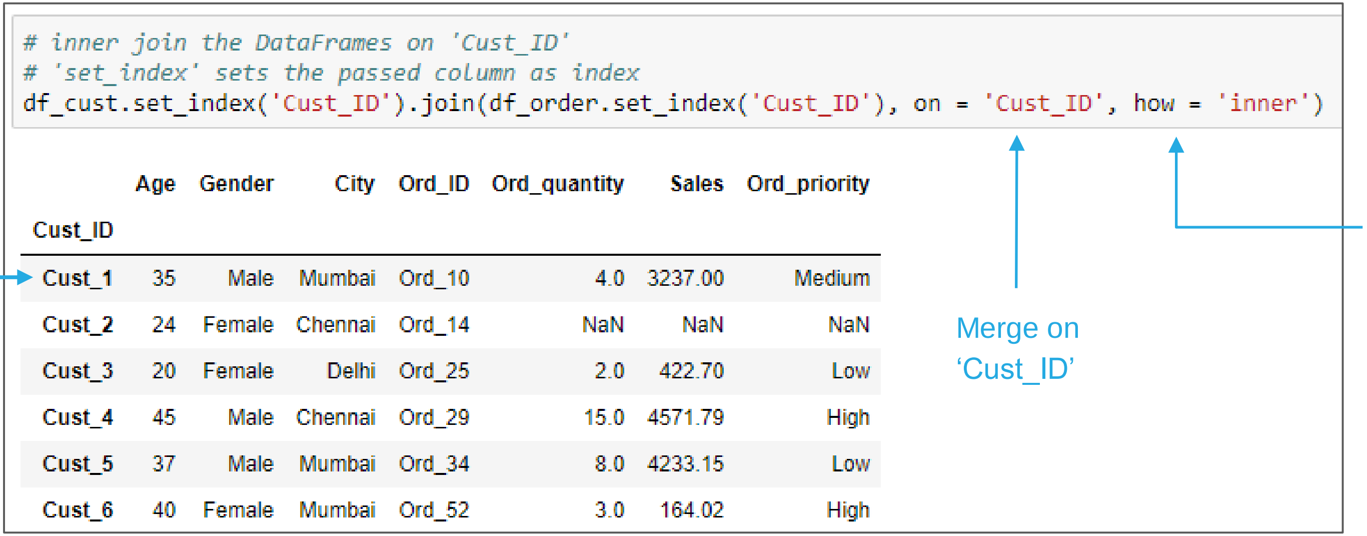
```
# load the data from 'Ord_data' of the 'Ecommerce_data.xlsx' file
# 'sheet_name' returns the specified excel sheet
df_order = pd.read_excel('Ecommerce_data.xlsx',  sheet_name='Ord_data')
df_order
```

	Ord_ID	Cust_ID	Ord_quantity	Sales	Ord_priority
0	Ord_10	Cust_1	4.0	3237.0000	Medium
1	Ord_14	Cust_2	NaN	NaN	NaN
2	Ord_25	Cust_3	2.0	422.7000	Low
3	Ord_29	Cust_4	15.0	4571.7900	High
4	Ord_34	Cust_5	8.0	4233.1500	Low
5	Ord_52	Cust_6	3.0	164.0200	High
6	Ord_71	Cust_11	1.0	147.6400	Low
7	Ord_94	Cust_8	7.0	3410.1575	Medium

Order details

Inner Join

Join the DataFrames to get the order details along with the customer information



```
# inner join the DataFrames on 'Cust_ID'
# 'set_index' sets the passed column as index
df_cust.set_index('Cust_ID').join(df_order.set_index('Cust_ID'), on = 'Cust_ID', how = 'inner')
```

	Age	Gender	City	Ord_ID	Ord_quantity	Sales	Ord_priority
Cust_ID							
Cust_1	35	Male	Mumbai	Ord_10	4.0	3237.00	Medium
Cust_2	24	Female	Chennai	Ord_14	NaN	NaN	NaN
Cust_3	20	Female	Delhi	Ord_25	2.0	422.70	Low
Cust_4	45	Male	Chennai	Ord_29	15.0	4571.79	High
Cust_5	37	Male	Mumbai	Ord_34	8.0	4233.15	Low
Cust_6	40	Female	Mumbai	Ord_52	3.0	164.02	High

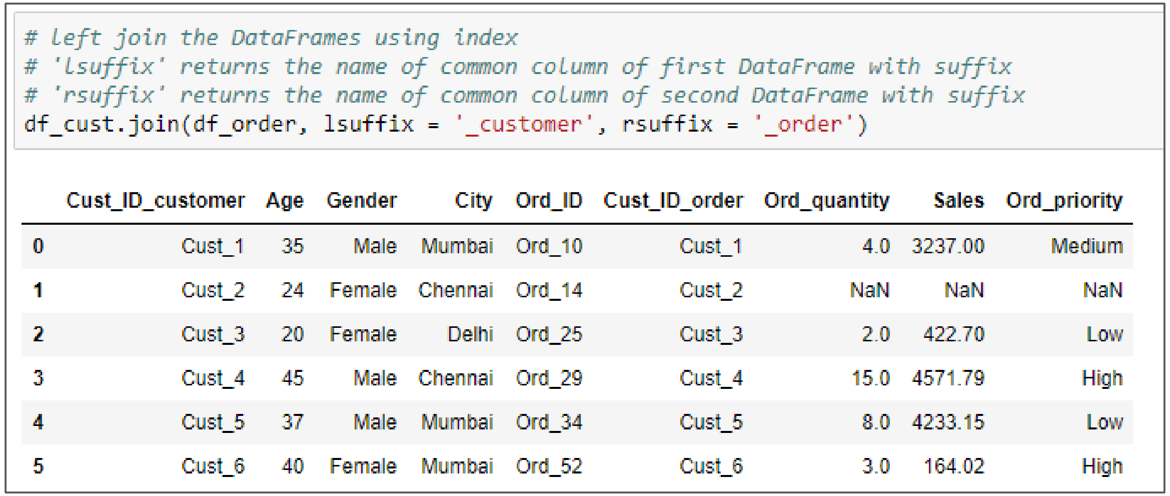
‘Cust_ID’ as index of the new DataFrame

Merge on ‘Cust_ID’

Merge includes the common IDs in both the DataFrames

Join using index

- Resultant DataFrame includes rows from both the DataFrames with same index as of ‘df_cust’
- This method is useful, only if the record have same index in both the DataFrames



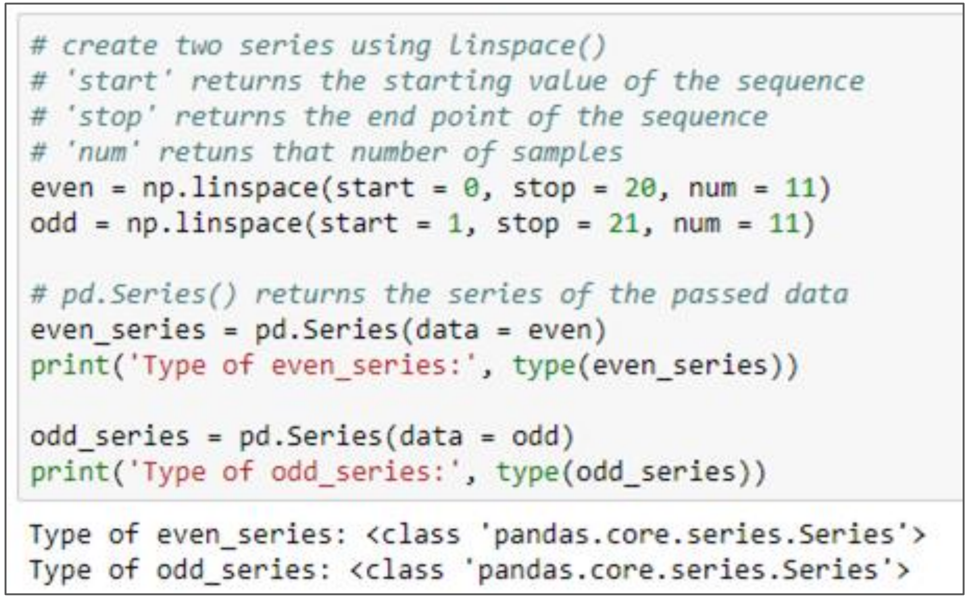
```
# left join the DataFrames using index
# 'lsuffix' returns the name of common column of first DataFrame with suffix
# 'rsuffix' returns the name of common column of second DataFrame with suffix
df_cust.join(df_order, lsuffix = '_customer', rsuffix = '_order')
```

	Cust_ID_customer	Age	Gender	City	Ord_ID	Cust_ID_order	Ord_quantity	Sales	Ord_priority
0	Cust_1	35	Male	Mumbai	Ord_10	Cust_1	4.0	3237.00	Medium
1	Cust_2	24	Female	Chennai	Ord_14	Cust_2	NaN	NaN	NaN
2	Cust_3	20	Female	Delhi	Ord_25	Cust_3	2.0	422.70	Low
3	Cust_4	45	Male	Chennai	Ord_29	Cust_4	15.0	4571.79	High
4	Cust_5	37	Male	Mumbai	Ord_34	Cust_5	8.0	4233.15	Low
5	Cust_6	40	Female	Mumbai	Ord_52	Cust_6	3.0	164.02	High

Unstack and Stack a Series

Create a series

Create python series as shown below:

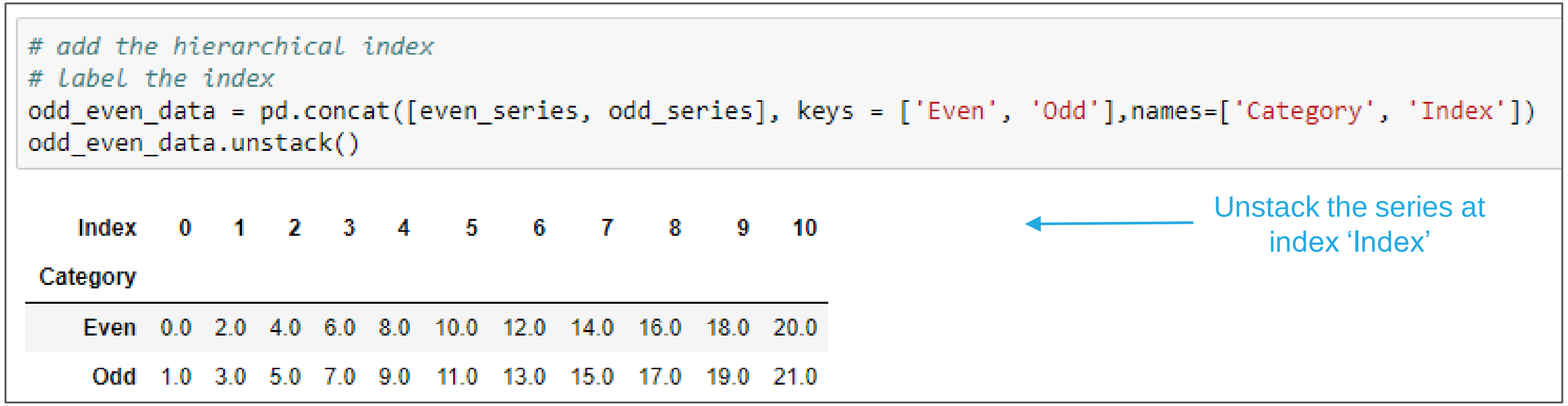


```
# create two series using linspace()
# 'start' returns the starting value of the sequence
# 'stop' returns the end point of the sequence
# 'num' retuns that number of samples
even = np.linspace(start = 0, stop = 20, num = 11)
odd = np.linspace(start = 1, stop = 21, num = 11)

# pd.Series() returns the series of the passed data
even_series = pd.Series(data = even)
print('Type of even_series:', type(even_series))

odd_series = pd.Series(data = odd)
print('Type of odd_series:', type(odd_series))
```

```
Type of even_series: <class 'pandas.core.series.Series'>
Type of odd_series: <class 'pandas.core.series.Series'>
```

Unstack a series

Unstacking can be used to rearrange the series with hierarchical index in a DataFrame



```
# add the hierarchical index
# label the index
odd_even_data = pd.concat([even_series, odd_series], keys = ['Even', 'Odd'],names=['Category', 'Index'])
odd_even_data.unstack()
```

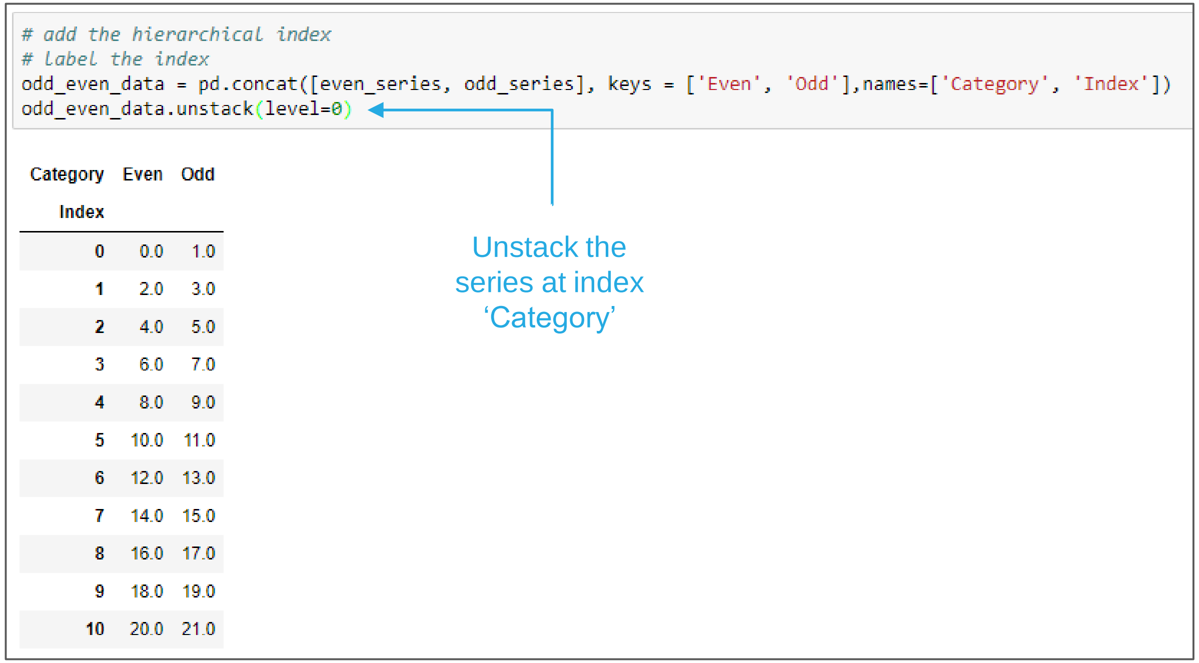
```
   Index   0    1    2    3    4     5     6     7     8     9    10
Category
    Even 0.0  2.0  4.0  6.0  8.0  10.0  12.0  14.0  16.0  18.0  20.0
     Odd 1.0  3.0  5.0  7.0  9.0  11.0  13.0  15.0  17.0  19.0  21.0
```

Unstack the series at index 'Index'

Unstack a series

- We can pass the level of index to unstack the series using parameter, ‘level’

- By default, the unstack() method uses the last level of index (-1) to unstack the series



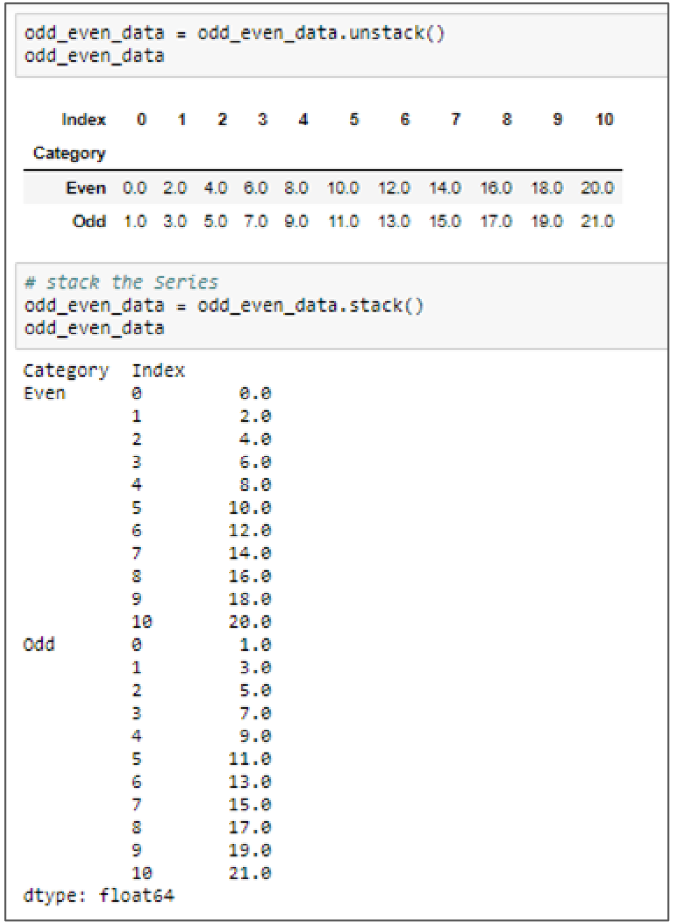
```
# add the hierarchical index
# label the index
odd_even_data = pd.concat([even_series, odd_series], keys = ['Even', 'Odd'],names=['Category', 'Index'])
odd_even_data.unstack(level=0)
```

```
Category  Even   Odd
   Index
       0   0.0   1.0
       1   2.0   3.0
       2   4.0   5.0
       3   6.0   7.0
       4   8.0   9.0
       5  10.0  11.0
       6  12.0  13.0
       7  14.0  15.0
       8  16.0  17.0
       9  18.0  19.0
      10  20.0  21.0
```

Unstack the series at index ‘Category’

Stack a series

- Stack is the inverse operation of unstack
- It returns a series with hierarchical index



```
odd_even_data = odd_even_data.unstack()
odd_even_data
```

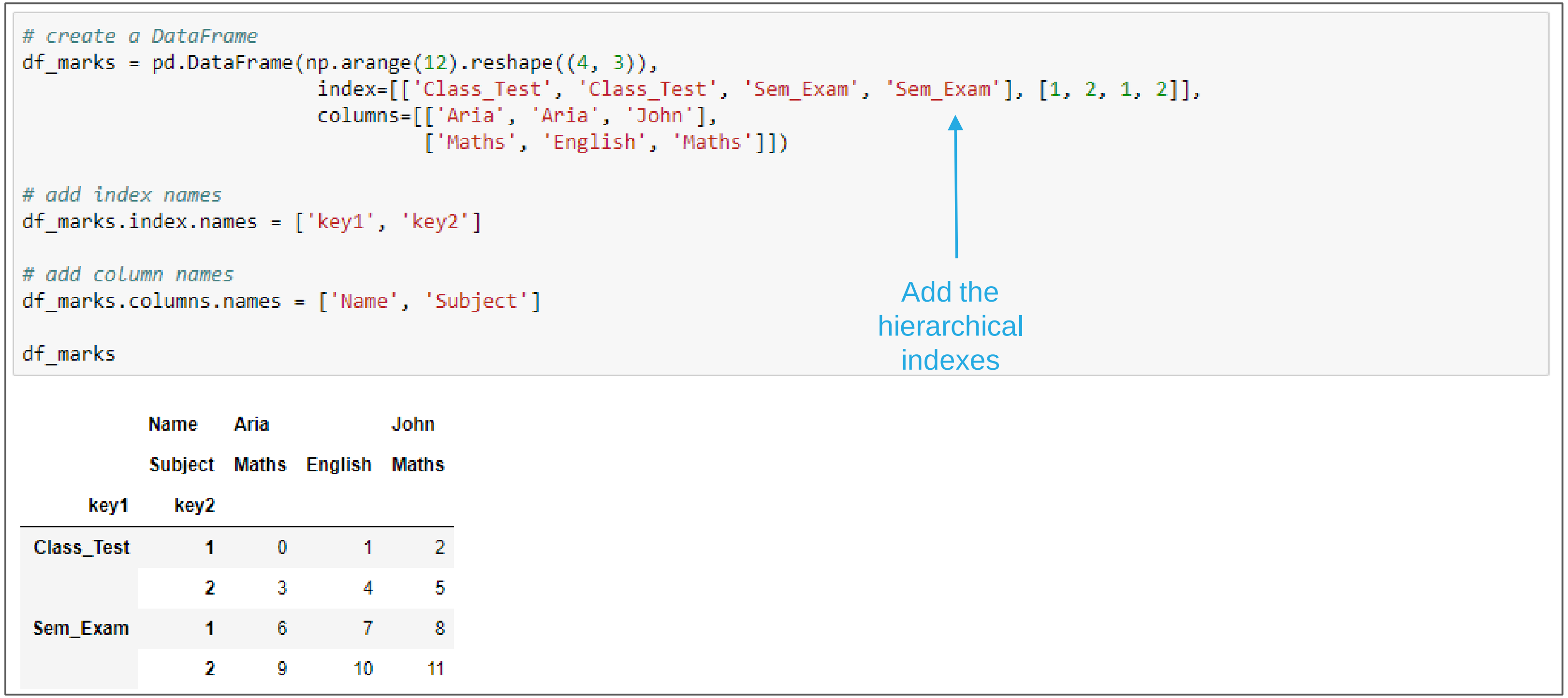
```
Index      0    1    2    3    4     5     6     7     8     9    10
Category
Even     0.0  2.0  4.0  6.0  8.0  10.0  12.0  14.0  16.0  18.0  20.0
Odd      1.0  3.0  5.0  7.0  9.0  11.0  13.0  15.0  17.0  19.0  21.0
```

```
# stack the Series
odd_even_data = odd_even_data.stack()
odd_even_data
```

```
Category  Index
Even      0         0.0
          1         2.0
          2         4.0
          3         6.0
          4         8.0
          5        10.0
          6        12.0
          7        14.0
          8        16.0
          9        18.0
          10       20.0
Odd       0         1.0
          1         3.0
          2         5.0
          3         7.0
          4         9.0
          5        11.0
          6        13.0
          7        15.0
          8        17.0
          9        19.0
          10       21.0
dtype: float64
```

Unstack and Stack the DataFrame

Create the DataFrame

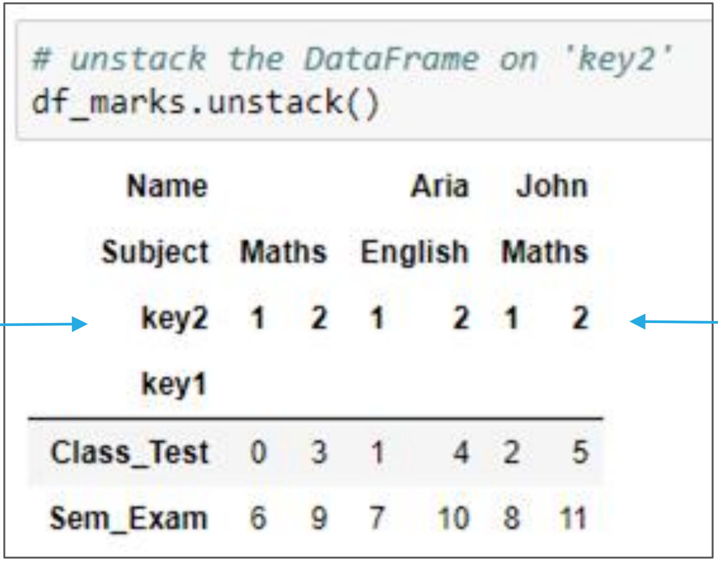


```
# create a DataFrame
df_marks = pd.DataFrame(np.arange(12).reshape((4, 3)),
                         index=[['Class_Test', 'Class_Test', 'Sem_Exam', 'Sem_Exam'], [1, 2, 1, 2]],
                         columns=[['Aria', 'Aria', 'John'],
                                  ['Maths', 'English', 'Maths']])

# add index names
df_marks.index.names = ['key1', 'key2']

# add column names
df_marks.columns.names = ['Name', 'Subject']

df_marks
```

```
                Name   Aria           John
             Subject  Maths  English  Maths
      key1      key2
 Class_Test        1      0        1      2
                   2      3        4      5
   Sem_Exam        1      6        7      8
                   2      9       10     11
```

Unstack the DataFrame

Unstacking a DataFrame returns a DataFrame having a new level of column label which consists of the pivoted index label



```
# unstack the DataFrame on 'key2'
df_marks.unstack()
```

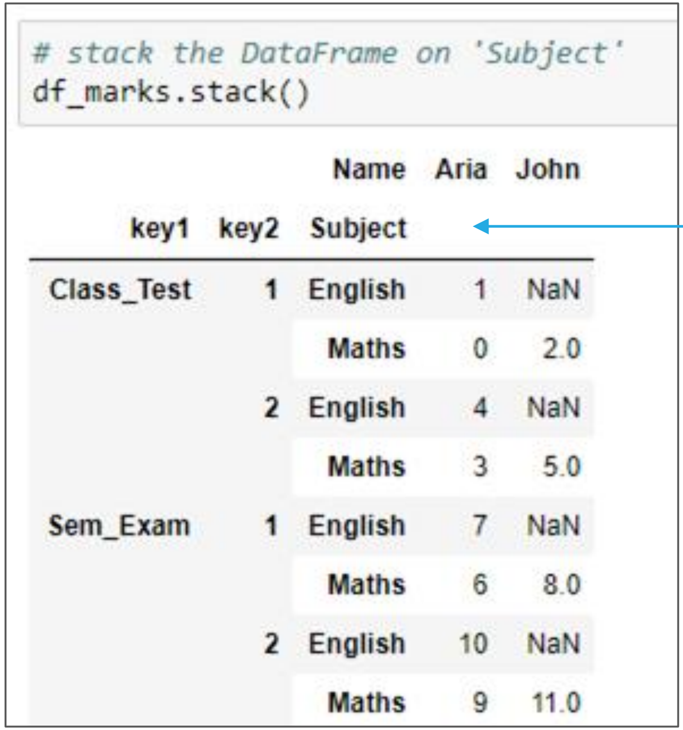
```
     Name              Aria      John
  Subject  Maths  English  Maths
     key2   1   2   1    2   1   2
     key1
Class_Test  0   3   1    4   2   5
 Sem_Exam   6   9   7   10   8  11
```

New level of column

Unstack the DataFrame at index ‘key2’

Stack the DataFrame

- Stacking a DataFrame returns a DataFrame having a new level of innermost index consisting of the the pivoted column label
- As the English marks for John are unknown, the NaNs are printed for corresponding observations



```
# stack the DataFrame on 'Subject'
df_marks.stack()
```

```
                   Name  Aria  John
key1      key2  Subject
Class_Test   1  English     1   NaN
                  Maths     0   2.0
             2  English     4   NaN
                  Maths     3   5.0
Sem_Exam     1  English     7   NaN
                  Maths     6   8.0
             2  English    10   NaN
                  Maths     9  11.0
```

New level of index