Programming Basics and Data Analytics with Python

simplilearn

Data Manipulation with Pandas

Learning Objectives

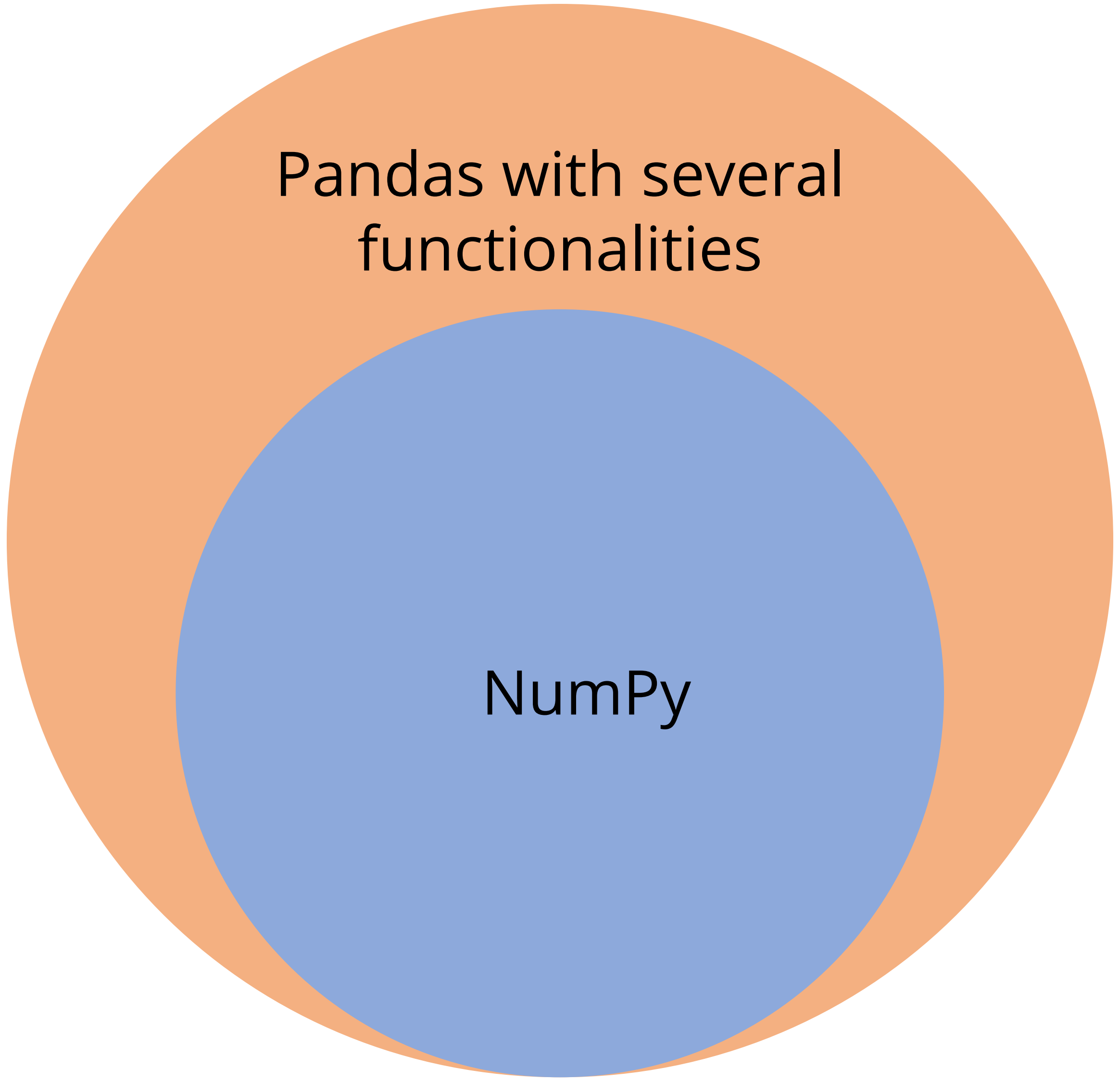
By the end of this lesson, you will be able to:

- Explain pandas and its features
- List different data structures of pandas
- Outline the process to create Series and DataFrame with data inputs
- Explain how to view, select, and access elements in a data structure
- Describe the procedure to handle vectorized operations
- Illustrate how to handle missing values
- Analyze data with different data operation methods

Introduction to Pandas

Why Pandas

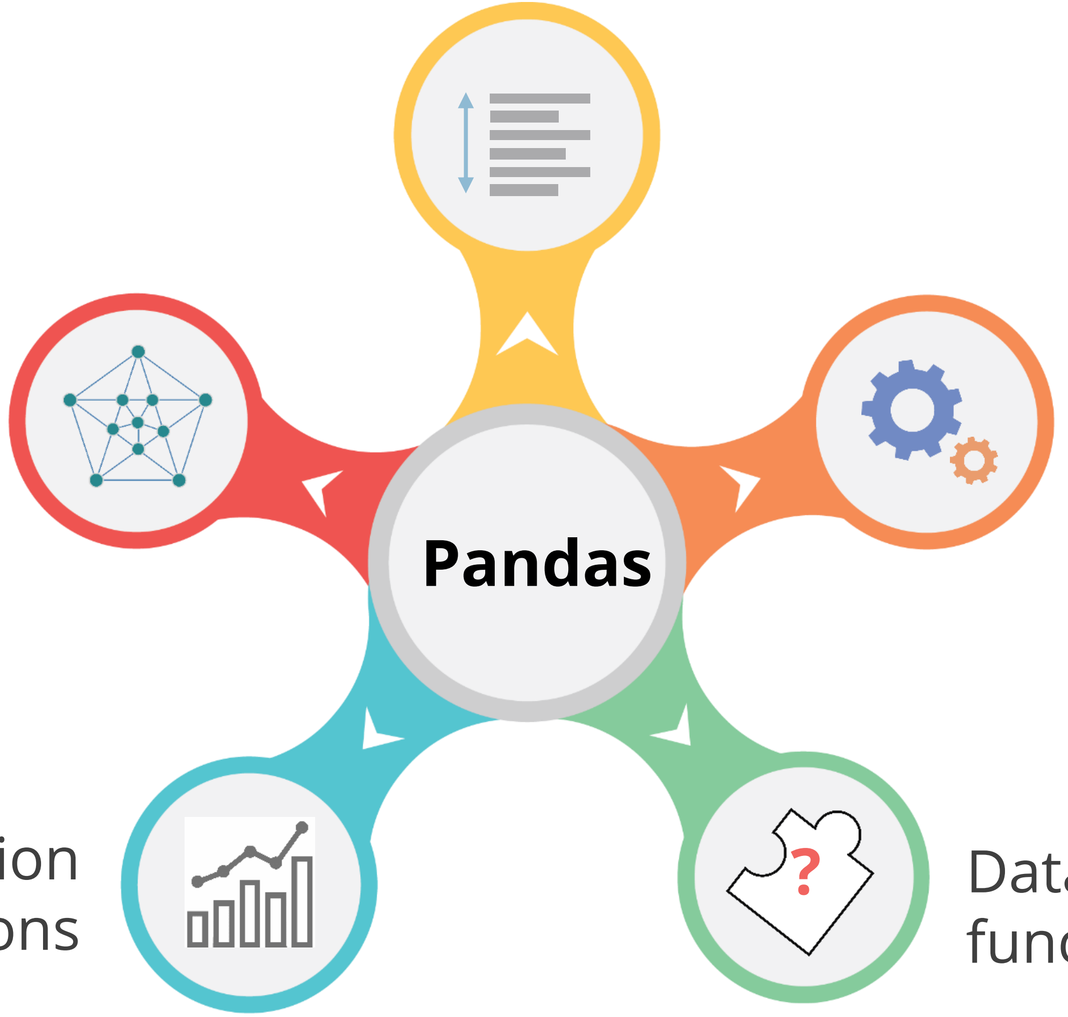
NumPy is great for mathematical computing, but, why do we need pandas?



Pandas with several functionalities

NumPy

Why Pandas



Intrinsic data alignment

Data structures handling major use cases

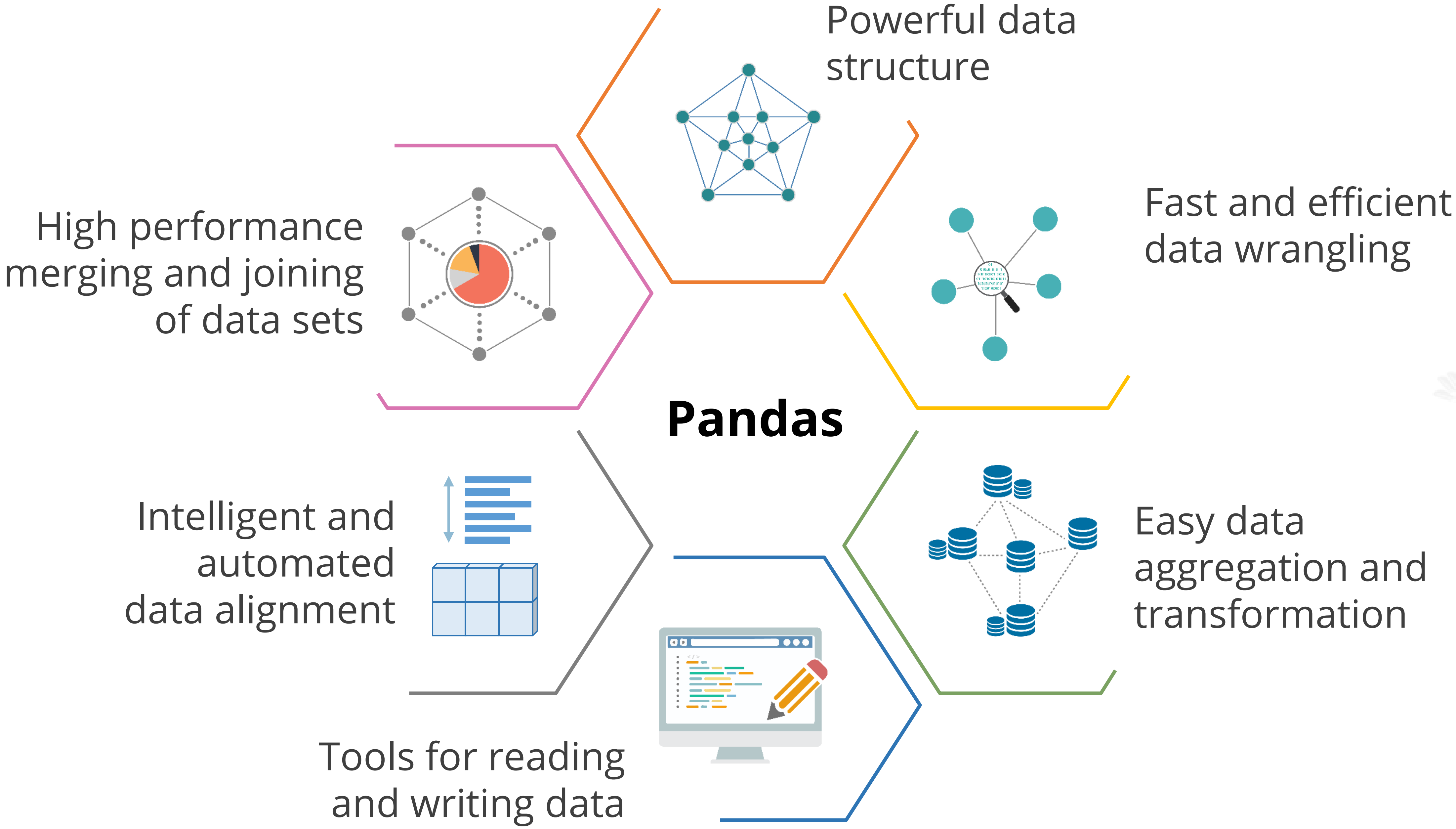
Data operation functions

Data standardization functions

Data-handling functions

Features of Pandas

Various features of Pandas make it an efficient library for Data Scientists.



Pandas

Powerful data structure

Fast and efficient data wrangling

Easy data aggregation and transformation

Tools for reading and writing data

Intelligent and automated data alignment

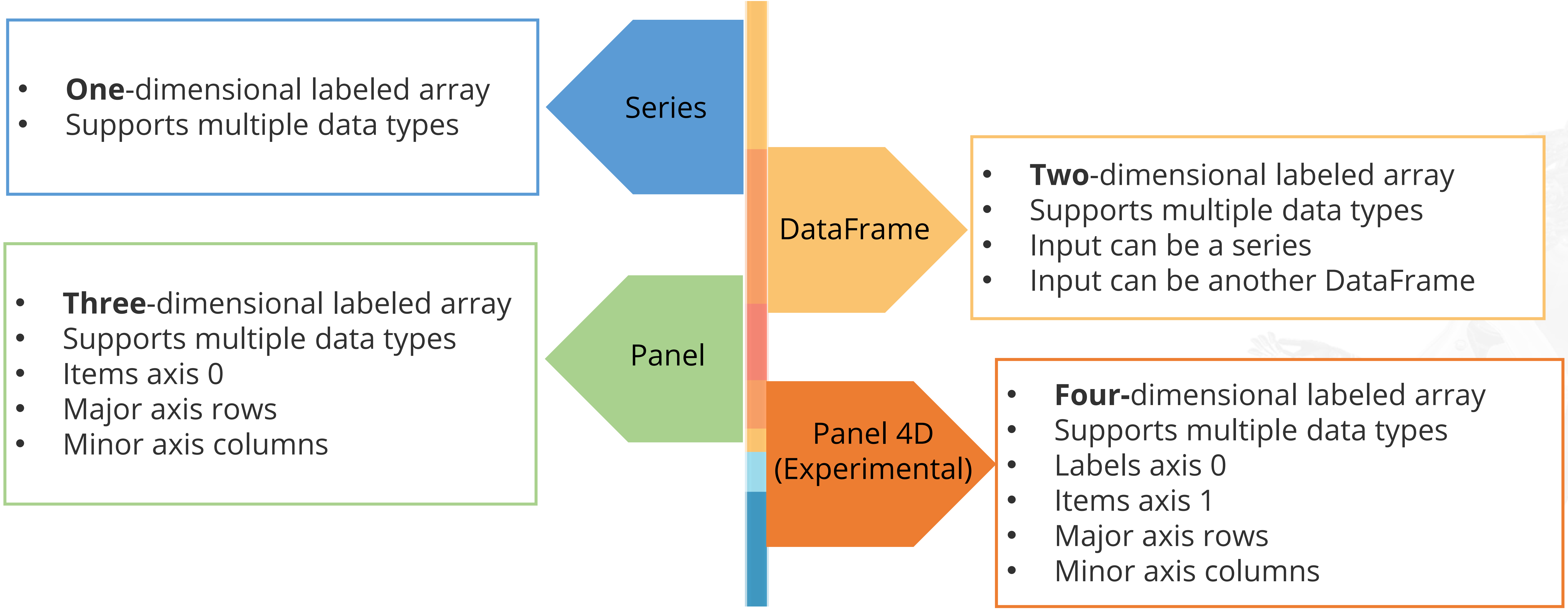
High performance merging and joining of data sets

DATA AND
ARTIFICIAL INTELLIGENCE

Data Structures

Data Structures

Data structures in Pandas library:



Series

- **One**-dimensional labeled array
- Supports multiple data types

DataFrame

- **Two**-dimensional labeled array
- Supports multiple data types
- Input can be a series
- Input can be another DataFrame

Panel

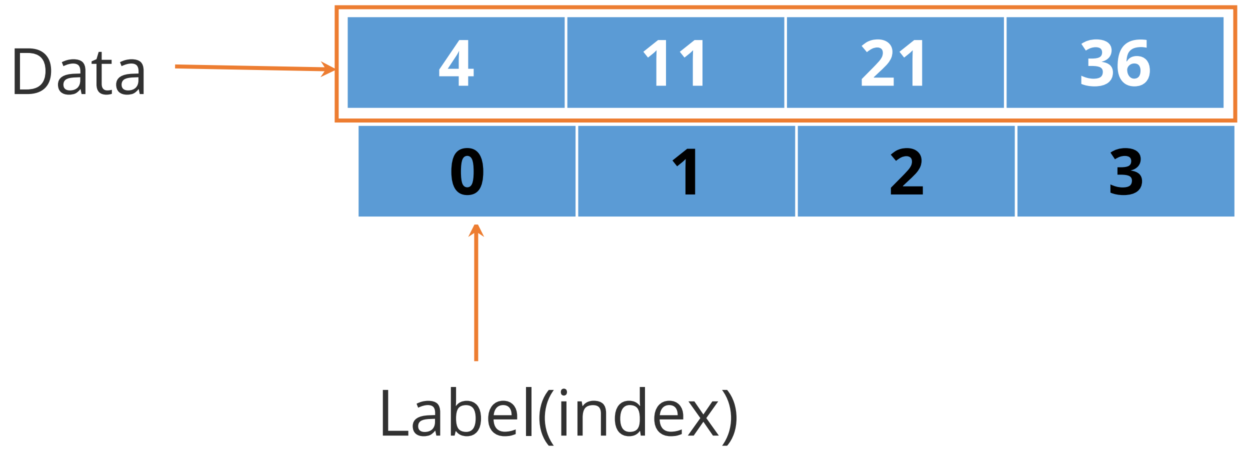
- **Three**-dimensional labeled array
- Supports multiple data types
- Items axis 0
- Major axis rows
- Minor axis columns

Panel 4D (Experimental)

- **Four-**dimensional labeled array
- Supports multiple data types
- Labels axis 0
- Items axis 1
- Major axis rows
- Minor axis columns

Understanding Series

Series is a one-dimensional array-like object containing data and labels (or index).

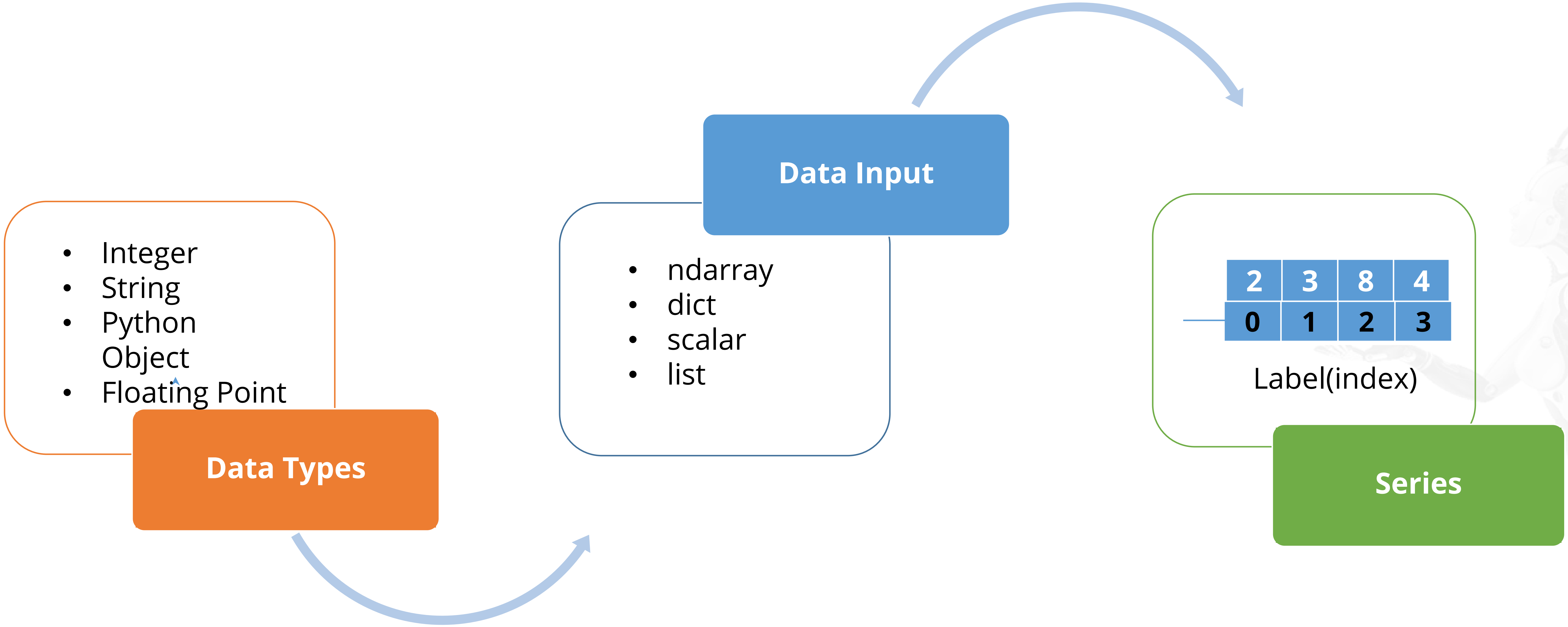


4	11	21	36
0	1	2	3

Data alignment is intrinsic and will not be broken until changed explicitly by program.

Series

Series can be created with different data inputs:



- Integer
- String
- Python Object
- Floating Point

Data Types

Data Input

- ndarray
- dict
- scalar
- list

2	3	8	4
0	1	2	3

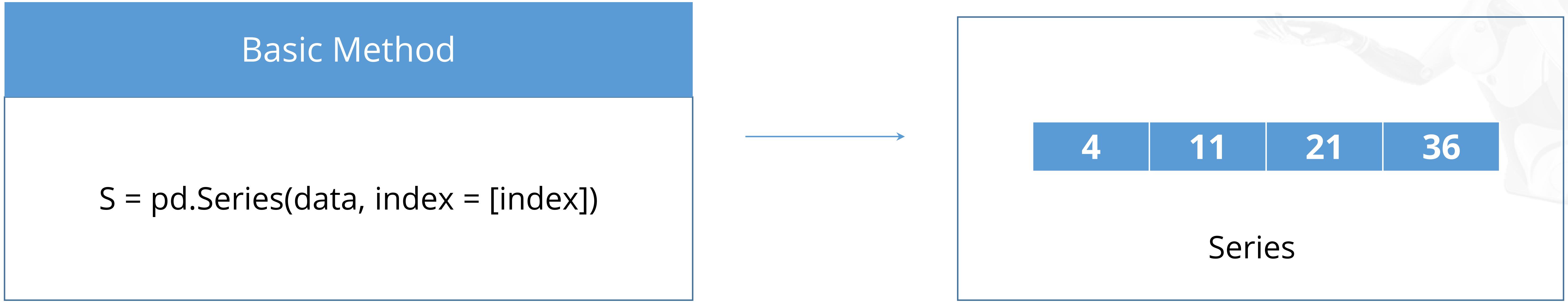
Label(index)

Series

How to Create Series?

Key points to note while creating a series are:

- Import pandas as it is in the main library (Import pandas as pd)
- Import NumPy while working with ndarrays (Import NumPy as np)
- Apply the syntax and pass the data elements as arguments



Basic Method

```
S = pd.Series(data, index = [index])
```

4	11	21	36

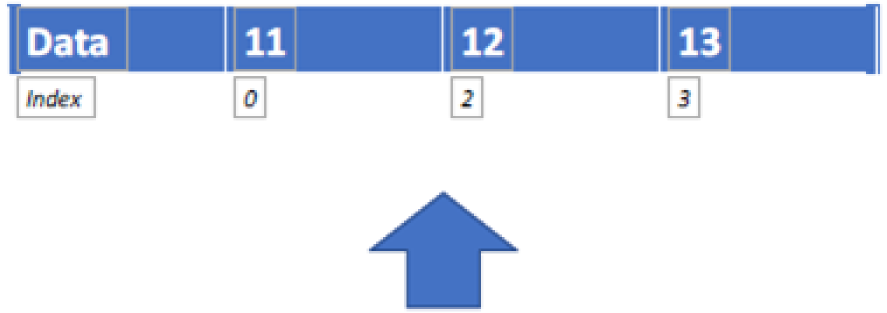
Series

Series: Example

Consider this example of series in pandas:

Code

```
data = [11,12,13]
s = pd.Series(data)
s
0   11
1   23
2   23
dtype: int64
S[1]
23
```

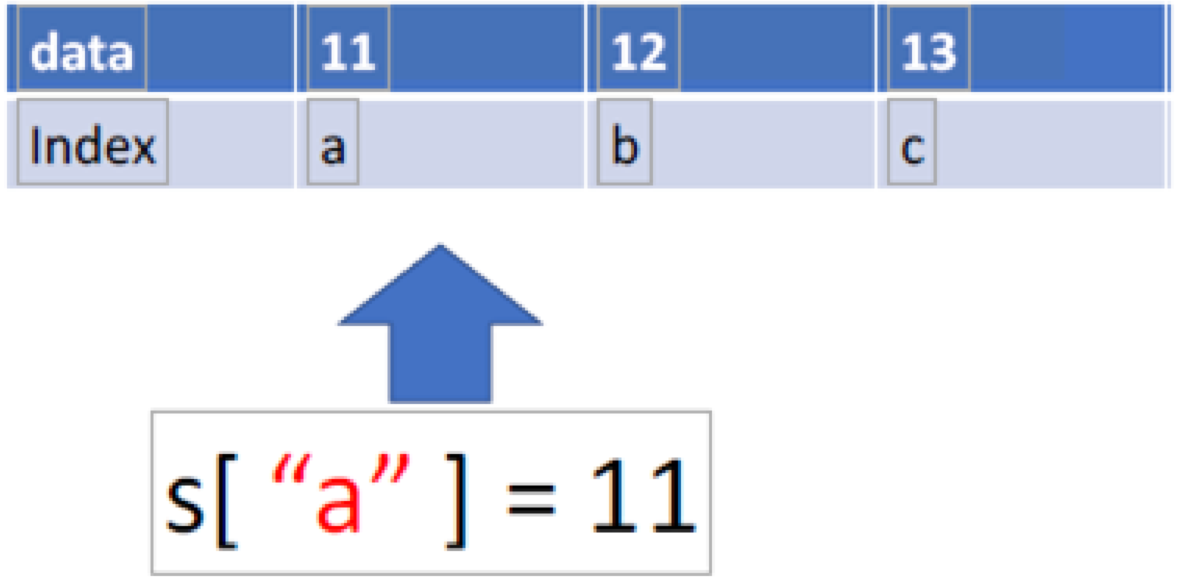


s[1] = 12

Series: Example

Code

```
data = [11,12,13]
index = [“a”,“b”,“c”]
s = pd.Series(data, index=ind)
s
a  11
b  23
c  23
s[“a”]
11
```



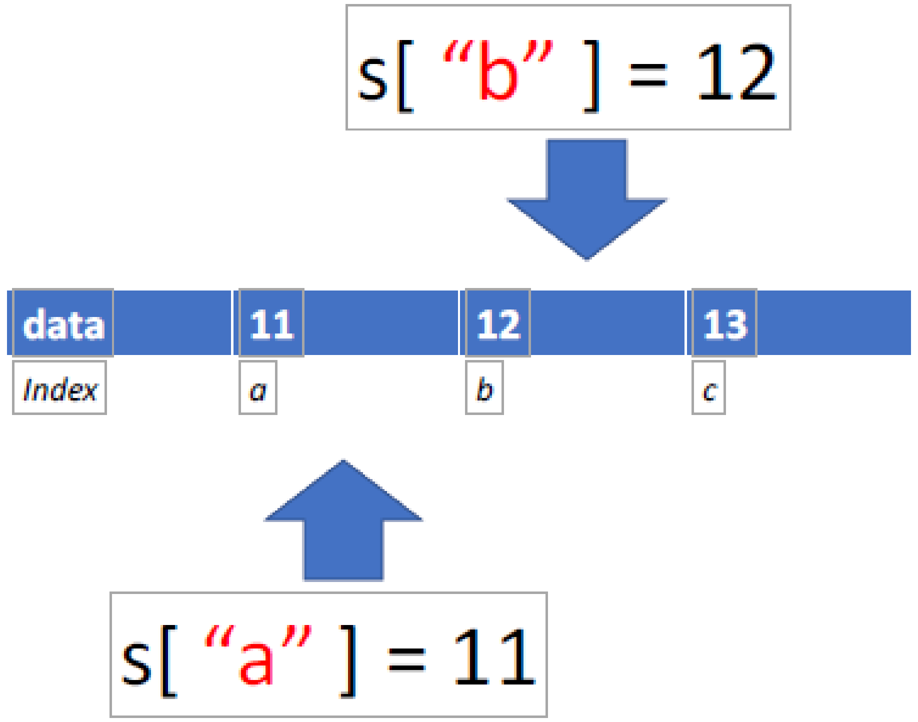
data	11	12	13
Index	a	b	c

s[“a”] = 11

Series: Example

Code

```
import pandas as pd
data = [11,12,13]
ind = [“a”,“b”,“c”]
S = pd.Series(data, index=ind)
S[[“a”, “b”]]
a  11
B  12
dtype: int64
```

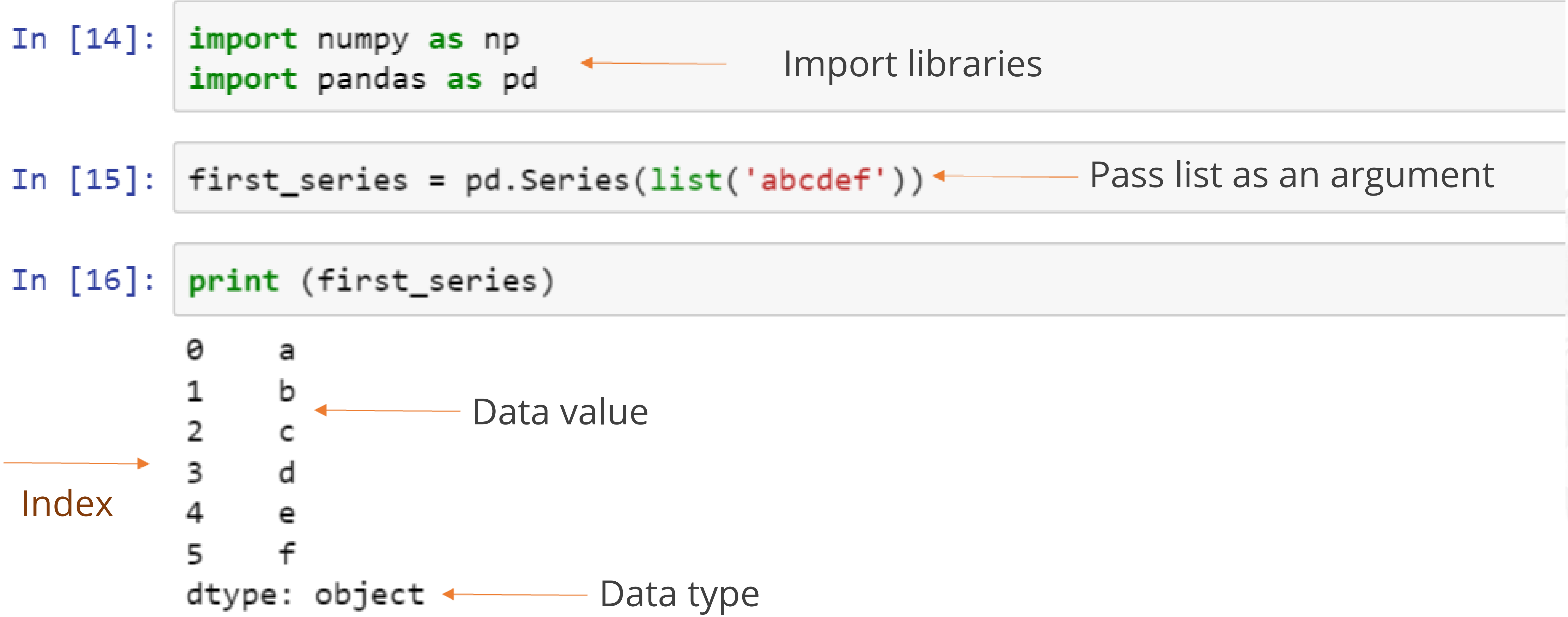


s[“b”] = 12

data	11	12	13
Index	*a*	*b*	*c*

s[“a”] = 11

Creating Series from a List



```
In [14]: import numpy as np
         import pandas as pd
```

Import libraries

```
In [15]: first_series = pd.Series(list('abcdef'))
```

Pass list as an argument

```
In [16]: print (first_series)

0     a
1     b
2     c
3     d
4     e
5     f
dtype: object
```

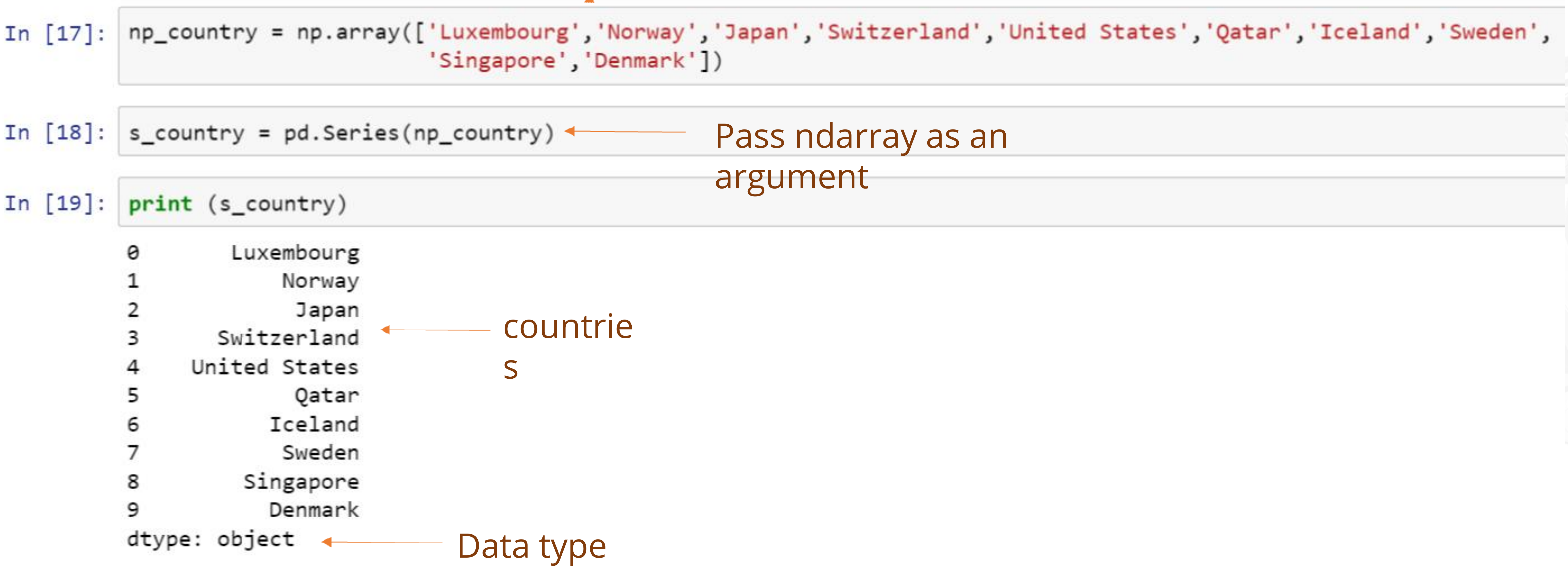
Data value

Index

Data type

Creating Series from an ndarray

ndarray for countries



```
In [17]: np_country = np.array(['Luxembourg','Norway','Japan','Switzerland','United States','Qatar','Iceland','Sweden',
                                'Singapore','Denmark'])
```

```
In [18]: s_country = pd.Series(np_country)
```

Pass ndarray as an argument

```
In [19]: print (s_country)

0       Luxembourg
1           Norway
2            Japan
3      Switzerland
4    United States
5            Qatar
6          Iceland
7           Sweden
8        Singapore
9          Denmark
dtype: object
```

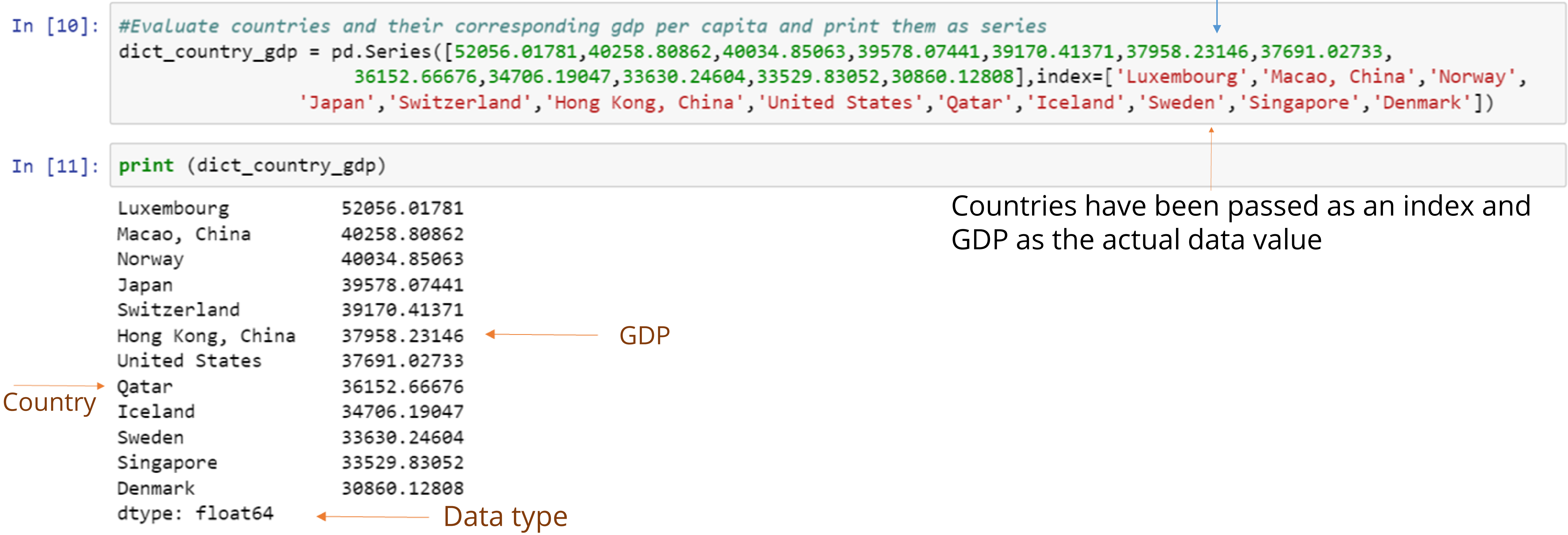
countrie
s

Data type

Creating Series from dict

A series can also be created with dict data input for faster operations.

dict for countries and their GDP



```
In [10]: #Evaluate countries and their corresponding gdp per capita and print them as series
         dict_country_gdp = pd.Series([52056.01781,40258.80862,40034.85063,39578.07441,39170.41371,37958.23146,37691.02733,
                                      36152.66676,34706.19047,33630.24604,33529.83052,30860.12808],index=['Luxembourg','Macao, China','Norway',
                                 'Japan','Switzerland','Hong Kong, China','United States','Qatar','Iceland','Sweden','Singapore','Denmark'])
```

```
In [11]: print (dict_country_gdp)
```

```
Luxembourg          52056.01781
Macao, China        40258.80862
Norway              40034.85063
Japan               39578.07441
Switzerland         39170.41371
Hong Kong, China    37958.23146
United States       37691.02733
Qatar               36152.66676
Iceland             34706.19047
Sweden              33630.24604
Singapore           33529.83052
Denmark             30860.12808
dtype: float64
```

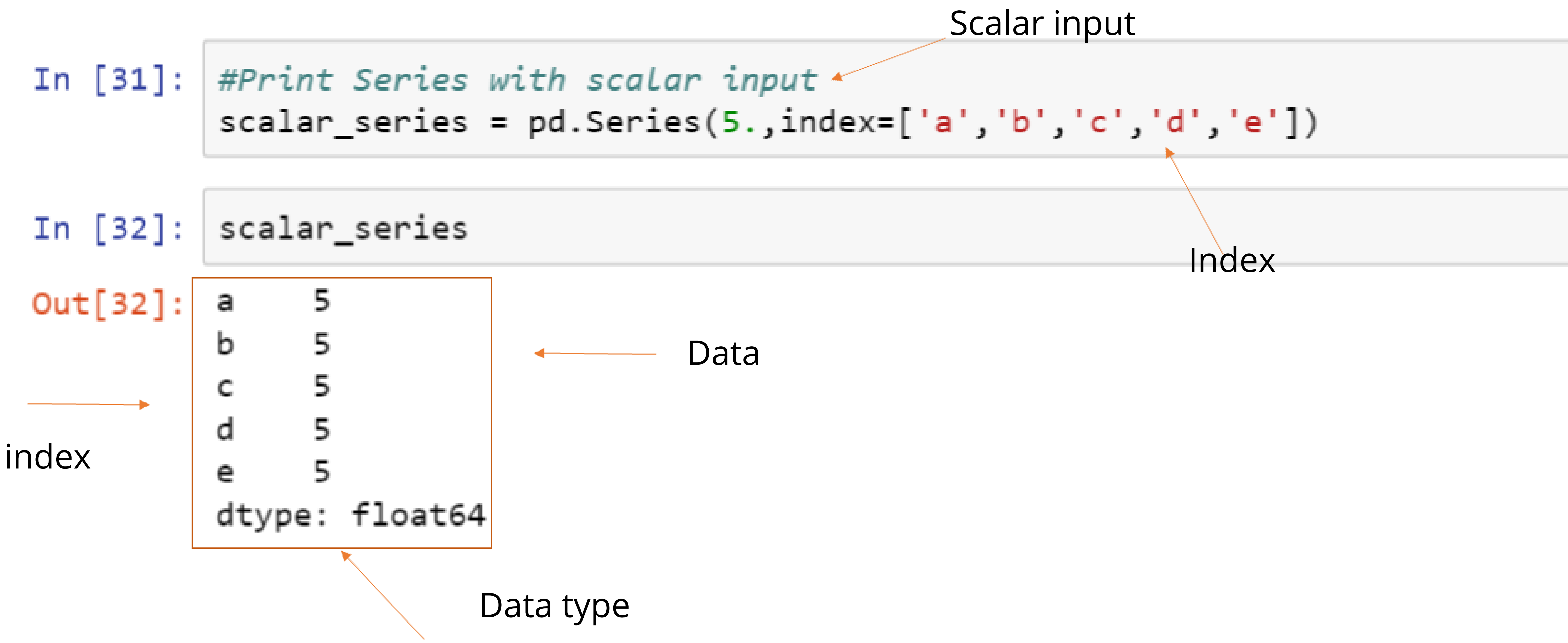
Countries have been passed as an index and GDP as the actual data value

GDP

Country

Data type

Creating Series from Scalar



```
In [31]: #Print Series with scalar input
         scalar_series = pd.Series(5.,index=['a','b','c','d','e'])

In [32]: scalar_series

Out[32]: a     5
         b     5
         c     5
         d     5
         e     5
         dtype: float64
```

Scalar input

Index

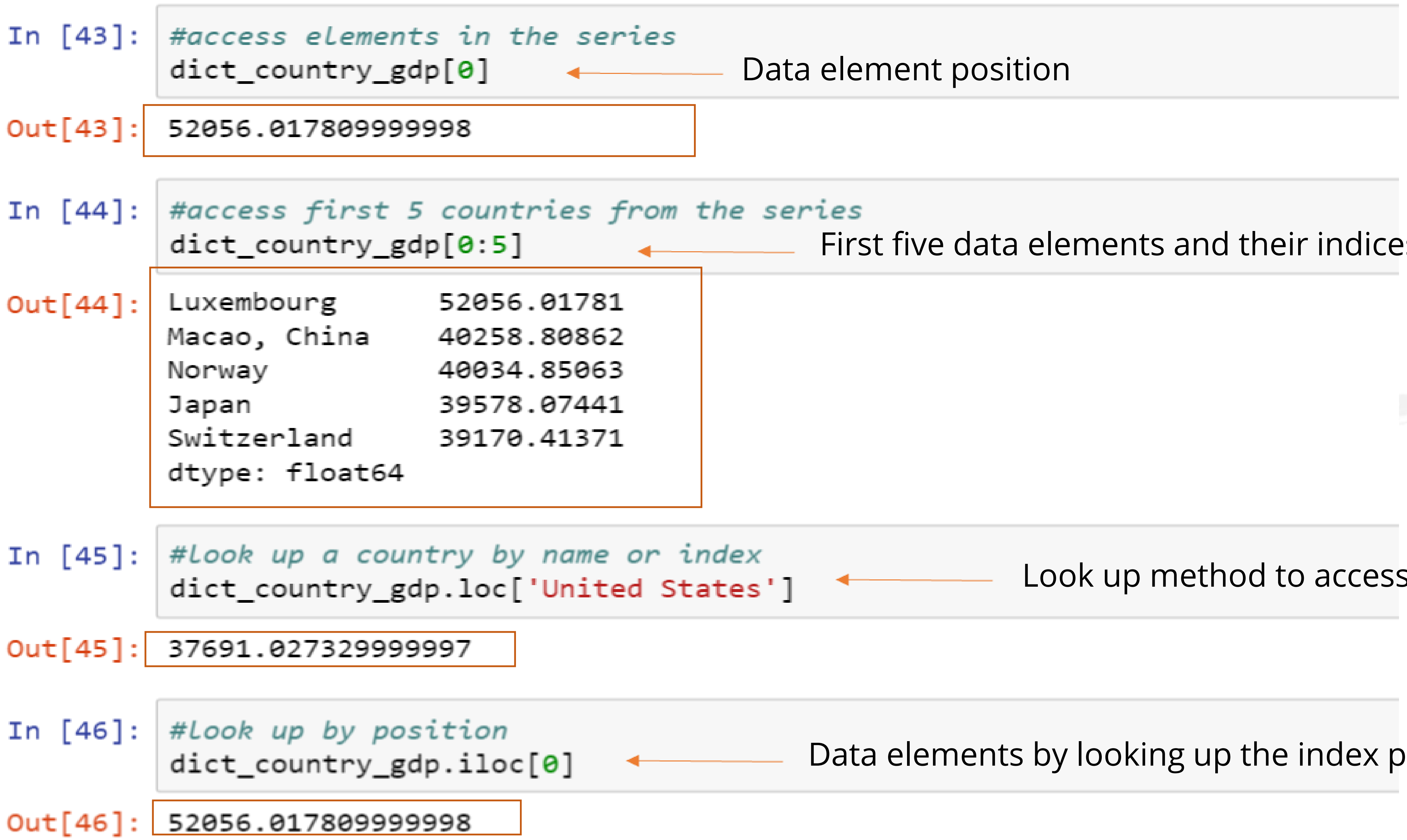
Data

index

Data type

Accessing Elements in Series

Data can be accessed through different functions like loc and iloc, by passing data element position or index range.



```
In [43]:  #access elements in the series
          dict_country_gdp[0]

Out[43]:  52056.017809999998
```

Data element position

```
In [44]:  #access first 5 countries from the series
          dict_country_gdp[0:5]

Out[44]:  Luxembourg       52056.01781
          Macao, China     40258.80862
          Norway           40034.85063
          Japan            39578.07441
          Switzerland      39170.41371
          dtype: float64
```

First five data elements and their indices

```
In [45]:  #look up a country by name or index
          dict_country_gdp.loc['United States']

Out[45]:  37691.027329999997
```

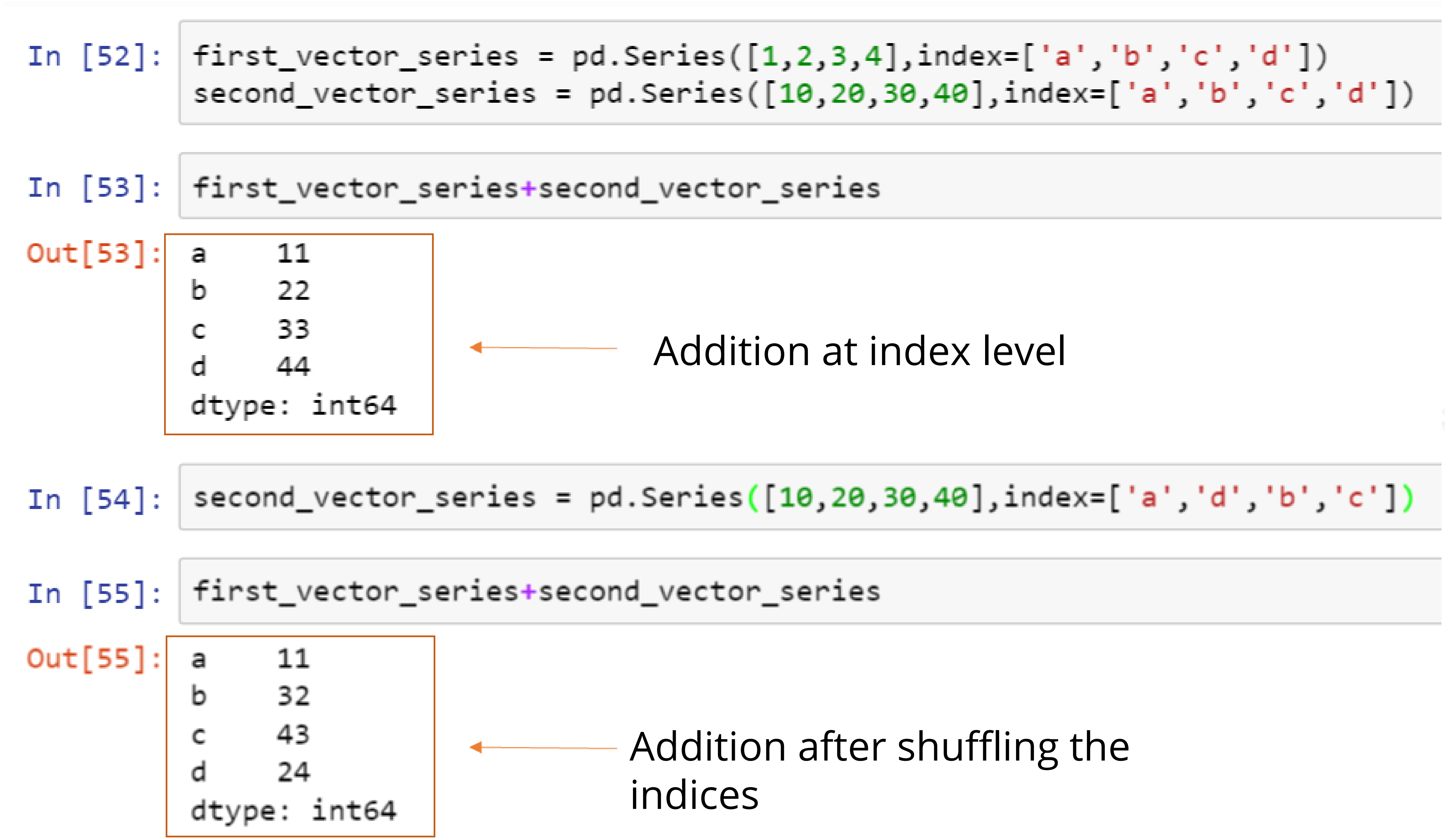
Look up method to access data

```
In [46]:  #look up by position
          dict_country_gdp.iloc[0]

Out[46]:  52056.017809999998
```

Data elements by looking up the index position

Vectorized Operations in Series

Vectorized operations are performed by the data element's position.



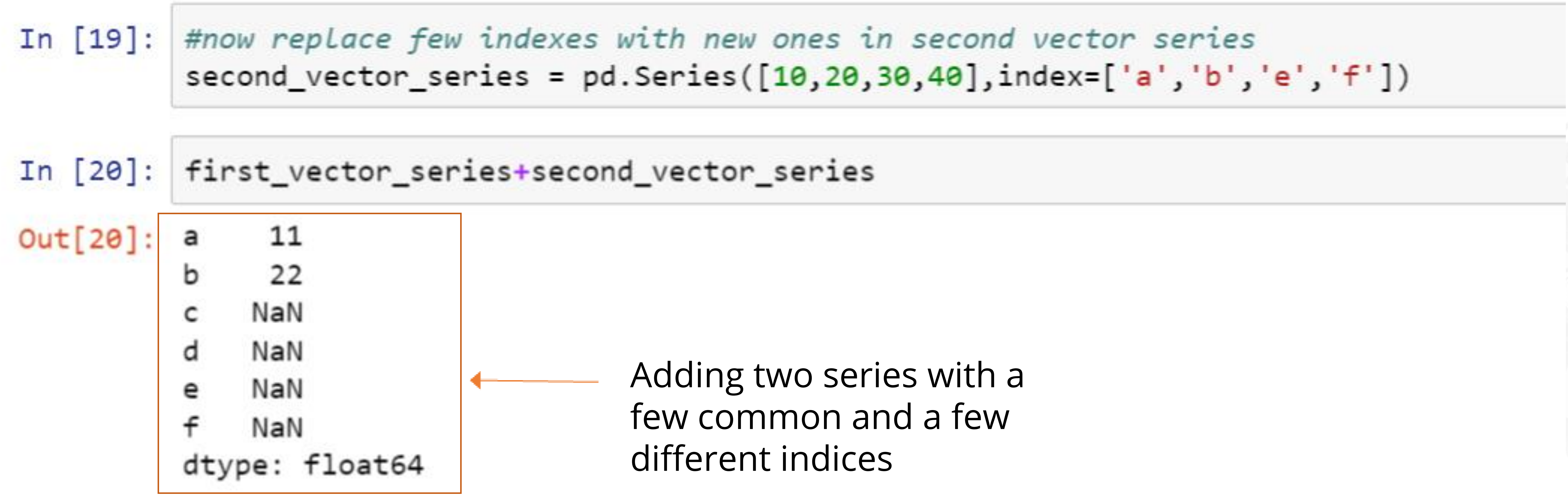
```
In [52]: first_vector_series = pd.Series([1,2,3,4],index=['a','b','c','d'])
         second_vector_series = pd.Series([10,20,30,40],index=['a','b','c','d'])

In [53]: first_vector_series+second_vector_series

Out[53]: a     11
         b     22
         c     33
         d     44
         dtype: int64

In [54]: second_vector_series = pd.Series([10,20,30,40],index=['a','d','b','c'])

In [55]: first_vector_series+second_vector_series

Out[55]: a     11
         b     32
         c     43
         d     24
         dtype: int64
```

Add the series

Addition at index level

Addition after shuffling the indices

Vectorized Operations in Series



```
In [19]: #now replace few indexes with new ones in second vector series
         second_vector_series = pd.Series([10,20,30,40],index=['a','b','e','f'])

In [20]: first_vector_series+second_vector_series

Out[20]: a     11
         b     22
         c    NaN
         d    NaN
         e    NaN
         f    NaN
         dtype: float64
```

Adding two series with a few common and a few different indices

Create Pandas Series

Objective: Create Pandas Series to input employee names and assign employee IDs to each employee. Also, retrieve first five employees from the Pandas Series using their IDs.

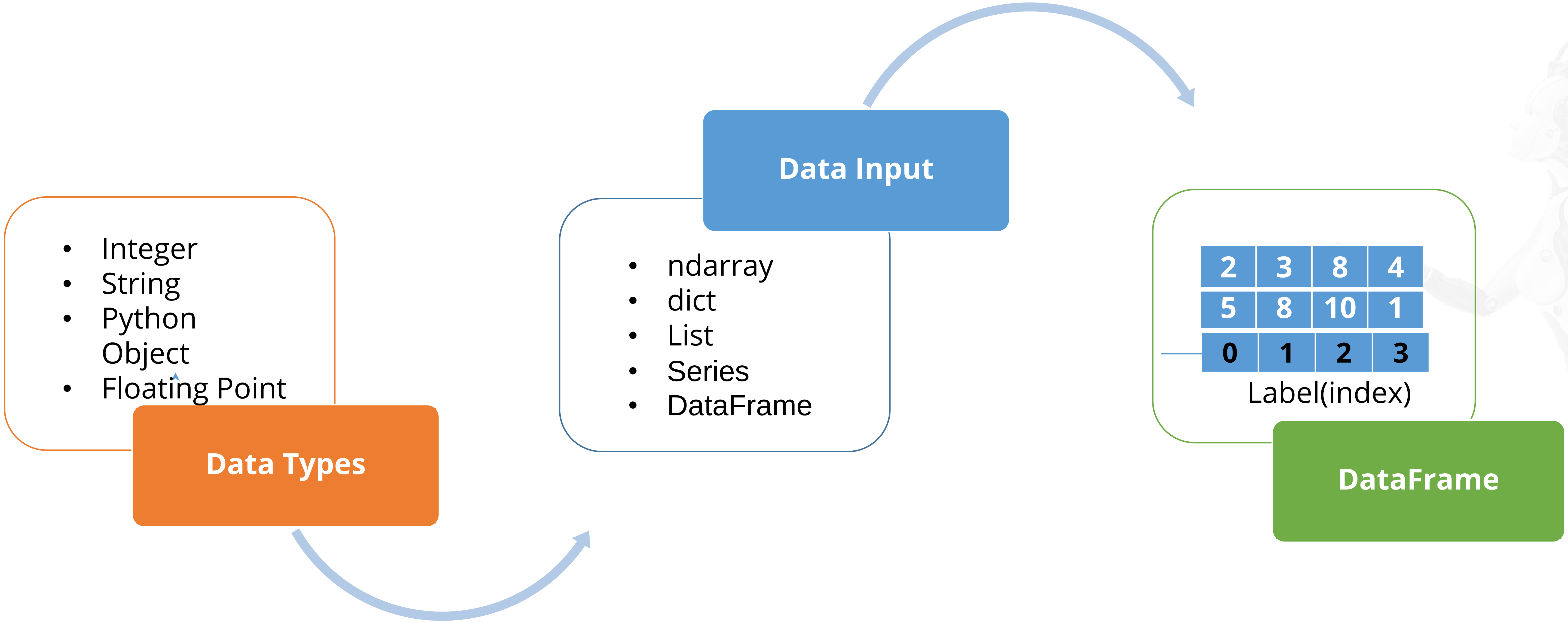
Access: To execute the practice, follow these steps:

- Go to the **PRACTICE LABS** tab on your LMS
- Click the **START LAB** button
- Click the **LAUNCH LAB** button to start the lab

DataFrames

DataFrame

DataFrame is a two-dimensional labeled data structure with columns of potentially different types.



- Integer
- String
- Python Object
- Floating Point

Data Types

Data Input

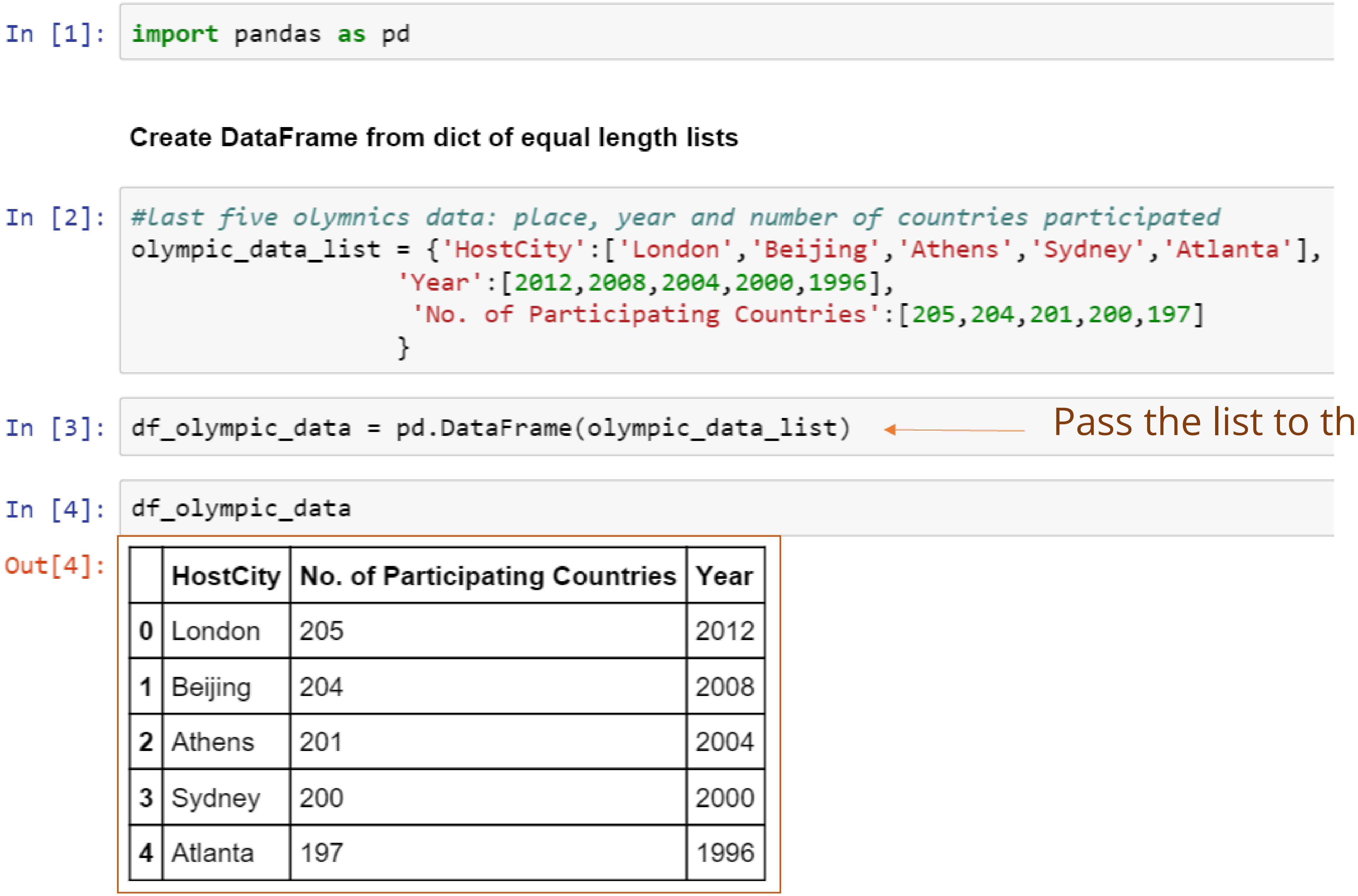
- ndarray
- dict
- List
- Series
- DataFrame

2	3	8	4
5	8	10	1
0	**1**	**2**	**3**

Label(index)

DataFrame

Creating DataFrame from Lists



```
In [1]: import pandas as pd
```

Create DataFrame from dict of equal length lists

```
In [2]: #last five olymnics data: place, year and number of countries participated
        olympic_data_list = {'HostCity':['London','Beijing','Athens','Sydney','Atlanta'],
                             'Year':[2012,2008,2004,2000,1996],
                              'No. of Participating Countries':[205,204,201,200,197]
                             }
```

```
In [3]: df_olympic_data = pd.DataFrame(olympic_data_list)
```

Pass the list to the DataFrame

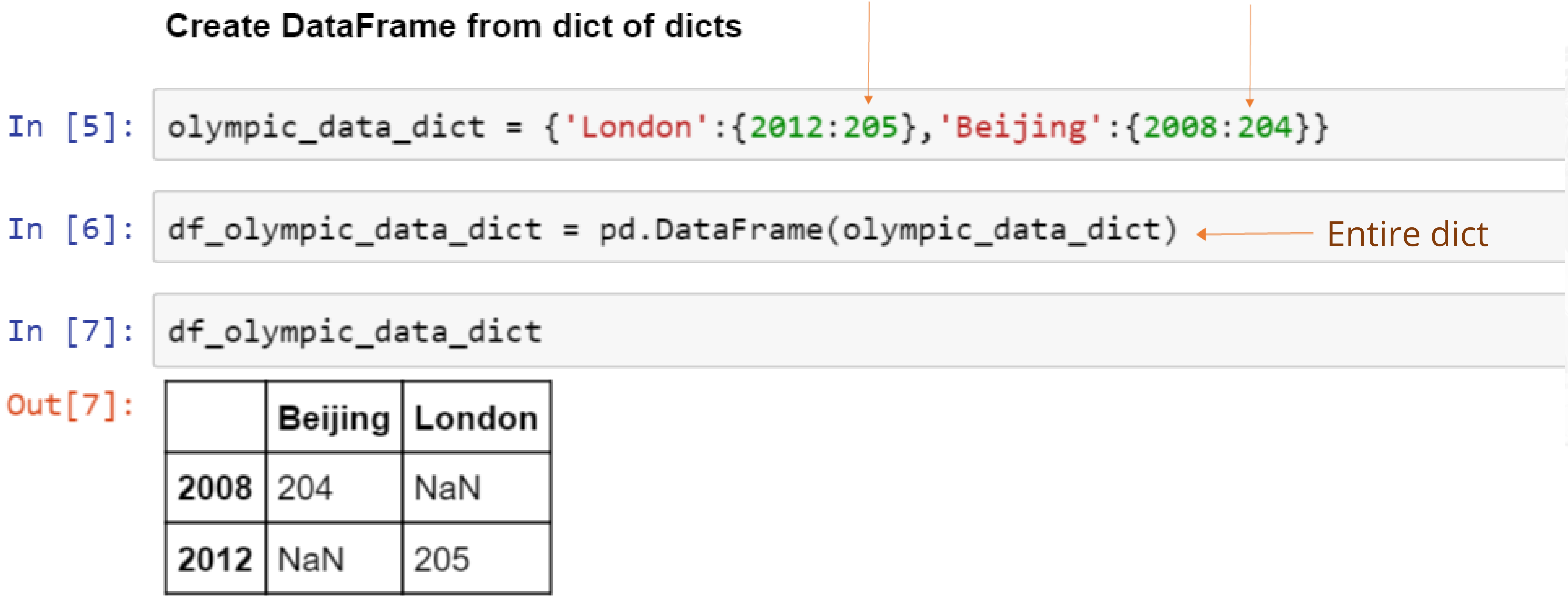
```
In [4]: df_olympic_data
```

Out[4]:

	HostCity	No. of Participating Countries	Year
0	London	205	2012
1	Beijing	204	2008
2	Athens	201	2004
3	Sydney	200	2000
4	Atlanta	197	1996

Creating DataFrame from dict

This example shows you how to create a DataFrame from a series of dicts:



Create DataFrame from dict of dicts

```
In [5]: olympic_data_dict = {'London':{2012:205},'Beijing':{2008:204}}

In [6]: df_olympic_data_dict = pd.DataFrame(olympic_data_dict)

In [7]: df_olympic_data_dict

Out[7]:
```

	Beijing	London
2008	204	NaN
2012	NaN	205

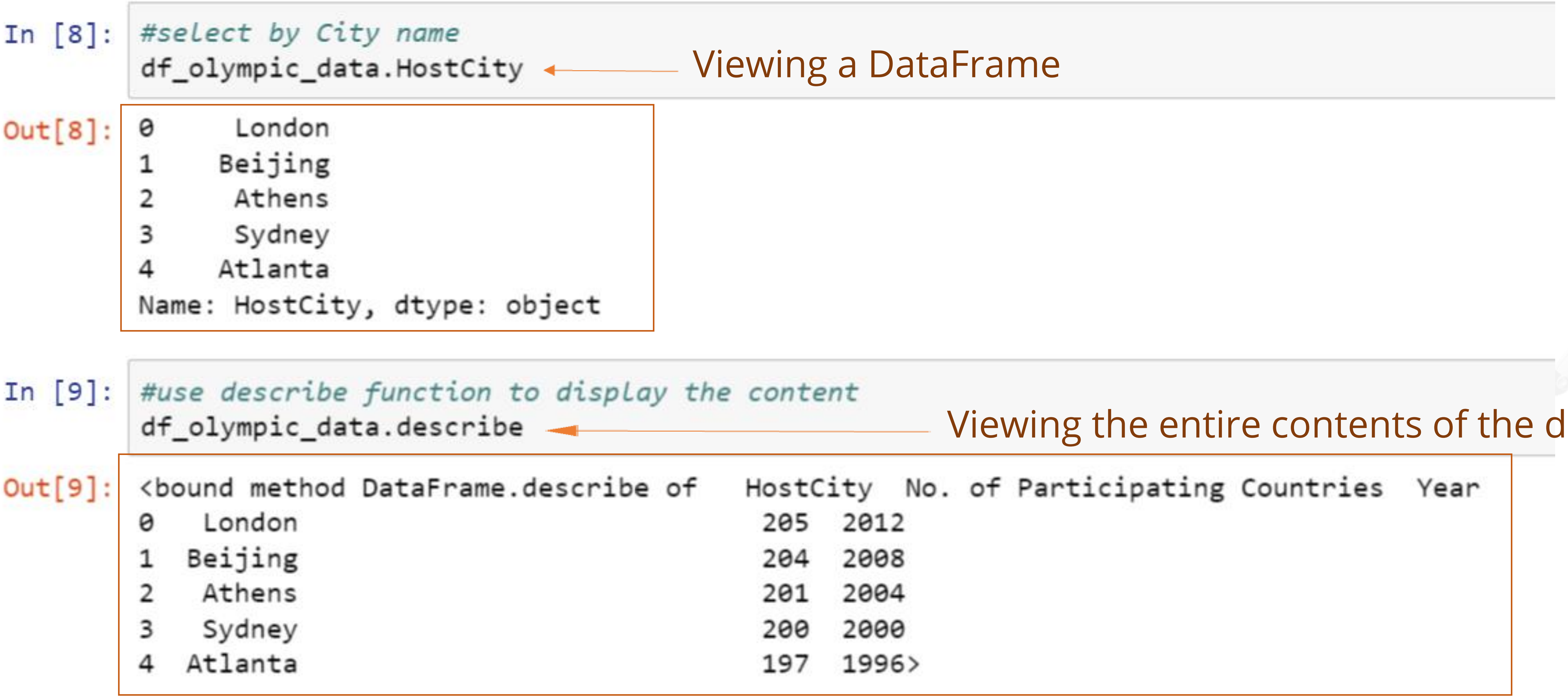
dict one

dict two

Entire dict

Viewing DataFrame

You can view a DataFrame by referring to the column name or the describe function.



```
In [8]:  #select by City name
         df_olympic_data.HostCity
```

Viewing a DataFrame

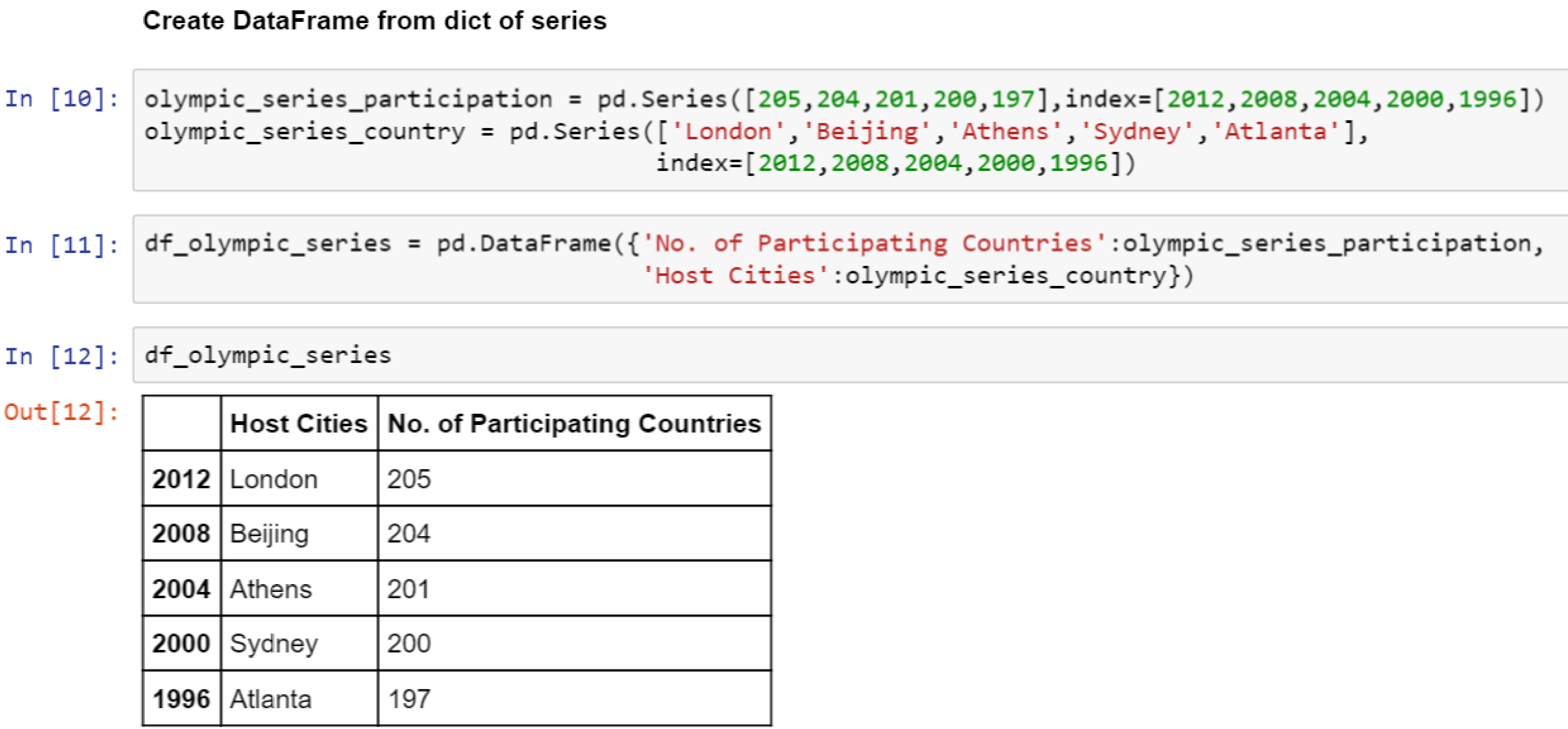
```
Out[8]:  0      London
         1     Beijing
         2      Athens
         3      Sydney
         4     Atlanta
         Name: HostCity, dtype: object
```

```
In [9]:  #use describe function to display the content
         df_olympic_data.describe
```

Viewing the entire contents of the dataset

```
Out[9]:  <bound method DataFrame.describe of    HostCity  No. of Participating Countries  Year
         0    London                               205  2012
         1   Beijing                               204  2008
         2    Athens                               201  2004
         3    Sydney                               200  2000
         4   Atlanta                               197  1996>
```

Creating DataFrame from dict of Series



Create DataFrame from dict of series

```
In [10]: olympic_series_participation = pd.Series([205,204,201,200,197],index=[2012,2008,2004,2000,1996])
         olympic_series_country = pd.Series(['London','Beijing','Athens','Sydney','Atlanta'],
                                           index=[2012,2008,2004,2000,1996])
```

```
In [11]: df_olympic_series = pd.DataFrame({'No. of Participating Countries':olympic_series_participation,
                                          'Host Cities':olympic_series_country})
```

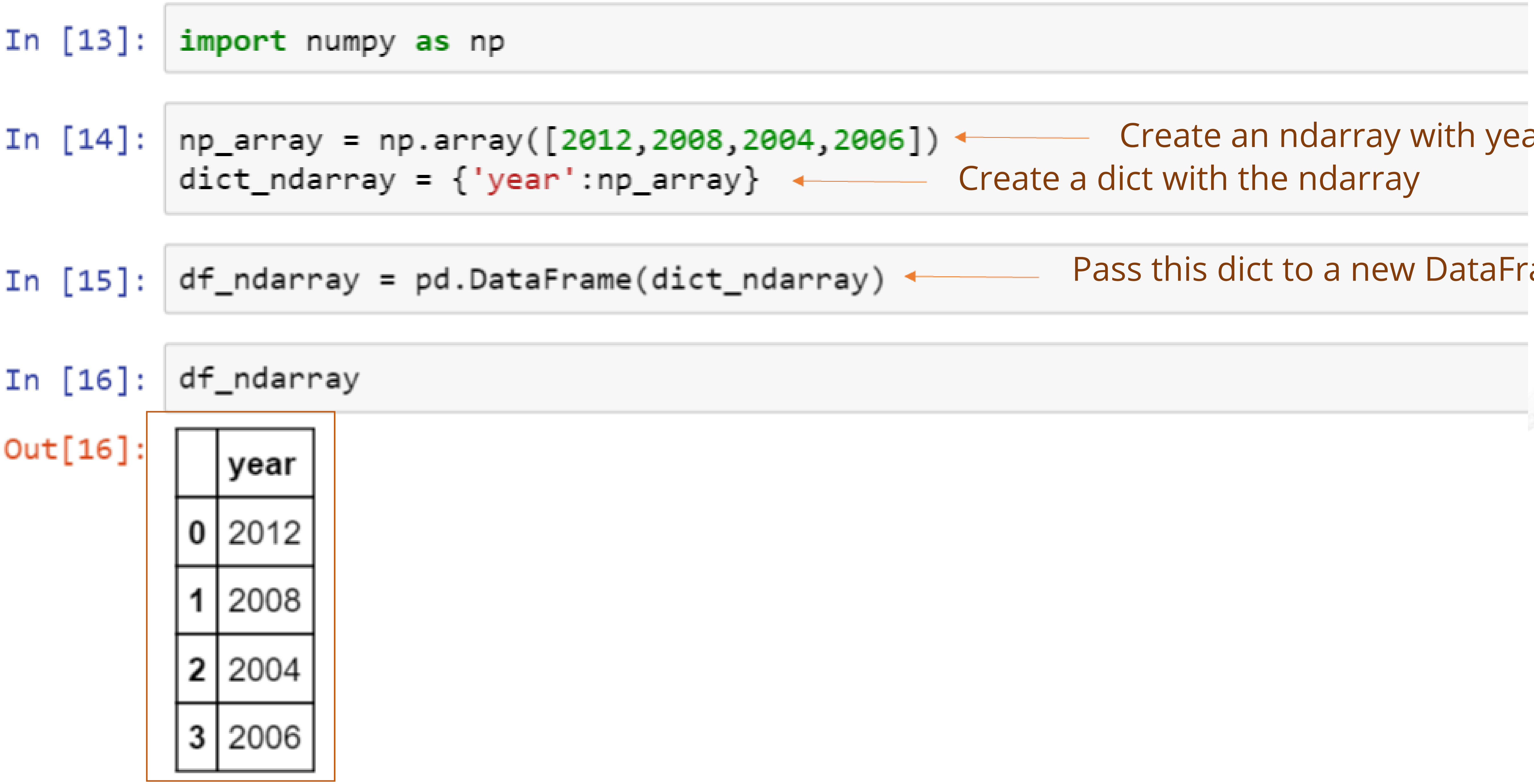
```
In [12]: df_olympic_series
```

Out[12]:

	Host Cities	No. of Participating Countries
2012	London	205
2008	Beijing	204
2004	Athens	201
2000	Sydney	200
1996	Atlanta	197

Creating DataFrame from ndarray

Create DataFrame from dict of ndarray



```
In [13]: import numpy as np

In [14]: np_array = np.array([2012,2008,2004,2006])
         dict_ndarray = {'year':np_array}

In [15]: df_ndarray = pd.DataFrame(dict_ndarray)

In [16]: df_ndarray
```

Out[16]:

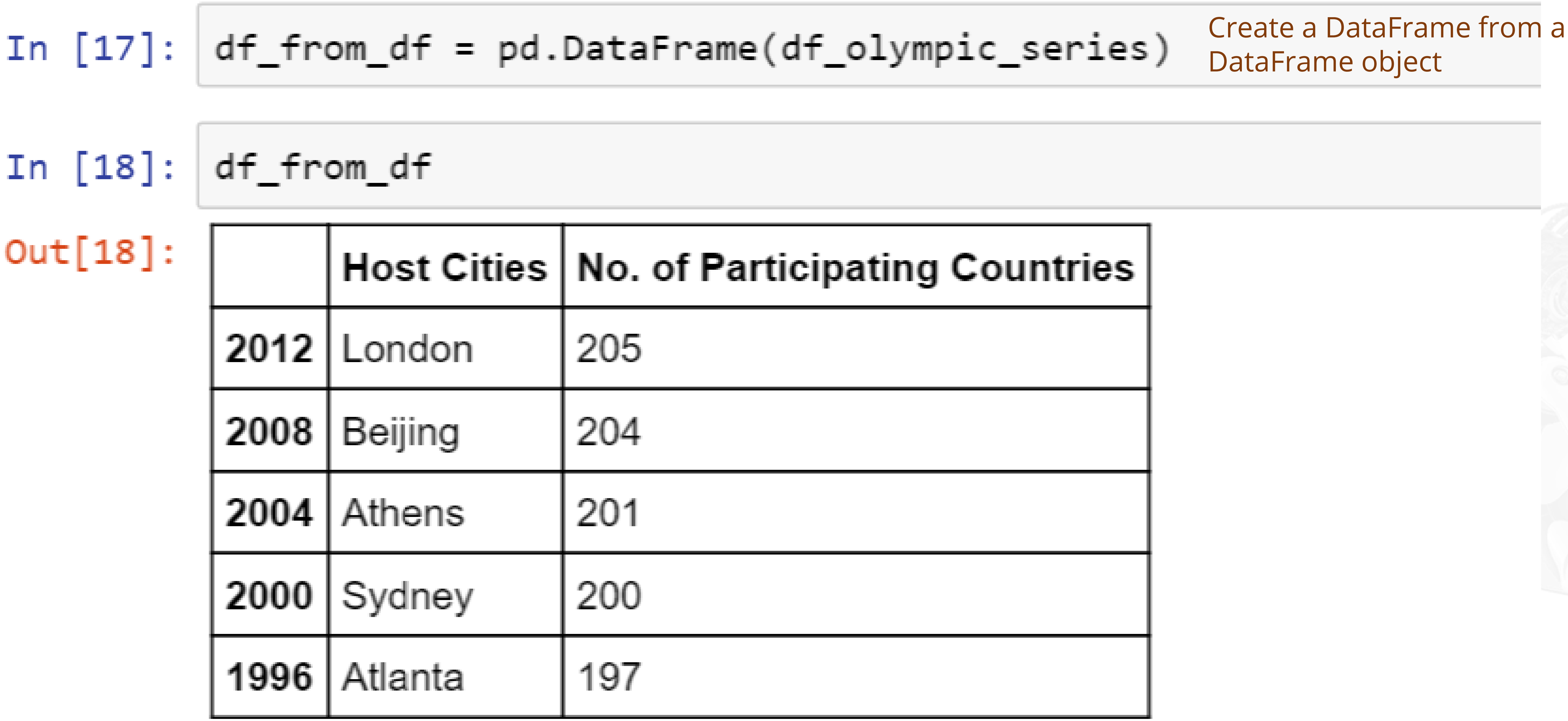
	year
0	2012
1	2008
2	2004
3	2006

Create an ndarray with years

Create a dict with the ndarray

Pass this dict to a new DataFrame

Creating DataFrame from DataFrame Object



```
In [17]: df_from_df = pd.DataFrame(df_olympic_series)
```

Create a DataFrame from a DataFrame object

```
In [18]: df_from_df
```

Out[18]:

	Host Cities	No. of Participating Countries
2012	London	205
2008	Beijing	204
2004	Athens	201
2000	Sydney	200
1996	Atlanta	197

Create Pandas DataFrames

Objective: Create DataFrames in the following conditions:

1. Input the marks of two subjects for five students and create a DataFrame using the two series
2. Extract data from the given SalaryGender CSV file and store the data from each column in a separate NumPy array
3. Create a DataFrame using dictionary of names and age of five people as input
4. Create a DataFrame from the dictionary of series

Access: To execute the practice, follow these steps:

- Go to the **PRACTICE LABS** tab on your LMS
- Click the **START LAB** button
- Click the **LAUNCH LAB** button to start the lab

ASSISTED PRACTICE

Create Pandas DataFrames

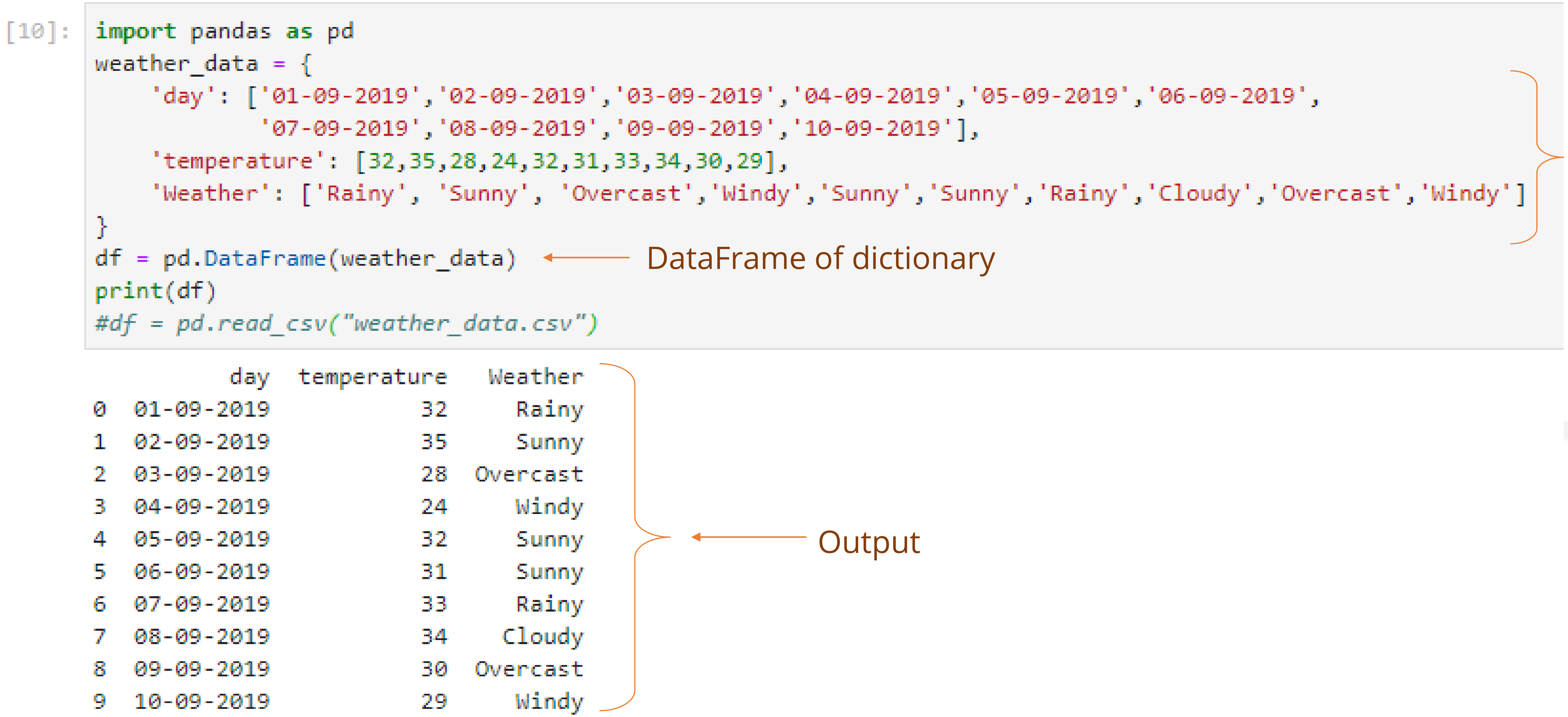
Objective: Create the following dataframes:

1. A DataFrame of weather data using dictionary with the keys like day, temperature, and weather conditions
2. Create a DataFrame with a list of dictionaries, rows, and columns

Access: To execute the practice, follow these steps:

- Go to the **PRACTICE LABS** tab on your LMS
- Click the **START LAB** button
- Click the **LAUNCH LAB** button to start the lab

Unassisted Practices: Create Pandas DataFrame



```
[10]: import pandas as pd
      weather_data = {
          'day': ['01-09-2019','02-09-2019','03-09-2019','04-09-2019','05-09-2019','06-09-2019',
                  '07-09-2019','08-09-2019','09-09-2019','10-09-2019'],
          'temperature': [32,35,28,24,32,31,33,34,30,29],
          'Weather': ['Rainy', 'Sunny', 'Overcast','Windy','Sunny','Sunny','Rainy','Cloudy','Overcast','Windy']
      }
      df = pd.DataFrame(weather_data)
      print(df)
      #df = pd.read_csv("weather_data.csv")
```

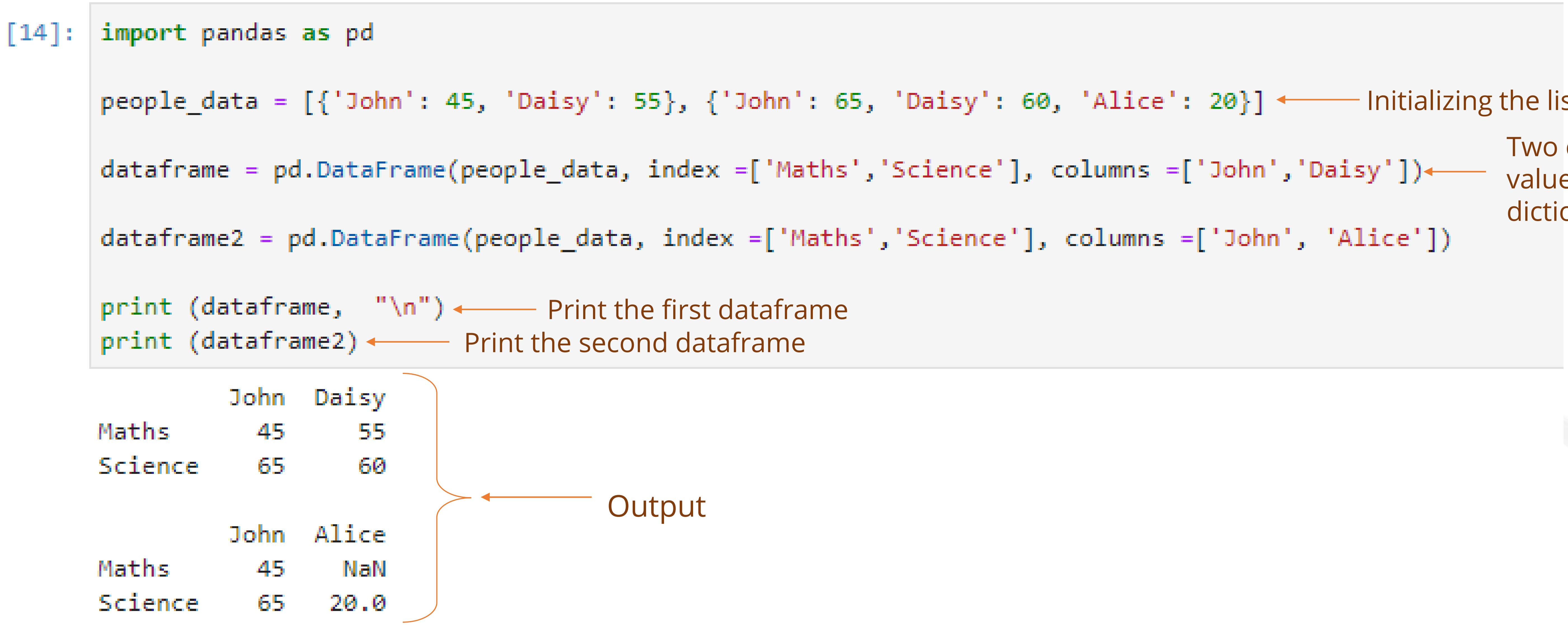
```
          day  temperature   Weather
0  01-09-2019           32     Rainy
1  02-09-2019           35     Sunny
2  03-09-2019           28  Overcast
3  04-09-2019           24     Windy
4  05-09-2019           32     Sunny
5  06-09-2019           31     Sunny
6  07-09-2019           33     Rainy
7  08-09-2019           34    Cloudy
8  09-09-2019           30  Overcast
9  10-09-2019           29     Windy
```

Dictionary of list of weather data

DataFrame of dictionary

Output

Unassisted Practices: Create Pandas DataFrame



```
[14]: import pandas as pd

      people_data = [{'John': 45, 'Daisy': 55}, {'John': 65, 'Daisy': 60, 'Alice': 20}]

      dataframe = pd.DataFrame(people_data, index =['Maths','Science'], columns =['John','Daisy'])

      dataframe2 = pd.DataFrame(people_data, index =['Maths','Science'], columns =['John', 'Alice'])

      print (dataframe,  "\n")
      print (dataframe2)
```

```
         John  Daisy
Maths      45     55
Science    65     60

         John  Alice
Maths      45    NaN
Science    65   20.0
```

Initializing the list data

Two column indices of values same as the dictionary keys

Print the first dataframe

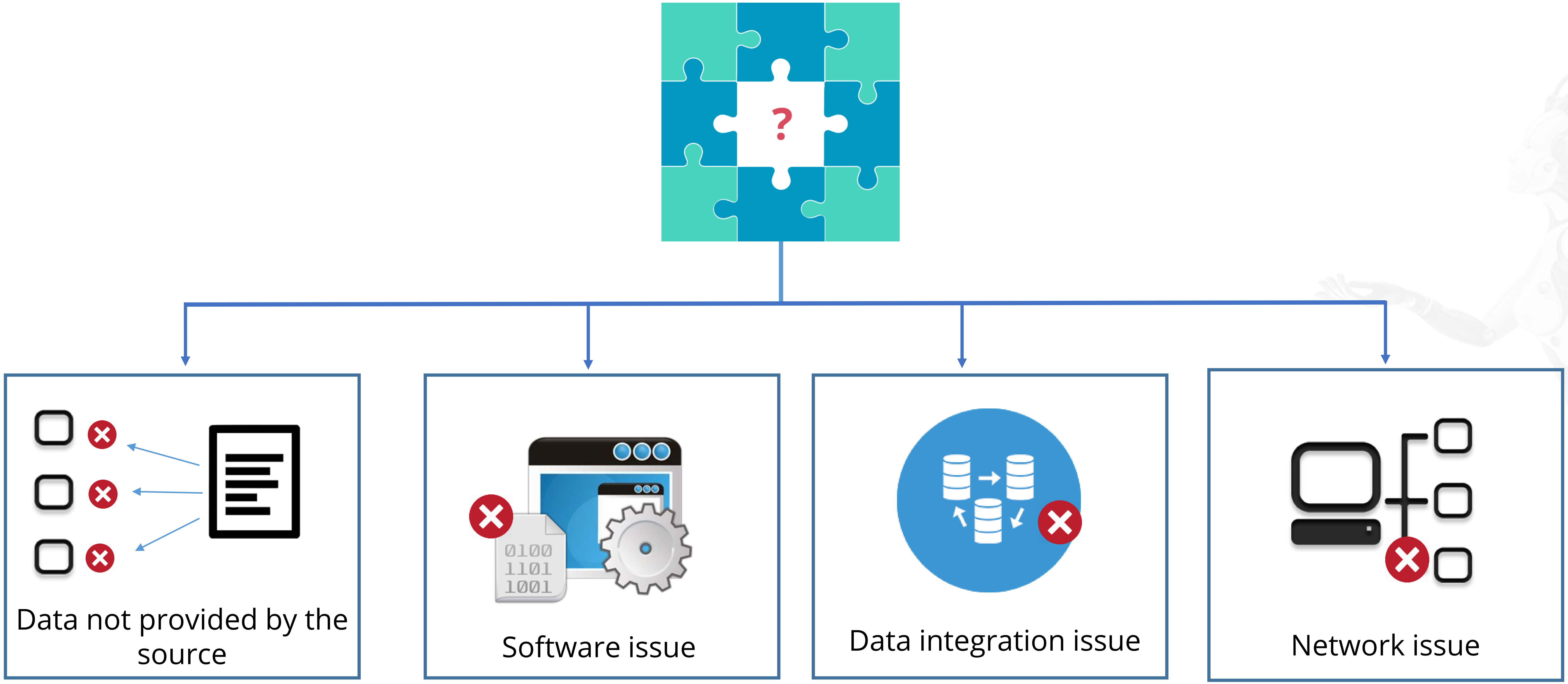
Print the second dataframe

Output

Missing Values

Missing Values

Various factors may lead to missing data values:



Data not provided by the source

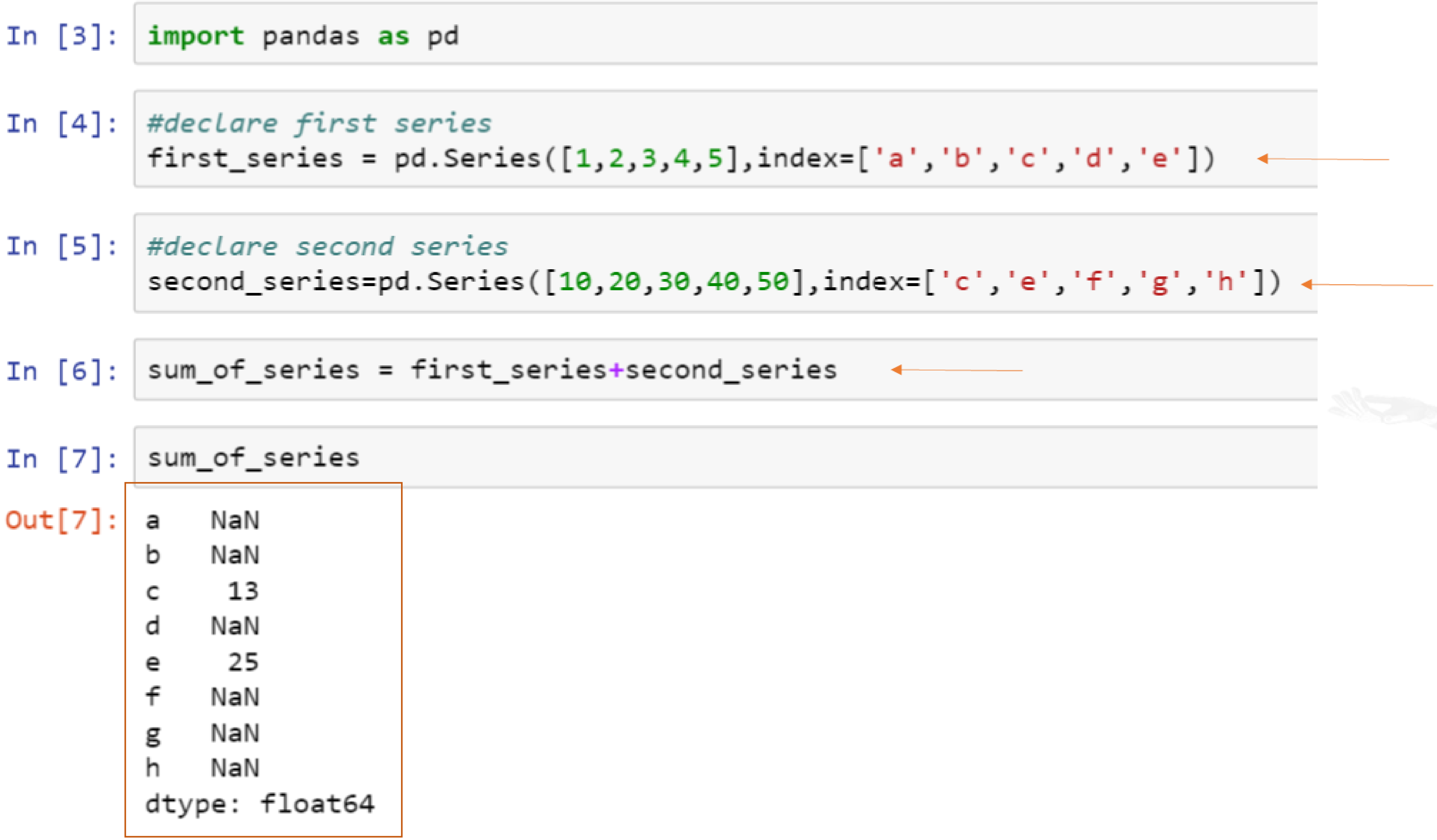
Software issue

Data integration issue

Network issue

Handling Missing Values

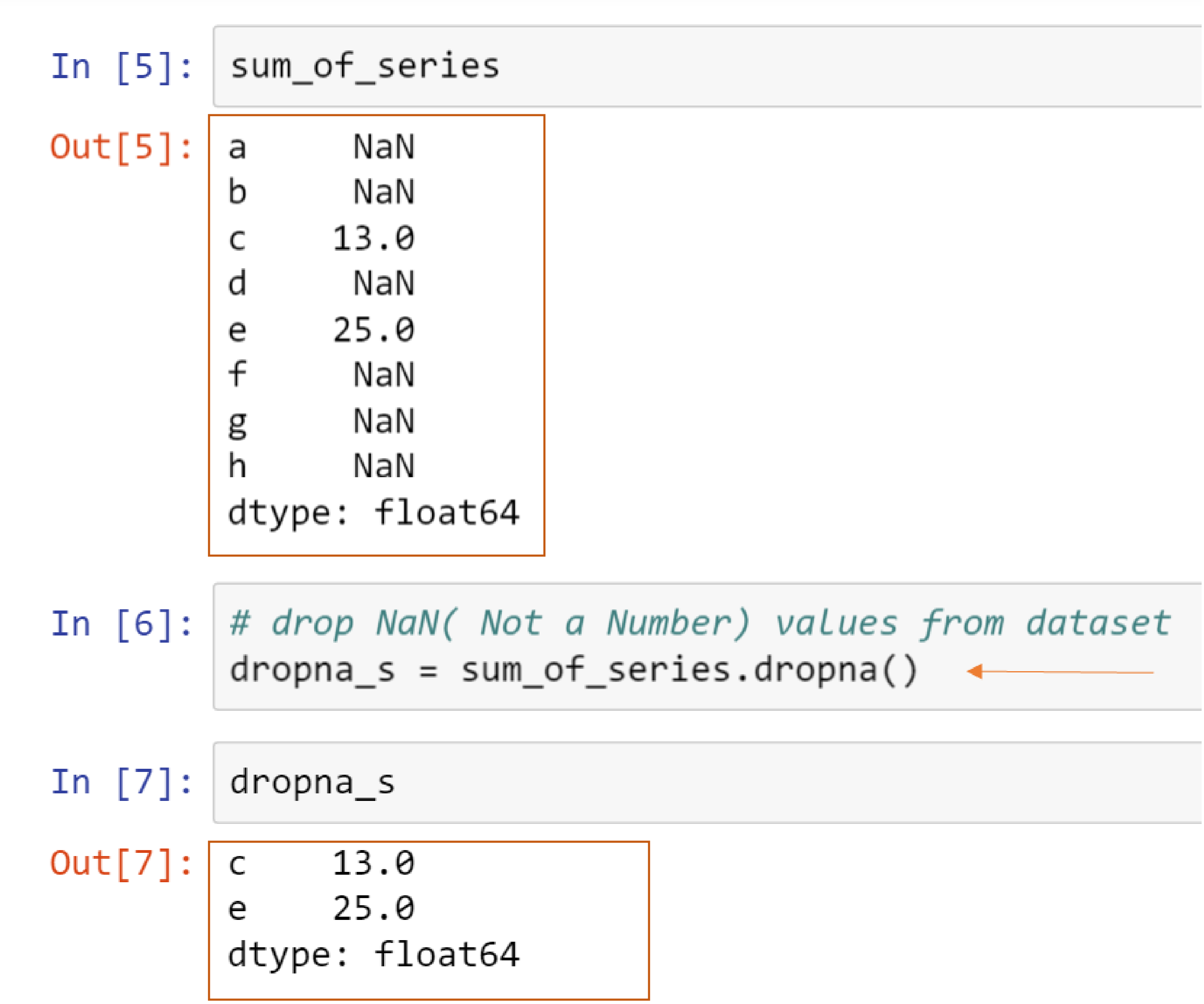
It’s difficult to operate a dataset when it has missing values or uncommon indices.



```
In [3]: import pandas as pd

In [4]: #declare first series
        first_series = pd.Series([1,2,3,4,5],index=['a','b','c','d','e'])

In [5]: #declare second series
        second_series=pd.Series([10,20,30,40,50],index=['c','e','f','g','h'])

In [6]: sum_of_series = first_series+second_series

In [7]: sum_of_series

Out[7]: a    NaN
        b    NaN
        c     13
        d    NaN
        e     25
        f    NaN
        g    NaN
        h    NaN
        dtype: float64
```

Handling Missing Values with Functions

The dropna function drops all the values with uncommon indices.

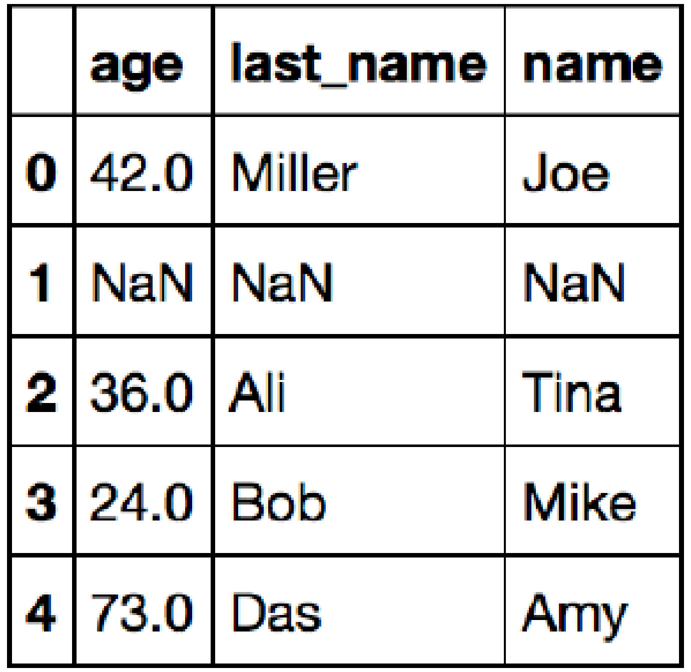


```
In [5]: sum_of_series

Out[5]: a      NaN
        b      NaN
        c     13.0
        d      NaN
        e     25.0
        f      NaN
        g      NaN
        h      NaN
        dtype: float64

In [6]: # drop NaN( Not a Number) values from dataset
        dropna_s = sum_of_series.dropna()

In [7]: dropna_s

Out[7]: c     13.0
        e     25.0
        dtype: float64
```

Handling Missing Values: Example

Code

```
import pandas as pd
import numpy      as np

Raw_data = {‘name’:[‘Joe’, np.nan, ‘Tina’, ‘Mike’, ‘Amy’],
              ‘last_name’:[‘Miller’, np.nan, 36,24,73]}
                   ‘AGE’:[42, np.nan, 36, 24,73]}
df =pd.DataFrame(raw_data)
df
```

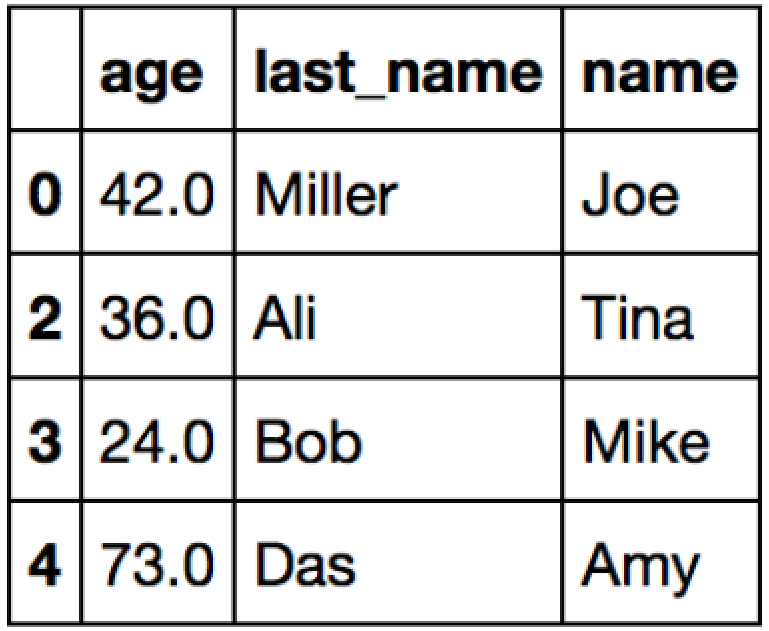


	age	last_name	name
0	42.0	Miller	Joe
1	NaN	NaN	NaN
2	36.0	Ali	Tina
3	24.0	Bob	Mike
4	73.0	Das	Amy

Handling Missing Values: Example

Code

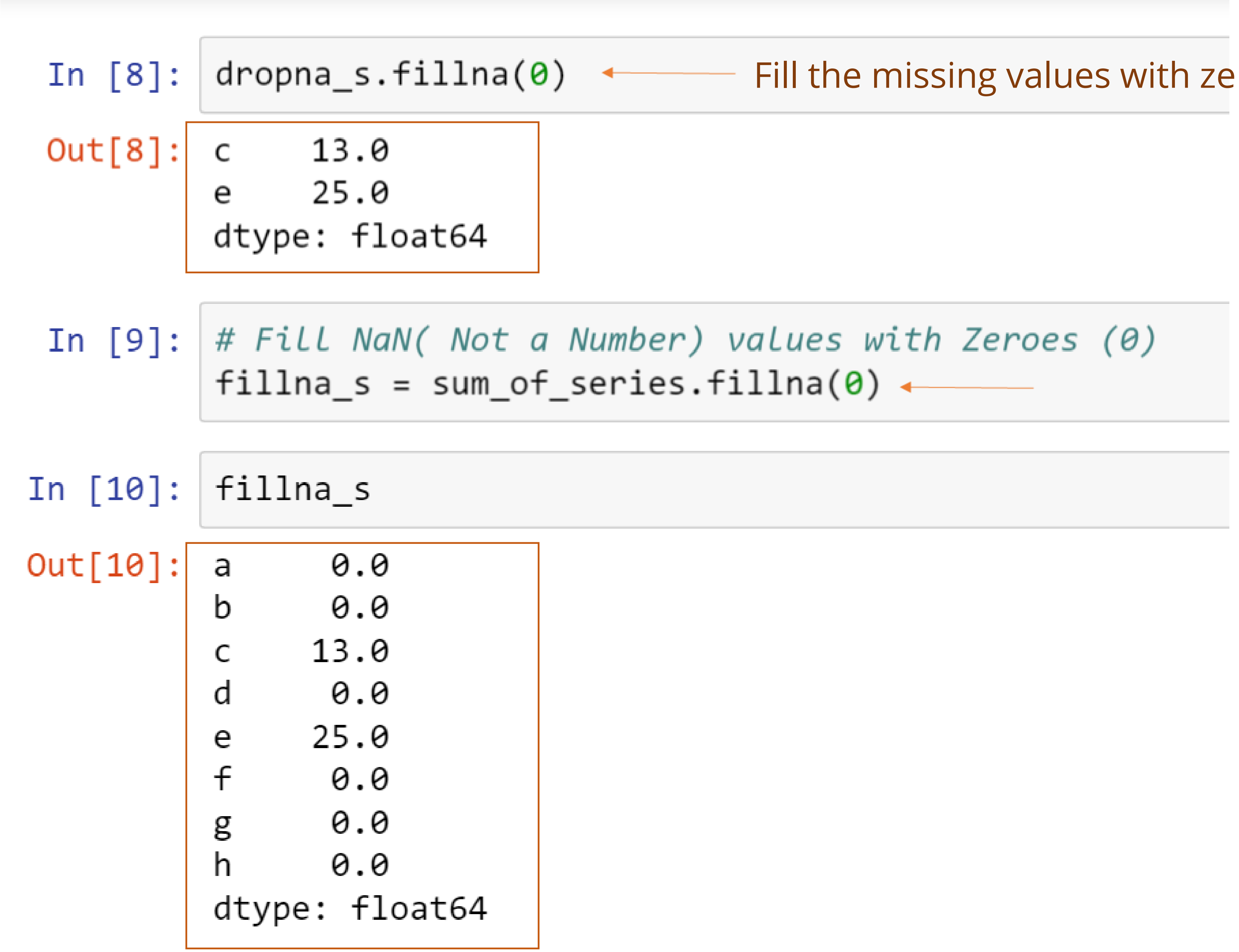
```
df.dropna()
```



	age	last_name	name
0	42.0	Miller	Joe
2	36.0	Ali	Tina
3	24.0	Bob	Mike
4	73.0	Das	Amy

Handling Missing Values with Functions

The fillna function fills all the uncommon indices with a number instead of dropping them.

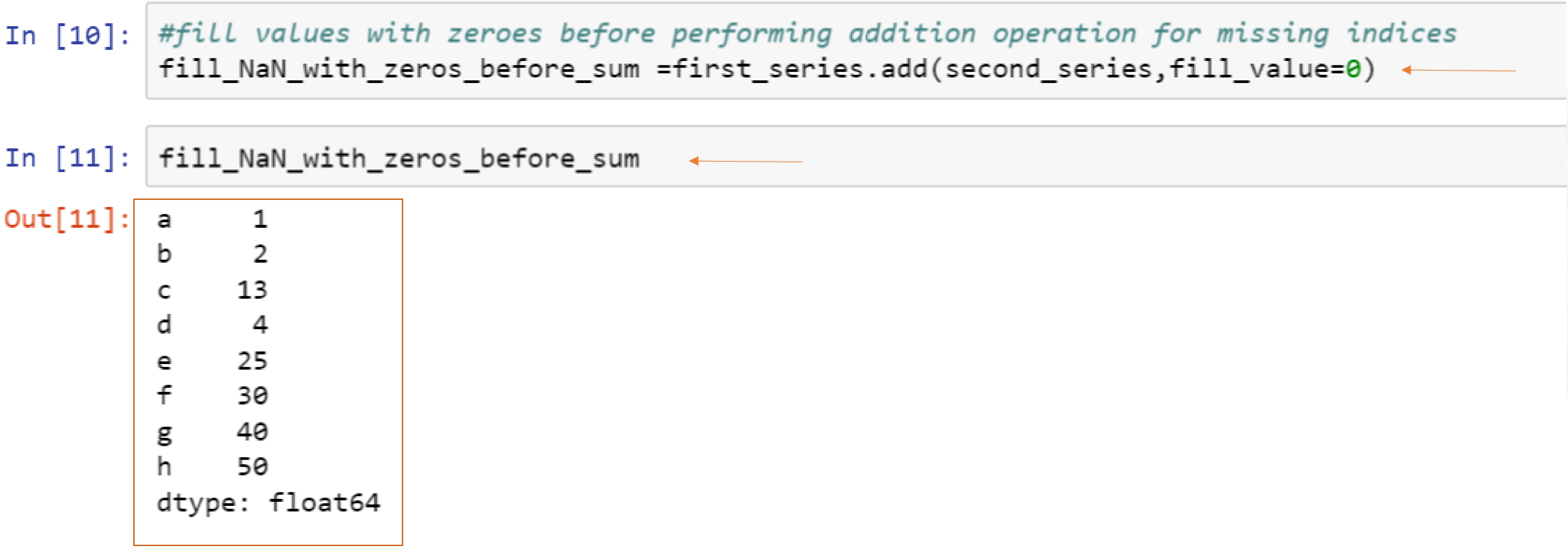


```
In [8]:  dropna_s.fillna(0)

Out[8]:  c     13.0
         e     25.0
         dtype: float64
```

Fill the missing values with zero

```
In [9]:  # Fill NaN( Not a Number) values with Zeroes (0)
         fillna_s = sum_of_series.fillna(0)

In [10]: fillna_s

Out[10]: a      0.0
         b      0.0
         c     13.0
         d      0.0
         e     25.0
         f      0.0
         g      0.0
         h      0.0
         dtype: float64
```

Handle Missing Values with Functions: Example



```
In [10]: #fill values with zeroes before performing addition operation for missing indices
         fill_NaN_with_zeros_before_sum =first_series.add(second_series,fill_value=0)

In [11]: fill_NaN_with_zeros_before_sum

Out[11]: a     1
         b     2
         c    13
         d     4
         e    25
         f    30
         g    40
         h    50
         dtype: float64
```

Handle Missing Values

Objective: Create a dataframe with a list of dictionaries, rows indices, and column indices with one index having a different name. Handle the missing values by:

1. Removing the NaN values
2. Filling all the uncommon or NaN values with a number, instead of dropping them

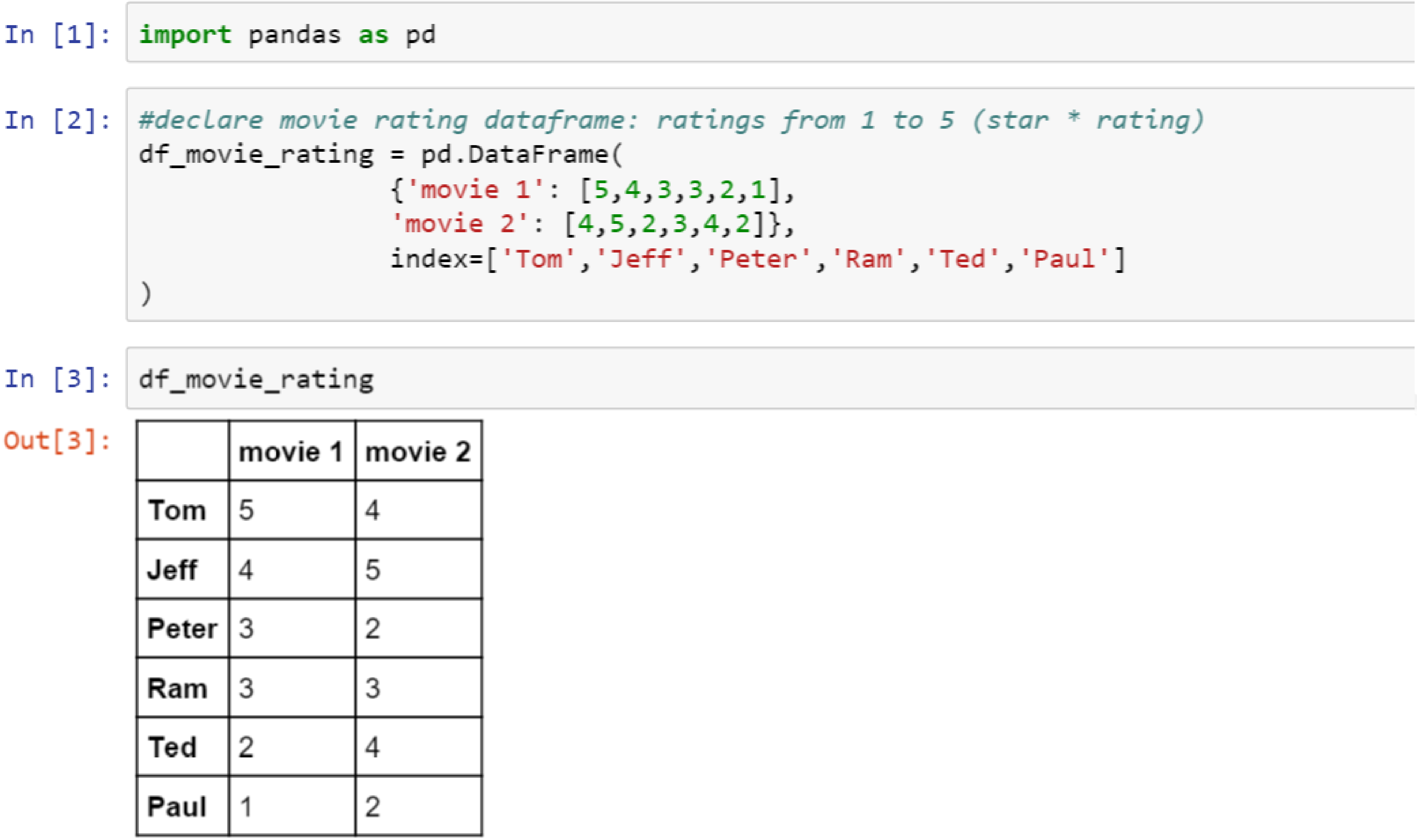
Access: To execute the practice, follow these steps:

- Go to the **PRACTICE LABS** tab on your LMS
- Click the **START LAB** button
- Click the **LAUNCH LAB** button to start the lab

Data Operation

Data Operation

Data operation can be performed through various built-in methods for faster data processing.



```
In [1]: import pandas as pd
```

```
In [2]: #declare movie rating dataframe: ratings from 1 to 5 (star * rating)
        df_movie_rating = pd.DataFrame(
                        {'movie 1': [5,4,3,3,2,1],
                        'movie 2': [4,5,2,3,4,2]},
                        index=['Tom','Jeff','Peter','Ram','Ted','Paul']
        )
```

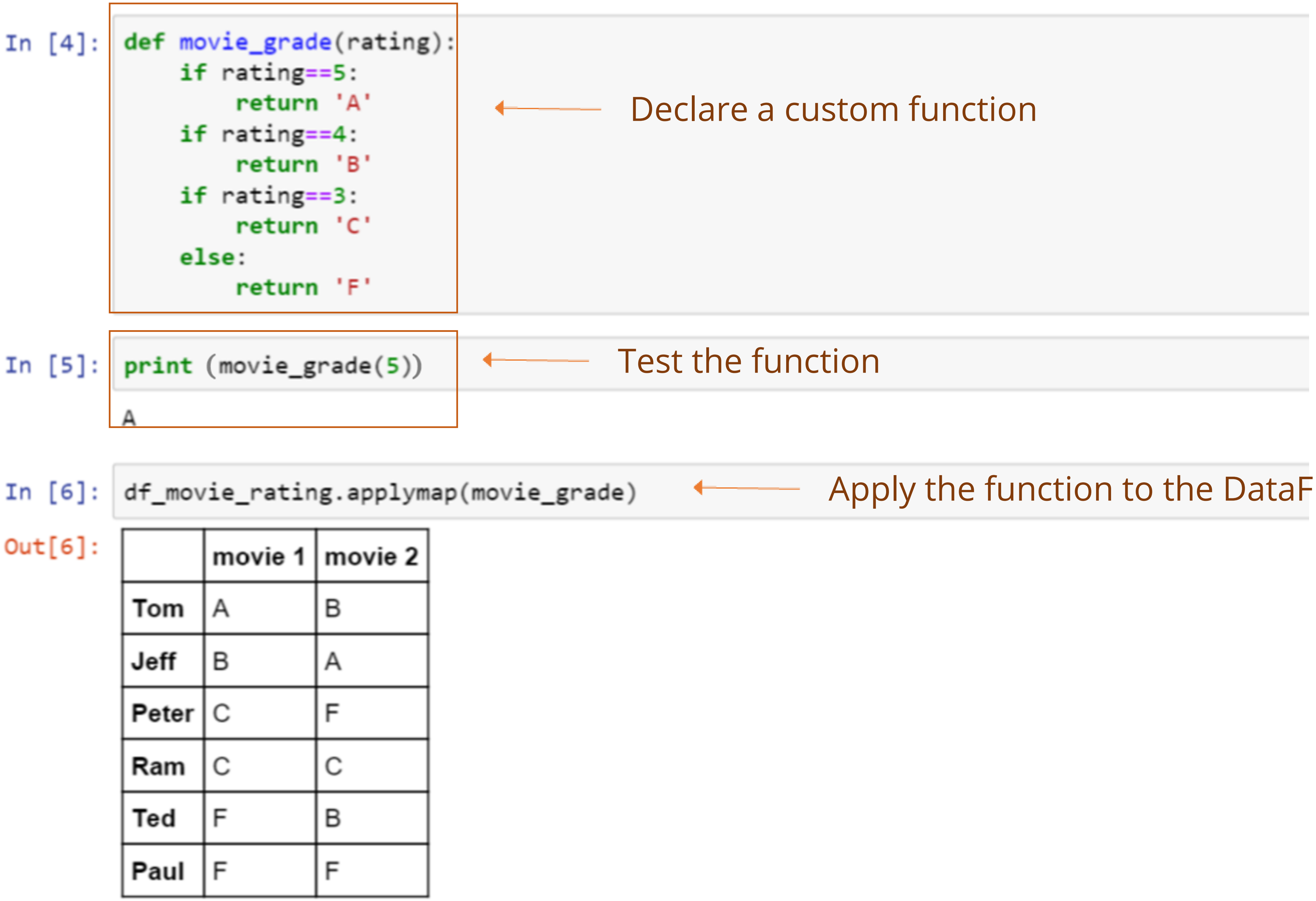
```
In [3]: df_movie_rating
```

Out[3]:

	movie 1	movie 2
Tom	5	4
Jeff	4	5
Peter	3	2
Ram	3	3
Ted	2	4
Paul	1	2

Data Operation with Functions

While performing data operation, custom functions can be applied using the applymap method.



```
In [4]: def movie_grade(rating):
            if rating==5:
                return 'A'
            if rating==4:
                return 'B'
            if rating==3:
                return 'C'
            else:
                return 'F'
```

Declare a custom function

```
In [5]: print (movie_grade(5))

A
```

Test the function

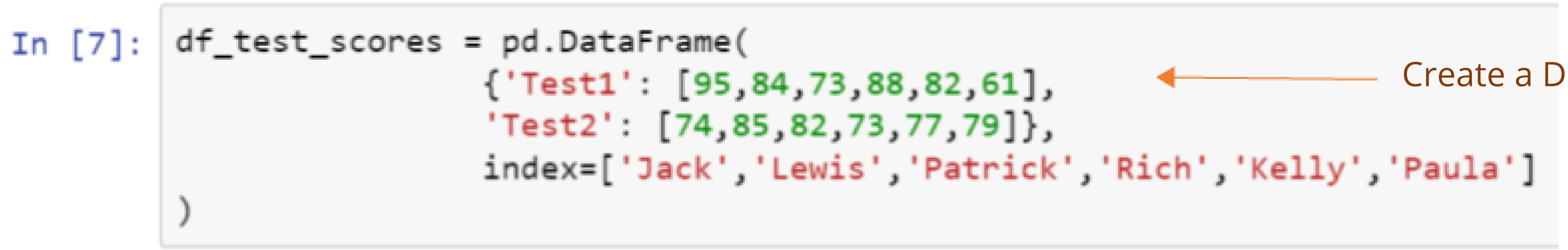
```
In [6]: df_movie_rating.applymap(movie_grade)
```

Apply the function to the DataFrame

Out[6]:

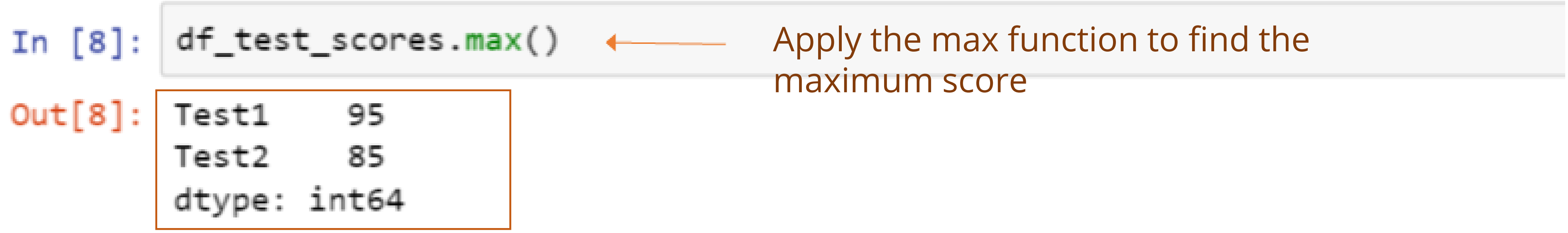
	movie 1	movie 2
Tom	A	B
Jeff	B	A
Peter	C	F
Ram	C	C
Ted	F	B
Paul	F	F

Data Operation with Statistical Functions



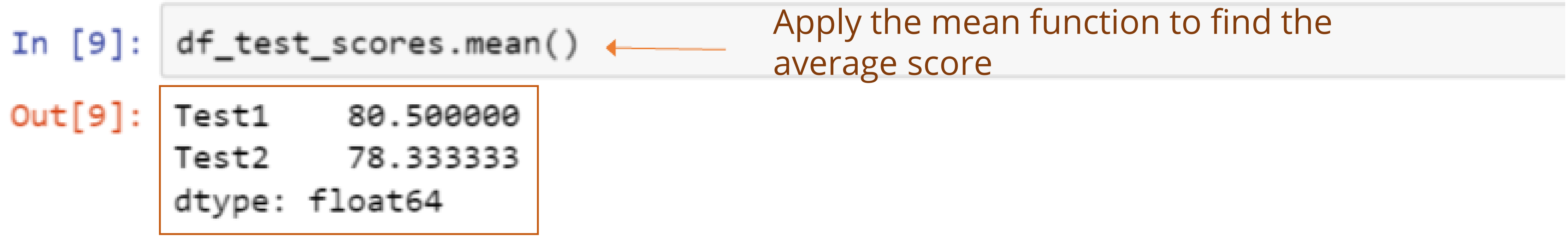
```
In [7]: df_test_scores = pd.DataFrame(
                        {'Test1': [95,84,73,88,82,61],
                        'Test2': [74,85,82,73,77,79]},
                        index=['Jack','Lewis','Patrick','Rich','Kelly','Paula']
        )
```

Create a DataFrame with two tests



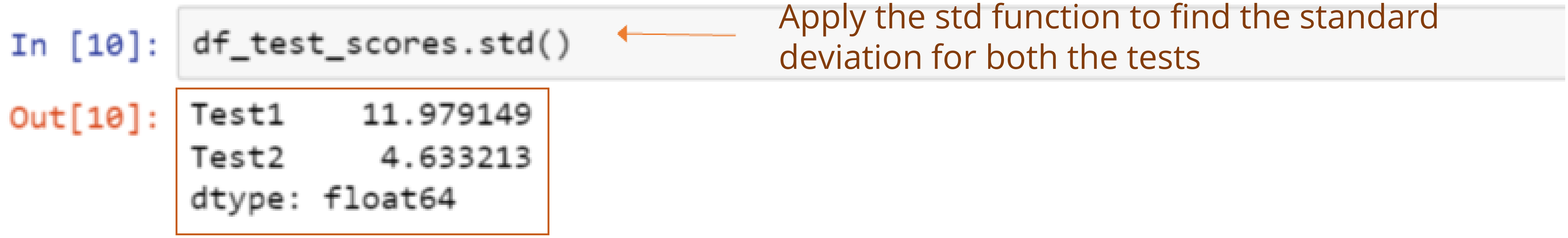
```
In [8]: df_test_scores.max()

Out[8]: Test1     95
        Test2     85
        dtype: int64
```

Apply the max function to find the maximum score



```
In [9]: df_test_scores.mean()

Out[9]: Test1     80.500000
        Test2     78.333333
        dtype: float64
```

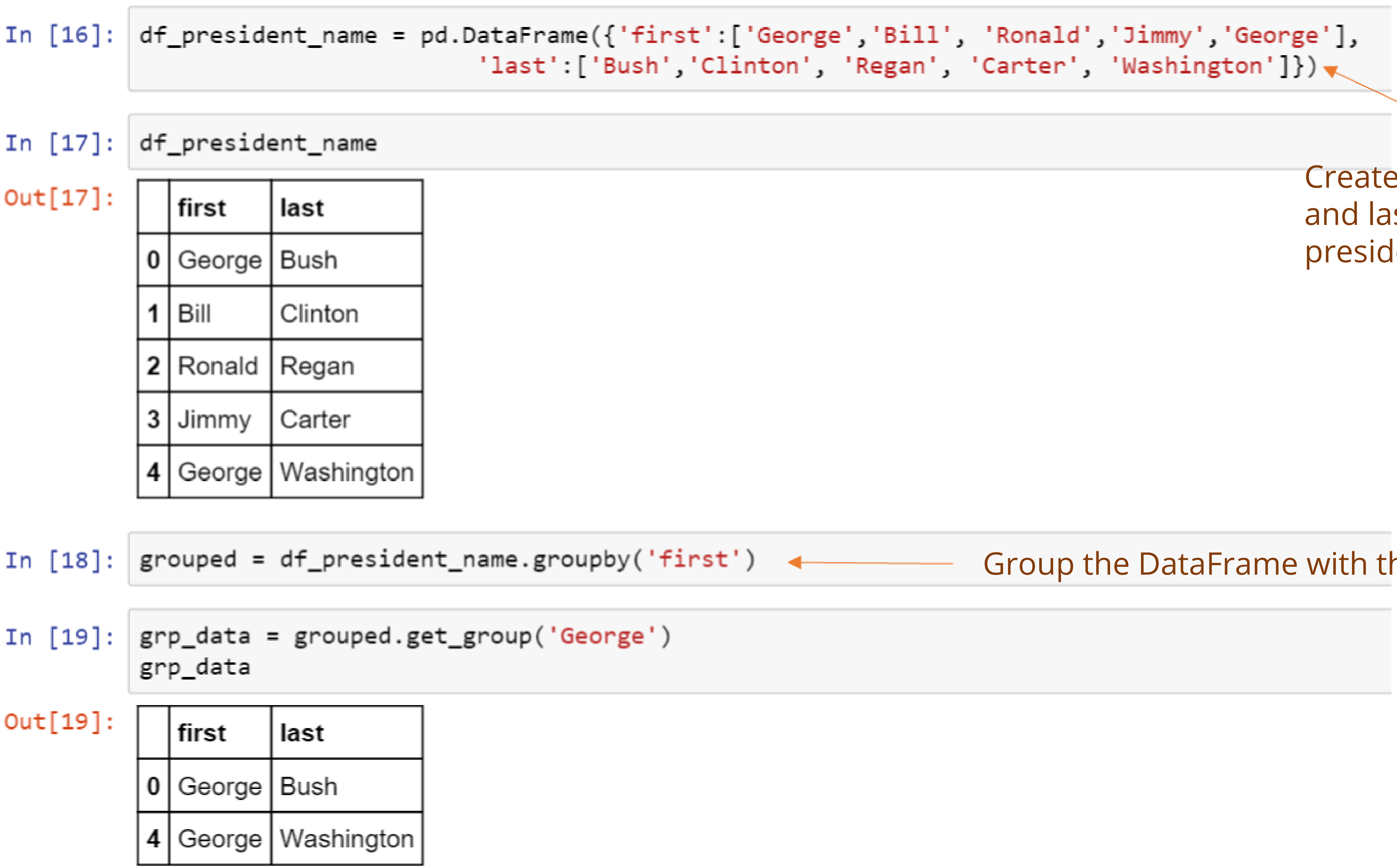
Apply the mean function to find the average score



```
In [10]: df_test_scores.std()

Out[10]: Test1     11.979149
         Test2      4.633213
         dtype: float64
```

Apply the std function to find the standard deviation for both the tests

Data Operation Using Groupby



In [16]:

```
df_president_name = pd.DataFrame({'first':['George','Bill', 'Ronald','Jimmy','George'],
                                  'last':['Bush','Clinton', 'Regan', 'Carter', 'Washington']})
```

Create a DataFrame with first and last name as former presidents

In [17]:

```
df_president_name
```

Out[17]:

	first	last
0	George	Bush
1	Bill	Clinton
2	Ronald	Regan
3	Jimmy	Carter
4	George	Washington

In [18]:

```
grouped = df_president_name.groupby('first')
```

Group the DataFrame with the first name

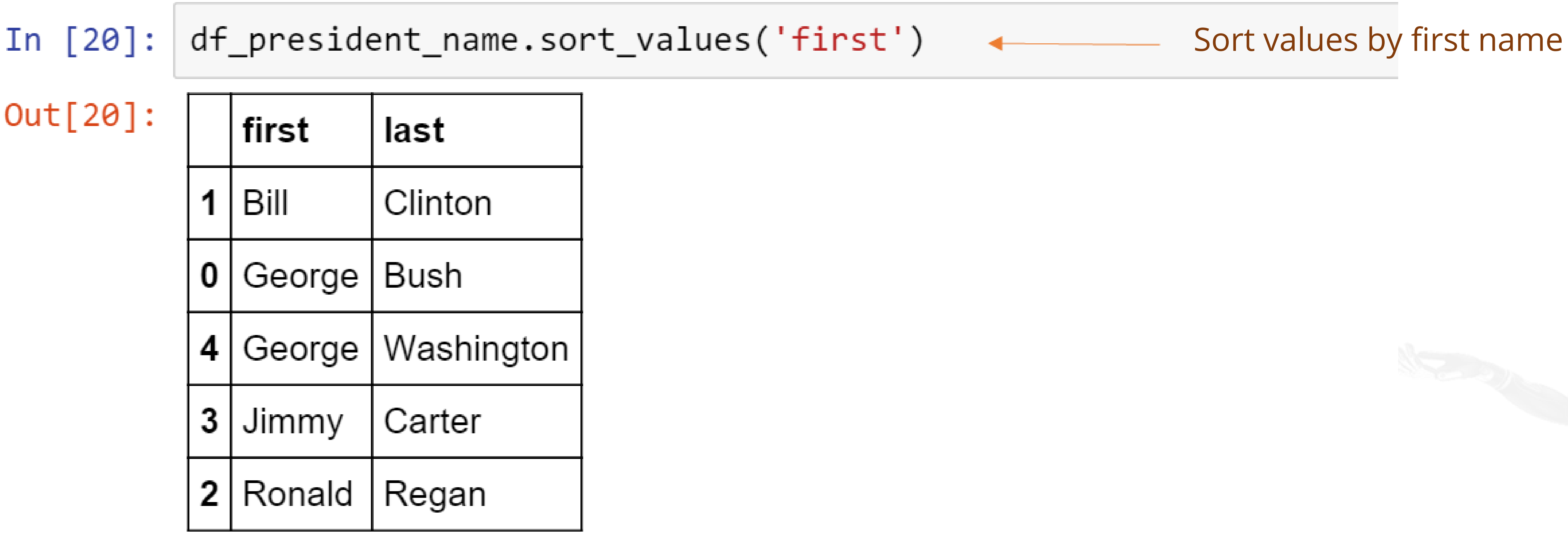
In [19]:

```
grp_data = grouped.get_group('George')
grp_data
```

Out[19]:

	first	last
0	George	Bush
4	George	Washington

Data Operation: Sorting



```
In [20]: df_president_name.sort_values('first')
```

Sort values by first name

Out[20]:

	first	last
1	Bill	Clinton
0	George	Bush
4	George	Washington
3	Jimmy	Carter
2	Ronald	Regan

Data Operations in Pandas DataFrame

Objective: Consider the **SalaryGender** dataset to perform the following operations on pandas dataframe:

1. Replace the values 0 and 1 of the gender column with female and male respectively
2. Find the maximum salary and the minimum salary
3. Find the number of men and women with PhD
4. Store age and Phd columns in the dataframe and remove all the people without PhD
5. Calculate the total number of PhD holders
6. Sort the dataframe on the basis of salary

Access: To execute the practice, follow these steps:

- Go to the **PRACTICE LABS** tab on your LMS
- Click the **START LAB** button
- Click the **LAUNCH LAB** button to start the lab

ASSISTED PRACTICE

Data Operations in Pandas DataFrame

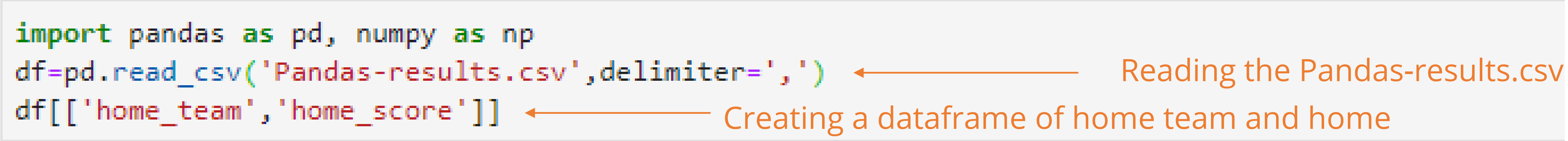
Objective: Consider the **Pandas-results.csv** dataset to perform the following operations on pandas dataframe:

1. Create a dataframe of home_team and home_score
2. Find the home team with maximum home scores
3. Find the teams with home score greater than zero and get the basic statistical details of the dataframe
4. Create a dataframe in the date range 2019-07-10 to 2019-07-20
5. Compare the home score and away score of the teams and add the winning team to a column named winner

Access: To execute the practice, follow these steps:

- Go to the **PRACTICE LABS** tab on your LMS
- Click the **START LAB** button
- Click the **LAUNCH LAB** button to start the lab

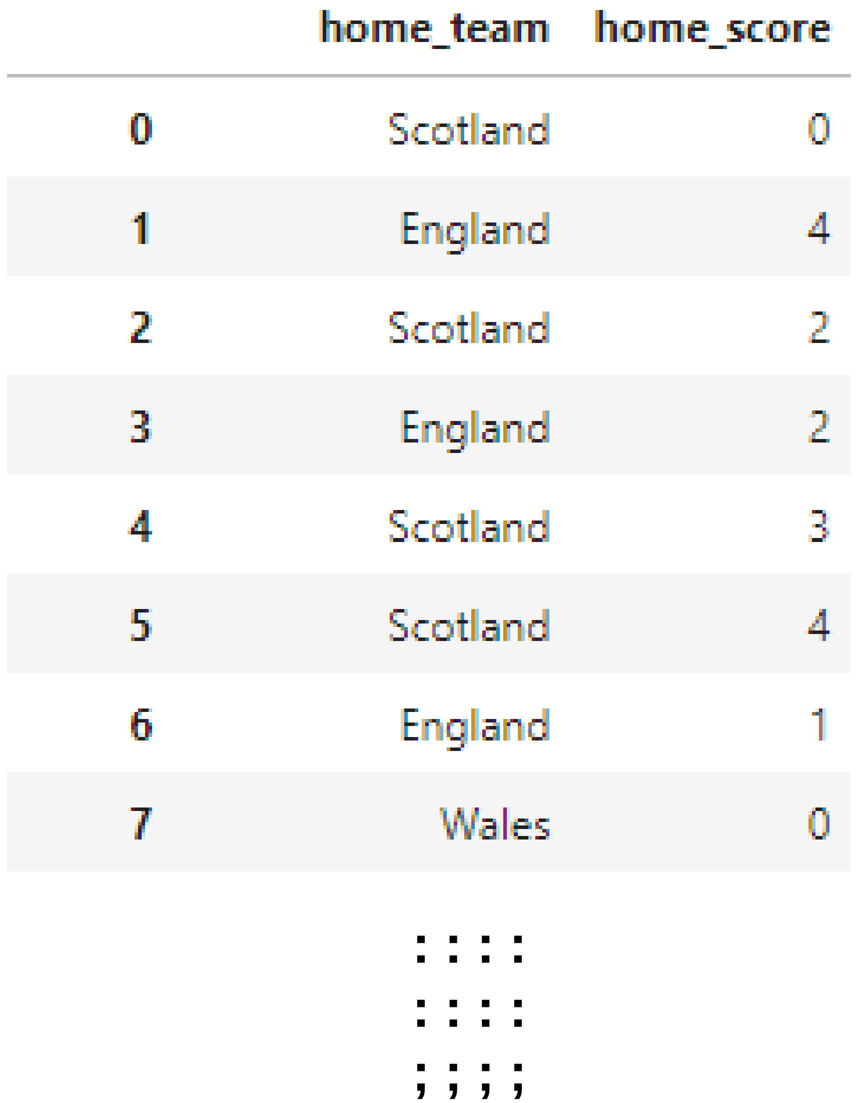
Unassisted Practice: Data Operations in Pandas DataFrame



```
import pandas as pd, numpy as np
df=pd.read_csv('Pandas-results.csv',delimiter=',')
df[['home_team','home_score']]
```

Reading the Pandas-results.csv file

Creating a dataframe of home team and home score

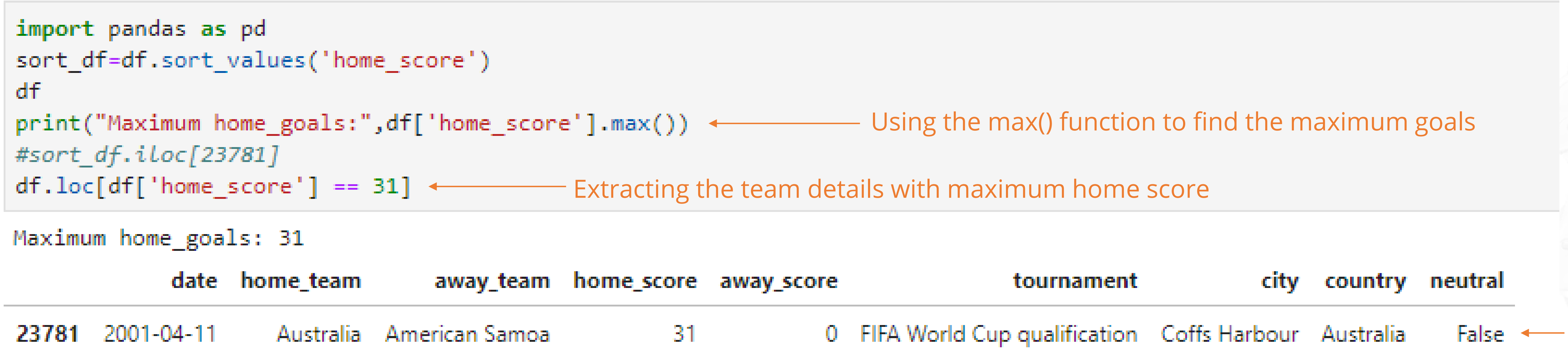


	home_team	home_score
0	Scotland	0
1	England	4
2	Scotland	2
3	England	2
4	Scotland	3
5	Scotland	4
6	England	1
7	Wales	0

: : : :
: : : :
; ; ; ;

Output

Unassisted Practice: Data Operations in Pandas DataFrame



```
import pandas as pd
sort_df=df.sort_values('home_score')
df
print("Maximum home_goals:",df['home_score'].max())
#sort_df.iloc[23781]
df.loc[df['home_score'] == 31]
```

Using the max() function to find the maximum goals

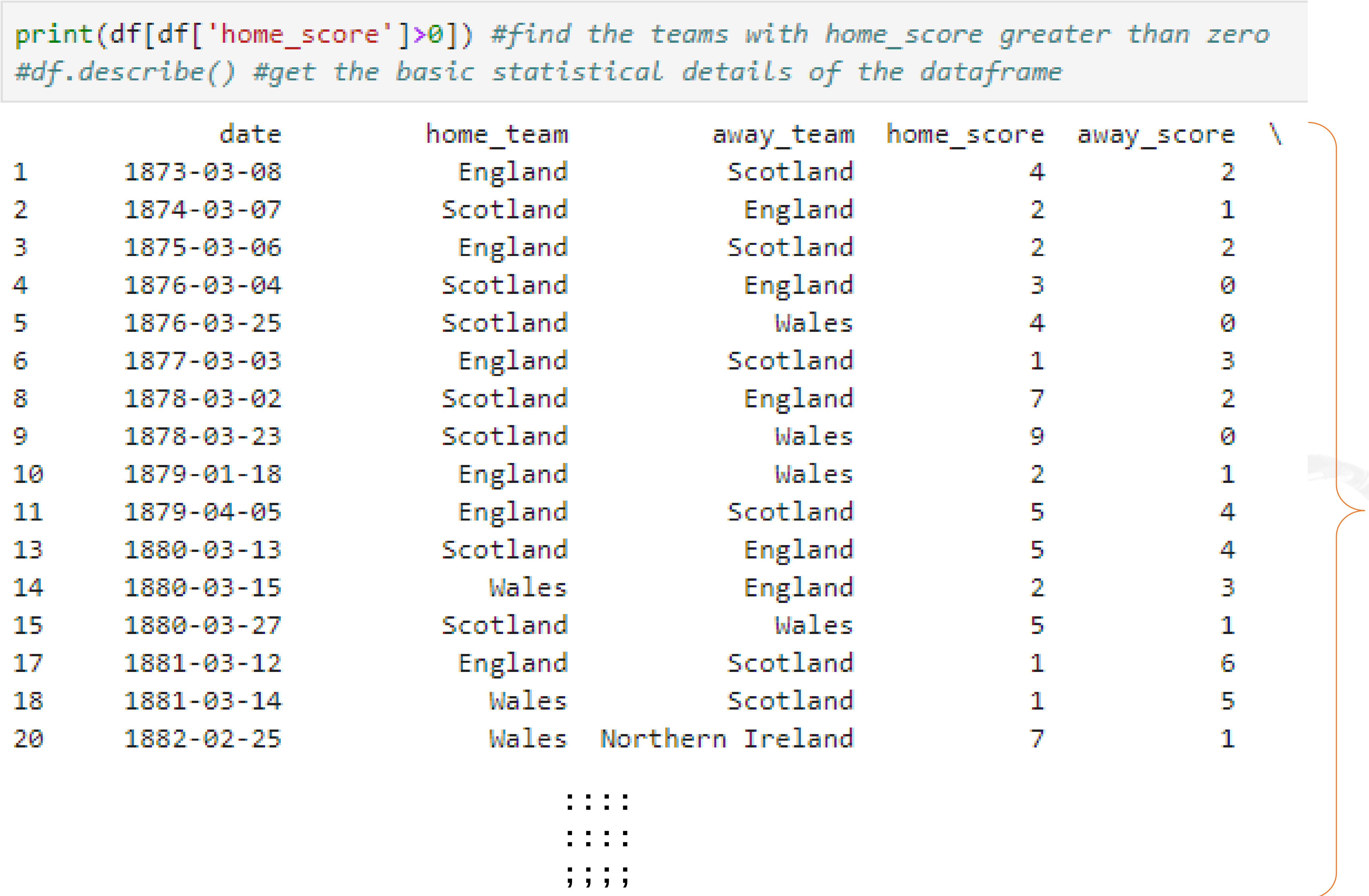
Extracting the team details with maximum home score

```
Maximum home_goals: 31
```

	date	home_team	away_team	home_score	away_score	tournament	city	country	neutral
23781	2001-04-11	Australia	American Samoa	31	0	FIFA World Cup qualification	Coffs Harbour	Australia	False

Output

Unassisted Practice: Data Operations in Pandas DataFrame

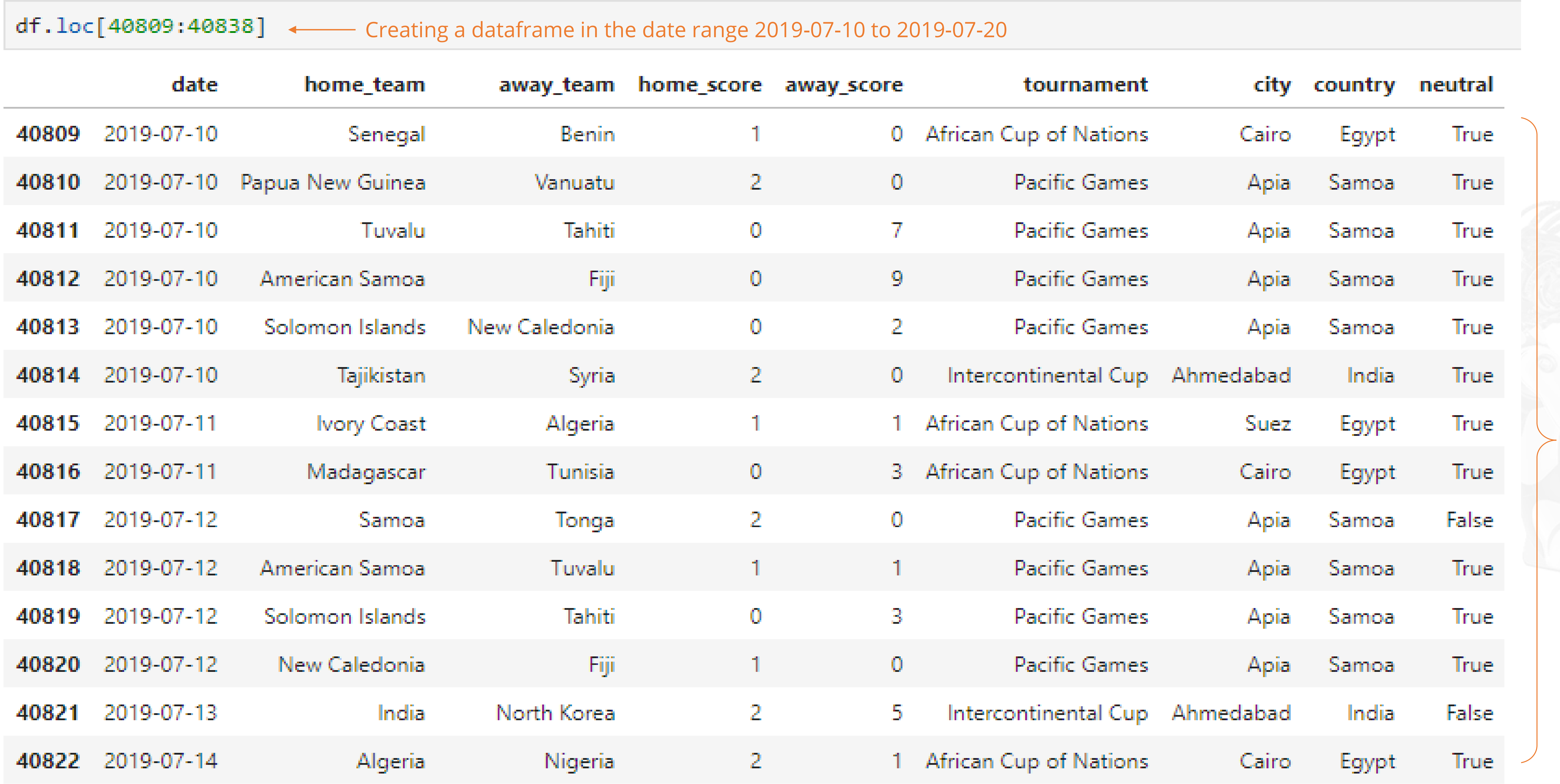


```
print(df[df['home_score']>0]) #find the teams with home_score greater than zero
#df.describe() #get the basic statistical details of the dataframe
```

```
          date    home_team         away_team  home_score  away_score  \
1   1873-03-08      England          Scotland           4           2
2   1874-03-07     Scotland           England           2           1
3   1875-03-06      England          Scotland           2           2
4   1876-03-04     Scotland           England           3           0
5   1876-03-25     Scotland             Wales           4           0
6   1877-03-03      England          Scotland           1           3
8   1878-03-02     Scotland           England           7           2
9   1878-03-23     Scotland             Wales           9           0
10  1879-01-18      England             Wales           2           1
11  1879-04-05      England          Scotland           5           4
13  1880-03-13     Scotland           England           5           4
14  1880-03-15        Wales           England           2           3
15  1880-03-27     Scotland             Wales           5           1
17  1881-03-12      England          Scotland           1           6
18  1881-03-14        Wales          Scotland           1           5
20  1882-02-25        Wales  Northern Ireland           7           1

                              : : : :
                              : : : :
                              ; ; ; ;
```

Output

Unassisted Practice: Data Operations in Pandas DataFrame



```
df.loc[40809:40838]
```

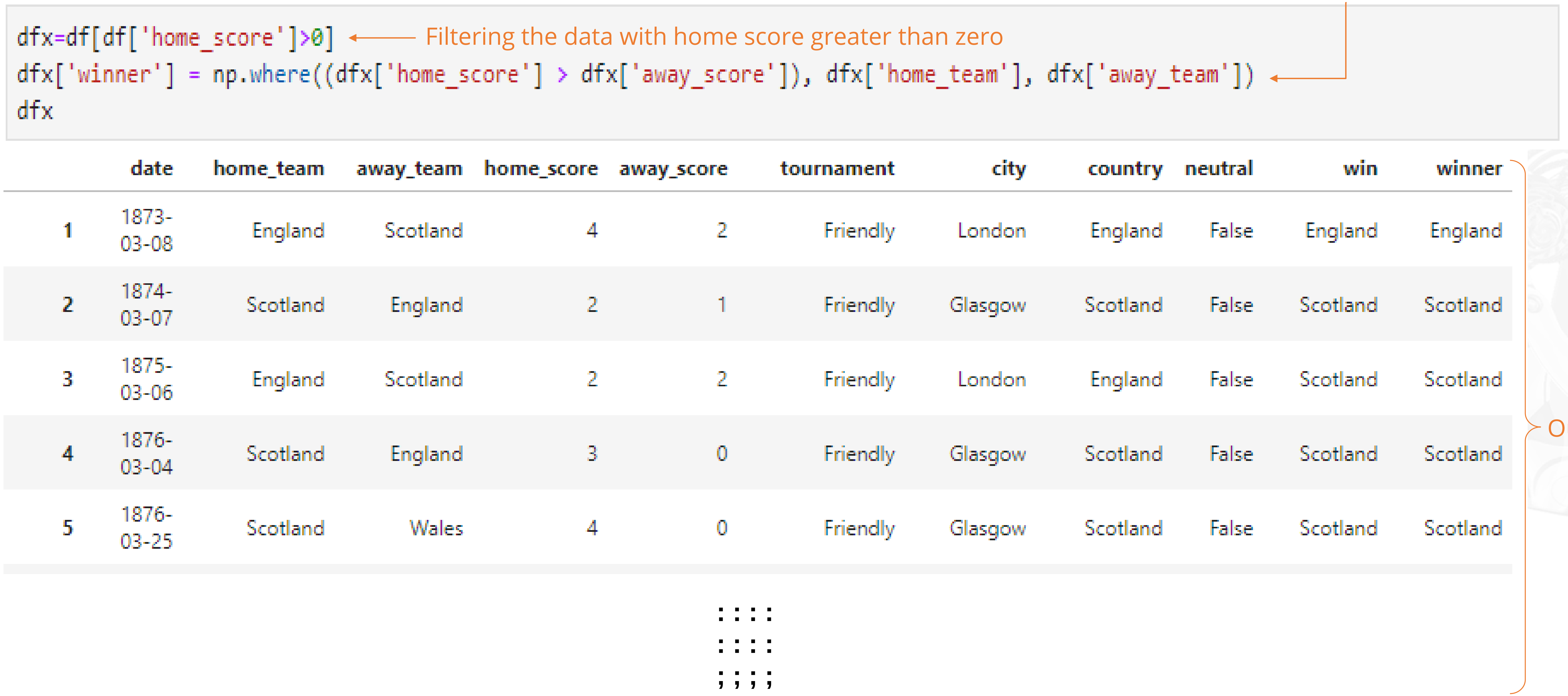
Creating a dataframe in the date range 2019-07-10 to 2019-07-20

	date	home_team	away_team	home_score	away_score	tournament	city	country	neutral
40809	2019-07-10	Senegal	Benin	1	0	African Cup of Nations	Cairo	Egypt	True
40810	2019-07-10	Papua New Guinea	Vanuatu	2	0	Pacific Games	Apia	Samoa	True
40811	2019-07-10	Tuvalu	Tahiti	0	7	Pacific Games	Apia	Samoa	True
40812	2019-07-10	American Samoa	Fiji	0	9	Pacific Games	Apia	Samoa	True
40813	2019-07-10	Solomon Islands	New Caledonia	0	2	Pacific Games	Apia	Samoa	True
40814	2019-07-10	Tajikistan	Syria	2	0	Intercontinental Cup	Ahmedabad	India	True
40815	2019-07-11	Ivory Coast	Algeria	1	1	African Cup of Nations	Suez	Egypt	True
40816	2019-07-11	Madagascar	Tunisia	0	3	African Cup of Nations	Cairo	Egypt	True
40817	2019-07-12	Samoa	Tonga	2	0	Pacific Games	Apia	Samoa	False
40818	2019-07-12	American Samoa	Tuvalu	1	1	Pacific Games	Apia	Samoa	True
40819	2019-07-12	Solomon Islands	Tahiti	0	3	Pacific Games	Apia	Samoa	True
40820	2019-07-12	New Caledonia	Fiji	1	0	Pacific Games	Apia	Samoa	True
40821	2019-07-13	India	North Korea	2	5	Intercontinental Cup	Ahmedabad	India	False
40822	2019-07-14	Algeria	Nigeria	2	1	African Cup of Nations	Cairo	Egypt	True

Output

Unassisted Practice: Data Operations in Pandas DataFrame

Adding a column named winner to the filtered dataframe



```
dfx=df[df['home_score']>0]
dfx['winner'] = np.where((dfx['home_score'] > dfx['away_score']), dfx['home_team'], dfx['away_team'])
dfx
```

Filtering the data with home score greater than zero

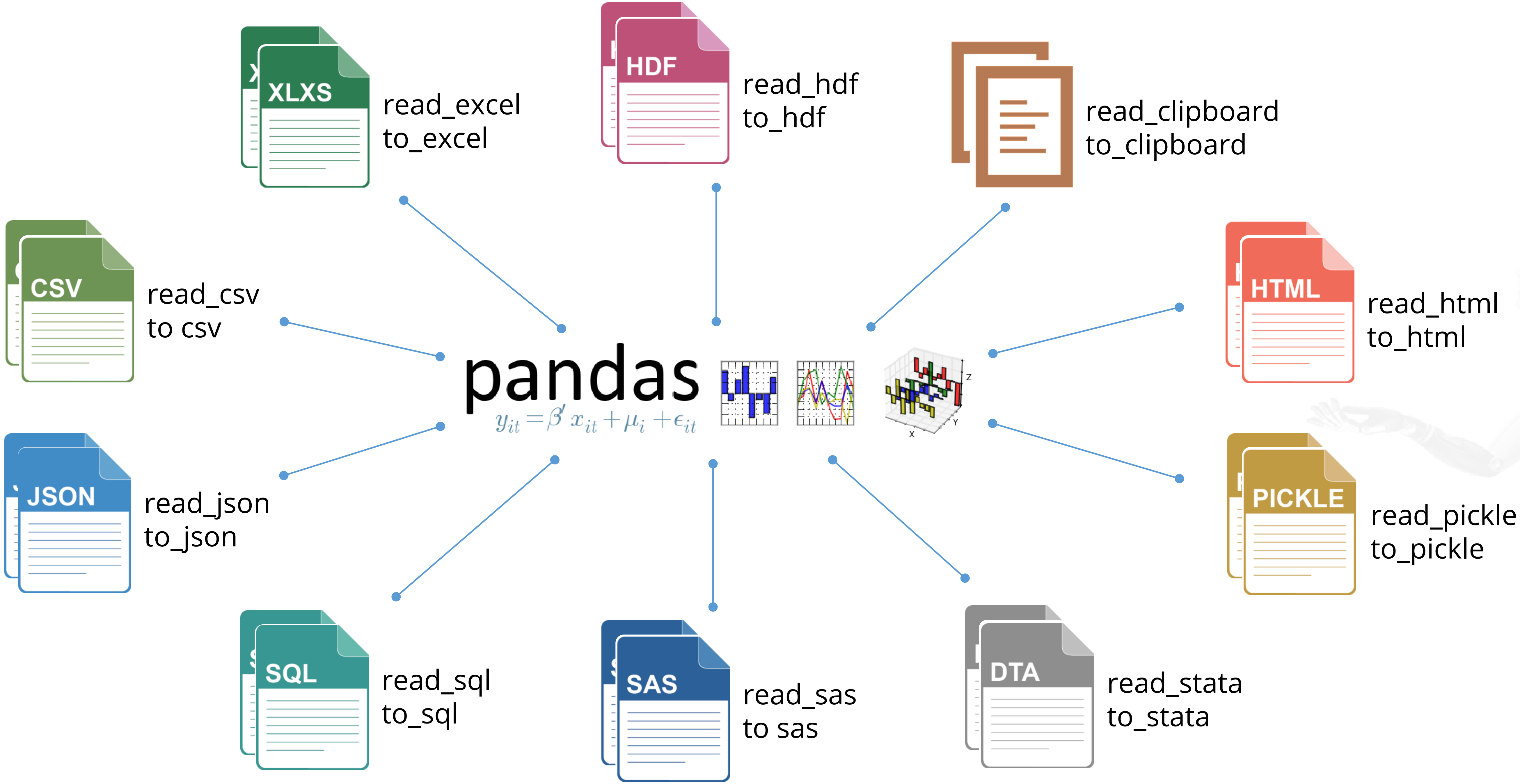
	date	home_team	away_team	home_score	away_score	tournament	city	country	neutral	win	winner
1	1873-03-08	England	Scotland	4	2	Friendly	London	England	False	England	England
2	1874-03-07	Scotland	England	2	1	Friendly	Glasgow	Scotland	False	Scotland	Scotland
3	1875-03-06	England	Scotland	2	2	Friendly	London	England	False	Scotland	Scotland
4	1876-03-04	Scotland	England	3	0	Friendly	Glasgow	Scotland	False	Scotland	Scotland
5	1876-03-25	Scotland	Wales	4	0	Friendly	Glasgow	Scotland	False	Scotland	Scotland

```
: : : :
: : : :
; ; ; ;
```

Output

Data Standardization

File Read and Write Support



read_excel
to_excel

read_hdf
to_hdf

read_clipboard
to_clipboard

read_csv
to csv

read_html
to_html

read_json
to_json

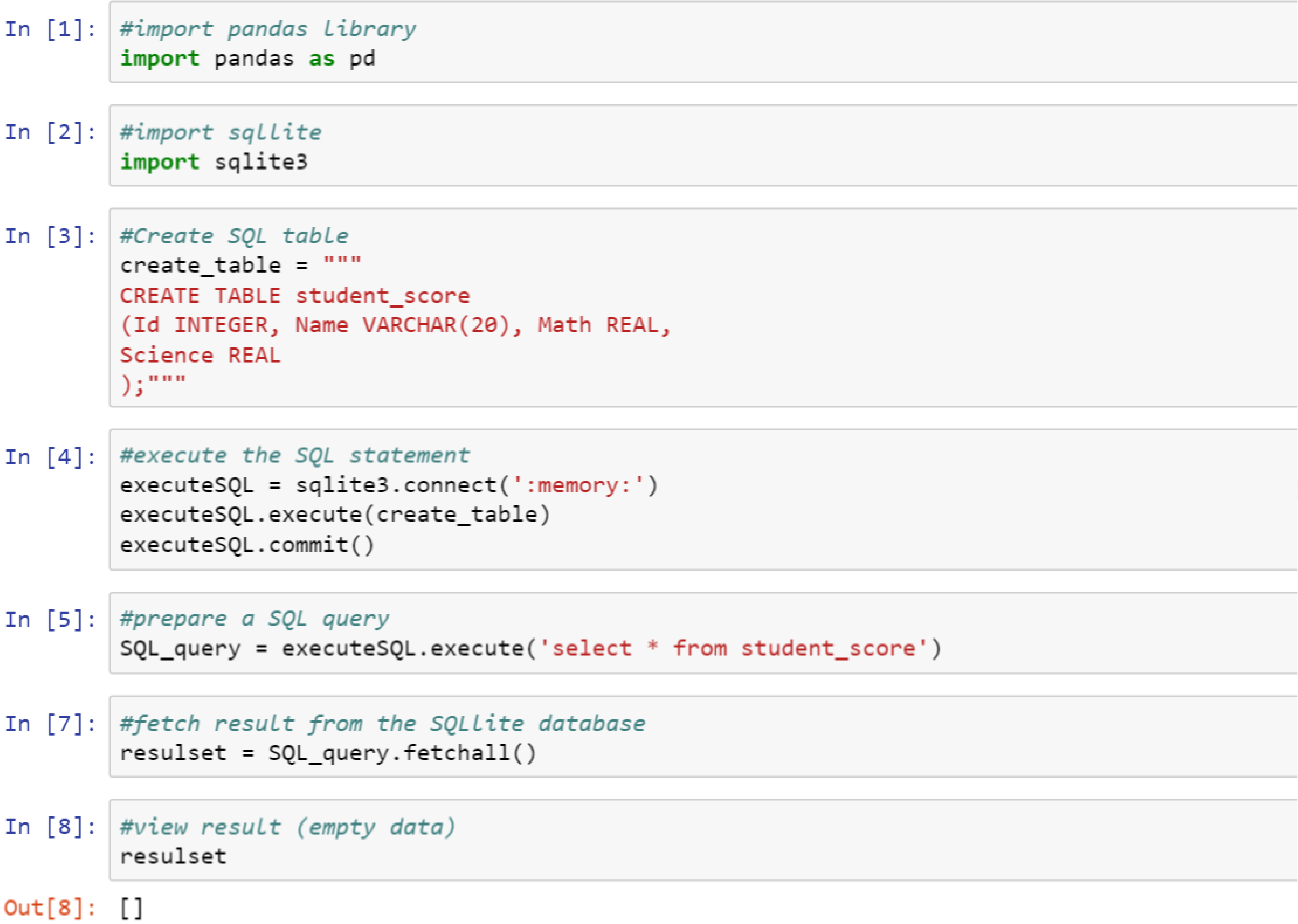
read_pickle
to_pickle

read_sql
to_sql

read_sas
to sas

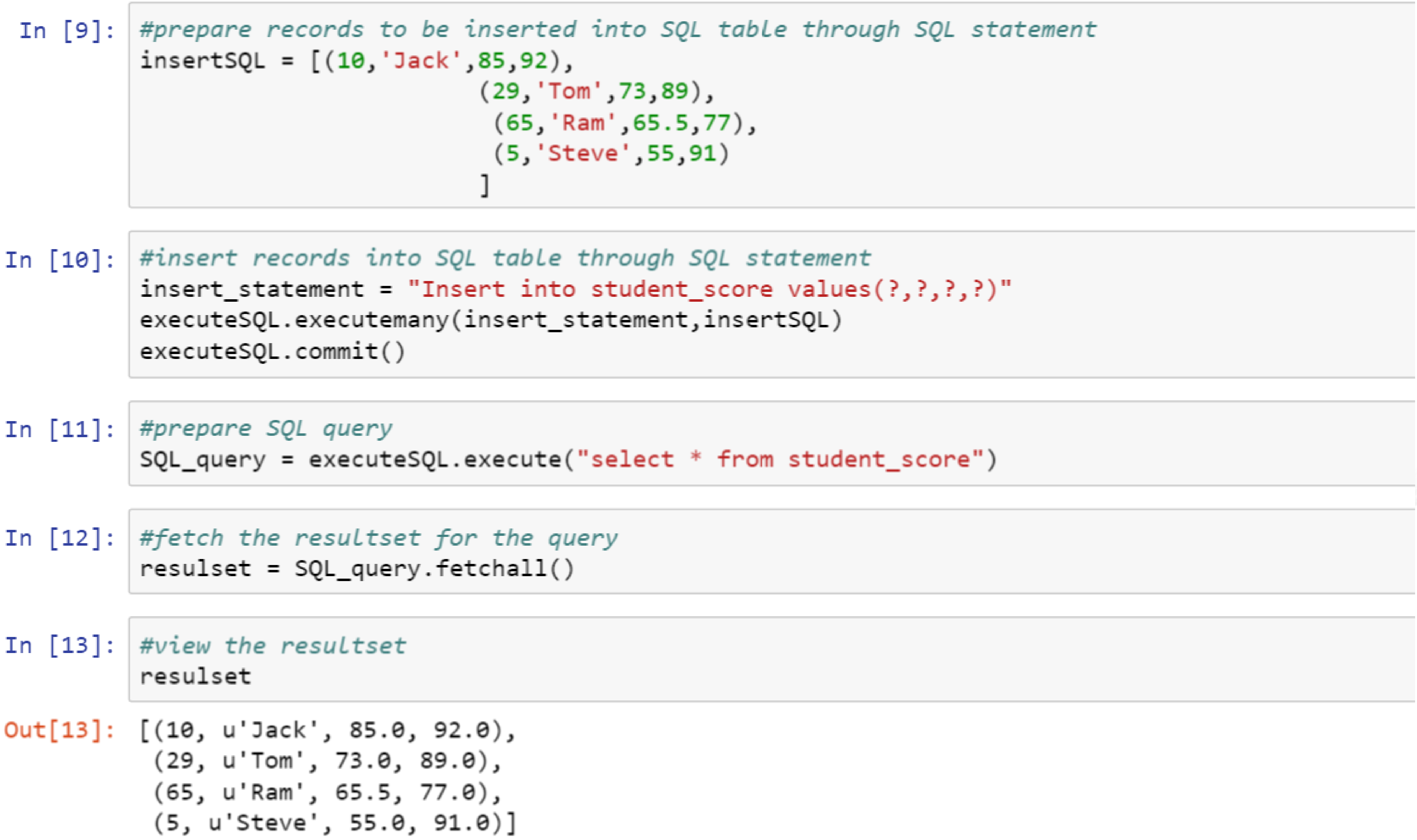
read_stata
to_stata

Pandas SQL Operation



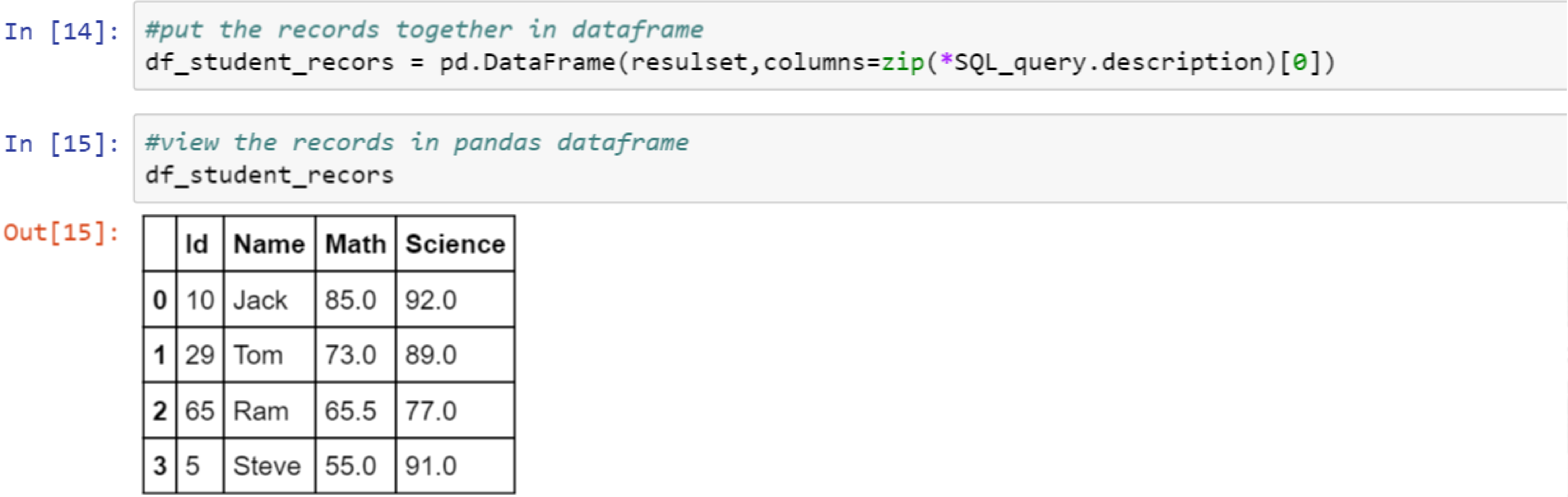
```
In [1]: #import pandas library
        import pandas as pd

In [2]: #import sqllite
        import sqlite3

In [3]: #Create SQL table
        create_table = """
        CREATE TABLE student_score
        (Id INTEGER, Name VARCHAR(20), Math REAL,
        Science REAL
        );"""

In [4]: #execute the SQL statement
        executeSQL = sqlite3.connect(':memory:')
        executeSQL.execute(create_table)
        executeSQL.commit()

In [5]: #prepare a SQL query
        SQL_query = executeSQL.execute('select * from student_score')

In [7]: #fetch result from the SQLlite database
        resulset = SQL_query.fetchall()

In [8]: #view result (empty data)
        resulset

Out[8]: []
```

Pandas SQL Operation



```
In [9]:  #prepare records to be inserted into SQL table through SQL statement
          insertSQL = [(10,'Jack',85,92),
                        (29,'Tom',73,89),
                         (65,'Ram',65.5,77),
                         (5,'Steve',55,91)
                        ]

In [10]:  #insert records into SQL table through SQL statement
          insert_statement = "Insert into student_score values(?,?,?,?)"
          executeSQL.executemany(insert_statement,insertSQL)
          executeSQL.commit()

In [11]:  #prepare SQL query
          SQL_query = executeSQL.execute("select * from student_score")

In [12]:  #fetch the resultset for the query
          resulset = SQL_query.fetchall()

In [13]:  #view the resultset
          resulset

Out[13]:  [(10, u'Jack', 85.0, 92.0),
           (29, u'Tom', 73.0, 89.0),
           (65, u'Ram', 65.5, 77.0),
           (5, u'Steve', 55.0, 91.0)]
```

Pandas SQL Operation



```
In [14]: #put the records together in dataframe
         df_student_recors = pd.DataFrame(resulset,columns=zip(*SQL_query.description)[0])
```

```
In [15]: #view the records in pandas dataframe
         df_student_recors
```

Out[15]:

	Id	Name	Math	Science
0	10	Jack	85.0	92.0
1	29	Tom	73.0	89.0
2	65	Ram	65.5	77.0
3	5	Steve	55.0	91.0

Pandas SQL Operations

Objective: Perform the following SQL operations with pandas:

1. Read the **Pandas-results.csv** dataset and create a dataframe
2. Write the SQL query to retrieve the first 10 items of the dataframe
3. Create a SQL table named movie rating with attributes like movie name, genre, and rating
4. Insert records into SQL table through SQL statement
5. Fetch and view the result
6. Create a dataframe of the result

Access: To execute the practice, follow these steps:

- Go to the **PRACTICE LABS** tab on your LMS
- Click the **START LAB** button
- Click the **LAUNCH LAB** button to start the lab

ASSISTED PRACTICE

Pandas SQL Operations

Objective: Perform the following SQL operations:

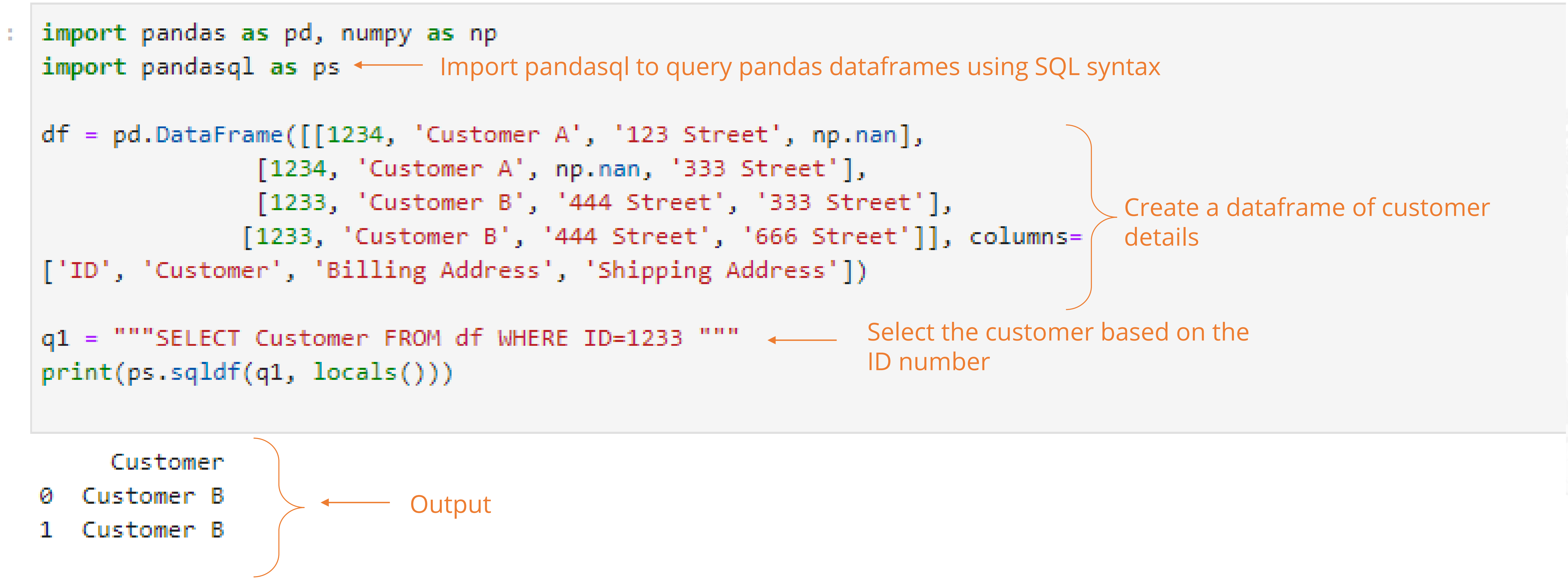
1. Create a dataframe of customer details with columns like ID, customer, billing address, and shipping address
2. Write a SQL query to retrieve customers based on a particular ID

   Example: For ID=1233, output is customer A
3. Create a small dataframe and write the records in dataframe to a SQL database

Access: To execute the practice, follow these steps:

- Go to the **PRACTICE LABS** tab on your LMS
- Click the **START LAB** button
- Click the **LAUNCH LAB** button to start the lab

Unassisted Practice: Pandas SQL Operations



```
import pandas as pd, numpy as np
import pandasql as ps

df = pd.DataFrame([[1234, 'Customer A', '123 Street', np.nan],
               [1234, 'Customer A', np.nan, '333 Street'],
               [1233, 'Customer B', '444 Street', '333 Street'],
              [1233, 'Customer B', '444 Street', '666 Street']], columns=
['ID', 'Customer', 'Billing Address', 'Shipping Address'])

q1 = """SELECT Customer FROM df WHERE ID=1233 """
print(ps.sqldf(q1, locals()))
```

```
     Customer
0  Customer B
1  Customer B
```

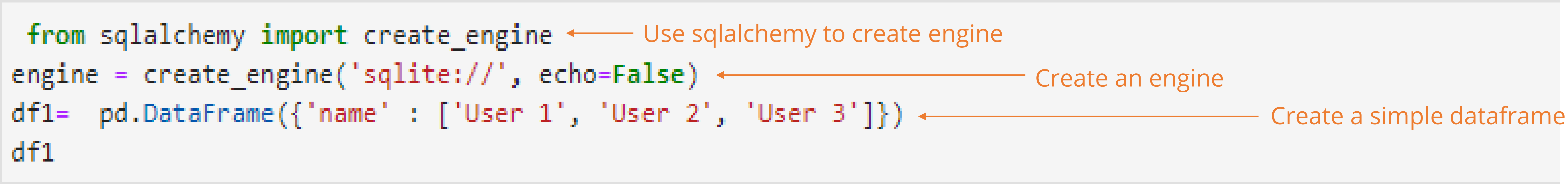
Import pandasql to query pandas dataframes using SQL syntax

Create a dataframe of customer details

Select the customer based on the ID number

Output

Unassisted Practice: Pandas SQL Operations

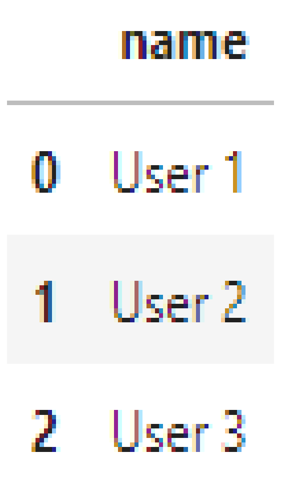


```
 from sqlalchemy import create_engine
engine = create_engine('sqlite://', echo=False)
df1=  pd.DataFrame({'name' : ['User 1', 'User 2', 'User 3']})
df1
```

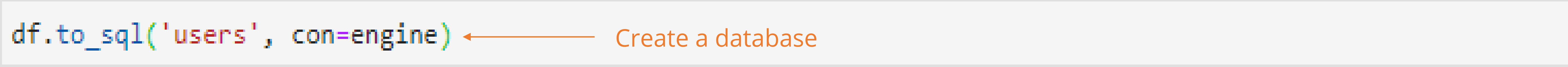
Use sqlalchemy to create engine

Create an engine

Create a simple dataframe

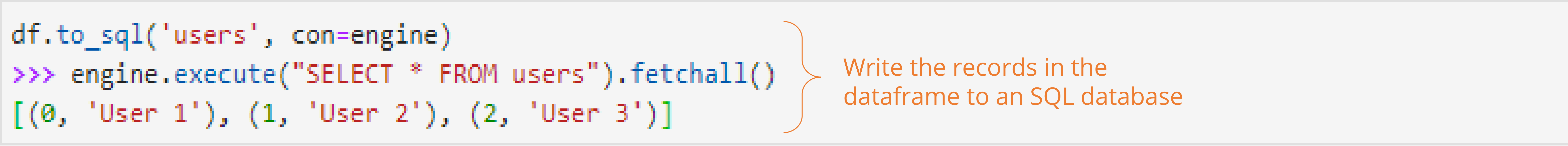


	name
0	User 1
1	User 2
2	User 3



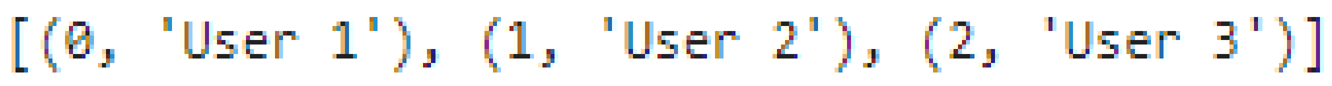
```
df.to_sql('users', con=engine)
```

Create a database



```
df.to_sql('users', con=engine)
>>> engine.execute("SELECT * FROM users").fetchall()
[(0, 'User 1'), (1, 'User 2'), (2, 'User 3')]
```

Write the records in the dataframe to an SQL database



```
[(0, 'User 1'), (1, 'User 2'), (2, 'User 3')]
```

Output

Key Takeaways

You are now able to:

- Explain Pandas and its features
- List different data structures of Pandas
- Outline the process to create series and DataFrame with data inputs
- Explain how to view, select, and access elements in a data structure
- Describe the procedure to handle vectorized operations
- Illustrate how to handle missing values
- Analyze data with different data operation methods

Knowledge Check

Knowledge Check

1

How is an index for data elements assigned while creating a Pandas series ? Select all that apply?

a. Created automatically

b. Needs to be assigned

c. Once created can not be changed or altered

d. Index is not applicable as series is one-dimensional

Knowledge Check

1

How is an index for data elements assigned while creating a Pandas series ? Select all that apply?

a. Created automatically

b. Needs to be assigned

c. Once created can not be changed or altered

d. Index is not applicable as series is one-dimensional

The correct answer is **a, b**

Data alignment is intrinsic in Pandas data structure and happens automatically. One can also assign index to data elements.

Knowledge Check

2

What will the result be in vector addition if label is not found in a series?

a. Marked as zeros for missing labels

b. Labels will be skipped

c. Marked as NaN for missing labels

d. Will prompt an exception, index not found

Knowledge Check

2

What will the result be in vector addition if label is not found in a series?

a. Marked as zeros for missing labels

b. Labels will be skipped

c. Marked as NaN for missing labels

d. Will prompt an exception, index not found

The correct answer is **c**

The result will be marked as NaN (Not a Number) for missing labels.

Knowledge Check

3

What is the result of DataFrame[3:9]?

a. Series with sliced index from 3 to 9

b. dict of index positions 3 and 9

c. DataFrame of sliced rows index from 3 to 9

d. DataFrame with data elements at index 3 to 9

Knowledge Check

3

What is the result of DataFrame[3:9]?

a. Series with sliced index from 3 to 9

b. dict of index positions 3 and 9

c. DataFrame of sliced rows index from 3 to 9

d. DataFrame with data elements at index 3 to 9

The correct answer is **c**

This is DataFrame slicing technique with indexing or selection on data elements. When a user passes the range 3:9, the entire range from 3 to 9 gets sliced and displayed as output.

Knowledge Check

4

What does the fillna() method do?

a. Fills all NaN values with zeros

b. Fills all NaN values with one

c. Fills all NaN values with values mentioned in the parenthesis

d. Drops NaN values from the dataset

Knowledge Check

4

What does the fillna() method do?

a. Fills all NaN values with zeros

b. Fills all NaN values with one

c. Fills all NaN values with values mentioned in the parenthesis

d. Drops NaN values from the dataset

The correct answer is **c**

fillna is one of the basic methods to fill NaN values in a dataset with a desired value by passing that in parenthesis.

Knowledge Check

5

Which of the following data structures is used to store three-dimensional data?

a. Series

b. DataFrame

c. Panel

d. PanelND

Knowledge Check

5

Which of the following data structures is used to store three-dimensional data?

a. Series

b. DataFrame

c. Panel

d. PanelND

The correct answer is **c**

Panel is a data structure used to store three-dimensional data.

Analyze the Federal Aviation Authority (FAA) Dataset using Pandas

Problem Statement:

Analyze the Federal Aviation Authority (FAA) dataset using Pandas to do the following:

1. View
   a. Aircraft manufacturer name
   b. State name
   c. Aircraft model name
   d. Text information
   e. Flight phase
   f. Event description type
   g. Fatal flag
2. Clean the dataset and replace the fatal flag NaN with "No"
3. Find the aircraft types and their occurrences in the dataset
4. Remove all the observations where aircraft names are not available
5. Display the observations where fatal flag is "Yes"

LESSON-END PROJECT

Analyze the Federal Aviation Authority (FAA) Dataset using Pandas

Instructions to perform:

Download the FAA dataset from the “Resource” tab. Upload the dataset to your Jupyter notebook to view and evaluate it.

LESSON-END PROJECT

Analyzing the Dataset

Problem Statement:

A dataset in CSV format is given for the Fire Department of the New York City.

Analyze the dataset to determine:

1. The total number of fire department facilities in the New York city
2. The number of fire department facilities in each borough
3. The facility names in Manhattan

Instructions to perform:

Download the FDNY dataset from the “Resource” tab. You can upload the dataset to your Jupyter notebook to use it.

LESSON-END PROJECT

Thank You