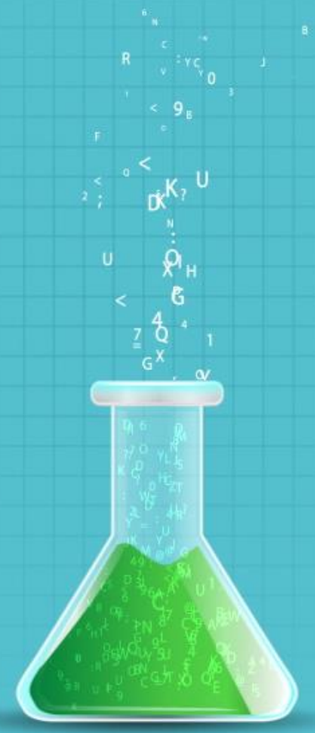
Data Science with Python

Lesson 7— Data Manipulation with Pandas

DATA SCIENCE

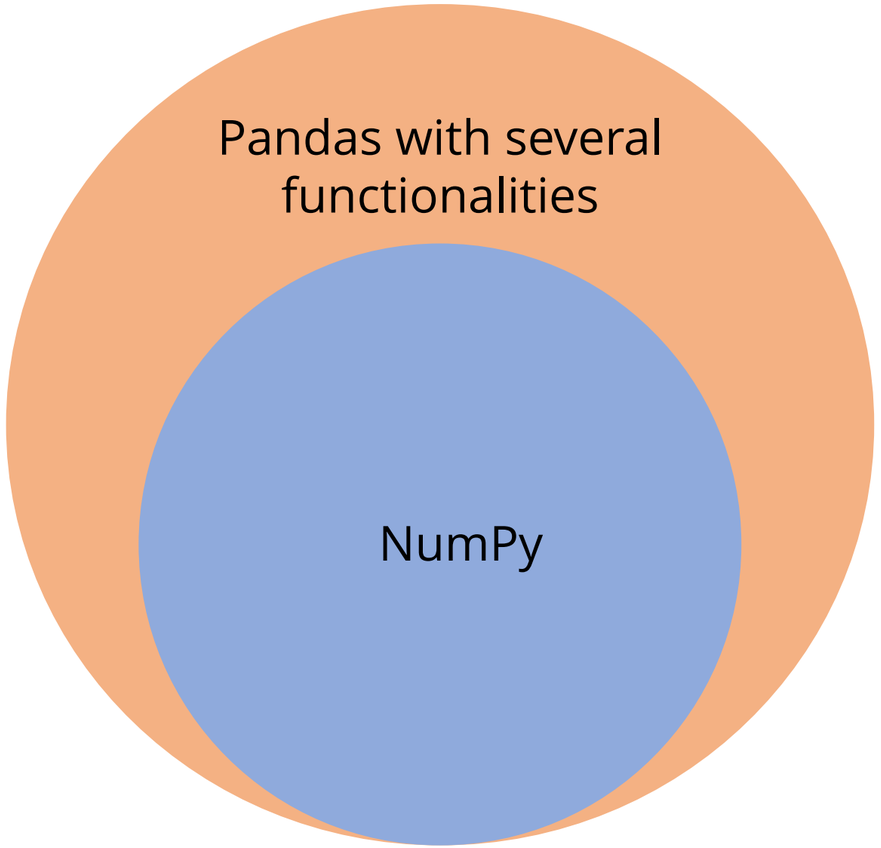
What You Will Learn

- Pandas and its features
- Different data structures of Pandas
- Creating Series and DataFrame with data inputs
- Viewing, selecting, and accessing elements in a data structure
- Handling vectorized operations
- Learning how to handle missing values
- Analyzing data with different data operation methods



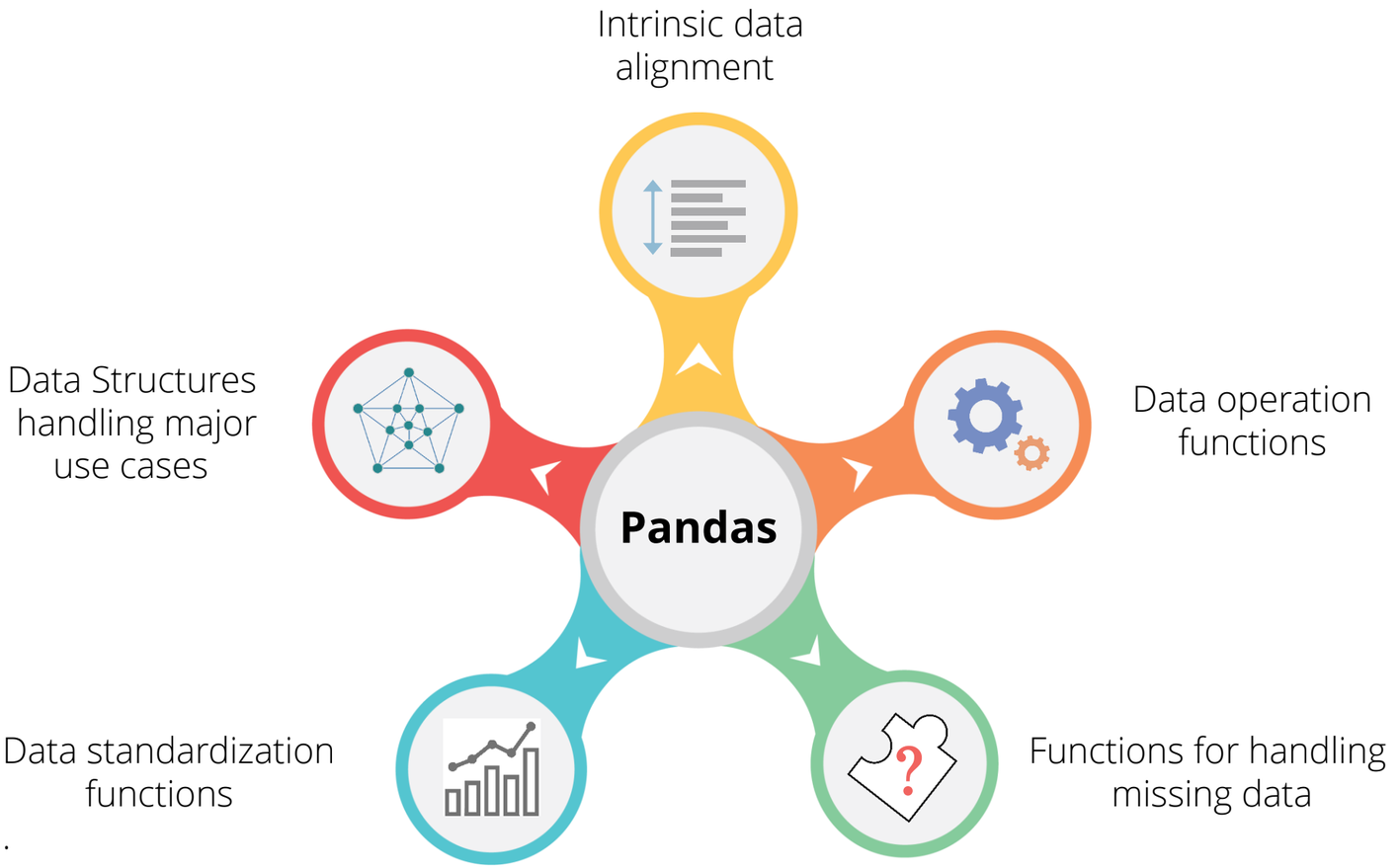
Why Pandas

NumPy is great for mathematical computing. Then why do we need Pandas?



Why Pandas

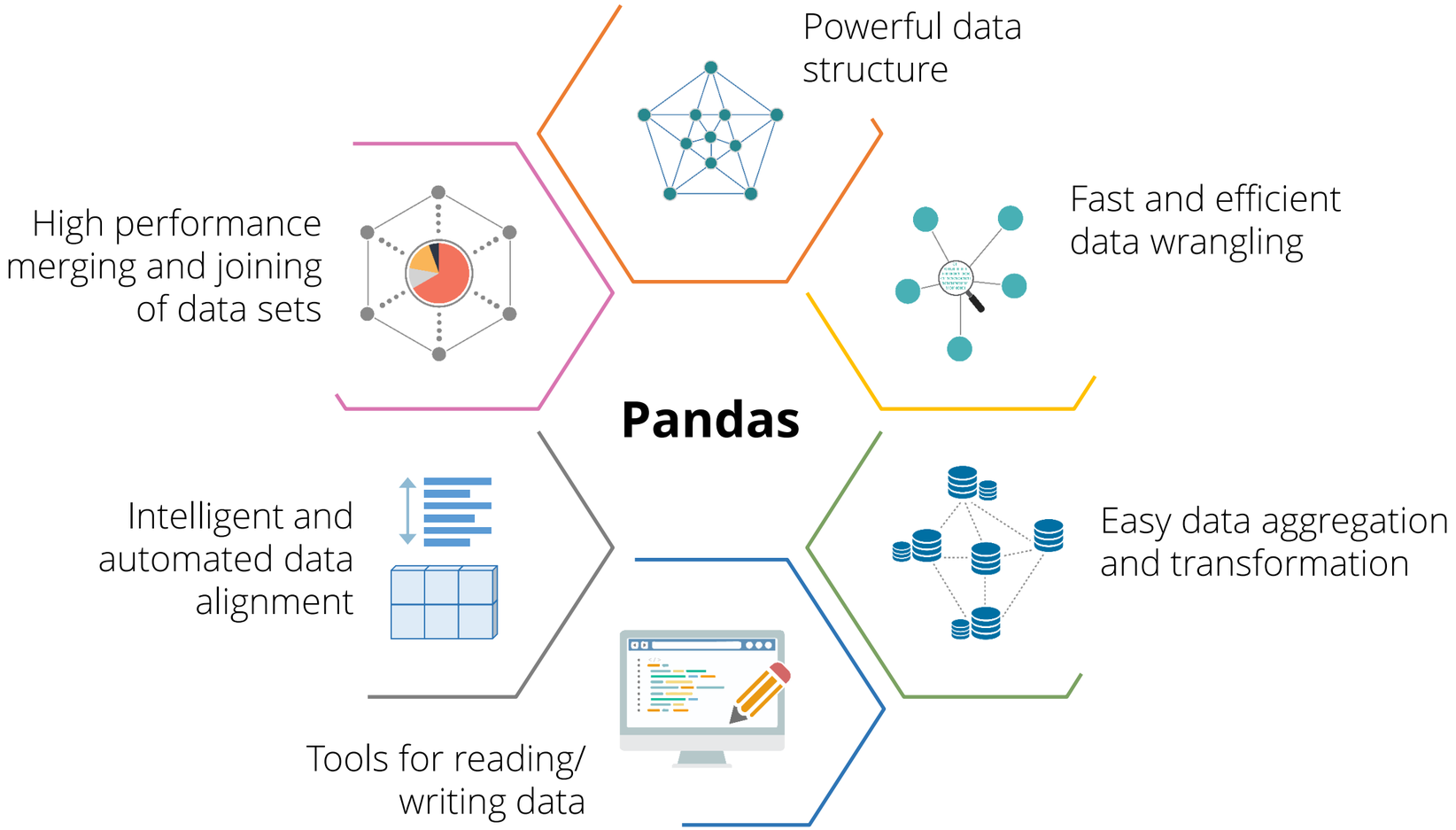
NumPy is great for mathematical computing. Then why do we need Pandas?



.

Pandas Features

The various features of Pandas makes it an efficient library for Data Scientists.



Data Structures

The four main libraries of Pandas data structure are:

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

Series

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

Data Frame

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

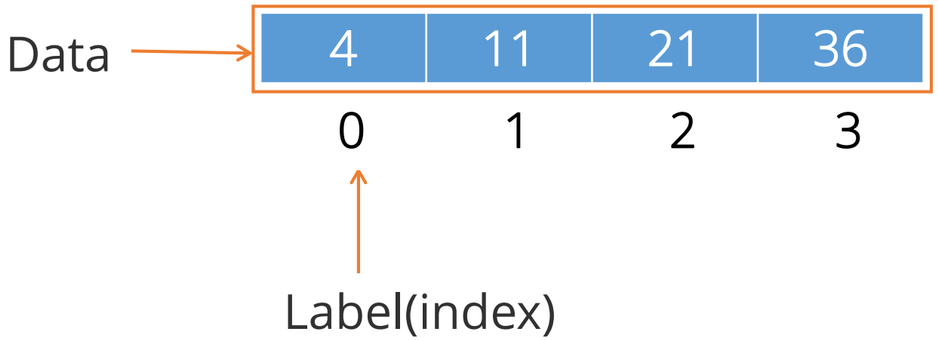
Panel

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

Panel 4D (Experimental)

Understanding Series

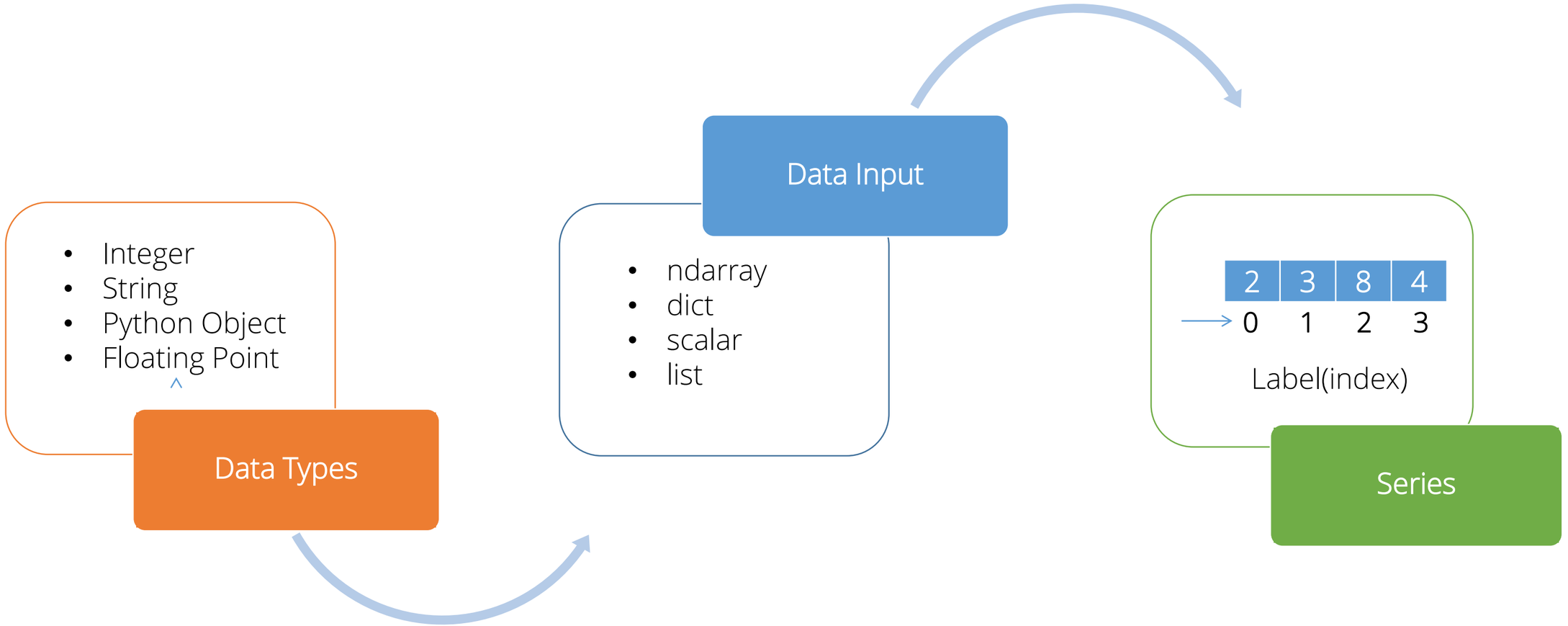
Series is a one-dimensional array-like object containing data and labels (or index).



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

Series

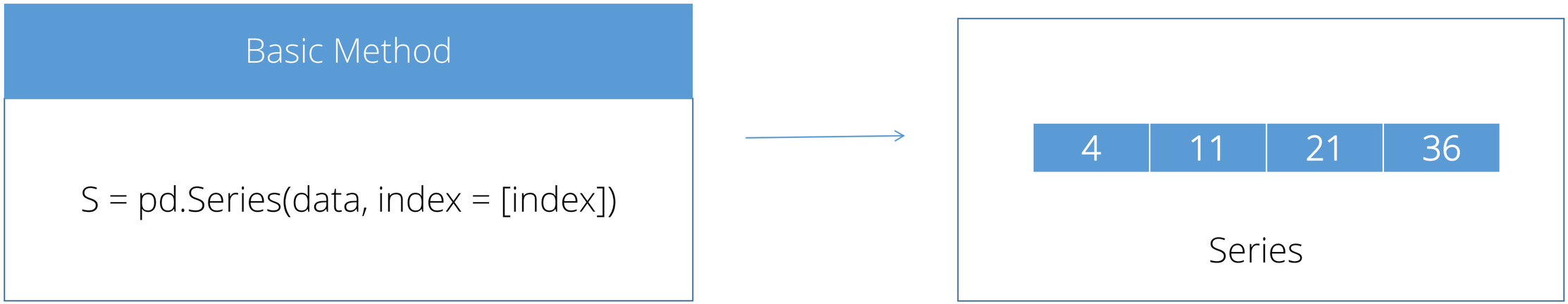
Series can be created with different data inputs:



How to Create Series

Key points to note while creating a series are as follows:

- Import Pandas as it is the main library
- Apply the syntax and pass the data elements as arguments
- Import NumPy while working with ndarrays



Create Series from List

This example shows you how to create a series from a list:

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

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

In [16]: print (first_series)

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

Import libraries

Pass list as an argument

Data value

Index

Data type

We have not created index for data but notice that data alignment is done automatically

Create Series from ndarray

This example shows you how to create a 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
```

countries

Data type

Create 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
```

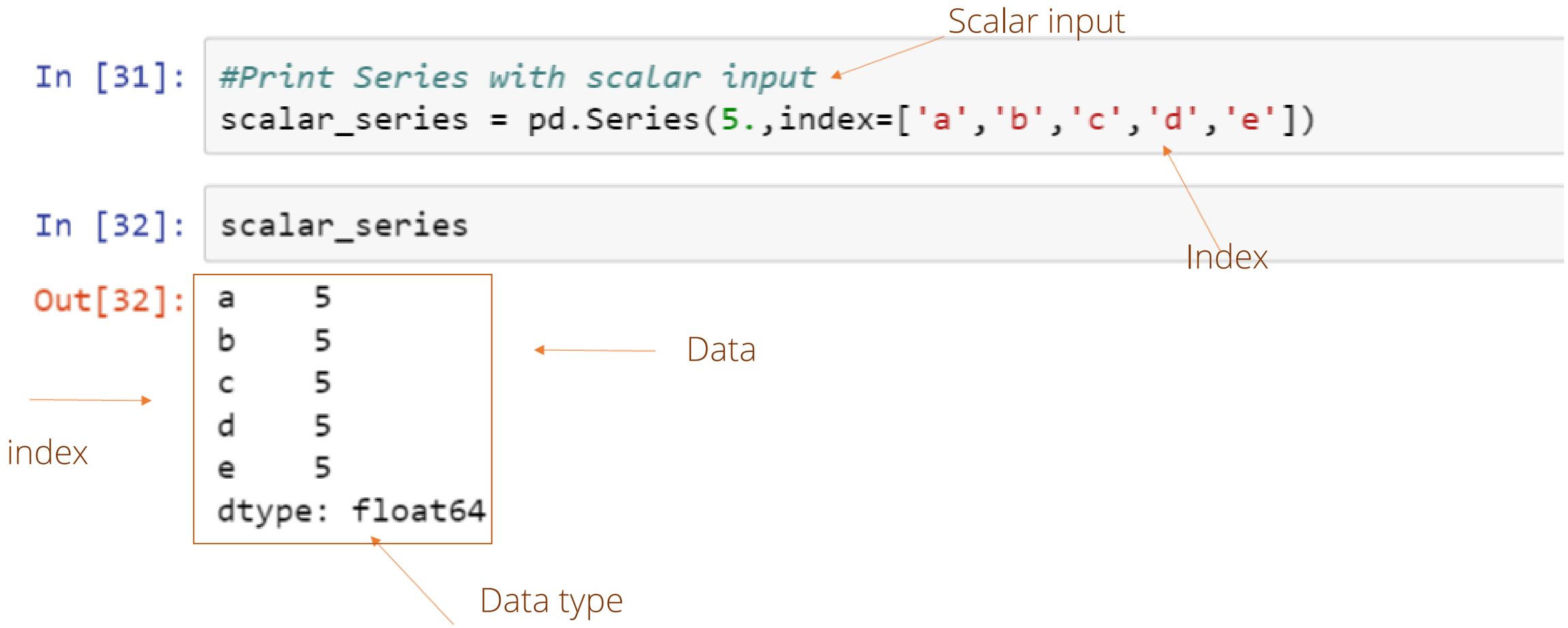
Countries have been passed as an index and GDP as the actual data value

GDP

Country

Data type

Create Series from Scalar



Accessing Elements in Series

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

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

Out[43]: 52056.017809999998

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

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

Out[45]: 37691.027329999997

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

Out[46]: 52056.017809999998
```

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

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
```

Knowledge Check

KNOWLEDGE CHECK

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

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** .

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

KNOWLEDGE CHECK

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 throw an exception, index not found

KNOWLEDGE CHECK

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

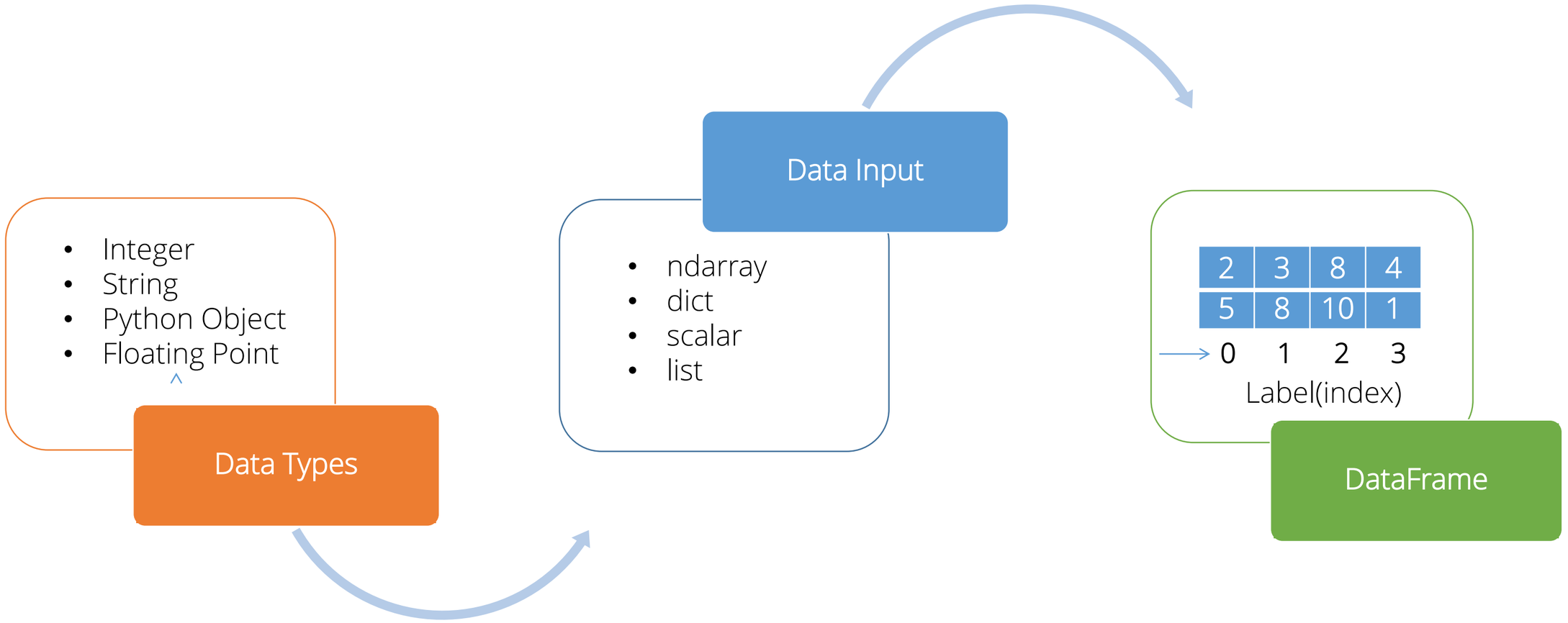
d. Will throw an exception, index not found

The correct answer is **c** .

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

DataFrame

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



Create DataFrame from Lists

Let’s see how you can create a 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

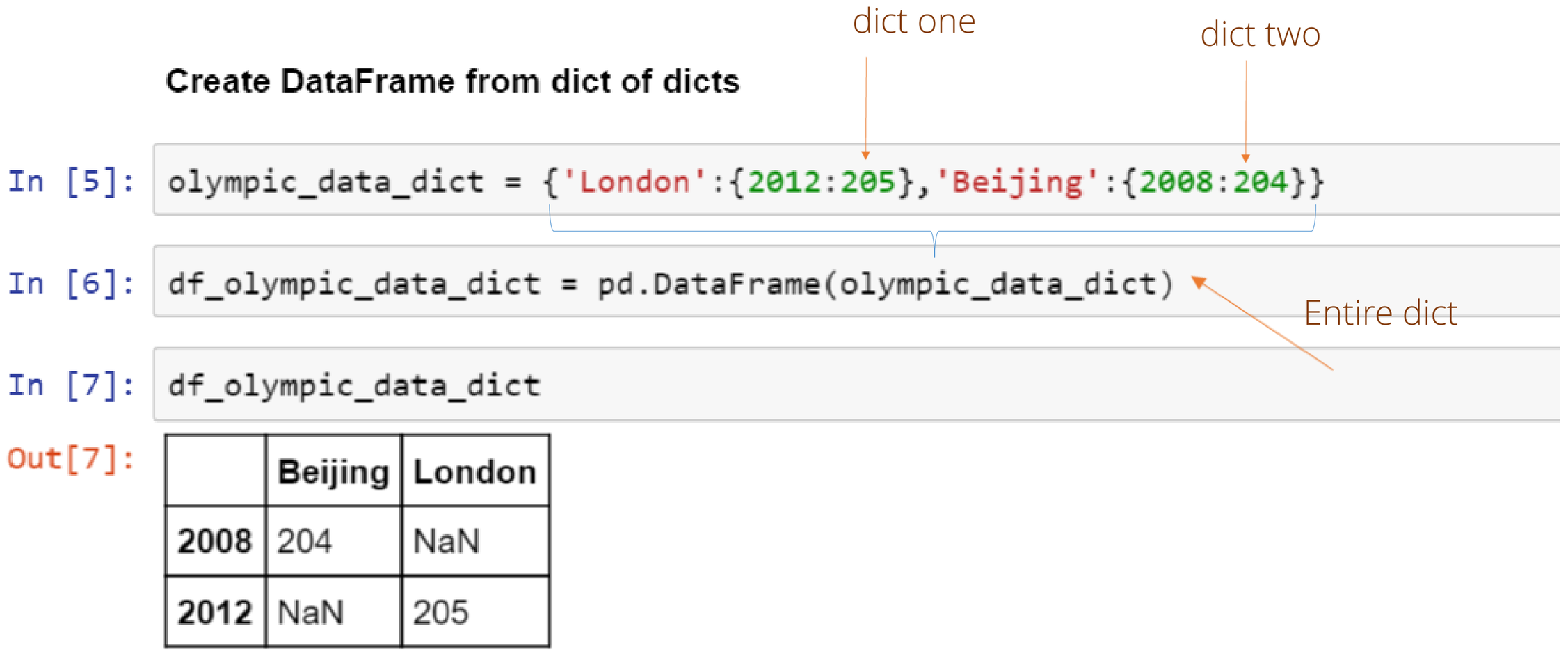
In [4]:
```
df_olympic_data
```

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

Create DataFrame from dict

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



	Beijing	London
2008	204	NaN
2012	NaN	205

View DataFrame

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

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

```
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
```

```
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>
```

Create 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

Create 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
```

Create an ndarrays with years

Create a dict with the ndarray

Pass this dict to a new DataFrame

Out[16]:

	year
0	2012
1	2008
2	2004
3	2006

Create DataFrame from DataFrame

Create DataFrame from DataFrame object

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

Create a DataFrame from a DataFrame

```
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

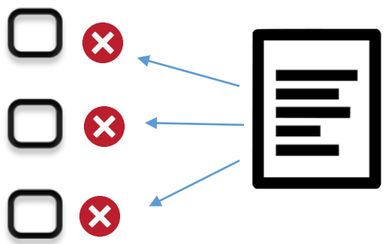
Demo 01—View and Select Data

Demonstrate how to view and select data in a DataFrame.

DATA SCIENCE

Missing Values

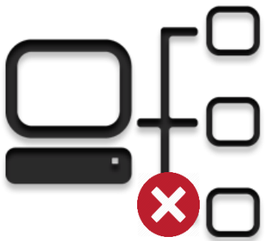
Various factors may lead to missing data values:



Data not provided by the source

Software issue

Data integration issue



Network issue

Handle Missing Values

It’s difficult to operate on 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
```

Handle 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
```

Handle 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)
```

Fill the missing values with zero

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

```
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
```

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 with 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

This example shows data operations with different 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 test

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

Apply the max function to find the maximum score

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

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

Apply the mean function to find the average score

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

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

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

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

Data Operation Using Groupby

This example shows how to operate data using the groupby function.

```
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
```

Group the DataFrame with the first name

Out[19]:

	first	last
0	George	Bush
4	George	Washington

Data Operation – Sorting

This example shows how to sort data

```
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

Demo 02—Data Operations

Demonstrate how to perform data operations.

DATA SCIENCE

Data Standardization

This example shows how to standardize a dataset.

```
In [11]: def standardize_tests(test):
             return (test-test.mean())/ test.std()
```

Create a function to return the standardize value

```
In [12]: standardize_tests(df_test_scores['Test1'])
```

```
Out[12]: Jack       1.210437
         Lewis      0.292174
         Patrick   -0.626088
         Rich       0.626088
         Kelly      0.125218
         Paula     -1.627829
         Name: Test1, dtype: float64
```

```
In [13]: def standardize_test_scores(datafrm):
             return datafrm.apply(standardize_tests)
```

Apply the function to the entire dataset

```
In [14]: standardize_test_scores(df_test_scores)
```

Out[14]:

	Test1	Test2
Jack	1.210437	-0.935276
Lewis	0.292174	1.438886
Patrick	-0.626088	0.791387
Rich	0.626088	-1.151109
Kelly	0.125218	-0.287777
Paula	-1.627829	0.143889

Standardized test data is applied for the entire DataFrame

Knowledge Check

KNOWLEDGE CHECK

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

a. Series with sliced index from 3 to 9

b. dict of index position 3 and index position 9

c. DataFrame of sliced rows index from 3 to 9

d. DataFrame with data elements at index 3 to index9

KNOWLEDGE CHECK

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

a. Series with sliced index from 3 to 9

b. dict of index position 3 and index position 9

c. DataFrame of sliced rows index from 3 to 9

d. DataFrame with data elements at index 3 to index9

The correct answer is . c

Explanation: 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

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

What does the fillna() method do?

a. Fills all NaN values with zeros

b. Fills all NaN values with One

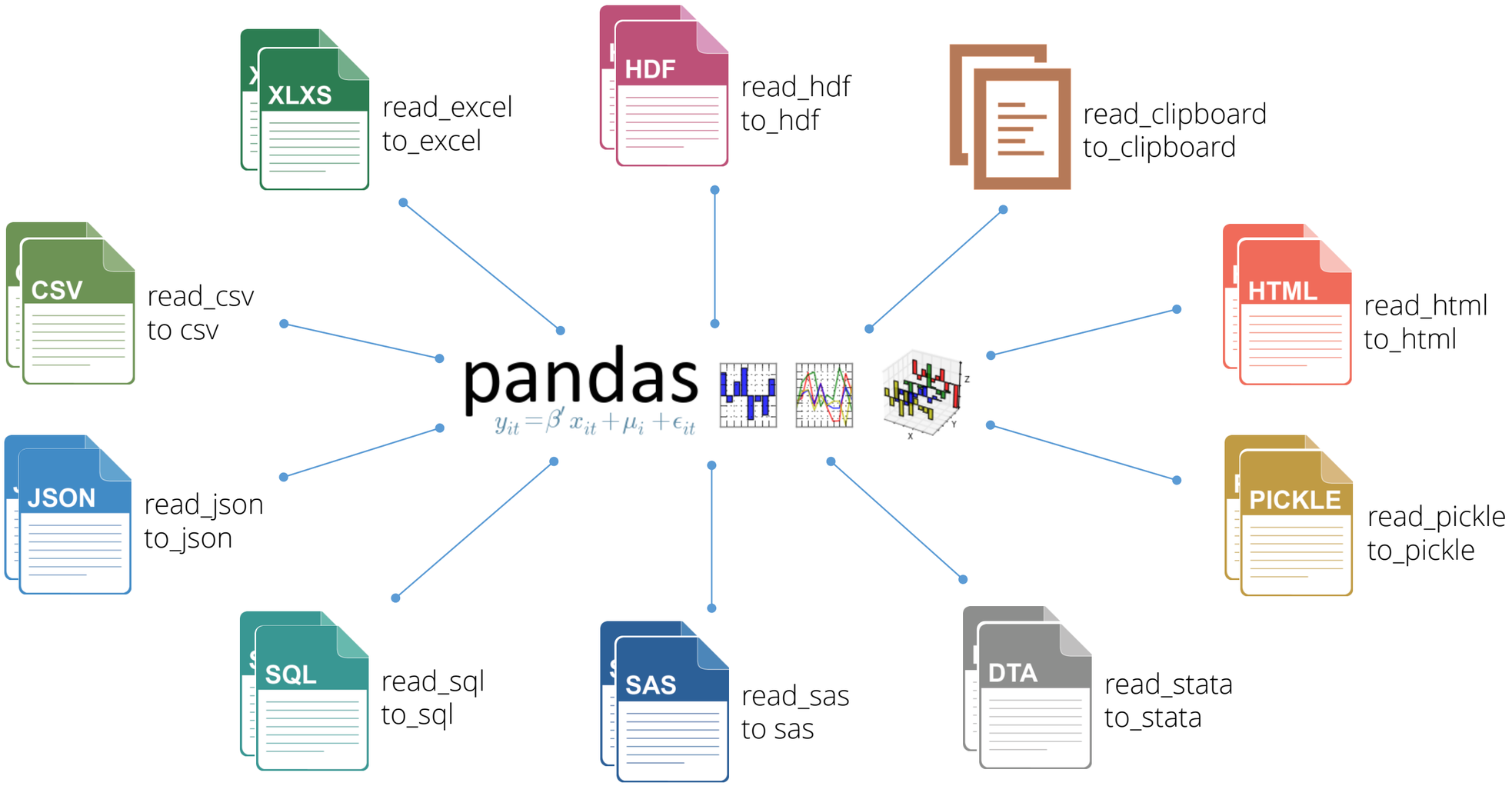
c. Fills all NaN values with values mentioned in the parenthesis

d. Drops NaN values from the dataset

The correct answer is . c

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

File Read and Write Support



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

Activity—Sequence it Right!

The code here is buggy. You have to correct its sequence to debug it. To do that, click any two code snippets, which you feel are out of place, to swap their places.

1

```
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']
```

2

```
print movie_grade(5)
```

```
A
```

3

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

4

```
df_movie_rating.applymap(movie_grade)
```

Click any two code snippets to swap them.

Assignment

Assignment 01

simplilearn

Problem | Instructions

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

1. View
a. Aircraft make 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”

Assignment 01

Problem | Instructions

Instructions to perform the assignment:

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

Common instructions:

- If you are new to Python, download the “Anaconda Installation Instructions” document from the “Resources” tab to view the steps for installing Anaconda and the Jupyter notebook.
- Download the “Assignment 01” notebook and upload it on the Jupyter notebook to access it.
- Follow the cues provided to complete the assignment.

Assignment

Assignment 02

Problem | Instructions

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

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

Assignment 02

Problem | Instructions

Instructions to perform the assignment:

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

Common instructions:

- If you are new to Python, download the “Anaconda Installation Instructions” document from the “Resources” tab to view the steps for installing Anaconda and the Jupyter notebook.
- Download the “Assignment 02” notebook and upload it on the Jupyter notebook to access it.
- Follow the cues provided to complete the assignment.

Quiz

QUIZ 1

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

a. Series

b. DataFrame

c. Panel

d. PanelND

QUIZ 1

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**.

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

QUIZ 2

Which method is used for label-location indexing by label?

a. iat

b. iloc

c. loc

d. std

QUIZ 2

Which method is used for label-location indexing by label?

a. iat

b. iloc

c. loc

d. std

The correct answer is **c**.

Explanation: The loc method is used to for label-location indexing by label; iat is strictly integer location and iloc is integer-location-based indexing by position.

QUIZ 3

While viewing a dataframe, head() method will _____.

a. return only the first row

b. return only headers or column names of the DataFrame

c. return the first five rows of the DataFrame

d. throw an exception as it expects parameter(number) in parenthesis

QUIZ 3

While viewing a dataframe, head() method will _____.

a. return only the first row

b. return only headers or column name of the DataFrame

c. return the first five rows of the DataFrame

d. throw an exception as it expects parameter(number) in parenthesis

The correct answer is **c**.

Explanation: The default value is 5 if nothing is passed in head method. So it will return the first five rows of the DataFrame.

Key Takeaways

Let us take a quick recap of what we have learned in the lesson:

- Pandas is an open source library for data analysis and is an efficient data wrangling tool in Python.
- The four main libraries of Pandas are Series, DataFrame, Panel, and Panel 4D.
- DataFrame is a two-dimensional labeled data structure with columns of potentially different data types.
- To access data elements in a series, 'loc' and 'iloc' methods can be used.

Key Takeaways

- The 'iat' method enables selection of elements in a DataFrame by index position and returns the corresponding data element.
- Missing data values in Pandas can be resolved through two built-in methods such as dropna and fillna.
- Pandas supports multiple files for data analysis such as Excel, PyTables, Clipboard, HTML, pickle, dta, SAS, SQL, JSON, and CSV.

This concludes “Data Manipulation with Pandas.”

The next lesson is “Machine Learning with SciKit Learn.”