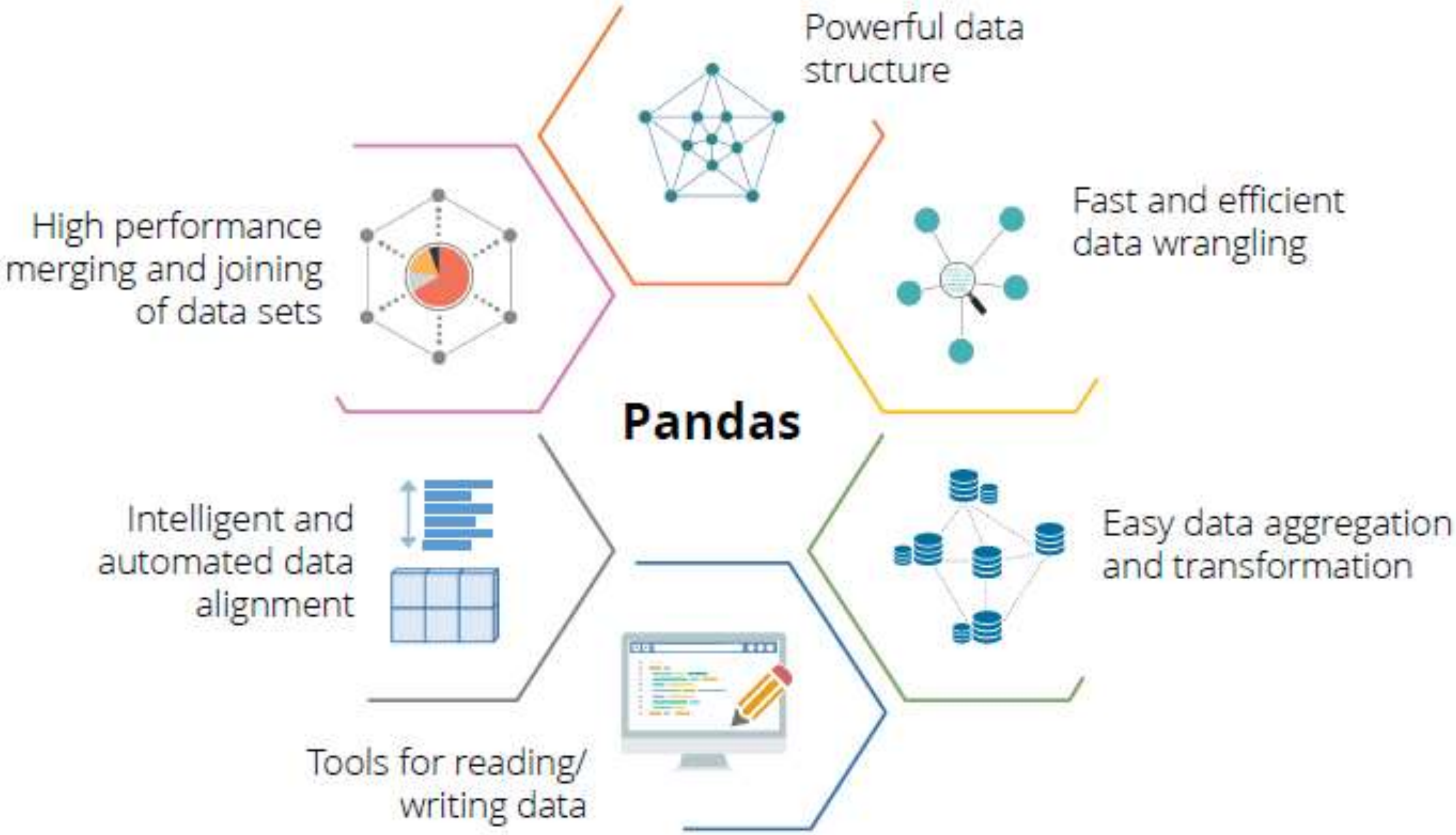
Pandas

Data Manipulation with 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:

Series

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

Data Frame

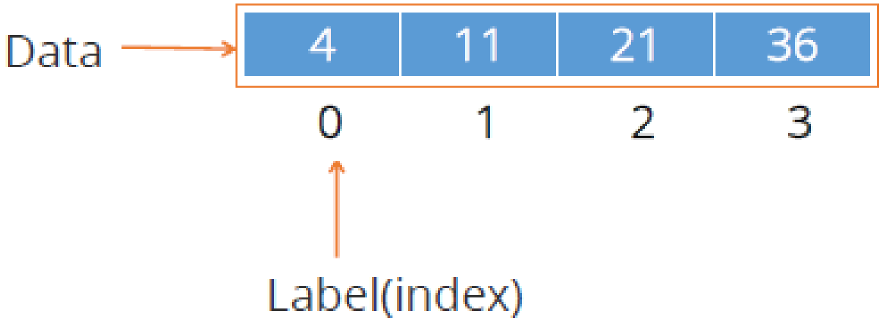
- **Two**-dimensional labeled array
- Supports multiple data types
- Input can be a Series
- Input can be another DataFrame

Panel

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

Understanding Series

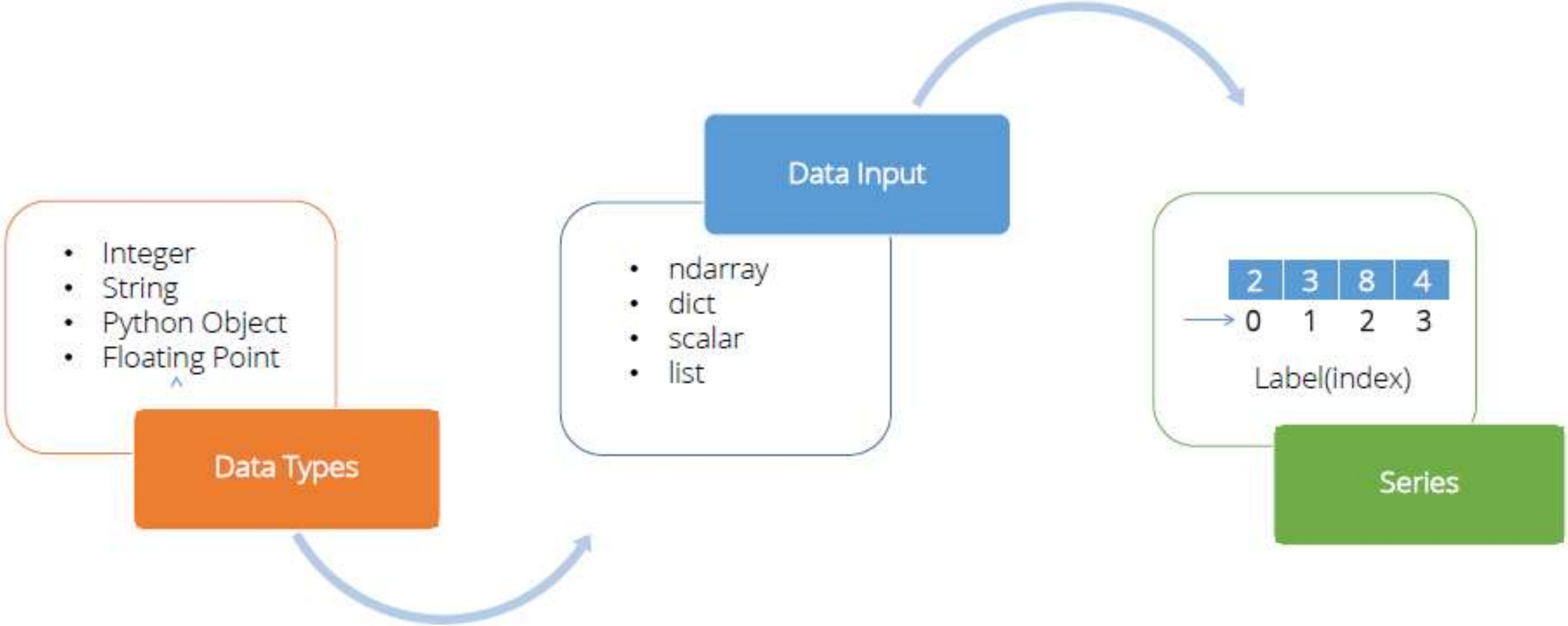
Series is a one-dimensional array-like object containing data and labels (or index).



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

Basic Method

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

→

4	11	21	36

Series

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
```
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
```
Index

Data value

Data type

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

Data Science

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 Scalar

Scalar input

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

```
In [32]: scalar_series
```

Index

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

Data

index

Data type

Data Science

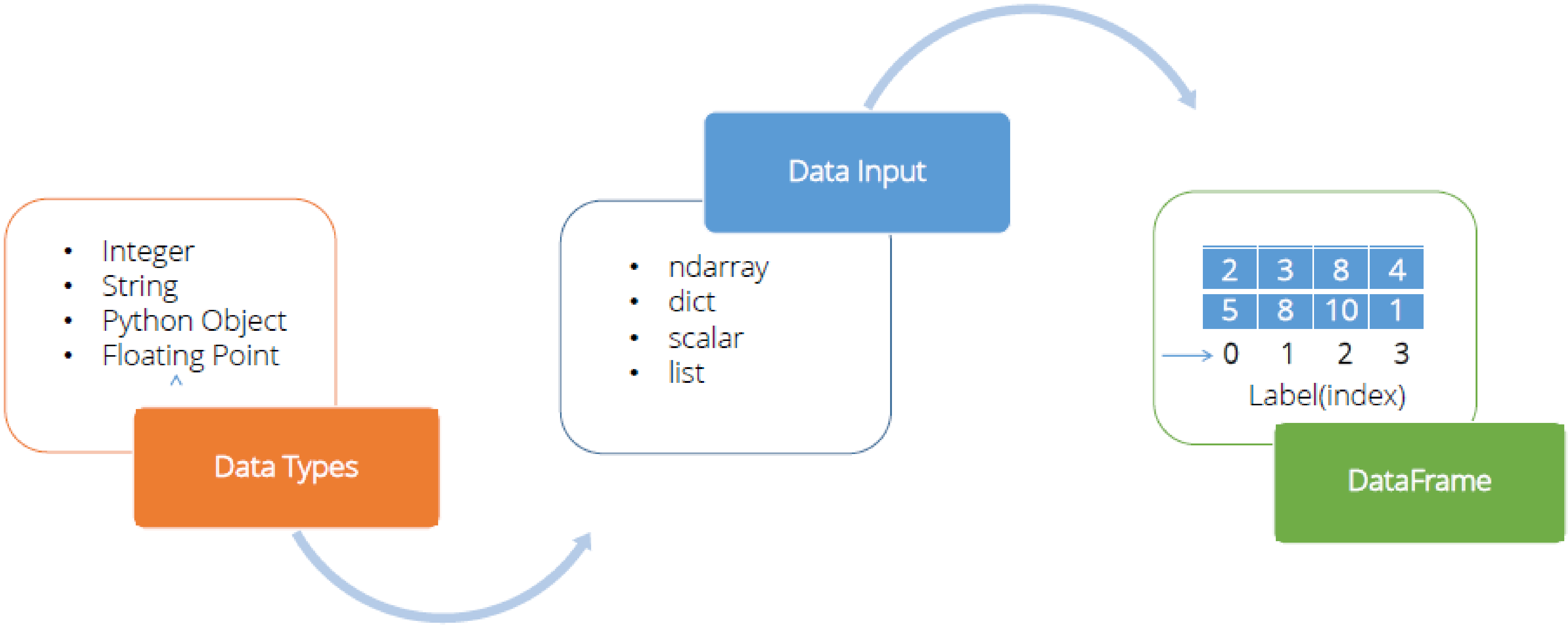
Accessing Elements in Series

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
```

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

Data Science

Create DataFrame from dict

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

dict one

dict two

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)
```

Entire dict

```
In [7]: df_olympic_data_dict
```

Out[7]:

	Beijing	London
2008	204	NaN
2012	NaN	205

View Data Frame

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>
```

Data Science

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}
```
Create an ndarrays with years

Create a dict with the ndarray

In [15]:
```
df_ndarray = pd.DataFrame(dict_ndarray)
```
Pass this dict to a new DataFrame

In [16]:
```
df_ndarray
```

Out[16]:

	year
0	2012
1	2008
2	2004
3	2006

Create DataFrame from DataFrame

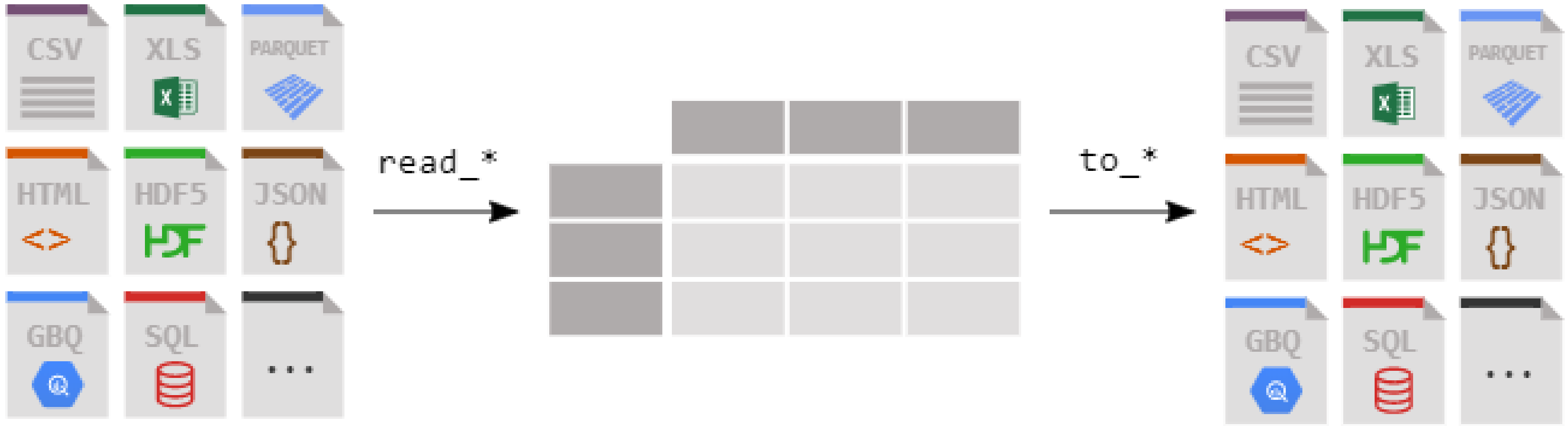
Create DataFrame from DataFrame object

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

How do I read and write tabular data?




```
In [2]: titanic = pd.read_csv("data/titanic.csv")
```

pandas provides the `read_csv()` function to read data stored as a csv file into a pandas `DataFrame`. pandas supports many different file formats or data sources out of the box (csv, excel, sql, json, parquet, …), each of them with the prefix `read_*`.

How do I read and write tabular data?

Make sure to always have a check on the data after reading in the data. When displaying a `DataFrame`, the first and last 5 rows will be shown by default:

```
In [3]: titanic
Out[3]:
     PassengerId  Survived  Pclass                                               Name     Sex  ...
0              1         0       3                            Braund, Mr. Owen Harris    male  ...
1              2         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  ...
2              3         1       3                             Heikkinen, Miss. Laina  female  ...
3              4         1       1       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  ...
4              5         0       3                           Allen, Mr. William Henry    male  ...
..           ...       ...     ...                                                ...     ...  ...
886          887         0       2                              Montvila, Rev. Juozas    male  ...
887          888         1       1                       Graham, Miss. Margaret Edith  female  ...
888          889         0       3           Johnston, Miss. Catherine Helen "Carrie"  female  ...
889          890         1       1                              Behr, Mr. Karl Howell    male  ...
890          891         0       3                                Dooley, Mr. Patrick    male  ...

[891 rows x 12 columns]
```

Data Science

head()

```
In [4]: titanic.head(8)
Out[4]:
   PassengerId  Survived  Pclass                                               Name     Sex  ...
0            1         0       3                            Braund, Mr. Owen Harris    male  ...
1            2         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  ...
2            3         1       3                             Heikkinen, Miss. Laina  female  ...
3            4         1       1       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  ...
4            5         0       3                           Allen, Mr. William Henry    male  ...
5            6         0       3                                   Moran, Mr. James    male  ...
6            7         0       1                            McCarthy, Mr. Timothy J    male  ...
7            8         0       3                     Palsson, Master. Gosta Leonard    male  ...

[8 rows x 12 columns]
```

To see the first N rows of a `DataFrame`, use the `head()` method with the required number of rows (in this case 8) as argument.

Head() , tail()

Note

Interested in the last N rows instead? pandas also provides a `tail()` method. For example, `titanic.tail(10)` will return the last 10 rows of the DataFrame.

A check on how pandas interpreted each of the column data types can be done by requesting the pandas `dtypes` attribute:

```
In [5]: titanic.dtypes
Out[5]:
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

For each of the columns, the used data type is enlisted. The data types in this `DataFrame` are integers (`int64`), floats (`float64`) and strings (`object`).

.dtype

Note

When asking for the `dtypes`, no brackets are used! `dtypes` is an attribute of a `DataFrame` and `Series`. Attributes of `DataFrame` or `Series` do not need brackets. Attributes represent a characteristic of a `DataFrame`/`Series`, whereas a method (which requires brackets) *do* something with the `DataFrame`/`Series` as introduced in the first tutorial.

To_excel()

```
In [6]: titanic.to_excel('titanic.xlsx', sheet_name='passengers', index=False)
```

Whereas `read_*` functions are used to read data to pandas, the `to_*` methods are used to store data. The `to_excel()` method stores the data as an excel file. In the example here, the `sheet_name` is named *passengers* instead of the default *Sheet1*. By setting `index=False` the row index labels are not saved in the spreadsheet.

The equivalent read function `read_excel()` will reload the data to a `DataFrame`:

.info()

The method `info()` provides technical information about a `DataFrame`, so let's explain the output in more detail:

- It is indeed a `DataFrame`.
- There are 891 entries, i.e. 891 rows.
- Each row has a row label (aka the `index`) with values ranging from 0 to 890.
- The table has 12 columns. Most columns have a value for each of the rows (all 891 values are `non-null`). Some columns do have missing values and less than 891 `non-null` values.
- The columns `Name`, `Sex`, `Cabin` and `Embarked` consists of textual data (strings, aka `object`). The other columns are numerical data with some of them whole numbers (aka `integer`) and others are real numbers (aka `float`).
- The kind of data (characters, integers,...) in the different columns are summarized by listing the `dtypes`.
- The approximate amount of RAM used to hold the DataFrame is provided as well.

Remember

- Getting data in to pandas from many different file formats or data sources is supported by `read_*` functions.
- Exporting data out of pandas is provided by different `to_*` methods.
- The `head`/`tail`/`info` methods and the `dtypes` attribute are convenient for a first check.

How to select a subset of a DataFrame?

```
In [4]: ages = titanic["Age"]

In [5]: ages.head()
Out[5]:
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
Name: Age, dtype: float64
```

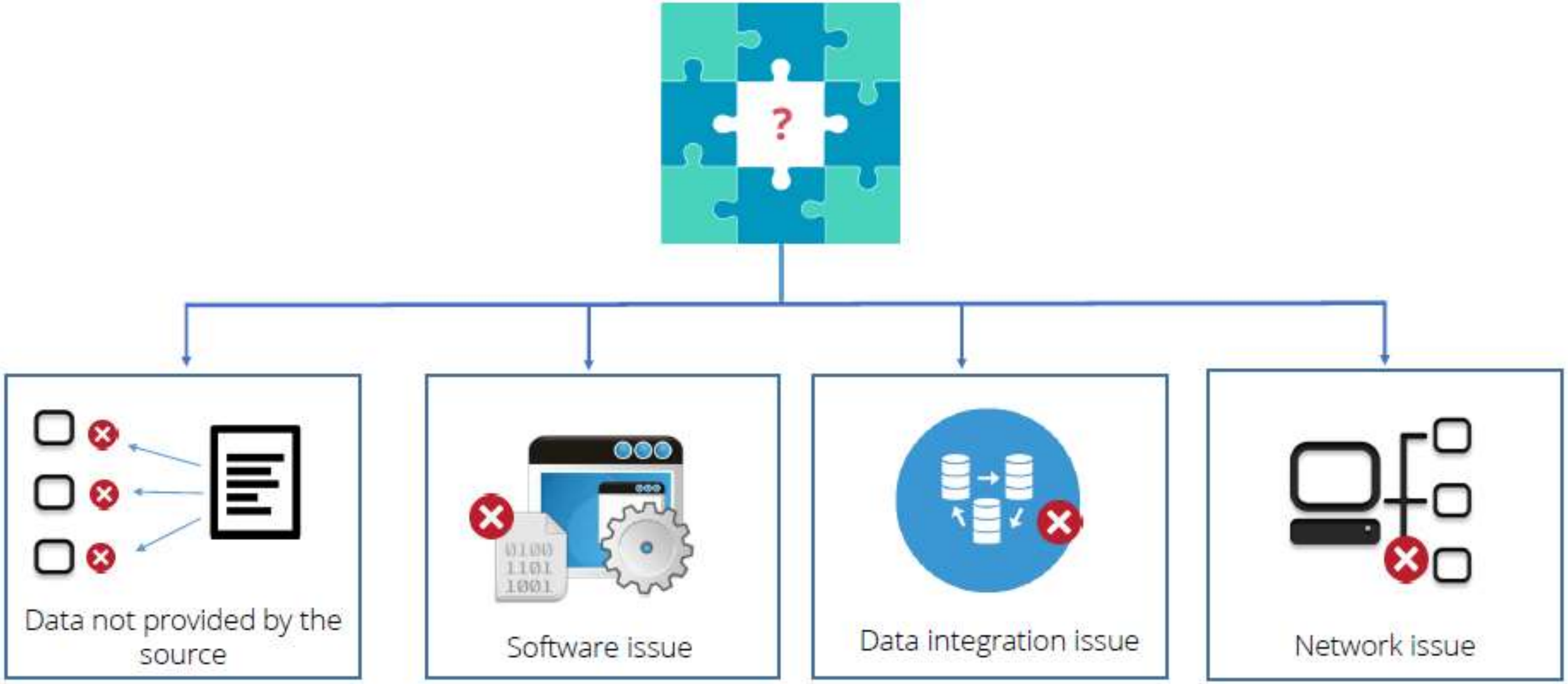
To select a single column, use square brackets `[]` with the column name of the column of interest.

Each column in a `DataFrame` is a `Series`. As a single column is selected, the returned object is a pandas `Series`. We can verify this by checking the type of the output:

```
In [6]: type(titanic["Age"])
Out[6]: pandas.core.series.Series
```

Missing Values

Various factors may lead to missing data values:



Handling 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
```

Data Science

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
```

Data Science

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)
```
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
```

Handling 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 Science

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 Science

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 Science

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


?
Quiz

Data Science

Which of the followings is used to store Two-dimensional data?

a. Series

b. DataFrame

c. Panel

d. PanelND

Data Science

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

a. iat

b. iloc

c. loc

d. std

Data Science

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

Data Science

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

Data Science

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