



Week3: Data Ingestion and Pandas

Aug 2024

Dept. of ICT,AIT

Outline

- Pandas
- Data Ingestion
- Web-scraping
- Reading Data from a Web API

Pandas (Pan(el)-da(ta)-s)



- Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures.
- Pandas is a widely used tool in data munging/wrangling
- Similar to how we use Excel to manipulate data
- Benefits of using Pandas over Excel
 - Easier to handle HTML, SQL, CSV
 - Automation, Speed, and Capacity
 - Open source

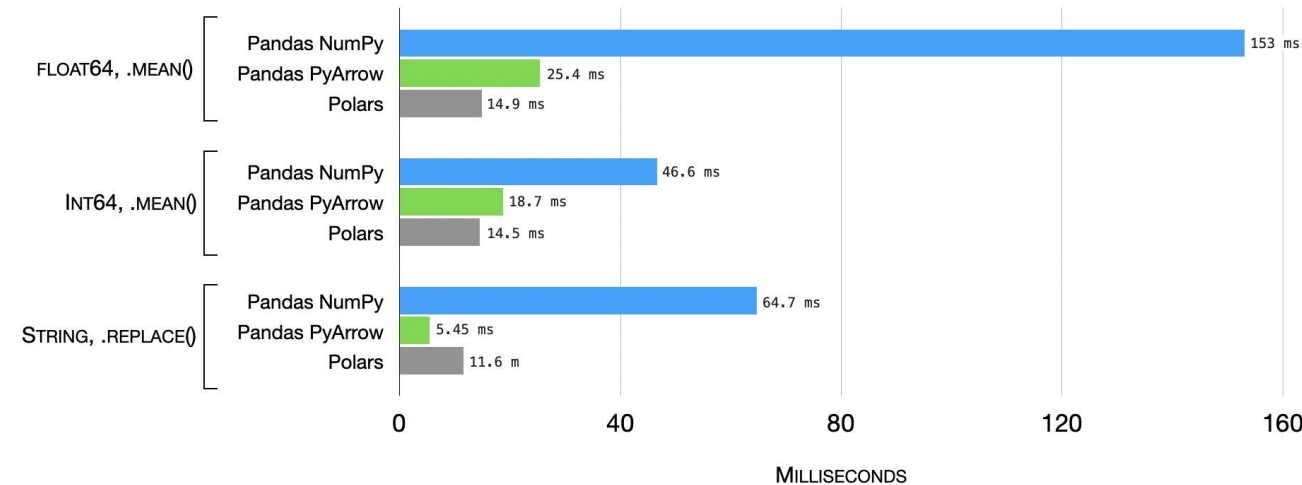


Pandas (**P**an(**e**l)-**d**a(**t**a)-**s**)

- Latest release v2.2.2 (April 2024)
- Major update in pandas2.0 (April 2023)
 - Improved Performance: Faster and More Memory-efficient Operations
 - **In-memory analytics** using PyArrow (Apache Arrow Python)
 - PyArrow is a Python library (built on top of Arrow) that provides an interface for handling large datasets using Arrow memory structures.



- Apache Arrow is a development platform for in-memory analytics. It contains a set of technologies that enable big data systems to store, process and move data fast.






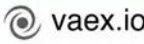



Polars is a Rust-based data manipulation library for Python that provides a DataFrame API similar to pandas⁴

Key Benefits of Pandas

- **Data Structures:** DataFrame (for tabular data) and Series (for one-dimensional data)—which simplify the management and analysis of structured data.
- **Ease of Use:** intuitive API, making it easier for users to perform complex data manipulations with less code compared to traditional Python methods.
- **Performance:** Built on optimized C and C++ code (with PyArrow for in-memory analytics), allowing it to handle large datasets efficiently.
- **Robust Functionality:** Pandas includes a wide range of functions for data cleaning, transformation, and analysis. It supports operations such as filtering, grouping, aggregating, and merging datasets, which are essential for comprehensive data analysis.
- **Integration with Other Libraries:** integrates seamlessly with other popular Python libraries, such as NumPy, Matplotlib and Seaborn for data visualization, and Scikit-learn for machine learning.
- **Versatile Data Import/Export:** Pandas can read from and write to various data formats, including CSV, Excel, and SQL databases, making it a flexible tool for data scientists.
- **Time Series Support:** The library offers built-in functionalities for time series analysis,

PANDAS VS ALTERNATIVES

	What it is	Good for	Not best fit for
	Go-to Python library for data analysis	Data manipulation and further analysis in different domains	Very large datasets, unstructured data
	Python library for numerical computing	Mathematical operations on arrays and matrices	Non-numerical data types, data manipulation tasks
	Python API for Apache Spark	Big data processing in distributed environment	Small-scale data tasks
	Python library for parallel and distributed computing	Processing of larger-than-memory datasets	Data manipulation tasks
	Python library for distributed computing of Pandas DataFrames	Manipulating datasets from 1MB to 1TB+	Small-scale data tasks
	Python library for larger-than-memory Pandas DataFrames	Visualizing and exploring big tabular datasets	Data manipulation tasks
	Statistical programming language	Data mining, data wrangling, data visualization, machine learning operations	Basic data manipulation tasks, big data projects

Aim of our study for Pandas

- Understand and create DataFrame objects
- Subsetting data and indexing data
- Arithmetic functions, and mapping with pandas
- Managing index
- Building style for visual analysis

Pandas deals with 2 different data structures

Data Structure	Dimensions	Description
Series	1	1D labelled homogeneous array, size-immutable
Data Frames	2	General 2D labeled, size-mutable tabular structure with potentially heterogeneously typed columns.

Pandas Series

A series consisting of two components: 1D data and the index

```
class pandas.Series(data, index, dtype, name, copy )
```

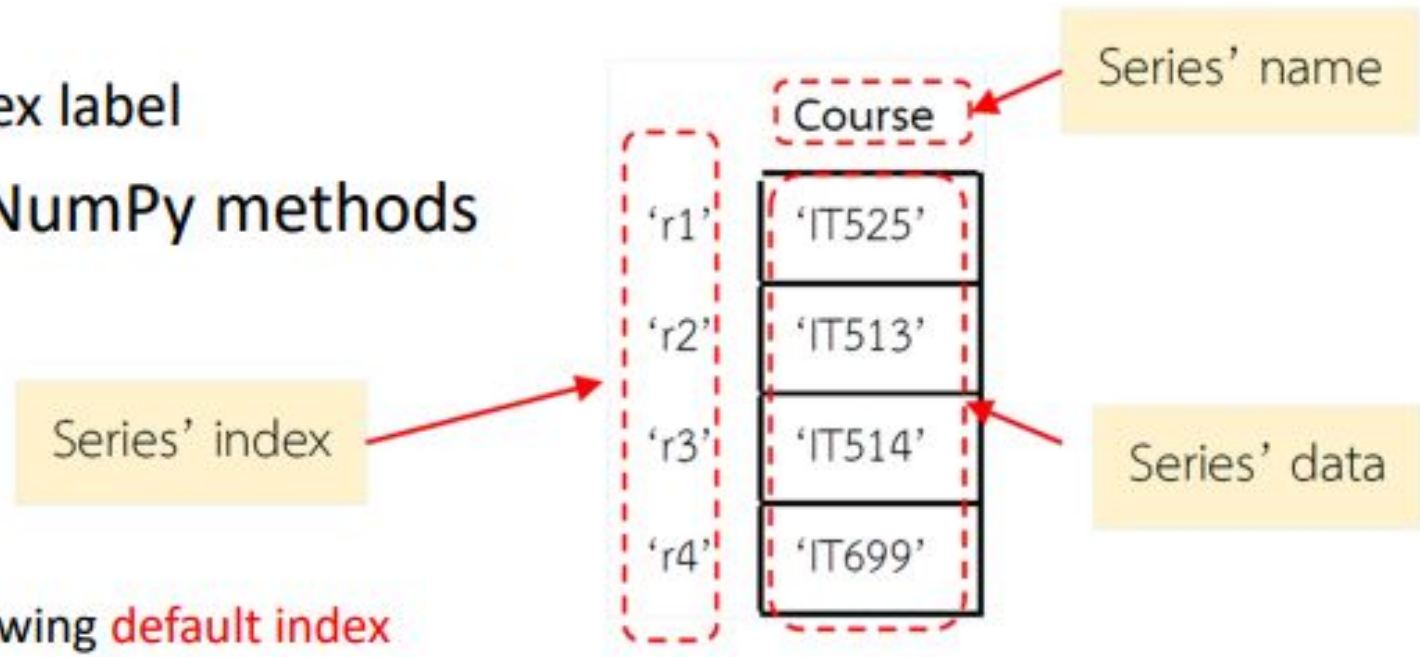
```
import pandas as pd  
ser = pd.Series(data, index = idx)
```

- A 1-dimensional labeled array
- Supports many data types
- Axis labels → index
 - get and set values by index label
- Valid argument to most NumPy methods

Data can be:

- ndarray
- Python dictionary
- list, a scalar value

If an index is not specified, the following **default index** [0,... n-1] will be created



Series `class pandas.Series(data, index, dtype, name)`

Parameters:

- **data**: array-like, Iterable, dict, or scalar value. Contains data stored in Series. If data is a dict, argument order is maintained.
- **index**: array-like or Index (1d). Values must have the same length as data. Non-unique index values are allowed.
 - Will default to **RangeIndex** (0, 1, 2, ..., n) if not provided.
 - If data is dict-like and index is none, then the keys in the data are used as the index. If the index is not None, the resulting Series is **reindexed** with the index values.
- **dtype**: Data type for the output Series. If not specified, this will be inferred from data
- **name**: name to give to the Series (default None)

Pandas Data Type

Pandas dtype	Python type	NumPy type	Usage
object	str or mixed	string_, unicode_, mixed types	Text or mixed numeric and non-numeric values
int64	int	int_, int8, int16, int32, int64, uint8, uint16, uint32, uint64	Integer numbers
float64	float	float_, float16, float32, float64	Floating point numbers
bool	bool	bool_	True/False values
datetime64	NA	datetime64[ns]	Date and time values
timedelta[ns]	NA	NA	Differences between two datetimes
category	NA	NA	Finite list of text values

A column could include integers, floats and strings which collectively are labeled as an **object**.

`pandas.RangeIndex(start, stop, step, dtype)`

- Immutable Index implementing a monotonic integer range.
- RangeIndex is a memory-saving special case of an Index limited to representing monotonic ranges with a 64-bit dtype (May in some instances improve computing speed)
- This is the default index type used by DataFrame and Series when no explicit index is provided by the user.

Parameters:

- startint (default: 0), range, or other RangeIndex instance. If int and “stop” is not given, interpreted as “stop” instead.
- stopint (default: 0)
- stepint (default: 1)
- dtype`np.int64`

Pandas DataFrame

- A 2-dimensional labeled data structure
- A dictionary of Series objects
 - Columns can be of potentially different types
 - Optional parameters for fine-tuning:
 - index (row labels)
 - columns (column labels)
- Pandas provides many constructors to create DataFrames!

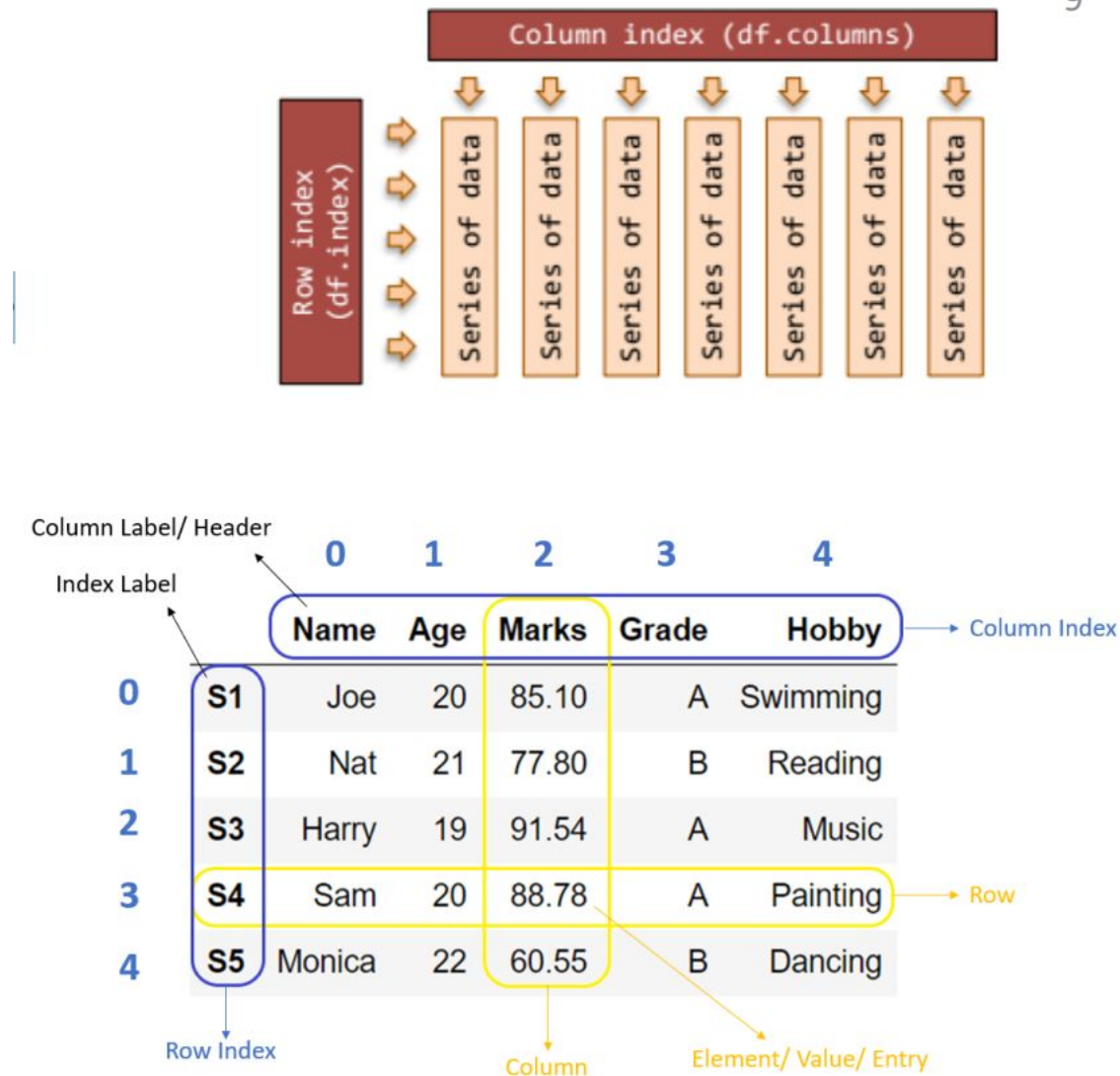


Image from: <https://pynative.com/python-pandas-dataframe/>

DataFrame `class pandas.DataFrame(data, index, column, dtype)`

Parameters:

- **data**: Series, arrays, constants, dataclass or list-like objects. If data is a dict, column order follows insertion-order.
- **index**: Index or array-like. Index to use for resulting frame. Will default to RangeIndex if no indexing information part of input data and no index provided.
- **column**: Index or array-like. Column labels to use for resulting frame when data does not have them, defaulting to RangeIndex(0, 1, 2, ..., n). If data contains column labels, will perform column selection instead.
- **datatype**: Data type to force. Only a single dtype is allowed. If None,

pandas.DataFrame

```
matrix_data = np.random.randint(1,20,size=20).reshape(5,4)
row_labels = ['A', 'B', 'C','D','E']
column_headings = ['W','X','Y','Z']
df2 = pd.DataFrame(data = matrix_data, index = row_labels, \
                    columns=column_headings)
```

df2

	W	X	Y	Z
A	17	9	8	15
B	17	4	15	18
C	12	1	19	4
D	8	19	11	10

Popular Attributes and Methods

Attribute/Method	Description
dtype	data type of values in series
empty	True if series is empty
size	number of elements
values	return values as ndarray
head()	First n elements
tail()	Last n elements
info	Print a concise summary of a Series.

Indexing and Slicing Columns

- There are two methods for indexing and slicing columns in a DataFrame.
- They are as follows:
 - The **DOT** method: The DOT method is good if you want to find a specific element.
 - The **bracket** method: You can access the data by the generic name/header of the column

Dot

```
df1.X
```

```
0    20
1    35
2    37
3    95
4    36
```

```
Name: X, dtype: int32
```

Bracket

```
df1['X']
```

```
0    20
1    35
2    37
3    95
4    36
```

```
Name: X, dtype: int32
```

```
df1
```

```
11
```

```
W  X  Y  Z
```

```
0  18  20  31  81
1  11  35  61  86
2  92  37  50  60
3  91  95  85  44
4  40  36   3  61
```

```
df1[['X', 'Y']]
```

```
X  Y
```

```
0  20  31
1  35  61
2  37  50
3  95  85
4  36   3
```

Indexing and Slicing Rows

```
matrix_data = np.random.randint(1,10,size=20).reshape(5,4)
row_labels = ['A','B','C','D','E']
column_headings = ['W','X','Y','Z']
df2 = pd.DataFrame(data=matrix_data, index=row_labels, \
                    columns=column_headings)
```

- Indexing and slicing rows in a DataFrame can also be done using the following methods:
 - The label-based **loc** method: you can access the data by **the generic name of the row**
 - The index-based **iloc** method: you can access the rows by their **numerical index**.

loc

```
df2.loc['C']
```

W	5
X	6
Y	9
Z	6

Name: C, dtype: int32

```
df2.loc[['C','D']]
```

	W	X	Y	Z
C	5	6	9	6
D	9	4	2	8

iloc

```
df2.iloc[2]
```

W	5
X	6
Y	9
Z	6

Name: C, dtype: int32

```
df2.iloc[[2,3]]
```

	W	X	Y	Z
C	5	6	9	6
D	9	4	2	8

df2

	W	X	Y	Z
A	7	8	5	8
B	2	1	3	7
C	5	6	9	6
D	9	4	2	8
E	3	8	9	5

Indexing and Slicing Rows

`loc[]` (label based)

- A single label, e.g. 5 or 'a' (Note that 5 is interpreted as a label of the index, not an integer position along the index).
- A list or array of labels ['a', 'b', 'c'].
- A slice object with labels 'a':'f' (include both the start and the stop)
- A boolean array (any NA values will be treated as False).

`iloc[]` (integer position based (from 0 to length-1 of the axis))

- An integer e.g. 5.
- A list or array of integers [4, 3, 0].
- A slice object with ints 1:7. (inclusive?)
- A boolean array (any NA values will be treated as False).

Data Ingestion

Data Sources

- **Databases:** Most of the CRM, ERP, and other business operations tools store data in a database. it can be a traditional or NoSQL database.
- **Web services:** Many of the business operations tools, especially Software as a Services (SaaS) tools, make their data accessible through Application Programming Interfaces ([APIs](#)) instead of a database. API returns packets of data in formats such as JSON or XML
- **Data files:** A lot of data for prototyping data science models comes as data files. Data is stored in a flat file such as a .txt file, or a .csv file.
- **Web and document scraping:** Two other sources of data are the tables and text present on web pages (htm, htm).

Data Ingestion: read_csv(file_path, ...)

- **Input:** Path to a Comma Separated File (could be url)
- **Output:** Pandas DataFrame object containing contents of the file


Load CSV File to Python Pandas DataFrame

Use .txt to load a text file

```
pd.read_csv('Path to load CSV file\File Name.csv')
```

Path e.g.
D:\Python\Tutorial\

File name e.g.
Example1.csv



CSV Data in Microsoft Excel

	A	B	C
1	first_name	degree	age
2	Sam	PhD	25
3	Ziva	MBA	29
4	Kia		19
5	Robin	MS	21

In data, columns are separated by comma

Same Data in text editor (Notepad++, sublime)

```
Example1.csv
1 first_name,degree,age
2 Sam,PhD,25
3 Ziva,MBA,29
4 Kia,,19
5 Robin,MS,21
```

Data after loading in Jupyter using Pandas

	first_name	degree	age
0	Sam	PhD	25
1	Ziva	MBA	29
2	Kia	NaN	19
3	Robin	MS	21

```
# Code to read csv file
import pandas as pd
pd.read_csv(path\Filename.csv')
```

Data Ingestion: read_json

- **Input:** Path to a JSON file or a valid JSON String
- **Output:** Pandas DataFrame or a Series object containing the contents
- key -> column name
- value -> data corresponding to each column name

```
{  
  "student_id": [1, 2, 3],  
  "student_name": ["Alice", "Bob", "Chris"],  
  "student_info": [  
    {"gender": "F", "age": 20},  
    {"gender": "M", "age": 22},  
    {"gender": "M", "age": 33}  
  ]  
}
```

	student_id	student_name	student_info
0	1	Alice	{'gender': 'F', 'age': 20}
1	2	Bob	{'gender': 'M', 'age': 22}
2	3	Chris	{'gender': 'M', 'age': 33}

Data Ingestion: read_html

Read HTML tables into a list of DataFrame objects)

```
html_string = """
<table>
  <thead>
    <tr>
      <th>date</th>
      <th>name</th>
      <th>year</th>
      <th>cost</th>
      <th>region</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>2020-01-01</td>
      <td>Jenny</td>
      <td>1998</td>
      <td>0.2</td>
      <td>South</td>
    </tr>
    <tr>
      <td>2020-01-02</td>
      <td>Alice</td>
      <td>1992</td>
      <td>-1.34</td>
      <td>East</td>
    </tr>
    <tr>
      <td>2020-01-03</td>
      <td>Tomas</td>
      <td>1982</td>
      <td>1.00023</td>
      <td>South</td>
    </tr>
  </tbody>
</table>

```

- **Input:** A URL or a file or a raw HTML String
 - Read HTML tables
- **Output:** A list of Pandas DataFrames

Year 2020				
date	name	year	cost	region
2020-01-01	Jenny	1998	1.2	South
2020-01-02	Alice	1992	-1.34	East

html_string

dfs = pd.read_html(html_string, header=1)

	date	name	year	cost	region
0	2020-01-01	Jenny	1998	1.20	South
1	2020-01-02	Alice	1992	-1.34	East

dfs[0]

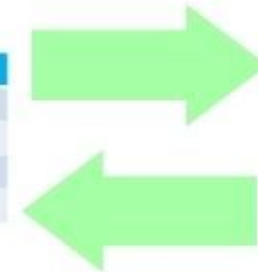
Data Ingestion: read_sql

- **Input1:** SQL Query
- **Input2:** Database connection
- **Output:** Pandas DataFrame object containing contents of the file

```
import mysql.connector
import pandas as pd
my_conn = mysql.connector.connect(
    host="localhost",
    user="userid",
    passwd="*****",
    database="my_tutorial"
)
##### end of connection #####
my_data = pd.read_sql("SELECT * FROM student WHERE class='four'",my_conn)
print(my_data)
```



Id	Name	Class	Mark	Gender
1	Alex	Four	68	Male
2	Sinee	Five	78	Female
3	Ron	Four	78	Male
4	Greek	Three	56	Female



	id	name	class	mark	sex
0	1	John Deo	Four	75	female
1	4	Krish Star	Four	60	female
2	5	John Mike	Four	60	female
3	6	Alex John	Four	55	male
4	10	Big John	Four	55	female
5	15	Tade Row	Four	88	male
6	16	Gimmy	Four	88	male
7	21	Babby John	Four	69	female
8	31	Marry Toeey	Four	88	male

Data Ingestion: read_sql_table

Read SQL database table into a DataFrame.

- **Input1:** Name of SQL table in database
- **Input2:** Database connection
- **Output:** Pandas DataFrame object containing contents of the table

select * from bookstore

ISBN_NO	SHORT_DESC	AUTHOR	PUBLISHER	PRICE
0201703092	The Practical SQL, Fourth Edition	Judith S. Bowman	Addison Wesley	39
0471777781	Professional Ajax	Jeremy McPeak, Joe Fawcett	Wrox	32
0672325764	Sams Teach Yourself XML in 21 Days, Third Edition	Steven Holzner	Sams Publishing	49
0764557599	Professional C#	Simon Robinson and Jay Glynn	Wrox	42
0764579088	Professional JavaScript for Web Developers	Nicholas C. Zakas	Wrox	35
1861002025	Professional Visual Basic 6 Databases	Charles Williams	Wrox	38
1861006314	GDI+ Programming: Creating Custom Controls Using C#	Eric White	Wrox	29

```
pd.read_sql_table('table_name', 'postgres:///db_name')
```

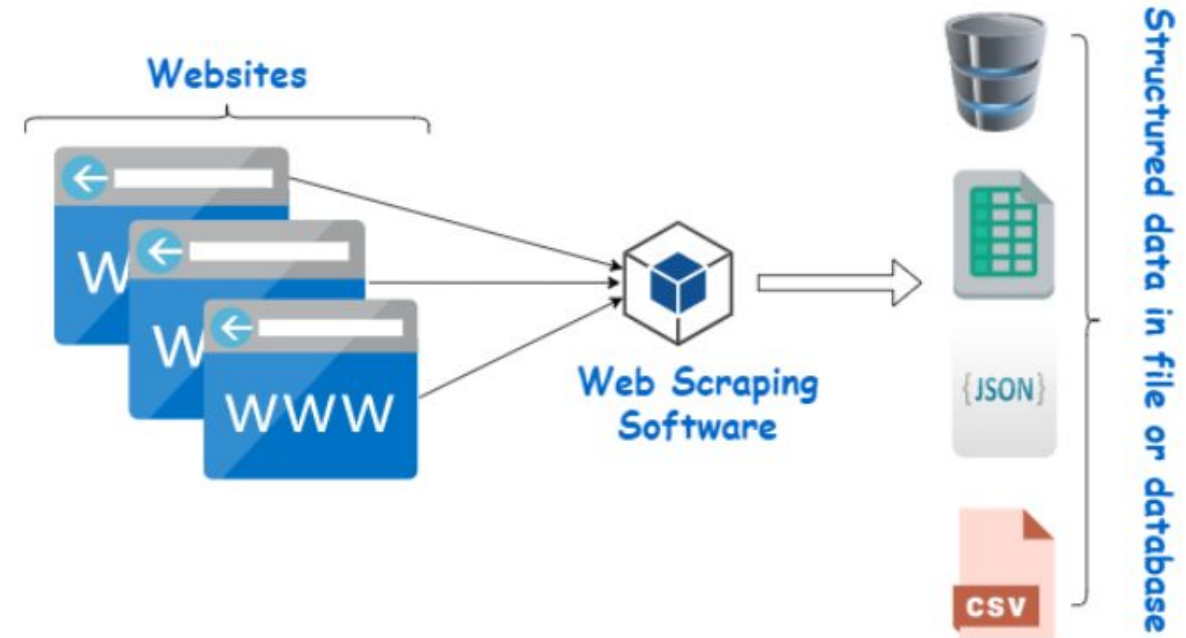
Others

There are many other methods available in Pandas to ingest data:

- Google Bigquery
- SAS files
- Excel tables
- Clipboard contents
- HDF formats

Web-scraping

- Web scraping is also known as web data extraction or web harvesting is the extraction of data from web.
- Web -scraping provides one of the great tools to automate most of the things a human does while browsing.
 - Data for Research
 - Products prices & popularity comparison
 - Sales and Marketing



BeautifulSoup Library

- BeautifulSoup (bs4) is a package that makes it simple to **navigate and extract** data from HTML documents.
- The BeautifulSoup class accepts two parameters to its constructor:
 - a string of HTML code
 - an HTML parser to use under the hood
- See <http://www.crummy.com/software/BeautifulSoup/bs4/doc/index.html> for the full documentation

BeautifulSoup

Requests Library

- The requests library is the de facto standard for **making HTTP requests in Python**.
- The HTTP request returns a Response Object with all the **response data (content, encoding, status, etc)**.
- It **abstracts the complexities** of making requests behind a beautiful, simple API so that users can focus on interacting with services and consuming data in their application.

References

- <https://pandas.pydata.org/docs/index.html>
- https://www.learnpython.org/en/Pandas_Basics
- <https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/>
- <https://www.geeksforgeeks.org/pandas-tutorial/>
- https://pbpython.com/pandas_dtypes.html
- <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>
- <https://data36.com/beautiful-soup-tutorial-web-scraping/>