

Lecture 3

*Data Collection I: **DataFrame**; Spyder IDE; Scrapping Web-tables with `pd.read_html()`*

Byeong-Hak Choe

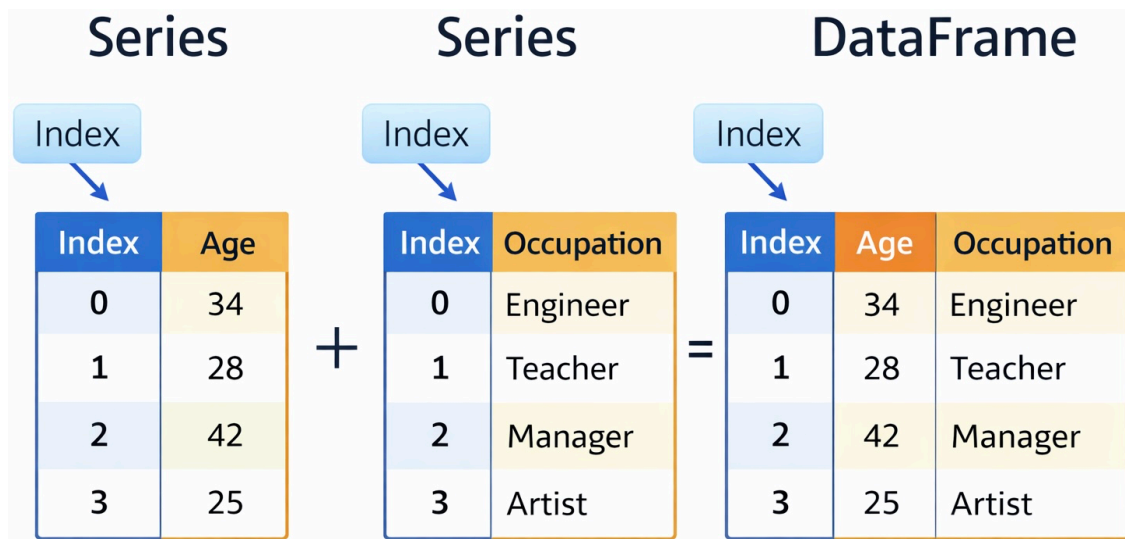
bchoe@geneseo.edu

SUNY Geneseo

February 9, 2026

Pandas **Series** and **DataFrame**

Pandas Series and DataFrame



- **Series**: A one-dimensional object containing a sequence of values (like a list).
- **DataFrame**: A two-dimensional table made of multiple **Series** columns sharing a common *index*.



Observations in DataFrame

- **Rows** in a **DataFrame** represent individual units or entities for which data is collected.
- **Examples:**
 - *Student Information*: Each row = one student
 - *Employee Information*: Each row = one employee
 - *Daily S&P 500 Index Data*: Each row = one trading day
 - *Household Survey Data*: Each row = one household

Variables in DataFrame

- **Columns** in a **DataFrame** represent attributes or characteristics measured across multiple *observations*.
- **Examples:**
 - *Student Data:* **Name, Age, Grade, Major**
 - *Employee Data:* **EmployeeID, Name, Age, Department**
 - *Customer Data:* **CustomerID, Name, Age, Income, HousingType**

Note

- In a **DataFrame**, a **variable** is a **column** of data.
- In general programming, a **variable** is the **name of an object**.

✨ Tidy DataFrame

Variables, Observations, and Values

The diagram shows three versions of a DataFrame with columns: country, year, cases, and population. The data is as follows:

country	year	cases	population
Afghanistan	1999	181	19987071
Afghanistan	2000	266	20095360
Brazil	1999	3737	17206362
Brazil	2000	8488	17404898
China	1999	21258	1272015272
China	2000	21366	128063583

Under the first table, vertical double-headed arrows connect the columns, labeled "variables".

Under the second table, horizontal double-headed arrows connect the rows, labeled "observations".

Under the third table, circles are placed over each cell, labeled "values".

- A **DataFrame** is *tidy* if it follows three rules:
 1. Each **variable** has its own *column*.
 2. Each **observation** has its own *row*.
 3. Each **value** has its own *cell*.
- A tidy **DataFrame** keeps your data organized, making it easier to understand, analyze, and share in any data analysis.



Spyder IDE

Anaconda Distribution

- **Anaconda is a free Python distribution** that includes Python, Conda (Python environment manager), and many commonly used data analytics packages.
- Install Anaconda from the official download page:
 - **Anaconda Distribution**
 - Click **Get Started**, then follow the installer steps for your operating system.



What is a Python Script?

- A Python script (*.py) is a plain-text file that contains Python code you can run from your computer (or an IDE like Spyder).
 - It is the standard format for writing **reusable Python programs**, such as data-cleaning pipelines, web scrapers, and automation tasks.
 - Scripts are commonly used in real-world analytics and software projects.
 - Compared to notebooks, scripts are typically better for **organized, production-style code** (functions, modules, and repeatable workflows).
- For **data collection** topics, we will write and run Python scripts mainly in **Spyder**, using **Anaconda Distribution** as our Python environment.



Script Editor

The screenshot displays the JupyterLab environment. The main window is the **Script Editor**, which contains a Python script named `google_trends_api.py`. A red box highlights the top toolbar of the Script Editor, which includes icons for creating, opening, and editing files. The script content includes imports for `pandas`, `numpy`, `pytrends`, and `time`, and defines variables for `us_states` and `years`.

To the right of the Script Editor is the **Variable Explorer**, which shows the current state of the variables in the environment. It contains a table with the following data:

Name	Type	Size	Value
df	DataFrame	(32, 5)	Column names: Unname...
keywords	list	2	['climate change', '...
month	list	12	[1, 2, 3, 4, 5, 6, 7...
pi	float	1	3.14192
sep	str	9	September
us_states	list	6	['US-CT', 'US-MA', '...
years	list	16	['2006-01-01 2006-12...

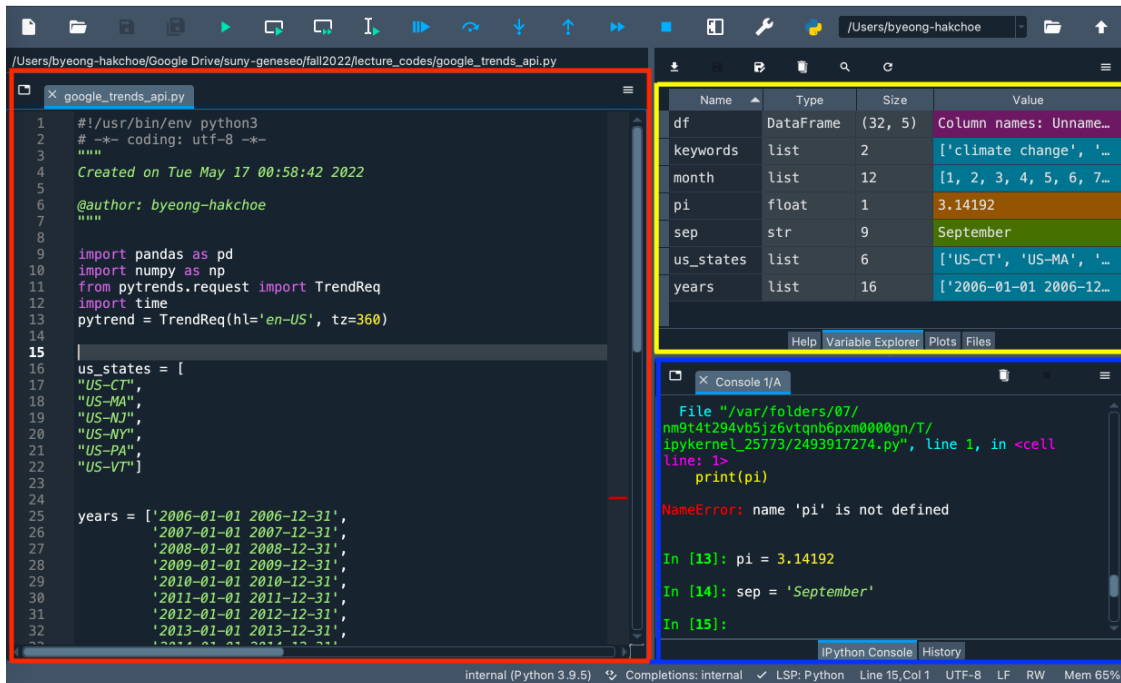
Below the Variable Explorer is the **IPython Console**, which shows the execution history of the script. It displays the following output:

```
File "/var/folders/07/nm9t4t294vb5jz6vtqnb6pxm0000gn/T/ipykernel_25773/2493917274.py", line 1, in <cell>
line: 1>
    print(pi)
NameError: name 'pi' is not defined

In [13]: pi = 3.14192
In [14]: sep = 'September'
In [15]:
```

- From **Script Editor** (red box), we can create, open and edit files.

Console Pane



The screenshot displays a Jupyter Notebook environment with three main components:

- Code Editor (Left):** Contains a Python script named `google_trends_api.py`. The script includes imports for `pandas`, `numpy`, `pytrends`, and `time`, and defines variables for US states and years.
- Variable Explorer (Top Right):** A table showing the current state of variables in the notebook.
- Console Pane (Bottom Right):** A blue box showing the output of the code execution, including a `NameError` and the values of `pi` and `sep`.

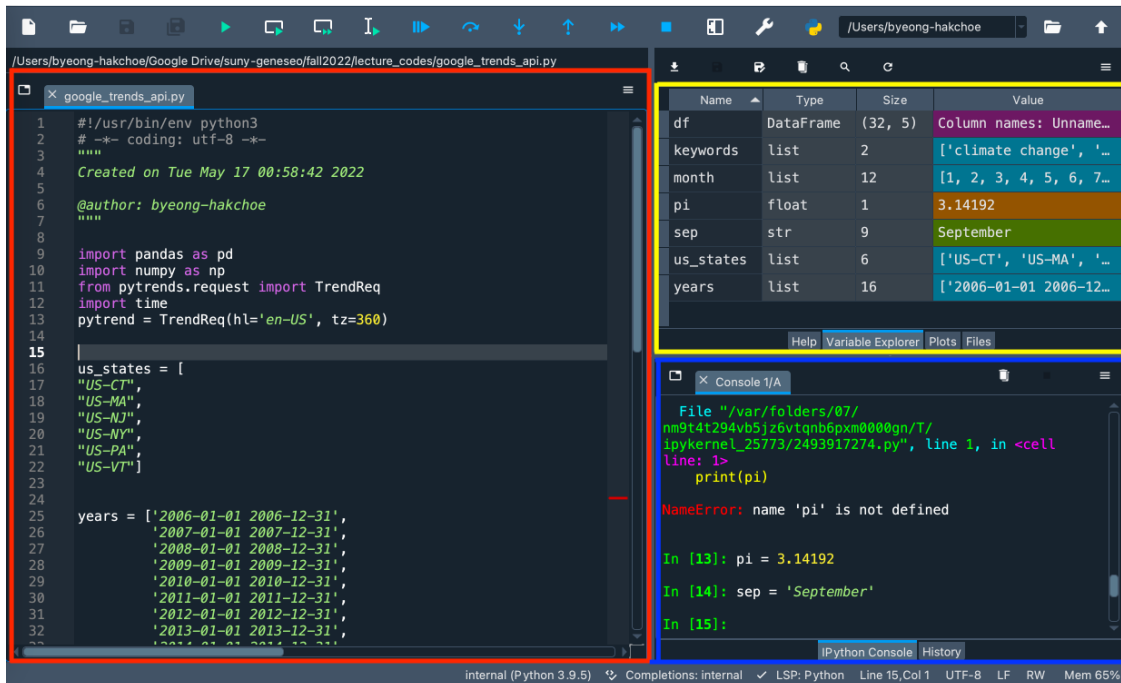
Name	Type	Size	Value
df	DataFrame	(32, 5)	Column names: Unname...
keywords	list	2	['climate change', '...
month	list	12	[1, 2, 3, 4, 5, 6, 7...
pi	float	1	3.14192
sep	str	9	September
us_states	list	6	['US-CT', 'US-MA', '...
years	list	16	['2006-01-01 2006-12...

```
File "/var/folders/07/nm9t4t294vb5jz6vtqnb6pxm0000gn/T/ipykernel_25773/2493917274.py", line 1, in <cell>
line: 1>
    print(pi)
NameError: name 'pi' is not defined

In [13]: pi = 3.14192
In [14]: sep = 'September'
In [15]:
```

- From **Console Pane** (blue box), we can interact directly with the Python interpreter, and type commands where Python will immediately execute them.

Variable Explorer



The screenshot displays a Jupyter Notebook environment with three main components:

- Code Editor:** A Python script named `google_trends_api.py` is shown. It includes imports for `pandas`, `numpy`, `pytrends.request`, and `TrendReq`. The script defines `us_states` as a list of US state abbreviations and `years` as a list of date ranges from 2006 to 2013.
- Variable Explorer (Yellow Box):** This panel shows the current state of the notebook's memory. It contains a table with the following data:

Name	Type	Size	Value
df	DataFrame	(32, 5)	Column names: Unname...
keywords	list	2	['climate change', '...
month	list	12	[1, 2, 3, 4, 5, 6, 7...
pi	float	1	3.14192
sep	str	9	September
us_states	list	6	['US-CT', 'US-MA', '...
years	list	16	['2006-01-01 2006-12...
- IPython Console:** This panel shows the execution history. It displays a `NameError` for `print(pi)` because `pi` was not defined at that point. Subsequent lines show the successful assignment of `pi = 3.14192` and `sep = 'September'`.

- From **Variable Explorer** (yellow box), we can see the values of variables, data frames, and other objects that are currently stored in memory.



Data Containers in Variable Explorer

The screenshot shows the Spyder IDE interface. The main editor displays a Python script with the following code:

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Tue Jan 24 10:10:28 2023
5
6 @author: byeong-hakchoe
7 """
8
9 3
10 2
11 "dan!"
12
13 a = 10
14
15
16 # Here we assign the integer value 5 to the variable x.
17 x = 5
18 # Now we can use the variable x in the next line.
19 y = x + 12
20 y
21
22
23
24
25
26 list_example = [10, 1.23, "like this", True, None]
27 print(list_example)
28 type(list_example)
29
30
```

The Variable Explorer on the right shows the following variables:

Name	Type	Size	Value
a	int	1	7
list_example	list	5	[10, 1.23, ...]
x	int	1	5
y	int	1	17







The IPython Console at the bottom shows the following output:

```
...: y = x + 12
...: y
...: a = 7
...: list_example = [10, 1.23, "like
this", True, None]
...: print(list_example)
...: type(list_example)
[10, 1.23, 'like this', True, None]
Out[1]: list
In [2]:
```

- If we double-click the objects such as `list` and `DataFrame` objects, we can see what data are contained in such objects.



Keyboard Shortcuts

- General shortcuts
 - **Undo:** Ctrl + z (command + z for Mac users)
 - **Redo:** Ctrl + Shift + z (command + shift + z for Mac users)
 - **Selection:** Ctrl + Shift + Arrow (   )
 - **Page Up/Down:** Fn +  / 
- Default shortcuts
 - **Comment (#):** Ctrl + 1 (command + 1 for Mac users)
 - **Block-comment:** Ctrl + 4 (command + 4 for Mac users)
 - **Run selection (or a current line):** F9
 - **Run cell:** Ctrl + Enter (**#** **%%** defines a **cell**)

Comments, Code Cells, and Keyboard Shortcuts

```
1 # %%  
2 # =====  
3 # SECTION TITLE  
4 # =====  
5 a = 1
```

- The **#** mark is Spyder's **comment** character.
- It is recommended to use a **coding block** (defined by **# %%**) with **block commenting** (Ctrl/command + 4) for separating code sections.
- To set your keyboard shortcuts,
 - **Preferences > Keyboard Shortcuts > Search “run” and/or “comment”**
 - Set the shortcuts for (1) run selection; (2) run cell; (3) toggle comment; and (4) blockcomment
 - I use **command + return** for **running a current line (selection)**

Scrapping web tables with `pd.read_html()`

Scrapping Tables with `pd.read_html()`

- Let's scrap the two tables in the following webpage:
 - **National Park Visitation Sets New Record as Economic Engines**

```
1 import pandas as pd
2
3 url = "https://www.nps.gov/orgs/1207/national-park-visitation-sets-new-record-as-economic-engines"
4 tables = pd.read_html(url)
5 len(tables)
6 df_0 = tables[0]
```

- `read_html()` read HTML tables into a **list** of `DataFrame` objects.

📌 Setting Column Names

- How can we set the **first row** of a DataFrame as its **column names**?
- How can we **remove** the first row ?

```
1 df_0 = tables[0]
2 df_0.columns = df_0.iloc[0] # Set the first row as column names
3 df_0 = df_0.iloc[1:] # Keeps rows from position 1 onward
```

✅ What is `DataFrame.iloc[]`?

- `DataFrame.iloc[...]` is **integer-location indexing**:
 - It selects **rows by position** (0, 1, 2, ...), not by index labels.
 - **Slicing works with `DataFrame.iloc[]`**
- `df_0.iloc[0]` returns the **first row** (position 0) as a **Series**.

Dot Operators, Methods, and Attributes

Dot operator

- The dot operator (`DataFrame.`) is used for an **attribute** or a **method** on objects.

Method

- A method (`DataFrame.METHOD()`) is a **function** that we can call on a `DataFrame` to perform operations, modify data, or derive insights.
 - e.g., `df.info()`

Attribute

- An attribute (`DataFrame.ATTRIBUTE`) is a **property** that provides information about the `DataFrame`'s structure or content without modifying it.
 - e.g., `df.columns`



Getting a Summary of a DataFrame

```
1 df_0.info()      # method
2 df_0.count()     # method
```

```
1 df_0.shape       # attribute
2 df_0.columns     # attribute
```

- Every **DataFrame** object has a **.info()** method that provides a summary of a DataFrame:
 - Variable names (**.columns**)
 - Number of observations and variables (**.shape**)
 - Number of non-missing values in each variable (**.count()**)
 - ▶ Pandas often displays missing values as **NaN**.

Absolute Pathnames

- An **absolute pathname** tells the computer the *exact location* of a file, starting from the very top folder of your computer.
 - This location never changes, no matter where you are working in Python.
- In Python, you can see the **working directory** — the folder where Python is currently “looking” for files — by running `os.getcwd()` in the **Console**.
- Examples of an absolute pathname for `custdata_rev.csv`:
 - On a Mac:
`/Users/user/documents/data/custdata_rev.csv`
 - On Windows:
`C:\\Users\\user\\Documents\\data\\custdata_rev.csv`
 - ▶ Note: In Windows, we use **double backslashes** (`\\`) because a single backslash (`\`) is treated as a special character in Python.

Relative Pathnames

- A **relative pathname** specifies the location of a file *relative to the working directory*.
- **Examples of a relative pathname for `custdata_rev.csv`:**
 - Absolute pathname:
`/Users/user/documents/data/custdata_rev.csv`
 - Working directory:
`/Users/user/documents/`
 - Relative pathname:
`data/custdata_rev.csv`



Finding the Absolute Path of a File/Folder

Windows 11

- **Step 1:** Navigate to your folder using File Explorer.
- **Step 2:** Right-click the desired file or folder.
- **Step 3:** Click **Copy as path**.
- **Step 4:** Paste the path into your Python script (**Ctrl + V**).
- **Step 5:** Adjust backslashes in the path:
 - **Option 1:** Replace backslashes (\) with forward slashes (/).
 - **Option 2:** Replace single backslashes (\) with double backslashes (\\).

Mac

- **Step 1:** Navigate to your folder using Finder.
- **Step 2:** Select the file or folder by clicking on it.
- **Step 3:** Copy the path (**Option + Command + C**).
- **Step 4:** Paste the path into your Python script (**Command + V**).



CSV Files

- A **CSV** (comma-separated values) file is a plain text file where each value is separated by a *comma*.
 - CSV files are widely used for storing data from spreadsheets and databases.
- **Example**
 - <https://bcdanl.github.io/data/tvshows.csv>

Exporting a DataFrame as a CSV File with `to_csv()`

- To export `DataFrame` as a **CSV** file, we use the `to_csv()` method.
 - Before exporting, you can set the **working directory (WD)** to organize and manage the location of CSV files.
 - Create a `data` directory within your **WD**. This helps in keeping your data analysis and exports well-organized.

```
1 # Import the os module to interact with the operating system
2 import os
3
4 # Set the working directory path
5 wd_path = 'ABSOLUTE_PATHNAME_OF_YOUR_WORKING_DIRECTORY' # e.g., '/Users/bchoe/Docum
6 os.chdir(wd_path) # Change the current working directory to wd_path
7 os.getcwd() # Retrieve and return the current working directory
8
9 # index=False to not write the row index in the CSV output
10 df_0.to_csv('data/table.csv', index =False)
```

Scrapping Tables with `pd.read_html()`

Let's do **Classwork 3**!