Python and Jupyter Lab

EN5422/EV4238 | Fall 2023
w01_PythonIntro.pdf
(Week 1 – 2/2)

Contents

1 Technical Requirements

- Mainly, our learning activities, homework assignments, and exercises will be conducted using the Google Colab platform.
- Alternatively, you have the flexibility to choose other cloud platforms like Deepnote. Additionally, you can install Jupyter Lab via Anaconda on your personal device.
- **Advantages of Google Colab:**
  - No manual package updates required.
  - Utilization of available free storage space.
  - Access to limited, free GPU resources for specific tasks.
- **Advantages of Installing Jupyter Lab:**
  - Opportunity to leverage more powerful GPUs and increased DRAM on your personal computer.
  - Enhanced data processing capabilities.
  - No reliance on an internet connection.
  - Responsibility for package installations and updates falls on you.
- For personalized assistance in setting up Jupyter Lab on your laptop, you're welcome to visit my office (Room 310, S6 Building). Please schedule an appointment in advance by contacting me via email at hyunglokkim@gist.ac.kr.

2 Introduction to Python with Jupyter Lab

2.1 Getting Help

- A good source of basic data analysis using Python is found in the free book Python Data Science Handbook.
- Web search, especially *stackoverflow.com* and *stats.stackexchange.com*
- Troubleshooting/Debugging.
  - Check one line of code at a time.
  - Run one code block at a time.
  - Use scripts.

2.2 Jupyter Lab

- To begin, log in to your into Google Colab account and select "New notebook" to initiate a new notebook.
- Utilize Google Drive as your storage solution for saving all *.ipynb files generated.
- After creating a new notebook, a designated "Colab Notebooks" folder will automatically be generated within your Google Drive account.
- Access your Colab Notebooks folder through this link: https://drive.google.com/drive/my-drive.
- If you are trying to install Jupyter Lab on your personal device, please let me know.

2.3 Using Python Packages

It takes two steps to use the functions and data in a Python package.

1. **Install the Package:**
    a. Download the package to your computer. This action is a one-time requirement.
    b. This step needs to be performed only once.
    c. There are two representative methods to install packages.
        i. Use the command:

```
!pip install package_name
```

        ii. For Anaconda environment:

```
!conda install -y package_name
```

2. **Load the Package:**
    a. Instruct Python to locate the package functions and/or data.
    b. This step is necessary every time you start Python and plan to use the package.
    c. Make sure to perform this step whenever you wish to access the package's functionality or data.
    d. Examples:

```
# Importing numpy
import numpy as np

# Importing the plot function from matplotlib
import matplotlib.pyplot as plt
```

2.3.1 Note on Python packages

- **matplotlib**: for creating graphics and visualizations.
- **pandas**: for data manipulation and analysis.
- **numpy**: for numerical computing.
- **seaborn**: for statistical data visualization.
- **scipy**: for scientific and technical computing.
- **statsmodels**: for statCal modeling.
- **plotly**: for interactive visualizations.
- **re**: for regular expressions and string manipulation.
- **category_encoders**: for categorical data encoding.

2.4 Homework submission

- Share a link of the *.ipynb file in Google Colab.
- A homework template will be provided for each homework.

2.5 Graphics with the matplotlib library

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations.

- See https://matplotlib.org/
- Keep the matplotlib cheatsheet handy

2.5.1 Graphics with the seaborn library

Seaborn provides a high-level interface for drawing attractive and informative statistical graphics.

- See https://seaborn.pydata.org/
- Keep the seaborn cheatsheet handy

2.6 Data Transformation with the pandas library

- See https://pandas.pydata.org/
- Keep the pandas cheatsheet handy

2.6.1 Single table functions

1. **query()**: filter or find certain rows
    - Can use boolean indexing for similar functionality
    - `iloc[]` for indexing by row number
2. **sort_values()**: reorder rows
    - Use the `ascending` parameter to toggle between ascending and descending order
3. **loc[] or iloc[]**: find/keep certain columns
    - Can also use dataframe filtering with column names
4. **assign()**: add/create new variables
    - You can also use simple assignment, e.g., `df['new_col'] = ...`
5. **describe() or agg()**: produce summary statistics
    - Use `groupby()` before `agg()` for grouped summaries
    - Don't confuse with `describe()` which provides basic statistics

2.6.2 Method Chaining

- Multiple operations can be chained together using the dot notation. This lets you focus on the operations or actions you are performing sequentially.
- It's a way to increase the readability and flow of data transformations.

Example using pandas:

```
import pandas as pd
import numpy as np
s = pd.Series([1, 2, 3, 4, 5, np.nan])
```

```
s.mean(skipna=True)
```

In this example, `skipna=True` ensures that the NaN value is ignored during the mean calculation.

Your Turn #1

1. Install nycflights13 package.

2. Load the nycflights13 package, which contains airline on-time data for all flights departing NYC in 2013. Also includes useful 'metadata' on airlines, airports, weather, and planes.

3. Load the pandas library.

4. Using the flights data,
   - Find all flights that were less than 1000 miles (distance)
   - Keep only the columns: dep_delay, arr_delay, origin, dest, air_time, and distance
   - Add the Z-score for departure delays
   - Convert the departure and arrival delays into hours
   - Calculate the average flight speed (in mph)
   - Order by average flight speed (fastest to slowest)
   - Return the first 12 rows

2.6.3 Other useful pandas functions

- drop_duplicates(): Retain unique/distinct rows.
- sample(): Select random rows.
- nsmallest() and nlargest(): Select rows with smallest and largest values respectively.
- assign(): Add new columns.
- Using fillna(): Replace missing values (NaN) with a specific value.

```
import pandas as pd
import numpy as np

# Sample data
x = pd.Series([1, 2, np.nan, 5, 5, np.nan])

# Replace NaN with 0
x.fillna(0, inplace=True)
print(x)
```

2.7 Groupwise operations

2.7.1 Split – Apply – Combine

The pandas operations become more powerful when used with grouping. The Split-Apply-Combine strategy involves:

- **Splitting** the data into groups based on certain criteria.
- **Applying** a function to each group.
- **Combining** the results into a data structure.

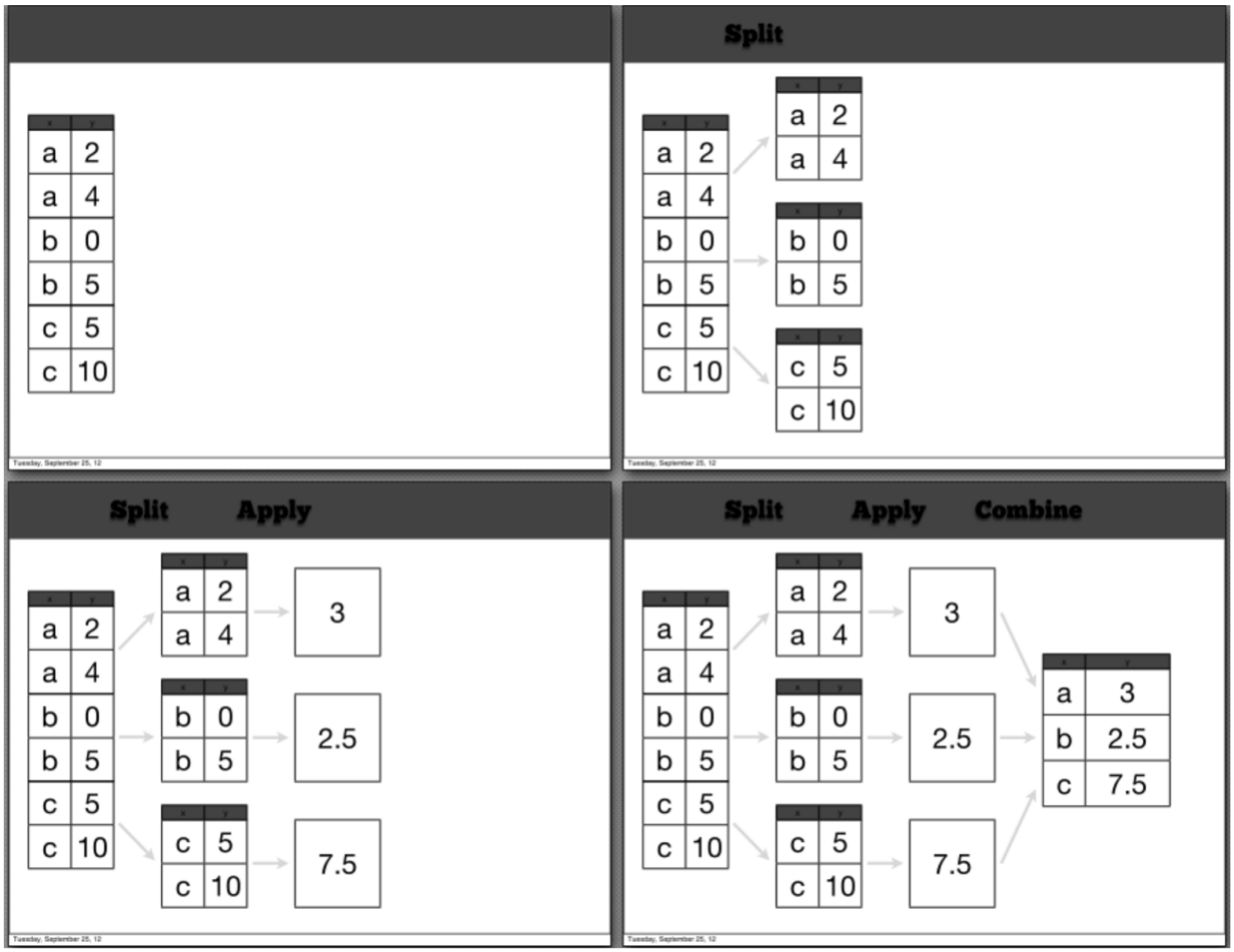

Image from Hadley Wickham UseR tutorial June 2014

Example:

```
import pandas as pd

# Sample dataset
data = {
    'location': ['Beach', 'Forest', 'Urban Park', 'Desert', 'Beach',
'Forest', 'Urban Park', 'Desert'],
    'parameter': ['Air Quality', 'Air Quality', 'Water Quality', 'Soil pH',
'Temperature', 'Temperature', 'Soil pH', 'Water Quality'],
```

```
    'date': ['2023-01-01', '2023-01-02', '2023-01-03', '2023-01-04', '2023-
01-01', '2023-01-02', '2023-01-03', '2023-01-04'],
    'value': [50, 45, 90, 7, 25, 20, 6.5, 85]
}
environmental_data = pd.DataFrame(data)

# Split: Group by 'location' and 'parameter'
grouped = environmental_data.groupby(['location', 'parameter'])

# Apply: Calculate average value for each group
average_values = grouped['value'].mean().reset_index()

# Rename columns for clarity
average_values.columns = ['location', 'parameter', 'average_value']

# Combine: In this case, the 'average_values' DataFrame is already a
combined result
print(average_values)
```

2.8 Data Importing

2.8.1 Data Importing

- Refer to the official pandas documentation.
- It's recommended to keep the pandas documentation handy as a reference.

Example:

```
# Load data directly from a website

import pandas as pd

# 1. Specify path to URL
data_dir = 'https://raw.githubusercontent.com/Hyunglok-
Kim/EN5422_EV4238/main/'
url = data_dir + 'w01_environmental_data.csv' # 10 hydrometeorological data
set

# 2. Load directly from the web
variables = pd.read_csv(url)

# 2.1. Check the dataframe
variables
```

```
# Download data first, then load into Python

import urllib.request

# 1. Specify path to URL
data_dir = 'https://raw.githubusercontent.com/Hyunglok-
Kim/EN5422_EV4238/main/'
url = data_dir + 'w01_environmental_data.csv' # 10 hydrometeorological data
set

# 2. Download the file
```

```
save_path = "w01_environmental_data"  # can be a relative path!
urllib.request.urlretrieve(url, save_path)

# 3. Load data from the local drive
variables = pd.read_csv(save_path)
# 3.1. Check the dataframe
```

2.9 Iteration

We will make good use of iteration in course. Be sure to review freeCodeCamp example more details.
Suppose we want to compare the performance of two models over multiple subsets of a data set.

```
import numpy as np
import pandas as pd

# Simulate fake data
np.random.seed(2022)
data = pd.DataFrame({'x': np.random.rand(100), 'y': np.random.randn(100)})

# Fake model output
def model_1(x):
    return np.random.normal(loc=-0.5, scale=1, size=len(x))

def model_2(x):
    return np.random.normal(loc=0.5, scale=2, size=len(x))

# Using a for loop to simulate and assess performance in each subset
n_subsets = 5
np.random.seed(876)
output = []
for i in range(n_subsets):
    # Sub-sample data (25 samples)
    data_sub = data.sample(25)

    # Get output from the models
    yhat_1 = model_1(data_sub['x'])
    yhat_2 = model_2(data_sub['x'])

    # Score models (using MSE)
    perf_1 = np.mean((yhat_1 - data_sub['y'])**2)
    perf_2 = np.mean((yhat_2 - data_sub['y'])**2)

    # Save results
    output.append({'perf_1': perf_1, 'perf_2': perf_2, 'iter': i + 1})

# Convert to DataFrame and summarize
result_df = pd.DataFrame(output)
avg_diff = result_df['perf_1'].mean() - result_df['perf_2'].mean()
summary = pd.DataFrame({'avg.diff': [avg_diff], 'n': [len(result_df)]})

print(summary)
```

Notice that every `for` loop in Python typically has three components:

1. **Initializing the output structure to store results.**
    - In the given example, we initialized the output as an empty list. This list will store results during each iteration.
2. **The sequence to iterate over.**
    - In this example, we iterated over the range of `n_subsets` using the `range(n_subsets)` function. In Python, it's common to use `range` for iteration sequences.
3. **The body of the loop.**
    - This is where the main operations are conducted during each iteration. In our example, this includes sub-sampling the data, getting model outputs, scoring the models, and appending the results to the output list.

2.10 Vectorization

Python works best (i.e., fastest) when you use vectorized calculations. This means you should avoid loops whenever a vectorized function is available.

```
x = range(1, 101)  # Python uses 0-based indexing, so range(1, 101)
generates numbers from 1 to 100

# Find squared value
x_sq = [i**2 for i in x]  # Using list comprehension

x_sq_slow = []
for i in x:
    x_sq_slow.append(i**2)

# Replace all even values with -1
x_even = [-1 if i % 2 == 0 else i for i in x]
```

Note: For handling more complex conditions, you might want to consider Python's built-in conditional expressions or use numpy's `np.where()` for array-based conditions.

2.11 Iterate over elements of a list or array with Python's map()

In some cases, there might not be a vectorized solution available, and looping becomes necessary. In Python, the native `map()` function provides a flexible way to iterate over elements and apply a function to them.

For every typical loop, there are three main components to consider:

1. Initializing the output structure **to store results.**
2. The sequence to iterate over**.**
3. The body of the loop**.**

In base Python, you'll need to explicitly handle each step. However, when using the `map()` function, you can often encapsulate the initialization and loop sequence, focusing mainly on the body of the loop.

Firstly, define a function that contains everything that the loop body would typically do. For instance, if we wanted to calculate the mean squared error (MSE) for predictive models:

```
import numpy as np
import pandas as pd

def calculate_mse(data_sub, model_1, model_2):
    yhat_1 = model_1(data_sub['x'].values)
    yhat_2 = model_2(data_sub['x'].values)

    perf_1 = np.mean((yhat_1 - data_sub['y'].values) ** 2)
    perf_2 = np.mean((yhat_2 - data_sub['y'].values) ** 2)

    return pd.DataFrame({'perf_1': [perf_1], 'perf_2': [perf_2]})
```

Now, instead of using a traditional `for` loop, you can utilize the `map()` function:

```
results = list(map(lambda _: calculate_mse(data.sample(25), model_1,
model_2), range(5)))
final_result = pd.concat(results).reset_index(drop=True)
```