MFRE Data Analytics Workshop Series

Workshop 4: Python II

Krisha Lim
University of British Columbia | Master of Food and Resource Economics
November 15, 2021

Workshop preparation

☑ Make sure you have shared with me your completed `Workshop3_Student.ipynb` in Google Colab

☑ Review Workshop 3 - Python I notes and codes

☑ Upload and open the `Workshop4_Student.ipynb` file in Google Colab

Overview

Python I

- Google Colab
- Variable Types - String, Integer, Float, Boolean
- Lists

- Dictionaries, Functions

Python II

- Importing packages in Python
- Importing files to Google Colab
- **Working with data and Python packages

Learning Outcomes

- Create and access values in a dictionary
- Formulate a simple function
- Install and load libraries into Colab
- Load data into Colab from a number of sources

Python I

Review - Lists

- Use **square brackets** to define lists
- Start at **index = 0**
- Use `list_name[start_included:end_excluded]` to access lists

```
carbon = [15.3, 16.6, 7.06, 9.14]

# extract second and third values
print(carbon[1:3])
```

```
## [16.6, 7.06]
```

Review - Dictionaries

- Use **curly brackets** to define dictionaries
- Contain *key:value* pairs

```
emissions = {
  'canada': 15.3,
  'usa':16.6,
  'china':7.06,
  'japan':9.14
}

emissions['china']
```

```
## 7.06
```

Dictionaries

To add an item to a dictionary, we assign a value to a new key.

```
emissions['south korea'] = 12.9

print(emissions)
```

```
## {'canada': 15.3, 'usa': 16.6, 'china': 7.06, 'japan': 9.14, 'south korea': 12.9}
```

We can also update the value of an existing key

```
emissions['south korea'] = 15

print(emissions)
```

```
## {'canada': 15.3, 'usa': 16.6, 'china': 7.06, 'japan': 9.14, 'south korea': 15}
```

We use the `del` function to delete a key:value pair.

```
del(emissions['south korea'])

print(emissions)
```

```
## {'canada': 15.3, 'usa': 16.6, 'china': 7.06, 'japan': 9.14}
```

Dictionaries

Dictionaries can contain key:value pairs where the values are also dictionaries, just like lists can also contain lists.

In this example, the keys are still country names, but the values are GDP and carbon emissions.

```
data = {
  'canada': {'gdp':44100,
             'carbon':15.3},
  'usa': {'gdp':55700,
          'carbon':16.6}
}

# To print Canada's emissions
print(data['canada']['carbon'])
```

```
## 15.3
```

Dictionaries

To add China's GDP and emissions to the `data` dictionary we just created

```
# first create the china key:value pair
china_data = {'gdp':16200, 'carbon':7.06}

# then add China to data
data['china'] = china_data
```

Dictionaries

Using `for` loops with dictionaries is a little more complicated.

1) Method 1

```
data = {'canada': {'gdp': 44100, 'carbon': 15.3},
        'usa': {'gdp': 55700, 'carbon': 16.6},
        'china': {'gdp': 16200, 'carbon': 7.06}}

for key, value in data.items():
  print(key, "→", value)
```

```
## canada → {'gdp': 44100, 'carbon': 15.3}
## usa → {'gdp': 55700, 'carbon': 16.6}
## china → {'gdp': 16200, 'carbon': 7.06}
```

Dictionaries

2) Method 2

```
data = {'canada': {'gdp': 44100, 'carbon': 15.3},
        'usa': {'gdp': 55700, 'carbon': 16.6},
        'china': {'gdp': 16200, 'carbon': 7.06}}

for key in data.keys():
  print(key, "→", data[key])
```

```
## canada → {'gdp': 44100, 'carbon': 15.3}
## usa → {'gdp': 55700, 'carbon': 16.6}
## china → {'gdp': 16200, 'carbon': 7.06}
```

Dictionaries

Recall last week we created two lists - `carbon` and `country`.

Can we use convert these two lists to a dataframe? **Yes**

```
country = ["canada", "usa", "china", "japan"]
gdp = [44100, 55700, 16200, 39300]
carbon = [15.3, 16.6, 7.06, 9.14]

dict_loop = {country[i]:carbon[i] for i in range(len(country))}
print(dict_loop)
```

```
## {'canada': 15.3, 'usa': 16.6, 'china': 7.06, 'japan': 9.14}
```

```
dict_zip = dict(zip(country, carbon))
print(dict_zip)
```

```
## {'canada': 15.3, 'usa': 16.6, 'china': 7.06, 'japan': 9.14}
```

Functions

A **function** is a code chunk that runs when called. You can pass inputs called **parameters** or **arguments** into a function and get a result back.

Defining a section of code as a function in Python is done using the `def` keyword.

Functions

In the example below, we have a function called `my_function` that takes one argument `fname`. When `my_function` is called, we pass along a first name, which is used inside the function to print the student name.

```
def my_function(fname):
  print("Hello " + fname + ". Welcome to class today!")

my_function("Krisha")
```

```
## Hello Krisha. Welcome to class today!
```

```
my_function("Janelle")
```

```
## Hello Janelle. Welcome to class today!
```

[1] W3 Schools Tutorial

Functions - give it a try!

How would you modify `my_function` so that it would print the inputted first and last names?

```
def my_function(fname):
  print("Hello " + fname + ". Welcome to class today!")
```

Create a function called `add_function` that takes 2 numbers and returns their sum

Python II

Overview

Python I

- Google Colab
- Variable Types - String, Integer, Float, Boolean
- Lists, Dictionaries, Functions

Python II

- Importing libraries in Python
- Importing files to Google Colab
- **Working with data and Python packages

About Packages

A library in Python contains a set of tools (called functions) that perform tasks on our data.

Importing a library is like getting a piece of lab equipment out of a storage locker and setting it up on the bench for use in a project.

Once a library is set up, it can be used or called to perform the task(s) it was built to do.

About Packages

Modules

- collection of related code to keep codes organized

Packages

- collection of modules
- Examples:
  - `NumPy` - scientific computing
  - `pandas` - data analysis

Libraries

- usually refers to a collection of packages (or related modules and packages)
- Examples:
  - `matplotlib` - data visualization
  - `TensorFlow` - machine learning

[1] Kateryna Koidan, "Difference between Python modules, packages, librariers, and frameworks"

Installing packages

- `pip` is Python's package manager
  - Need to frontend with `!` to execute a shell/cmd command in Colab
- `!pip list` prints out the lists of Python packages that is already in Colab
- `!pip install [package_name]` to install packages / libraries
  - `!pip install pandas` or `!pip install matplotlib`
  - `!pip install pandas matplotlib` to install multiple packages at once

Importing packages

- In Python, we use the `import` function to import packages, modules, and libraries
  - It's like we have to use `library()` to load a package we have already installed using `install.packages()` in R
- Some syntax
  - `import numpy` -> Import the package Numpy and its functions
  - `import numpy as np` -> Shortens the imported name to `np` for better code readability
  - `from numpy import arange` -> Imports the `arange` function from `NumPy`

[1] NumPy: the absolute basics for beginners

Using functions in packages

- Each time we call a function, we use the syntax `LibraryName.FunctionName`.
- Adding the library name with a `.` before the function name tells Python where to find the function.

Importing modules or packages

```
import numpy
a = numpy.arange(6)
a
```

```
## array([0, 1, 2, 3, 4, 5])
```

```
import numpy as np
a = np.arange(6)
a
```

```
## array([0, 1, 2, 3, 4, 5])
```

```
from numpy import arange
a = arange(6)
a
```

```
## array([0, 1, 2, 3, 4, 5])
```

Importing modules or packages

When to use which syntax? *It depends*

- If you will use only one function (e.g., `arange`) from the package multiple times, then use `from numpy import range`.
- If you will use multiple functions from the package, then use `import numpy as np`.
- There is actually another way (`from numpy import *`) but is not advisable when there are overlapping function names from different modules or packages

[1] Read here for more details

Data

- 3 months of historical sales of a supermarket with 3 branches in Myanmar (formerly Burma)
- Downloaded from Kaggle

Data Import - Local Drive

- Say you have the `.csv` or `.xlsx` file stored in your device
- Run this code first

```
# import pandas
import pandas as pd

# initiate file import
from google.colab import files
uploaded = files.upload()
```

- A pop up will appear where you can select the file you want to work with.
- Use `df.read_csv()` to read `.csv` files and `df.read_xlsx()` to read `.xlsx` files as `pandas` dataframe

```
# load file to object called 'supermarket'
supermarket = pd.read_csv("myanmar-supermarket.csv")
supermarket.head(5)
```

Data Import - Google Drive

- Say you have the `.csv` or `.xlsx` file stored your Google Drive
- Run this code to mount your Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

- Click the link, give authorization, and copy the authorization code and paste into the textbox in Colab

```
# load file to object called 'from_drive'
from_drive = pd.read_csv("/content/drive/MyDrive/myanmar-supermarket.csv")
from_drive.head(5)
```

Data Import - Google Drive

- Say you want to import a lot of files from the same directory in Google Drive
- It would be better to specify the working directory to your Google Drive folder
    - We will use a package called `os` which will allow us to get the working directory

```
# import os package
import os

# print the current wd
os.getcwd()

# We want the dir to be "/content/drive/MyDrive"

# if current wd is "/content" then
dir = os.getcwd() + "/drive/MyDrive/"

# if current wd is "/content/drive/"
dir = os.getcwd() + "/MyDrive/"

# load file using `dir` in file path to object called `from_drive_path`
from_drive_path = pd.read_csv(dir + "myanmar-supermarket.csv")
from_drive_path.head(5)
```

Data Import - Package/API

- Recall we used the `{cansim}` package to access Statistics Canada data in R during the R bootcamp and Workshop 2 (Data Visualization)
- The equivalent in Python is `stats_can`

```
# install package
!pip install stats-can

# import stats_can package from StatsCan
from stats_can import StatsCan

# instantiate a StatsCan object
sc = StatsCan()

# read statscan table into a panda dataframe
df = sc.table_to_df("36-10-0222-01")
df.head(5)
```

Case Study

Kim has been asked to use Python to analyze the following dataset.

	Jan		Feb		Mar	
Products	Total Quantity	Quantity Rejected	Total Quantity	Quantity Rejected	Total Quantity	Quantity Rejected
Oranges	3000	1400	6000	3500	12000	6700
Bananas	500	700	3000	1750	15000	3350

Specifically, her tasks are:

- Create one or more Python "dictionaries" for all data in the above table (Hint: You can use nested dictionaries for fruits and quantity/rejected, and use lists to hold the values)
- Print fruit dictionary
- Print the sum of all "Orange" Total Quantity and sum of all "Orange" Quantity Rejected
- Calculate and print the rejection rate
- Calculate and print the maximum of "Orange" Total Quantity (Hint: use `max()` function)

Recap

- Created and accessed values in a dictionary
- Formulated a simple function
- Installed and imported libraries into Colab
- Loaded data into Colab from a number of sources

Over the holidays

- Review Python I and Python II notes - be ready for a quiz in 521D 🤓
- Work through the "Working with Data" section (Exploring our Data and Calculating Summary Statistics) in the Workshop 4 codes
- UBC MDS' Programming in Python for Data Science

References

- Data Analysis and Visualization in Python for Ecologists by Data Carpentry