

Week 6

1 Read .csv files

Sometimes, you may want to pull data from an external source. It would be extremely frustrating to pull each individual value into an array and create a DataFrame manually. Luckily, Python can read excel files and convert them to DataFrames for you.

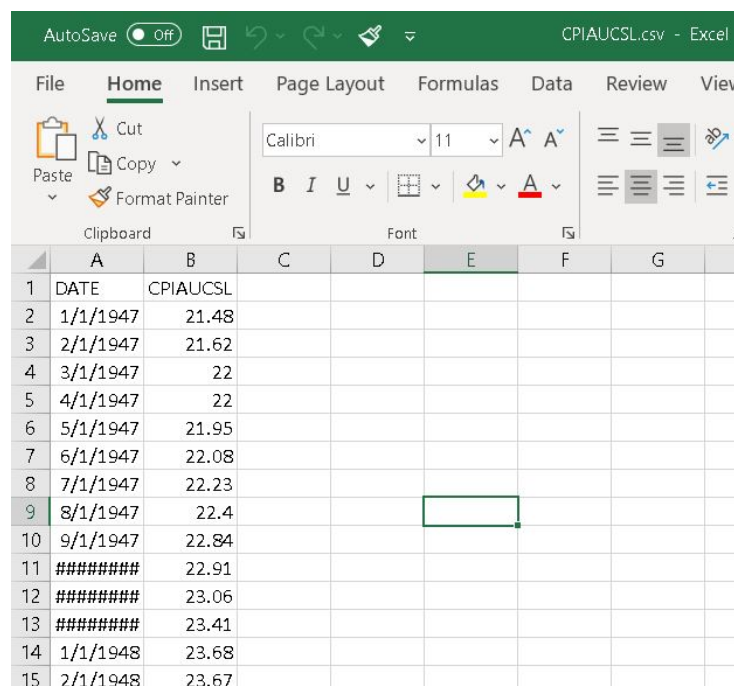
1.1 What is a .csv?

Csv stands for comma separated values. It is a simple file format to store tabular data such as the ones in excel spreadsheets. Csv files are in the same format as dictionaries in Python, and when you import them into your Jupyter Notebook, they will automatically be turned into a dictionary.

1.2 `pd.read_csv()`

Reading a .csv file is easiest in the pandas package. You will use the command `pd.read_csv()` to take a file from your computer and read it into your Jupyter Notebook file.

Say you have a .csv file that looks like this:



	A	B	C	D	E	F	G
1	DATE	CPIAUCSL					
2	1/1/1947	21.48					
3	2/1/1947	21.62					
4	3/1/1947	22					
5	4/1/1947	22					
6	5/1/1947	21.95					
7	6/1/1947	22.08					
8	7/1/1947	22.23					
9	8/1/1947	22.4					
10	9/1/1947	22.84					
11	#####	22.91					
12	#####	23.06					
13	#####	23.41					
14	1/1/1948	23.68					
15	2/1/1948	23.67					

This is an actual .csv file from the FRED database. You can import a .csv file by using the following code:

```
1 data = pd.read_csv(filepath)
```

In this example, I wrote "filepath" inside of the parenthesis. You will not do this when you import a .csv file into your work; you will put the actual file path. Depending on where your file is stored, you will write a different section of code to tell Python where the file is located. We will go over that in sections 1.2.1 and 1.2.3.

For now, we are going to focus on one more thing you should do when you are importing your work. If you take a look at the screenshot of the excel file above, you will see that the column names are in all caps, and one of the names is complicated and hard to remember. When you import a .csv file into your work, Python will automatically convert the .csv to a DataFrame, and it will use the first row of data in your .csv file to name each column in the DataFrame.

In the above example, the .csv file would be converted to a DataFrame with two arrays (columns) named "DATE" and "CPIAUCSL". If you would rather have arrays named something else, then you can do so with a very simple section of code:

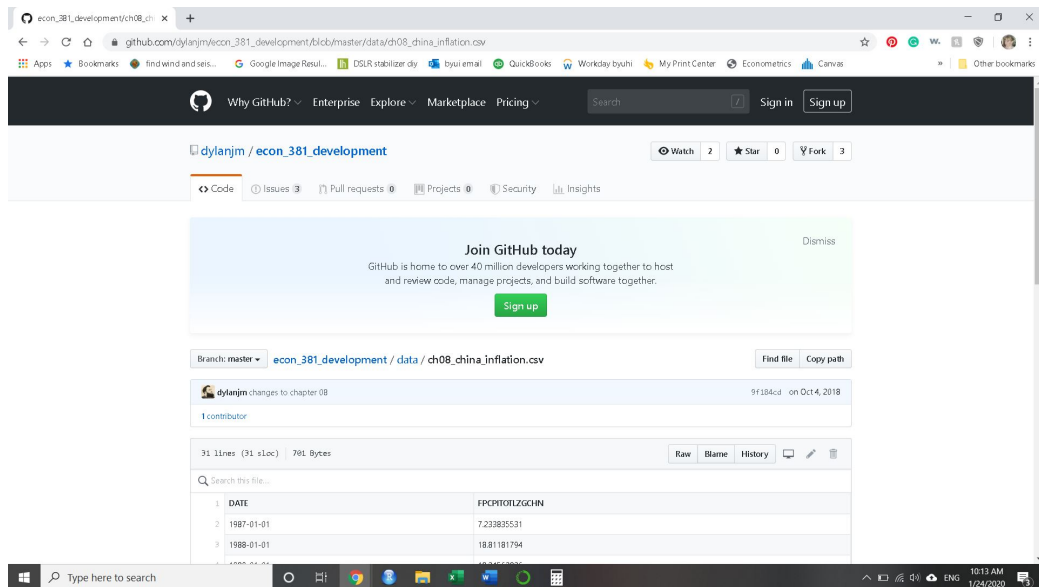
```
1 data = pd.read_csv(filepath, names = ["date", "cpi"])
```

Notice the *names* = command. Using this command, you essentially are telling Python to ignore the first row in your data and name your arrays "date" and "cpi" respectively. You don't need to specify which column gets which name because it will do it in order. Column 1 will always be named whatever name you type first, column 2 will always be named the second name you type, and so forth. You cannot omit columns, so if you have a .csv file with 10 columns, and you want to name the 8th, you will have to name all ten columns or find a different command that allows you to name only one column.

1.2.1 Files on a Remote Server

Now we will go over how to type your file's path in the *read_csv()* command. The easiest file to read into Python is one on a remote server.

To read a file from a remote server, first you need to locate the file on the server and click on the button that says "raw."



This will take you to a version of the file that doesn't have anything except the .csv (no ads, no other words, etc.). Once you are there, all you have to do is put the url of the file in quotes and insert it where your file path should be in the `read_csv()` command. See the example below for more information.

```

1 #Option 1
2 filepath = "https://raw.githubusercontent.com/dylanjm/
   econ_381_development/master/data/
   ch08_italy_inflation.csv"
3 data = pd.read_csv(filepath, names = ["date", "cpi"])
4
5 #Option 2
6 data = pd.read_csv("https://raw.githubusercontent.com/
   dylanjm/econ_381_development/master/data/
   ch08_italy_inflation.csv", names = ["date", "cpi"])

```

1.2.2 Files on Canvas

During the semester, you may be asked to download .csv files from Canvas or directly from your homework assignments. These files are already formatted in a way that is Python-Friendly, so all you have to worry about is uploading it into JupyterLab.

Once you have downloaded the file that you need, save it to a folder on your computer. You can use whatever folder you want.

Now that the folder is in your computer, you need to find the file path. You can do this easily by copying the file path.

For windows:

1. Navigate to the file in the file explorer.
2. Shift + Right click on the file
3. Press "Copy as path."

For Macs:

1. Navigate to the file in the file explorer
2. Press Command + i to summon "Get Info"
3. Click and drag alongside "Where" to select the path, then hit Command+C to copy the full path to the clipboard

Once you have copied the file path, you need to paste it into python. Your file path should look something like this:

```
1 #Option 1
2 filepath = "C:/Users/lizzi/Google Drive/Homework/file .
   csv"
3 data = pd.read_csv(filepath , names = ["date" , "cpi" ])
4
5 #Option 2
6 data = pd.read_csv("C:/Users/lizzi/Google Drive/
   Intermediate Macro/Homework/file.csv" , names = [ "
   date" , "cpi" ] )
```

Notice that all the slashes are forward slashes (/). When you copy your file path, you may see backslashes (\) in your file path. **This will not work.** You need to do something to reverse the backslashes, or you will get an error message. There are a few ways to do this, but the easiest is to put a lower-case "r" in front of your file path:

```
1 #Option 1
2 filepath = r "C:\Users\lizzi\Google Drive\Homework\file
   .csv"
3 data = pd.read_csv(filepath , names = ["date" , "cpi" ])
4
5 #Option 2
6 data = pd.read_csv(r "C:\Users\lizzi\Google Drive\
   Homework\file.csv" , names = ["date" , "cpi" ] )
```

There's also a few ways to shorten your file path. You can use what is called a relative file path to do this. Since relative file paths are difficult to understand by reading, you will need to watch the video linked below.

<https://www.youtube.com/watch?v=fe6GA200dks>

Note: *This video is for R, and it will talk about a working directory and how to set your working directory. In JupyterLab, the working directory is always automatically set to whatever folder your JupyterNotebook file is in.*

1.2.3 Files on FRED

You can take files on FRED and import them into your work. Doing this is quite simple, but if you are importing multiple files to be put in the same graph or DataFrame, you need to be a little careful.

You need to make sure that all files are using the same date intervals (years, quarters, months, etc.). If one of your files has data that was collected every year, and the other has data collected every quarter, you are going to have one file with many more data points than the other.

Luckily, fixing this problem is simple. Navigate to the graph that you want to download. Before you press download, press "Edit Graph." There should be a drop-down button called "Frequency." Change the frequency to whatever frequency you want (I usually recommend the highest frequency that all the data sets have in common- usually quarters). Then press download \gg CSV.

After you have done this, you can read the .csv file into your JupyterLab like normal. Follow instructions in section 1.2.2 to do this.

1.3 Datetime and .csv files

One of the trickiest parts of importing historical data from a csv is getting Python to format the dates correctly. At this point in the course, you might have noticed that if you don't format the dates properly, it creates many issues with every aspect of your code. Your plots will look funny, your DataFrames and arrays won't work, and you'll probably get lots of errors every step of the way.

Luckily, there is a way to avoid all these problems 99.9% of the time. This method may not work 100% of the time since Python is an open source program, but for all of the assignments that you have in this class, it will work. Here is the code:

```

1 #Import the file per above instructions
2 myfile = read.csv(filepath, names = ["Date", "GDP"])
3
4 #create an array called date
5 Date = np.array(myfile["Date"], dtype = np.datetime64)
6
7 #always reference array instead of csv
8 print(Date)
9
10 #DO NOT do this
11 print(myfile["Date"])

```

If that code doesn't work, then you can do this:

```

1 #Import the file per above instructions
2 myfile = read.csv(filepath, names = ["Date", "GDP"])
3
4 #convert column to datetime format using pd.to_datetime
5 myfile["Date"] = pd.to_datetime(myfile["Date"])
6
7 #always reference the column when using this method
8 print(myfile["Date"])
9
10 #DO NOT do this
11 print(Date)

```