(http://www.pieriandata.com)

Inputs and Outputs

NOTE: Typically we will just be either reading csv files directly or using pandas-datareader to pull data from the web. Consider this lecture just a quick overview of what is possible with pandas (we won't be working with SQL or Excel files in this course)

Data Input and Output

This notebook is the reference code for getting input and output, pandas can read a variety of file types using its pd.read_ methods. Let's take a look at the most common data types:

In [1]:
```
import numpy as np
import pandas as pd
```

Check out the references here!

This is the best online resource for how to read/write to a variety of data sources!

https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html
(https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html)

Format Type	Data Description	Reader	Writer
text	CSV (https://en.wikipedia.org/wiki/Comma-separated_values)	read_csv	to_csv
text	JSON (https://www.json.org/)	read_json	to_json
text	HTML (https://en.wikipedia.org/wiki/HTML)	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel (https://en.wikipedia.org/wiki/Microsoft_Excel)	read_excel	to_excel

Format Type	Data Description	Reader	Writer
binary	OpenDocument (http://www.opendocumentformat.org)	read_excel	
binary	HDF5 Format (https://support.hdfgroup.org/HDF5/whatishdf5.html)	read_hdf	to_hdf
binary	Feather Format (https://github.com/wesm/feather)	read_feather	to_feather
binary	Parquet Format (https://parquet.apache.org/)	read_parquet	to_parquet
binary	Msgpack (https://msgpack.org/index.html)	read_msgpack	to_msgpack
binary	Stata (https://en.wikipedia.org/wiki/Stata)	read_stata	to_stata
binary	SAS (https://en.wikipedia.org/wiki/SAS_(software))	read_sas	
binary	Python Pickle Format (https://docs.python.org/3/library/pickle.html)	read_pickle	to_pickle
SQL	SQL (https://en.wikipedia.org/wiki/SQL)	read_sql	to_sql

Reading in a CSV

Comma Separated Values files are text files that use commas as field delimeters.
Unless you're running the virtual environment included with the course, you may need to install `xlrd` and `openpyxl`.
In your terminal/command prompt run:

```
conda install xlrd
conda install openpyxl
```

Then restart Jupyter Notebook. (or use pip install if you aren't using the Anaconda Distribution)

Understanding File Paths

You have two options when reading a file with pandas:

1. If your .py file or .ipynb notebook is located in the **exact** same folder location as the .csv file you want to read, simply pass in the file name as a string, for example:

```
df = pd.read_csv('some_file.csv')
```

2. Pass in the entire file path if you are located in a different directory. The file path must be 100% correct in order for this to work. For example:

```
df = pd.read_csv("C:\\Users\\myself\\files\\some_file.csv")
```

Print your current directory file path with pwd

In [53]:
```
pwd
```

Out[53]:
```
'C:\\Users\\Marcial\\Pierian-Data-Courses\\Machine-Learning-MasterClass\\03-Pandas'
```

List the files in your current directory with ls

In [54]:
```
ls
```

```
 Volume in drive C has no label.
 Volume Serial Number is 3652-BD2F

 Directory of C:\Users\Marcial\Pierian-Data-Courses\Machine-Learning-MasterClass\03-Pandas

07/04/2020  06:10 PM    <DIR>          .
07/04/2020  06:10 PM    <DIR>          ..
07/02/2020  05:40 PM    <DIR>          .ipynb_checkpoints
06/30/2020  04:51 PM           565,390 00-Series.ipynb
07/01/2020  12:48 PM           208,957 01-DataFrames.ipynb
07/01/2020  12:48 PM           194,591 02-Conditional-Filtering.ipynb
07/02/2020  07:02 PM           196,047 03-Useful-Methods.ipynb
07/01/2020  03:32 PM            64,227 04-Missing-Data.ipynb
07/04/2020  01:28 PM           219,627 05-Groupby-Operations-and-MultiIndex.ipynb
07/04/2020  03:19 PM            62,966 06-Combining-DataFrames.ipynb
07/02/2020  07:02 PM            29,356 07-Text-Methods.ipynb
07/02/2020  06:38 PM            35,705 08-Time-Methods.ipynb
07/04/2020  06:10 PM            53,007 09-Inputs-and-Outputs.ipynb
```

NOTE! Common confusion point! Take note that all read input methods are called directly from pandas with pd.read_ , all output methods are called directly off the dataframe with df.to_

CSV Input

In [55]:
```
df = pd.read_csv('example.csv')
```

In [56]:
```
df
```

Out[56]:

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

In [57]:
```
df = pd.read_csv('example.csv',index_col=0)
```

In [58]:
```
df
```

Out[58]:

a	b	c	d
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

In [59]:
```
df = pd.read_csv('example.csv')
```

In [60]:
```
df
```

Out[60]:

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

CSV Output

Set index=False if you do not want to save the index , otherwise it will add a new column to the .csv file that includes your index and call it "Unnamed: 0" if your index did not have a name. If you do want to save your index, simply set it to True (the default value).

In [61]:
```
df.to_csv('new_file.csv',index=False)
```

HTML

Pandas can read table tabs off of HTML. This only works if your firewall isn't blocking pandas from accessing the internet!

Unless you're running the virtual environment included with the course, you may need to install `lxml`, `htmllib5`, and `BeautifulSoup4`.
In your terminal/command prompt run:

```
conda install lxml
```

or

```
pip install lxml
```

Then restart Jupyter Notebook (you may need to restart your computer). (or use pip install if you aren't using the Anaconda Distribution)

read_html

HTML Input

Pandas read_html function will read tables off of a webpage and return a list of DataFrame objects. NOTE: This only works with well defined
objects in the html on the page, this can not magically read in tables that are images on a page.

In [62]:
```
tables = pd.read_html('https://en.wikipedia.org/wiki/World_population')
```

In [63]:
```
len(tables) #tables
```

Out[63]:
```
26
```

Not Useful Tables

Pandas found 26 tables on that page. Some are not useful:

In [64]:
```
tables[0]
```

Out[64]:

	0	1
0	NaN	An editor has expressed concern that this arti...

Tables that need formatting

Some will be misaligned, meaning you need to do extra work to fix the columns and rows:

In [65]:
```
tables[1]
```

Out[65]:

World population (millions, UN estimates)[14]

	#	Top ten most populous countries	2000	2015	2030[A]
0	1	China[B]	1270	1376	1416
1	2	India	1053	1311	1528
2	3	United States	283	322	356
3	4	Indonesia	212	258	295
4	5	Pakistan	136	208	245
5	6	Brazil	176	206	228
6	7	Nigeria	123	182	263
7	8	Bangladesh	131	161	186
8	9	Russia	146	146	149
9	10	Mexico	103	127	148
10	NaN	World total	6127	7349	8501
11	Notes: ^ 2030 = Medium variant. ^ China exclud...	Notes: ^ 2030 = Medium variant. ^ China exclud...	Notes: ^ 2030 = Medium variant. ^ China exclud...	Notes: ^ 2030 = Medium variant. ^ China exclud...	Notes: ^ 2030 = Medium variant. ^ China exclud...

In [66]:
```
world_pop = tables[1]
```

In [67]:
```
world_pop.columns
```

Out[67]:
```
MultiIndex([('World population (millions, UN estimates)[14]', ...),
            ('World population (millions, UN estimates)[14]', ...),
            ('World population (millions, UN estimates)[14]', ...),
            ('World population (millions, UN estimates)[14]', ...),
            ('World population (millions, UN estimates)[14]', ...)],
           )
```

In [68]:
```
world_pop = world_pop['World population (millions, UN estimates)[14]'].drop
```

In [69]:
```
world_pop.columns
```

Out[69]:
```
Index(['Top ten most populous countries', '2000', '2015', '2030[A]'], dtyp
e='object')
```

In [70]:
```
world_pop.columns = ['Countries', '2000', '2015', '2030 Est.']
world_pop = world_pop.drop(11,axis=0)
```

In [71]:
```
world_pop
```

Out[71]:

	Countries	2000	2015	2030 Est.
0	China[B]	1270	1376	1416
1	India	1053	1311	1528
2	United States	283	322	356
3	Indonesia	212	258	295
4	Pakistan	136	208	245
5	Brazil	176	206	228
6	Nigeria	123	182	263
7	Bangladesh	131	161	186
8	Russia	146	146	149
9	Mexico	103	127	148
10	World total	6127	7349	8501

Tables that are intact

In [72]:
```
tables[6]
```

Out[72]:

	Rank	Country	Population	Area (km2)	Density (Pop. per km2)
0	1	Singapore	5703600	710	8033
1	2	Bangladesh	168870000	143998	1173
2	3	Lebanon	6855713	10452	656
3	4	Taiwan	23604265	36193	652
4	5	South Korea	51780579	99538	520
5	6	Rwanda	12374397	26338	470
6	7	Haiti	11577779	27065	428
7	8	Netherlands	17480000	41526	421
8	9	Israel	9220000	22072	418
9	10	India	1364080000	3287240	415

Write to html Output

If you are working on a website and want to quickly output the .html file, you can use to_html

In [73]:
```
df.to_html('simple.html',index=False)
```

read_html is not perfect, but its quite powerful for such a simple method call!

Excel Files

Pandas can read in basic excel files (it will get errors if there are macros or extensive formulas relying on outside excel files), in general, pandas can only grab the raw information from an .excel file.

NOTE: Requires the openpyxl and xlrd library! Its provided for you in our environment, or simply install with:

```
pip install openpyxl
pip install xlrd
```

Heavy excel users may want to check out this website: https://www.python-excel.org/ (https://www.python-excel.org/)

You can think of an excel file as a Workbook containin sheets, which for pandas means each sheet can be a DataFrame.

Excel file input with read_excel()

In [74]:
```
df = pd.read_excel('my_excel_file.xlsx',sheet_name='First_Sheet')
```

In [75]:
```
df
```

Out[75]:

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

What if you don't know the sheet name? Or want to run a for loop for certain sheet names? Or want every sheet?

Several ways to do this: https://stackoverflow.com/questions/17977540/pandas-looking-up-the-list-of-sheets-in-an-excel-file (https://stackoverflow.com/questions/17977540/pandas-looking-up-the-list-of-sheets-in-an-excel-file)

In [76]:
```
# Returns a list of sheet_names
pd.ExcelFile('my_excel_file.xlsx').sheet_names
```

Out[76]:
```
['First_Sheet']
```

Grab all sheets

In [77]:
```
excel_sheets = pd.read_excel('my_excel_file.xlsx',sheet_name=None)
```

In [78]:
```
type(excel_sheets)
```

Out[78]:
```
dict
```

In [79]:
```
excel_sheets.keys()
```

Out[79]:
```
dict_keys(['First_Sheet'])
```

In [80]:
```
excel_sheets['First_Sheet']
```

Out[80]:

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

Write to Excel File

In [81]:
```
df.to_excel('example.xlsx',sheet_name='First_Sheet',index=False)
```

SQL Connections

NOTE: Highly recommend you explore specific libraries for your specific SQL Engine. Simple search for your database+python in Google and the top results should hopefully include an API.

- MySQL (https://www.google.com/search?q=mysql+python)
- PostgreSQL (https://www.google.com/search?q=postgresql+python)
- MS SQL Server (https://www.google.com/search?q=MSSQLserver+python)
- Orcale (https://www.google.com/search?q=oracle+python)
- MongoDB (https://www.google.com/search?q=mongodb+python)

Let's review pandas capabilities by using SQLite, which comes built in with Python.

Example SQL Database (temporary in your RAM)

You will need to install sqlalchemy with:

```
pip install sqlalchemy
```

to follow along. To understand how to make a connection to your own database, make sure to review: https://docs.sqlalchemy.org/en/13/core/connections.html (https://docs.sqlalchemy.org/en/13/core/connections.html)

In [82]:
```
from sqlalchemy import create_engine
```

In [83]:
```
temp_db = create_engine('sqlite:///:memory:')
```

Write to Database

In [85]:
```
tables[6]
```

Out[85]:

	Rank	Country	Population	Area (km2)	Density (Pop. per km2)
0	1	Singapore	5703600	710	8033
1	2	Bangladesh	168870000	143998	1173
2	3	Lebanon	6855713	10452	656
3	4	Taiwan	23604265	36193	652
4	5	South Korea	51780579	99538	520
5	6	Rwanda	12374397	26338	470
6	7	Haiti	11577779	27065	428
7	8	Netherlands	17480000	41526	421
8	9	Israel	9220000	22072	418
9	10	India	1364080000	3287240	415

In [86]:
```
pop = tables[6]
```

In [87]:
```
pop.to_sql(name='populations',con=temp_db)
```

Read from SQL Database

In [89]:
```
# Read in an entire table
pd.read_sql(sql='populations',con=temp_db)
```

Out[89]:

	index	Rank	Country	Population	Area (km2)	Density (Pop. per km2)
0	0	1	Singapore	5703600	710	8033
1	1	2	Bangladesh	168870000	143998	1173
2	2	3	Lebanon	6855713	10452	656
3	3	4	Taiwan	23604265	36193	652
4	4	5	South Korea	51780579	99538	520
5	5	6	Rwanda	12374397	26338	470
6	6	7	Haiti	11577779	27065	428
7	7	8	Netherlands	17480000	41526	421
8	8	9	Israel	9220000	22072	418
9	9	10	India	1364080000	3287240	415

In [92]:
```
# Read in with a SQL Query
pd.read_sql_query(sql="SELECT Country FROM populations",con=temp_db)
```

Out[92]:

	Country
0	Singapore
1	Bangladesh
2	Lebanon
3	Taiwan
4	South Korea
5	Rwanda
6	Haiti
7	Netherlands
8	Israel
9	India

It is difficult to generalize pandas and SQL, due to a wide array of issues, including permissions,security, online access, varying SQL engines, etc... Use these ideas as a starting off point, and you will most likely need to do your own research for your own situation.