Pandas Library

Series, DataFrames and reading in flat files

- Pandas is the most popular Python library for data analysis!
- It is for tabular datasets.
- It is built on the Numpy package.
- The convention is to import pandas using the following code: `import pandas as pd`
- Pandas uses two core objects: `DataFrame` and `Series`.
- A `DataFrame` is like a table, a `Series` is like a list.
- DataFrames are really useful for data analysis.

Creating a Pandas Series

A series is like a table with a single column.

Lets create a series with Pandas library. Don't forget to import pandas first!

In [1]:
```
!pip install pandas # to install pandas
```

```
Requirement already satisfied: pandas in c:\programdata\anaconda3\lib\site-packages (2.0.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\programdata\anaconda3\lib\site-packages
(from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\programdata\anaconda3\lib\site-packages (from pand
as) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\programdata\anaconda3\lib\site-packages (from pa
ndas) (2023.3)
Requirement already satisfied: numpy>=1.21.0 in c:\programdata\anaconda3\lib\site-packages (from pan
das) (1.24.3)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-packages (from python-d
ateutil>=2.8.2->pandas) (1.16.0)
```

In [32]:
```
# command to import pandas
import pandas as pd # where alias is pd, which we will use in the rest of the jupyter notebook to re
```

- To create a series use `pd.Series()`

In [33]:
```
# create a Series
mySeries = pd.Series([1, 3, 5, 7, 9, 11, 13, 15, 17])
```

In [34]:
```
print(type(mySeries))
```

```
<class 'pandas.core.series.Series'>
```

In [37]:
```
print(mySeries)
```

```
0     1
1     3
2     5
3     7
4     9
5    11
6    13
7    15
8    17
dtype: int64
```

- A Series is like a single column of a DataFrame.
- Series can have 'index' names but only one 'name'.

Example:

In [40]:
```
mySeries = pd.Series( [10500, 13850, 15000, 23750],
                index = [ "2016", "2017", "2018", "2019" ],
                name = "Sales")
mySeries
```

Out[40]:
```
2016    10500
2017    13850
2018    15000
2019    23750
Name: Sales, dtype: int64
```

Pandas Series and DataFrame

- We can think of a Series as a single column of a DataFrame.
- We think of a DataFrame as a collection of Series glued together in columns.
- Series can have 'index' names but only one 'name'. This 'name' is the column name in the DataFrame.
- DataFrames can contain different data types in the different columns.

Creating a Pandas DataFrame

DataFrames can be created in two ways:

1. From a list of dictionaries (row by row).
2. From a dictionary of lists (column by column).

Each of these methods is outlined in the following cells.

DataFrames can be read in from files using Pandas commands. We will see some of these commands later in the lecture.

Creating a Pandas DataFrame from a **list** of **dictionaries** (row by row)

```
list_of_dicts = [
                { "colname_1": r1c1, "colname_2": r1c2,…, "colname_n": r1cn },
                { "colname_1": r2c1, "colname_2": r2c2,…, "colname_n": r2cn},
                …
                { "colname_1": rmc1, "colname_2": rmc2,…, "colname_n": rmcn}
                ]
```

To create a dataframe use the `pd.DataFrame()`.

When using lists of dictionaries use `row_by_row_df = pd.DataFrame(list_of_dicts)` command creates the DataFrame

In the above code, `ricj` represents the value in row `i`, column `j`.

You can use `.print()` to show the values of a `Dataframe`. But, you can use function `.display()` to print a dataframe with a better style

Example to create a DataFrame using a list of dictionaries:

In [46]:
```
list_countries = [{"country":"Ireland", "capital":"Dublin"},{ "country":"France", "capital":"Paris"}
print(list_countries)
```

```
[{'country': 'Ireland', 'capital': 'Dublin'}, {'country': 'France', 'capital': 'Paris'}, {'country':
'Germany', 'capital': 'Berlin'}, {'country': 'Spain', 'capital': 'Madrid'}]
```

In [47]:
```
countries = pd.DataFrame(list_countries)
display(countries)
```

	country	capital
0	Ireland	Dublin
1	France	Paris
2	Germany	Berlin
3	Spain	Madrid

Creating a Pandas DataFrame from a dictionary of lists (column by column)

```
dict_of_lists = { “colname_1”: [r1c1, r2c1,…, rmc1],
                  “colname_2”: [r1c2, r2c2,…, rmc2],
                 …..“colname_n”: [r1cn, r2cn,…, rmcn] }
```

Use `col_by_col_df = pd.DataFrame(dict_of_lists)` to create a DataFrame using a dictionary of lists.

In the above code, `ricj` represents the value in row `i`, column `j`.

In each dictionary, the key is the column name, and the value is the list of column values.

Example to create a Pandas DataFrame using dictionary of lists

- As we have seen, a Pandas DataFrame can be built using a dictionary whose keys are the column names, and whose values are a list of entries.
- You can use `.print()` to show the values in a `Dataframe`. But, you can use function `.display()` to show a dataframe with a better style

The example below creates a DataFrame with two columns called ‘country’ and ‘capital’, and four rows.

In [48]:
```
# list_countries = [{"country":"Ireland", "capital":"Dublin"},{ "country":"France", "capital":"Paris

dict_countries = {"country": ["Ireland", "France", "Germany", "Spain"], "capital": ["Dublin", "Paris
countries = pd.DataFrame(dict_countries)
display(countries)
```

	country	capital
0	Ireland	Dublin
1	France	Paris
2	Germany	Berlin
3	Spain	Madrid

- We can assign indexes to give the DataFrame row names:

In [49]:
```
countries.index = ["IE", "FR", "GY", "SP"]
display(countries)
```

	country	capital
IE	Ireland	Dublin
FR	France	Paris
GY	Germany	Berlin
SP	Spain	Madrid

- Or they can be given as part of the pd.DataFrame specification:

In [50]:
```
countries = pd.DataFrame(dict_countries, index = ["IE", "FR", "GY", "SP"])
display(countries)
```

	country	capital
IE	Ireland	Dublin
FR	France	Paris
GY	Germany	Berlin
SP	Spain	Madrid

In [51]:
```
print(type(countries))
```

```
<class 'pandas.core.frame.DataFrame'>
```

- Index names are useful for extracting certain rows from the DataFrame.
- Index values do **not** have to be unique.

In [54]:
```
print(countries["country"])
```

```
IE    Ireland
FR     France
GY    Germany
SP      Spain
Name: country, dtype: object
```

In [55]:
```
print(countries.iloc[0])
```

```
country    Ireland
capital     Dublin
Name: IE, dtype: object
```

In [56]:
```
print(countries.loc["IE"])
```

```
country    Ireland
capital     Dublin
Name: IE, dtype: object
```

Reading in datasets as Pandas DataFrames

It is standard and best practice to use Pandas for most types of flat files.

File types include:

- CSV files
- txt files
- Pickled files
- Excel files (.xlsx)
- SAS files
- Stata files
- HDF5 files
- Matlab files
- Relational databases

Reading in .csv files

- `pd.read_csv("path/name.csv")`
- Default of `pd.read_csv` is that there is a header. Use the `header = None` argument to say there is no header.

- Use the `nrows = 10` argument to read the first 10 rows.
- Use the `sep = argument` to specify the character that separates the values.
- `na_values` argument takes a list of strings to recognise as `NaN` eg. `na_values = ["Nothing", "Missing"]` means all "Nothing" or "Missing" values are read as NaNs.

In [25]:
```
# help(pd.read_csv)
```

In [57]:
```
directory = "C:/Users/cepedazk/Jupyter Notebook/Datasets/"
```

In [60]:
```
import pandas as pd
grid_rain = pd.read_csv(directory+"IRL_MON_RR_2011_grid.csv", nrows=10)
display(grid_rain)
```

	east	north	201101	201102	201103	201104	201105	201106	201107	201108	201109	201110	201111	20
0	25000	61000	101.5	156.4	56.9	51.6	107.5	93.4	70.1	80.3	157.8	174.7	245.3	
1	25000	96000	82.7	157.6	52.3	62.8	111.1	90.4	74.0	85.1	169.0	164.8	194.7	
2	26000	96000	82.6	157.8	51.6	61.0	108.0	88.3	69.8	81.2	161.9	159.1	197.9	
3	27000	97000	82.3	157.7	52.2	62.6	113.9	88.6	73.9	86.3	169.7	162.5	193.9	
4	27000	100000	82.5	158.8	53.9	61.9	106.1	86.1	68.0	78.5	157.7	155.8	195.8	
5	31000	101000	82.3	157.8	52.4	62.4	110.3	88.4	66.9	77.7	155.4	151.5	195.8	
6	32000	97000	85.3	148.7	50.4	58.9	106.0	92.0	66.2	81.2	155.3	144.8	189.7	
7	32000	98000	84.5	150.2	50.9	59.0	106.2	90.7	66.1	80.5	154.8	143.8	191.5	
8	32000	99000	83.9	153.4	51.8	61.6	113.6	91.5	70.9	85.6	163.7	150.8	190.9	
9	32000	100000	82.6	155.8	51.3	60.8	110.0	88.8	66.3	79.2	155.0	147.3	196.6	

In [31]:
```
# grid_rain['201101'].float()
```

Reading in .txt files

- `pd.read_csv("path/name.txt", sep = " ")`
- Notice that we add `sep=" "` argument, leaving a blank space between the quotes, to read in a .txt file.

In [63]:
```
pd.read_csv(directory+"Data.txt", sep = " ")
```

Out[63]:

	Patient_ID	Site	Sex	Height	Weight	IQ	Disease_Status	Blood_Pressure	Survey_Response	Weights_After_6_Mon
0	1	1	1	1.78	77.7	85	N	119	1	7
1	2	3	1	1.68	74.9	103	N	131	4	7
2	3	3	1	1.89	82.8	103	D	122	4	8
3	4	1	1	2.02	90.2	103	N	144	2	8
4	5	3	1	1.90	92.0	91	N	132	4	9
5	6	1	1	1.92	81.2	108	N	140	1	7
6	7	3	1	1.71	72.3	101	N	109	4	7
7	8	1	1	1.90	79.4	121	N	157	3	7
8	9	3	1	1.76	81.4	109	N	122	4	8
9	10	1	1	1.99	83.8	111	N	130	2	8
10	11	3	1	1.78	82.2	79	D	148	4	8
11	12	3	1	1.59	69.1	96	N	101	4	7
12	13	1	1	1.71	75.7	118	D	133	1	7
13	14	2	1	1.82	85.2	90	N	153	3	8
14	15	2	1	1.57	70.0	81	N	115	3	7
15	16	3	1	1.80	82.0	107	N	138	5	8
16	17	1	1	1.67	79.6	105	N	141	1	8
17	18	3	1	1.68	65.2	114	N	109	4	6
18	19	1	1	1.66	68.5	105	N	112	1	6
19	20	2	1	1.94	84.6	78	D	139	3	8
20	21	3	1	1.88	83.8	108	N	150	4	8
21	22	3	1	1.67	76.9	67	D	103	4	7
22	23	2	1	1.87	91.3	104	N	177	3	9
23	24	3	1	1.58	72.3	108	N	122	4	7
24	25	3	1	1.67	76.5	82	N	111	4	7
25	26	1	1	1.75	69.8	100	N	101	2	7
26	27	1	1	1.67	72.2	86	D	127	3	7
27	28	2	1	1.94	84.2	94	D	127	3	8
28	29	1	1	1.65	75.7	101	D	133	1	7
29	30	3	1	1.77	78.3	98	D	166	4	8
30	31	3	2	1.57	72.8	82	N	112	4	7
31	32	3	2	1.66	75.1	119	N	145	4	7
32	33	1	2	1.48	65.3	94	N	107	2	6
33	34	1	2	1.46	61.7	93	N	79	1	6
34	35	3	2	1.62	67.0	91	N	92	4	6
35	36	2	2	1.68	70.1	135	N	116	3	7
36	37	3	2	1.85	83.8	85	N	152	4	8
37	38	1	2	1.45	65.7	68	N	102	1	6
38	39	3	2	1.78	78.5	95	N	144	5	7
39	40	1	2	1.68	73.6	108	N	146	2	7
40	41	1	2	1.81	78.3	96	N	126	1	7
41	42	3	2	1.85	82.2	84	N	145	4	8

	Patient_ID	Site	Sex	Height	Weight	IQ	Disease_Status	Blood_Pressure	Survey_Response	Weights_After_6_Mon
42	43	3	2	1.79	68.3	72	N	129	4	6
43	44	3	2	1.50	63.8	122	N	61	4	6
44	45	3	2	1.47	59.0	77	N	107	4	6
45	46	1	2	1.41	58.3	92	N	96	2	5
46	47	3	2	1.66	72.9	82	N	135	5	7
47	48	3	2	1.65	74.9	118	N	129	4	7
48	49	3	2	1.73	64.4	117	N	108	4	6
49	50	2	2	1.62	67.3	99	D	72	3	6
50	51	1	2	1.84	77.7	104	N	141	2	7
51	52	2	2	1.61	73.7	100	N	122	3	7
52	53	2	2	1.35	64.0	94	N	84	3	6
53	54	3	2	1.76	77.3	110	N	123	4	7
54	55	3	2	1.83	79.1	107	D	138	4	7
55	56	2	2	1.52	63.3	125	D	119	3	6
56	57	1	2	1.57	68.7	105	N	113	2	6
57	58	3	2	1.69	68.1	122	D	95	5	6
58	59	1	2	1.80	77.1	94	N	124	1	7
59	60	3	2	1.79	75.3	95	N	144	5	7

Writing Pandas DataFrames to .csv and .txt files

- `new_df.to_csv("path/new_df.csv")` to write the Pandas DataFrame new_df to a csv file called `new_df.csv` in the folder specified in path.
- `new_df.to_csv("path/new_df.txt", sep = " ")` to write the Pandas DataFrame new_df to a txt file called `new_df.txt` in the folder specified in path.

Specify the working directory

- You may not want to keep specifying the path every time you read in data from the same folder.
- By specifying the working directory, you tell Python where to look by default when it is reading in datasets. It uses the os package.

For example:

In [64]:
```
import os

print(os.getcwd()) # shows the current working directory
```

```
C:\Users\cepedazk\Jupyter Notebook
```

In [65]:
```
directory = "C:/Users/cepedazk/Jupyter Notebook/Datasets/"
os.chdir(directory) # change the current working directory
print(os.getcwd())
```

```
C:\Users\cepedazk\Jupyter Notebook\Datasets
```

Examples of reading in files using os.chdir()

In [67]:
```
os.chdir(directory)
mydata_txt = pd.read_csv("Data.txt", sep = " ")
mydata_csv = pd.read_csv("IRL_MON_RR_2011_grid.csv")
```

In [68]:
```
display(mydata_txt)
```

	Patient_ID	Site	Sex	Height	Weight	IQ	Disease_Status	Blood_Pressure	Survey_Response	Weights_After_6_Mon
0	1	1	1	1.78	77.7	85	N	119	1	7
1	2	3	1	1.68	74.9	103	N	131	4	7
2	3	3	1	1.89	82.8	103	D	122	4	8
3	4	1	1	2.02	90.2	103	N	144	2	8
4	5	3	1	1.90	92.0	91	N	132	4	9
5	6	1	1	1.92	81.2	108	N	140	1	7
6	7	3	1	1.71	72.3	101	N	109	4	7
7	8	1	1	1.90	79.4	121	N	157	3	7
8	9	3	1	1.76	81.4	109	N	122	4	8
9	10	1	1	1.99	83.8	111	N	130	2	8
10	11	3	1	1.78	82.2	79	D	148	4	8
11	12	3	1	1.59	69.1	96	N	101	4	7
12	13	1	1	1.71	75.7	118	D	133	1	7
13	14	2	1	1.82	85.2	90	N	153	3	8
14	15	2	1	1.57	70.0	81	N	115	3	7
15	16	3	1	1.80	82.0	107	N	138	5	8
16	17	1	1	1.67	79.6	105	N	141	1	8
17	18	3	1	1.68	65.2	114	N	109	4	6
18	19	1	1	1.66	68.5	105	N	112	1	6
19	20	2	1	1.94	84.6	78	D	139	3	8
20	21	3	1	1.88	83.8	108	N	150	4	8
21	22	3	1	1.67	76.9	67	D	103	4	7
22	23	2	1	1.87	91.3	104	N	177	3	9
23	24	3	1	1.58	72.3	108	N	122	4	7
24	25	3	1	1.67	76.5	82	N	111	4	7
25	26	1	1	1.75	69.8	100	N	101	2	7
26	27	1	1	1.67	72.2	86	D	127	3	7
27	28	2	1	1.94	84.2	94	D	127	3	8
28	29	1	1	1.65	75.7	101	D	133	1	7
29	30	3	1	1.77	78.3	98	D	166	4	8
30	31	3	2	1.57	72.8	82	N	112	4	7
31	32	3	2	1.66	75.1	119	N	145	4	7
32	33	1	2	1.48	65.3	94	N	107	2	6
33	34	1	2	1.46	61.7	93	N	79	1	6
34	35	3	2	1.62	67.0	91	N	92	4	6
35	36	2	2	1.68	70.1	135	N	116	3	7
36	37	3	2	1.85	83.8	85	N	152	4	8
37	38	1	2	1.45	65.7	68	N	102	1	6
38	39	3	2	1.78	78.5	95	N	144	5	7
39	40	1	2	1.68	73.6	108	N	146	2	7
40	41	1	2	1.81	78.3	96	N	126	1	7
41	42	3	2	1.85	82.2	84	N	145	4	8

	Patient_ID	Site	Sex	Height	Weight	IQ	Disease_Status	Blood_Pressure	Survey_Response	Weights_After_6_Mon
42	43	3	2	1.79	68.3	72	N	129	4	6
43	44	3	2	1.50	63.8	122	N	61	4	6
44	45	3	2	1.47	59.0	77	N	107	4	6
45	46	1	2	1.41	58.3	92	N	96	2	5
46	47	3	2	1.66	72.9	82	N	135	5	7
47	48	3	2	1.65	74.9	118	N	129	4	7
48	49	3	2	1.73	64.4	117	N	108	4	6
49	50	2	2	1.62	67.3	99	D	72	3	6
50	51	1	2	1.84	77.7	104	N	141	2	7
51	52	2	2	1.61	73.7	100	N	122	3	7
52	53	2	2	1.35	64.0	94	N	84	3	6
53	54	3	2	1.76	77.3	110	N	123	4	7
54	55	3	2	1.83	79.1	107	D	138	4	7
55	56	2	2	1.52	63.3	125	D	119	3	6
56	57	1	2	1.57	68.7	105	N	113	2	6
57	58	3	2	1.69	68.1	122	D	95	5	6
58	59	1	2	1.80	77.1	94	N	124	1	7
59	60	3	2	1.79	75.3	95	N	144	5	7

In [69]:
```
display(mydata_csv)
```

	east	north	201101	201102	201103	201104	201105	201106	201107	201108	201109	201110	20111
0	25000	61000	101.5	156.4	56.9	51.6	107.5	93.4	70.1	80.3	157.8	174.7	245.
1	25000	96000	82.7	157.6	52.3	62.8	111.1	90.4	74.0	85.1	169.0	164.8	194.
2	26000	96000	82.6	157.8	51.6	61.0	108.0	88.3	69.8	81.2	161.9	159.1	197.
3	27000	97000	82.3	157.7	52.2	62.6	113.9	88.6	73.9	86.3	169.7	162.5	193.
4	27000	100000	82.5	158.8	53.9	61.9	106.1	86.1	68.0	78.5	157.7	155.8	195.
...	...	...	...	...	...	...	...	...	...	...	...	...	...
70187	333000	191000	41.2	111.4	20.4	17.6	39.3	96.8	44.9	28.6	50.3	104.8	89.
70188	333000	192000	39.7	110.3	20.0	17.0	39.0	93.6	43.7	28.2	49.4	103.5	89.
70189	333000	193000	38.1	110.6	19.5	16.0	38.5	88.8	41.2	27.0	47.3	101.2	90.
70190	334000	192000	38.6	105.8	19.1	16.4	33.9	89.1	42.1	26.8	47.2	99.3	85.
70191	340000	405000	59.3	114.8	45.8	20.3	56.1	82.9	68.7	69.3	118.3	226.2	97.

70192 rows × 14 columns

Excel files

- When reading in an Excel file with one sheet, it is usually best to convert it to a .csv file before reading it in to Python.
- To read in an Excel file with multiple sheets (a .xlsx file), use the command `pd.ExcelFile('filename.xlsx')`.
- `ExcelFile()` parses specified sheet(s) into a DataFrame

Example: An Excel file containing sales figures, with each year in a separate sheet.

In [70]:
```
sales = pd.ExcelFile('sales.xlsx')
print(sales.sheet_names)                # Prints sheet names
```

```
['sales']
```

In [71]:
```
df_sales = sales.parse('sales') # Assigns sheet ‘sales’ to df_sales
```

In [72]:
```
display(df_sales)
```

	order_id	order_date	ship_date	ship_mode	customer_name	segment	state	country	market	regi
0	AG-2011-2040	2011-01-01	2011-01-06	Standard Class	Toby Braunhardt	Consumer	Constantine	Algeria	Africa	Afri
1	IN-2011-47883	2011-01-01	2011-01-08	Standard Class	Joseph Holt	Consumer	New South Wales	Australia	APAC	Ocear
2	HU-2011-1220	2011-01-01	2011-01-05	Second Class	Annie Thurman	Consumer	Budapest	Hungary	EMEA	EMI
3	IT-2011-3647632	2011-01-01	2011-01-05	Second Class	Eugene Moren	Home Office	Stockholm	Sweden	EU	Nor
4	CA-2011-1510	2011-01-01	2011-01-08	Standard Class	Joseph Holt	Consumer	New South Wales	Australia	APAC	Ocear
...	...	...	...	...	...	...	...	...	...	
51313	MO-2014-8720	2014-09-20	2014-09-24	Standard Class	Alan Hwang	Consumer	Catalonia	Cameroon	Africa	Afri
51314	IN-2014-75736	2014-09-21	2014-09-27	Standard Class	Seth Vernon	Consumer	New York	Indonesia	APAC	Southea As
51315	CA-2014-121580	2014-09-23	2014-09-27	Standard Class	Hallie Redmond	Home Office	Panama	Egypt	Africa	Afri
51316	IN-2014-65131	2014-09-23	2014-09-29	Standard Class	Trudy Schmidt	Consumer	Goiás	United States	US	Cent
51317	US-2014-139703	2014-09-24	2014-09-28	Standard Class	Jonathan Doherty	Corporate	New South Wales	Turkey	EMEA	EMI

51318 rows × 24 columns

In [73]:
```
df = sales.parse(0) # Assigns first sheet in sales to df using index 0 (first sheet in excel)
```

In [74]:
```
display(df)
```

	order_id	order_date	ship_date	ship_mode	customer_name	segment	state	country	market	regi
0	AG-2011-2040	2011-01-01	2011-01-06	Standard Class	Toby Braunhardt	Consumer	Constantine	Algeria	Africa	Afri
1	IN-2011-47883	2011-01-01	2011-01-08	Standard Class	Joseph Holt	Consumer	New South Wales	Australia	APAC	Ocear
2	HU-2011-1220	2011-01-01	2011-01-05	Second Class	Annie Thurman	Consumer	Budapest	Hungary	EMEA	EMI
3	IT-2011-3647632	2011-01-01	2011-01-05	Second Class	Eugene Moren	Home Office	Stockholm	Sweden	EU	Nor
4	CA-2011-1510	2011-01-01	2011-01-08	Standard Class	Joseph Holt	Consumer	New South Wales	Australia	APAC	Ocear
...	...	...	...	...	...	...	...	...	...	
51313	MO-2014-8720	2014-09-20	2014-09-24	Standard Class	Alan Hwang	Consumer	Catalonia	Cameroon	Africa	Afri
51314	IN-2014-75736	2014-09-21	2014-09-27	Standard Class	Seth Vernon	Consumer	New York	Indonesia	APAC	Southea As
51315	CA-2014-121580	2014-09-23	2014-09-27	Standard Class	Hallie Redmond	Home Office	Panama	Egypt	Africa	Afri
51316	IN-2014-65131	2014-09-23	2014-09-29	Standard Class	Trudy Schmidt	Consumer	Goiás	United States	US	Cent
51317	US-2014-139703	2014-09-24	2014-09-28	Standard Class	Jonathan Doherty	Corporate	New South Wales	Turkey	EMEA	EMI

51318 rows × 24 columns

What do you think the following command does?

```
sales.parse(0, skiprows = [1], usecols = [0], names = [“Shop”])
```

- Note that all arguments are lists.
- Note that the sheet number and usecols arguments are zero-indexed, but skiprows starts from 1.

In [75]:
```
sales.parse(0, skiprows = [1], usecols = [0], names=["Shop"])
```

Out[75]:

	Shop
0	IN-2011-47883
1	HU-2011-1220
2	IT-2011-3647632
3	CA-2011-1510
4	IN-2011-79397
...	...
51312	MO-2014-8720
51313	IN-2014-75736
51314	CA-2014-121580
51315	IN-2014-65131
51316	US-2014-139703

51317 rows × 1 columns

In [76]:
```
help(sales.parse)
```

```
Help on method parse in module pandas.io.excel._base:

parse(sheet_name: 'str | int | list[int] | list[str] | None' = 0, header: 'int | Sequence[int] | None' = 0, names=None, index_col: 'int | Sequence[int] | None' = None, usecols=None, converters=None, true_values: 'Iterable[Hashable] | None' = None, false_values: 'Iterable[Hashable] | None' = None, skiprows: 'Sequence[int] | int | Callable[[int], object] | None' = None, nrows: 'int | None' = None, na_values=None, parse_dates: 'list | dict | bool' = False, date_parser: 'Callable | lib.NoDefault' = <no_default>, date_format: 'str | dict[Hashable, str] | None' = None, thousands: 'str | None' = None, comment: 'str | None' = None, skipfooter: 'int' = 0, dtype_backend: 'DtypeBackend | lib.NoDefault' = <no_default>, **kwds) -> 'DataFrame | dict[str, DataFrame] | dict[int, DataFrame]' method of pandas.io.excel._base.ExcelFile instance
    Parse specified sheet(s) into a DataFrame.

    Equivalent to read_excel(ExcelFile, ...)  See the read_excel
    docstring for more info on accepted parameters.

    Returns
    -------
    DataFrame or dict of DataFrames
        DataFrame from the passed in Excel file.
```

SAS and Stata files

SAS and Stata are two softwares for statistics and data science.

- To import a SAS file:

```
import pandas as pd

from sas7bdat import SAS7BDAT

with SAS7BDAT('grass_experiment.sas7bdat') as file:

    df = file.to_data_frame()
```

- To import a Stata file:

```
import pandas as pd

df = pd.read_stata('grass_experiment.dta')
```

Pickled files

- Pickled files are native to Python. They should only be used when data is being stored and read back in to Python. Why is this useful?
- It is also useful for storing data files that are native to Python such as dictionaries.
- The JSON format makes dictionaries human-readable.
- Use more readable formats if you want to look at the data elsewhere.

```
import pickle

with open('grass_exp.pkl', 'rb') as file:

data = pickle.load(file)
```

'rb' means the file is read-only and binary.

Pandas exercises

1. Create a Pandas DataFrame in Python containing the data in the table below both row by row and column by column:

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.3	3.3	6.0	2.5	virginica
5.8	2.7	5.1	1.9	virginica

1. There are three files on Moodle under today's lecture: a .csv file, a .txt file, and a .xlsx file.

   Save each of these to your computer, and read in to Python.

   Parse the sheets of the Excel file to separate DataFrames.

   Use .shape, .head() and other commands to inspect the DataFrames.