Loading and Saving Data with Pandas

Pandas is able to ingest and egest data from and to multiple data sources. Following are the most frequently used ones.

- CSV
- Excel
- Databases
- JSON

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

1. CSV Files

Read from CSV File

Use `read_csv()` to read a CSV (comma delimited values) or TSV (tab delimited values) files into a dataframe.

- The delimiter is by default comma `,`.
- For TSV files, set delimiter to `\t` instead.

In [2]:
```
1 df = pd.read_csv('data/class1_test1.tsv', delimiter='\t')
2 df.head()
```

Out[2]:

	name	english	maths	science
0	Aaron	70	46	47
1	Adrian	72	40	95
2	Alby	49	65	64
3	Abner	86	40	96
4	Benett	50	98	69

Write to CSV File

Use `to_csv()` function to write dataframe to a CSV File.

In [3]:
```
1 df.to_csv('class1_test1_1.csv')
```

By default, row and column labels, i.e. index and columns, will be exported too.

- To ignore `index` in the output, add parameter `index=False`.
- To ignore `header` in the output, add parameter `header=False`.

In [4]:
```
1 df.to_csv('class1_test1_2.csv', index=False)
```

In [5]:
```
1 df.to_csv('class1_test1_3.csv', header=False)
```

2. Excel Files

Write to Excel File

Use `to_excel()` function to save a dataframe to an Excel file.

- **NOTE:** This will overwrite existing data in the Excel file. You cannot insert a new sheet to Excel file.

In [6]:
```
1 df.to_excel('class1.xlsx')
```

Similarly, you can set both `index` and `header` parameter to `False` to omit index and header in the output.

You can also choose what columns to be exported using `columns` parameter.

- Note: Seems to be broken at current version.

In [7]:
```
1 df.to_excel('class1.xlsx', columns=['name', 'english'], index=False)
```

Read from Excel File

Use `read_excel()` function to read from Excel file. By default, it will read the first sheet.

In [8]:
```
df1 = pd.read_excel('class1.xlsx')
df1.head()
```

Out[8]:

	Unnamed: 0	name	english
0	0	Aaron	70
1	1	Adrian	72
2	2	Alby	49
3	3	Abner	86
4	4	Benett	50

You can also specify **sheet name** as 2nd parameter if you would like to read a specific sheet.

- You can also specify `index_col` if you would like a column to be index, Else pandas will create an index column for you.

In [9]:
```
df2 = pd.read_excel('class1.xlsx', 'Sheet1', index_col=0)
df2.head()
```

Out[9]:

	name	english
0	Aaron	70
1	Adrian	72
2	Alby	49
3	Abner	86
4	Benett	50

Write to a Sheet in Excel File

Engine `xlsxwriter`

By default, Pandas uses `xlsxwriter` engine to write to Excel file. But `xlsxwriter` doesn't support working with individual sheet.

Engine `openpyxl`

To work with individual sheets in an Excel file, make use of `openpyxl` engine.

In [29]:
```
!pip install openpyxl
import openpyxl
```

```
Requirement already satisfied: openpyxl in c:\users\isszq\anaconda3\lib\sit
e-packages (3.0.4)
Requirement already satisfied: et-xmlfile in c:\users\isszq\anaconda3\lib\s
ite-packages (from openpyxl) (1.0.1)
Requirement already satisfied: jdcal in c:\users\isszq\anaconda3\lib\site-p
ackages (from openpyxl) (1.4.1)
```

Create a new empty Excel file.

In [12]:
```
df0 = pd.DataFrame()
df0.to_excel('test.xlsx')
```

When create `ExcelWriter`, set engine to `openpyxl`.

- If the sheet name already exists, it will append numeric value, e.g. `1`, behind the sheet name.

In [13]:
```
with pd.ExcelWriter('test.xlsx', engine='openpyxl', mode='a') as writer
    df.to_excel(writer, sheet_name = 'Sheet5', index=False)
    df.to_excel(writer, sheet_name = 'Sheet6', index=False)
```

Get Sheet Names of an Excel

In [14]:
```
workbook=openpyxl.load_workbook('test.xlsx')
workbook.sheetnames
```

Out[14]: `['Sheet1', 'Sheet5', 'Sheet6']`

Remove a Sheet from Excel

In [15]:
```
workbook=openpyxl.load_workbook('test.xlsx')
worksheet=workbook['Sheet6']
workbook.remove(worksheet)
```

3. JSON Files

In [16]:
```
df = pd.read_csv('data/class1_test1.tsv', delimiter='\t')
```

Write to JSON File

To save a dataframe to a JSON file, use `to_json()` method of dataframe.

- By default, dataframe exports to JSON file by columns.

In [17]:
```
1 df.to_json('class1_test1_1.json')
```

To write a dataframe to JSON file by rows, add parameter `orient='table'` in `to_json()` function.

In [18]:
```
1 df.to_json('class1_test1_2.json', orient='table')
```

Read from JSON

To read JSON data, use `pandas.read_json()` function.

In [19]:
```
1 df1 = pd.read_json('class1_test1_1.json')
```

Similarly, you need to specify parameter `orient='table'` when reading a JSON file written by rows.

In [20]:
```
1 df2 = pd.read_json('class1_test1_2.json', orient='table')
```

Compare the 2 dataframes to make sure they are equal.

In [21]:
```
1 df1.equals(df2)
```

Out[21]: True

4. Read Data from HTML table

Pandas is able to extract data from `<table>` tag in HTML code.

- This is only possible when HTML code is rendered in server.
- Returned result is a dataframe list. Multiple dataframe will be returned if the webpage contains more than 1 table.

In [22]:
```
1 dfs = pd.read_html("https://www.skysports.com/premier-league-table")
2 type(dfs)
3 type(dfs[0])
4 df = dfs[0]
```

Preview the data with data using `head()` and `tail()`.

In [23]:
```
1 df.head()
```

Out[23]:

	#	Team	Pl	W	D	L	F	A	GD	Pts	Last 6
0	1	Everton	5	4	1	0	14	7	7	13	NaN
1	2	Liverpool	6	4	1	1	15	14	1	13	NaN
2	3	Aston Villa	5	4	0	1	12	5	7	12	NaN
3	4	Leeds United	6	3	1	2	12	9	3	10	NaN
4	5	Crystal Palace	6	3	1	2	8	9	-1	10	NaN

Set the index column to column `#` .

In [24]:
```
1 df.set_index('#', inplace=True)
2 df
```

Out[24]:

	Team	Pl	W	D	L	F	A	GD	Pts	Last 6
#										
1	Everton	5	4	1	0	14	7	7	13	NaN
2	Liverpool	6	4	1	1	15	14	1	13	NaN
3	Aston Villa	5	4	0	1	12	5	7	12	NaN
4	Leeds United	6	3	1	2	12	9	3	10	NaN
5	Crystal Palace	6	3	1	2	8	9	-1	10	NaN
6	Chelsea	6	2	3	1	13	9	4	9	NaN
7	Leicester City	5	3	0	2	12	8	4	9	NaN
8	Arsenal	5	3	0	2	8	6	2	9	NaN
9	Wolverhampton Wanderers	5	3	0	2	5	7	-2	9	NaN
10	Tottenham Hotspur	5	2	2	1	15	8	7	8	NaN
11	West Ham United	6	2	2	2	12	8	4	8	NaN
12	Manchester City	5	2	2	1	8	8	0	8	NaN
13	Southampton	5	2	1	2	8	9	-1	7	NaN
14	Newcastle United	5	2	1	2	7	9	-2	7	NaN
15	Manchester United	5	2	1	2	9	12	-3	7	NaN
16	Brighton and Hove Albion	5	1	1	3	9	11	-2	4	NaN
17	West Bromwich Albion	5	0	2	3	5	13	-8	2	NaN
18	Burnley	4	0	1	3	3	8	-5	1	NaN
19	Sheffield United	6	0	1	5	3	9	-6	1	NaN
20	Fulham	6	0	1	5	5	14	-9	1	NaN

5. Relational Database

Pandas can also interface with relational databases.

Write to a SQLite File

Python has built-in support for SQLite file.

In [25]:
```
1 import sqlite3
```

Use function `to_sql()` to export data to SQLite File.

- SQLite file will be created if the file doesn't exists.
- If same table already exists in SQLite file, exception will be raised.

In [26]:
```
1 conn = sqlite3.connect('test.sqlite')
2 df.to_sql('test1', conn, if_exists='replace')
3 conn.close()
```

```
C:\Users\isszq\Anaconda3\lib\site-packages\pandas\core\generic.py:2653: Use
rWarning: The spaces in these column names will not be changed. In pandas v
ersions < 0.14, spaces were converted to underscores.
  sql.to_sql(
```

Read Data from SQLite

Function `read_sql_query()` can be used to read data from a SQLite file.

- With SQLAlchemy, function `read_sql_table()` can also be used.
- Another function `read_sql()` is just a wrapper around `read_sql_table()` and `read_sql_query()` .

In [27]:
```
1 with sqlite3.connect('test.sqlite') as conn:
2     df1 = pd.read_sql('SELECT * FROM test1', conn)
3
4 df1.head()
```

Out[27]:

	#	Team	Pl	W	D	L	F	A	GD	Pts	Last 6
0	1	Everton	5	4	1	0	14	7	7	13	None
1	2	Liverpool	6	4	1	1	15	14	1	13	None
2	3	Aston Villa	5	4	0	1	12	5	7	12	None
3	4	Leeds United	6	3	1	2	12	9	3	10	None
4	5	Crystal Palace	6	3	1	2	8	9	-1	10	None

Supporting Other Relational Databases

Other relational databases like MySQL and PostgreSQL are also supported.

In [28]:

```
# import MySQLdb
# conn = MySQLdb.connect(host="localhost",    # your host, usually loca
#                        user="root",         # your username
#                        passwd="root",  # your password
#                        db="sakila")        # name of the data base
# pd.read_sql_query("select * from sakila.film", conn )
```