Importing Data Using Pandas

Introduction

Pandas is a popular library for efficiently wrangling data. It is particularly optimized to work with two-dimensional tabular data that is organized in rows and columns. In this lesson, you will learn how to import tabular data as a Pandas DataFrame object, how to access and manipulate the data in DataFrame objects, and how to export DataFrames to some common file formats.

For more information on Pandas, refer to https://pandas.pydata.org/pandas-docs/stable/ (https://pandas.pydata.org/pandas-docs/stable/) .

Objectives

You will be able to:

- Use pandas to import data from a CSV and and an Excel spreadsheet
- Use pandas to export a DataFrame to a file

Loading Pandas

When importing Pandas, it is standard to import it under the alias `pd`

In [9]:
```
import pandas as pd
```

Importing Data

There are a few main functions for importing data into a Pandas DataFrame including:

- `pd.read_csv()`
- `pd.read_excel()`
- `pd.read_json()`
- `pd.DataFrame.from_dict()`

Most of these functions are fairly straightforward; you use `read_csv()` for csv files, `read_excel()` for excel files (both new and old `.xlx` and `.xlsx` formats), and `read_json()` for json files. That said, there are a few nuances you should know about. The first is that the `read_csv()` format can be used for any plain-text delimited file. This may include (but is not limited to) pipe (|) delimited files (`.psv`) and tab separated files (`.tsv`).

Let's look at an example by investigating a file, `'bp.txt'` , stored in the `Data` folder.

In [2]:
```
# Import 'bp.txt' file
df = pd.read_csv('Data/bp.txt', delimiter='\t')
```

We've now loaded the data from a file into a DataFrame. To investigate the DataFrame, we can use a method called `.head(n)` or `.tail(n)`, which will respectively return first and last **n** items in the DataFrame.

In [3]:
```
# Look at the first 3 rows
df.head(3)
```

Out[3]:

	Pt	BP	Age	Weight	BSA	Dur	Pulse	Stress
0	1	105	47	85.4	1.75	5.1	63	33
1	2	115	49	94.2	2.10	3.8	70	14
2	3	116	49	95.3	1.98	8.2	72	10

In [4]:
```
# Look at the last 4 rows
df.tail(4)
```

Out[4]:

	Pt	BP	Age	Weight	BSA	Dur	Pulse	Stress
16	17	106	46	87.0	1.87	3.6	62	18
17	18	113	46	94.5	1.90	4.3	70	12
18	19	110	48	90.5	1.88	9.0	71	99
19	20	122	56	95.7	2.09	7.0	75	99

This example shows that the data was tab delimited (`\t`), so an appropriate file extension could have also been `.tsv` . Once we've loaded the dataset, we can export it to any format we would like with the related methods:

- `df.to_csv()`
- `df.to_excel()`
- `df.to_json()`
- `df.to_dict()`

There are also several other options available, but these are the most common.

Skipping and Limiting Rows

Another feature that you may have to employ is skipping rows when there is metadata stored at the top of a file. You can do this using the optional parameter `skiprows` . Similarly, if you want to only load a portion of a large file as an initial preview, you can use the `nrows` parameter.

In [5]:
```
# Import the first 100 rows of 'ACS_16_5YR_B24011_with_ann.csv' file
df = pd.read_csv('Data/ACS_16_5YR_B24011_with_ann.csv', nrows=100)
# Look at the first five rows
df.head()
```

Out[5]:

	GEO.id	GEO.id2	GEO.display-label	HD01_VD01	HD02_VD01	HD01_VD02	HD02_VD02	
0	Id	Id2	Geography	Estimate; Total:	Margin of Error; Total:	Estimate; Total: - Management, business, scien...	Margin of Error; Total: - Management, business...	M
1	0500000US01001	01001	Autauga County, Alabama	33267	2306	48819	1806	
2	0500000US01003	01003	Baldwin County, Alabama	31540	683	49524	1811	
3	0500000US01005	01005	Barbour County, Alabama	26575	1653	41652	2638	
4	0500000US01007	01007	Bibb County, Alabama	30088	2224	40787	2896	

5 rows × 75 columns

Notice the first row is descriptions of the variables

We could manually remove:

In [6]:
```
# Delete the first row
df = df.drop(0)
df.head(2)
```

Out[6]:

	GEO.id	GEO.id2	GEO.display-label	HD01_VD01	HD02_VD01	HD01_VD02	HD02_VD02	HD
1	0500000US01001	01001	Autauga County, Alabama	33267	2306	48819	1806	
2	0500000US01003	01003	Baldwin County, Alabama	31540	683	49524	1811	

2 rows × 75 columns

Or if we knew from the start, we could use the skiprows argument:

In [7]:
```
# Import the first 100 rows of 'ACS_16_5YR_B24011_with_ann.csv' file while skippi
df = pd.read_csv('Data/ACS_16_5YR_B24011_with_ann.csv', skiprows=1, nrows=100)
df.head()
```

Out[7]:

	Id	Id2	Geography	Estimate; Total:	Margin of Error; Total:	Estimate; Total: - Management, business, science, and arts occupations:	Margin of Error; Total: - Management, business, science, and arts occupations:	Estimate Total: Management business science, and arts occupations Management business and financia occupations
0	0500000US01001	1001	Autauga County, Alabama	33267	2306	48819	1806	55557
1	0500000US01003	1003	Baldwin County, Alabama	31540	683	49524	1811	57150
2	0500000US01005	1005	Barbour County, Alabama	26575	1653	41652	2638	51797
3	0500000US01007	1007	Bibb County, Alabama	30088	2224	40787	2896	50069
4	0500000US01009	1009	Blount County, Alabama	34900	2063	46593	2963	47003

5 rows × 75 columns

Header

Related to `skiprows` is the `header` parameter. This specifies the row where column names are and starts importing data from that point:

In [8]:
```
# Look at the error output once you run this cell. What type of error is it?
df = pd.read_csv('Data/ACS_16_5YR_B24011_with_ann.csv', header=1)
df.head()
```

```
---------------------------------------------------------------------------
UnicodeDecodeError                        Traceback (most recent call last)
/tmp/ipykernel_184/2769776417.py in <module>
      1 # Look at the error output once you run this cell. What type of error i
s it?
----> 2 df = pd.read_csv('Data/ACS_16_5YR_B24011_with_ann.csv', header=1)
      3 df.head()

/opt/conda/lib/python3.9/site-packages/pandas/util/_decorators.py in wrapper(*a
rgs, **kwargs)
    309                     stacklevel=stacklevel,
    310                 )
--> 311             return func(*args, **kwargs)
    312
    313         return wrapper

/opt/conda/lib/python3.9/site-packages/pandas/io/parsers/readers.py in read_csv
(filepath_or_buffer, sep, delimiter, header, names, index_col, usecols, squeez
e, prefix, mangle_dupe_cols, dtype, engine, converters, true_values, false_valu
es, skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na,
 na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep
_date_col, date_parser, dayfirst, cache_dates, iterator, chunksize, compressio
n, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapec
har, comment, encoding, encoding_errors, dialect, error_bad_lines, warn_bad_lin
es, on_bad_lines, delim_whitespace, low_memory, memory_map, float_precision, st
orage_options)
    584     kwds.update(kwds_defaults)
    585
--> 586     return _read(filepath_or_buffer, kwds)
    587
    588

/opt/conda/lib/python3.9/site-packages/pandas/io/parsers/readers.py in _read(fi
lepath_or_buffer, kwds)
    486
    487     with parser:
--> 488         return parser.read(nrows)
    489
    490

/opt/conda/lib/python3.9/site-packages/pandas/io/parsers/readers.py in read(sel
f, nrows)
   1045     def read(self, nrows=None):
   1046         nrows = validate_integer("nrows", nrows)
-> 1047         index, columns, col_dict = self._engine.read(nrows)
   1048
   1049         if index is None:

/opt/conda/lib/python3.9/site-packages/pandas/io/parsers/c_parser_wrapper.py in
read(self, nrows)
    221         try:
    222             if self.low_memory:
--> 223                 chunks = self._reader.read_low_memory(nrows)
```

```
    224                 # destructive to chunks
    225                 data = _concatenate_chunks(chunks)

/opt/conda/lib/python3.9/site-packages/pandas/_libs/parsers.pyx in pandas._lib
s.parsers.TextReader.read_low_memory()

/opt/conda/lib/python3.9/site-packages/pandas/_libs/parsers.pyx in pandas._lib
s.parsers.TextReader._read_rows()

/opt/conda/lib/python3.9/site-packages/pandas/_libs/parsers.pyx in pandas._lib
s.parsers.TextReader._tokenize_rows()

/opt/conda/lib/python3.9/site-packages/pandas/_libs/parsers.pyx in pandas._lib
s.parsers.raise_parser_error()

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xf1 in position 21229: inv
alid continuation byte
```

Encoding

Encoding errors like the one above are always frustrating. This has to do with how the strings within the file itself are formatted. The most common encoding other than `utf-8` that you are likely to come across is `latin-1` .

In [ ]:
```
# Import the 'ACS_16_5YR_B24011_with_ann.csv' file using a proper encoding
df = pd.read_csv('Data/ACS_16_5YR_B24011_with_ann.csv', header=1, encoding='latir
df.head()
```

Selecting Specific Columns

You can also select specific columns if you only want to load specific features.

In [ ]:
```
# Import the file with specific columns
df = pd.read_csv('Data/ACS_16_5YR_B24011_with_ann.csv',
                 usecols=[0, 1, 2, 5, 6], encoding='latin-1')
df.head(2)
```

or

In [ ]:
```
# Import the file with specific columns
df = pd.read_csv('Data/ACS_16_5YR_B24011_with_ann.csv', usecols=['GEO.id', 'GEO.i
df.head(2)
```

Selecting Specific Sheets

You can also select specific sheets for Excel files! This can be done by index number.

In [ ]:
```
# Import an Excel file
df1 = pd.read_excel('Data/Yelp_Selected_Businesses.xlsx', header=2)
df1.head()
```

In [ ]:
```
# Import a specific sheet of an Excel file
df2 = pd.read_excel('Data/Yelp_Selected_Businesses.xlsx', sheet_name=2, header=2)
df2.head()
```

Or the name of the sheet itself

In [ ]:
```
# Import a specific sheet of an Excel file
df = pd.read_excel('Data/Yelp_Selected_Businesses.xlsx', sheet_name='Biz_id_RESDU
df.head()
```

Loading a Full Workbook and Previewing Sheet Names

You can also load an entire excel workbook (which is a collection of spreadsheets) with the `pd.ExcelFile()` function.

In [ ]:
```
# Import the names of Excel sheets in a workbook
workbook = pd.ExcelFile('Data/Yelp_Selected_Businesses.xlsx')
workbook.sheet_names
```

In [ ]:
```
# Import a specific sheet
df = workbook.parse(sheet_name=1, header=2)
df.head()
```

Saving Data

Once we have data loaded that we may want to export back out, we use the **`.to_csv()`** or **`.to_excel()`** methods of any DataFrame object.

In [ ]:
```
# Write data to a CSV file
# Notice how we have to pass index=False if we do not want it included in our out
df.to_csv('NewSavedView.csv', index=False)
```

In [ ]:
```
# Write data to an Excel file
df.to_excel('NewSavedView.xlsx')
```

Summary

We've spent some time looking into how data importing with Pandas works and some of the methods you can use to manage the import and access of data. In the next lesson, you'll get some hands on practice!