Fetching data in Python: pandas

Paul Bradshaw

This week:

- Using **pandas** for importing data
- Dealing with different data **formats**: Excel, CSV, JSON, etc.
- More jargon: **attributes** and **properties**
- **Exporting** data from Colab

The **pandas** library

- `import pandas as pd`
- Functions for handling/analysing data
- Uses a **dataframe** to hold data

pandas functions (pd = pandas)

- **pd.read_csv( )** - import CSV (“url”)
- **pd.read_excel( )** - import XLS sheet (“url”)
- **pd.read_json( )** - for JSON
- **pd.ExcelFile( )** - import whole file/sheets

Extra parameters in read_excel()

- **sheet_name=** which sheet/number to import
- **header=** specify headings row
- **usecols=** specify columns to keep
- **nrows=** specify rows to keep
- **skipfooter=** specify rows to leave out at end

‘Objects’ have built-in attributes

- Objects created by certain functions *(you might see this called a ‘class’ of object)* often have certain **attributes**
- E.g. A ‘makeacar’ object might have wheels, gears, current speed, top speed, etc.
- These can be accessed with built-in code, e.g. **car.wheels**

Dataframe ‘object’ attributes

- **df.shape** - show number of rows, columns
- **df.columns** - show names of columns
- **df.dtypes** - show types of columns
- **df.size** - show number of cells

‘Objects’ have built-in functions

- Objects created by certain functions also often have certain built-in functions (called **methods**)
- E.g. A ‘makeacar’ object might be able to ‘accelerate’ or ‘turn left’
- These can be accessed with built-in code too, attached to it with a period, e.g. **car.turnleft()**

Dataframe ‘object’ functions (let’s call it df)

- **df.head()** - show first 5 (or specified) rows
- **df.append()** - add row (a dictionary)
- **df.apply()** - apply function along an axis
- **df.combine()** - with another dataframe

Methods vs attributes

- Built-in functions **(methods)** look like functions, but are attached to an *object* not the library, e.g. `df.head()`
- **Attributes** are attached to the object but don’t have brackets, e.g. `df.columns`

Accessing columns and rows

- **df[‘crimes’]** - access column named
- **df[0]** - access column at index
- **df.iloc[0,2]** - access cell by column, row
- **df.iloc[[2]]** - access row by index

datacamp

Python For Data Science
Pandas Basics Cheat Sheet

Learn Pandas Basics online at www.DataCamp.com

Pandas

The **Pandas** library is built on NumPy and provides easy-to-use **data structures** and **data analysis** tools for the Python programming language.

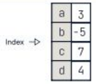
Use the following import convention:

```
>>> import pandas as pd
```

> Pandas Data Structures

Series

A **one-dimensional** labeled array capable of holding any data type

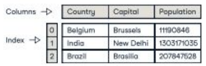


a	3
b	-5
c	7
d	4

```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

Dataframe

A **two-dimensional** labeled data structure with columns of potentially different types


Columns
Index

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasilia	207847528

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
         'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
         'Population': [11190846, 1303171035, 207847528]}
>>> df = pd.DataFrame(data,
                  columns=['Country', 'Capital', 'Population'])
```

> Dropping

```
>>> s.drop(['a', 'c']) #Drop values from rows (axis=0)
>>> df.drop('Country', axis=1) #Drop values from columns(axis=1)
```

> Asking For Help

```
>>> help(pd.Series.loc)
```

> Sort & Rank

```
>>> df.sort_index() #Sort by labels along an axis
>>> df.sort_values(by='Country') #Sort by the values along an axis
>>> df.rank() #Assign ranks to entries
```

> I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> df.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

`read_sql()` **is a convenience wrapper around** `read_sql_table()` **and** `read_sql_query()`

```
>>> df.to_sql('myDf', engine)
```

> Selection

Also see NumPy Arrays

Getting

```
>>> s['b'] #Get one element
 -5
>>> df[1:] #Get subset of a DataFrame
   Country Capital Population
 1 India New Delhi 1303171035
 2 Brazil Brasilia 207847528
```

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]] #Select single value by row & column
 'Belgium'
>>> df.iat([0],[0])
 'Belgium'
```

By Label

```
>>> df.loc[[0], ['Country']] #Select single value by row & column labels
 'Belgium'
>>> df.at([0], ['Country'])
 'Belgium'
```

By Label/Position

```
>>> df.ix[2] #Select single row of subset of rows
 Country Brazil
 Capital Brasilia
 Population 207847528
>>> df.ix[:,'Capital'] #Select a single column of subset of columns
 0 Brussels
 1 New Delhi
 2 Brasilia
>>> df.ix[1,'Capital'] #Select rows and columns
 'New Delhi'
```

Boolean Indexing

```
>>> s[~(s > 1)] #Series s where value is not >1
>>> s[(s < -1) | (s > 2)] #s where value is <-1 or >2
>>> df[df['Population']>1200000000] #Use filter to adjust DataFrame
```

Setting

```
>>> s['a'] = 6 #Set index a of Series s to 6
```

> Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape #(rows,columns)
>>> df.index #Describe index
>>> df.columns #Describe DataFrame columns
>>> df.info() #Info on DataFrame
>>> df.count() #Number of non-NA values
```

Summary

```
>>> df.sum() #Sum of values
>>> df.cumsum() #Cummulative sum of values
>>> df.min()/df.max() #Minimum/maximum values
>>> df.idxmin()/df.idxmax() #Minimum/Maximum index value
>>> df.describe() #Summary statistics
>>> df.mean() #Mean of values
>>> df.median() #Median of values
```

> Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f) #Apply function
>>> df.applymap(f) #Apply function element-wise
```

> Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
 a 10.0
 b NaN
 c 5.0
 d 7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_values=0)
 a 10.0
 b -5.0
 c 5.0
 d 7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

Exporting (more methods)

- **df.to_csv()** - export as (specified) CSV
- **df.to_excel()** - export as (specified) XLS
- **df.to_json()** - export as (specified) JSON
- **df.to_html()** - export as (specified) HTML

e.g. `df.to_csv(“mydata.csv”)`

The ExcelFile ‘object’ (let’s call it xlfile)

- **xlfile.sheet_names** - show sheet names

Key points

- Use **pandas** for importing, questioning, and exporting data in various formats
- Typically imported **as pd**
- ‘Objects’ created by some functions include certain **attributes** and **methods**

Task

- Go through the FOI notebook (notebook 6)
- Replicate the calculation for another section (exemption) - which organisation is worst for those refusals?

Cheat sheets

- https://blog.finxter.com/pandas-cheat-sheets/