Importing Data in Python

Little summary on different ways to import different data

Sergi Lehkyi
Sep 12, 2018 · 6 min read

Recently I finished two courses on data import in Python at DataCamp and I was really surprised of the amount of sources that can be used to get data. Here I would like to summarize all those methods and at the same time keen my knowledge. Also I think for others it will be useful as well. So, let's begin.

There is huge variety of files that can be used as the data source:

- flat files — csv, txt, tsv etc.
- pickled files
- excel spreadsheets
- SAS and Stata files
- HDF5
- MATLAB
- SQL databases
- web pages
- APIs

Flat files

Flat files — txt, csv — are easy and there are few ways to import them using numpy or pandas.

numpy.recfromcsv — Load ASCII data stored in a comma-separated file. The returned array is a record array (if usemask=False, see recarray) or a masked record array (if usemask=True, see ma.mrecords.MaskedRecords).

numpy.loadtxt — This function aims to be a fast reader for simply formatted files. The *genfromtxt* function provides more sophisticated handling of, e.g., lines with missing values.

```
data = np.loadtxt('file.csv', delimiter=',', skiprows=1, usecols=
[0,2])
```

numpy.genfromtxt — Load data from a text file, with missing values handled as specified. Much more sophisticated function that has a lot of parameters to control your import.

```
data = np.genfromtxt('titanic.csv', delimiter=',', names=True,
dtype=None)
```

With pandas it's even easier — one line and you have your file in a DataFrame ready. Also supports optionally iterating or breaking of the file into chunks.

```
data = pd.read_csv(file, nrows=5, header=None, sep='\t',
comment='#', na_values='Nothing')
```

Pickle

What the hell is pickle? It is used for serializing and de-serializing a Python object structure. Any object in python can be pickled so that it can be saved on disk. What pickle does is that it "serialises" the object first before writing it to file. Pickling is a way to convert a python object (list, dict, etc.) into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another python script. The code below will print a dictionary that was created somewhere and stored in the file — pretty cool, isn't it?

```
import pickle
with open('data.pkl', 'rb') as file:
       d = pickle.load(file)

print(d)
```

Excel

With *pandas.read_excel* that reads an Excel table into a pandas DataFrame and has a lot of customization importing data have never been more pleasant (sounds like TV commercial :D). But it is really true — documentation for this function is clear and

you are actually able to do whatever you want with that Excel file.

```
df = pd.read_excel('file.xlsx', sheet_name='sheet1')
```

SAS and Stata

SAS stands for Statistical Analysis Software. A SAS data set contains data values that are organized as a table of observations (rows) and variables (columns). To open this type of files and import data from it the code sample below will help:

```
from sas7bdat import SAS7BDAT
with SAS7BDAT('some_data.sas7bdat') as file:
     df_sas = file.to_data_frame()
```

Stata is a powerful statistical software that enables users to analyze, manage, and produce graphical visualizations of data. It is primarily used by researchers in the fields of economics, biomedicine, and political science to examine data patterns. Data stored in .dta files and the best way to import it is *pandas.read_stata*

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

HDF5

Hierarchical Data Format (HDF) is a set of file formats (HDF4, HDF5) designed to store and organize large amounts of data. HDF5 is a unique technology suite that makes possible the management of extremely large and complex data collections. HDF5 simplifies the file structure to include only two major types of object:

- Datasets, which are multidimensional arrays of a homogeneous type
- Groups, which are container structures which can hold datasets and other groups

This results in a truly hierarchical, filesystem-like data format. In fact, resources in an HDF5 file are even accessed using the POSIX-like syntax */path/to/resource*. Metadata is stored in the form of user-defined, named attributes attached to groups and datasets. More complex storage APIs representing images and tables can then be built up using datasets, groups and attributes.

To import HDF5 file we'll need *h5py* library. Code sample below made everything easier and totally understandable for me.

```
import h5py

# Load file:
data = h5py.File('file.hdf5', 'r')

# Print the keys of the file
for key in data.keys():
    print(key)

# Now when we know the keys we can get the HDF5 group
group = data['group_name']

# Going one level deeper, check out keys of group
for key in group.keys():
    print(key)

# And so on and so on
```

MATLAB

A lot of people work with MATLAB and store data in .mat files. So what those files are? These files contain list of variables and objects assigned to them in MATLAB workspace. It's not surprising that it is imported in Python as dictionary in which keys are MATLAB variables and values — objects assigned to these variables. To write and read MATLAB files *scipy.io* package is used.

```
import scipy.io
mat = scipy.io.loadmat('some_project.mat')
print(mat.keys())
```

Relational Databases

Using drivers to connect to a database we can grab data directly from there. Usually it means: create connection, connect, run the query, fetch the data, close connection. It is possible to do it step by step, but in pandas we have an awesome function that does it for us, so why bother ourselves? It only requires a connection that can be created with *sqlalchemy* package. Below is the example on connecting to sqlite database engine and getting data from it:

```
from sqlalchemy import create_engine
import pandas as pd
# Create engine
engine = create_engine('sqlite:///localdb.sqlite')

# Execute query and store records in DataFrame
df = pd.read_sql_query("select * from table", engine)
```

Data from Web

A separate article should be written on this, but I will highlight few things to at least know where to start. First of all, if we have a direct url to a file we can just use standard *pandas.read_csv/pandas.read_excel* functions specifying it in the parameter "file="

```
df = pd.read_csv('https://www.example.com/data.csv', sep=';')
```

Apart from this, to get data from the web we need to use HTTP protocol and especially GET method (there are a lot of them, but for the import we don't need more). And package *requests* does an incredible job doing this. To access a text from the response received by *requests.get* we just need to use method .text.

```
import requests
r = requests.get('http://www.example.com/some_html_page')
print(r.text)
```

r.text will give us a web-page with all html-tags on it — not very useful, isn't it? But here is where the fun begins. We have a *BeautifulSoup* package that can parse that HTML and extract the information we need, in this case all hyperlinks (continuing previous example):

```
from bs4 import BeautifulSoup
html_doc = r.text

# Create a BeautifulSoup object from the HTML
soup = BeautifulSoup(html_doc)

# Find all 'a' tags (which define hyperlinks)
a_tags = soup.find_all('a')

# Print the URLs to the shell
for link in a_tags:
    print(link.get('href'))
```

API

In computer programming, an application programming interface (API) is a set of subroutine definitions, communication protocols, and tools for building software. In general terms, it is a set of clearly defined methods of communication between various components. There are a lot of different APIs and the first thing that has to be done is documentation checked, but the truth is almost all APIs return data in JSON format. And we have to be able to catch that result. And again package *requests* will help us with it. (we have to send HTTP GET request to get data from API).

```
import requests
r = requests.get('https://www.example.com/some_endpoint')

# Decode the JSON data into a dictionary:
json_data = r.json()

# Print each key-value pair in json_data
for k in json_data.keys():
    print(k + ': ', json_data[k])
```

As we can see, data is everywhere and we have to know all the ways to get it. At this point my little summary gets to its end. Hopefully it will be useful not only for me.

)

Data Science Python Data