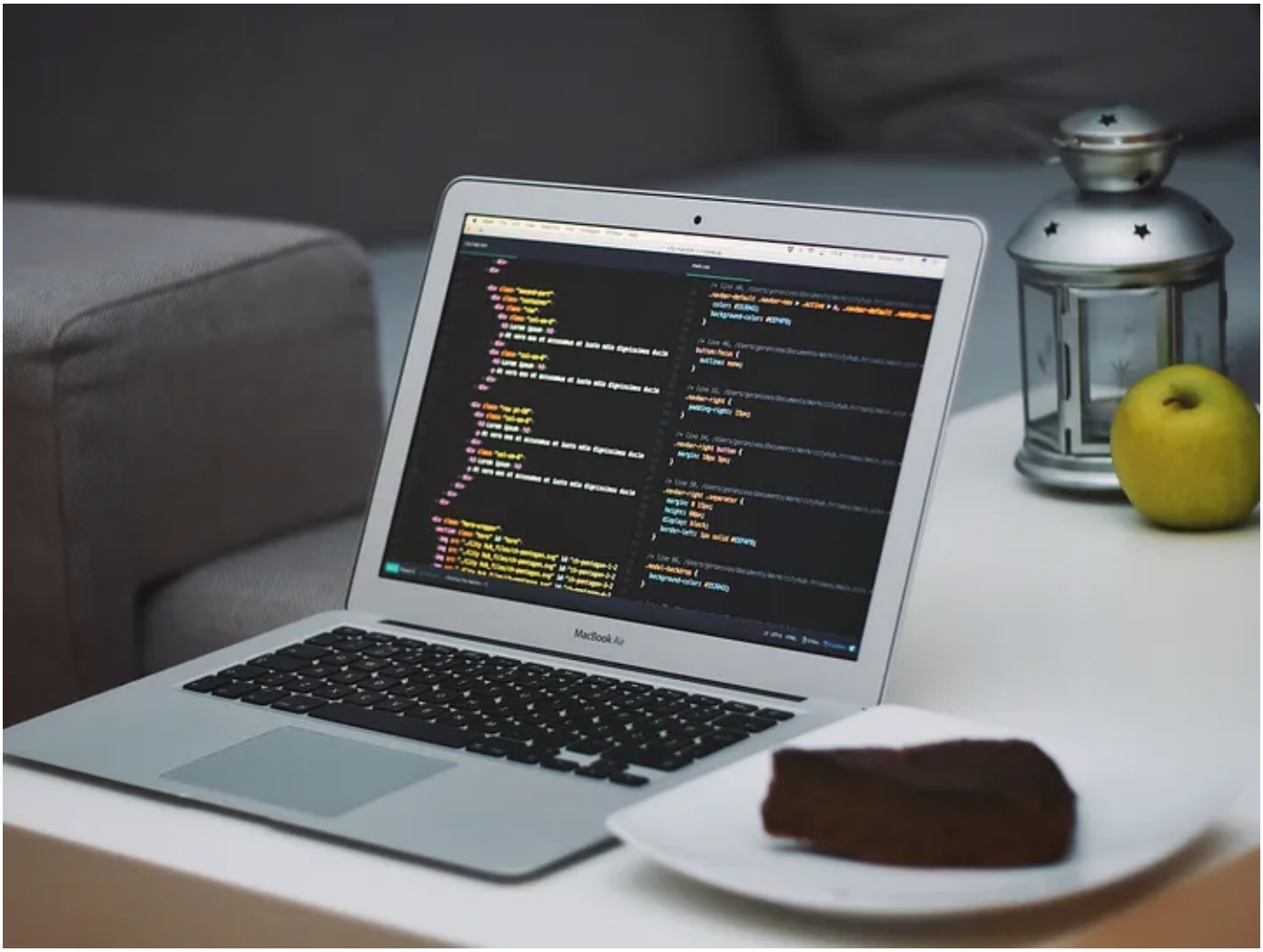
Published in Towards Data Science

George Seif Follow

Apr 29, 2019 · 3 min read · ✨ · Listen

The easy way to work with CSV, JSON, and XML in Python



> *Want to be inspired? Come join my **Super Quotes newsletter**.* 😎

Python's superior flexibility and ease of use are what make it one of the most popular programming language, especially for Data Scientists. A big part of that is how simple it is to work with large datasets.

Every technology company today is building up a data strategy. They've all realised that having the right data: insightful, clean, and as much of it as possible, gives them a key competitive advantage. Data, if used effectively, can offer deep, beneath the surface insights that can't be discovered anywhere else.

Over the years, the list of possible formats that you can store your data in has grown significantly. But, there are 3 that dominate in their everyday usage: CSV, JSON, and XML. In this article, I'm going to share with you the easiest ways to work with these 3 popular data formats in Python!

CSV Data

A CSV file is the most common way to store your data. You'll find that most of the data coming from Kaggle competitions is stored in this way. We can do both read and write of a CSV using the built-in Python csv library. Usually, we'll read the data into a list of lists.

Check out the code below. When we run `csv.reader()` all of our CSV data becomes accessible. The `csvreader.next()` function reads a single line from the CSV; every time you call it, it moves to the next line. We can also loop through every row of the csv using a for-loop as with `for row in csvreader` . Make sure that you have the same number of columns in each row, otherwise, you'll likely end up running into some errors when working with your list of lists.

```python
import csv

filename = "my_data.csv"

fields = []
rows = []

# Reading csv file
with open(filename, 'r') as csvfile:
    # Creating a csv reader object
    csvreader = csv.reader(csvfile)

    # Extracting field names in the first row
    fields = csvreader.next()

    # Extracting each data row one by one
    for row in csvreader:
        rows.append(row)

# Printing out the first 5 rows
for row in rows[:5]:
    print(row)
```

csv_python_1.py hosted with ❤ by GitHub view raw

Writing to CSV in Python is just as easy. Set up your field names in a single list, and your data in a list of lists. This time we'll create a `writer()` object and use it to write our data to file very similarly to how we did the reading.

```
import csv

# Field names
fields = ['Name', 'Goals', 'Assists', 'Shots']

# Rows of data in the csv file
rows = [ ['Emily', '12', '18', '112'],
         ['Katie', '8', '24', '96'],
         ['John', '16', '9', '101'],
         ['Mike', '3', '14', '82']]

filename = "soccer.csv"

# Writing to csv file
with open(filename, 'w+') as csvfile:
    # Creating a csv writer object
    csvwriter = csv.writer(csvfile)

    # Writing the fields
    csvwriter.writerow(fields)

    # Writing the data rows
    csvwriter.writerows(rows)
```

csv_python_2.py hosted with ❤ by GitHub view raw

Of course, installing the wonderful Pandas library will make working with your data far easier once you've read it into a variable. Reading from CSV is a single line as is writing it back to file!

```python
import pandas as pd

filename = "my_data.csv"

# Read in the data
data = pd.read_csv(filename)

# Print the first 5 rows
print(data.head(5))

# Write the data to file
data.to_csv("new_data.csv", sep=",", index=False)
```

csv_python_3.py hosted with ❤ by GitHub view raw

We can even use Pandas to convert from CSV to a list of dictionaries with a quick one-liner. Once you have the data formatted as a list of dictionaries, we'll use the `dicttoxml` library to convert it to XML format. We'll also save it to file as a JSON!

```python
import pandas as pd
from dicttoxml import dicttoxml
import json

# Building our dataframe
data = {'Name': ['Emily', 'Katie', 'John', 'Mike'],
        'Goals': [12, 8, 16, 3],
        'Assists': [18, 24, 9, 14],
        'Shots': [112, 96, 101, 82]
        }

df = pd.DataFrame(data, columns=data.keys())

# Converting the dataframe to a dictionary
# Then save it to file
data_dict = df.to_dict(orient="records")
with open('output.json', "w+") as f:
    json.dump(data_dict, f, indent=4)

# Converting the dataframe to XML
# Then save it to file
xml_data = dicttoxml(data_dict).decode()
with open("output.xml", "w+") as f:
    f.write(xml_data)
```

csv_python_4.py hosted with ❤ by GitHub view raw

JSON Data

JSON provides a clean and easily readable format because it maintains a dictionary-style structure. Just like CSV, Python has a built-in module for JSON that makes reading and writing super easy! When we read in the CSV, it will become a dictionary. We then write that dictionary to file.

```python
import json
import pandas as pd

# Read the data from file
# We now have a Python dictionary
with open('data.json') as f:
    data_listofdict = json.load(f)

# We can do the same thing with pandas
data_df = pd.read_json('data.json', orient='records')

# We can write a dictionary to JSON like so
# Use 'indent' and 'sort_keys' to make the JSON
# file look nice
with open('new_data.json', 'w+') as json_file:
    json.dump(data_listofdict, json_file, indent=4, sort_keys=True)

# And again the same thing with pandas
export = data_df.to_json('new_data.json', orient='records')
```


And as we saw before, once we have our data you can easily convert to CSV via pandas or use the built-in Python CSV module. When converting to XML, the `dicttoxml` library is always our friend.

```python
import json
import pandas as pd
import csv

# Read the data from file
# We now have a Python dictionary
with open('data.json') as f:
    data_listofdict = json.load(f)

# Writing a list of dicts to CSV
keys = data_listofdict[0].keys()
with open('saved_data.csv', 'wb') as output_file:
    dict_writer = csv.DictWriter(output_file, keys)
    dict_writer.writeheader()
    dict_writer.writerows(data_listofdict)
```

json_python_2.py hosted with ❤ by GitHub view raw

XML Data

XML is a bit of a different beast from CSV and JSON. Generally, CSV and JSON are widely used due to their simplicity. They're both easy and fast to read, write, and interpret as a human. There's no extra work involved and parsing a JSON or CSV is very lightweight.

XML on the other hand tends to be a bit heavier. You're sending more data, which means you need more bandwidth, more storage space, and more run time. But XML does come with a few extra features over JSON and CSV: you can use namespaces to build and share standard structures, better representation for inheritance, and an industry standardised way of representing your data with XML schema, DTD, etc.

To read in the XML data, we'll use Python's built-in XML module with sub-module ElementTree. From there, we can convert the ElementTree object to a dictionary using the `xmltodict` library. Once we have a dictionary, we can convert to CSV, JSON, or Pandas Dataframe like we saw above!

```python
import xml.etree.ElementTree as ET
import xmltodict
import json

tree = ET.parse('output.xml')
xml_data = tree.getroot()

xmlstr = ET.tostring(xml_data, encoding='utf8', method='xml')


data_dict = dict(xmltodict.parse(xmlstr))

print(data_dict)

with open('new_data_2.json', 'w+') as json_file:
    json.dump(data_dict, json_file, indent=4, sort_keys=True)
```

xml_python_1.py hosted with ❤ by GitHub view raw

Like to learn?

Follow me on twitter where I post all about the latest and greatest AI, Technology, and Science! Connect with me on LinkedIn too!

Artificial Intelligence Technology Innovation Programming Coding

Emails will be sent to humera.shaziya@gmail.com. Not you?

Get this newsletter